Operational Lessons from Chameleon

Kate Keahey*, Jason Anderson†, Paul Ruth‡, Jacob Colleran†, Cody Hammock§, Joe Stubbs§ and Zhuo Zhen†
*Mathematics and Computer Science Division
Argonne National Laboratory, Lemont, IL 60439
Email: keahey@anl.gov
†University of Chicago, Chicago, IL
Emails: jasonanderson,jakecoll,zhenz@uchicago.edu
‡RENCI, Chapel Hill, NC
Email: pruth@renci.org
§TACC, Austin, TX
Emails: hammock,jstubbs@tacc.utexas.edu

***Abstract*—Chameleon is a large-scale, deeply reconfigurable testbed built to support Computer Science experimentation. Unlike traditional systems of this kind, Chameleon has been configured using an adaptation of a mainstream open source infrastructure cloud system called OpenStack. We show that operating cloud systems requires both more skill and extra effort on the part of the operators - in particular where those systems are expected to evolve quickly - which can make systems of this kind expensive to run. We discuss three ways in which those operations costs can be managed: innovative monitoring and automation of systems tasks, building "operator co-ops", and collaborating with users.**

***Keywords*-cloud computing, cloud operations, DevOps**

I. Introduction

Computer Science experimental testbeds allow investigators to explore a broad range of different state-of-the-art hardware options, assess scalability of their systems, and provide conditions that allow deep reconfigurability and isolation so that one user does not impact the experiments of another. An experimental testbed is also in a unique position to support methods facilitating experiment analysis and improve repeatability and reproducibility of experiments. Providing these capabilities at least partially within a commodity framework improves the sustainability of systems experiments and thus makes them available to a broader range of experimenters.

Chameleon [1], [2] is a large-scale, deeply reconfigurable testbed built specifically to support the features described above. It currently consists of almost 20,000 cores, a total of 5PB of total disk space hosted at the University of Chicago and TACC, and leverages 100 Gbps connection between the sites. The hardware includes a large-scale homogenous partition to support large-scale experiments, as well as a diversity of configurations and architectures including Infiniband, GPUs, FPGAs, storage hierarchies with a mix of HDDs, SDDs, NVRAM, and high memory as well as non-x86 architectures such as ARMs and Atoms. To support systems experiments, Chameleon provides a configuration system giving users full control of the software stack including root privileges, kernel customization, and console access.

Unlike traditional experimental infrastructures such as Grid'5000 [3], GENI [4], Emulab [5], or CloudLab [6] which provide experimental capabilities by developing in-house infrastructures, Chameleon created an approach that provides similar and partially enhanced capabilities by building on and extending a mainstream open source Infrastructure-as-a-Service implementation: OpenStack [7]. Chameleon is built using OpenStack components, such as Ironic [8] for bare metal provisioning, Neutron [9] for network provisioning and automated layer-2 isolation, Heat [10] for orchestration, Glance [11] for disk image storage, Gnocchi [12] for long-term time-series metrics storage, and Swift [13] for object storage. Chameleon is also a core contributor to the OpenStack Blazar system [14], which is used to provide advanced reservations of physical hosts, network segments, and IP addresses [15]. Using OpenStack as a foundational layer allows us to leverage the effort of the extensive OpenStack developer community and greatly increases the pool of potential operators of the testbed as many have had experience with OpenStack. The benefit of this approach is akin to using well-supported libraries; we as operators benefit from patches submitted by authors or other members of the wider community, and we know that others are actively testing the same code we are using on a daily basis to run our infrastructure.

Chameleon has been used to support many projects in research on operating systems, virtualization, power management, networking, high-performance computing (HPC), cloud computing, data science, machine learning, and others. A sizable proportion of Chameleon users also leverage the system for education in computer science. To date, Chameleon has supported 3,000+ users working on 500+ projects.

In this paper, we present the experiences of the Chameleon operations team, discuss challenges that come up in operating a testbed of this kind, and compare and contrast the

level of operational challenges with the operations of major HPC datacenters.

II. Clouds Versus HPC Resources

A typical HPC resource emphasizes performance and utilization against flexibility and user control enabled by virtualization or use of containers. On traditional HPC resources, users run as non-privileged accounts (e.g., no sudo access, limited access to various file systems, etc). On cloud resources, users tend to have root level privileges on their resources typically provided via virtual machines. Chameleon takes this a step further by giving users access to bare metal resources and the ability to make changes to components such as the BIOS, allowing them to boot from custom kernel or use the serial console. Further, Chameleon users can also configure complex networking technologies including the use of virtual switches [16]. This imposes obvious challenges for operations from a security as well as operations standpoint.

Further, goals for resource utilization in a traditional HPC resource tend to be simplistic: high CPU utilization is the fundamental objective for HPC. The approach to resource management thus optimizes provider concerns, i.e., resources are scheduled "on availability" [17]; traditional HPC workloads sit in queues, sometimes for hours, and run once node resources are available. Cloud and especially experimental resources on the other hand, often require support for interactive and co-scheduled resources (e.g., combining nodes with networks). This requires satisfying hard user constraints for resource availability at a given time [15]. When a resource is available thus becomes an important requirement and may result in lesser utilization.

Most importantly, HPC and cloud resources differ in complexity and the associated operations cost. Compared with infrastructure clouds which essentially supply a user with their own computer for a limited time, the HPC job scheduler interface is simple and limited. HPC job schedulers do not manage networks or varying levels of security configurations for users. Cloud systems are more complex because they solve a significantly more complex problem. This places a demand on operations specialists as they require both broader expertise and potentially more effort to manage. Bare-metal experimental resource increase this complexity (and consequent operational demands) even more.

III. OpenStack in Chameleon

Chameleon uses OpenStack to provide its experimental capabilities. While most users are familiar with OpenStack as infrastructure that deploys virtual machines, the Ironic component of OpenStack allows it to be used with bare metal instances. Most components of OpenStack have now been integrated to work with Ironic - though some important capabilities (e.g., Ironic's use for Cinder, the OpenStack remote block storage component) are still not available. At the same time, it is clear that the Ironic integration gets proportionally less attention (in terms of performance management for example) than more commonly used components of OpenStack: we have observed performance bottlenecks for scheduling of an instance on a bare metal machine, during provisioning of networks on physical switches, during the PXE-based instance provisioning itself, or even for user interface response.

The complexity of the system also made upgrades to OpenStack painful in the past. This has got much better in recent years with projects like Kolla (packaging systems in Docker containers) and Kolla-Ansible (deployment-as-code using Ansible to instantiate OpenStack containers on physical hosts) [18]. OpenStack also is working to make rolling upgrades more reliable, so operators do not have to schedule full downtimes to perform upgrades.

OpenStack's distributed nature can lead to difficulties around state synchronization between all of the systems. This can often arise if network connectivity is interrupted in the middle of an asynchronous transaction across the system. These transactions are not true "transactions" (are not atomic) and cannot be easily rolled back. This is an unfortunate consequence of the highly distributed nature of the OpenStack components and plugins.

IV. Networking

Since cloud resources are by nature remote, networking is an important aspect of cloud computing experimentation. Most public cloud providers have rolled out advanced networking services that are simple to access by any cloud user, e.g. routing between regions and private networking spaces within a cloud. However, access to low-level, externally facing cloud network services such as AWS Direct Connect [19], Azure ExpressRoute [20], and Google Dedicated Interconnect [21] is difficult, expensive, or even simply impossible to most researchers without complicated support by campus IT staff, as well as national and regional network providers. As a result, most research on these services is being done by a few select scientists or campus IT staff themselves. Reducing the hurdles that prevent individual researchers from accessing these services enables a wide array cloud experiments, as well as provide expanded training to the next generation of campus IT staff in the use of these otherwise inaccessible services. Using networks for Computer Science experimentation is thus in a class by itself in terms of the demands it places on operators and the capabilities it offers to users.

The most significant challenge to increasing access to direct cloud network connection services is automation of key provisioning steps that currently require IT staff intervention. Public cloud providers provide a set of connection points (that we call "stitchports" [22]) in various geographic locations. Attaching a campus facility to a public cloud using a direct cloud connection requires a provisioning a

series private network circuits between the campus and a cloud connection point. Each circuit in the series will be provisioned by a different regional or national network transit provider. A typical direct cloud connection between a public cloud and a campus might make use of a shared cloud connection point provided by a national transit providers (e.g., Internet2). This national provider will provision a circuit between the cloud connection point and the campus' regional network provider. The regional provider will, in turn, provision a circuit between the national provider and the campus. Once in the campus, IT staff can connect the circuit to the desired local facility. Each of these circuits and connection points requires intervention by IT staff authorized to provision the appropriate infrastructure. In some cases (e.g. Internet2 and ESnet) network providers have APIs that the researcher, or more commonly the campus IT staff, can use to deploy the required infrastructure on their own. However, most regional providers and campus infrastructure have no such API and require manual intervention by trained IT staff.

Recent additions to Chameleon have enabled users to design and run experiments using AWS Direct Connect, Azure ExpressRoute, and/or Google Dedicated Interconnect without involving institutional IT staff. These experiments can connect Chameleon resources with the public cloud direct connections. Users can mimic existing or imagined campus infrastructure by deploying large scale experiments using Chameleon hardware and directly connect these "campuses" to public clouds. These connections take advantage of Chameleon's direct stitching capabilities and it's isolated user controlled OpenFlow networks using Corsa DP2000 series switches.

V. Chameleon Operations

Chameleon users interact with staff via submitting tickets [23]. This mode of interaction was selected over support via mailing list in anticipation of the scale of the system as it provides reliable ticket tracking and thereby ensures that no user problem is left behind. It also greatly facilitates weekly reviews of tickets which are one of our primary user feedback mechanisms and motivate the development of many new features. On the flip side, it prevents uses from interacting directly which may need us to revisit this choice. The submitted tickets are processed by University of Chicago and TACC staff on alternating weeks; Chameleon is thus operated as a "single instrument" with ticketmaster staff having sufficient level of privilege on both sites to handle most problems.

The ticket categories range between routine user profile management (comprising PI requests, allocation review and renewals, etc.), user questions (e.g., resulting from imperfect understanding of the system), system problems (e.g., network, hardware, software, or configuration failures), to allocation adjustment requests. The latter comprise a variety of special resource requests (e.g., lease extensions, FPGA access, or early user access to newly released capabilities) as well as fair sharing requests, e.g., the assignment of public IPs which may become unavailable [24] due to users' failure to release them.

Overall, we find that a proper functioning of the system is a collaboration between the operations team and the users: an independently minded and knowledgeable user community, sensitive to the providers concerns can significantly cut down on operations costs. Specifically, users can help operators manage the system in two important ways: one of them is submitting tickets, the other one is not submitting them. When things don't work, e.g., an instance does not deploy properly, or a feature is not sufficiently documented users often deal with the problem themselves (e.g., by deploying to another node) without reporting the problem to the operator. This is not helpful as it may delay maintenance actions and prevents us from improving documentation. User tickets are also an invaluable source of feedback on desired new features, research artifacts such as appliances, or training vehicles.

On the other hand, much user confusion arises out of understanding the system imperfectly, using it in ways in which it was not designed to be used, or ignoring best practices. In particular, proper management of shared resources, such as e.g., releasing nodes or public IP addresses or abstaining from stacking leases, is clearly required to implement fair sharing on the testbed; failure to do so often results in a significant support burden on the operators as they arbitrate the resources between users.

VI. Streamlining Operations

Interaction with users is of course only a small part of ongoing monitoring on the health of the system. Over time we have made efforts to drive the cost of operating Chameleon down as low as possible by combining a strong "base" of commodity systems with layers of automation around error detection and resolution.

Automated Issue Detection. Automation helps us detect issues early at scale to minimize user impact, and also helps our operators multiply their impact. Every Chameleon site is provisioned with an internal Prometheus server and a range of Prometheus metrics exporters [25]. We continuously monitor machines for high load, disk partitions running out of space, abnormal levels of network traffic, as well as various bits of OpenStack state, such as how many instances are in a failed state, or which system APIs are no longer responding. We use a central Grafana server [26] to visualize this monitoring data from each site Chameleon is deployed at, to allow operators to quickly get an overview of the testbed as a whole. Alerts are triggered when certain measured conditions are met; these alerts are broadcasted to a Slack [27] channel so they are visible immediately to operators online that day. Besides operational metrics,

we also track and visualize usage patterns on the testbed, such as the percentage of nodes reserved, and how many of those reserved nodes are ultimately provisioned by users. This data can be broken out by project or resource type, which can be helpful for identifying interesting use-cases or future capacity planning.

Chameleon also maintains several sets of periodic jobs. First, we maintain a suite of acceptance tests that test various "happy paths" through the system, e.g. reserving a node, provisioning the node with a Chameleon-provided disk image, assigning a public IP to the node, and ensuring that external SSH connectivity works and that monitoring metrics are being automatically pushed from the node once it is running. Depending on the scope of the test, they may run hourly, to give quick feedback on a particular system that may be in an error state, or daily, to act as an overall "smoke test" in to the health of the system from a typical user's point of view.

Automated Repair. We have a separate set of periodic jobs that have a different purpose: fixing known issues in an automated fashion. We call these jobs "hammers". They solve problems that we see pop up occasionally enough that the time investment in automation pays off; typically these are issues where various OpenStack systems have not converged to a steady state, and a simple adjustment is often all that is needed. While it is true that such issues should theoretically not arise, or should be prevented at the source, our experience has told us that such circumstances end up being a reality of running a testbed at this scale, and the simple solution of whacking the system in a few precise areas does the trick.

Managing Appliances. Finally, we rely on a series of tools that automate most of our disk image release process. In particular, we use DiskImage-Builder [28], a tool provided by OpenStack, to manage our image build; it captures the disk image build process and manages modifications to it so that a new image can be automatically generated based on updates. Thus, as new upstream OS releases are released, we automatically trigger rebuilds of our disk images bound to that OS. These images are then released to the testbed as the latest version of the OS image. We also periodically test that these images deploy correctly on our various hardware configurations; for each type of machine, we schedule a test provision every week, for each disk image Chameleon provides.

Currently, Chameleon operators still spend more time than we'd like on diagnosing failures and fixing up the root cause of failures. Our ongoing focus is on expanding our alerting infrastructure to capture more types of failures. Additionally, we will experiment with automated error response: running an automated set of remedies on the system in an attempt to solve the problem before an operator has to even look. It is our goal to ultimately have most seen failure modes well-documented and alerted on. This makes it easier for Chameleon to be run by operators less experienced with the testbed. Ultimately, Chameleon aims to be akin to an installable and upgradeable package, where operators at various host institutions can subscribe to changes and apply patches authored by Chameleon to improve the reliability of their deployed testbeds.

VII. Packaging Operations

Chameleon's utility is increased with every deployment: more sites mean more specialized hardware, more capacity for experimentation, and more interesting experiments. To this end, we have worked to package Chameleon (and the automated operational structure described above) such that others can deploy it with minimal configuration. Reproducing Chameleon using vanilla OpenStack components is possible, but challenging. For one, Chameleon maintains several forks of OpenStack services with minimal patch sets that are custom-tailored to the needs of a testbed; while most of them have been contributed it sometimes takes a long time for a contribution to be accepted. Furthermore, configuring a functional testbed that supports bare metal provisioning requires significant understanding of the underlying configuration of all the various OpenStack components. The human challenge then becomes: how do we make Chameleon feasible to install for as large community of operators without requiring them to absorb what is complex and specialized knowledge? In order to address it, we have worked to generalize the internal deployment of Chameleon such that it can be released and deployed by any user; we call this CHI-in-a-Box.

We boiled down the interface for CHI-in-a-Box is a set of configuration files. The site operator must define the inventory, or which physical machines will host which components of Chameleon, a globals YAML [29] configuration file, which defines important variables such as subnets and which parts of Chameleon should be enabled, and finally an encrypted passwords YAML configuration file, which contains secrets such as MySQL [30] passwords for each system user. By default, such passwords can be automatically generated using random strings if the operator wishes.

The deployment of CHI-in-a-Box, and indeed with the primary Chameleon sites, is handled via a commodity tool developed by the OpenStack community called Kolla-Ansible. Ansible is used to coordinate the deployment of pre-built Docker images on the operator's physical infrastructure. Rolling upgrades are supported, solving one of the biggest pain points with maintaining an OpenStack deployment. CHI-in-a-Box can also automatically install the various alerting and automated maintenance scripts ("hammers") to the operator's environment to reduce their operational costs.

Chameleon has chosen to "dogfood" CHI-in-a-Box internally. This means that any bug fixes or features that Chameleon operators add to the internal deployment of

Chameleon can immediately be leveraged by any other operators running Chameleon as an associate site.

VIII. Conclusions

In this paper we have discussed the operational experience behind Chameleon, a testbed for Computer Science research.

We argue that operating infrastructure clouds is significantly more complex than operating traditional HPC systems because clouds solve a more complex problem and thus typically place a higher level of experience and skill demands as well as potentially higher level of time commitment from the human operators. This high cost is significant because in practice it limits the usefulness of the system or it limits the number of users who can use it: in Chameleon specifically, it means that fewer experiments are available to fewer investigators.

This human burden can be alleviated by streamlining and automating operations via various mechanisms we described above. First, where possible operations can be automated and streamlined, and (as in the case of networking) tools can be developed that take the human out of the loop as possible. In the case of an experimental system that is designed to evolve in the set of use cases it supports this will by nature be an ongoing effort; thus the operations of an experimental system will always contain a development component. Secondly, packaging of a system, as in the case of CHI-in-a-box, create the potential of alleviating these costs by creating a "devops co-op" in practice where operations staff with high expertise packages operations in such a way that others with lesser investment of expertise and time can participate in operations. Finally, the operations of a system is always a covenant between users and operators; the ways a user community helps systems operations should not be underestimated.

Acknowledgment

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth, "Chameleon: a Scalable Production Testbed for Computer Science Research," in *Contemporary High Performance Computing: From Petascale toward Exascale*, 1st ed., ser. Chapman Hall/CRC Computational Science, J. Vetter, Ed. Boca Raton, FL: CRC Press, May 2019, vol. 3, ch. 5, pp. 123–148.

[2] Chameleon. [Online]. Available: https://www.chameleoncloud.org

[3] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: A Large Scale and Highly Reconfigurable Experimental Grid Testbed," *The International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[4] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A Federated Testbed for Innovative Network Experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.

[5] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: A Robotic Wireless and Sensor Network Testbed," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, 2006, pp. 1–12.

[6] R. Ricci, E. Eide, and The CloudLab Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *USENIX*, vol. 39, no. 6, Dec. 2014. [Online]. Available: https://www.usenix.org/publications/login/dec14/ricci

[7] Openstack. [Online]. Available: https://www.openstack.org/

[8] Openstack Ironic. [Online]. Available: https://docs.openstack.org/ironic

[9] Openstack Neutron. [Online]. Available: https://docs.openstack.org/neutron

[10] Openstack Heat. [Online]. Available: https://docs.openstack.org/heat

[11] Openstack Glance. [Online]. Available: https://docs.openstack.org/glance

[12] Gnocchi. [Online]. Available: https://gnocchi.xyz

[13] Openstack Swift. [Online]. Available: https://docs.openstack.org/swift

[14] Openstack Blazar. [Online]. Available: https://docs.openstack.org/blazar

[15] K. Keahey, P. Riteau, J. Anderson, and Z. Zhen, "Managing Alocatable Resources," in *Proceedings of The IEEE International Conference on Cloud Computing*. IEEE Press, 2019.

[16] P. Ruth, M. Cevik, K. Keahey, and P. Riteau, "Wide-area Software Defined Networking Experiments using Chameleon," in *Proceedings of The 5th IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds*. IEEE Press, 2019.

[17] F. Liu, K. Keahey, P. Riteau, and J. Weissman, "Dynamically Negotiating Capacity between On-demand and Batch Clusters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 38.

[18] Openstack Kolla-Ansible. [Online]. Available: https://docs.openstack.org/kolla-ansible

[19] AWS Direct Connect. [Online]. Available: https://aws.amazon.com/directconnect/

[20] Azure ExpressRoute. [Online]. Available: https://azure.microsoft.com/en-us/services/expressroute/

[21] Google Dedicated Interconnect. [Online]. Available: https://cloud.google.com/interconnect/docs/concepts/dedicated-overview

[22] I. Baldin, J. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills, "ExoGENI: A Multi-domain Infrastructure-as-a-service Testbed," in *The GENI Book*. Springer, 2016, pp. 279–315.

[23] Chameleon tickets. [Online]. Available: https://consult.tacc.utexas.edu

[24] Save the Planet, Use Fewer IPs. [Online]. Available: https://www.chameleoncloud.org/blog/2019/02/27/save-planet-use-fewer-ips/

[25] Prometheus. [Online]. Available: https://prometheus.io/

[26] Grafana. [Online]. Available: https://grafana.com/

[27] Slack. [Online]. Available: https://slack.com/

[28] OpenStack Diskimage-builder. [Online]. Available: https://docs.openstack.org/diskimage-builder/latest/

[29] YAML. [Online]. Available: https://yaml.org/

[30] MySQL. [Online]. Available: https://www.mysql.com/