# CHI-in-a-Box: Reducing Operational Costs of Research Testbeds

Kate Keahey
keahey@mcs.anl.gov
Argonne National Laboratory
Lemont, Illinois, USA

Jason Anderson
Michael Sherman
jasonanderson@uchicago.edu
shermanm@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Cody Hammock
hammock@tacc.utexas.edu
Texas Advanced Computing Center
Austin, Texas, USA

Zhuo Zhen
zhenz@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Jenett Tillotson
jtillots@ucar.edu
National Center for Atmospheric Research
Boulder, Colorado, USA

Timothy Bargo
Lance Long
tim@timbargo.com
llong4@uic.edu
University of Illinois Chicago
Chicago, Illinois, USA

Taimoor Ul Islam
Sarath Babu
Hongwei Zhang
tislam@iastate.edu
sarath4@iastate.edu
hongwei@iastate.edu
Iowa State University
Ames, Iowa, USA

François Halbach
fhalbach@tacc.utexas.edu
Texas Advanced Computing Center
Austin, Texas, USA

## ABSTRACT

Making scientific instruments for computer science research available and open to all is more important than ever given the constantly increasing pace of opportunity and innovation– yet, such instruments are expensive to build and operate given their complexity and need for rapid evolution to keep pace with the advancing frontier of science. This paper describes how we can lower the cost of computer science testbeds by making them easier to deploy and operate. We present CHI-in-a-Box, a packaging of CHameleon Infrastructure (CHI) underlying the Chameleon testbed, describe the practices that went into its design and implementation, and present three case studies of its use.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

## KEYWORDS

cloud computing, testbeds, maintainability

**ACM Reference Format:**
Kate Keahey, Jason Anderson, Michael Sherman, Cody Hammock, Zhuo Zhen, Jenett Tillotson, Timothy Bargo, Lance Long, Taimoor Ul Islam, Sarath Babu, Hongwei Zhang, and François Halbach. 2022. CHI-in-a-Box: Reducing Operational Costs of Research Testbeds. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3491418.3530768



*PEARC '22, July 10–14, 2022, Boston, MA, USA*



ACM ISBN 978-1-4503-9161-0/22/07.
https://doi.org/10.1145/3491418.3530768

## 1 INTRODUCTION

Scientific instruments for computer science research are notoriously hard to build and operate. To provide a platform for research on topics such as power management, operating system design, or networking they must operate very close to the hardware, support a deep level of reconfigurability, and give their users a high level of privilege. With all this, they also need to constantly, and sometimes rapidly, evolve as the science evolves to provide a viable instrument for answering new questions – and they must do so securely. The complexity of this undertaking and the rapid evolution rate of testbeds makes them expensive to deploy and operate—and thus poses a barrier to their widespread adoption and deployment which in practice limits resources available for computer science research.

At the same time, widespread access to instruments for computer science research is more important than ever. Opportunities and technological innovations are increasing at a faster rate than they have ever done before, including topics in composable hardware, programmable networking, and machine learning to name just a few. It is important that we keep pace in addressing them by bringing to bear the full collective talent and experience of the research community —and this entails access to experimental platforms that are not only broad in their capabilities but also broadly accessible. General access to a shared experimental platform can also encourage the development and sharing of community digital artifacts

such as e.g., electronic teaching materials, and encourage repeating or reproducing experiments by removing the resource availability barrier [18]. This raises the following questions: Can we make computer science research testbeds more available by lowering their cost? Can we capture the advanced capabilities, challenging operations, and steep evolution curve of such testbeds in a way comparable to a mainstream, "shrinkwrapped" infrastructure? How close can we get to this rough goal and what are the strategies that allow us to do so in a cost-effective manner?

This paper describes a research testbed infrastructure distribution based on our experience of operating Chameleon [20], a primarily bare metal reconfigurable research testbed for edge to cloud experimentation. Since its public availability in July 2015, Chameleon has supported 6,000+ users working on 800+ projects giving us runway and scale to understand and address the challenges connected to operating research infrastructure sites. The resulting insight, expressed as the CHI-in-a-Box testbed distribution created to facilitate and streamline the deployment and operation of Chameleon sites, employs a number of strategies including hub and spoke management, containerization, and automated inventory management to enable Bring Your Own Device (BYOD) composable infrastructure pattern, to combat complexity, minimize risk, and streamline effort involved in research infrastructure deployment and operation. We articulate the sources of complexity and cost in deploying and operating a research testbed, present the CHI-in-a-Box design, and describe three case studies or CHI-in-a-Box deployments ranging from a supercomputing center to smaller-scale university labs, evaluated by personnel with broad experience level ranging from a senior systems engineer with many years of experience to an undergraduate student.

## 2 CHAMELEON INFRASTRUCUTRE (CHI): WHAT'S IN THE BOX?

Chameleon is an NSF-funded distributed testbed that supports computer science research, education, and emergent applications [9]. Our users' projects range from research on power management, operating systems and networking, to data science, machine learning, and security. To cover the broadest possible range of experimental requirements, most Chameleon resources are presented as a reconfigurable bare-metal cloud, giving users full control of the software stack including root privileges, the ability to customize the kernel, or experiment with software defined networking. A small part of the system is configured as a virtualized KVM cloud to balance the cost of strong isolation essential for some projects, with finer-grained sharing provided by virtualization sufficient for others [20]. Most recently, to support experimentation on edge devices, Chameleon added an edge testbed, CHI@Edge [19], with reconfiguration based on container deployment. The system thus supports multiple *flavors*: bare metal reconfiguration, VM deployment, and container deployment (for edge); although not all features are available in each flavor, all support API interfaces that allows them to connect to the same *central services*.

The core components of CHI (pronounced as "chee") have been described in detail in [22]. Briefly, users can allocate resources either on-demand or by making an advance reservation using a combination of OpenStack [14] Nova and Blazar services, the latter developed with significant contributions from the Chameleon team. The allocated resources can range across nodes, networks, and IP addresses [21] and are treated as *non-fungible entities* since many of our users want to allocate a specific resource in order to control for variability [20]. Once resources are allocated, bare-metal reconfiguration allows users to deploy their experiments *as close as possible to physical configuration* using OpenStack Ironic; users can use one of the bare metal images supported by the Chameleon team, or configure their own and snapshot it using a Chameleon-provided tool. Users can also configure network slices using the ExoGENI [4] libraries (to be subsumed by FABRIC [3] integration soon) integrated with Chameleon, or Bring Your Own Controller (BYOC) [8] functionality to experiment with software-defined networking. Configuration of complex experiment topologies such as virtual clusters or distributed networking experiments is supported through programmable interfaces: OpenStack Heat, or by the python-chi library [30] available via JupyterHub [1] integrated with the testbed.

Similar to many other projects, Chameleon provides central user services. They include *authentication via federated identity* using Globus Auth [2, 33]; the *user portal* which provides a central management interface for a user's profile information; *resource discovery* via a browseable catalogue aggregating fine-grained information about hardware available on individual sites; *availability calendar*, where users can see the availability of specific hardware at different times and locations. In addition, we provide a shared Jupyter environment that facilitates testbed access and can be used to package repeatable experiments [1], Trovi [32], a central repository of experiment notebooks and accompanying data and scripts where users can share and discover packaged experiments, and the Chameleon Daypass [26] that provides short-term access to the testbed for reproducibility purposes.

Unlike traditional research testbeds, Chameleon builds on a mainstream open source cloud system (OpenStack) rather than a custom implementation. This has a range of practical benefits including familiar interfaces for both users and operators, workforce development opportunity, compatibility with mainstream tools which facilitates portability to other clouds [27], the ability to leverage contributions of a large development community—as well as multiplying our impact by contributing to a widely used infrastructure ourselves. At the same time, OpenStack does not fully support our use case, necessitating reliance on tools developed by others—specifically, Grid5000 [5], ExoGENI [4], and the Jupyter projects – and development of our own. Among our own contributions, most notable are bare metal snapshotting (not available from the main OpenStack installation), the ability to authenticate via federated identity to all OpenStack services, more powerful instance customization (e.g., provisioning multiple SSH keys, also not available upstream), user-configurable OpenFlow Corsa switching for SDN (to be expanded to include P4), integration with Jupyter, and resource discovery services. We also made many modifications to OpenStack components themselves, in particular, Blazar, i.e., the service for making and managing advance reservations, Keystone, Horizon, Zun and Neutron. At present, we estimate that CHI is comprised of roughly 50% OpenStack and 50% of other systems and our adaptations.

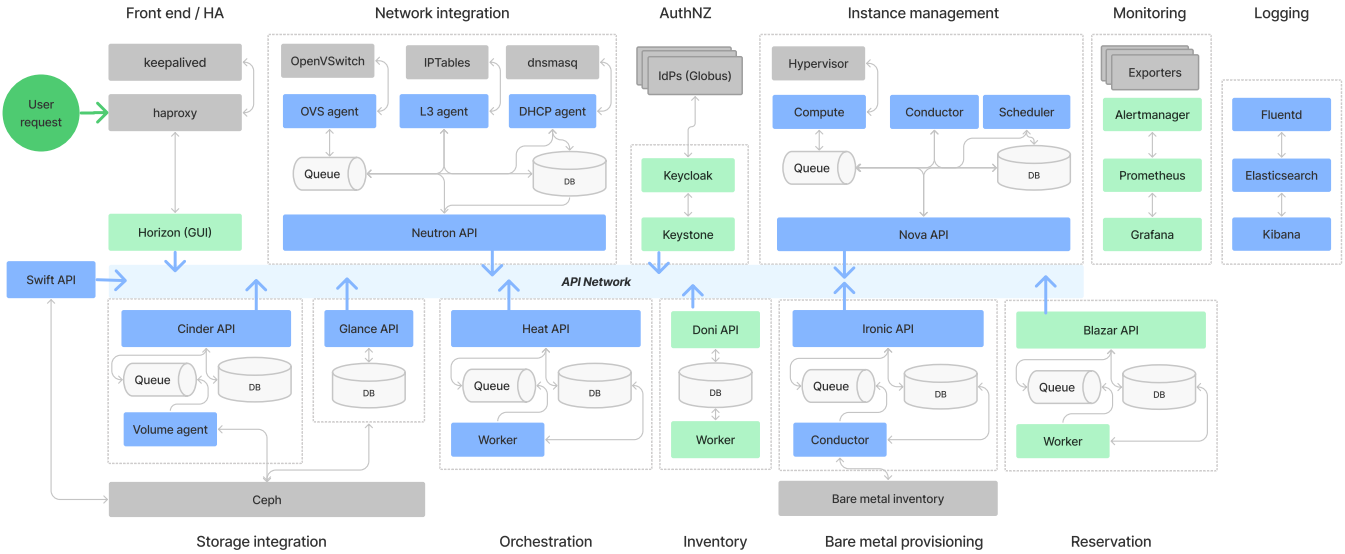


**Figure 1: CHI-in-a-Box services and relationships. The items in green denote additional services contributed by Chameleon, or services with significant capabilities not available upstream.**

## 3 COMPLEXITY AND COST

In its original implementation, Chameleon was deployed across two sites: the University of Chicago (UC) and the Texas Advanced Computing Center (TACC). While we wanted it to scale across more sites, this was not easy due to the complexity—and thus cost—that operating a testbed of this kind entails. One example of this complexity is networking: to deploy a user *instance* (bare metal node, VM, or container) the infrastructure has to dynamically connect it to potentially multiple networks; this relies on a range of features from physical network configuration to reliable infrastructure implementation to work in concert. Another is the correct management of security in a system where users get root permissions to instances they configure themselves; this means a large attack surface to protect and many attack types to defend against. Supporting bare-metal reconfiguration specifically increases the complexity even more as system operators cannot rely on an additional layer of abstraction that facilitates sandboxing and monitoring user actions. And while much of this complexity is inherent in any cloud infrastructure, a research cloud faces the additional challenge of rapid evolution, typically across the stack. Research clouds are thus operationally complex because they solve a complex problem; below we tease apart and describe in detail the dynamics of this complexity so that we may later illustrate how they are addressed by our packaging.

*Operational complexity.* Research testbeds entail managing many interconnected systems and services. For example, a control node of CHI typically hosts around 60 different services providing functions around network realization (DHCP, DNS, IPAM, NAT, VLAN provisioning), bare metal reconfiguration (iPXE, boot/power management, serial console proxy), and lifecycle management for all user-facing domains such as instances, security groups, subnets, SSH keys, disk images, etc., not to mention storage cluster systems to support user data persistence. Each of these services logs in a different way, has their own set of configuration options, and may need to be upgraded on a different schedule with conflicting dependencies between upgrades. Manually keeping track of it all across this many services requires broad knowledge, high qualifications, and makes understanding the state of the system at any given time difficult.

*Mapping to a variety of site configurations.* Since CHI operates as close as possible to physical configuration, the task of packaging and operating it on a variety of sites is challenging, as their physical configurations differ and are generally not designed with supporting computer science research in mind. In particular, not all sites can be configured to support all the features CHI provides: lack of specialized hardware (e.g., Corsa switches) and constraints on site network configurations are the most significant examples. This calls for packaging that not only supports a broad range of configurations and hardware, but can also allow operators to cleanly (i.e., without impacting other system components) "opt out" of supporting various CHI features. Further, site differences also have to be taken into account in operations so that configurations across sites can be consistent in some (project-wide) aspects, but divergent in others.

*End-User artifacts and support.* In most academic systems, administrators maintain one configuration, and users port their applications to run on this configuration. This situation is inverted in the cloud world: while in principle users want to manage their own configurations, they also expect providers to create and maintain a range of "base" images supporting specialized hardware (e.g., CUDA images for GPU), or orchestration/profiles for complex deployments. For example, Chameleon maintains 20 base images and 3 complex appliances (images + orchestration templates), many of them across different CHI flavors. In addition, the operations team needs to constantly bridge the gap between what a typical user needs to know about system administration and what they

actually know; this results in increased time for operations (e.g., preventing security incidents), support (fielding user questions), and user education.

*Maturity.* Having existed far less time, cloud computing is a less mature paradigm than e.g., HPC systems. Rapid rate of development implies less complete functionality at any given time and thus need for frequent upgrades (e.g., Ironic was not a part of OpenStack when Chameleon started), less stability in the system, and the need to frequently step in and fix or develop features our users need. Since the beginning of the project the UC team has spent 440 person-days contributing and reviewing OpenStack patches, 246 for Blazar alone [15] (this does not include effort that goes into features not contributed due to the niche scope of our system). Added to that is the challenge of managing OpenStack upgrades, a task that involves reading changelogs for 10+ separate projects, understanding what breaking changes or new assumptions may exist, researching migration paths for deprecated configurations, backporting unavoidable custom patches, and updating default configurations. Thus, while basing Chameleon on a mainstream system saved us much effort, general lack of maturity in cloud systems still resulted in additional operational cost.

*Evolution.* Scientific instruments need to evolve as the science they support evolves [12]; thus, the last and largest challenge of is the keeping up with the rapid rate of feature evolution needed by our community. Over the lifetime of the project, we had to adapt the system to add support for new hardware types, new user-requested features, new types of experiments (e.g., in networking [8]), and a whole new experimental domain in edge computing [19]. Discovering, designing, and implementing these adaptations and extensions requires not only deep and broad expertise and significant development effort; integrating them into the system such that its stability is preserved requires additional operational effort and processes. Even then, instabilities can manifest themselves due to the rate of evolution: one example is a bug in a third-party networking tool triggered by a routine and innocuous configuration change that resulted in a significant system instability and took a week of highly qualified effort to fix.

## 4 CHI-IN-A-BOX DESIGN

These challenges meant that operating CHI became costly early in the project: even routine operations like OpenStack upgrades would require significant effort and high level of expertise including skills in security, networking, and development. This made the system particularly vulnerable to personnel rotation, adding another risk factor to an already challenging operation. Further, despite the best efforts of the team, the configuration on the two operating sites was constantly diverging; it was thus impossible to replicate problems across sites and meant that each site had to solve the same problems from first principles, effectively doubling our operational costs. Since this would of course limit our potential to scale to more sites, we adopted what initially was simply an internal packaging of OpenStack – and eventually grew into CHI-in-a-Box, i.e., packaging of all of site-specific CHI components (whether OpenStack or not) as well as its operations model. Ultimately, the objective of the system became to bring the cost of installing and operating the testbed to a lower bound of a non-experimental "shrinkwrapped system" (such as e.g., Ceph), i.e., make the operational burden of individual sites comparable to that of a stable technology.

With this basic CHI-in-a-Box shape in mind, we defined the following target use cases:

(1) *Chameleon associate sites*, that join the Chameleon testbed, i.e., make their resources available to Chameleon users full time and leverage the central services and user support provided by the project.
(2) *Chameleon part-time associate sites*, that are associate sites with inventory that can only be committed on a part-time basis by defining up-front availability windows. This requires support for adding and removing resources dynamically and programmatically via *Bring Your Own Device (BYOD)*, a critical requirement for composable infrastructures.
(3) *Independent testbeds*, that configure new testbeds similar to Chameleon but unconnected to the project; they provide their own central user services and user support.

The most significant step in managing the challenges of an experimental testbed, was the adoption of a *hub and spoke model* in which one *hub* team (University of Chicago, in our case) takes on the responsibility for creating an internal release: integrates and tests OpenStack upgrades; develops, integrates, and tests any new experimental features; and then publishes them, along with documentation, site configurations, and relevant operational tools to a central location [10] – the *spoke* sites then independently download and install the software. This approach gives site administrators full autonomy in administering their own site, while relieving them of the burden of much of the complexity associated with rapidly developing research infrastructure.

We take this principle one step further and apply it to site configuration management: we express a site configuration as code (called "site config") and manage it via version control system (VCS); this configuration is effectively an overlay that adapts the core CHI configuration to a site's specific requirements, preserving the mapping of CHI to the site as new features and upgrades are released. Site configs contain an association of hostnames to the roles they will play in the deployment, and a set of configuration options and toggles expressed as YAML, which override default values used when invoking the provisioning tooling to, e.g., deploy or reconfigure a given service. For example, the site's hostname and public and private IP addresses would be contained in this file, as that is different for each site. Secrets such as TLS certificate keys and passwords are securely stored within the site config in a separate YAML file encrypted with a per-site key; this allows those to be committed and stored under VCS as well, useful in disaster recovery scenarios. Site configs reduce the integration surface area significantly as they allow site operators to express their instantiation of CHI as succinctly as possible: this makes rollbacks easier, adds an audit trail to identify when a possible regression emerged, limits the amount of work necessary to adopt future releases and upgrades – yet provides flexibility when it is needed to accommodate idiosyncrasies in the host environment.

While adopting the hub and spoke approach does not in itself solve the complexity, maturity, and evolution challenges, it largely compartmentalizes them to one team at the cost of moderate additional effort of packaging; that cost is paid once and leveraged

across many sites. To illustrate: it takes roughly one person month (assuming medium level of OpenStack experience) of core team development time to address all the challenges related to OpenStack upgrade described in Section 3 and create a new CHI-in-a-Box distribution; applying it on a sites usually takes only roughly one day of time and requires only enough experience to ensure the process completed smoothly (i.e., understand how to find and read logs), assuming support provided by the hub team. The fact that UC both packages and installs/operates CHI-in-a-Box avoids the "ivory tower" problem, giving us insights into pain points and allowing us to test proposed changes for regressions in performance or reliability.

## 4.1 Packaging and Installation

To address the challenge of mapping to different site configurations, we designed most CHI capabilities to be optional. For example, stitching [8] is only possible if the host environment has logical access to L2 stitchports, and isolated L2 networking can only be accomplished if CHI can manage switches on the network fabric: if either of these assumptions cannot be satisfied, these features are simply not available to users of the site. Each flavor of CHI deployment similarly has requirements on the underlying host infrastructure: for example, bare metal provisioning at minimum requires an out-of-band network due to reliance on IPMI/BMC while VM provisioning works best with some shared storage infrastructure (e.g., Ceph, NFS) and a stable number of nodes dedicated to running hypervisors.

To make navigating the intricacies of all of these combinations capable of clean separation, we took a two-pronged approach: first, we attempt to group functionality under coarse-grained configuration flags (e.g., "enable bare metal"), which then influences many other configuration options; secondly, we documented how to enable and configure each testbed capability separately, describing what the capability is, how it works, what its requirements are, how to enable it, and what further options for customization are available if desired. Combined, the two strategies give operators an "easy button" that can help get them started, and a springboard to more fine-tuning later.

## 4.2 Managing Inventory

Managing inventory tends to be overlooked in mainstream clouds because it is seen as an infrequent operation; resources are seen as fungible so that inaccuracies in inventory management carry less penalty. In research testbeds on the other hand, resources are not only non-fungible [20] but also versioned for reproducibility purposes so it is essential that resource information is up to date and consistent across all relevant infrastructure services including user-facing resource discovery and availability calendar. Small changes to the resources (e.g., disk or memory upgrades) need to be tracked and surfaced; failure to ensure this could lead our users to draw misleading conclusions when repeating experiments. Achieving a high level of accuracy and detail manually can be error-prone and onerous: it requires producing correct and detailed resource descriptions and consistently registering them with all the relevant infrastructure services, each requiring a different format. A reverse process takes place when retiring a node, not to mention the complexities of ensuring all services agree that a node is in maintenance (and thus not reservable/not in service) while an operator is fixing an issue. Finally, the requirement for BYOD support in our part-time associate site use case implies a composable system; this means support for making resources available or unavailable dynamically and potentially frequently.

To automate collection of hardware properties during enrollment or after changing some aspect of the hardware we use a modified form of Grid5000's g5k-checks [24] utility, which leverages ohai to introspect and collect a summary of hardware details (itself utilizing Linux utilities like lsblk, ethtool, and procfs), before transforming the output into a JSON description following a set schema. These JSON files are then checked into a Git repository, all of which is served via a g5k-api instance, which transforms the repository into a versioned HTTP API, consumable by remote clients.

To support adding and removing hardware to the testbed dynamically we developed an internal service called *Doni* [11]; it presents a single view of site hardware to operators, allowing them to add the hardware once and have the event reliably cascade through the relevant services. Doni implements the part-time associate testbed use case by managing dynamic inventory changes: once hardware is added, operators can define windows of availability that are communicated to users through the availability calendar. In addition, Doni helps simplify operational tasks: for example, when administrative changes are prevented by the system because the hardware is in use, Doni allows operators to queue up changes so that they can "set it and forget it", as opposed to setting alarms or reminders to perform updates when the hardware is no longer being used.

## 4.3 Operations

One of the most significant sources of complexity in operating a system with many services are inter-library dependencies and conflicts between them; this often prevents agile upgrades to select services or can lead to surprises during rollout that prolong planned maintenance windows. A general solution in this space is to *containerize* the services by isolating them in VMs or containers: this eliminates any dependencies or conflicts via file system isolation and allows the services to be managed independently – as well as greatly facilitates porting the services to new systems. Here, we were able to leverage the benefits of a mainstream implementation base by building on Kolla, an OpenStack project that provides a set of Docker build recipes and tooling that makes it simpler to assemble and customize the container images for each of the numerous OpenStack services our project uses.

In addition, CHI-in-a-Box also leverages the related Kolla Ansible project that implements Ansible automation around installation, reconfiguration, and upgrades of an OpenStack-based deployment and additionally makes implicit dependencies between service configurations more explicit. For example, if an additional networking service should be deployed when the bare metal reconfiguration service is enabled, this relationship is modeled in code; all the site operator needs to specify is whether to enable the capability and all required services are deployed and configured accordingly. Besides managing the OpenStack services, Kolla Ansible can also deploy helpful operations infrastructure such as centralized logging (all control nodes report logs for each service

to a central searchable Elasticsearch index) and fine-grained monitoring (Prometheus) along with user-friendly visualization GUIs for each (Kibana, Grafana). CHI-in-a-Box inherits any functionality provided by the Kolla Ansible project (included in CHI).

### 4.4 Monitoring, Detection, and Remediation

To further assist site operators, CHI-in-a-Box includes operational tools that assist with system monitoring, anomaly detection, and provide a variety of remediation routes. Specifically, we maintain a set of scripts called "hammers" [31] that work by testing propositions about the state of testbed and issue a few sharp hits in the right places to bring things back into order. For example, a hammer might ensures that nodes no longer reserved by any user are returned to the freepool, i.e., set of nodes available for reservation (this involves multiple steps and can potentially be interrupted.) Hammers are typically run as periodic Cron jobs or systemd timers, but can also be run ad-hoc by operators as needed. As most systems, we maintain a set of "smoke tests" that exercise various typical user flows on the system; since they are typically run outside of the system, they are not packaged as part of CHI-in-a-Box, but are open source and can be independently deployed; we run them periodically against CHI@UC and CHI@TACC via a Jenkins executor.

In cases where automatic action is not sufficient or possible, we extended Prometheus monitoring, adding alerts on certain signal metrics, which can be configured to log to Slack along with a hyperlink to a runbook hosted on our documentation portal describing the problem in more detail, and giving ideas on how to diagnose further and fix the root cause. Because runbooks are also stored in our source code repository, we can accept outside contributions and update information as the packaging changes over time, allowing Chameleon site operators to contribute to the body of knowledge about system operations.

### 4.5 End-User Services

In addition to the central services (Section 2), we maintain a set of user artifacts including images, orchestration templates, and notebooks that can be used on any site. CHI supports a variety of disk formats (e.g., raw, qcow2, AWS AMI) and we publish a set of default images in qcow2 format (which can be used in either KVM or for bare metal provisioning) based on the latest two stable releases for the CentOS and Ubuntu Linux distributions, and additional derived images that add various types of tools and utilities. We use OpenStack's diskimagebuilder to generate each tree of images and store the build configuration for each in a separate VCS repository to track changes over time; additionally, we have automated the build process via Jenkins, which we can not only trigger manually when releasing changes, but also will automatically trigger new builds when a new distribution version is released upstream. Initially disk images are published to a central public location; CHI-in-a-Box includes an operator tool that can sync a local site's default images to this central repository, updating the local site's image registry accordingly. These measures ensure that we can provide and port a relatively large number of centrally-supported images to many sites, giving users a range of artifacts to choose from and lessening the temptation to use insecure images.

Orchestration templates and notebooks, which users can extend and customize, can be used in conjunction with disk images to express a complex experiment topology. We package notebooks together with supporting example data and scripts in Trovi [32], the central Chameleon experiment saving/sharing repository. Both orchestration templates and notebooks can be ported across CHI sites, but each may need small updates to reflect the different hardware options available at a target site. Lastly, we provide centralized user support via a virtual helpdesk, with user support responsibilities shared between UC and TACC, and leveraged by all Chameleon associate sites.

## 5 CASE STUDIES

We describe below insights from CHI-in-a-Box deployments at two Associate Sites and one Independent Site.

### 5.1 Associate Site: NCAR

The National Center for Atmospheric Research (NCAR) is an NSF funded research center for university climate scientists from across the United States; our Computational and Information Systems Lab (CISL) maintains supercomputing resources to support this research and is home to the Cheyenne supercomputer, a 5.3PF HPE/SGI ICE cluster which provides the majority of computational power utilized by our community. As part of the procurement process at NCAR, we evaluated the ARM ThunderX2 processors as possible candidates for our next computational system, after which we wanted to make them available to other researchers for experimentation and we felt Chameleon provided a good avenue to do that. One of CISL's senior systems engineers, having many years of experience with various cluster managing software but no direct experience with OpenStack, performed the installation.

Chameleon staff were required to make a few adaptations to CHI-in-a-Box due to our site-specific network configuration and the unique hardware, including flat network support to accommodate our use of the side-band interface for BMC connectivity and expanding support to include ARM64 architectures and UEFI boot modes on the bare metal nodes. Two hardware issues had to be resolved by modifying the default configuration: secure erase was disabled, as the disks erroneously reported support for this feature, but would lock up during deployment; and power state monitoring was disabled, as the BMCs frequently reported transient error states. Both of these are likely due to the developmental and cutting-edge nature of these machines. Overall, Chameleon was no more difficult to install or configure than any other method used to setup these nodes before. The main difficulty came from the uniqueness of the hardware and the differences between the network configuration of the machines and what Chameleon expected. We plan for the ARM cluster to eventually be operated by student workers. It would be an excellent opportunity for a junior systems engineer to gain experience with the technologies used by Chameleon and learn how to support a semi-production environment.

### 5.2 Associate Site: EVL

The Electronic Visualization Laboratory (EVL) at the University of Illinois Chicago (UIC) is an interdisciplinary research laboratory in the Computer Science department that specializes in visual data

science and advanced cyberinfrastructure. EVL configured a CHI-in-a-Box bare metal site using Chameleon's phase 1 hardware (22 Haswell nodes) donated as part of the Chameleon legacy program; the deployment supports a full complement of bare metal features with the exception of stitching and BYOC (due to logistical complexity of configuring layer 2 connectivity via campus networking). CHI@EVL was deployed by an undergraduate student who has experience working on several EVL compute clusters, and most recently deployed EVL's NSF-funded COMPaaS system [7, 23], a 24-node Composable Infrastructure using Kubernetes and Metal as a Service (MaaS) provisioning service. The team's OpenStack experience was limited to understanding the fundamental vocabulary of the platform.

In installation, we encountered problems with an incompatible default setting (confounded by insufficient error propagation in OpenStack) and the inherent complexity of multi-tenant bare metal networking. At the same time, the CHI-in-a-Box network is significantly simpler to understand when compared to our experience with Kubernetes, which creates many more virtual interfaces and opaque networks. More meaningful error messages and expanded documentation, including a operations verification checklist, will help mitigate this type of issue going forward. In addition, device removal and re-enrollment exposed a propagation bug in the Doni service. Chameleon's team helped debug and fix this issue, but this particular experience was less seamless than Kubernetes, which can adapt to most changes in cluster state without manual intervention. All in all, we found the deployment of CHI-in-a-Box seamless and impressive in its scope, providing overall a more robust cluster infrastructure system than our experience with Kubernetes and MaaS. Based on our experience, we believe that once the above-mentioned issues are resolved, the deployment process will be faster and more hands-free when compared to other "shrink wrapped" infrastructures.

### 5.3 Independent Site: ARA

As a part of the NSF Platforms for Advanced Wireless Research (PAWR) program, ARA [17] is a wireless testbed being deployed in central Iowa to support research, education, and innovation in advanced wireless and applications in domains such as precision agriculture and rural education. ARA decided to adopt CHI-in-a-Box as a baseline testbed implementation because it already has mature support for edge-to-cloud provisioning of compute, storage, and network fabrics, and provides end-to-end experiment control and user services. Further, since CHI-in-a-Box is based on a mainstream open-source platform, we could extend its resource management capabilities to support management of wireless resources (e.g., SDRs with related attributes such as frequency, transmission power, and gain), wireless BYOD, wireless virtualization, and experiment monitoring and interference control. The deployment was performed by a first-year Ph.D. student with extensive networking background but no prior experience with OpenStack.

During deployment, the ARA team encountered issues due to hardware and configuration differences; in particular, differences in networking hardware required modification of Neutron plugins and drivers, and the configuration of interface bridges required different handling. These are hard to address for operators with no prior OpenStack experience and required support and collaboration from the Chameleon team. We contributed comments that resulted in general improvements to the documentation and inclusion of a baseline deployment diagram that will make this easier going forward. An important finding was that CHI does not provide all the capabilities our testbed requires, in particular support for wireless experiments and greater flexibility in management of network resources. It represents a good starting point – that we plan to extend to integrate field-deployed wireless resources with edge and cloud resources – and contribute these features back to CHI-in-a-Box.

## 6 RELATED WORK

Some research testbeds such as GENI [6] and ORBIT [16] have provided packaging of their infrastructure though it was generally targeted to a fixed hardware configuration, usually shipped to each new site. Emulab/CloudLab [13, 34] packaging is the closest to our approach, but to our knowledge does not abstract site configuration from packaging, or employ containerization or BYOD techniques described here. Further, as an in-house implementation it does not leverage the advantages that mainstream implementation brings (e.g., we were able to add the UEFI image implementation required by NCAR very quickly because of existing OpenStack implementation).

Research clouds such as Jetstream [29] and the Massachusetts Open Cloud [25] operate OpenStack clouds with some features for science but do not develop their own packaging. StackHPC provides a commercial OpenStack distribution focused on research [28] but it targets mainly virtualized clouds operated for domain sciences rather than bare metal reconfigurable experimental computer science systems testbed that is our focus. Finally, there are multiple OpenStack projects that support or facilitate various aspects of packaging like the Kolla-Ansible projects described earlier; we leverage them in our packaging as appropriate.

## 7 CONCLUSIONS

When building distributed research infrastructure on top of a mainstream system it is tempting to use a public distribution and have each site apply the relevant adaptations. We found that where there is a high level of research-specific customization creating an *internal* distribution (CHI-in-a-Box) results in significantly lesser cost of testbed operation. In this paper we explain why, and share the strategies of hub and spoke management, containerization, and BYOD inventory management that make our approach feasible. We also show how the use of mainstream infrastructure lowers the cost of not only building and using research infrastructure but also *operating* it through support for containerization, image management tools, or access to a broad range of capabilities against future needs.

The three presented use cases illustrate different motivations and deployment scenarios for this type of research infrastructure and provide an evaluation of our approach. Working with them, we revisit many of issues contributing to complexity and therefore the cost of operating research testbeds: the need to adapt the system to new hardware, mapping to a variety of site configurations, and demand for mature and refined tooling. Despite those challenges however in all cases institutions were able to provide a new

research capability quickly, with relatively little effort, and given little previous experience with similar systems. Most importantly, in all cases, the contributions inherent in a new installation became part of common development further helping to mitigate the cost of research infrastructure.

## ACKNOWLEDGMENTS

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 as well as by the NSF award 2130889 and NIFA award 2021-67021-33775.

## REFERENCES

[1] Jason Anderson and Kate Keahey. 2019. A Case for Integrating Experimental Containers with Notebooks. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Sydney, NSW, Australia, 151–158. https://doi.org/10.1109/CloudCom.2019.00032

[2] Jason Anderson and Kate Keahey. 2022. Migrating towards Single Sign-On and Federated Identity. In *Proceedings of the Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery.

[3] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang Ching Wang, Tom Lehman, Paul Ruth, and Ewa Deelman. 2019. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing* 23 (2019), 38–47. Issue 6. https://doi.org/10.1109/MIC.2019.2958545

[4] Ilia Baldine, Yufeng Xin, Anirban Mandal, Paul Ruth, Chris Heerman, and Jeff Chase. 2012. ExoGENI: A multi-domain infrastructure-as-a-service testbed. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering* 44 LNICST (2012), 97–113. https://doi.org/10.1007/978-3-642-35576-9_12

[5] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Perez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. 2012. Adding Virtualization Capabilities to the Grid'5000 Testbed. *Communications in Computer and Information Science* 367 CCIS (2012), 3–20. https://doi.org/10.1007/978-3-319-04519-1_1

[6] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A federated testbed for innovative network experiments. *Computer Networks* 61 (3 2014), 5–23. https://doi.org/10.1016/J.BJP.2013.12.037

[7] Maxine Brown, Luc Renambot, Lance Long, Timothy Bargo, and Andrew E. Johnson. 2019. COMPaaS DLV: Composable Infrastructure for Deep Learning in an Academic Research Environment. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, Chicago, IL, USA, 1–2. https://doi.org/10.1109/ICNP.2019.8888070

[8] Mert Cevik, Paul Ruth, Kate Keahey, and Pierre Riteau. 2019. Wide-area Software Defined Networking Experiments using Chameleon. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, Paris, France, 811–816. https://doi.org/10.1109/INFCOMW.2019.8845093

[9] Chameleon. 2022. Chameleon main page. Retrieved Feb 16, 2022 from https://chameleoncloud.org/

[10] Chameleon. 2022. CHI-in-a-Box source repository. https://github.com/ChameleonCloud/chi-in-a-box

[11] Chameleon. 2022. Doni: OpenStack Inventory Service. https://github.com/ChameleonCloud/doni

[12] Peter Couvares, Kate Keahey, and Frédérique Marion. 2021. Finding the Gravitational Wave: A History of Discovery Written in Software. *SoftwareX* 15 (2021), 100715. https://doi.org/10.1016/j.softx.2021.100715

[13] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. USENIX Association, Renton, WA, 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[14] OpenInfra Foundation. 2022. OpenStack Components. https://www.openstack.org/software/project-navigator

[15] OpenInfra Foundation. 2022. Stackalytics. https://www.stackalytics.io/?release=all&company=universityofchicago&metric=person-day

[16] Abhimanyu Gosain and Ivan Seskar. 2017. GENI wireless testbed: An open edge ecosystem for ubiquitous computing applications. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, Kona, HI, USA, 54–56. https://doi.org/10.1109/PERCOMW.2017.7917520

[17] Hongwei Zhang et al. 2021. ARA: A Wireless Living Lab Vision for Smart and Connected Rural Communities. In *WiNTECH*. ACM, New Orleans, LA, USA, 9–16.

[18] Kate Keahey. 2020. The Silver Lining. *IEEE Internet Computing* 24, 4 (2020), 55–59. https://doi.org/10.1109/MIC.2020.3013361

[19] Kate Keahey, Jason Anderson, Michael Sherman, Zhuo Zhen, Mark Powers, Isabel Brunkan, and Adam Cooper. 2021. Chameleon@Edge Community Workshop Report. https://doi.org/10.5281/zenodo.5777344

[20] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbah, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, online, 219–233. https://www.usenix.org/conference/atc20/presentation/keahey

[21] Kate Keahey, Pierre Riteau, Jason Anderson, and Zhuo Zhen. 2019. Managing Allocatable Resources. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, Milan, Italy, 41–49. https://doi.org/10.1109/CLOUD.2019.00019

[22] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2019. Chameleon: A Scalable Production Testbed for Computer Science Research. In *Contemporary High Performance Computing*. CRC, Boca Raton, 123–148. https://doi.org/10.1201/9781351036863-5

[23] Lance Long, Tim Bargo, Luc Renambot, Maxine Brown, and Andrew Johnson. 2022. Composable Infrastructures for an Academic Research Environment: Lessons Learned. (3 June 2022). submitted.

[24] David Margery, Emile Morel, Lucas Nussbaum, Olivier Richard, and Cyril Rohr. 2014. Resources Description, Selection, Reservation and Verification on a Large-Scale Testbed. In *Testbeds and Research Infrastructure: Development of Networks and Communities*, Victor C.M. Leung, Min Chen, Jiafu Wan, and Yin Zhang (Eds.). Springer International Publishing, Cham, 239–247.

[25] MOC. 2022. Mass Open Cloud – An Open Cloud Exchange Public Cloud. https://massopen.cloud/

[26] Mark Powers. 2022. Interactive Science Made Easy with Chameleon Daypass. https://www.chameleoncloud.org/blog/2022/01/24/interactive-science-made-easy-with-chameleon-daypass/

[27] Paul Ruth, Kate Keahey, Mert Cevik, Zhuo Zhen, Cong Wang, and Jason Anderson. 2021. Overcast: Running Controlled Experiments Spanning Research and Commercial Clouds. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE.

[28] StackHPC. 2022. Kayobe: An Introduction. https://www.stackhpc.com/pages/kayobe.html

[29] Craig A. Stewart, Timothy M. Cockerill, Ian Foster, David Hancock, Nirav Merchant, Edwin Skidmore, Daniel Stanzione, James Taylor, Steven Tuecke, George Turner, Matthew Vaughn, and Niall I. Gaffney. 2015. Jetstream: a self-provisioned, scalable science and engineering cloud environment. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure (XSEDE '15)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2792745.2792774

[30] Chameleon Team. 2022. Chameleon Cloud Python API. https://python-chi.readthedocs.io/en/latest/

[31] Chameleon Team. 2022. Hammers: Percussive Maintenance. https://chameleoncloud.gitbook.io/chi-in-a-box/operations/chameleon-tools/hammers

[32] Chameleon Team. 2022. Trovi: Practical Open Reproducibility. https://chameleoncloud.gitbook.io/trovi/

[33] Steven Tuecke, Rachana Ananthakrishnan, Kyle Chard, Mattias Lidman, Brendan McCollam, Stephen Rosen, and Ian Foster. 2016. Globus auth: A research identity and access management platform. In *2016 IEEE 12th International Conference on e-Science (e-Science)*. IEEE, Baltimore, MD, USA, 203–212. https://doi.org/10.1109/eScience.2016.7870901

[34] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*. USENIX Association, ACM, Boston, MA, 255–270.