Introducing Configuration Management Capabilities into CloudLab Experiments

Dmitry Duplyakin
Computer Science
University of Colorado, Boulder, USA
dmitry.duplyakin@colorado.edu

Robert Ricci
School of Computing
University of Utah, Salt Lake City, USA
ricci@cs.utah.edu

***Abstract*—Users of CloudLab (and other GENI-derived testbeds) commonly use image snapshots to preserve their working environments and to share them with other users. While snapshots re-create software environments byte-for-byte, they are not conducive to composing multiple environments, nor are they good for experiments that must run across many versions of their environments with subtle differences. This paper describes our initial work on an alternative experiment management system. This system is built on expendable instances of the Chef configuration management system, and can be used "on top of" existing testbeds.**

Keywords—testbed; cloud; configuration management system; reconstructability.

I. INTRODUCTION

Software environments in modern distributed systems demonstrate great diversity and high complexity. Hypervisors, virtual machines (VMs), cloud frameworks, containers, databases, and computing frameworks – these are some of the types of components used in modern software stacks. Individual components need to efficiently utilize available hardware resources as well as interact with one another. Integration of independently designed and developed tools with cutting-edge versions and capabilities is particularly challenging.

CloudLab [1] is an NSF-funded flexible scientific infrastructure for supporting fundamental advances in cloud architectures and applications. It is designed to facilitate research and development involving cutting-edge software environments. CloudLab provides root-level access to heterogeneous resources at three geographic locations: datacenters at the University of Utah, Clemson University, and the University of Wisconsin. CloudLab users combine resources from these sites into environments to experiment with architectures and software of their choice. CloudLab is built on the Emulab software base[16], which has evolved to incorporate ideas from the GENI [5] facility.

The Adaptable Profile-driven Testbed (Apt) [2], also developed and hosted at the University of Utah, is designed with an emphasis on consistency, transparency, and repeatability. Apt and CloudLab are tightly integrated: experiments on Apt and CloudLab can be managed through the CloudLab portal [3], and in this study we treat Apt as another cluster within the CloudLab environment. The two systems share the following concepts:

- *profiles* – XML-based RSpec [4] documents (developed as part of the GENI [5] project) describing specific experiment hardware and software. The purpose of these profiles is to describe an environment that is either used repeatedly by one user to run many experiments, or is used as a way for users to share software artifacts with one another.
- *experiments* – instances of profiles. When a profile is instantiated, its specification is realized on available resources that satisfy the environment described in the RSpec. The experiment owner is granted root-level access to these resources for a fixed period of time.
- *snapshots* – saved copies of node filesystems [6]. Users create these full-disk custom images when they need to preserve custom configurations in their experiments. They create new ones or update the original profiles to launch future experiments with their snapshots rather than images provided by the facility.

The resource allocation model on CloudLab is built around temporary leases: every resource, after being utilized by one user, is reclaimed and given to another user. If "setting up" the environment to run an experiment is complicated and/or time-consuming, the user is encouraged to snapshot the configured nodes in order to instantiate identical configurations with no additional effort. The snapshot model fits will the "build once, test repeatedly" experimentation, where users perform the bulk of necessary configuration work at once, snapshot the customized nodes, and proceed to testing: simulate workloads, run benchmarks, gather utilization statistics, visualize it, etc.

In contrast, while developing experiments to analyze performance and energy efficiency of the CloudLab hardware, we encountered a situation where the selected software stack continually evolved. We frequently added new and modified existing software components. We confronted the following drawbacks:

- Custom images cannot be automatically "merged". If Image A and Image B each have a different set of tools installed, there is no simple way to create an Image C that combines both sets of tools.
- Much effort is required to evaluate multiple versions of the selected software. For instance, to evaluate the impact of a set of compiler options on the performance of a particular tool, we need to rebuild all performance-sensitive tools multiple times with different option sets, with each requiring an additional custom image.

The coarse granularity of the snapshotting mechanism is the root cause of these drawbacks. With no fine-grained mechanisms, users encounter the described overhead when developing complex software environments, contributing to the rapid growth of the number of snapshots. More importantly, if we share our snapshots with other CloudLab users, we cannot expect them to reassemble those snapshots in order to reuse individual tools. This raises a platform-level question: how can CloudLab support efficient tool reuse among its users? Since CloudLab has over 800 users and 1,000 profiles after one year of operation, the importance of this question cannot be overstated.

In this paper, we describe our design and implementation of the CloudLab infrastructure for efficient management and reuse of software components. We describe the software artifacts we have developed, which CloudLab users can leverage and customize. Using the developed profiles and custom scripts, we streamline the creation of experiments with instances of a configuration management system called Chef [7]. We demonstrate how we use expendable, short-term instances of Chef to manage nodes inside CloudLab experiments and consistently build complex software environments. By describing our progress with orchestration of software of our choice using Chef, we aim to set an appealing example for other users and encourage them to invest effort in the development of infrastructure code. We also demonstrate how we structure the developed infrastructure code to increase its usability.

II. MOTIVATION

The experiment that we use as motivation and example in this paper is one that facilitates performance analysis of the CloudLab hardware using the GCC compiler, OpenMPI for message passing, the PETSc [8] library with routines for scientific computing, and HPGMG [9], a robust and diverse benchmark that is representative of HPC applications. (We do not assume reader familiarity with these tools in this study.) We suppose that each tool can be properly administered (e.g., using instructions from the corresponding websites) and treat them as components of a complex software environment. In such environments, issues related to customization and composability are among our main concerns.

After we install and configure these tools on a CloudLab node, we can snapshot the node. With this snapshot, we can recreate this configuration in future experiments. However, as soon as we add higher-level tools for execution management, in our case SLURM [10], we need to create two images: one for the SLURM controller node, and another for compute nodes. A distributed filesystem, such as NFS, requires the server- and client-side images as well. The number of images begins to multiply even further if we consider running experiments on different hardware architectures. Thus, the Utah cluster is built with 64-bit ARMv8 CPUs, Wisconsin – Haswell CPUs, Clemson – Ivy Bridge CPUs, and APT – Sandy Bridge CPUs. To optimize performance, we must build libraries and benchmarks with cluster-specific compilation flags. The desired performance analysis requires the configuration of 4 experiments, each with at least 2 SLURM nodes. This amounts to 8 node configurations with unique optimizations applied.

We expect the software environment to evolve over time. In the worst-case scenario, we will encounter the overhead of applying each modification to the created images up to 8 times. For instance, over the course of our experiments we evaluated three versions of GCC: 4.8.2, part of the the default Ubuntu 14.04 image, the 5.2 version with performance improvements for the hardware on the Utah clusters, and also the latest 5.3 version. We performed limited testing of several configurations with these compilers, but chose to search for an efficient alternative to the manual creation of all 24 node configurations. We also plan to equip our experiments with tools for tracking instantaneous power draw and the total energy consumed over time (described in section 5). Similarly, a stand-alone image with such tools will be insufficient; instead, we intend to run the energy-focused tools alongside the performance tools. We would need to create many configurations by composing these sets of tools, each time with unique modifications applied to either of the sets.

Aiming to reduce the overhead in the configuration of our experiments, we decided to invest effort in automation of installation procedures. We consider components, such as SLURM and NFS, for which the installation processes are exactly the same across all node types and independent of the rest of the software environment. After we develop the automated installation scripts once, we can run them in numerous configurations with minimal effort. In contrast, we need to incorporate platform-specific flags into our scripts for building high-performance libraries and benchmarks. Despite the difference in the flags, many actions in those scripts should be similar for different platforms. A single investment in configuration scripts will help save time in configuring future experiments, even if we need to adapt script parameters for particular platforms. We are confident that automation will reduce the amount of overhead, as well as greatly increase consistency. The developed code should also yield more transparency comparing to a growing number of custom images.

III. APPROACH

In order to systematize the outlined automation effort, we leverage a configuration management system (CMS). CMSes are built around the "infrastructure as code" principle, which dictates that all administrative procedures must be implemented as code and maintained, versioned, shared, and reused similar to other types of software. Embracing this code-centric vision, we plan to establish a public repository with the CloudLab infrastructure code, encourage community members to contribute, and incrementally expand the set of supported components.

In our previous work [11], we used a CMS called Chef (developed by Chef Software, Inc.) to manage a small group-owned experimental HPC cluster. In that environment, Chef proved to be a powerful tool with support for a variety of administrative procedures. We showed how Chef can be deployed in highly available configurations with increased reliability and longevity, which are desired in production scenarios. In contrast, this study investigates creation of expendable, short-term CMS deployments, which correspond to the CloudLab resource model, where nodes are allocated to users on a temporary basis. We enable the following workflow for the CMS-based experiment management: 1) a user launches an experiment with a fully functional instance of the selected CMS, 2) he or she develops infrastructure code and uses the CMS to execute it on the nodes under his control, 3) before the experiment expires, preserves his or her code, and 4) at later times, creates new instances of the CMS, obtains copies of the code, proceeds to manage new experiments, and continues the code development.

Our experience suggests that Chef is an appropriate system for this workflow. We attempt to automate the installation of necessary components and dependencies to create complete, working Chef environments. Much of the deployment process will be hidden from the user. To obtain identical instances of Chef in steps 1 and 4, CloudLab users will simply choose a profile and instantiate it via the CloudLab portal. They may run several instances of the profile simultaneously on the same or different clusters.

We must pay special attention to how we develop infrastructure code in order to increase its usability. In Chef, infrastructure code is arranged into so-called *cookbooks*, which include one or more Ruby scripts called *recipes*. Recipes typically use Chef *resources,* which are high-level constructs specifically designed to describe many common administrative tasks such as install a package, download a file, mount a filesystem, etc. Even though Chef recipes can include arbitrary Ruby code, resources help reduce the amount of Ruby code and spare the recipe developers from being Ruby experts.

While developing cookbooks in step 2, we focus on the following goals:

- Develop the cookbooks which run on the nodes of user choice with no modification. While running these cookbooks, novice users will install the corresponding components and practice the workflow that is the same all cookbooks: assign cookbooks to nodes, run the cookbooks, and check the output.
- Ensure that advanced users can perform customization with minimal effort. Such parameters as version numbers, URLs of packages, compilation flags, installation paths, etc., need to be easily customizable rather than hard-coded.
- Transparently support different platforms. Where possible, moving from one cluster to another should require only slight parameter changes (e.g., platform-specific compilation flags) and minimal modification in the code. Additionally, we should avoid code redundancies: rather than developing cookbooks such as ARM-*X*, IvyBridge-*X*, and Haswell-*X* with similar code, we should develop a single cookbook for *X* which is capable of supporting *X* on all appropriate architectures.

Preservation of the code in step 3 of the workflow is straightforward, since all Chef cookbooks and other code artifacts are developed inside *chef-repo*, a directory that is version controlled by default. The code can be “pushed” to a public repository, e.g., hosted at GitHub, and later “pulled” on the same or ad different Chef server.

IV. ARCHITECTURE

A minimal Chef deployment includes four components – a server, a client, a workstation, and a chef-repo – all installed on the same node. The command line utility called *knife* plays the role of the workstation and provides an interface to server operations: bootstrap a client (i.e. install necessary packages and register the client with the server), assign cookbooks to the client, and query the environment, etc. The server is a central authority for storing cookbooks and cookbook-to-node assignments. The client contacts the server to obtain the latest configuration when the “chef-client” command is issued (Chef supports the pull model, as opposed to the push model used in other CMSes). The server returns the assigned cookbooks, and the client executes them. In multi-node environments, each node runs a client and receives updates from the server.

We developed a CloudLab profile called Chef-Cluster [12]. It provides uncomplicated control over the deployment and configuration of Chef environments using the profile parameters, including:

- *N*, the desired number of client nodes,
- URL for the `chef-repo` used for this experiment,
- cookbooks obtained from Chef Supermarket [13] – the repository with many cookbooks developed by the global community of Chef developers,
- a set of cookbooks assigned to all clients.

When instantiating Chef-Cluster via the portal, we choose appropriate values for these parameters and proceed to the experiment placement step. We select the CloudLab site on which we prefer to instantiate Chef-Cluster with the specified parameters. The portal displays the experiment’s physical topology where all created nodes are interconnected with a single experiment network. In contrast, the logical structure inside the experiment is such that all client nodes can be configured from the server node. Fig. 1 illustrates both the physical (in black) and logical structure (in red) inside an experiment with N=4. Since the server node also runs a client, we can use *knife* on the server node to configure all five experiment nodes.

Once the cluster is selected and the experiment name is specified, the experiment is launched. At the back end, a Python script generates an RSpec using geni-lib [14]. The RSpec is populated with the XML elements that describe the environment for the experiment. Additional elements are appended to the RSpec for the described parameters, and the node attributes “Install Tarball” and “Execute Command” are used to download and run startup scripts.

To produce a working Chef environment with the shown logical structure, we need to convert a pool of

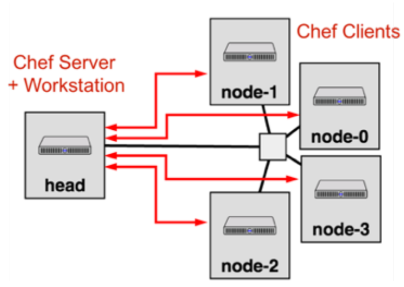

Fig. 1. Physical and logical structures in a 5-node experiment – instance of the Chef-Cluster profile. The small box in the middle represents a networking switch connecting all nodes.

interconnected but independent nodes into a cluster environment with centralized management. The startup scripts perform two tasks: enable the server to issue remote commands to the clients, and inform the server about the available clients. For the former, our scripts use the *geni-get key* call to generate experiment-specific public and private keys. These keys are the same among all nodes, enabling secure host-based authentication within the experiment. The rest of our scripts run on the server node and leverage this authentication. For the latter, our scripts process experiment metadata included in the *manifest*. In contrast with the abstract requests in the profile RSpec, experiment manifests contain information about the actual experiment resources. Obtained via the *geni-get manifest* call, this information allows our scripts to register the client nodes with the Chef server. The scripts also assign the specified cookbooks to the nodes based on the specified parameter value, which becomes available in the manifest.

The configuration process usually takes several minutes. When it completes, the script sends a notification via email to the user who launched the experiment. To reduce the learning curve for novice users, the email includes several knife commands for querying the created Chef experiment and their output. For instance, the output of *knife cookbook list* and *knife node list* reports which cookbooks and nodes are recognized by the server.

After inspecting the environment, the user can proceed to configuring the nodes using their assigned cookbooks. We intentionally do not automate this step in our scripts, allowing the user to log into the nodes in the desired order, trigger the configuration processes by issuing the *chef-client* command, and examine the output.

V. Implementation

We established the `emulab/chef-repo` repository [15], which provides public access to the cookbooks developed in accordance with the goals from section 3. Currently, the repository includes 13 Chef cookbooks and 24 recipes, which focus primarily on the configuration of performance- and energy-related tools. CloudLab users can fork this repository, customize the existing cookbooks, and use the URLs of their derived versions of the repository when they create their Chef-Cluster experiments. Pull requests can also be submitted; the new code will be thoroughly tested before becoming a part of the official repository.

If [15] is selected when Chef-Cluster is instantiated, the repository is cloned and the cookbooks listed in Table 1 become available on the created server. Even though we developed these cookbooks to configure our specific software stack, some of them can be immediately useful to other CloudLab users. For instance, `emulab-gcc`, `emulab-R`, and `emulab-nfs` are general-purpose cookbooks which install the latest versions of the corresponding components with minimal effort. To enable easy customization, cookbooks optionally include *attributes* – the variables which change the cookbook's behavior. We added attributes in our cookbooks for setting version numbers, URLs of packages, compilation flags, installation paths, among other parameters.

We use the `emulab` prefix in the names of our cookbooks to emphasize that they are developed for Emulab [16] and derived testbeds such as CloudLab and Apt. This helps distinguish them from the Supermarket cookbooks, which are installed in the same directory on the server node. Our cookbooks leverage four Supermarket cookbooks – `apt`, `mysql`, `nfs`, `apache2` – which are sufficiently documented and actively supported by many Chef developers.

The cookbooks `emulab-R`, `emulab-shiny`, and `emulab-powervis` demonstrate an example of cookbook chaining. CloudLab provides access to power consumption traces from every physical server. To simplify the usage of this data, we developed PowerVis, a dashboard for analysis and visualization of power traces. This dashboard is an application in Shiny, the environment for interactive web applications powered by the R statistical

TABLE I. Developed Chef Cookbooks, Their Dependencies, and Components.

Cookbooks	Description	Required Cookbooks	Recipes	Attributes
`emulab-gcc`	Install the latest available version of GCC	`apt`	1	N/A
`emulab-openmpi`	Install and make available OpenMPI	`emulab-gcc`	1	4
`emulab-slurm`	Configure SLURM controller and compute nodes	`apt, mysql`	3	18
`emulab-nfs`	Install NFS server and clients, export/mount directory	`nfs`	3	12
`emulab-petsc`	Obtains and builds PETSc	`emulab-openmpi`	1	2
`emulab-hpgmg`	Obtains and builds HPGMG	`emulab-petsc`	1	4
`emulab-R`	Install the latest available version of R package	`emulab-gcc`	3	1
`emulab-shiny`	Install Shiny server and necessary R library	`emulab-R`	1	2
`emulab-powervis`	Install PowerVis and its dependencies	`emulab-shiny, apache2`	1	1

language. With the help of dependencies expressed in the cookbook metadata files, `emulab-powervis` calls `emulab-shiny`, which, in its turn, calls `emulab-R`. Therefore, `emulab-powervis` is a top-level cookbook which installs all necessary underlying components and enables analysis and visualization of the experiment-wide power data. Similarly, `emulab-hpgmg` consolidates the installation of HPGMG, PETSc, OpenMPI, and GCC.

Since a single model cannot fit perfectly all scenarios, Chef provides the developer with enough flexibility to organize the code into structures that match specific environments and applications. Some of our cookbooks support different platforms with the help of multiple recipes. For instance, inside `emulab-R`, the default cookbook identifies the node architecture and calls either the `aarch64.rb` (on ARMv8 nodes) or the `x86.rb` (on the rest of the nodes) recipe to take platform-specific actions. In contrast, `emulab-hpgmg` uses a single recipe which selects optimal compilation flags from its attributes and runs the same code on all platforms.

VI. EXPERIENCE

We used Chef to manage configurations inside experiments on each of the CloudLab sites and turned many experiments into fully functional computing clusters with the described performance and energy tools. We often incrementally built our clusters: we started by installing HPGMG (and its dependencies); after single-node tests, we enabled multi-node runs by installing SLURM and NFS; then, we installed PowerVis to perform power and energy analysis of the performed runs.

Using the node names shown in Fig. 1, below we illustrate how we configured our clusters. Initially, we ran:

```
# knife node run_list set head "emulab-hpgmg"
```

Then, the “chef-client” command executed on *head* triggered the execution of the assigned `emulab-hpgmg` cookbook. These two steps, the assignment and the configuration, can easily be repeated many times throughout the experiment lifecycle. We can use the *add* function to modify *run_lists*, the lists which define what is executed on individual nodes:

```
# knife node run_list add head "emulab-slurm"
# knife exec -E 'nodes.find("name:node-*") {
|n| n.run_list.add("emulab-slurm") ; n.save}'
```

While the first command assigns `emulab-slurm` to the server node (head), the second command assigns the same cookbook to the client nodes (node-[0-3]) using the wildcard matching for the node names. The client nodes can now be configured one at a time. Alternatively, if the order is insignificant, we can trigger the batch update by running “`knife ssh 'name:*' chef-client`” on the server node (emulating the push-based model). We can assign additional code artifacts to our nodes:

```
# knife exec -E 'nodes.find("node:*") {
|n| n.run_list.add("<artifact>") ; n.save}'
```

where `<artifact>` can take one of these forms: “`recipe [<cookbook name>::<recipe name>]`”, “`recipe [<cookbook name>]`”, or “`<cookbook name>`”. In the last two cases the cookbook’s `default.rb` recipe is used. Moreover, we can also use the “`role[<role name>]`” notation to assign Chef *roles* – the higher-level code artifacts which set attributes, complementing and possibly overriding attributes specified inside individual cookbooks, and consolidate ordered groups of cookbooks.

In our computing clusters configured inside CloudLab experiments, we typically ran `emulab-hpgmg`, `emulab-slurm`, `emulab-nfs`, and `emulab-powervis` (along with their dependencies) on the server nodes and executed `emulab-hpgmg`, `emulab-nfs`, and `emulab-slurm` on the clients. After performing the aforementioned incremental development and thoroughly testing each component, we streamlined this assignment with the help of custom roles. The developed *controller* and *compute* roles (available at [15]) consolidated the server- and the client-side cookbooks. We also ensured that the individual cookbooks included in these roles work together. For instance, the roles set the attributes such that HPGMG, PETSc, and OpenMPI are installed inside the directory used as the cluster NFS share. These tools, once installed on the server, become available on all clients when the NFS share is mounted. Consequently, we can reduce the number of cookbooks included in the compute role.

The controller and compute roles have sufficiently satisfied our configuration needs. We used these roles to consistently create computing clusters at different CloudLab sites and performed many benchmarking experiments. While running HPGMG, we gained better understanding of the benchmark code and the properties of the available hardware. In addition to the performance analysis, we processed power traces using PowerVis. Not only can we identify the CloudLab site that is the most appealing for running HPGMG in terms of Watts and Joules, we can use energy efficiency as the deciding factor. Thus, for our 8-core HPGMG runs, the estimated energy efficiency of the Utah nodes is between 0.06 and 0.11 GF/W (gigflop per second per watt). For the same runs, this efficiency is higher by 42-55% at Clemson and 62-70% on Apt. In future work, we will perform a comprehensive analysis of the energy efficiency of the CloudLab resources using a number of diverse benchmarks. From the infrastructure standpoint, we will reuse the existing components and develop additional cookbooks only for the new benchmarks and their dependencies.

VII. RELATED WORK

In [17], the authors introduce cloudinit.d, a utility for contextualization of VMs, i.e. assignment of application roles to individual nodes in virtual clusters. AWS CloudFormation templates [18] allow launching virtual clusters on the AWS cloud in pre-configured states. In both cases, the specified configurations are realized when nodes boot. Similarly, CloudLab allows users to create and use custom images. In contrast, we developed a profile where

Chef naturally supports recontextualization, i.e. incremental development and modifications in node configurations, throughout the experiment lifecycle.

In [19], the authors propose custom configuration and reconfiguration capabilities. Kameleon in [20] is a software appliance builder which promotes reconstructability using a custom configuration specification syntax. In contrast, we demonstrate how users of CloudLab can create instances of Chef, a modern CMS with an active and diverse global user community. They can use Chef to develop custom infrastructure code and also take full advantage of the publicly available code developed by the community.

Among the GENI-focused projects, GENI Experiment Engine [21] and LabWiki [22] are the most relevant experiment management solutions. GENI Experiment Engine provides support for configuring experiments with Ansible, a free, open-source automation tool. While Ansible is considered intuitive and easy to operate, we anticipate that hierarchies of roles, cookbooks, and recipes in Chef are more conducive to supporting customization and composability of configurations in multi-node software environments. Additionally, Chef recipes equipped with fully featured Ruby code will likely provide more flexibility in complex experiments than Ansible playbooks implemented in YAML.

VIII. Summary and Future Work

In this paper, we demonstrate how the Chef configuration management system can help orchestrate components of software environments in CloudLab experiments. Chef helps us address customization and composability issues encountered when developing multi-component and multi-node software stacks capable of running on multiple hardware platforms.

We will continue examining possible organizations of Chef components that are appropriate for CloudLab experiments. For instance, we will investigate if it is advantageous to install and configure Chef workstations on user machines to manage CloudLab resources. We will also explore another model where a single CloudLab-wide Chef server is used, nodes in every experiment become its clients, and different users and experiments are isolated from each other using unique server credentials. We will aim to enable configuration management not only on new experiments but also transparently support existing experiments. Rather than using a custom profile, we consider delegating configuration of Chef components to the CloudLab back-end services and adding to the portal an optional per-node checkbox for enabling Chef. This will spare the users from managing custom tarballs and startup scripts in their profiles. We plan to expand the established repository with infrastructure code beyond the performance- and energy-focused tools and develop Chef cookbooks for a variety of applications used on CloudLab and related testbeds.

Ansible, CFEngine and Puppet are commonly used Chef alternatives, each with unique advantages and disadvantages. In future work, we will investigate which particular advantages each of these systems can bring to the users on CloudLab and in testbed environments in general.

Acknowledgments

We would like to thank the CloudLab and Emulab staff at the University of Utah for feedback on the design of this system, as well as participants in the GENI project for inspiration by developing a variety of other experiment tools. This material is based upon work supported by the National Science Foundation under Grant Nos. 1338155 and 1419199.

References

[1] R. Ricci and E. Eric, "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications," *;login:, Usenix*, no. 2014.

[2] R. Ricci *et al.*, "Apt: A Platform for Repeatable Research in Computer Science," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, Jan. 2015.

[3] "CloudLab," 2016. [Online]. Available: http://cloudlab.us/. Accessed: Jan. 20, 2016.

[4] "RSpec Documents in GENI". [Online]. Available: http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs/. Accessed: Jan 26, 2016.

[5] M. Berman et al., "GENI: A federated testbed for innovative network experiments", *Computer Networks*, vol. 61, Pages 5-23, Mar. 2014.

[6] M. Hibler et al. "Fast, Scalable Disk Imaging with Frisbee." *USENIX Annual Technical Conference*, General Track. 2003.

[7] "All about Chef,". [Online]. Available: https://docs.chef.io/. Accessed: Jan. 20, 2016.

[8] "PETSc," [Online]. Available: http://www.mcs.anl.gov/petsc/. Accessed: Jan 26, 2016.

[9] "HPGMG,". [Online]. Available: https://hpgmg.org/. Accessed: Jan. 20, 2016.

[10] "SLURM,". [Online]. Available: http://slurm.schedmd.com/. Accessed: Jan. 26, 2016.

[11] D. Duplyakin, M. Haney, and H. Tufo, "Architecting a Persistent and Reliable Configuration Management System," *Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud '15). ACM*, pp. 11–16.

[12] "Chef-Cluster - CloudLab Profile," 2015. [Online]. Available: https://www.cloudlab.us/p/abdb27f1-9392-11e5-88c8-277b2fdb9c32. Accessed: Jan. 20, 2016.

[13] "Chef Supermarket,". [Online]. Available: https://supermarket.chef.io/. Accessed: Jan. 20, 2016.

[14] "geni-lib's documentation,". [Online]. Available: http://geni-lib.readthedocs.org/. Accessed:Jan. 20, 2016.

[15] "Emulab chef-repo," GitHub. [Online]. Available: https://github.com/emulab/chef-repo. Accessed: Jan. 20, 2016.

[16] "Emulab,". [Online]. Available: https://www.emulab.net/. Accessed: Jan. 20, 2016.

[17] J. Bresnahan *et al*., "Managing appliance launches in infrastructure clouds," In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery* (TG '11). ACM, New York, NY, USA.

[18] "AWS CloudFormation," 2010. [Online]. Available: https://aws.amazon.com/cloudformation/. Accessed: Jan. 20, 2016.

[19] D. Armstrong *et al*., "Contextualization: Dynamic configuration of virtual machines," *Journal of Cloud Computing*, vol. 4, no. 1, Jul. 2015.

[20] C. Ruiz et al., "Reconstructable software appliances with Kameleon," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 80–89, Jan. 2015.

[21] A. Bavier et al., "The GENI experiment engine," in *Teletraffic Congress (ITC), 2014 26th International*, pp. 1-6,9-11 Sept. 2014.

[22] "LabWiki," [Online]. Available: http://labwiki.mytestbed.net/. Accessed: Jan 26, 2016.