Development and Operations on the Defense Advanced Research Project Agency's Spectrum Collaboration Challenge

Anthony T. Plummer Jr. and Kevin P. Taylor

ABSTRACT

The Johns Hopkins University Applied Physics Laboratory (APL) developed a complex test bed of software and hardware called the Colosseum to support the Defense Advanced Research Projects Agency (DARPA) Spectrum Collaboration Challenge (SC2). Following a development and operations (DevOps) approach was critical to the team's ability to design and build the Colosseum. Such an approach enhances collaboration between operations and development teams and takes advantage of technology, particularly automation tools. Tasks for the DevOps team included developing software codebases, deploying system configurations, and monitoring hardware system status such as power levels, system temperature, fans, and system uptime. The team accomplished these tasks by following a DevOps approach and using a variety of tool sets. This article describes the processes and tools the team used to design, build, and maintain the Colosseum.

INTRODUCTION

Development and operations, or DevOps, represents a change in software development and information technology (IT) culture, focusing on rapid IT service innovation through the adoption of agile,[1] lean practices in the context of a systems-oriented approach. DevOps emphasizes people (and culture) and seeks to improve collaboration between operations and development teams. DevOps implementations use technology, especially automation tools, that can leverage an increasingly programmable and dynamic infrastructure.[1] DevOps merges two disciplines: software development and system administration.

Traditionally, software development teams and system administration teams work independently of each other. Software developers design, code, and test new software, websites, and databases. They focus on the design and architecture of the system, capabilities, and features that will be delivered to the customer and the appropriate languages and tools to realize the solution. IT professionals, or system administrators, are responsible for the software installation, daily management, upkeep, and configuration of computer systems of an organization. Systems include desktop and laptop computers, servers, networks, IT security systems, and other critical IT infrastructure. System admins are also responsible for determining appropriate IT policies for businesses, supervising lower-level technician staff, and sometimes overseeing the purchasing of IT equipment.

In DevOps, software development and IT administration teams work closely to deliver a product, service, or application to a sponsor or customer. Many teams follow the agile methodology,[2] where tasks are determined and executed as a team in 2- to 4-week sprints. Using this approach, tasks are considered based on sponsor and team priorities, feature development goals, maintenance efforts, team configuration, and other factors. The software development team's efforts are evaluated alongside the system administrators' maintenance needs. As defects or bugs are discovered during operations, the system administrators add them to a unified task tracking system for planning during the next sprint.

As part of the Defense Advanced Research Projects Agency (DARPA) Spectrum Collaboration Challenge (SC2), APL designed, developed, and built a wireless research test bed known as the Colosseum. (See the article by Coleman et al. in this issue for an overview of the Colosseum.) The Colosseum's collection of resources facilitated research in autonomous spectrum management across a set of collaborative intelligent radio networks (CIRNs) during SC2. The resources included software-defined radios (SDRs), a wireless channel emulator, emulated backhaul networks, data streams representing realistic user applications, and an emulated GPS service. The Colosseum provided services for research (e.g., secure data storage) and competition (e.g., scorekeeping). It was remotely accessible and was used by more than 100 researchers across 30 teams spanning 5 different countries over the 3 years of the competition (2016–2019).

Maintenance of Colosseum operations required significant software tool sets and management systems. The APL team followed a DevOps approach when designing, developing, and maintaining the Colosseum. This article discusses the DevOps processes and tool sets and provides an overview of some of the challenges the team faced. After providing an overview of the system, the article reviews the tools that were used to build and maintain the system. The complementary SC2 project management process is discussed in detail in the article by Freeman et al. in this issue.

WHY WAS DEVOPS NEEDED FOR SC2?

DARPA's SC2 was an ambitious undertaking to address the question of collaborative spectrum sharing. Achieving the goals of the program required a large and intricate test bed. Such a test bed did not exist at the time SC2 was launched, so it had to be designed and built from the ground up. Given the scale of the Colosseum, a large team of software developers, system administrators, team managers, and facilities personnel had to collaborate on designing a one-of-a-kind system to meet DARPA's goals. One major constraint on the project was a short timeline. In a traditional development approach, the software development team would first design and develop the codebase and then work with the facilities team to build the system and deploy the software to the servers. Then the system administrators would collaborate with the software and facilities team to develop a system maintenance and monitoring plan. Finally, once the system was built, users would be given access to the system to execute their tests and to participate in the competition events.

The compressed SC2 schedule required that almost all these activities be executed in parallel. To manage these concurrent activities, the team adopted a DevOps approach. DevOps provided a way to systematically enable the simultaneous development and operations efforts to come together to meet the program goals. To meet competition deadlines, competitors needed access to the system while major parts of the codebase were under development, system administration tools were being implemented, and some equipment was being installed. Additionally, the selected system management tools had to be flexible to administer a dynamic system environment. Tools that enabled effective monitoring of the health and status of the system were also critical to the success of the project.

COLOSSEUM SCALE

The Colosseum consisted of hundreds of servers, networking equipment, SDRs, software packages, and facility installations. Figure 1 shows a top-down view of the facilities that held all the Colosseum hardware. A single room with 21 racks of equipment, each with different types of hardware, was divided into four similar quadrants. As shown in the figure, each quadrant included one rack (green) that contained the network distribution infrastructure; two racks (blue) that contained 12 standard radio node (SRN) servers each; one rack (yellow) that contained 8 SRNs and 32 Universal Software Radio Peripherals (USRPs); and one rack (red) that contained 32 USRPs and the radio frequency (RF) emulation field-programmable gate array (FPGA) hardware. A single rack, rack 6 (purple), in the middle of the room contained the demilitarized zone (DMZ) external connections including internet access, GPS-based timing, web servers, and firewalls. Other systems existed within the four network distribution racks, including blade server chassis, storage systems, build servers, RF management servers, and external partner equipment.

The following statistics on the Colosseum system illustrate its size and complexity. In addition to the hardware components, hundreds of software applications executed tasks on the system daily.

- Hardware
  - 900 TB of network-attached storage (NAS)
  - 171 high-performance servers

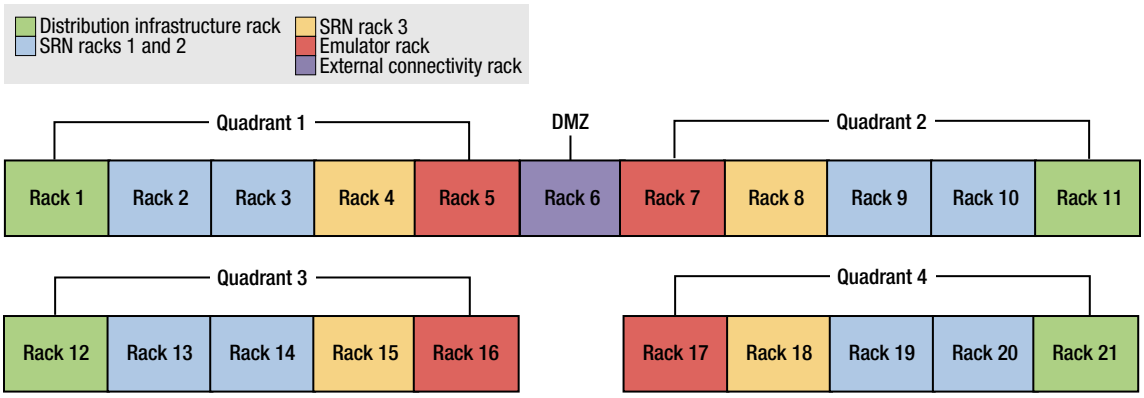


Figure 1. Colosseum facilities overview. The system was composed of 21 server racks of equipment with different types of hardware and was divided into four quadrants that contained 32 SRNs each. Until October 2019, it was housed on APL's main campus in Laurel, Maryland.

- 24 virtual machines
- 6 ESXi servers
- 256 USRPs
- 16 10-G switches
- 2 40-G switches
- 4 National Instruments BEEcube systems
- 17 FPGA modules
- 19 clock distribution systems
- 100s of high-speed optical connections
- 100s of networking and power connections

- Facility
  - 21 racks in a 30-foot by 20-foot equipment room
  - 40-ton heating, ventilation, and air conditioning (HVAC)
  - 65-kW, 208/120-VAC three-phase for equipment
- Users
  - 30 teams
  - 378 user accounts
  - 100s of system reservations per week

COLOSSEUM SYSTEM ADMINISTRATION ARCHITECTURE OVERVIEW

The system administration and design was decomposed into three major areas, as shown in Figure 2: configuration management, deployment, and health and status monitoring. In the configuration management area, software repositories hosted the source code that the applications deployed on the Colosseum. Additionally, static and dynamic configurations of the systems were maintained. Developers and system administrators uploaded all code and configurations to these repositories before deploying them to the Colosseum hardware. System administrators and the development team used the deployment system to deploy new and updated software, configurations, and tests to the Colosseum on demand. The system offered a consistent method of updating systems to reduce errors and increase reliability. Last, the health and status monitoring system actively evaluated the well-being of the Colosseum through monitoring hardware, services, and applications. System administrators could observe the system status at any time through web-based viewers. In addition, the system sent alerts to the system administrators when it detected issues.

CONFIGURATION MANAGEMENT

Repository Systems

Repository systems are centralized locations that store and manage development code, configurations, software packages, and user data. The following sections discuss the different repository systems the Colosseum used.

GITLAB

GitLab[3] is a Git repository manager that the SC2 team used to store source code and configuration information. At the time of this writing, there were 60 repositories

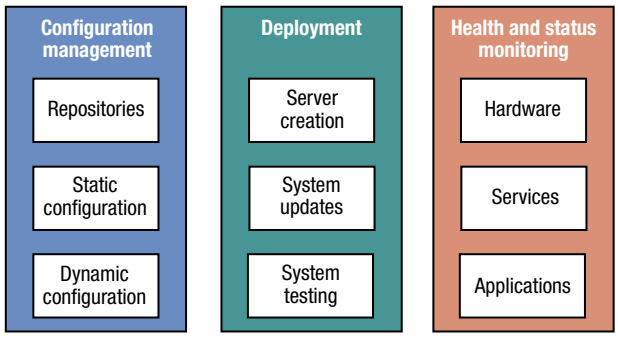


Figure 2. Colosseum system administration decomposition. System administration was broken down into three main areas: configuration management, deployment, and health and status monitoring.

on the server spanning various software components and configuration stores. There were dozens of users on the system who uploaded and downloaded code to the server.

Lightweight Directory Access Protocol and Authentication

The "389 Directory Server," or Lightweight Directory Access Protocol (LDAP),[4] stored all user information for the Colosseum. The LDAP database stored each user's ID, email, password, and Secure Shell (SSH) key for use across the entire system. The SSH keys were stored in the LDAP database via uploading to the competitor website to provide authentication access to the competitor SSH gateway. Once users authenticated with their SSH keys, they used their regular passwords to connect to other systems throughout the Colosseum. All the systems inside the Colosseum used the same LDAP server for authenticating users as well as for obtaining user ID information. The storage servers used LDAP for identifying users and groups for maintaining access control for files, which helped protect competitor data from unauthorized access. The LDAP application ran on a virtual machine on one of the ESXi[5] servers in the Colosseum.

Ubuntu Repository

The Colosseum maintained an offline Ubuntu repository[6] for use by internal servers. The repository was a copy of the entire Ubuntu online repository (~155 GB) including additional specialized packages.

Python Pip Repository

Most of the software developed for the Colosseum was written in Python. For Python dependencies, the Colosseum maintained an offline Python Pip repository.[7] Given the relatively small number of Python dependencies, the system maintained only the required dependencies.

Static Configuration

All servers within Colosseum maintained a base or static configuration that generally did not change during normal operations. Static configurations included the operating system, third-party software packages, and network configurations. This configuration category was maintained through a software tool called Puppet.[8] Puppet is a configuration management utility that keeps all the systems it manages consistent. Each host in the Colosseum ran a Puppet agent that queried the Puppet master server to get its configuration (known as a manifest). Figure 3 shows the Puppet deployment architecture. The Puppet configurations were stored in GitLab and then pushed by the Jenkins[10] deployment system to a server, called sc2-build, that hosted the Puppet master. The Puppet master communicated with Puppet agents

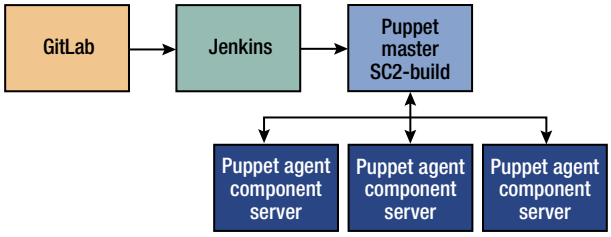


Figure 3. Puppet architecture. Puppet is a configuration management utility that keeps all the systems it manages consistent. Each host in the Colosseum ran a Puppet agent that queried the Puppet master server to get its configuration. The Puppet agents executed the latest configuration received from the Puppet master. At a fixed interval, the agents checked in with the master to determine whether there were any new updates.

running on all systems in the Colosseum. The Puppet agents executed the latest configuration received from the Puppet master. At a fixed interval (normally set to 10 minutes), the agents checked in with the master to determine whether there were any new updates.

The advantage to this system is that if a user or an administrator made a change on a single system, the next time Puppet ran, it replaced the configuration with the one on the remote Puppet server, thus guaranteeing a known configuration. This is also its disadvantage. Puppet could possibly overwrite a file being used for testing or temporary changes, so developers working in a test environment had to be careful. A precautionary measure usually included temporarily disabling the Puppet agent on the server that was being tested.

The Colosseum had numerous Puppet manifests that configured each aspect of the system. Standard manifests ensured that base software packages were installed, and configurations for connecting to the LDAP server, storage mount points, host files, etc. were defined. Additionally, each special environment of the test bed (wireless channel emulator, traffic controller, traffic generator, SRN) had separate Puppet manifests with configurations specific to it.

Dynamic Configuration

In the Colosseum's day-to-day operations, a few short-term, or dynamic, configurations typically remained active for a few hours or weeks. These included assigning SRNs to specific quadrants or environments (production or pre-production), changing data storage paths for test events, or changing traffic generator server locations. The primary tool used for this purpose was Consul. Consul[9] is a tool for discovering and configuring services in an infrastructure. The SC2 team primarily used it to configure the system layout (SRN quadrant assignments), to make traffic generation server assignments, and to make HTTP REST (representational state transfer) application programming interface end point

assignments. It is similar in architecture to Puppet, but it specializes in supporting dynamic configurations that may change often, whereas Puppet is more suited for static system configurations.

Settings for the SC2 Consul system were maintained within a repository on the SC2 GitLab server. These settings were never modified on a server directly by administrators or users. Automation processes were created within Jenkins to modify Consul settings to ensure that all changes to the Colosseum configuration were logged and executed in a controlled, repeatable manner. This not only greatly reduced the chance of misconfiguration but also provided a detailed history of the exact configuration of the Colosseum at any point in time. The addition of this process to the Colosseum DevOps procedures enabled the team to quickly and reliably adapt Colosseum configuration as needed.

DEPLOYMENT SYSTEM

A reliable and efficient process to update the Colosseum software and services was required to enable periodic feature updates, bug fixes, and maintenance tasks. The deployment system's purpose was to install, deploy, and manage software that supported SC2 operations. The system consisted of a collection of software tool sets, physical and virtual servers, networking equipment, and other special-purpose hardware. Each tool set had a specific purpose but could be categorized into three broad categories: repository, deployment, and agent. The repository system stored and managed the software source code, users' information, and system configuration information. The deployment system delivered new software code and configuration to the servers in the Colosseum. It managed the servers that each software codebase was deployed to and the methods to access each server. Last, the agents were the pieces of software executing on the Colosseum servers to enable desired capabilities and management actions. Many of the agents were constantly or periodically running and executing tasks autonomously. In contrast, the software in the deployment category was primarily used on an on-demand basis when a user had to execute a task. Most of the tool sets executed actions on the management network. As shown in Figure 2 in the article by Coleman et al. in this issue, the management network was connected to nearly all Colosseum systems.

The deployment system could be used for different use cases including:

- **Building a new system**—A server initially has no operating system installed. The deployment system installed the operating system, set the Internet Protocol (IP) addresses and media access control (MAC) addresses, added all the required software dependencies and source code, and started all the services.
- **Deploying new software code updates**—Each time there was a new update to a software component, the deployment system deployed, installed, and started the new software.

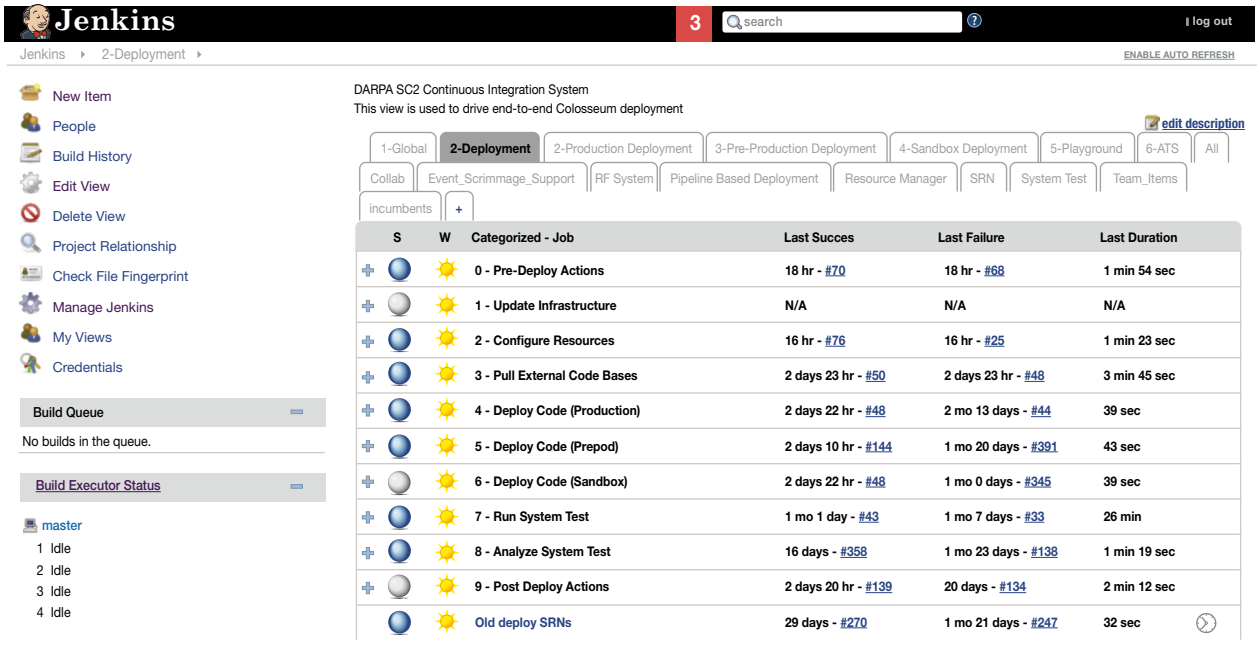


Figure 4. Jenkins development tab. Jenkins was the main software tool for deploying software and configurations in the Colosseum. Jenkins tabs organized multiple projects, and each project contained deployment-related code that accessed a system in the Colosseum or ran a set of commands on a system.

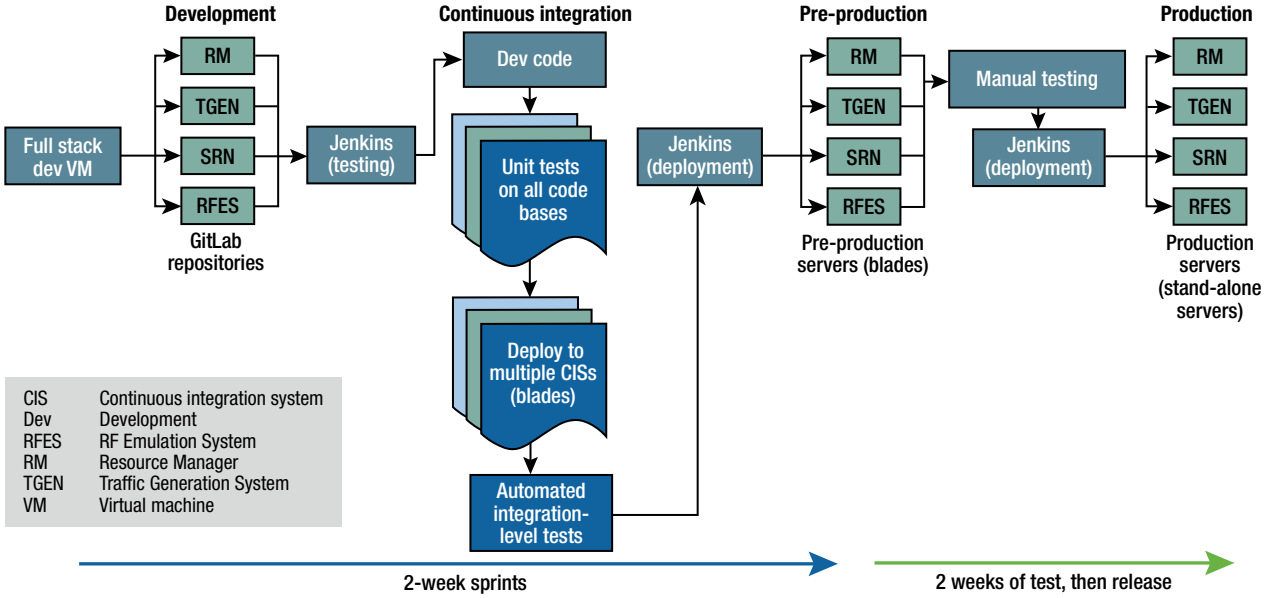


Figure 5. Development process and deployment. The figure shows the different environments in the Colosseum (development, continuous integration, pre-production, and production). At each step, Jenkins deployed and executed the required software.

- **Pushing updated system configurations**—System configurations were modified often as needs changed, and the deployment system deployed updated configurations to target servers.
- **System testing**—Before a new software package was deployed to the production system, it had to pass unit and system tests. The deployment system facilitated the automated operations of these tests.

System Updates and Testing

Jenkins was the primary software tool for deploying software and configurations in the Colosseum. All software was deployed from Jenkins. It provided a single point for software developers and administrators to deploy new updates to the Colosseum. Jenkins was used in the Colosseum for many purposes, including deploying new software codebases; running continuous integration unit tests; updating the configuration of Puppet and data collection systems; version-tagging codebases; restarting applications; disabling external web interfaces; updating remote repositories; and reconfiguring Colosseum resources across domains. Figure 4 shows the Jenkins Deployment tab. This tab and other Jenkins tabs organized multiple Jenkins projects. Each project contained deployment-related code that accessed a system in the Colosseum or ran a set of commands on a system.

Figure 5 shows the development process and deployment for the different environments in the Colosseum: development, continuous integration, pre-production, and production. The continuous integration and pre-production environments contained a full set of servers and applications that replicated the production environment. During the development phase, software developers designed and implemented new features using full-stack development virtual machines and then uploaded the software code to the GitLab repository. As new and updated features were completed, they were deployed from GitLab to the Colosseum in the continuous integration environment for unit testing. Features that passed the unit tests were deployed to the pre-production environment for additional system-level testing. This process occurred during the 2-week development sprints. Last, the tested features were deployed to the production environment during maintenance windows and were then available to Colosseum users. During each step in the process, Jenkins was used to deploy and execute the required software deployments.

Server Creation

The build system was one of the primary use cases for the deployment system. The build system typically aims to construct a component server from bare metal (e.g., a server with no operating system). Example Colosseum servers included those for the Resource Manager and the Traffic Generation System. The build process generally followed the steps outlined below.

1. A system that was being built or rebuilt sent a request for a network address to be configured. The Dynamic Host Configuration Protocol (DHCP) server used the MAC address of the network interface making the request to assign it its IP address.
2. Next, DHCP directed the system to the Trivial File Transfer Protocol (TFTP) server, which held the Preboot Execution Environment (PXE) image used for installing an operating system.

3. After the PXE environment was loaded, the Ubuntu deployment system automatically partitioned the hard disk as appropriate and installed a base software image including Puppet.

4. Once the system was fully installed and restarted, the Puppet agent on the newly built system checked in with the Puppet server and downloaded any specific configurations to bring the system into a usable state.

5. The system was ready to install the component software that generally differentiated the system from other servers. This software was deployed by Jenkins. Jenkins copied or "checked out" the source code of a specific component from GitLab and loaded it onto the new server build.

6. As part of the Jenkins deployment, the component application service was started and was then ready for use.

Ubuntu Deployment System

Automated deployment of new systems (or rebuilding of existing systems) in the Colosseum required configuration of several pieces of software: DHCP[11] (for automatic assigning of network addresses); PXE/TFTP[12,13] (allowing systems to automatically boot and install base configuration software); Puppet (to give the systems their configurations); and Jenkins[10] (to install component software).

The DHCP[11] server enabled systems to request their network configurations without having to manually set the address on individual hosts. For the Colosseum, DHCP was configured to assign addresses specific to the MAC address of the network interface requesting an address. This prevented any unknown system from automatically assigning itself an address and having to identify the unknown system to resolve a potential conflict. When a system was being built, DHCP directed the system to the TFTP server to get its initial image.

The TFTP[13] server transferred a small Linux system image via PXE[12] boot, which minimally booted the system and started the launch of the Ubuntu deployment system. This setup had the ability to launch different installation parameters based on the particular system being built. There were generally two configurations: SRNs (which contained multiple hard drives) and everything else (based on a single hard drive). Identifying which system got which configuration made a completely unattended installation possible.

After the PXE image was loaded, control was passed on to a minimal Ubuntu kernel, which was used to perform a software install. For Ubuntu, a preseed file[14] was used to answer standard questions about which software to install, how to configure the network, how to partition disks, etc. This preseed could also be configured to execute any number of commands after the software finished installing. Executing commands at the end makes it possible to install extra software packages and configuration files that cannot easily be defined in the main preseed configuration.

The Colosseum team initially experienced issues with the automatic disk partitioning mechanism built into the Ubuntu preseed configuration. As part of the deployment server, at the end of the preseed configuration, a shell script was launched to repartition the disks for SRNs (the main disk as well as the secondary drive), configure custom software, and install Puppet so that when the system rebooted it was ready to receive its configuration.

Common Colosseum Build System

Figure 6 shows the Colosseum build system. The build system provided a structure for the server creation process and was divided into six layers, each supported by software tool sets, as shown in Figure 6 and described below:

- **Layer 1, Operating system**—This was the lowest layer and contained the operating system (Base Ubuntu 14.04 Linux) as well as initial networking and IP and MAC address configurations.

- **Layer 2, Base system configuration**—This layer set the configurations of available Colosseum resources, including the Ubuntu and Python repositories, LDAP, and hardware management tools.

- **Layer 3, Component Puppet module**—This layer established the Puppet configuration, which installed the component dependencies and configuration files.

- **Layer 4, Component software**—This layer installed the component software and initial database configurations on the target server. This layer is what truly differentiated server functionality (e.g., Resource Manager from Traffic Controller).

- **Layer 5, Consul configuration**—This layer set the Consul service configuration.

- **Layer 6, Process monitoring**—This layer configured the monitoring systems for the component.

HEALTH AND STATUS MONITORING

The Colosseum's hundreds of active users depended on the availability of its systems for development and test activities as well as for competitions, which required a high level of integrity to ensure fair results. Actively monitoring the system's health and status to identify issues and ensure a good operational state was an

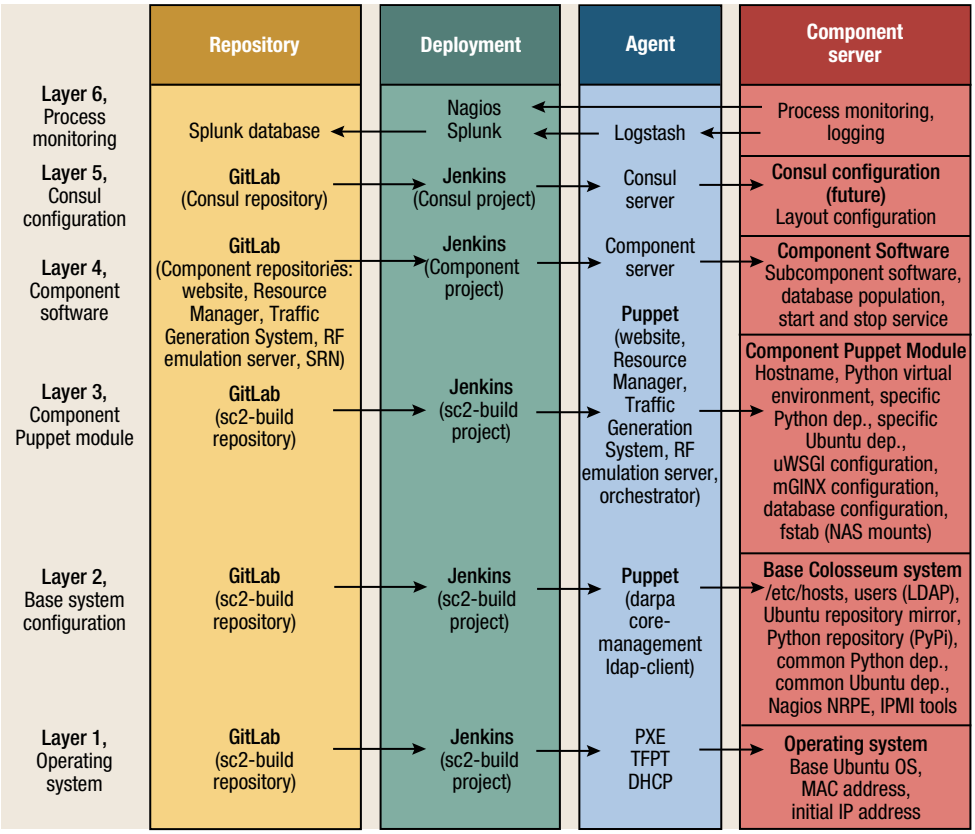


Figure 6. Colosseum build system. Shown are the six layers of the build and the software tool sets supporting each layer.

important activity for the APL team. The Health and Status Monitoring System was a collection of autonomously running and on-demand software that actively collected information, statistics, and data from most of the system's components. A key component of the monitoring system was the process of collecting data from Colosseum servers and applications. Analytics were run against these data to determine the system's health and status. Figure 7 shows the Colosseum data collection, which included three sources of data:

1. **Hardware level**—This was the server- level or hardware-level information, such as information on power levels, system temperature, fans, and system uptime.

2. **Service level**—This information concerned whether a software service was active or not. For example, was the Resource Manager, orchestrator, or SRN application running or not?

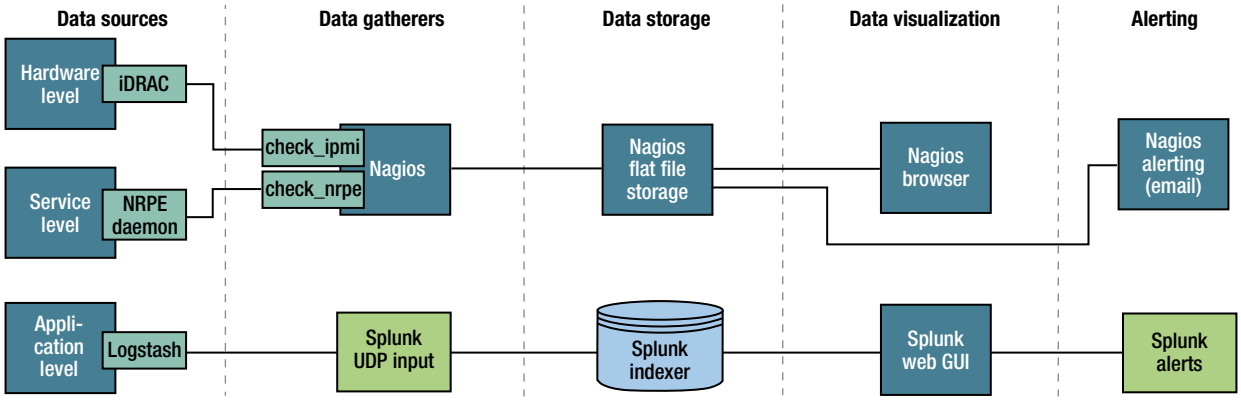


Figure 7. Colosseum data collection. The system collected hardware-, service-, and application-level data. These data sources supplied other tool sets, as shown in the figure, that processed the data for administrator consumption.

3. **Application level**—This was application- or component-level information that provided details about a specific application's performance or statistics (e.g., the number of SRNs being used by competitors or the current state of a reservation).

These data sources supplied other tool sets that processed the data for administrator consumption. Data gatherers collected information from other systems. Data storage tools stored and organized the collected data. Data visualization tools displayed graphical views of data. Most were web-based interfaces. And, finally, alerting tools sent emails to admins or maintained an event logging system that admins could view periodically.

The Colosseum collected several types of information. Just a few examples are the number of (active/teams/users) reservations; SRN allocation status; server load averages, disk and memory usage; and reservation status across system components.

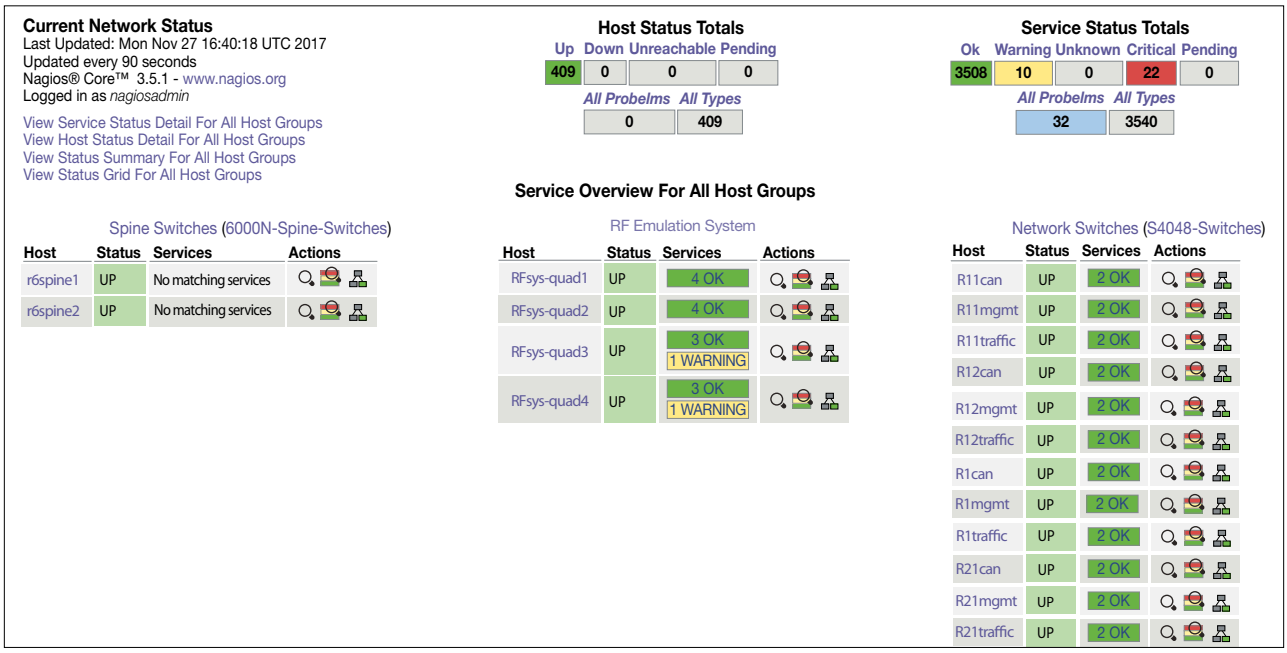
The following sections detail each of the health and status monitoring tools.

Nagios

The open-source package Nagios[15] was the main center of system monitoring and alerting for the Colosseum. Alerts could be configured to email specific support staff if an issue arose. Other checks provided informational status inside of Nagios; these checks did not email an alert but showed a warning or critical status on the web interface. Nagios checks, some shown in Figure 8, included the following:

System uptime/downtime—Nagios checked to see whether a host was alive, and if the host could not be reached on the first check, Nagios rechecked several more times. If the system was not responsive after the last check, Nagios sent an email alert noting that the system was down.

Hardware status—Nagios was configured to query the IPMI[16] interface on many of the servers inside the Colosseum test bed. If the system temperature got too high or a power supply stopped reporting, an alert was emailed. Fan and power consumption was also monitored for informational purposes.



Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
analysis-server-1	check_disk_sda1	CRITICAL	2017-11-27 16:37:32	4d 13h 20m 42s	4/4	DISK CRITICAL - free space: / 337 MB (0% inode=89%):
wce-quad3	check_disk_sda1	WARNING	2017-11-27 16:37:31	10d 2h 23m 43s	4/4	DISK WARNING - free space: / 14871 MB (19% inode=85%):
wce-quad4	check_disk_sda1	WARNING	2017-11-27 16:37:40	16d 21h 33m 35s	4/4	DISK WARNING - free space: / 13134 MB (17% inode=85%):
sc2-srn-003	Check software version	CRITICAL	2017-11-27 16:35:34	4d 9h 40m 40s	4/4	CRITICAL - Web version: - Disk version: "origin/release"
	Check for SRN status	WARNING	2017-11-27 16:36:34	4d 9h 39m 40s	4/4	ERROR - no web info returned
	Check_srn_service-srn_ctrl	CRITICAL	2017-11-27 16:35:33	4d 9h 40m 41s	4/4	ERROR - 2 srn_ctrl_start processes running

Figure 8. Nagios host groups and problems view. Nagios, an open-source package, was at the center of system monitoring and alerting for the Colosseum. Alerts could be configured to email specific support staff when an issue arose. Other checks, like those shown in the problems view (bottom), did not email an alert but showed a warning or critical status on the web interface.

Disk usage—Nagios sent an alert report when the system disk usage got above 80% utilization.

Service checks—The bulk of the Nagios checking on the Colosseum was based on checking various services and functionality in the test bed. Most of the checks simply returned an up or down status, but some reported a more detailed status message. Following is a partial list of the service checks performed on the Colosseum:

- LDAP service responding on the LDAP server
- Verification that the Network Time Protocol (NTP)[17] server and the NTP service on hosts were functioning
- System load on the systems
- RF emulation server services running
- Web service running on the hosts it was supposed to be running on
- Verification of whether USRP USB connection was good
- Verification that SRN services were running on the SRN hosts
- Verification that the User Datagram Protocol (UDP) service was started for the wireless channel emulator
- Verification of software versions for key pieces of the test bed environment

Service restarts—For a few of the most critical services in the test bed (such as the SRN services, SRN NTP[17] services, and channel update process services), Nagios was configured to execute a command if the service entered a critical state. These commands executed a restart of the service when instructed by Nagios. This helped guarantee system uptime if one of these services crashed or failed.

Integrated Dell Remote Access Controller

Dell[18] provided out-of-band management of its equipment using the Integrated Dell Remote Access Controller (iDRAC) available on each server. Through the iDRAC, the SC2 team was able to monitor hardware components (CPU temperature, fans, power supply status) and remotely control systems (power them on/off, remote console, change boot options). Because this was out of band, the servers themselves did not need to be online or even powered on for this functionality to be available.

Splunk and Logstash

Splunk[19] is an advanced log collection and analysis software package. It performed complex searches against all the data collected by test bed components and created reports and alerts based on events found in log files. Figure 9 shows an example Splunk query.

Logstash[20] is an open-source server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to a data collector. The main software components used a utility that captured all Python logging messages and sent them over UDP to the sc2-log server that ran the Splunk application. The combination of Splunk and Logstash created a powerful capability for analysis of system events and logs. Figure 10 shows the flow of Logstash

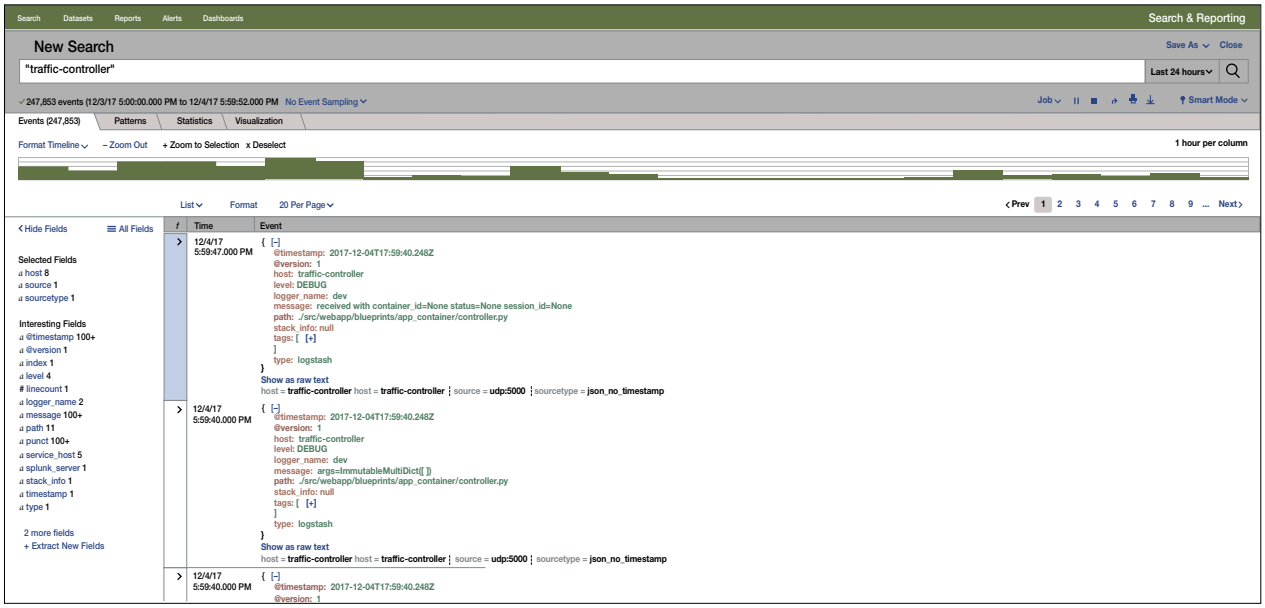


Figure 9. Splunk search interface showing an example query. Splunk is an advanced log collection and analysis software package that performed complex searches against all the data collected by test bed components and created reports and alerts based on events found in log files.

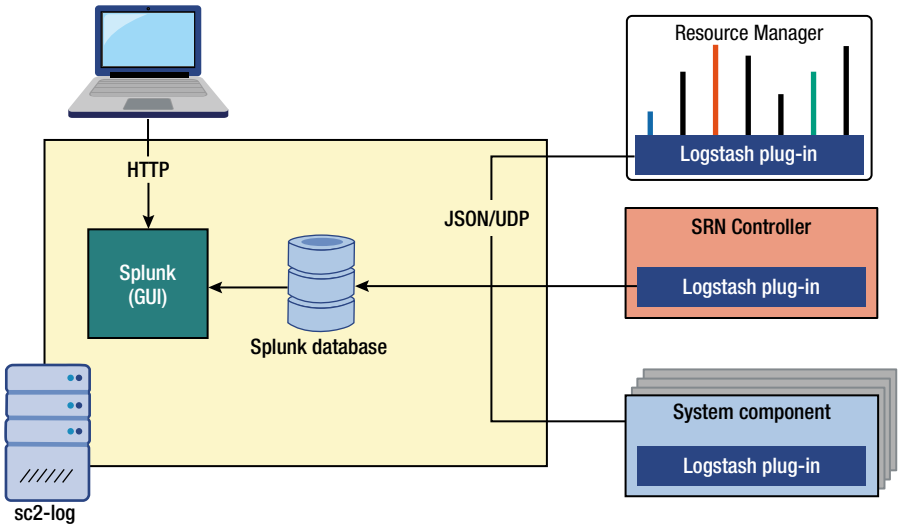


Figure 10. Logstash message flow to Splunk. Logstash is an open-source server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to a data collector like Splunk. Together these tools created a powerful aid for analysis of system events and logs.

messages to the Splunk tool set. Users could log into the Splunk interface to access the stored data.

StatusCake

StatusCake[21] is an online service that checked the Colosseum's external interfaces (i.e., internet-facing systems). It used 100+ monitoring servers across the world to periodically check whether a web link was reachable. StatusCake monitored the main competitor website and the competitor gateway.

CONCLUSION

DevOps provided a guiding process and set of tools that helped the APL team build, design, and maintain the Colosseum. DevOps principles and systems enabled many of the Colosseum's required operational actions, including deploying new software codebases; running continuous integration unit tests; updating the system configuration; restarting applications; monitoring hardware system status, such as power levels, system temperature, fans, and system uptime; and monitoring application-level performance and statistics. The SC2 team accomplished these tasks by using a variety of tool sets that all served different purposes but in many cases worked together. The team considered many trade-offs during the implementation of the system and ultimately selected the DevOps tools that best helped it to design, build, and manage the Colosseum.

ACKNOWLEDGMENTS: We thank Paul Tilghman (DARPA SC2 program manager) and Craig Pomeroy and Kevin Barone (Systems Engineering and Technical Assistance at DARPA) for their invaluable collaboration and support. We also thank the many APL SC2 contributors, whose names are listed on the inside back cover of this issue of the *Digest*, and in particular we acknowledge Kenneth R. McKeever, Uthman Adediran, Kurt T. Yoder, Emery Annis, Robert W. Grimes, Cherita Corbett, and Jordan Kraus. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the US government.

REFERENCES

[1]"DevOps." Gartner Glossary. https://www.gartner.com/it-glossary/devops/ (accessed Aug. 26, 2019).
[2]K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, et al., 2001. "Manifesto for agile software development." http://agilemanifesto.org/ (accessed Aug. 26, 2019).
[3]"About." GitLab. https://about.gitlab.com/ (accessed Aug. 26, 2019).
[4]"389 directory server." http://directory.fedoraproject.org/ (accessed Aug. 26, 2019).
[5]"ESXi." VMware. https://www.vmware.com/products/esxi-and-esx.html (accessed Aug. 26, 2019).
[6]"Ubuntu packages search." Ubuntu. https://packages.ubuntu.com/ (accessed Aug. 26, 2019).
[7]"pip 19.2.3." Python PIP. https://pypi.python.org/pypi/pip (accessed Aug. 26, 2019).
[8]"Puppet." https://puppet.com/ (accessed Aug. 26, 2019).
[9]"Consul: easy service networking." https://www.consul.io/ (accessed Aug. 26, 2019).
[10]"Jenkins." https://jenkins.io/ (accessed Aug. 26, 2019).
[11]"ISC DHCP." Internet Systems Consortium. https://www.isc.org/downloads/dhcp/ (accessed Aug. 26, 2019).
[12]"PXEInstallServer." Ubuntu. https://help.ubuntu.com/community/PXEInstallServer (accessed Aug. 26, 2019).
[13]"The TFTP Protocol (revision 2)." IETF. https://tools.ietf.org/html/rfc1350 (accessed Aug. 26, 2019).

[14]"B.2. Using preseeding. Appendix B. Automating the installation using preseeding." Ubuntu. https://help.ubuntu.com/lts/installation-guide/armhf/apbs02.html (accessed Aug. 26, 2019).
[15]"Nagios."https://www.nagios.org/ (accessed Aug. 26, 2019).
[16]"Intelligent Platform Management Interface (IPMI)." Intel. https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html (accessed Aug. 26, 2019).
[17]"NTP: Network Time Protocol." Network Time Foundation. http://www.ntp.org/ (accessed Aug. 26, 2019).
[18]"iDRAC." Dell. http://www.dell.com/learn/us/en/15/solutions/integrated-dell-remote-access-controller-idrac (accessed Aug. 26, 2019).
[19]"Splunk." https://www.splunk.com/ (accessed Aug. 26, 2019).
[20]"Logstash." Elastic. https://www.elastic.co/products/logstash (accessed Aug. 26, 2019).
[21]"StatusCake." https://www.statuscake.com/ (accessed Aug. 26, 2019).

Anthony T. Plummer Jr., Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Dr. Anthony T. Plummer Jr. is the supervisor of the Spectrum Analysis Section in the Tactical Communications Systems Group in APL's Asymmetric Operations Sector. He received a BS in electrical engineering from Morgan State University in 2005 and an MS and a PhD in electrical engineering from Michigan State University in 2007 and 2011, respectively. His interests include the design and implementation of software systems and researching approaches to applying machine learning to communication and networking applications. His email address is anthony.plummer@jhuapl.edu.

Kevin P. Taylor, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Kevin P. Taylor is a Linux systems administrator in APL's Asymmetric Operations Sector. He holds a bachelor of arts degree from Towson University. Kevin has over 25 years of experience managing UNIX and Linux systems. Kevin supported the facility and systems administration for the DARPA SC2 project, including installing and setting up the Ubuntu environment consisting of an automated deployment system (for automating installation and reinstallation of systems), monitoring of services and system environments (using Nagios), and configuration management (using Puppet). His email address is kevin.taylor@jhuapl.edu.