Journal of Physics: Conference Series **396** (2012) 042056 doi:10.1088/1742-6596/396/4/042056

Centralized Fabric Management Using Puppet, Git, and GLPI

Jason A. Smith, John S. De Stefano Jr., John Fetzko, Christopher Hollowell, Hironori Ito, Mizuki Karasawa, James Pryor, Tejas Rao, William Strecker-Kellogg

Brookhaven National Laboratory, PO Box 5000, Upton, NY 11973, USA

E-mail: smithj4@bnl.gov, jd@bnl.gov, jfetzko@bnl.gov, hollowec@bnl.gov, hito@bnl.gov, mizuki@bnl.gov, pryor@bnl.gov, raot@bnl.gov, willsk@bnl.gov

Abstract. Managing the infrastructure of a large and complex data center can be extremely difficult without taking advantage of recent technological advances in administrative automation. Puppet is a seasoned open-source tool that is designed for enterprise class centralized configuration management. At the RHIC and ATLAS Computing Facility (RACF) at Brookhaven National Laboratory, we use Puppet along with Git, GLPI, and some custom scripts as part of our centralized configuration management system. In this paper, we discuss how we use these tools for centralized configuration management of our servers and services, change management requiring authorized approval of production changes, a complete version controlled history of all changes made, separation of production, testing and development systems using puppet environments, semi-automated server inventory using GLPI, and configuration change monitoring and reporting using the Puppet dashboard. We will also discuss scalability and performance results from using these tools on a 2,000+ node cluster and 400+ infrastructure servers with an administrative staff of approximately 25 full-time employees (FTEs).

1. Motivation

As recently as two years ago, most of the administrative work at the RACF was performed via a combination of tools developed in-house, and manual editing of files on live production servers, including a great deal of SSH loop commands and scripts deployed from a secure administrator gateway. This type of time-consuming work kept our staff operating in a reactive, fire-fighting mode of preventive administration. Since we have a staff of about 25 FTEs, split into roughly four different groups, each group and sometimes each person had their own way of doing things. There was very little sharing of work and experience, which resulted in segregated, highly-specialized administrative domains and afforded very little backup expertise among staff during off-hours and vacation and sick time.

To improve this situation, about two years ago, a few of us decided to look for some tools to help make our daily work more efficient, and to centralize and standardize fabric management across the entire facility as much as possible. We evaluated many of the available tools, mostly free and open-source, and selected several which we could use to build into a new, centralized configuration management system. We were looking for a solution with an easy to understand configuration language that would provide a self-documenting system build and configuration system. Other

requirements included the ability to track all changes that were made for auditing purposes, and a modular design, which would allow us to replicate our production server configurations to test and development servers with little extra effort.

2. Components

The various requirements were broken down into basic functional components: provisioning new systems, configuration management of the systems (after initial install and throughout the entire life cycle of the server), asset management, source code management of the entire configuration catalog, and various web-based user interfaces needed for controlling and monitoring systems.

Cobbler and Red Hat Enterprise Virtualization (RHEV) were chosen for building new systems with a standard "golden image," or a carefully-curated and immutable operating system image. FusionInventory Agent and GLPI were chosen for semi-automated asset management and can also be used to help configure our servers. Puppet was chosen as our centralized configuration management system, and Git was selected as the source code revision control system for our Puppet catalog. Puppet also comes with web-based "dashboard" user interface for easy monitoring of all managed systems.

Together, these components provide an almost completely automated system build and configuration system. Under normal circumstances, the only manual steps required are to sign each system's Puppet client certificate request (which is not required but implemented for added security), assigning the configuration classes and parameters needed for each server, and restoring any user data as necessary. Aside from any initial development work required to create a new Puppet configuration, new servers can be provisioned and configured in a matter of minutes using this centralized system.

2.1. Provisioning

Two different systems are used for provisioning new servers: Cobbler for bare-metal hardware installations, and RHEV for virtual machines (VMs). Each system uses a standard minimal "golden image" for the initial OS install, and bootstraps the rest of the centralized management system by installing the `fusioninventory-agent` and `puppet` client packages. The initial "golden images" are, for the most, part identical, since the VM template used in RHEV is based on the same minimal installation used for hardware installations done by Cobbler.

2.1.1. Cobbler[1]. Cobbler is used for deploying hardware installations; see poster #539[2] for more details. We chose Cobbler because of its powerful Cheetah template language, and configuration and code reuse with "Snippets". These features enabled the creation of a single kickstart template that can be used for nearly all hardware installations, since it can be used on almost all of the facility's various hardware configurations. In the Cobbler web interface, we simply specify the desired operating system (OS) version and architecture, network information (MAC address, IP address, etc.), and possibly some template metadata in order to install the base OS, including the necessary client packages.

A single Cobbler master node serves three Cobbler slave servers that manage systems on some of our external firewalled networks, all of which are synchronized using Cobbler's master-slave automated replication. The Cobbler server also manages local Yum package repository mirrors of external, third-party repositories, including package collections for OSG and XRootD.

2.1.2. RHEV[3]. RedHat Enterprise Virtualization version 3.0 is used for provisioning and managing our VM images. A single template image is used for the installation of nearly all new systems. Our current RHEV installation consists of a 10 node cluster with 4 TB of shared fiber storage, which can

support several hundred VMs. We also have a similar RHEV cluster to support RHIC computing; both clusters can be easily expanded as needed to accommodate future growth.

2.2. GLPI[4] and FusionInventory[5]

These software tools serve both for asset management and as our primary node classification system for Puppet. The FusionInventory Agent automatically collects and updates common hardware and OS level configuration metadata for all of our servers. The GLPI software is a central database that collects the FusionInventory Agent data, and provides a web interface for displaying and searching of the asset inventory database. GLPI provides support for change management and auditing with a historical record of all changes made to each system. It's also used to specify the Puppet environment assigned to each node using its "Status" field and, using a Custom Fields plug-in, can define Puppet classes and parameters that get assigned to each node by our custom ENC (External Node Classifier) script. While GLPI is currently our primary ENC for most servers, our Linux Farm's worker node cluster uses a separate, custom MySQL DB for asset management, which they've maintained separately for several years. Our ENC script can query both the Puppet dashboard and the worker node database. However, the functionality of the Puppet dashboard is currently somewhat limited, as it does not support parameterized classes, parameter array values, or selection of non-production environments; some of these features may be added to a future Puppet version.

2.3. Git[6]

Git serves as our source code management system. Our Puppet master server uses the `git-http-backend` over authenticated HTTPS to serve our Git repositories, which, for added security, are also only available from within the local BNL network. We chose Git over other alternatives for several reasons, including: distributed workflow; the ability to do almost all work offline; a complete history contained in the local copy; reduced single point of repository failure; flexibility in merging changes between many branches or repositories; and simple, fast, and clean branching and merging, which facilitate a workflow in which different code change sets are isolated in different branches.

2.4. Puppet[7]

Previous attempts to use CFEngine version 2 proved it to be a cumbersome tool. Other automated management systems (chef, etch, bcfg2, AutomateIt), were not chosen for various reasons: little code reuse (copy and paste required instead of Puppet's class and modular architecture), non-uniform configuration language, and a procedural methodology (which is good only in the case of a known initial system state) instead of a declarative one with a dependency graph model for determining execution order and incomplete functionality (no central server, and reliance on other means, such as a shared filesystem, to distribute configuration catalogs). Puppet was selected for several reasons, including: its simple yet powerful DSL Domain-Specific Language (DSL) and Resource Abstraction Layer (RAL); explicitly declared dependencies, which provide better deterministic state convergence from unknown initial conditions; a central configuration catalog and dependency resolution (which provide better security, conflict resolution, and logic analysis); an included web dashboard and GraphViz configuration visualization; a long development history; a stable codebase and large user base; and the fact that it's a free, open-source project with optional commercial support.

3. Puppet server configuration and scalability

We are currently using the latest version (2.6.16) of Puppet from the 2.6 maintenance release on a RHEL-5 server with Ruby 1.8.5. The server runs on a Dell PowerEdge R410 with 32 GB of

memory and 4x15k RPM SAS disks. In addition to Puppet, the server is also running primary instances of Cobbler, GLPI, Git, Apache, and MySQL. For added scalability, Puppet is configured to run behind Apache with Phusion Passenger (via `mod_rails`). Puppet's `storeconfigs` feature enables support for exported resources in our Puppet modules, which are used for things like automated Nagios server configuration, SSH host-based authentication configuration, and Cobbler master/slave configuration. To optimize this feature, Puppet is configured to use the queue daemon with ActiveMQ for fast database updates. The Puppet inventory service is also enabled, but is viewable only through the Puppet dashboard. Over 2,000 agents are currently reporting via Puppet. The only scalability issue seen thus far is an occasional MySQL error updating the inventory service when clients connect to the server at a rate higher than one per second. To address this issue, we increased and staggered our update intervals to reduce the risk of concurrent connections. We are also testing an upgrade to Puppet to the latest version (2.7) on a RHEL-6 server with Ruby 1.8.7, on similar hardware. In the future, we may investigate running Puppet behind with Tomcat and JRuby, since early tests by others show promising signs of improved scalability, but currently this deployment scheme still has a few problems that make it unusable in a production environment.

3.1. Puppet Environments

The Puppet server uses the recommended and standard practice of a three-environment configuration, which can be easily expanded via Git as needed. Git hooks are configured to automatically update the Puppet environment directories to obviate the need for manual Puppet configuration edits to match environmental changes. This also prevents mistakes that might occur if administrators were permitted to manually modify files directly in Puppet on the live production server. To promote rapid development, it is also very easy to run `puppet apply` against the working copy of a modified Git cloned repository using an appropriately modified Puppet client configuration. This saves time and and effort, and allows administrators the opportunity to test changes without having to push all changes to the Puppet and Git server, and then run the agent manually only to discover that a simple and avoidable mistake was made and must be corrected through a similar push process through Git and Puppet.

Our three Puppet environments are directly linked to three synonymous Git branches: *development* is used for extensive module creation and editing; *testing* is used for small changes and to stage all changes to production, so they can be manually tested by a wider audience before being merged into production use; and *production* is used for live server management. Pushed changes to Puppet's Git repository are first verified by the Git update hook, which validates code syntax and file size limits, and also blocks the deletion of environment branches. Attempted updates to the production branch are also trapped and diverted into a temporary branch, which alerts a select group of administrators to a pending peer review and approval of the code change. After review and manual approval, the temporary branch is automatically merged and synchronized into production. The post-update Git hook syncs changes to Puppet and records the metadata of the merged change to the server system log.

3.2. Production Approval

As mentioned above, users and administrators are not permitted to push code changes directly into production: a Git server hook traps these changes into a new branch. After the requested changes to the production environment are diverted into a temporary branch, a custom Puppet approval CGI script isolates the proposed changes, and alerts code reviewers to review and either approve or reject the change set. Links on the web page open to the appropriate cgit [8] website to make it easy for administrators to view the commit logs and code differences in the pending changes, and enable approved staff members to accept and merge or reject and delete the pending changes. When

changes are approved, a commit log is automatically added to git showing who approved the change and any comment they wish to add. This combination of the review and approval CGI script and Git log audit trail will ease the implementation of a full change management system when we are ready in the future.

3.3. Puppet Dashboard

Puppet provides a dashboard web interface, which is useful for monitoring all Puppet clients in one central location. This dashboard includes a summary view on the top page and provides a “drill down” mechanism to view individual hosts and reports for specific Puppet agent runs. In addition to monitoring, the dashboard presents Puppet's inventory data and can be used as an External Node Classifier to configure hosts with assigned classes and parameters.

4. Future Plans

In the near future, as we create more Puppet modules and convert more production servers to our Puppet infrastructure, we will operate under a more formal change management system, using these technologies as a base. We will also investigate using our RHEV cluster to create a miniature replica of all production servers, and create an automated testing and validation system of our Puppet manifests.

4.1. Change Management

Basically this is all about creating policies and procedures that are used to control changes made to production systems. Some common terminology used to describe the methods and best practices used to implement such policies are the Information Technology Infrastructure Library (ITIL)[9] and Development and Operations (DevOps)[10]. Some of the main principals are: changes made only during officially designated time periods; absolutely no unauthorized changes; no "cowboy" type behavior tolerated (i.e., no administrators swooping in and out of the production environment to make unannounced and uncoordinated changes to live systems); a full testbed environment to vet changes before putting them into production; and document all changes that are made with comments and a full audit trail. The tools discussed in this paper, including Puppet, Git, and GLPI, can not only help make the changes in a central and organized way, but also help in keeping a complete historical change record.

4.2. Automated Validation

We are in the initial stages of designing an automated validation system using our test Puppet 2.7 server. The basic idea is to have a validation Git branch and Puppet environment automatically created and populated with the contents of production, with the addition of any and all changes pending production approval. We will also create a replica of all important production services using RHEV VMs. Changes to the validation environment can only be made by the automated approval system and will initiate Puppet runs on all of the VMs to validate that everything is still working as expected, and that the proposed changes don't create problems with unrelated modules used in different combinations on different hosts. We will also implement a complete Nagios monitoring system to further validate that the proposed changes do not cause any problems to related services in the complete production replica environment. A custom dashboard will also be created to summarize all VM status results (both Puppet agent runs and Nagios monitoring) to give administrators an automated confirmation that their proposed changes to production won't cause any unforeseen problems.

5. Benefits

Our prior mode of operation consisted of uncontrolled and undocumented change, unless someone remembered to take notes while working feverishly to put out the current administrative fire and fixing a problem. Using this centralized management infrastructure will allow us to use a common set of standard modules and make changes when necessary and in only one place, instead of the old method of manual changes by each administrator on the servers they were responsible for, often in different ways, and sometimes leaving these changes undocumented. Uncontrolled and undocumented change can work sometimes, but will often cause self-inflicted problems, future firefighting episodes, and upgrade nightmares. Our ultimate goal is to stop duplicating work and effort, standardize across our administrators and servers, and stop making manual, time-consuming changes. Without this, servers become like snowflakes: they may all start out identical, but over time, configuration drift will eventually make each one unique and more difficult to manage.

As more system administration and configuration automation processes are implemented, administrative staff can start devoting time to more interesting projects and planning for the future, such as improving our overall monitoring infrastructure, ensuring more reliable services, and deploying smooth upgrades. The end result will be a shift of staff time from a perpetually reactive mode ("firefighting") that only addresses problematic symptoms, to more proactive work that addresses the root causes of problems (fire prevention). Another benefit is a repeatable and standard build and configuration process, which often makes it faster and easier to rebuild problematic servers, rather than wasting hours or days troubleshooting and trying to fix them.

References

[1] Cobbler - Linux install and update server, http://cobbler.github.com
[2] Pryor, James. Automating Linux Deployment with Cobbler presented at CHEP 2012, NY, NY
[3] Red Hat Enterprise Virtualization, http://www.redhat.com/products/virtualization
[4] GLPI - Gestionnaire libre de parc informatique, http://www.glpi-project.org
[5] FusionInventory - The opensource IT inventory solution, http://fusioninventory.org/wordpress
[6] Git - distributed-is-the-new-centralized, http://git-scm.com
[7] Puppet Labs: IT Automation Software for System Administrators, http://puppetlabs.com
[8] cgit – CGI for Git, http://hjemli.net/git/cgit
[9] ITIL - Information Technology Infrastructure Library, http://www.itil-officialsite.com
[10] DevOps - portmanteau of development and operations, http://devops.com