SaltStack Whitepaper – SaltStack for System Engineering and Administration

By: Tom Barnes & Garrett Bolling

February 12, 2017

325 Gambrills Road, Suite D,
Gambrills, Maryland 21054
410-439-1944 P/410-439-1945 F
www.fuseENG.com

Table of Contents

Executive Summary 3

Introduction 3

Problem Statement 4

Solution 5

Solution Test and Evaluation 6

Conclusion 9

325 Gambrills Road, Suite D, Gambrills, Maryland 21054
410-439-1944 P/410-439-1945 F
www.fuseENG.com

Executive Summary

System engineering and administration technologies are constantly changing, thus, it is imperative technologists keep well-informed of trending technologies and allow our mindsets to be open to new ways of managing ourselves, and our technology. By leveraging SaltStack technology, Fuse Engineering is changing the way system administrators manage the entire enterprise cluster, follow industry best practice, focus on configuration management, and remain focused on meeting strict system security guidelines. SaltStack systems management software effectively automates time-consuming IT operations tasks and essential processes to ease the management of IT systems within the enterprise. This white paper will outline how SaltStack, combined with other technologies such as Git and Jenkins, provides the benefits of a modern systems and management platform built for agile, efficient and secure delivery of IT-dependent services. The SaltStack integration bridges an all-too-common gap in our business between the people and process and the technology they need to get work done and sustain competitive advantage.

Introduction

Utilizing SaltStack, system engineers and administrators can manage an entire enterprise cluster with ease. Industry system administrators *say* they are responsible and follow best practices, but rarely is the tedious work of configuration management and versioning control completed correctly, which leads to inefficient testing and evaluation (T&E) before moving from

development to integration, and ultimately, production environments. By introducing system administrators to IT configuration management tools such as Git and Jenkins, efficient tools assist in the automation, testing, and documentation of new releases. The central idea is that these types of products and their practices will reduce surprises when new releases move into production. System administrators are then held responsible for the changes they implement, documentation will exist for future team members, and historical knowledge of system development is readily available while ensuring compliant solutions are delivered to customer on time and on budget.

Problem Statement

Configuration management is a necessity in the world of system administration, but how do you push changes out? Who validates those changes? How are they documented? All too often the role of a system administrator requires pushing system changes quickly and efficiently, in between “fires”. System administrators exchange thoroughness for quickness, often compromising documentation. SaltStack can quickly deploy changes across an enterprise; wouldn’t it be nice if teams could also concurrently document these changes? What if it was possible to test those changes in development before moving them to a production environment? Using legacy revision control systems has long been an accepted solution. File checkout; make changes; file check-in. Rinse, repeat. If acting on the fly, sometimes a system administrator would “comment” what was completed, but most times comments are left empty or vague for someone to try to figure it out later. It is almost impossible to troubleshoot

an issue when you have poor documentation to review, and a co-worker does not recall, or confess, to what changes were actually implemented. How do we solve this problem?

Consider the similarities between the scripts used when writing Salt States, in comparison to actual code, and begin to think more like a developer. Utilize available configuration management and version control tools and products; track changes and test changes before they are deployed to production. When we begin to treat Salt States like code, we can become responsible for the changes we deploy.

Solution

In our environment, the system administrators noticed by implementing SaltStack, reboots were no longer required after each change per the prior Revision Control System (RCS). Additionally, system administration tasks no longer had to be completed as “root”, which was directly affecting production and security protocols. Gone, was poor documentation and deploying those changes to every individual box in the enterprise. Never again, allowing system administrators’ commenting code changes with “made change” and other administrators then asking, “What change?”

Using SaltStack, organizations can implement system changes across the entire enterprise. After accessing the Salt Master, system administrators may manage a file or start a service across each system attached to the SaltStack. In engineering a method to quickly track streamlined changes, our system administrators implemented methods and tools commonly utilized by software developers. This was completed by deploying Git and Jenkins technologies into the environment. Git, a version control system (VCS) for tracking changes in computer files

and coordinating work on those files among multiple people; Jenkins, cross-platform, continuous integration and continuous delivery application that allows teams to build and test software projects continuously making it easier integrate changes to the project.

Solution Test and Evaluation

Our environment utilized multiple Salt Masters, thus initially, our system administrators started small and only chose one cluster to begin. Each Salt State for that cluster was checked into a Git repository, utilizing BitBucket, a version control system that makes it easy to collaborate with teams. With Git functions such as “git add”, “git commit” and “git push”, changes were made inside files. Git function “git commit” forces the system administrator to add work comments. Our system administrators were able to make changes in the Git repository and could see those changes tracked. Next, utilizing a bash script and cron schedule, we were able to automate this process. The script, set to run every 10 minutes, sufficient since the salt minions on the servers check for compliance every 15-30 minutes. It was important to ensure that changes weren’t being made on the Salt Master, so the bash script includes a “git reset --hard" so any unauthorized changes would be reset within the 10-minute window and then pull the latest changes from the repository.

Example.

```
### If Running, kill and run again below:
if ! [[ -z `ps -ef |grep “/usr/bin/git --git-dir=/srv/.git pull –q” |grep -v grep` ]]
then
ps -ef |grep “/usr/bin/git --git-dir=/srv/.git pull –q” | grep -v grep | awk ‘{print $2}’
|xargs kill
sleep 1
fi
```

```
### Run the Git Reset command
cd /srv; /usr/bin/git --git-dir=/srv/.git reset –hard origin/master –q ; /usr/bin/git --git-dir=/srv/.git pull -q
```

When using Git, it is important to remember to annotate where changes are being pushed to, in the Salt Master. System administrators should avoid pushing changes to the master branch. This means, even for testing a particular state, it pushes out to production. There's no validation that the state will work, or it will cause an error and stop all of Salt from functioning.

To manage this risk, our system administrators implemented a test server to validate the code. Utilizing Jenkins, an open source automation server boasting flexibility and community support, our system administrators were able to get this service configured to work with the Salt-API via a community supported plug-in. Our system administrators configured Jenkins to pull code automatically from Bitbucket. Since Bitbucket requires credentials, our system administrators created SSH keys, giving public key access to read the repository, via the Deploy Keys section of our Git repository. Additionally, a configuration was made in Jenkins to poll Bitbucket every 2 minutes, ensuring that any changes could be quickly tested before the Salt Master was set to pull them.

Upon configuring Salt-API, a discovery was made that additional changes were required with the Salt Master before continuing with Jenkins. This is a simple rpm install of salt-api, which will include python-cherrypy. Once installed, system administrators need to modify the Salt Master configuration to enable functionality.

Example.

/etc/salt/master:

```
rest_cherrypy:
port:8000
disable_ssl:True
external_auth:
  pam:
    jenkins:
      - .*
```

To finalize this solution, our system administrators created a new path Jenkins, so that the cloned Git repo would be added into the Salt Master configuration. Locating the workspace directory for the build, our system administrators modified the Salt Master configuration again:

```
/etc/salt/master:
file_roots:
base:
- /root/.jenkins/jobs/Salt-Dev/workspace/salt
common:
- /root/.jenkins/jobs/Salt-Dev/workspace/common
server:
- /root/.jenkins/jobs/Salt-Dev/workspace/server
```

Restarting salt-master and salt-api services, the changes were recognized by the system.

Testing another build allowed confirmation that this service was working and testing the states on all of our production boxes. Additionally, our system administrators added a notification configuration, which sends an email to anyone who's broken the build, so that it will be known immediately if there is an error to correct.

Conclusion

Changing our thought process on how we view and implement our SaltStates was vital to develop a fully functional system. We now have the ability to hold people accountable for their changes. We have documentation and we're creating a process for configuration management and our customer. We'll also have historical knowledge that's documented right in the code. We can see how a small state grew into a large one. Even better, we'll be able to show any new team member the changes instead of telling them about an email that was sent long before they even started working with us.

IT system administrators are stretched thin with the increasing scale, diversity and complexity of infrastructure typical to the modern enterprise. Hybrid and multi-OS environments are the norm, shadow IT is pervasive, applications and code are everywhere, security threats are rampant and the management tools built decades ago are not fit for today's environment. Legacy systems and configuration management software tools are simply not designed to orchestrate and automate management of current infrastructures.

While SaltStack gained much of its popularity as a tool for orchestrating DevOps and configuration management, it was originally built for fast and scalable command and control of infrastructure operations. By utilizing SaltStack's ability to provide scalable automation and flexible orchestration, combined with native configuration management, we can provide significant advantage over alternatives for the efficient management of enterprise applications and IT resources that often run on multiple clouds and on in-house infrastructure. Combined with version control software, this toolset can be custom tool for effective system administration.