De-Orbiting Satellite

Keiran Sweet, Linux Administrator

Overview

This document details how a medium sized Linux environment was successfully migrated from RedHat Satellite server to a more robust platform using Puppet 2.7, Foreman 1.x and Apache backed by some simple shell scripts.

In doing this we were able to:

- Define a known good baseline for all servers in the fleet
- Deploy new servers rapidly across the estate to the known baseline
- Reconfigure existing servers to the new baseline
- Quickly deploy and roll back changes to the server estate
- Automate regular administrative tasks
- Provide an audit trail of changes to the fleet
- Significantly reduce cost through software licensing and administration overheads
- Provide a CMDB for the entire Linux estate

Introduction

An audit of the Linux environment was done to generate a list of regular operational issues. The majority were a direct result of inconsistency of the operating system configuration and the inability to deploy changes in a timely fashion with a suitable rollback plan and change log.

This document serves to capture the lessons learnt when migrating from Satellite and related technologies to Puppet and to serve as a case study for other administrators who may be attempting a similar exercise.

It is important to note that whilst this approach to migrating from Satellite server was ideal for this particular business and environment, it is not suitable for everyone. It is also worth mentioning that a number of the Puppet techniques used in this document may no longer be considered best practice as the product evolves rapidly and features that are now available such as hiera did not exist at the time the environment was being designed and deployed.

Environment Overview

The environment described in this document is a relatively common deployment of Redhat Enterprise Linux (RHEL) in a medium to large enterprise environment. This approach however can also apply to any Enterprise Linux based deployment that uses RPM packages with management tools such as Spacewalk, YUM & up2date.

When this migration was initiated, the environment was a mix of RHEL 4 & 5 with with an imminent requirement to start deploying RHEL6. One of the driving forces behind the migration and re-architecture of the environment was the Satellite server version in use was 5.3 which did not support RHEL 6.

Deployments were done via a combination of kickstart installations from the Satellite server followed by extensive manual and often error prone administration work to bring the system into a production ready state. There were also a large number of “rogue” servers that had been built from from Redhat installation media and never integrated with the Satellite server, and as such were configured to use external software repositories such as Redhat Network or EPEL.

In regards to the servers that were connected to the Satellite server, the environment made use of the following Satellite server components:

- Software channels - For storing and deploying RedHat, 3rd Party and in-house RPM packages and patches.
- Configuration channels - For deploying basic configuration files to the estate.
- DHCPD & Cobbler - These bundled aspects of Satellite server were responsible for managing the PXE boot components as well as the Linux DHCPD scopes

Issues with Satellite server and review of the platform
We were able to identify the following shortcomings with the Satellite server deployment in the environment.

- Poor reporting
    - What changed overnight / recently ?
    - Has anything not checked in recently ?
    - Any Stale or Duplicate nodes ?
    - Inventory and CMDB functionality extremely limited
      (How many servers do we actually have ?)

- Software deployment
    - Slow installations of packages via YUM & up2date
    - Adding a single package to a software channel could take up to an hour for the metadata to be generated
    - Mirrors of the software had to be done via Satellite server proxies at additional cost

- Configuration channels
    - Duplicate configuration channels that should deploy the same files
        - ie, multiple sudo, ntp, access.conf files
    - Were not location aware
        - All servers got a static set of configuration files not optimised for their datacenter location
        - Incorrectly configured macro support

- Kickstart
    - Slow builds
        - The client registration during kickstart would take minutes before the OS install started
        - The OS was patched after initial installation via a yum update.
          An inefficient two step process
    - High touch
        - Hosts were being built as localhost.localdomain then had to be manually renamed before use
        - All production readiness tasks were done manually
          ie, installation of platform packages (VMWare Tools, etc)

- Vendor lock in
    - Unable to manage other Linux and UNIX platforms
      ie, Ubuntu, CentOS, Solaris

- Expensive
    - Satellite server costs were tens of thousands annually
    - Additional functionality incurred additional cost

- Overly complicated architecture
    - Oracle , Jabber, Java, Apache, Custom filesystem structure

- Limited functionality
    - Lack of user and service management
    - Limited version control
    - Changes made to the kickstart components could not be applied to the existing fleet

- Internal administration issues
    - Satellite best practices weren't followed
        - Too many cloned software channels
        - Too many cloned configuration channels
    - Allowed configuration channels and files to fork for no reason
      (sudoers, ntp, sysctl, etc)

In addition to addressing the above, we also required the following additional functionality:

- Flexible Configuration management
    - We needed to be able to override legacy settings on a per server basis when required

- Provisioning
    - Integrate with kickstart and provisioning workflows to build new systems with ease
    - Ability to test changes before promoting them into production

- Software management and distribution
    - Support yum and up2date on modern OS's for RHEL 4, 5 & 6 clients
    - Support HTTPS & Signed packages for security
    - Support importing of errata and new OS versions from Redhat when required
    - Fault tolerance - yum & up2date mirror support
    - Simple - We wanted to be able to upgrade this platform easily in the future

New management platform

We decided to address the shortcomings of the system in place by designing a modern configuration management and software deployment platform that supported RHEL 4, 5 & 6 on both new and existing instances.

Based on past experience with Puppet and distributed YUM platforms, the new configuration management platform was to be as follows:

- Configuration Management
    - Puppet 2.7
        - Apache httpd
        - Mod_passenger for scale
    - The Foreman 1.x
        - Node classification
        - Report visualisation
        - Provisioning
        - DHCP / TFTP / Puppet management via smart-proxy
    - Puppet Dashboard
        - Used for additional report visualisation
    - Version control
        - Subversion - Already in production
        - WebSVN - Subversion Visualisation

- Software management and distribution
    - Apache
        - Createrepo fand yum-arch or generating YUM client metadata
        - yum-arch for generating up2date client metadata
    - rsync for syncing data to slave software distribution servers
    - Simple bash scripts to trigger metadata generation when required
    - Puppet modules to wrap it all up cleanly on the server, slave and client side
    - Reliable documentation on how to use the system

Integrating the new platform
Work began with the build out of the core infrastructure and related puppet modules, slowly bringing the existing servers under Puppet management.

As this is a production environment running a live financial business, care had to be taken that proactive work did not negatively impact the production environment during business hours.

So, how did we get there ?

Cleaning up the Satellite server
Before we could start moving forward with a new configuration management tool, we needed to clean up the Satellite server to a level that could be easily migrated from, there were two aspects to this:

- The configuration channels
- The software repositories

The configuration channels that had been forked and duplicated had to be consolidated into single set that was suitable for the whole fleet.

Examples of the tasks undertaken were:

- Multiple sudoers, ntpd, ssh and access configuration files files were consolidated where possible
- Where multiple configuration files were required we collapsed them into small sets that could be easily selected via top scope variables using Puppet and Foreman
- Where there were inefficient configurations in use we simplified them
  ie, Moving from complex configuration files to simple LDAP groups

This was a complex and time consuming process where caution was needed as it had the potential to negatively impact your fleet.

Once the configuration channels had been simplified we then started a similar task with the software repositories.

A number of the issues that were experienced in the software repository space was for the most part self inflicted. Rather than keeping things simple and focus on a small number of OS versions and baselines, the previous administration staff had regularly cloned the various RHEL channels on RHN, leaving us with:

- 15+ RHEL 5.x Channels, each with cloned sub-channels with no labels or descriptions of when the snapshot/clone was taken
- 10+ RHEL 4.x Channels, each with cloned sub-channels with no labels or descriptions of

when the snapshot/clone was taken

The result was that all of our Satellite managed servers (about ~200) were spread across 25+ channels. Some channels were no different to others, but cloned anyway, some had just one server in them.

The solution was to re-baseline the software.

While developing the configuration management platform, we had decided that any system that didn't fall into one of these 3 categories were to be migrated accordingly.

- RHEL4 - 32-Bit
- RHEL5 - x86_64
- RHEL6 - x86_64

I created a new snapshot of a recent stable version of RHEL 4 & 5 with all suitable sub-channels, beneath this I created a "System support" channel that was used to store all major version specific packages we had such as HP PSP's, VMWare tools and administration utilities and when ready I started moving the entire fleet into this new baselined OS version.

- RHEL 5.8 Server - SOE OS - Cloned: 20/4/2012
    - RHEL 5.8 Server packages
    - RHEL 5.8 VT Virtualization packages
    - RHEL 5.8 RHN Tools packages
    - RHEL 5 - System Support packages

- RHEL 4.8 Server - SOE OS - Cloned: 20/4/2012
    - RHEL 4.8 Server packages
    - RHEL 4.8 RHN Tools packages
    - RHEL 4 - System Support packages

This also took some time as we needed to reboot the platforms to enable the new Kernel and patches, as well as update the non-puppet kickstart environments to use the consolidated software and configuration channels.

When complete I started getting the Puppet software ready and writing all the puppet modules.

Software and package builds

As Puppet is a Ruby application, we required a feature rich Ruby installation for each of the RHEL versions that we had in the environment. As the Redhat provided Ruby versions for each RHEL release varied greatly in version and functionality, we opted to build our own feature rich Ruby version that installed in /opt.

At the time the preferred version for Puppet was Ruby 1.8.7.
As we were still relatively new to Ruby at the time with opted to go with Ruby Enterprise as it not only provided us with fully functioning gems and passenger support, but it also contained a number of fixes that addressed memory leaks that were relevant to environments running Puppet in daemon mode.

It is worth noting that Ruby Enterprise has since been end of lifed and all fixes rolled into the Ruby mainline which should be used instead.

To ensure a consistent feature set for Puppet across all OS platforms, libraries such as augeas, mysql and LDAP were installed. We also wanted to ensure that the versions of gems that were in use for each OS platform were as close to identical as possible,for this task we leveraged the *rubygems_snapshot* tool that allowed us to export a list of gems and their versions and apply them to Ruby installations on other platforms.

The gems that we bundled with the Ruby installation were derived from the Puppet, Foreman & Smart-Proxy requirements documentation.

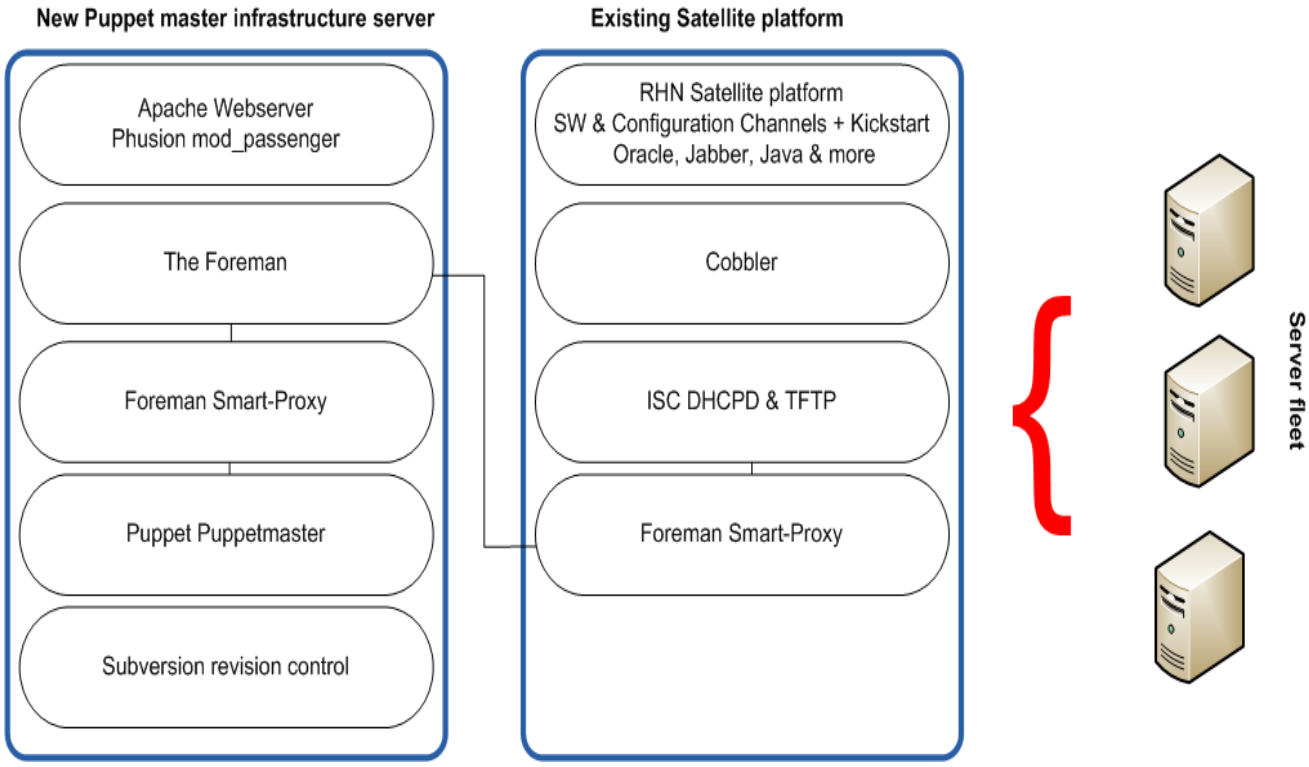
The binaries for each Ruby installation were then packaged into RPM's for each particular OS type and imported into the relevant Satellite server channels alongside its dependencies. This ensured that they were available for installation and we began to test them thoroughly.

Once the base Ruby foundation was ready, Packages of the latest stable Facter and Puppet versions were created using the bundled RPM SPEC files that are in the source tarballs which were retrieved from the Puppet website.

We were then left with a feature rich Puppet & Facter installation for all our core operating platforms.

Building the Puppetmaster and developing the Puppet modules

Building the Puppetmaster using our new Ruby and Puppet packages was relatively simple, we selected Foreman as our external node classifier because it was more feature rich than others available and the ability it had to quickly graph fact data as well as manage the provisioning workflow through the DHCP, TFTP, Kickstart and Puppet phases from a single interface.



We opted for a parallel deployment of the new configuration management platform as it was important to be able to revert to the original system in the event of issues. Fortunately this was easily done by hooking Foreman into the Satellite servers DHCP and TFTP services via an instance of the smart-proxy on the Satellite server as its DHCP service supported the OMAPI API.

Due to this approach, any server booting off the network that had not been setup in Foreman would be presented with the standard Satellite PXE build environment. However nodes that were being provisioned via Puppet and Foreman would have their own DHCP reservation and pxelinux.cfg configuration stanza put in place. This allowed the two to platforms to live together without conflict.

Puppet module development and ENC design

To keep things modular and simple, a module was written to manage each configuration component within Satellite. All modules and configuration files versions were controlled using Subversion.

In almost all cases we developed one module per service, following the Package -> Configuration ~> Service model. However in the case of some more complicated functionality, such as authentication, smaller services were merged into a single module for ease of management.

As we were importing an existing fleet into a configuration management tool and the fleet was full of unknown local configurations it was extremely important to be able to easily and quickly override the Puppet configuration of a particular server. This influenced the design of the Puppet modules and the ENC greatly.

When designing the Foreman and ENC structure we opted to avoid pushing too many overrides for legacy hosts into the ENC variable namespace. We chose to store them in the Puppet modules via host overrides which allowed us to track overrides for specific hosts to a svn commit message explaining who put it there and why.

We implemented this as follows (In order of preference):

File{} source precedence
We found the file precedence feature extremely useful in being able to quickly override host specific configuration files on a per node basis without having to edit puppet manifest logic or converting complex configuration files into ERB templates.

For every file resource in the modules we put this framework in place.

```
file { /etc/sysctl.conf :
        source  => [ puppet:///modules/tg_ostune/sysctl.conf.$::hostname,
                     puppet:///modules/tg_ostune/sysctl.conf,
                   ]
    }
```

ERB Template logic & ENC Values
Ruby templates are very flexible especially when more dynamic configuration files are required, such as those that require the hostname of the server to be specifically defined in the configuration file. Although we don't use templates heavily at this stage, a number of them are able to manipulated using ENC level variables.

The below example shows how we can override the backup server a host uses rather than the one that is defined from the site definitions module.

```
<%# -%>
<% if has_variable?("arkeia_backup_server") then -%>
<%= @arkeia_backup_server +':617' %>
<% else -%>
<%= scope.lookupvar('tg_site::backup::server') %>:617
<% end -%>
```

Foreman as an ENC

At the time of development & deployment, Foreman didn't support parameterized classes and we needed flexibility in the order in which the modules were applied to the hosts. To address this shortcoming, I opted to wrap up the modules up in another class which was responsible for applying the base set of modules to the server in the correct order.

I then used foremans host group functionality to apply this class to nodes within that hostgroup. The wrapper class that applies all the standard SOE modules to the host is shown below.

```
class tg_applysoe::baseserver {

    include stages
    class { 'tg_users': stage => pre }
    class { 'tg_puppet::client': stage => pre }
    class { 'tg_site': stage => pre }
    class { 'tg_rhn': stage => pre }
    include tg_snmp
    include tg_ntp::client
    include tg_platform
    include tg_syslog
    include tg_backups
    include tg_banners
    include tg_monitoring
    include tg_sudo
    include tg_useraccess
    include tg_resolv
    include tg_ostune
    include tg_mail
    include tg_sar
    include tg_ssh
    class { 'tg_auth::ad': stage => post }

}
```

Foreman host group configuration
Foreman's host groups functionality provides the ability to group similar servers into common groups for the application of specific values and configurations.These values can then be used by Puppet to adjust the configuration when it is applied to a server or to apply a specific set of puppet modules and classes .

Nested host groups were used to provide a host the following information

- Top Level Host group: *Production Servers Host group*
    - Servers in or below this host group are placed into the production Puppet environment

    - Nested Host group : *Data center Location*
        - Servers in this host group inherit the “datacenter = NAME” variable which is used in the Puppet logic to ensure the host is configured to use its most geographically relevant infrastructure services.

        - The nested host groups also ensure that the correct Puppet classes are applied to the member servers.
          For all data center host groups the class “tg_applysoe::baseserver” is applied at the nested host group level.

For each datacenter location, we have a corresponding nested host group, and we replicate the structure above for additional Puppet environments so that we can test changes to puppet logic and ENC parameters before applying them to production.

This structure is as follows

- Production Servers
    - Global Switch 1 Servers
    - Global Switch 2 Servers
    - Slough DC 1 Servers
    - Slough DC 2 Servers
    - Unique servers*

- Testing Servers
    - Global Switch 1 Servers
    - Global Switch 2 Servers
    - Slough DC 1 Servers
    - Slough DC 2 Servers
    - Unique servers*

** The unique server hostgroup does not apply the tg_applysoe::baseserver wrapper class to hosts present in it, and is used for servers we have to apply customised sets of puppet modules to.*

Cleaning up the provisioning system

While all this work was going on, provisioning was still being performed manually, and it urgently needed to be streamlined in a way that would produce a more consistent server build with less manual interaction.

Although there were a number of tools to help automate the creation, installation and management of both physical and virtual machines alongside Puppet (LibVirt, Razor, etc), we opted to focus on the common platform in place within the environment, specifically PXE boot & Kickstart.

We then utilised an in-house Puppet module called tg_platform that would be responsible for applying platform specific configuration changes to a host such as the installation of VMWare Tools and HP support packs.

To achieve this a new provisioning and kickstart workflow was defined as follows:

1. A new host is defined within Foreman, specifically providing
    a. The host name
    b. OS version
    c. Disk partitioning layout
    d. Selection of the appropriate Data Center Hostgroup
    e. Primary MAC address.

2. Foreman communicates to the smart-proxy on the Satellite server putting the following in place
    a. A DHCP reservation for the new server
    b. A suitable pxelinux.cfg snippet for the host based on the primary MAC address of the server. This contains all the information required for the host to start the kickstart process.

3. The host is then booted from the network using PXE, retrieving the kickstart file and performing the following steps
    a. Set the hostname of the server
    b. Partition the disk as required
    c. Install the basic OS on the server
    d. Register the server with Satellite so that it can obtain the latest approved patches and install 3rd party RPMs*
    e. Run a yum update to patch the server to the latest RHEL baseline
    f. Install our in-house Ruby and Puppet packages using YUM
    g. Put in place a basic Puppet configuration file that defines the puppet master and environment for the server
    h. Run puppet once using the --tag puppet::client option to bootstrap only the puppet client classes, finalising the puppet configuration on the new node

i. Run a full puppet run via *puppet agent -tov* that fully configures the node
j. Trigger a notification to Foreman to advise the build is complete and that the pxelinux.cfg snippets for the node can be removed
k. Trigger a server reboot

4. Once the host has rebooted, the Puppet reports for the node are checked to ensure there are no failures, and the server is ready for use.

** To achieve this, A new set of installation keys on the Satellite server were configured so that servers that registered with it only had access to clean software channels we had created, without configuration channels.*

Importing existing nodes into the new system

The existing fleet of servers had to be carefully imported into the Puppet once the server provisioning component of the new platform had been finalised.

The existing servers were audited and we placed them into one of two categories:

1. Candidates for importation into Puppet
   These were servers that we had suitable Ruby and Puppet packages for that we scheduled for "Puppetization"

2. Candidates for rebuild and migration
   These were servers that were too old and were unable to be put under Puppet management. We had to migrate these applications to Puppet managed servers.

We took the following approach where servers could be imported:,

1. Ensure that the server was registered with Satellite for software installation, specifically to the consolidated software channels we had created.

2. If required, upgrade the server to the latest OS baseline via yum

3. Ensure that the server was not configured to use any of the Satellite configuration channels - Puppet would now be responsible for this functionality.

4. Using yum or up2date - Install the puppet packages from Satellite

5. Place a basic puppet configuration file on the host that was suitable for bootstrapping the client

6. Run puppet once using the --tag puppet::client option to bootstrap only the puppet client classes and register the server in Foreman

7. On the Foreman GUI the server will now be visible in the hosts list

8. Using the Foreman GUI, edit the host and place it into the required nested host group based on environment and datacenter location

9. Run puppet again to apply all the base SOE modules to the host and bring the server up to a known baseline - *“puppet agent -tov”*

Using this approach we were able to get the entirety of the fleet under Puppet control and as such could remove the configuration channels. The core OS components of the Linux estate were now completely managed by Puppet.

For the first time in the fleets history control had been regained; common issues such as running out of file descriptors, users not being in access.conf files and NTP services not running were a thing of the past allowing administrators to focus on more pressing issues.

The Removal of Satellite for software management

As the configuration of the servers base OS was now under Puppet control, our focus turned to the framework that provided the software to the servers for installation. This was being handled by the software management component of Satellite server, alongside Satellite / RHN plugins for YUM or up2date.

This platform had a number of issues, not limited to:

- There was no RHEL 6 support in the version we were using
- Adding a single package to a channel or repository could take in excess of an hour to generate the YUM or up2date metadata
- Replicating package data to mirrors required additional commercial software or unsupported tools
- Registration of a new server to the Satellite server to access the RPMs was extremely slow
- The current kickstart procedure required the host to be patched after the initial OS installation, thus build times took twice as long as they needed to
- The whole platform was backed by a complicated file system structure and an Oracle database which made management problematic.

As such, we decided to replace it.

The Satellite replacement solution

Our requirements were simple, we needed something that:

- Served RPMs and their metadata to Linux servers across the network via HTTP/HTTPS
- Allowed RPMs to be added/removed easily and the metadata to be quickly updated
- Supported mirrors so that clients could be aware of multiple sources across the environment
- Supported both YUM and up2date clients
- Easy to use and extend
- Compatible with kickstart and provisioning

The technology we used to achieve this was:

- Apache - Responsible for exposing the packages and metadata to the network
- Createrepo for generating YUM client metadata for RHEL 5+ systems
- yum-arch for generating up2date client metadata for RHEL 4 systems
- rsync for syncing data to slave software distribution servers
- Some simple bash scripts to trigger metadata generation when required
- Puppet modules to wrap it all up cleanly on the server, slave and client side
- Reliable documentation on how to use the system and add additional software from RHN or trusted 3rd parties.

Building the software distribution platform

The first stage was to build a primary yum server that would store all the software packages that were currently being served by the existing Satellite server. A simple filesystem and URL structure was developed that allowed nodes to locate their appropriate YUM and up2date repositories via Facter data (self classification).

Exporting the data from Satellite into the YUM server proved initially quite problematic. We encountered quite a number of issues with mrepo and similar tools when talking to Satellite, specifically with SSL communication and up2date-uuid files. We used the reposync(1) utility as a work around to export the data from each of the Satellite server channels into the required directory structure on the yum server. This was unfortunately a manual task, however as all the software channels had been consolidated manually it was relatively painless.

In the case of RHEL6 , We took a stable snapshot of the software and packages from RHN and placed it alongside the other versions on the YUM server.

All the software packages were now on the yum server. Unfortunately the reposync utility only downloads the package files, it does not obtain or create any of the required metadata for

package managers such as yum or up2date.

We were required to support both up2date and YUM clients and therefore generated the metadata for each repository using the following tools:

- createrepo
    - Creates YUM metadata from directories that contain RPM packages
    - Suitable for RHEL 5 and above clients that use YUM for package management
    - Provided with RHEL5 and above via the createrepo package

- yum-arch
    - Creates up2date metadata from directories that contain RPM packages
    - Suitable for RHEL4 and similar clients that use up2date for package management
    - No longer available in modern RHEL releases, as such needs to be built from legacy SRPMS

A number of shell scripts and cron jobs were written to assist in the modification of the repository data and the generation of the package manager metadata.

The second stage of the platform build was to implement yum server mirrors for distribution across the environment. These were to be used in the event of server failure or scheduled maintenance of the YUM servers. We achieved this by defining a "yum slave" role.

A “yum slave” is a read only replica of the primary yum server. For simplicity the slaves would sync via cronjob that pulled the package data from an rsync server on the primary YUM server. rsync was chosen for this functionality as it allowed all slave nodes to consume both the RPMs and the repository metadata which then didn't need to be regenerated on the slaves and it also provided quick updates after the initial sync. Once a new YUM slave was brought online the yum definitions for the servers could be updated so that they were aware of the new package sources.

Once complete, we wrapped up the YUM server and slave architecture up in a Puppet module that let us deploy and manage them easily within the fleet.

Migrating to the new platform

The new platform functionality was thoroughly tested by manually configuring test clients against the new yum servers as well as creating an alternate kickstart configuration that would have the clients install the OS and all updates from the new platform.

These tests gave us a high level of confidence that the platform was fully functional and production ready and we then used our Puppet environment to swing the clients over from Satellite in the following manner:

Existing servers
Initially, A RHN module was developed and rolled out to the fleet which was responsible for ensuring that the Satellite client configuration files and daemons were functioning correctly. This provided the ability to alter the fleets RHN/Satellite client configuration from a central location.

We then developed a new module that we could swap with the RHN module called tg_pkgmanagement, this module was responsible for disabling the Satellite YUM and up2date plugins on each node, replacing their functionality with a set of pure YUM or up2date repository definitions where the same software packages could be installed independently of Satellite.

We were able to then easily make this change in our tg_applysoe::baseserver class by replacing tg_rhn with tg_pkgmanagement. Upon completion the nodes slowly checked in over the period of an hour finalising the migration from Satellite to the new software distribution platform.

Provisioning changes
Once the existing fleet had been transitioned to a Satellite-less environment, the final component was to ensure that all new server builds were using the YUM framework and no longer registering with Satellite server.

To do this the Satellite server registration components in the kickstart file had to be removed and replaced with a set of additional YUM repo definitions as shown below:

Original Kickstart functionality

```
rhnreg_ks --activationkey=$rhn_activation_key
```

New Functionality (Using Foreman templates)

```
repo --name="Kickstart Redhat
<%=@host.operatingsystem.major-%>.<%=@host.operatingsystem.minor-%> Server packages"
--baseurl=http://yum.domain.tld/yum/<%=@host.architecture-%>/YUM/RHEL/<%=@host.operati
gsystem.major-%>.<%=@host.operatingsystem.minor-%>/rhel-x86_64-server-5/
```

As Foreman was being used for the ENC and provisioning frontend, it was possible able to alter the Kickstart file's YUM repository configuration dynamically using Foremans template functionality. There were now consolidated kickstart profiles for both RHEL 5 & 6.

In addition to this change, we had to also update the Operating system installation media definitions in Foreman to so that they would reference the media on the YUM server rather than

Satellite.

Turning off the Satellite server
At the end of this project we were still left with a Satellite integrated DHCP and TFTP server on an aging RHEL4 host.The final tasks before we were able to turn off the Satellite server was to migrate these applications to a new server running the Foreman smart-proxy and power off the host. This was a simple change and closed out the Satellite server migration for the group.

Post migration conclusions, roadmap and summary
The project was completed within 12 months despite concurrently juggling BAU administration tasks. The outcome of the project shows that it is possible to migrate from Satellite to Puppet in a staged approach while limiting the risk to the existing production environment.

On conclusion of the project the benefits of moving to Puppet were immediately evident:

- Our costs are down
    - Our Redhat licensing costs dropped by thousands of pounds just on Satellite
    - There is no longer guesswork on how many RHEL licences we need

- The business applications are more stable
    - Applications no longer fail due to misconfigured configuration files
    - Daemons such as NTP, SSH, SNMP and SMTP are ensured to be functioning
    - New application servers are pre-configured with all required dependencies (Libraries, SVN, htop, etc)

- Our users have a common experience across the fleet
    - No more "Why does this server not have XYZ installed ?"
    - A lot less of "Why can't I login to server ABC ?"
    - No more “I cannot run this sudo command on server X”

- Management of the fleet is much easier
    - We know how many servers we have, where they are and what they do
    - We can deploy changes to the entire fleet in less than an hour, and if required, roll them back
    - We can test changes to the fleet before full scale deployment
    - Our environment is simpler and easier to maintain
    - Server provisioning now takes 5-10 minutes for both physical and virtual machines
    - We have a known baseline of the Operating system
    - We can focus our time on more important or interesting work
    - We now have a visual representation of the configuration status of the fleet and can generate reports on them (Foreman)
    - We can deploy new data centers and sites in hours instead of weeks

    - We can track changes in the Linux estate via Subversion commit logs and reference them in changes and tickets
    - We can add new versions of Linux easily to the YUM server for deployment and patch the environment to newer versions RHEL
    - We can deliver other projects such as monitoring as our SNMP configuration files are standardised

Where to now ?

The removal of Satellite and managing the base OS configuration with Puppet was only the first phase in getting the fleet to a more manageable state. Each week we select the next “lowest hanging fruit” from the sites problem tree and solve it using Puppet.

We are now regularly building new modules that sit on-top of the base SOE modules that provide additional Puppet managed functionality such as:

- Oracle server core requirements
- Proxy, Web and Subversion servers
- Collectd for additional performance metrics

The long term goal is being able to deploy all our platform functionality via a couple of mouse clicks from the Foreman web interface.

In addition to this we are also evaluating other components of the Puppet ecosystem such as MCollective to deploy changes more rapidly and to offer real time reporting and monitoring tasks. We are also investigating the adoption of Puppet Enterprise to allow us to upgrade the platform rapidly.

Additional lessons learnt and technical comments

What follows are some additional lessons and notes about things we encountered during the migration that may help other administrators achieve with Puppet what we did.

Satellite server & Smart-Proxy

Integration of Satellite with a smart-proxy for DHCP/TFTP work building out a provisioning system works very well. It never caused us issues and allowed us to phase the new system in and instantly not use it if ever required.

The steps in the smart-proxy documentation detail how to integrate it with the DHCP servers OMAPI API and permit the smart-proxy user from creating and modifying files under /tftpboot on the TFTP server.

Ruby and packaging
As we had a number of OS platforms to support we opted to build our own feature rich version of Ruby as the OS bundled version wasn't suitable for use. The approach we took was to bundle everything into a single RPM that we could have Puppet and Facter reference directly. This worked well initially, however as time went on we found that maintaining these installations became quite an issue when we needed to make changes to or add additional Ruby gems.

At the time tools such as FPM were not available so this was extremely manual.

Looking at the approach made by Redhat and Puppetlabs, it seems that it is preferable to build a Ruby framework where each additional Ruby component is delivered using an individual RPM. In our case, it is likely we'll look to move to Puppet Enterprise and leave the packaging to the experts.

Ruby Enterprise has now been end of lifed and all fixes rolled into the Ruby mainline. If you are looking to build your own Ruby install, it is recommended to use the official Ruby source for your builds.

Debugging YUM & up2date issues over HTTP
When we were initially testing the new YUM/up2date over HTTP framework we ironed out all the bugs from the clients by not using SSL initially. We used tools such as HTTPry. an excellent utility for watching HTTP connections in real time, rather than tcpdump and a number of command line switches.

The File{} provider and precedence
Although file precedence is extremely useful for file based overrides, the downside is that apache will generate a 404 error for each path that is searched but not found.

Using the below example, a 404 error will be generated if the file referenced by *sysctl.conf.$::hostname* does not exist within the module and Puppet will then try the next source location.

```
file { "/etc/sysctl.conf":
      source => [
                 "puppet:///modules/tg_ostune/sysctl.conf.$::hostname",
                 "puppet:///modules/tg_ostune/sysctl.conf,
                ]
      }
```

In this particular environment there are large numbers of file resources with hostname based

override functionality in place, resulting in thousands of 404 errors. This makes reporting on your Apache log files rather difficult.
This issue can be solved using Hiera, ERB templates or general refactoring of the module to suit your site specific needs.

Puppet filebucket
When importing existing servers into a Puppet environment, the original configuration files are often replaced with standardised ones causing unexpected issues.

Puppets filebucket functionality aids this greatly by sending the original file back to the Puppetmaster server for later retrieval if you ever need a copy of the original configuration file that was in place on the host to understand the configuration or to put it back in place.

The usefulness of this feature cannot be understated.

Foreman packaging
After struggling with packaging Foreman ourselves in the initial 0.4x versions we opted to run it out of a cloned environment using git. Fortunately recent versions of Foreman are available via RPMS. It is suggested that you use these as it eases the management of the application greatly.

Appendix & References

- Mirroring RHN with mrepo on RHEL6
- YUM repository using Redhat Satellite Network (RHN)
- Incompatibilities between createrepo-0.9.8-4.el6 and RHEL5: [Errno -3] Error performing checksum
- Mrepo and customising directory structure
- NYSE Case Study - Deploying Data Centers with Puppet
- Anaconda YUM repo documentation
- Foreman Template writing
- Pulp
- The Foreman
- Keiran Sweet @ Github
- Subversion
- HTTPry
- Extra Packages for Enterprise Linux (EPEL)
- The Art of system administration
- Unix philosophy
- The horrible state of Ruby in a production environment
- The rubygems_snapshot tool
- OMAPI API
- Smart-Proxy OMAPI API Support

About the Author

Keiran Sweet is a London based, Linux administrator who has a passion for automation and configuration management.

Comments, suggestions and bugs can be provided via email.

Special thanks to Alan Byrne from Cogmotive for assisting with this Paper.

Revision: 1.0, 29/4/2013