Red Hat Satellite 6.0 Core-SOE

Recommended Practices and Reference Architecture

Justin Sherrill Grant Gainey Todd Warner

Red Hat Satellite 6.0 Core-SOE

Recommended Practices and Reference Architecture

Justin Sherrill
Red Hat Engineering
jsherrill@redhat.com

Grant Gainey
Red Hat Engineering
ggainey@redhat.com

Todd Warner
Red Hat Product Management
taw@redhat.com

Abstract

Recommended Practices and Reference Architecture

Table of Contents

Chapter 1. Introduction

The purpose of this document is to describe recommended practices for building and maintaining **Standard Operating Environments (SOE)** using Red Hat Satellite, specifically the version 6.

This document has two major divisions: (1) an introduction and discussion of the process and (2) an example with enough detail to get you started both developing and managing your SOEs and introducing some automation.

Standard Operating Environments can span a number of use cases. For our purposes we are limiting our description to what we are calling the “Core-SOE”. The development of Core-SOE describes what many would call the development of Gold Images or Gold System Definitions. I.e., a small set of tested and standardized operating system definitions plus a limited layer of supporting technologies. These Core-SOEs are then used as the standard baseline platform for all the permutations of a company’s application deployments that will be layered on top.

These Core-SOEs generally are developed by a specialized team (depending on the resources of a company’s IT staff) and go through all the same, but separate, life cycle development processes as all other application deployments.

This document intends to describe in a generic way how one develops these Core-SOEs using Satellite version 6 and how they are maintained throughout a defined life cycle. This description serves as a recommended practice for not only Core-SOE development, but also a practice that extends to generalized Red Hat Satellite usage, and at a very high level, generalized Systems Management best practice.

1.1. Objectives

With this document you will learn how to:

1. Organize your team into roles
2. Define standard procedures for life cycle process
3. Customize Red Hat content (operating system releases) for your own standardization procedures
4. Add content (layered OS augmentations) to those “spins”
5. Organize configuration
6. Create system definitions and kickstart profiles connecting these content spins and configuration
7. Promote these definitions through your now standardized procedures of life cycle (e.g. DEV -> QA -> Production)
8. Automate everything.

1.2. Some Definitions

The docuemnt makes use of many terms which may be new to Satellite 5 customers. The glossary at the end of the document provides definitions for these terms.

Chapter 2. SOE Discussion

2.1. Top-level Organization

2.1.1. Organizational Structure

2.1.1.1. Multi-Org

If a company has multiple divisions, Red Hat Satellite allows systems, content and configuration to be siloed into multiple organizations. At the outset, admins need to determine whether they need to divide their Satellite into multiple organizations. For conceptual purposes, consider each organization as separately managed Satellites.

Many Satellite deployments have a single organization, but for the purposes of this document, descriptions will assume a deployment that includes two organizations: one that is tasked with maintaining an SOE, and one responsible for building an in-house application to be deployed as part of an SOE. All recommended practices can be applied to each organization as if each organization is a different Satellite with the additional benefit of being able to share content between organizations. Thus, the content of a Core-SOE is refined in the SOE organization and shared to the Application Development organization.

multi-Org or multi-Satellite

One may be tempted to use the multi-Org and multi-Satellite features to divide life cycle processes between organizations or even Satellites (a Development Org, a QA Satellite, whatever). Using the multi-org or multi-Satellite features is not advised for life cycle separation due to the restrictive nature of those features. For Core-SOE purposes, only content refinements can be shared, which is often sufficient (and then configuration and kickstart-profile portions of Core-SOE are duplicated) but adding life cycle workflow on top of that would become overly process heavy.

2.1.2. Managing your Admins

Often the application teams and Core-SOE teams are different groups of people. For best-practice purposes this arrangement is what we advocate. Even if the actual people are the same (common with smaller teams), different roles should be imagined when developing Core-SOEs and the applications layered on top. Best practice suggests that Core-SOE development is done separately and then once validated as Generally Available (GA'ed), that Core-SOE is utilized as a baseline for application development. If the Core-SOE has to be adjusted for the purposes of an application stack, some mingling of these processes makes sense, but ultimately, a Core-SOE is intended to be leveraged as a platform for multiple applications. The intention is to isolate complexity to the layer of focus: Perfect the SOE; once perfected, refine and perfect the application stack on top.

2.1.2.1. Administrator Roles

Red Hat Satellite 6 features a fine grained access control system that allows you to customize sets of roles to precisely the desired allowed actions. Each action on a resource type (Host, Host Collection, Lifecycle Environment, etc..) can be controlled within each Role and the individual objects can be restricted via a search query.

2.2. System Definitions and Life Cycle

Ultimately, a life cycle is a process of refinement through a path of “development”, “testing” and “production” (or “GA”). In the case of systems management, Red Hat Satellite 6 helps you do this via a process of managing Lifecycle Environments and Content Views. Content Views are snapshots of products and repositories and can be moved through the defined Lifecycle Environments.

2.2.1. Content Views

Content Views provide a way to snapshot content and customize it for a particular use case. In the example below, “Content View A” has one product with two of its repositories selected, as well as various filters created. When the content view is published to a new version, the selected content is cloned and the filters are applied.

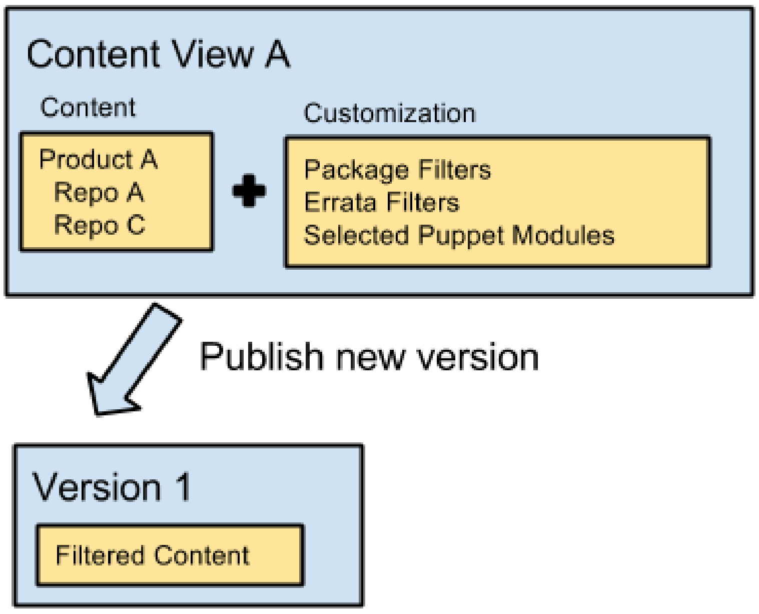

Figure 2.1. Content View

2.2.2. Moving content views through Lifecycle Environments

Initially publishing a content view only publishes the new version into the Library environment. New versions need to be promoted from one environment to the next. The purpose is to allow each version to go through the normal development, testing, and staging steps before making it into production.

After publishing a content view the first time, a Version 1 will exist in Library:

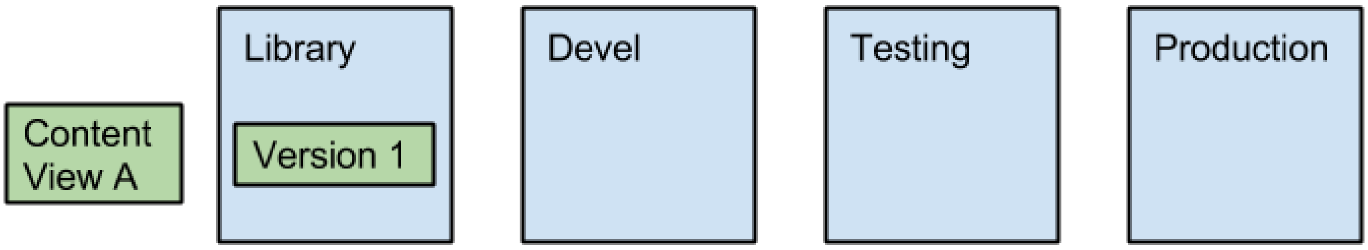

Figure 2.2. Initial Publish

In order to make that content available to Devel, Testing, and Production we will promote it first to Dev, then to Testing, and finally to Production, performing any verification or testing along the way:

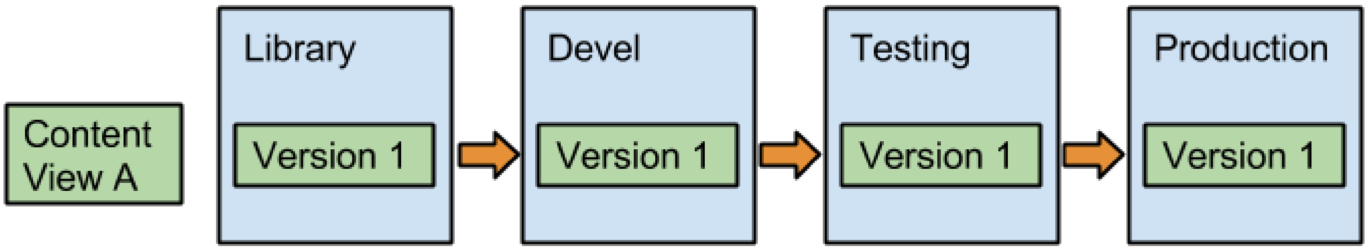

Figure 2.3. Initial Promotion

When new content is released in the upstream repositories that you want to make available in your content view, simply re-publish the content view to generate Version 2 in library:

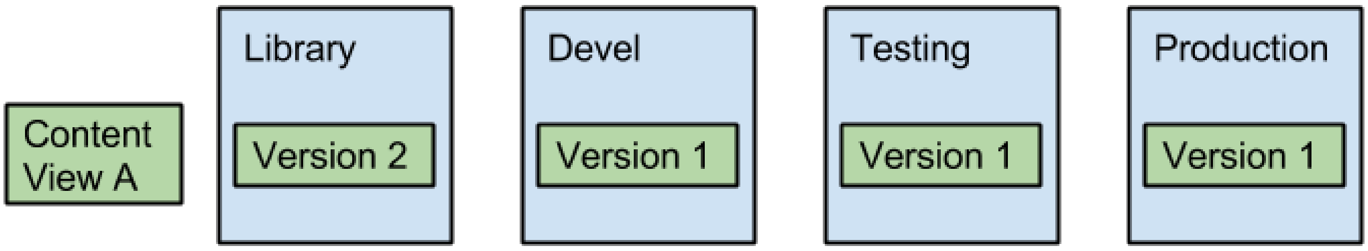

Figure 2.4. Version 2

Again to push Version 2 of the content view into Devel, simply promote it to the Devel Environment:

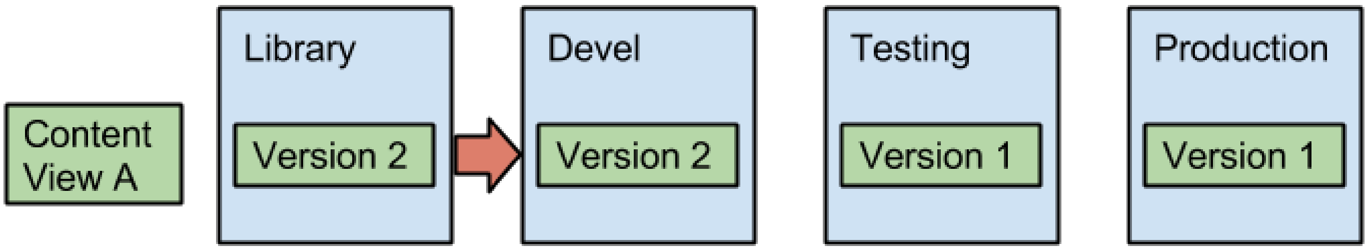

Figure 2.5. Second Promotion

In a more advanced scenario, we have two content views (A & B), and you can see how the different versions have made their way from Library to Devel to Testing and finally Production.

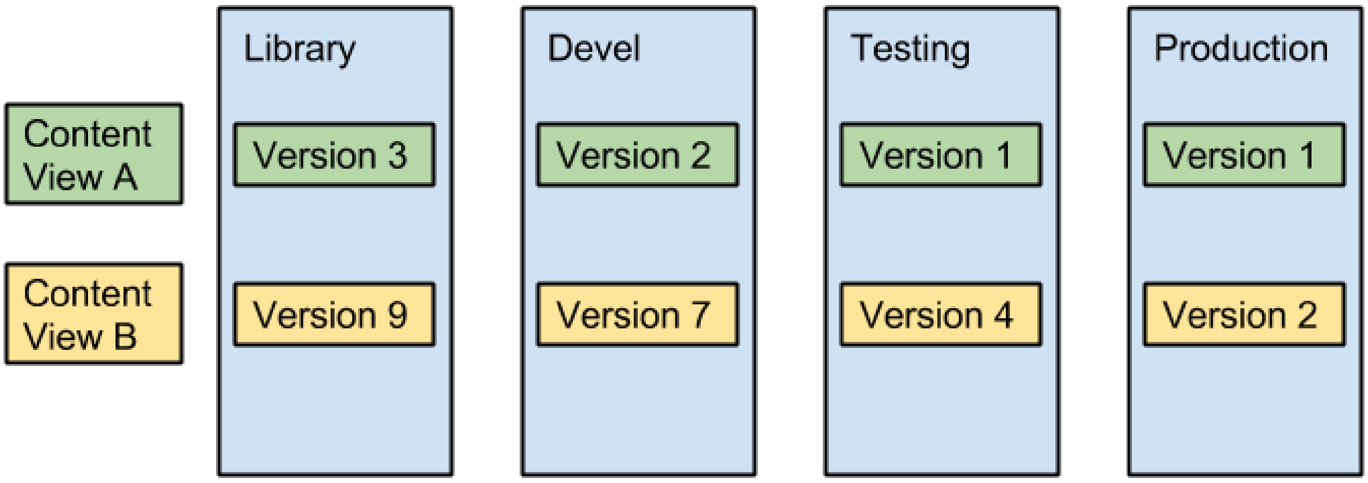

Figure 2.6. Advanced Views

2.2.3. “Spinning” Operating Systems

The first step towards a Standard Operating Environment is done by creating a customized base operating system. The best way to do this is to trim down the operating system to the bare essentials, and then slowly adding what is needed.

To do this, create a new Content View named “MySOE” and add the desired repositories from the “Red Hat Enterprise Linux Server” Product. At a minimum “Red Hat Enterprise Linux Server 6Server RPMS for x86_64” should be added (assuming x86_64 is the desired architecture). You would then create a filter associated to this repository and set it to only include packages from the “@core” and “@server-policy” package groups. To include additional packages simply add additional filters and include additional filters for more package groups or packages.

Some admins refer to this minimal system definition as a **JEOS (Just Enough Operating System)**.

2.2.4. Managing Custom Content

Just as Red Hat content is organized into Products and Repositories, custom content is too.

The process for uploading custom content involves:

1. Creating a custom product
2. Creating a custom repository
3. Either uploading content or syncing content from an external repository.

2.2.5. Managing Configuration

Configuration is handled in Satellite 6 via the use of parameterized Puppet classes and can be managed via Content Views in much the same way. A puppet class can define many different aspects of a system including:

- What packages need to be installed
- What services need to be running
- The structure and options of a configuration file

In order to upload :

1. Write parameterized classes
2. Test classes
3. Package into a puppet module
4. Upload to Satellite server
5. Add module to Content View and publish

Once your puppet modules are published within a Content View, they will be available in a Puppet Environment corresponding to the Lifecycle Environment/Content View. For example, a puppet module published to a content view (MyView) and promoted to Development, would be deployed to a puppet master within the “Development/MyView” puppet environment. This allows content and configuration to be bundled together through each of the Lifecycle Environments.

2.2.6. Kickstart Profiles and System Definitions

Kickstart is the underlying technology for provisioning client systems. They define how a system is constructed including what the partition table should look like, a minimum package set of what needs to be installed, and network configuration. In Satellite 5, a new kickstart would need to be created for each SOE.

In Satellite 6 Kickstart templates are designed to be much more generic and apply to all your hosts. Any customization should come by tying together puppet classes, Content Views, Lifecycle Environments, Activation Keys, and Partitioning Tables with a Host Group representing your SOE.

2.2.7. Host Collections and Automation

Taking action on systems already registered to Red Hat Satellite can be done to a single system or groups that are arbitrarily selected, or on already predefined groups of systems. Often it makes sense for an admin to create a defined Host Collection that defines some set of systems for permission or logical purposes.

Activation keys allow for systems to be added to Host Collections automatically at registration time, applying subscription in the process. Multiple activation keys can be chained together in order to layer specific elements of a bootstrapping process.

2.3. Conclusion

Hopefully this discussion gives you a solid introduction for why SOEs are useful and how Red Hat Satellite 6 can help you achieve a more standardized base platform for which to predictably layer applications on top of a customized operating system. We call this the Core-SOE. The process develops a much more disciplined approach to defining and managing end systems in a predictable, more secure, and more understood manner.

Chapter 3. Simple Example: Modeling a Standard Operating Environment

3.1. Overview

Most organizations have one or more SOEs. An organization may have different SOEs for different purposes.

For the purposes of this document, we will define one SOE based on a past version of Red Hat Enterprise Linux (RHEL) that the organization has standardized on.

3.1.1. SOE6.4 based on RHEL6.4

You may desire to create multiple SOEs based a different release such as Red Hat Enterprise Linux 5.10 or Red Hat Enterprise Linux 6.1. The same steps can be followed to create a similar SOE based on those older versions.

This SOE has the following goals: 1. a critical sudoers setting packaged in a sudoers puppet module will be deployed to each system 1. the vim_enhanced RPM will be installed by default 1. all packages starting with “emacs” will not be in the SOE

3.1.2. Using Hammer for CLI interactions

Satellite 6 ships with a command line interface called ‘hammer’. Hammer ships with default settings in /etc/hammer/ pointing to a Satellite server running localhost with a default username and password. You can configure hammer by modifying these config files, copying them to ~/.hammer, or copying them to any directory of your choice and adding the -c /path/to/files to specify the config file location. All of the hammer commands listed assume a proper configuration file with a valid username and password.

Within this document we will present steps on accomplishing each task using both the Web Interface as well as the Command Line Interface. At each step only one method is required.

3.2. Prep Work - Structuring the Satellite Server

3.2.1. Create Two Organizations

As a Satellite Administrator, create an organization and call it **Example_Org1**.

To create an organization:

1. Navigate to Administer > Organizations
2. Click “New Organization”
3. Specify a name and label
4. Click Submit.

An organization may also be created from from the command line:

```
hammer organization create --name=Example_Org1
```

In order to divide up subscriptions between these orgs, simply create additional entitlement distributors within the Red Hat Support Portal for each organization on your Satellite, download each manifest, and import it into the corresponding organization.

3.2.2. Create Three Lifecycle Environments in Example_Org1

To facilitate lifecycle management, we will create three Lifecycle environments:

1. Navigate to Content > Lifecycle Environments
2. Click “+” Beside the "Library" environment
3. Enter “Devel” for the name.
4. Click submit
5. Click the “+” next to Devel
6. Enter “Testing” for the name and click submit
7. Click the “+” next to Testing
8. Enter “Production” for the name and click submit

From the cli:

```
hammer lifecycle-environment create --organization Example_Org1 --name=Devel --
prior=Library

hammer lifecycle-environment create --organization Example_Org1  --name=Testing -
-prior=Devel

hammer lifecycle-environment create --organization Example_Org1 --name=Production
--prior=Testing
```

3.2.3. Activate Your Red Hat Satellite

Download your Subscription Manifest from the Red Hat Customer Support portal. Upon uploading the manifest to your Satellite, Red Hat Products and Repositories will be available to sync and subscriptions should be made available for systems to consume.

To upload your manifest, simply login to your Satellite server and:

1. navigate to Content > Subscriptions > Red Hat Subscriptions
2. Click “Import Manifest” at the top right
3. Browse to locate your manifest and click upload

From the cli:

```
hammer subscription upload  --organization=Example_Org1 --
file=/path/to/manifest.zip
```

3.3. Creating the initial version with Red Hat Content

3.3.1. Enable desired RHEL repositories

Within each organization you will want to enable each desired RHEL repository. This allows you to focus only on the repositories important to your organization.

1. Navigate to Content > Repositories > Red Hat Repositories 2. Expand the “Red Hat Enterprise Linux Server” Product 3. Select the “Red Hat Enterprise Linux 6 Server (RPMs)” repository set underneath the product. (This may take a few moments). 4. Select the desired repositories underneath the repository set. It is recommend for most users to select “Red Hat Enterprise Linux 6 Server RPMs x86_64 6Server”

In order to kickstart systems from the Satellite server, a kickstart tree needs to be synced. On the same page:

1. Select the “Red Hat Enterprise Linux 6 Server (Kickstart)” repository set underneath the same product as above.

2. Select the desired kickstart tree “Red Hat Enterprise Linux 6 Server Kickstart x86_64 6.4”

To perform the same actions from the command line:

```
hammer repository-set enable --product='Red Hat Enterprise Linux Server' --
organization=Example_Org1 --name='Red Hat Enterprise Linux 6 Server (RPMs)' --
releasever=6Server --basearch=x86_64

hammer repository-set enable --product='Red Hat Enterprise Linux Server' --
organization=Example_Org1 --name='Red Hat Enterprise Linux 6 Server (Kickstart)' -
-releasever=6.4 --basearch=x86_64
```

3.3.2. Syncing Content

For Red Hat Repositories or any custom Repositories with an external url defined, simply: 1. Navigate to Content > Sync Management > Sync Status 2. Expand the desired product 3. Select the desired repository 4. Click “Synchronize Now”

It is recommended to sync:

Red Hat Enterprise Linux 6 Server Kickstart x86_64 6.4 Red Hat Enterprise Linux 6 Server RPMs x86_64 6Server

From the cli:

```
hammer repository synchronize --name='Red Hat Enterprise Linux 6 Server Kickstart
x86_64 6.4' --organization=Example_Org1

hammer repository synchronize --name='Red Hat Enterprise Linux 6 Server RPMs
x86_64 6Server' --organization=Example_Org1
```

Async Flag

You may want to use the --async flag and run them both together.

3.3.3. Create a content view

Next create a content view:

1. Navigate to Content > Content Views
2. Click “Create New View”
3. Enter “SOE-OS” for the name
4. Click “Save”.

From the CLI:

```
hammer content-view create --organization Example_Org1 --name=SOE-OS
```

3.3.4. Add Repositories

We need to specify which repositories we want within our Content View.

1. Navigate to Content > Content Views
2. Click on “SOE-OS”
3. Click “Content” and in the submenu click “Repositories”
4. Add the following repositories to the Content View:
5. Red Hat Enterprise Linux 6 Server Kickstart x86_64 6.4
6. Red Hat Enterprise Linux 6 Server RPMs x86_64 6Server

From the CLI:

```
hammer content-view add-repository --organization  Example_Org1 --repository 'Red
Hat Enterprise Linux 6 Server Kickstart x86_64 6.4' --name=SOE-OS

hammer content-view add-repository --organization  Example_Org1 --repository 'Red
Hat Enterprise Linux 6 Server RPMs x86_64 6Server' --name=SOE-OS
```

3.3.5. Add Filters

Typically we want to control package content going into our content view by errata date. This allows us to control which errata make it into our content view regardless of what is in Library.

First we want create a filter with all the packages with no errata:

1. Within our Content View click “Content” and in the submenu, click “Filters”
2. Click “New Filter”
3. Give it the name “Original Packages”
4. Set the Content Type to ‘Package‘.
5. Change the type to “Include”
6. Click “Save”.
7. Check the checkbox beside the text “Include packages that contain no errata”

From the CLI:

```
hammer content-view filter create --organization=Example_Org1  --content-
view=SOE-OS  --type=rpm --name='Original Packages'  --inclusion=true  --original-
packages=true
```

Next we'll create a filter to include all errata up to a specific date:

1. Within our content view click Filters
2. Click "New Filter"
3. Give it the name "Restrict Errata"
4. Set the Content Type to 'Errata by date and type'
5. Change the type to "Exclude"
6. Click "Save".
7. Set the start date to last day you want errata. For RHEL 6.4 that is 2013-11-20.

From the CLI:

```
hammer content-view filter create --organization=Example_Org1  --content-
view=SOE-OS  --type=erratum --name='Restrict Errata'  --inclusion=false

hammer content-view filter rule create --content-view=SOE-OS  --content-view-
filter='Restrict Errata' --organization=Example_Org1  --start-date=2013-11-20
```

Next we want to blacklist all packages that start with 'emacs'.

1. Within our content view click Filters
2. Click "New Filter"
3. Give it the name "Exclude Emacs"
4. Set the Content Type to "Package"
5. Change the type to "Exclude"
6. Click "Save".
7. Under Package Name, enter "emacs*" and click "+ Add"

From the CLI:

```
hammer content-view filter create --organization=Example_Org1  --content-
view=SOE-OS  --type=rpm --name='Exclude Emacs'  --inclusion=false

hammer content-view filter rule create --content-view=SOE-OS  --content-view-
filter='Exclude Emacs'  --organization=Example_Org1  --name='emacs*'
```

3.3.6. Publish Content View

To Publish the Content View:

1. Click 'Publish New Version'

2. In the upper right hand corner and click “Save”. This will generate “Version 1” of the content view.
3. With these two Repositories added, the publish process should take less than 5 minutes.

From the CLI:

```
hammer content-view publish --name=SOE-OS --organization=Example_Org1
```

Wait Until Sync is Done

You should not publish the initial version of the Content View until after all associated Repositories have finished syncing.

3.4. Adding Custom Configuration

Now that we have a snapshot containing Red Hat content, lets add some configuration to set up a sudoers file and require that ‘vim-enhanced’ is installed. For this example we have two pre-built puppet modules named acme_corp-sudo-1.0.tar.gz & acme_corp-vim-1.0.tar.gz.

acme_corp-sudo-1.0.tar.gz requires that sudo is installed and its custom configuration file is deployed

acme_corp-vim-1.0.tar.gz requires that vim-enhanced is installed.

Some users may already have Puppet modules deployed within a git repository. Satellite 6 has the capability of building and importing Puppet modules from git or any simple directory of un-tarred modules. In this section, we discuss uploading a puppet module to a repository.

3.4.1. Creating a repository for Puppet modules

First we need to create a Product for our Puppet modules:

Navigate to Content > Products > New Product

Then add a new product:

1. For name, enter ‘Custom Configuration’
2. Click “Save”.

From the cli:

```
hammer product create --organization=Example_Org1 --name="Custom Configuration"
```

Next create the repository:

Navigate to Content > Products > Click on the “Custom Configuration” product. Click “Create Repository” on the left.

And then add a custom repository:

1. Click ‘New Repository’ in the upper right hand corner.
2. Fill out a name and label (we will use ‘Custom Configuration’ as the name).

3. Select “Puppet” as the type.

4. Click “Save”.

From the CLI:

```
hammer repository create  --organization=Example_Org1 --product='Custom
Configuration' --name='Puppet Modules' --content-type=puppet
```

3.4.2. Uploading Puppet Modules

To upload our custom puppet module simply do one of the following:

To upload a puppet module using the satellite cli:

```
hammer repository upload-content  --name='Puppet Modules'  --path=~/puppetlabs-
firewall-1.0.2.tar.gz  --organization=Example_Org1
--product='Custom Configuration' --path can point to a single file, or a directory
of puppet modules
```

To upload a puppet module using your web browser visit the “Custom Configuration” Repository details page and select the upload file dialogue on the right.

3.4.3. Importing Puppet Modules from a GIT repo

Satellite 6 ships a utility called 'pulp-puppet-module-builder' from the pulp-puppet-tools rpm. This utility is already installed on your Satellite 6 Server but can be installed on another machine if desired. By running this tool against a GIT repository, it will checkout the repository, build all of the modules contained, and publish them in a structure that Satellite 6 can syncronize against.

One common method is to run the utility on the satellite itself and publish to a localfile system directory and sync against that directory.

For example:

1. mkdir /modules

2. chmod 755 /modules

3. pulp-puppet-module-builder --output-dir=/modules --url=git@mygitserver.com:mymodules.git --branch=develop

this would checkout the 'develop' branch of the GIT repository located at 'git@mygitserver.com:mymodules.git' and publish them to /modules. Then from within Satellite 6 simply set the url field on your Puppet Repository to 'file:///modules'. You can then sync it just like any other Repository.

If you are running this on a remote machine you can publish these modules to an HTTP server such as apache and have satellite sync them there. For example:

1. mkdir /var/www/html/modules/

2. chmod 755 /var/www/html/modules/

3. pulp-puppet-module-builder --output-dir=/var/www/html/modules/ --url=git@mygitserver.com:mymodules.git --branch=develop

and then in Satellite 6 set the Repository's url to 'http://HOSTNAME/modules/' and sync it normally.

3.4.4. Adding our puppet module to the Content View

Once we have a repository in Library with puppet modules populated we can add these modules to our content view. You may lock a content view to a particular version or simply have Satellite 6 pull in the newest version available every time you publish a Content View.

To add a puppet module to the content view:

1. Navigate to Content Views > Select our View (SOE-OS) > Puppet Modules.
2. Click the “+ Add New Module” button at the top right.
3. You will be presented with the available puppet module list.
4. Find the puppet module name you want (for this example we’ll use “sudo”), and then click “Select a Version”
5. The list of available puppet module versions are listed by author. If you wish a specific version of a module to be used select the “Select Version” button beside the the desired version. Alternatively, to always have the latest version used by a particular author, find the row with “Latest Version” on the author you desire and select the “Select Version” button. For this example we’ll select “Latest Version” beside the “acme_corp” author.

Repeat steps 2 through 5 for our acme_corp-vim module.

From the CLI:

```
hammer content-view puppet-module add --author=acme_corp --name=sudo  --content-
view=SOE-OS --organization=Example_Org1

hammer content-view puppet-module add --author=acme_corp--name=vim  --content-
view=SOE-OS --organization=Example_Org1
```

3.4.5. Publish Version Two

Assuming you are already within the SOE-OS Content View, click “Publish New Version” in the upper right hand part of the screen.

From the CLI:

```
hammer content-view publish --name=SOE-OS --organization=Example_Org1
```

We now have a published version of our SOE with our customized operating system and a defined configuration.

Now that we have a version of our Content View with Red Hat Content, Custom Content, and Custom Configuration we can Promote our View to the “Devel” Lifecycle Environment and finalize our SOE.

3.4.6. Promoting a Content View

To promote the content view to the Devel Lifecycle Environment, simply:

1. Click "Promote" beside “Version 2”.

2. Select the Devel environment.

3. Click “Promote Version”.

From the CLI:

```
hammer content-view version promote --version=2 --environment=Devel
```

3.4.7. Creating an Activation Key

An activation key in Satellite 6 is identical in concept to an activation key in Satellite 5. You will want to create an activation key for each lifecycle environment.

To create an activation key:

1. Navigate to Content > Activation Keys.

2. Click “New Activation Key” in the upper right hand corner

3. Give the name of the activation key “SOE-OS_Devel”

4. Select the “Devel” environment.

5. Select the “SOE-OS” Content View.

6. Click Save.

7. Click “Subscriptions” and then click “Add”

8. Select the desired Red Hat subscription which includes access to the “Red Hat Enterprise LInux Server” Product

9. Click “Add Selected”

From the CLI:

```
hammer activation-key create --name=SOE-OS_Devel  --organization=Example_Org1 --
content-view=SOE-OS --lifecycle-environment=Devel
```

List available subscriptions:

hammer subscription list --organization='Example_Org1'

```
  ----------------------------------------|----------|---------|--------------|-
-------------
  NAME                                    | CONTRACT | ACCOUNT | SUPPORT      |
ID
  ----------------------------------------|----------|---------|--------------|-
-------------
  Red Hat Enterprise Linux               | 10169839 | 1212729 | SELF-SUPPORT |
ff808081
  ----------------------------------------|----------|---------|--------------|-
-------------
```

And then add the selected subscription:

```
hammer activation-key add-subscription --id=1 --subscription-id=ff808081
```

3.5. Provisioning Configuration

Before continuing to provision your SOE, a few things need to be configured.

To Kickstart a Host we will use the bootdisk feature of Satellite 6. Currently the Bootdisk has a few modes of operating:

- Static networking
- Using DHCP where you know the IP Address the host's MAC address will receive ahead of time

In both of these cases, you need to know the MAC address of the new Host ahead of time. If you are in an environment where you do not know this, you may want investigate the Discovery feature of Satellite 6.

If you do not know the IP address of the new host prior to provisioning, see this Bugzilla for a workaround:

https://bugzilla.redhat.com/show_bug.cgi?id=1120762

3.5.1. Associate iPXE Template

The Satellite 6 Bootdisk uses iPXE to simulate a PXE environment and boot the machine. We need to configure the iPXE templates for our specific Operating System.

1. Navigate to Hosts > Provisioning Templates.
2. Select "Kickstart default iPXE".
3. Select the "Association Tab".
4. Select "RHEL Server 6.4" in the "Applicable Operating Systems" table.
5. Click 'Submit'.
6. Navigate to Hosts > Operating Systems.
7. Select "RHEL Server 6.4".
8. Select the "Templates" tab.
9. In the iPXE dropdown choose "Kickstart default iPXE".

From the CLI:

First lookup the operating system ID:

```
hammer os list
```

And then associate the OS with the correct template (replacing n with the correct operating system id):

```
hammer template add-operatingsystem --name='Kickstart default iPXE' --
operatingsystem-id=2
hammer  os add-config-template --id=n --config-template="Kickstart default iPXE"
```

3.5.2. Creating a Domain

A domain is a literal network domain such as example.com where hosts will live. For example, a set of hosts 'webserver' and 'dbserver' may both exist in domain example.com. Their fully qualified domain names would be webserver.example.com and dbserver.example.com respectively.

In order to provision a host, we'll need to create a domain for it:

1. Navigate to: Infrastructure > Domains > New Domain
2. Create a new Domain with name: 'example.com' (or whatever your valid domain is)
3. Select the "Default Location".
4. Click Submit.

From the CLI:

```
hammer domain create --name=example.com
hammer organization add-domain --domain=example.com --name='Example_Org1'
hammer location add-domain --domain=example.com --name='Default Location'
```

From this point forward when referencing domains we will refer to example.com, but you should substitute your relevant domain.

3.5.3. Creating a Subnet

A Subnet in Satellite 6 is the definition of your actual network subnet. To create one, navigate to:

Infrastructure > Subnets > New Subnet

Create a new Subnet with:

Name: 'mysubnet' Network Address: 192.168.0.0 Network Mask: 255.255.255.0

Replace the example values given here with the actual values from the network you are doing your provisioning on. Feel free to fill out any addition information for this particular subnet, but for this exercise these are the only required options.

1. Click Submit.
2. Click on the newly created subnet 'mysubnet'.
3. Click the 'Domains' tab.
4. Selet the Domain we created in the previous step (example.com)
5. Select the 'Locations' tab.
6. Select the 'Default Location'.
7. Click Submit.

From the CLI:

```
hammer subnet create --name=mysubnet --network=192.168.0.0 --mask=255.255.255.0
hammer organization add-subnet --subnet=mysubnet --name='Example_Org1'
hammer location add-subnet --subnet=mysubnet --name='Default Location'
```

3.5.4. Associating the Partition Table with the Operating System

Navigate to Hosts > Operating Systems > Select “RHEL 6.4” > Select “Partition Table”

Check “Kickstart default” Click “Submit”

From the CLI:

```
hammer os list

---|------------|--------------|-------
ID | FULL NAME  | RELEASE NAME | FAMILY
---|------------|--------------|-------
1  | Redhat 6.4 |              | Redhat
---|------------|--------------|-------
```

Then add the partition table:

```
hammer os add-ptable  --id=1 --ptable="Kickstart default"
```

3.5.5. Creating a Host Group

Host Groups are a sort of template that bring together a kickstart template, puppet classes, content view, and environment to build an SOE.

To create a host group:

1. Navigate to:

Configure > Host Groups

2. Create a new Host Group with:

```
name: “SOE-Devel”
Environment: Devel
Content View: SOE-OS
Puppet CA: select the fqdn of your puppet CA (usually the same as your Satellite
server)
Puppet Master: select the fqdn of your puppet master (usually the same as your
Satellite server)
```

3. On the Puppet Classes tab, select the appropriate classes from your puppet module
4. On The Operating Systems tab:
5. select “x86_64” for the architecture,
6. Select “RedHat 6.4” for the Operating System,
7. Enter a default root password
8. Click “Resolve Templates”
    a. The default provision template “Katello Kickstart Default for RHEL” should be selected.
9. Under the Network tab, select the appropriate Domain.
10. Under Activation Key, enter the name of the key we created: “SOE-OS_Devel”.

11. Click “Save”

3.5.6. Host Group and Activation Key Interaction

Currently when creating a host group you can select the desired Content View and Lifecycle Environment. You also can select this as part of the activation key. These selections on each object perform different functions and will likely be unified in future.

The Content View and Lifecycle Environment

- On the Host Group:

1. Used to determine which Puppet Environment is used to consume puppet classes from

    a. Used to determine what kickstart url to use upon provisioning

- On the Activation Key:

2. At registration time, determines which Lifecycle Environment and Content View the software content the client is using will come from.

3.5.7. Kickstart Systems to Match SOEs Exactly

1. Navigate to Hosts > All Hosts
2. Click “New Host” in the upper right hand corner
3. Enter the host name under “Name”.
4. Select the Host Group SEO-Devel.
5. Under the Network Tab, enter the MAC address of the system to provision.
6. Select the subnet that was created.
7. Type in the IP address the host will have.
8. Click “Submit”.

We now have a Host defined within foreman. Next, we will download the bootdisk and boot our system with it.

1. On this specific host's page, click “Boot Disk” in the upper right.
2. If you are using static networking select 'Download bootdisk for host "host.example.com". If you are using dhcp click “Download Generic Boot Disk”.
3. Either:

    a. Boot a VM with the downloaded Boot Disk iso

    b. Burn the Boot disk to a cd/USB disk and boot the specified system with the boot disk

    c. To copy it to a USB disk, run: 'dd if=filename.iso of=/dev/sdb' where sdb is your USB disk.

The system should boot, install, and configure itself.

If Satellite 6 has been configured to control the PXE (tftp) server and the client in question is set to boot from the pxe server, all that is needed is to reboot the client system the bootdisk is not needed.

3.5.8. Validate Kickstarted SOE Systems

Once your systems have been kickstarted, you can see that your SOE is now in place.

1. Validate the version of RHEL that is installed:

```
# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 6.4
```

2. Validate the correct channels are subscribed to:

```
# yum repolist
```

3. Validate the puppet module you uploaded has been applied. Check /etc/sudoers to verify your changes are in place.

4. Validate that vim-enhanced is installed.

At this point, your Devel SOE machines are ready for experimentation and tweaking. Your development team should document any changes that are needed, and modifications should be made to your content view filters and puppet modules. Once the development team has agreed upon the final version, it's time to migrate the Devel SOE to the Test SOE, for use by your Quality Engineering team!

3.6. Promote your Standard Operating Environments to Testing

3.6.1. Promote Content View To Testing

To promote the SOE-OS content view navigate to Content > Content Views > Click on “SOE-OS” > Click on “Versions” 1. Click “Promote” next to “Version 2” 2. Select the “Testing” environment 3. Click “Submit”

From the CLI:

```
hammer content-view version promote --version=2 --environment=Testing
```

3.7. Create New Activation Key

Navigate to Content > Subscriptions > Activation Keys

1. Click New Key in the upper right hand corner

2. Give the name of the activation key “SOE-OS_Test”

3. Select the “Test” environment

4. Select the “SOE-OS” Content View

5. Click Save

6. Click “Subscriptions” and then click “Available”

7. Select the desired Red Hat subscription which includes access to the “Red Hat Enterprise LInux Server” Product

8. Click “Add to Activation Key”

Or via the cli:

```
hammer activation-key create --name=SOE-OS_Test --organization=Example_Org1 --
content-view=SOE-OS --lifecycle-environment=Test
```

3.7.1. Create New Host Group

Navigate to:

Configure > Host Groups

Create a new Host Group with:

name: “SOE-OS_Testing” Environment: Test Content View: SOE-OS Puppet CA: select the fqdn of your puppet CA (usually the same as your Satellite server) Puppet Master: select the fqdn of your puppet master (usually the same as your Satellite server)

On the Puppet Classes tab, select the appropriate classes from your puppet module

On The Operating Systems tab:

1. select “x86_64” for the architecture,
2. Select “RedHat 6.4” for the Operating System,
3. Enter a default root password
4. Click “Resolve Templates”
    a. The default provision template “Katello Kickstart Default for RHEL” should be selected.

Under the Network tab, select the appropriate Domain

Under Activation Key, enter the name of the key we created: “SOE-OS_Test”.

Click Save

Production?

We leave promotion of the Standard Operating Environments to Production as an exercise for the reader.

Chapter 4. Another Example: The Layered Application

4.1. Assumption

The example application is of architecture type “noarch” and exists for each variety OS and is already packaged in an RPM that has a dependency set to require the httpd package. The example applications will be named **myApp-1.0.el6.noarch.rpm**.

4.2. Managing Layered Content

In order to upload custom content (RPMs), first create a custom product:

1. Navigate to Content > Repositories > Products > New Product (top right)
2. Fill out name and label. We’ll use the name “Custom Application”.
3. Click ‘create’

or via the CLI:

```
hammer product create --organization=Example_Org1 --name='Custom Application'
```

And then add a custom repository:

1. Click ‘New Repository’ in the upper right hand corner
2. Fill out a name and label (we will use ‘Custom Repository’).
3. Select “Yum” as the type.

or via the CLI:

```
hammer repository create --organization=Example_Org1 --product='Custom
Application' --name='Custom Repository'  --content-type=yum
```

4.3. Uploading Layered Content

Adding your RPM to the custom repository:

- To upload an rpm simply use the katello cli:

```
hammer repository upload-content  --name='Custom Repository'  --
path=~/filename.rpm  --organization=Example_Org1
```

- To upload an entire directory of rpms use:

```
hammer repository upload-content  --name='Custom Repository'  --
path=~/path/to/directory/  --organization=Example_Org1
```

While not required rpms should be digitally signed with your own custom GPG key as well. If you take this additional step to secure your packages, the GPG key also needs to be uploaded to the Satellite server and associated to the corresponding repository.

4.4. Syncing Layered Content

If your custom application is hosted in a yum repository already, simply add a URL such as “http://server.domain.com/path/to/yum_repo” when creating the repository in the UI or update its details later on.

Or via CLI:

```
 hammer repository update --url=http://linuxdownload.adobe.com/linux/x86_64/ --
organization=Example_Org1 --name='Custom Repository'
```

Then navigate to Content > Sync Status, select the repository and click ‘Sync’.

Or via CLI:

```
hammer repository synchronize --organization=Example_Org1 --product='Custom
Application' --name='Custom Repository'
```

4.5. Adding the custom repository to our Content View

To include the custom repository to the content view, navigate to Content Views > Click on our Content View “SOE-OS” > Select “Content” > Select “Repositories” > Select the “Add” tab

Select your custom repository and click “Add Selected”.

Or via CLI:

```
hammer content-view add-repository --organization  Example_Org1 --repository
'Custom Repository’  --name=SOE-OS
```

4.6. Publishing the third Content View version

Click ‘Publish’ in the upper right hand corner. This will generate “Version 3” of the content view.

From the CLI:

```
hammer content-view publish --name=SOE-OS --organization=Example_Org1
```

4.7. Promoting to Devel

To promote the our SOE-OS content view navigate to Content > Content Views > Click on “SOE-OS” > Click on “Versions” 1. Click “Promote” next to “Version 3” 2. Select the “Devel” environment 3. Click “Submit”

From the CLI:

```
hammer content-view version promote --version=3 --environment=Devel
```

4.8. Using Composite Content Views

Instead of bundling your layered application with the SOE-OS, you may wish to use Composite Content Views to bundle your layered application with an SOE-OS Content View. For more information about Composite Views, see the Satellite 6 Users Guide.

Chapter 5. Glossary

The following terms are used throughout this document, and are important for the users understanding of Satellite 6.

Activation Key

A registration token which can be used in a kickstart file to control actions at registration. These are similar to Activation Keys in Spacewalk, but they provide a subset of features because after registration, Puppet takes control of package and configuration management.

Application Lifecycle Environment

Steps in a promotion path through the Software (Development) Life Cycle (SDLC). Content (packages, puppet modules) can be moved through lifecycle environments via content view publishing/promotion. Traditionally these environments are things like Development -> Test -> Production. Channel cloning was used to implement this concept for this in Spacewalk.

Attach

Associating a Subscription to a Host which provides access to RPM content.

Capsule

An additional "server" that can be used in a Satellite 6 deployment to facilitate content federation and distribution in addition to other localized services (Puppet master, DHCP, DNS, TFTP, and more).

Compute Profile

Default attributes for new virtual machines on a compute resource.

Compute Resource

A virtual fabric, or cloud infrastructure, where hosts can be deployed by Satellite 6. Examples include RHEV-M, OpenStack, EC2, and VMWare.

Content

Software packages (RPMS), Package Groups, Errata, and Puppet modules. These are synced into the Library and then promoted into Lifecycle Environments via Content Views in order to be used/consumed by Hosts.

Content Delivery Network (CDN)

The mechanism to deliver Red Hat content in a geographically co-located fashion. For example, content which is synced by a Satellite 6 in Europe will pull content from a source in Europe.

Content View

A definition of content that combines products, packages, errata and Puppet modules, with capabilities for intelligent filtering and snapshotting. Content Views are a refinement of the combination of channels and cloning from Spacewalk.

External Node Classifier

A Puppet construct that provides additional data for a Puppet master to be used for configuring Hosts. Foreman acts as an External Node Classifier to Puppet Masters in a Satellite deployment.

Facter

A program that provides information (facts) about the system on which it is run (eg: total memory, operating system version, architecture, etc.) Facter facts can be used in Puppet modules in order to enable specific configurations based on Host data.

Hammer

The command line tool for Satellite 6. Hammer can be used as a standard cli (and used in scripts) and can also be used as a shell in the same way that spacecmd, virsh and others work.

Host

A system, either physical or virtual, which is managed by Satellite 6.

Host Group

A template for how a Host should be built. This includes the content view (which defines the available RPMs and Puppet modules), and the Puppet classes to apply (which determines the ultimate software and configuration).

Location

A collection of default settings which represent a physical place. These can be nested so that a user can set up defaults, for example, for Europe, which are refined by Tel Aviv, which are refined by DataCenter East, and then finally by Rack 22.

Library

The Library is the single origin of all content which can be used. If you are an Information Technology Infrastructure Library (ITIL) shop, it is your definitive media library.

Manifest

The means of transferring subscriptions from a Subscription Provider (such as the Red Hat Customer portal) to Satellite 6. This is similar in function to certificates used with Spacewalk.

Organization

A tenant in Satellite 6. Organizations, or orgs, are isolated collections of hosts, content and other functionality within a Satellite 6 deployment.

Permission

The ability to perform an action.

Product

A collection of content repositories.

Promote

The act of moving content from one Application Lifecycle Environment to another.

Provisioning Template

User defined templates for Kickstarts, snippets and other provisioning actions. These provide similar functionality to Kickstart Profiles and Snippets in Satellite 6.

Puppet Agent

An agent that runs on a Host that applies configuration changes to that Host.

Puppet Class

A Puppet Class is re-usable named block of puppet manifest, similar to a class in an object-oriented programming language. Puppet classes must be included/instantiated in order to use their functionality. Puppet Classes can be parameterized - they can take parameters when they are included/instantiated and those parameters may be used by the underlying manifest to affect the ultimate configuration.

Puppet Manifest

A Manifest is a simple set of Puppet instructions. Manifests typically have the .pp extension. A manifest is much like a procedure in programming terms.

Puppet Master

A Capsule component that provides Puppet manifests to Hosts for execution by the Puppet Agent.

Puppet Module

A Puppet Module is a set of Puppet manifests/classes, template files, tests and other components packaged together in a specific directory format. Puppet Modules are typically associated with specific software (eg ~ NTP, Apache, etc) and contain various classes used to assist in the installation and configuration of that software. Puppet Labs maintains a repository of official and user-contributed modules called the Puppet Forge.

Pulp Node

A Capsule component that mirrors content. This is similar to the Spacewalk Proxy in Spacewalk. The main difference is that content can be pre-staged on the Pulp Node before it is used by a Host.

Repository

A collection of content (yum repository, puppet repository).

Role

A collection of permissions that are applied to a set of resources (such as Hosts).

Smart Proxy

A Capsule component that can integrate with external services, such as DNS or DHCP.

Smart Variable

A configuration value that controls how a Puppet Class behaves. This can be set on a Host, a Host Group, an Organization, or a Location.

Standard Operating Environment (SOE)

A controlled version of the operating system on which applications are deployed

A controlled version of the operating system on which applications are deployed.

Subscription

The right to receive content and service from Red Hat. This is purchased by customers.

Syncing

Mirroring content from external resources into an organization's Library.

Sync Plans

Scheduled execution of syncing content.

Usergroup

A collection of roles which can be assigned to a collection of users. This is similar to the Role in Spacewalk.

User

A human who works in Satellite 6. Authentication and authorization can be done via built in logic, or using external LDAP or kerberos resources.