Red Hat Satellite 6.4

Transitioning from Red Hat Satellite 5 to Red Hat Satellite 6

Supporting transition from Satellite 5 to Satellite 6

Last Updated: 2018-10-03

Red Hat Satellite 6.4 Transitioning from Red Hat Satellite 5 to Red Hat Satellite 6

Supporting transition from Satellite 5 to Satellite 6

Red Hat Satellite Documentation Team
satellite-doc-list@redhat.com

Legal Notice

Abstract

This document describes how to perform a transition of an existing Satellite 5.6 or 5.7 Server to a new Satellite 6 Server. It describes the necessary preparations and prerequisites, the bootstrap script, which is used for the migration of clients, and how to retire the old Satellite 5.6 or 5.7 deployment after the transition is finished.

Table of Contents

CHAPTER 1. INTRODUCTION

Satellite enables you to perform the transition from Satellite 5 Server to Satellite 6 Server. There is a significant difference between Satellite 5 and Satellite 6. Therefore, the in-place upgrade process (such as from version 4.x to 5.x) does not apply for version 5 to 6. You need to install Satellite 6 on a new server and migrate clients to the new Satellite 6 instance. Consequently, this is referred to as a transition process and not an upgrade process. Red Hat gives customers a whole year of duplicate Satellite subscriptions so that you can build and test a Satellite 6 Server before migrating **only** the Satellite 5 clients to the new system.

Transitioning Workflow

The transition process from Satellite 5 to Satellite 6 consists of the following steps:

1. Review the documentation, learn and understand the basics of the Satellite 6 Product. If you have Satellite 5.8 running on a s390 system, you must build a fresh Satellite 6 on an x86_64 architecture system. Satellite 6 cannot run on s390 systems.

2. Install Satellite 6 on a new machine, activate it with a manifest and synchronize Red Hat content from CDN.

3. Install the bootstrap script on client systems.

4. Make sure that you meet the prerequisites and run the bootstrap script to migrate your clients.

Supported Versions of Red Hat Enterprise Linux

Unlike Satellite 5, Satellite 6 does not support clients run on Red Hat Enterprise Linux 4 and older. Ensure all your clients are configured on one of the following Red Hat Enterprise Linux versions before you start the transition:

- Red Hat Enterprise Linux 5.7 and later.
- Red Hat Enterprise Linux 6.1 and later.
- Red Hat Enterprise Linux 7.0 and later.

CHAPTER 2. COMPARING SATELLITE 5 AND SATELLITE 6

2.1. DESIGN AND CONCEPTS

2.1.1. Red Hat Satellite 5

Red Hat Satellite 5 is life cycle management tool that includes the ability to deploy, manage and monitor a large number of systems. Satellite 5 can be set up in a connected or disconnected mode in which Red Hat software is distributed to client systems using the original pooled subscription approach. The pooled subscription concept is similar to the way in which clients consume entitlements from Red Hat Network Classic.

Features and Functionality

The popular functionality of Satellite 5 includes the ability to provision a large number of systems using kickstart files and activation keys to install and configure systems to a predictable state. This provisioning process associates systems to designated organizations, software and configuration channels, as well as placing systems in predefined system groups. The Satellite 5 provisioning functionality enables administrators to provision thousands of systems in a consistent manner.

Another popular feature is the ability to manage software and configuration files across large numbers of systems in local or remote environments after those systems have been provisioned. One of the well understood concepts of managing software and configuration files in Satellite 5 is the concept of channels. All software and configuration is managed and distributed through channels, and any client needing access to software or configuration content needs to be associated with one or more relevant channels. Further, the ability to clone channels enabled administrators to create the much needed development-production environments required by most enterprises.

Red Hat Satellite 5 provides organizations with the benefits of Red Hat Network without the need for public Internet access for servers or client systems. This brings together the tools, services, and information repositories needed to maximize the reliability, security, and performance of your systems.

2.1.2. Red Hat Satellite 6

Red Hat Satellite 6 is the evolution of Red Hat's life cycle management platform. It provides the capabilities that administrators have come to expect in a tool focused on managing systems and content for a global enterprise. Satellite 6 covers the use cases requested by Satellite 5 customers, but also includes functionality that enables larger scale, federation of content, better control of systems during the provisioning process, and a much more simplified approach to life cycle management. Satellite 6 also further evolves the inherent approach to certificate-based entitlements and integrated subscription management.

2.1.3. Comparison of Concepts

The following table outlines some key concepts and their respective implementation in both Satellite 5 and Satellite 6.

Table 2.1. Comparison of Satellite 5 and Satellite 6 Concepts

Concept	Description	Satellite 5	Satellite 6

Concept	Description	Satellite 5	Satellite 6
Open source projects	A single project approach versus a modular approach.	Spacewalk	Foreman, Katello, Puppet, Candlepin, and Pulp
Subscription types	Pool- or channel-based versus certificate-based. Subscription management has improved over the years from a pool- or channel-based approach to a more specific certificate-based approach. Certificate-based subscription management provides better overall control of subscriptions used by clients.	Entitlements	Subscriptions
Subscription methods (or Satellite subscription consumption).	The way that Satellite is enabled to synchronize and distribute Red Hat content. Certificates are activated during installation; manifests are uploaded after installation.	Certificate file	Manifest file
Organization management	Both Satellite 5 and 6 have a concept of multiple organizations, but Satellite 6 also includes functionality to include the context of the location.	Organizations	Organizations and Locations

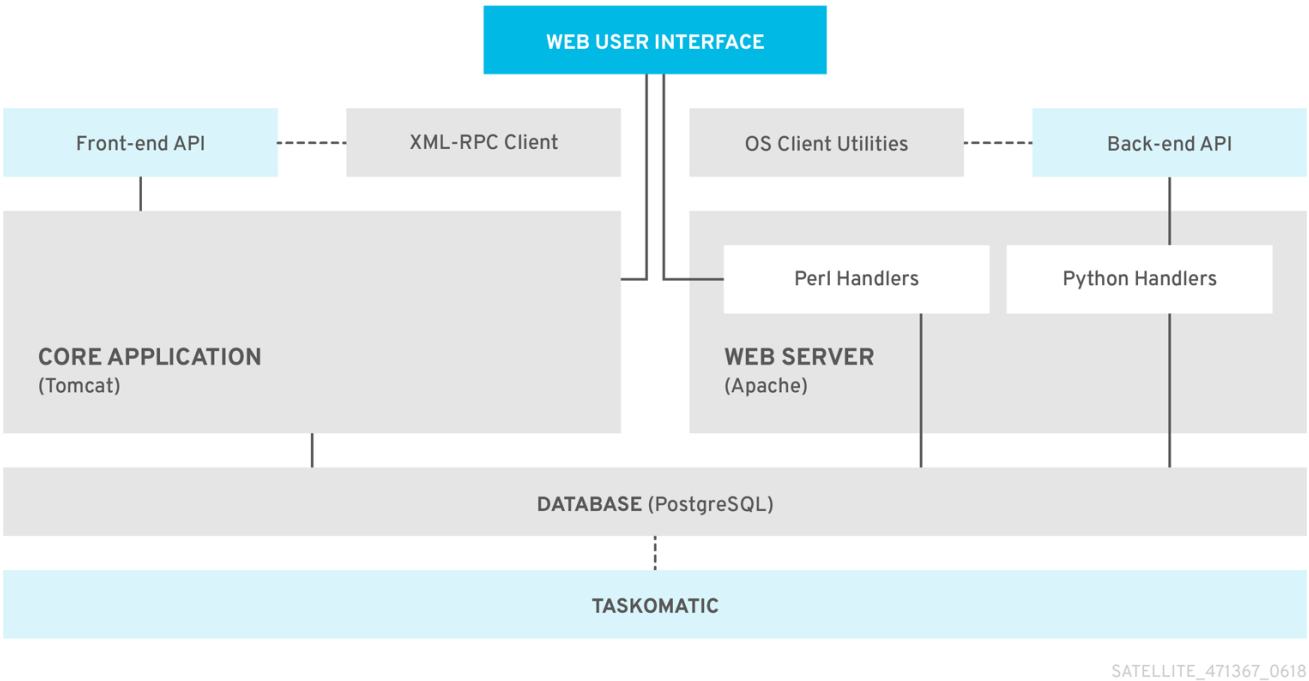
Concept	Description	Satellite 5	Satellite 6
Software and configuration content	Distributed through channels versus distributed through content views published and promoted through environments. In Satellite 6 a content view contains a chosen set of software repositories and configuration modules that are published and promoted to an environment. Client systems consume its software and configurations through its environment associations.	Software Channels	Products and repositories
Configuration		Configuration Channels	Puppet Repositories
Proxy services		Red Hat Satellite Proxy Server	Red Hat Satellite Capsule Server
Command-line tools		Various CLI tools	Hammer
Virtualization and cloud providers		KVM and Xen	OpenStack, Red Hat Enterprise Virtualization, KVM, VMware, EC2
Database support		Embedded PostgreSQL, managed PostgreSQL, external PostgreSQL, Oracle Database 10g Release 2 or 11g (Standard or Enterprise Edition)	Embedded PostgreSQL for 6.0.

2.2. SYSTEM ARCHITECTURES

2.2.1. Red Hat Satellite 5

Red Hat Satellite 5 is based on an open source project called Spacewalk and is comprised of several key components arranged in the following architecture.

Figure 2.1. Red Hat Satellite 5 System Architecture



Web UI

The Satellite web UI runs through an Apache web server and provides the main entry point for Satellite operations.

Front-end API

The front-end API provides the ability to interact with Satellite 5 through an XML-RPC API. This allows system administrators to write scripts to perform repetitive tasks, or develop third-party applications around Satellite. The front-end API exposes most of the web UI functionality using XML-RPC.

Back-end API

The back end provides a set of APIs that the different client utilities (**`rhn_register`**, **`yum`**) connect to. These are not documented and are used solely by the client utilities.

Taskomatic

Taskomatic is a separate service within Red Hat Satellite 5 that runs various asynchronous jobs, such as cleaning up the sessions table, or sending email notifications for new errata. The majority of these jobs run periodically, and you can adjust the frequency with which they occur.

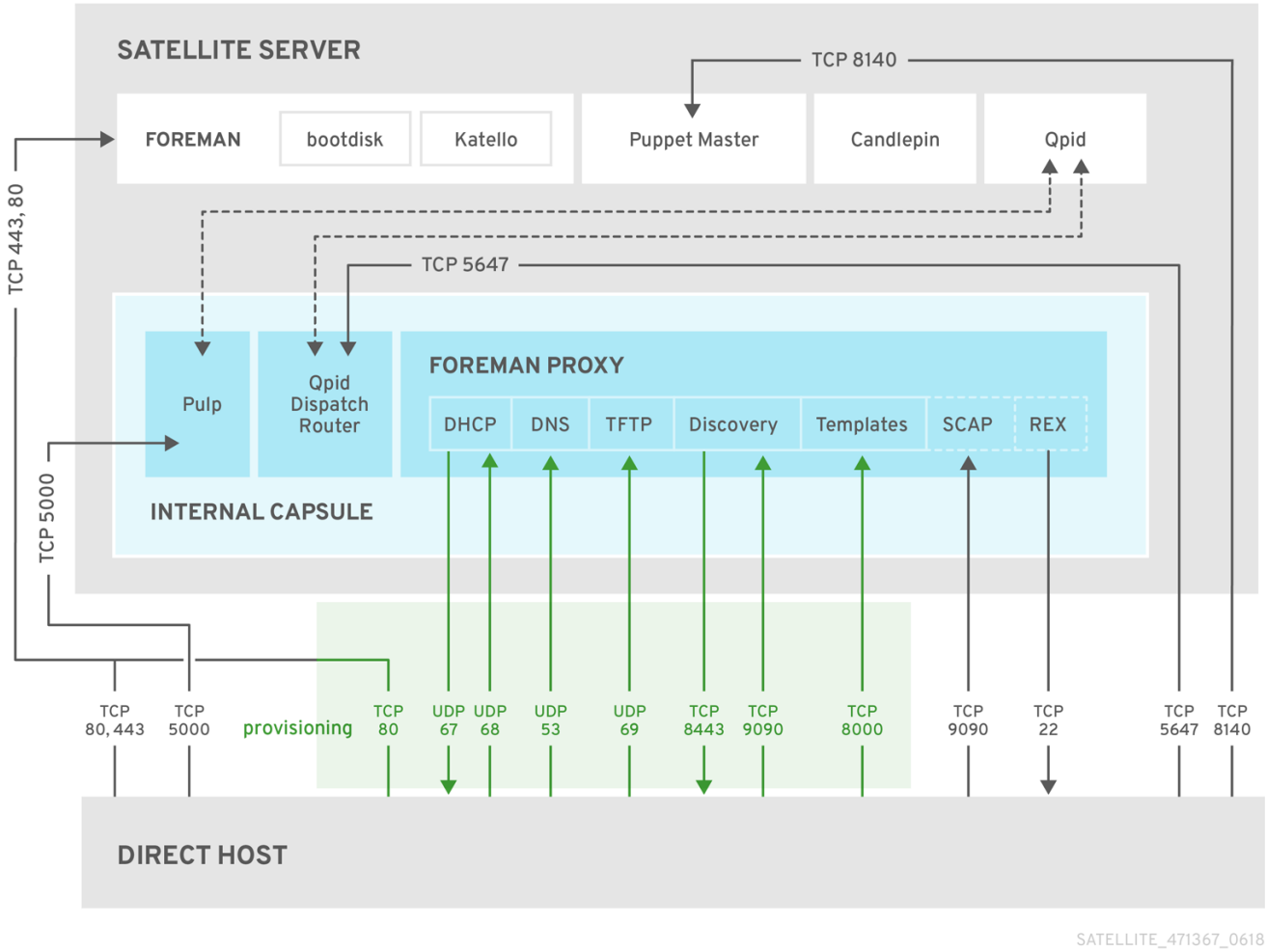
Search Server

Satellite contains a standalone search server that runs as a daemon that allows you to quickly find a system, package, or errata, as opposed to paging through hundreds of items in a list. It uses Apache's Lucene search engine library, which provides more relevant search results and a richer query language.

2.2.2. Red Hat Satellite 6

Red Hat Satellite 6 is based upon several open source projects arranged in the following architecture.

Figure 2.2. Red Hat Satellite 6 System Architecture



Foreman

Foreman is an open source application used for provisioning and life cycle management of physical and virtual systems. Foreman automatically configures these systems using various methods, including kickstart and Puppet modules. Foreman also provides historical data for reporting, auditing, and troubleshooting.

Katello

Katello is a subscription and repository management application. It provides a means to subscribe to Red Hat repositories and download content. You can create and manage different versions of this content and apply them to specific systems within user-defined stages of the application life cycle.

Candlepin

Candlepin is a service within Katello that handles subscription management.

Pulp

Pulp is a service within Katello that handles repository and content management.

Hammer

Hammer is a CLI tool that provides command line and shell equivalents of most web UI functions.

REST API

Red Hat Satellite 6 includes a REST-based API service that allows system administrators and developers to write custom scripts and third-party applications that interface with Red Hat Satellite.

Capsule

Red Hat Satellite Capsule Server acts as a proxy for some of the main Satellite functions including repository storage, **DNS**, **DHCP**, and Puppet Master configuration. Each Satellite Server also contains integrated Capsule Server services.

Note that Red Hat Satellite 6 can be installed only on x86_64 architecture systems.

2.3. CONTENT MANAGEMENT

2.3.1. Red Hat Satellite 5

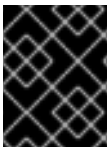
Red Hat Satellite 5 architecture includes Red Hat Satellite Proxy Server, a package-caching mechanism that reduces the bandwidth requirements for Red Hat Satellite and enables custom package deployment. The Satellite Proxy acts as a go-between for client systems and the Satellite Server.

From the client's perspective, there is no difference between a Satellite Proxy and a Satellite. From the Satellite Server's perspective, a Satellite Proxy is a special type of Satellite client.

Satellite Proxy servers are exclusive to Satellite 5; you cannot use Satellite Proxy servers with Satellite 6. Instead, Satellite 6 introduces the concept of Capsules, which provide much the same functionality.

2.3.2. Red Hat Satellite 6

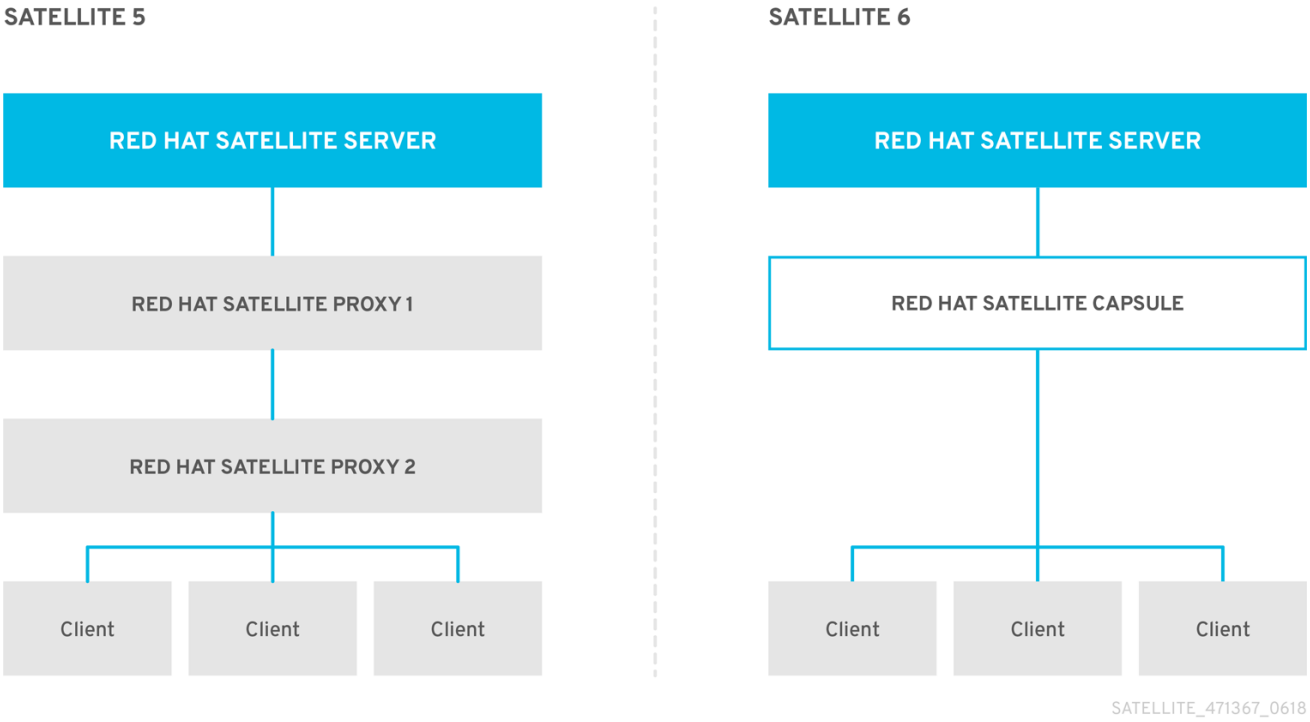
Satellite 6 architecture includes Capsule Servers to provide a similar level of functionality for Satellite 6 that Proxy servers provide for Satellite 5.



IMPORTANT

You cannot tier Capsule Servers the same way you can with Proxy servers. You can see this in the following illustration.

Figure 2.3. Comparison of Satellite 5 Proxy and Satellite 6 Capsule Servers



The first release of Capsule Servers, delivered with Satellite 6.0, can provide the following functionality:

- Mirror repository content (Pulp Node). Content can be staged on a Pulp Node before it is used by a host.
- Mirror Puppet content (Puppet Master)
- Use DHCP, DNS, and TFTP, and integrate with Identity Management (IdM).

2.4. DISCONNECTED CONTENT MANAGEMENT

A key difference between Satellite 5 and Satellite 6 is in the area of "disconnected" content management. Both versions of Satellite can provision and keep hosts synchronized without direct connection to the Internet, but the way they achieve this is slightly different.

2.4.1. Red Hat Satellite 5

Red Hat Satellite 5 achieves disconnected content management by using the **`katello-disconnected`** utility and a synchronization host. An intermediary system with an Internet connection is needed to act as a synchronization host. This synchronization host is in a separate network from Satellite Server.

The synchronization host imports content from the Red Hat Content Delivery Network (CDN). The content is then exported onto a media, such as DVDs, CDs, or external hard drives and transferred to the disconnected Satellite Server.

2.4.2. Red Hat Satellite 6

Red Hat Satellite 6 achieves disconnected content management by using a second Internet facing Satellite and the Inter-Satellite Synchronization (ISS) feature.

The connected Satellite imports content from the Red Hat Content Delivery Network (CDN). The content is then exported onto a media, such as DVDs, CDs, or external hard drives and transferred to the disconnected Satellite Server. The Inter-Satellite Synchronization feature enables full or incremental updates to be made. See Synchronizing Content Between Satellite Servers in the *Content Management Guide* for more information.

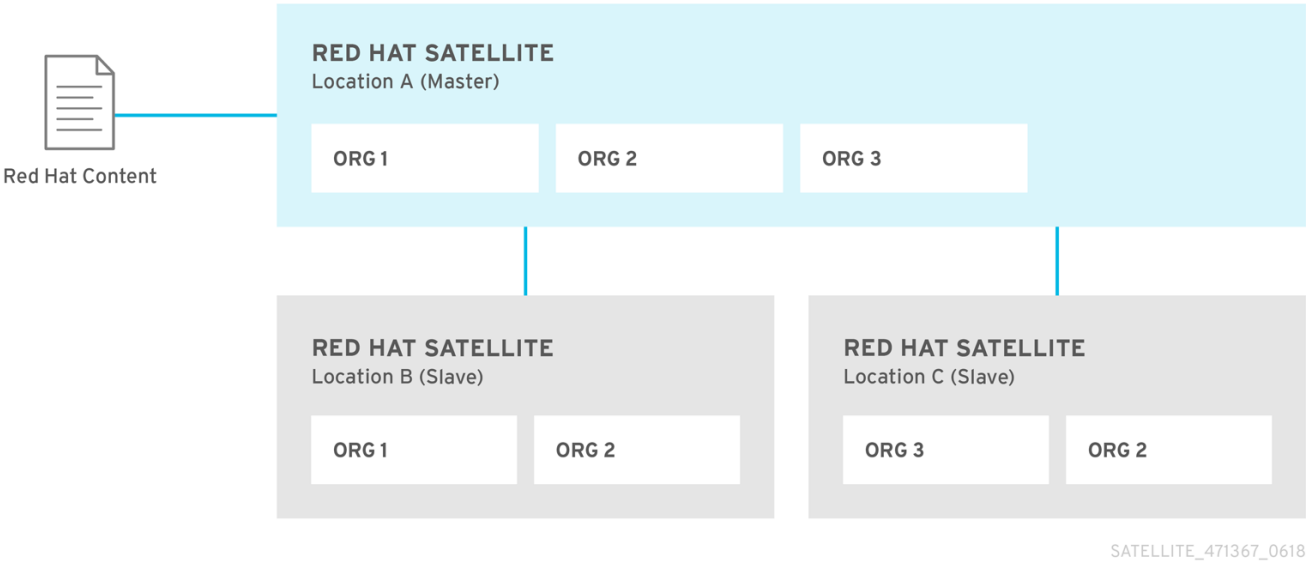
2.5. ORGANIZATIONAL STRUCTURES

2.5.1. Red Hat Satellite 5

Red Hat Satellite 5 can group systems, content, and configuration into multiple and distinct organizations. This is useful for companies with multiple divisions, such a Finance, Marketing, Sales, and so forth. Each organization acts as an individually managed Satellite, each with their own configuration and system profiles. However, Satellite can share content (software channels) among multiple organizations.

Red Hat Satellite 5 also has the ability to synchronize content across multiple Satellite Servers. This allows administrators to provide geographically dispersed Satellites that share the same software channels. For example, a company might have offices in three separate locations and each might require a Satellite, but synchronizing content from a chosen parent Satellite Server.

Figure 2.4. Example Topology for Red Hat Satellite 5

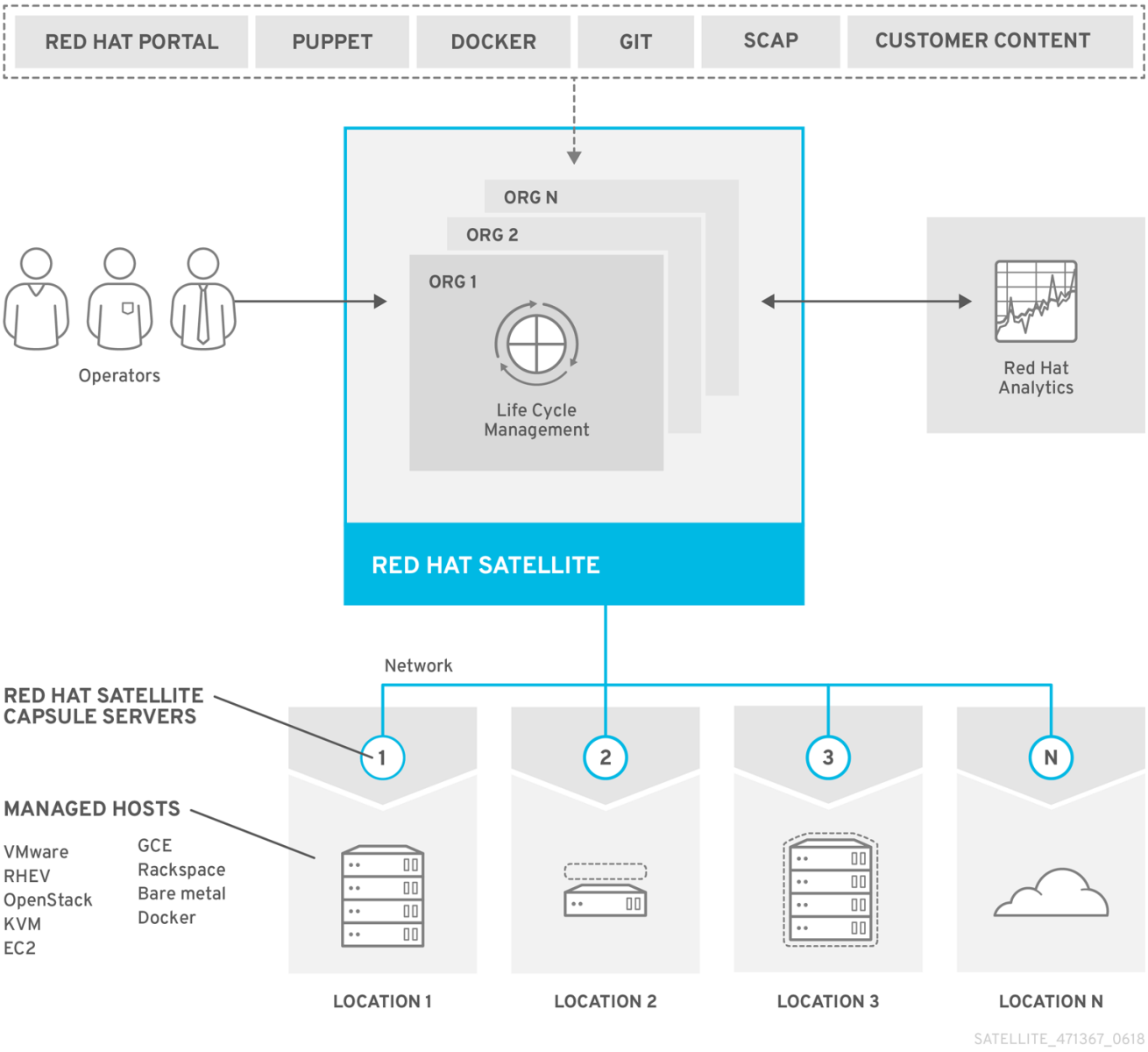


All systems management operations occur on the Satellite to which they are registered.

2.5.2. Red Hat Satellite 6

Red Hat Satellite 6 takes a consolidated approach to Organization and Location management. System administrators define multiple Organizations and multiple Locations in a single Satellite Server. For example, a company might have three Organizations (Finance, Marketing, and Sales) across three countries (United States, United Kingdom, and Japan). In this example, the Satellite Server manages all Organizations across all geographical Locations, creating nine distinct contexts for managing systems. In addition, users can define specific locations and nest them to create a hierarchy. For example, Satellite administrators might divide the United States into specific cities, such as Boston, Phoenix, or San Francisco.

Figure 2.5. Example Topology for Red Hat Satellite 6



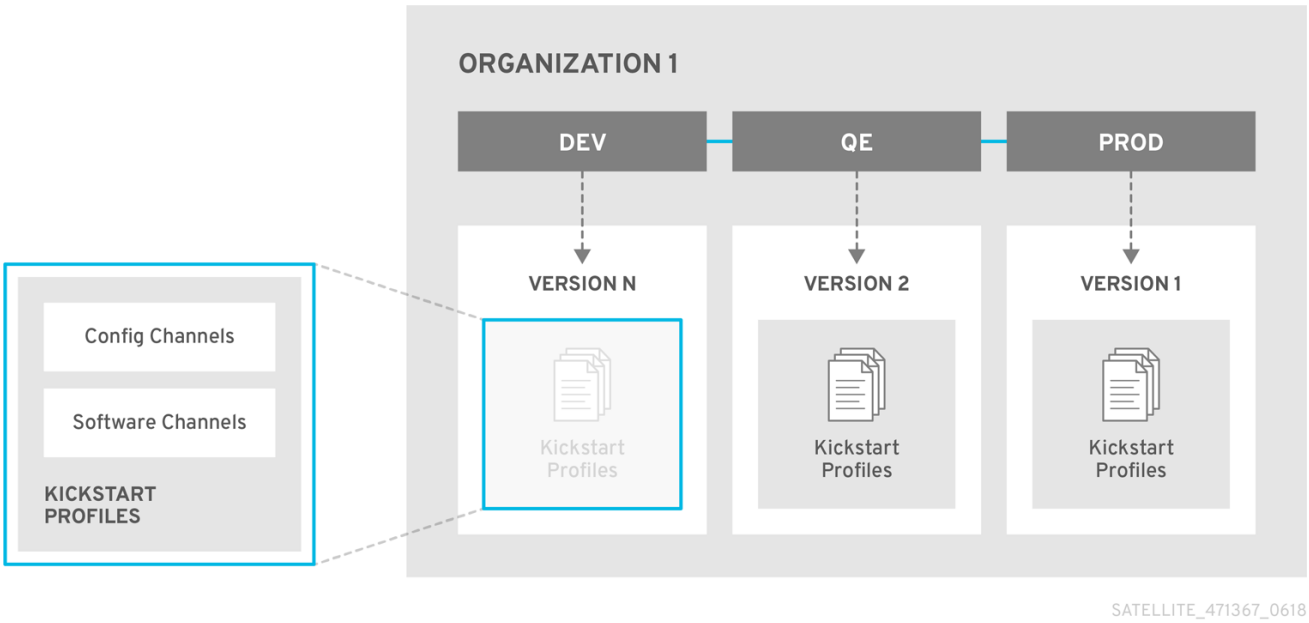
The main Satellite Server retains the management function, while the content and configuration is synchronized between the main Satellite Server and a Satellite Capsule assigned to certain locations.

2.6. APPLICATION LIFE CYCLES

2.6.1. Red Hat Satellite 5

The application life cycle in Red Hat Satellite 5 follows stages in a path. For example, applications might move along a path from "Development" to "Testing" to "General Availability". Each stage in Red Hat Satellite 5 uses System Definitions to manage systems at a particular point in the life cycle. A System Definition in Red Hat Satellite 5 is a set of Software Channels and Configuration Channels that are used in Kickstart Profiles.

Figure 2.6. The Application Life Cycle of Red Hat Satellite 5

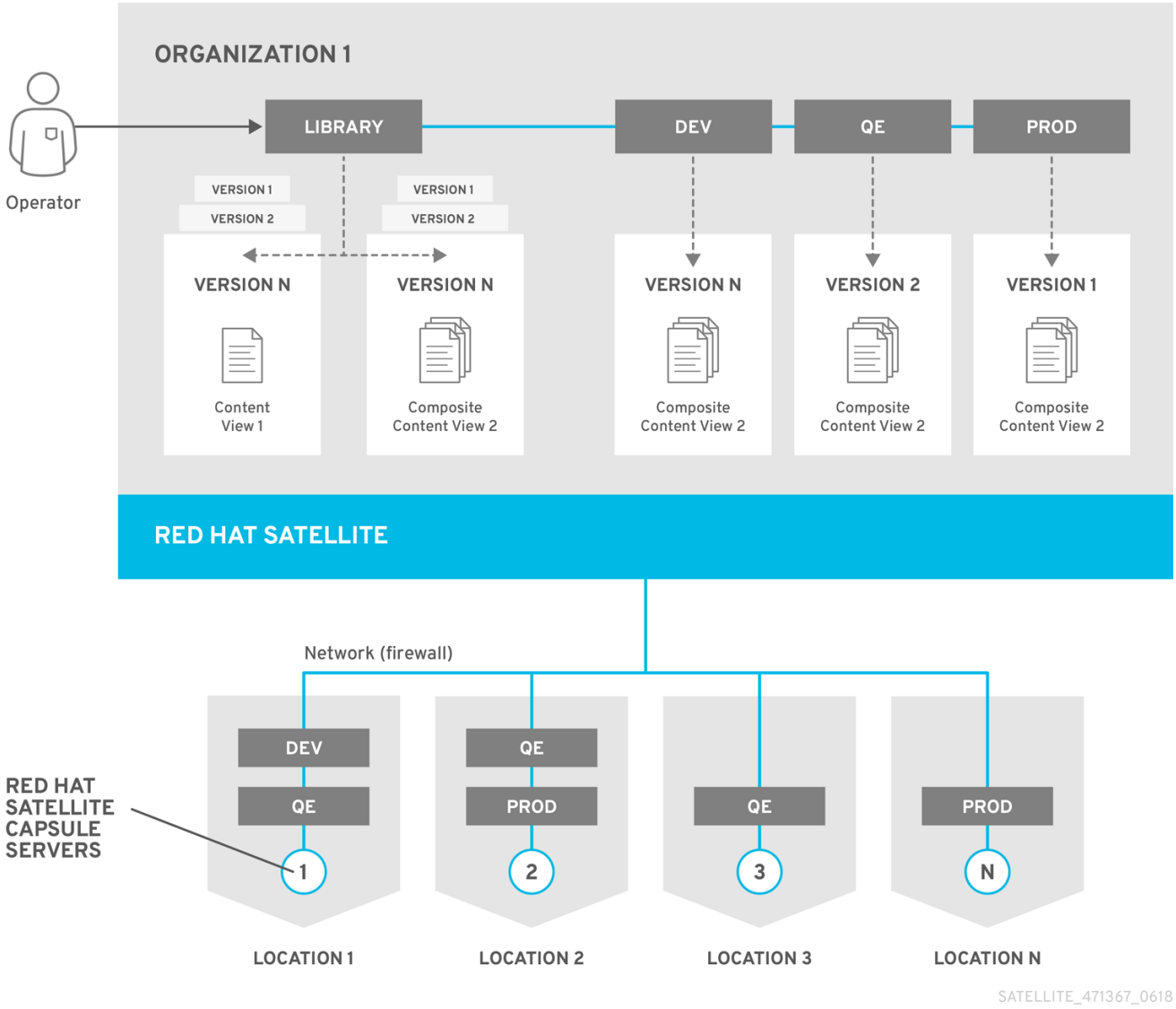


2.6.2. Red Hat Satellite 6

The Red Hat Satellite 6 application life cycle is broken up into two key components: Life Cycle Environments and Content Views.

The application life cycle is divided into *life cycle environments*, which mimic each stage of the life cycle. These life cycle environments are linked in an *environment path*. You can promote content along the environment path to the next life cycle stage when required. For example, if development completes on a particular version of an application, you can promote this version to the testing environment and start development on the next version.

Figure 2.7. An Environment Path Containing Four Environments.



Content views are managed selections of content, which contain one or more yum or Puppet repositories with optional filtering. These filters can be either inclusive or exclusive, and tailor a system view of content for life-cycle management. They are used to customize content to be made available to client systems. Content view versions are promoted along an environment path during the application life cycle. Published content views are used with life cycle environments.

CHAPTER 3. TRANSITIONING FROM SATELLITE 5 TO 6

This chapter describes how to perform migrating of clients from the old Satellite 5 to Satellite 6 using the bootstrap script, and how to retire the old Satellite instance after the transition process is finished.

The bootstrap script used for transition handles content registration, Product certificates, and Puppet configuration. The bootstrap script has the advantage of subscribing a Red Hat Enterprise Linux system to a Satellite 6, regardless of where it is registered (RHN, Satellite 5, SAM, RHSM), or if it is registered at all.

The bootstrap script enables you to:

- Migrate a client from Satellite 5 to Satellite 6.
- Migrate a client from one Satellite 6 instance to another.
- Register a new Red Hat Enterprise Linux system to Satellite 6.

3.1. PREREQUISITES

- If you have Satellite 5.8 running on a s390 system, you must build a fresh Satellite 6 on an x86_64 architecture system. Satellite 6 cannot run on s390 systems.
- Ensure you start with a fully-functional, up-to-date Satellite 6.
- The Satellite 6 instance must be synchronized with the required Red Hat content before you start the transition process.
- You have configured activation keys for the hosts. For more information on configuring activation keys, see Managing Activation Keys in the *Content Management Guide*.
- Optionally, configure host collections to match your environment. For more information on creating a host collection, see Configuring Host Collections in *Managing Hosts*.

3.2. THE TRANSITIONING WORKFLOW

The workflow includes the following steps:

1. Installing the Bootstrap Script on a Client.
2. Migrating a Red Hat Enterprise Linux System:
    a. Running the bootstrap script on a client system.
    b. Approving the Puppet certificate in the web UI as an administrator.
    c. Ensuring the completing of the transition process on a client system.

3.3. PERFORMING THE TRANSITION

Satellite 6 uses the bootstrap script to migrate existing clients from the old Satellite 5 instance.

The bootstrap script handles content registration, Product certificates, and Puppet configuration.

3.3.1. Installing the Bootstrap Script

The bootstrap script package, **katello-client-bootstrap**, is installed by default on the base system of Satellite Server and the script itself is installed in the /**var**/**www**/**html**/**pub**/ directory to make it available to clients. It can be accessed using a URL in the following form:

```
satellite.example.com/pub/bootstrap.py
```

The script includes documentation in a readme file. To view the file on the Satellite CLI:

```
$ less /usr/share/doc/katello-client-bootstrap-version/README.md
```

Installing the Bootstrap Script on a Client

As the script is only required once, and only for the **root** user, you can place it in **/root** and remove after use. As **root**, install the bootstrap script on client systems as follows:

1. Ensure you are in the **root** directory:

   ```
   # cd
   ```

2. Download the script:

   ```
   # curl -O http://satellite.example.com/pub/bootstrap.py
   ```

   This will install the script to the current directory.

3. Make the script executable:

   ```
   # chmod +x bootstrap.py
   ```

4. To confirm that the script can now be run, view the usage statement as follows:

   ```
   # ./bootstrap.py -h
   ```

5. Optionally, when the transition process is complete, remove the script:

   ```
   # cd
   # rm bootstrap.py
   ```

3.3.2. Migrating a Red Hat Enterprise Linux 6 System

Migrating a Red Hat Enterprise Linux 6 System

1. Enter the bootstrap command as follows with values suitable for your environment.
   For the **--server** option, specify the FQDN name of Satellite Server or Capsule Server. For**--location**, **--organization**, and **--hostgroup** options, use quoted names, not labels, as arguments to the options.

   ```
   # bootstrap.py --login=admin \
   --server satellite6.example.com \
   --location="Example Location" \
   ```

```
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key
```

The script will prompt you for the password corresponding to the Satellite user name you entered with the **`--login`** option.

2. The script will run and send notices of progress to **`stdout`**. Watch for output prompting you to approve the Puppet certificate. For example:

```
[NOTIFICATION], [2016-04-26 10:16:00], [Visit the UI and approve
this certificate via Infrastructure->Capsules]
[NOTIFICATION], [2016-04-26 10:16:00], [if auto-signing is disabled]
[RUNNING], [2016-04-26 10:16:00], [/usr/bin/puppet agent --test --
noop --tags no_such_tag --waitforcert 10]
```

3. The client will wait until an administrator approves the Puppet certificate. Sign the Puppet certificate as follows:

   a. In the web UI, navigate to **Infrastructure** > **Capsules**.

   b. Select **Certificates** to the right of the name of the Capsule corresponding to the FQDN given with the **`--server`** option.

   c. In the **Actions** column select **Sign** to approve the client's Puppet certificate.

   d. Return to the client to see the remainder of the bootstrap process completing.

4. In the web UI, navigate to **Hosts** > **All hosts** and ensure that the client is connected to the correct host group.

For more information about using the bootstrap script, see Registering Hosts to Satellite 6 Using The Bootstrap Script in *Managing Hosts*.

3.4. RETIRING THE OLD SATELLITE 5 SERVER

This section shows how to retire the old Satellite 5 Server after you have performed the complete migration from Red Hat Satellite 5 Server to a new Satellite 6 Server and you do not need the old Satellite 5 Server anymore. If you have migrated your Satellite 5 to the new RHSM system, proceed to Section 3.4.1, "Migrating Subscriptions to the New Satellite 6".

If you have not migrated your Satellite 5 to the new RHSM system, create new manifests as described in Managing Subscriptions in the *Content Management Guide* and add them to the new Satellite 6. Optionally, run the following Python script to delete the profile of the old Satellite from RHN:

```
#!/usr/bin/env python

import getpass
import os
import sys
import libxml2
import xmlrpclib
from optparse import OptionParser

DEFAULT_SERVERFQDN="xmlrpc.rhn.redhat.com"
```

```
parser = OptionParser()
parser.add_option("-l", "--login", dest="login", help="Login user for RHN
Satellite/Hosted", metavar="LOGIN")
parser.add_option("-p", "--password", dest="password", help="Password for
specified user. Will prompt if omitted", metavar="PASSWORD")
parser.add_option("-s", "--server", dest="serverfqdn", help="FQDN of
satellite server - omit https:// (default: %s)" % DEFAULT_SERVERFQDN,
metavar="SERVERFQDN", default=DEFAULT_SERVERFQDN)
(options, args) = parser.parse_args()

if not options.login:
    print "Must specify login option.  See usage:"
    parser.print_help()
    print "\nExample usage: ./decommissionServer.py -l admin -p password"
    sys.exit(1)
else:
    login = options.login
    password = options.password
    serverfqdn = options.serverfqdn

if not password: password = getpass.getpass("%s's password:" % login)

SATELLITE_URL = "https://%s/rpc/api" % serverfqdn
SATELLITE_LOGIN = login
SATELLITE_PASSWORD = password
SYSTEMID_FILE = "/etc/sysconfig/rhn/systemid"

# Check For root
# Have to be root to open the SYSTEMID_FILE which is chmod'd 0600
if os.getuid() != 0:
    print "This script requires root-level access to run."
    sys.exit(1)

# Log into Satellite and get an authentication token
client = xmlrpclib.Server(SATELLITE_URL, verbose=0)
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Parse /etc/sysconfig/rhn/systemid to ge the system ID
# Use the systemid and delete the systems RHN profile
parsed_file = libxml2.parseDoc(file(SYSTEMID_FILE).read())
systemid = parsed_file.xpathEval('string(//member[*
="system_id"]/value/string)').split('-')[1]
print systemid
try:
    client.system.deleteSystems(key,int(systemid))
    print "The system with SystemID " + systemid + " was successfully
deleted"
except xmlrpclib.Fault, e:
    print "XMLRPC fault \n\t%s" % e

# Logout of Satellite
client.auth.logout(key)
```

3.4.1. Migrating Subscriptions to the New Satellite 6

This section describes how to migrate subscriptions from the old Satellite 5 Server to the new Satellite 6 Server.

1. Configure new subscriptions:

   a. Log in to the Customer Portal and go to **SUBSCRIPTUINS**.

   b. Navigate to **Inventory** and click **Satellite Organizations**.

   c. Select the check box to the left of the name of the old Satellite 5 Server and click **Delete Selected**. In the alert box that appears, click **Delete** to delete the unit.

   d. Click on the name of the new Satellite 6 Server and select the check box to the left of the name of the transition subscription. Click **Remove selected**. In the alert box click **Remove** to remove the subscription.

   e. Click **Attach a subscription**, select the real Satellite subscription, and click **Attach selected**.

   f. Run the following command on the new Satellite 6 Server to refresh subscription manager.

   ```
   # subscription-manager refresh
   ```

2. Enable repositories on the new Satellite 6 Server

   a. On the old Satellite 5 Server use the following command to view what channels you had enabled.

   ```
   # spacewalk-channel -l
   ```

   b. On the new Satellite 6 Server use the following command to list enabled repositories for an organization:

   ```
   # hammer product list --organization "organization_name"
   ```

   c. Configure the new Satellite 6 Server to ensure all necessary repositories are enabled.
   For Web UI users

   Navigate to **Content** → **Red Hat Repositories** and select repositories to be enabled.

   For CLI Users

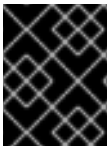
   Enable repositories using either the name or ID number. Optionally, include the release version and base architecture.

   ```
   # hammer repository-set enable \
   --product "product_name" \
   --name "repository_name" \
   --organization "org_name" \
   --releasever "" \
   --basearch "x86_64"
   ```

CHAPTER 4. TRANSITIONING TO THE SATELLITE 6 API

One of the many differences between Red Hat Satellite 5 and Satellite 6 is the API. Satellite 5 uses an XMLRPC-based API. Satellite 6 uses a REST-based API. This fundamental difference requires that any existing scripts or tools that have been integrated with the Satellite 5 API must be reviewed and at least partially rewritten before use with the Satellite 6 REST API.

This section provides some comparisons of how to achieve the same use case within each Product. This is not designed to be a tutorial in any specific programming language. Neither are the scripts secured over HTTPS. They provide a starting point for anyone maintaining Satellite 5 API scripts to start the transition to the Satellite 6 API.



IMPORTANT

The Satellite 6 transition tools do not transition the Satellite 5 API or scripts to Satellite 6. Use this section as a starting point to begin your own transition process.

Further Information

You can find further API documentation in the *API Guide* and at the following locations on your own Satellite Servers:

- Satellite 6: https://satellite6.example.com/apidoc/v2.html
- Satellite 5: https://satellite5.example.com/rpc/api

4.1. EXAMPLE API SCRIPTS

The examples in this section cover the following:

1. Systems and hosts in Red Hat Satellite
    a. Authenticate and request a list of systems (Satellite 5) or hosts (Satellite 6) available to the user who logged in.
2. Users and Roles
    a. Authenticate and request a list of all users visible to the user who logged in.
    b. Delete the **example** user if it exists.
    c. Create a new **example** user and ensure that its role is set to *Administrator*.

This section provides a total of five different ways to achieve the same result; two examples for Satellite 5 and three examples for Satellite 6.

- A Satellite 5 Python script
- A Satellite 6 Python script
- A Satellite 6 Ruby script
- A Satellite 5 **`spacecmd`** example
- A Satellite 6 **`hammer`** example

NOTE

The **spacecmd** command is not a supported feature in Satellite 5.6. It is supported in Satellite 5.7, and is shown here for the sake of completeness.

A total of 10 examples are provided. The first examples cover the simpler use cases of listing systems and hosts, followed by the more complex examples covering users and roles.

4.1.1. Listing Systems and Hosts

The examples in this section describe different ways to list systems and hosts available to the user who logged in.

Using Python to List Available Systems on Satellite 5

This example uses a Python script to connect to Satellite 5, authenticate, and retrieve a list of available systems.

```
#!/usr/bin/python
import xmlrpclib

# Define Satellite location and login details
SATELLITE_URL = "http://localhost/rpc/api"
SATELLITE_LOGIN = "admin"
SATELLITE_PASSWORD = "password"

client = xmlrpclib.Server(SATELLITE_URL, verbose=0)

# Authenticate and get session key
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Get list of systems available to user
list = client.system.listSystems(key)
for system in list:
   print system.get('id')
   print system.get('name')

# Logout
client.auth.logout(key)
```

Using Python to List Available Hosts on Satellite 6

This example uses a Python script to connect to Satellite 6, authenticate, and retrieve a list of available hosts.

```
#!/usr/bin/python
import json
import requests

# Define Satellite location and login details
SAT_API = "https://localhost/api/v2/"
USERNAME = "admin"
PASSWORD = "changeme"
SSL_VERIFY = False
```

```
def get_json(location):
    """
    Performs a GET using the passed URL location
    """

    r = requests.get(location, auth=(USERNAME, PASSWORD),
verify=SSL_VERIFY)

    return r.json()

def main():
    # List all hosts available to the user
    hosts = get_json(SAT_API + "hosts/")

    # Pretty Print the returned JSON of Hosts
    print json.dumps(hosts, sort_keys=True, indent=4)

if __name__ == "__main__":

main()
```

Using Ruby to List Available Hosts on Satellite 6

This example uses a Ruby script to connect to Satellite 6, authenticate, and retrieve a list of available hosts.

```
#!/usr/bin/env ruby
require 'json'
require 'rest-client'

url = 'https://localhost/api/v2/'
$username = 'admin'
$password = 'changeme'

def get_json(location)
    response = RestClient::Request.new(
        :method => :get,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

hosts = get_json(url+"hosts/")
#puts JSON.pretty_generate(hosts)

puts "Hosts within Satellite are:"
hosts['results'].each do |name|
      puts name['name']
end

exit()
```

Using the Command Line to List Available Systems on Satellite 5

Use the following command to list available systems on Satellite 5:

```
# spacecmd -u username -p password system_list
```

An example session might appear as follows:

```
# spacecmd -u admin -p password system_list

INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
test_02.example.com
```

Using the Command Line to List Available Hosts on Satellite 6

Use the following command to list available hosts on Satellite 6:

```
# hammer host list
```

An example session might appear as follows:

```
# hammer host list

[Foreman] password for admin:
---|-----------------|-------------------|------------|--------------|-
-----------------
ID | NAME            | OPERATING SYSTEM  | HOST GROUP | IP           |
MAC
---|-----------------|-------------------|------------|--------------|-
-----------------
1  | test.example.com | RHEL Server 6.5  |            | 10.34.34.235 |
e4:1f:13:6b:ed:0c
```

4.1.2. Deleting and Creating Users

The examples in this section describe different ways to locate, create, and delete users.

Using Python to Manage Users on Satellite 5

This example uses a Python script to connect to and authenticate against a Satellite 5 Server. It then goes on to search for a specific user (**example**), to delete that user if it exists, and then recreate it with Administrator privileges.

```
#!/usr/bin/python
import xmlrpclib

# Define Satellite location and login details
SATELLITE_URL = "http://localhost/rpc/api"
SATELLITE_LOGIN = "admin"
SATELLITE_PASSWORD = "password"

client = xmlrpclib.Server(SATELLITE_URL, verbose=0)
```

```
# Authenticate and get session key
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Get list of users
list = client.user.list_users(key)
print "Existing users in Satellite:"
for user in list:
   print user.get('login')

# Look for user example and if found, delete the user
for user in list:
   if user.get('login') == 'example':
       deleteuser = client.user.delete(key, 'example')
       if deleteuser == 1:
           print "User example deleted"

# Create a user called example
createuser = client.user.create(key, 'example', 'password', 'Example',
'User', "root@localhost")
if createuser == 1:
    print "User example created"
    # Admin Org Admin role to the example user
    adminrole = client.user.addRole(key, 'example', 'org_admin')
    if adminrole == 1:
        print "Made example an Org Admin"

# Logout
client.auth.logout(key)
```

Using Python to Manage Users on Satellite 6

This example performs the same task as the previous example. That is, it uses a Python script to connect to and authenticate against a Satellite 6 Server, search for a specific user, delete that user if it exists, and then recreate it with Administrator privileges.

```
#!/usr/bin/python
import json
import requests

# Define Satellite location and login details
SAT_API = "https://localhost/api/v2/"
POST_HEADERS = {'content-type': 'application/json'}
USERNAME = "admin"
PASSWORD = "changeme"
SSL_VERIFY = False

def get_json(location):
    """
    Performs a GET using the passed URL location
    """
    r = requests.get(location, auth=(USERNAME, PASSWORD),
verify=SSL_VERIFY)

    return r.json()

def post_json(location, json_data):
```

```
    """
    Performs a POST and passes the data to the URL location
    """
    result = requests.post(
        location,
        data=json_data,
        auth=(USERNAME, PASSWORD),
        verify=SSL_VERIFY,
        headers=POST_HEADERS)

    return result.json()

def delete_json(location):
    """
    Performs a DELETE and passes the id to the URL location
    """
    result = requests.delete(
        location,
        auth=(USERNAME, PASSWORD),
        verify=SSL_VERIFY)

    return result.json()

def main():
    # List all users within the Satellite
    users = get_json(SAT_API + "users/")

    #print json.dumps(users, indent=4)
    print "Users known are:"
    for login in users['results']:
        print login['login']

    # Look for user example and if found, delete the user
    for delete in users['results']:
        if delete['login'] == 'example':
            id = delete ['id']
            id = str(id)

            delete = delete_json(SAT_API + "/users/" + id)
            #print json.dumps(delete, indent=4)
            print "User example deleted"

    # Create a user called example as admin role
    createuser = post_json(SAT_API + "/users/", json.dumps({ "mail":
"root@localhost", "firstname": "Example", "lastname": "User", "login":
"example", "password": "redhat", "admin": 'true', "auth_source_id": 1 }))
    #print json.dumps(createuser, indent=4)
    print "Admin user example created"

if __name__ == "__main__":
    main()
```

Using Ruby to Manage Users on Satellite 6

This example uses Ruby to perform the same task as the previous examples.

```
#!/usr/bin/env ruby
require 'json'
require 'rest-client'

url = 'https://localhost/api/v2/'
$username = 'admin'
$password = 'changeme'

def get_json(location)
    response = RestClient::Request.new(
        :method => :get,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

def post_json(location, json_data)
    response = RestClient::Request.new(
        :method => :post,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json},
        :payload => json_data
    ).execute
    results = JSON.parse(response.to_str)
end

def delete_json(location)
    response = RestClient::Request.new(
        :method => :delete,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

# List all users within the Satellite
users = get_json(url+"users/")

#puts JSON.pretty_generate(users)
puts "Users known are:"
users['results'].each do |name|
    puts name['login']
end

# Look for user example and if found, delete the user
users['results'].each do |name|
```

```
    if name['login'] == 'example'
        id = name['id']
        delete = delete_json(url+"users/"+id.to_s)
        #puts JSON.pretty_generate(delete)
        puts "User example deleted"
    end
end

# Create a user called example as admin role
data = JSON.generate({ :mail => "root@localhost", :firstname => "Example",
:lastname => "User", :login => "example", :password => "password", :admin
=> 'true', :auth_source_id => 1})
createuser = post_json(url+"users/", data)

#puts JSON.pretty_generate(createuser)
puts "Admin user example created"

exit()
```

Using the Command Line to Manage Users on Satellite 5

This example uses the **spacecmd** command to perform the same task as the previous examples.

```
# spacecmd -u admin -p password user_list
# spacecmd -u admin -p password user_delete example
# spacecmd -u admin -p password user_create
# spacecmd -u admin -p password user_addrole example org_admin
```

An example session might appear as follows:

```
# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin
example

# spacecmd -u admin -p password user_delete example
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin

Delete this user [y/N]: y

# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin

# spacecmd -u admin -p password user_create
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
Username: example
First Name: Example
Last Name: User
Email: root@localhost
PAM Authentication [y/N]: n
Password:
```

```
Repeat Password:

# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin
example

# spacecmd -u admin -p password user_addrole example org_admin
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin

# spacecmd -u admin -p password user_details example
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
Username:      example
First Name:    Example
Last Name:     User
Email Address: root@localhost
Organization:  MY ORG
Last Login:
Created:       8/19/14 8:42:52 AM EDT
Enabled:       True

Roles
-----
activation_key_admin
channel_admin
config_admin
monitoring_admin
org_admin
system_group_admin
```

Using the Command Line to Manage Users on Satellite 6

This example uses the **`hammer`** command to perform the same task as the previous examples.

```
# hammer shell
> user list
> user delete --id 4 --login example
> user create --admin true --firstname Example --lastname User --login
example --mail root@localhost --password redhat --auth-source-id 1
```

An example session might appear as follows:

```
# hammer shell
[Foreman] password for admin:
Welcome to the hammer interactive shell
Type 'help' for usage information

hammer> user list
---|---------|--------------|---------------
ID | LOGIN   | NAME         | EMAIL
---|---------|--------------|---------------
4  | example | Example User | root@localhost
```

```
3  | admin   | Admin User   | root@localhost
---|---------|--------------|--------------

hammer> user delete --id 4 --login example
User deleted
hammer> user list
---|-------|------------|--------------
ID | LOGIN | NAME       | EMAIL
---|-------|------------|--------------
3  | admin | Admin User | root@localhost
---|-------|------------|--------------

hammer> user create --admin true --firstname Example --lastname User --
login example --mail root@localhost --password redhat --auth-source-id 1
User created
hammer> user list
---|----------|--------------|--------------
ID | LOGIN    | NAME         | EMAIL
---|----------|--------------|--------------
3  | admin    | Admin User   | root@localhost
5  | example  | Example User | root@localhost
---|----------|--------------|--------------
hammer>
```

4.2. SATELLITE 6 API TIPS

This section provides some general tips to help optimize your experience with the Satellite 6 API.

Browsing the Satellite 6 API

If you have already logged in to the Satellite 6 web UI, you can see the default results of GET requests at /api/v2/<API-NAME>. For example:

- https://satellite6.example.com/api/v2/users/
- https://satellite6.example.com/api/v2/users/3

Using Satellite 6 API Requests on the Command Line

You can use the **`curl`** command to interact with the Satellite 6 API. For example:

Example 4.1. Example GET Requests

Example GET requests to list organizations, hosts, and users within Satellite 6.

```
# SATUSER=admin
# SATPASS='changeme'
# SATURL="https://localhost"

# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/organizations | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/hosts | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
```

```
'Accept: application/json' $SATURL/api/v2/users | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/users/3 | json_reformat
```

Example 4.2. Example DELETE Request

An example DELETE request to remove an existing user with known ID "9" based on a previous list of users.

```
# curl -k -u $SATUSER:$SATPASS -X DELETE -H \
'Accept: application/json' $SATURL/api/v2/users/9 | json_reformat
```

Example 4.3. Example POST Request

An example POST request to create a new user called **example**, passing the **`true`** flag to enable administrator privileges for the user.

```
# curl -k -u $SATUSER:$SATPASS -X POST \
-d '{ "mail": "root@localhost", "firstname": "Example", \
"lastname": "User", "login": "example", "password": "redhat", \
"admin": 'true', "auth_source_id": 1 }' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
$SATURL/api/v2/users | json_reformat
```

Example 4.4. Example PUT Request

An example PUT request to change the email address of the existing **example** user with known ID "10" to *example@localhost*.

```
# curl -k -u $SATUSER:$SATPASS -X PUT \
-d '{ "id": 10, "mail": "example@localhost" }' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
$SATURL/api/v2/users/10 | json_reformat
```

APPENDIX A. GLOSSARY OF TERMS

The following terms are used throughout this document. Familiarize yourself with these terms to help your understanding of Red Hat Satellite 6.

Activation Key

A registration token used in a Kickstart file to control actions at registration. These are similar to Activation Keys in Red Hat Satellite 5, but provide a subset of features because Puppet controls package and configuration management after registration.

Application Life Cycle Environment

An Application Life Cycle Environment represents a step, or stage, in a promotion path through the Software Development Life Cycle (SDLC). Promotion paths are also known as development paths. Content such as packages and Puppet modules move through life cycle environments by publishing and promoting Content Views. All Content Views have versions, which means you can promote a specific version through a typical promotion path; for example, from development to test to production. Channel cloning implements this concept in Red Hat Satellite 5.

Attach

The process of associating a Subscription to a Host that provides access to RPM content.

Capsule

A Capsule is an additional server that can be used in a Red Hat Satellite 6 deployment to facilitate content federation and distribution in addition to other localized services (Puppet Master, **DHCP**, **DNS**, **TFTP**, and more).

Catalog

A Catalog is a document that describes the desired system state for one specific computer. It lists all of the resources that need to be managed, as well as any dependencies between those resources.

Compute Profile

Compute Profiles specify default attributes for new virtual machines on a compute resource.

Compute Resource

A Compute Resource is virtual or cloud infrastructure, which Red Hat Satellite 6 uses for deployment of hosts and systems. Examples include Red Hat Enterprise Virtualization Manager, OpenStack, EC2, and VMware.

Content

Content includes software packages (RPM files) and Puppet modules. These are synchronized into the Library and then promoted into Life Cycle Environments using Content Views so that they can be consumed by Hosts.

Content Delivery Network (CDN)

The Content Delivery Network (CDN) is the mechanism used to deliver Red Hat content in a geographically co-located fashion. For example, content that is synchronized by a Satellite in Europe pulls content from a source in Europe.

Content Host

A Content Host is the part of a host that manages tasks related to content and subscriptions.

Content View

A Content View is a definition of content that combines Products, packages, and Puppet modules with capabilities for intelligent filtering and creating snapshots. Content Views are a refinement of the combination of channels and cloning from Red Hat Satellite 5.

External Node Classifier

An External Node Classifier is a Puppet construct that provides additional data for a Puppet Master to use when configuring Hosts. Red Hat Satellite 6 acts as an External Node Classifier to Puppet Masters in a Satellite deployment.

Facter

Facter is a program that provides information (facts) about the system on which it is run; for example, Facter can report total memory, operating system version, architecture, and more. Puppet modules enable specific configurations based on host data gathered by Facter.

Hammer

Hammer is a command line tool for Red Hat Satellite 6. Use Hammer to manage Red Hat Satellite 6 as a standard CLI, for scripts, and also through an interactive shell.

Hiera

Hiera is a key-value look-up tool for configuration data which allows keeping site-specific data out of Puppet manifests.

Host

A Host refers to any system, either physical or virtual, that Red Hat Satellite 6 manages.

Host Collection

A Host Collection is equivalent to a Satellite 5 System Group, that is, a user defined group of one or more Hosts.

Host Group

A Host Group is a template for building a Host. This includes the content view (which defines the available RPM files and Puppet modules) and the Puppet classes to apply (which ultimately determines the software and configuration).

Location

A Location is collection of default settings that represent a physical place. These can be nested so that you can set up an hierarchical collection of locations. For example, you can set up defaults for "Middle East", which are refined by "Tel Aviv", which are further refined by "Data Center East", and then finally by "Rack 22".

Library

The Library contains **every** version, including the latest synchronized version, of the software that the user will ever deploy. For an Information Technology Infrastructure Library (ITIL) organization or department, this is the Definitive Media Library (previously named the Definitive Software Library).

Manifest

A manifest transfers subscriptions from the Customer Portal to Red Hat Satellite 6. This is similar in function to certificates used with Red Hat Satellite 5.
For more information about certificates and subscription types, see:

- Subscription Concepts and Workflows.
- Migrating from RHN Classic.

Organization

An Organization is an isolated collection of systems, content, and other functionality within a Satellite 6 deployment.

Product

A collection of content repositories. Products can be Red Hat Products or newly-created Products made up of software and configuration content.

Promote

The act of moving a content view comprised of software and configuration content from one Application Life Cycle Environment to another, such as moving from development to QA to production.

Provisioning Template

A Provisioning Template is a user-defined template for Kickstart files, snippets, and other provisioning actions. In Satellite 6 they provide similar functionality to Kickstart Profiles and cobbler Snippets in Red Hat Satellite 5.

Pulp Node

A Pulp Node is a Capsule Server component that mirrors content. This is similar to the Red Hat Satellite 5 Proxy. The main difference is that content can be staged on the Pulp Node before it is used by a Host.

Puppet Agent

The Puppet Agent is an agent that runs on a Host and applies configuration changes to that Host.

Puppet Master

A Puppet Master is a Capsule Server component that provides Puppet manifests to Hosts for execution by the Puppet Agent.

Puppet Module

A Puppet module is a self-contained bundle of code and data that you can use to manage resources such as users, files, and services.

Repository

A Repository provides storage for a collection of content. For example, a yum repository or a Puppet repository.

Role

A Role specifies a collection of permissions that are applied to a set of resources, such as Hosts.

Smart Proxy

A Smart Proxy is a Capsule Server component that can integrate with external services, such as **DNS** or **DHCP**.

Smart Variable

A Smart Variable is 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 Standard Operating Environment (SOE) is a controlled version of the operating system on which applications are deployed.

Subscription

Subscriptions are the means by which you receive content and service from Red Hat.

Synchronizing

Synchronizing refers to mirroring content from external resources into the Red Hat Satellite 6 Library.

Synchronization Plans

Synchronization Plans provide scheduled execution of content synchronization.

User Group

A User Group is a collection of roles which can be assigned to a collection of users. This is similar to a Role in Red Hat Satellite 5.

User

A user is anyone registered to use Red Hat Satellite. Authentication and authorization is possible through built-in logic, through external LDAP resources, or with Kerberos.