Red Hat Performance Briefs

Performance Tuning of Satellite 6.1 and Capsules

Pradeep Surisetty, Alex Krzos

Version 1.0

RHEL 7.2

June 2016

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 4950
Fax: +1 919 800 3804

Linux is a registered trademark of Linus Torvalds. Red Hat, Red Hat Enterprise Linux and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

Dell, the Dell logo and PowerEdge are trademarks of Dell, Inc.

Intel, the Intel logo and Xeon are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

© 2016 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at http://www.opencontent.org/openpub/).

The information contained herein is subject to change without notice. Red Hat, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Distribution of modified versions of this document is prohibited without the explicit permission of Red Hat Inc.

Distribution of this work or derivative of this work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from Red Hat Inc.

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Table of Contents

1 Executive Summary

Information Technology has been constantly evolving where the volume of change is exponentially rising, while the time interval of change is shrinking. To keep up with the pace, the infrastructure has to be on par meeting the scaling and diversity challenges. It is key to efficiently and effectively deploy and manage the IT infrastructure. Red Hat Satellite is a complete system management product that allows system administrators to manage the full life cycle of Red Hat deployments across physical, virtual, and private clouds. Red Hat Satellite delivers system provisioning, configuration management, software management, and subscription management- all while maintaining high Scalability and security. Red Hat Satellite 6.1 is the second release of a new generation of Systems Management for Red Hat Enterprise Linux (RHEL) that demonstrates measurable improvements over Red Hat Satellite 5 environment on the following topics:

- Content management
- Container management
- Automation provisioning
- Discovery
- Configuration management
- Federated services
- Overall performance
- Scalability
- Common vulnerabilities and exposure management

This document provides basic guidelines and considerations for tuning Red Hat Satellite 6.1 for performance and Scalability. Many factors drive the performance of a Satellite 6.1 deployment and testing should be conducted before performing any of the suggested tuning. There is no one-size-fits-all configuration as tuning will vary based on your environmental factors, such as the hardware Satellite 6.1 is deployed on or the complexity of Puppet manifests. It is important to establish a baseline within your environment in order to determine how to scale Satellite 6.1 to meet your needs for life-cycle management of systems..

For further details on Red Hat Satellite Server, please refer to documentation at

https://access.redhat.com/documentation/en-US/Red_Hat_Satellite/6.1/html/User_Guide/sect-Red_Hat_Satellite-User_Guide-Red_Hat_Satellite_Server_6_Basic_Configuration_Workflow.html

2 Components Overview

Red Hat Satellite is a system management solution that makes Red Hat infrastructure easier to deploy, scale, and manage across physical, virtual, and cloud environments. Satellite helps users provision, configure, and update systems to ensure they run efficiently, securely, and in compliance with various standards. By automating most tasks related to maintaining systems, Satellite helps organizations increase efficiency, reduce operational costs, and enable IT to better respond to strategic business needs. Red Hat Satellite automates many tasks related to system management and easily integrates into existing work flow frameworks. The centralized console provides administrators one place for accessing reports and for provisioning, configuring, and updating systems.

2.1 Provisioning

Provision on bare metal, virtualized infrastructure, and on public or private clouds—all from one centralized console and with one simple process.

• Quickly provision and update your entire bare-metal infrastructure.
• Easily create and manage instances across virtualized infrastructure or private and public clouds.
• Create complex Kick start and PXE scenarios with powerful variables and snippets.
• Discover and search across non-provisioned hosts for rapid deployment.

2.2 Configuration Management

Analyze and automatically correct configuration drift and control, and enforce the desired host end-state, all from the Red Hat Satellite user interface (UI). This lets you configure Red Hat Enterprise Linux systems more efficiently for more agility.

• Integrate synchronizing Puppet modules. This integration provides the ability to manage, promote, and distribute configuration easily across your environment.
• Automatically correct system state with complete reporting, auditing, and history of changes.
• Integrate synchronizing Puppet modules. This integration provides the ability to manage, promote, and distribute configuration easily across your environment.
• Automatically correct system state with complete reporting, auditing, and history of changes

2.3 Software Management

Red Hat Satellite helps ensure a systematic process is used to apply content (including patches) to deployed systems— whether they are deployed on physical, virtual, or cloud infrastructure—in all stages from dev to production. This ensures better consistency and availability of systems, freeing IT to quickly respond to business needs and vulnerabilities.

• Content views are collections of RPMs, container content, or Puppet modules refined with filters and rules. Content views are published and promoted throughout life cycle environments, enabling end-to-end system management. While Satellite 5 used channels and cloning, content views in Satellite 6.1 contain both software and configuration content in one place, greatly simplifying managing the life cycles of systems.

• Integrated with the Red Hat CDN to let users control synchronization of Red Hat content straight from the UI.

• Distribution and federation of provisioning, configuration, and content delivery via Red Hat Satellite Capsule Server.

2.4 Subscription Management

Easily report and map your Red Hat products to registered systems for end-to-end subscription consumption visibility. Easily import and manage the distribution of your Red Hat software subscriptions.

• Report and map your purchased products to registered systems within Red Hat
• Satellite for end-to-end subscription usage visibility.

2.5 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 kick start and Puppet modules. Foreman also provides historical data for reporting, auditing, and troubleshooting.

2.6 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.

2.7 Candlepin

Candlepin is a service within Katello that handles subscription management.

2.8 Pulp

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

2.9 Hammer

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

2.10 REST API

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

2.11 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. For further details on Satellite Server components, refer to documentation

https://access.redhat.com/products/red-hat-satellite/overview

3 Top Performance Considerations

1. Deploy on RHEL 7 - Section 4.3
2. Apache configuration - Section 4.4
3. Adjust passenger tunables on satellite 6.1 and capsule(s) - Section 4.5
4. Candlepin - Section 4.6
5. Pulp - Section 4.7
6. Foreman -Section 4.8
7. Puppet- Section 4.9
8. External capsules - Section 4.10
9. Adjust Apache KeepAlive and Passenger configurations - Section 4.11
10. Hammer timeout configuration: Section 4.12
11. Apache configuration for scale: Section 4.13
12. Postgresql configuration: Section 4.14
13. Storage media for Database workloads - section 4.15
14. Mongodb configuration - Section 4.16
15. Content View -Section 4.17

4 Environment

4.1 Versions Tested

Satellite 6.1		
Red Hat Enterprise Linux	6	7
Kernel	2.6.32-629.el6.x86_64	3.10.0-123.9.3.el7.x86_64
Satellite 6.1	6.1.8	6.1.8
Apache	2.2.15-53	2.4.6-40
Candlepin	0.9.49.12-1	0.9.49.12-1
Elasticsearch	0.90.10-7	0.90.10-7
Foreman	1.7.2.55-1	1.7.2.55-1

Katello	2.2.0.19-1	2.2.0.19-1
Mongodb	2.4.6-2	2.4.9-3
Passenger	4.0.18-21	4.0.18-21
Postgres	8.4.20-6	9.2.1003-1
Pulp	2.6.0.20-1	2.6.0.20-1
Puppet	3.6.2-4	3.6.2-4
Python	2.6.6-64	2.7.5-34
Qpid-dispatch-router	0.4-11	0.4-11
System Ruby - Passenger, Puppet, Dynflow, Smart-proxy, Installer, and Hammer	1.8.7.374-4	2.0.0.598-25
Ruby - Foreman, Katello	2.0.0.23-1	2.0.0.23-1
Tomcat	6.0.24-95	7.0.42-8

Satellite 6.1 Capsule		
Red Hat Enterprise Linux	6	7
Kernel	2.6.32-504.el6.x86_64	3.10.0-327.el7.x86_64
Mongodb	2.4.6-2	2.4.9-3
Pulp	2.6.0.20-1.	2.6.0.20-1
Puppet	3.6.2-1	3.6.2-4
Python	2.6.6-64	2.7.5-34
System Ruby	1.8.7.374-4	2.0.0.598-25
Qpid	0.4-11	0.4-11

4.2 Hardware Considerations

Selecting your hardware is the first component of setting up a performance and scalable Satellite 6 deployment.

CPU

A typical Satellite 6 deployment will run many tasks concurrently, which means that more physical CPU cores available to Satellite 6 or a Capsule will allow for greater throughput of tasks. Balancing throughput and task latency will be specific to each customer's needs, however more CPU cores will improve the scale of a Satellite 6 deployment and should be the first consideration in CPU hardware chosen.

Memory

Adequate memory should be resourced as well. Satellite 6 contains many software components, all of which need to be accounted for when deciding how much memory is required. Memory should be accounted and monitored for the following processes: Apache, ElasticSearch, Foreman, MongoDB, Postgres, Pulp, Puppet, Tomcat, Qpid, and the file system cache. A performant system will not swap even when Apache, Foreman, Puppet and any other software is scaled to its maximum on a single server.

Disk

In addition to disk capacity, Input/Output Operations Per Second (IOPS) must be a consideration. The file system can be partitioned over separate disks as necessary to increase capacity/IOPS on the directories most often accessed. Critical directories for IOPS and monitoring are /var/lib/pulp, /var/lib/pgsql, and /var/lib/mongodb.

Network

Consistently in tests, network hardware has not been found to be a bottleneck before CPU or configuration limits that had been revealed. The hardware tested included a 10Gb network between Satellite 6, Capsules and an emulated Content Delivery Network (CDN). It is likely that the internet connection to the CDN will be a bottleneck but is outside the scope of this brief.

Server Power Management

Servers usually ship with settings in place to conserve power which often leads to less than desirable performance. Prior to installing Red Hat Enterprise Linux the server's BIOS should be configured to allow OS Host Power Control which allows Red Hat Enterprise Linux to control power consumption. Dependent upon customer requirements, performance versus

power consumption might need to be balanced when adjusting any power control settings.

4.3 Red Hat Enterprise Linux

RHEL 6.x vs. RHEL 7.x

Satellite 6 can be installed on both Red Hat Enterprise Linux 6 and on Red Hat Enterprise Linux 7. RHEL 7 is the preferred operating system on which to have Satellite 6 installed. Many enhancements have been made to improve the performance on RHEL 7 together with updated major versions of software including Apache, Postgres and Ruby. These major versions typically have a number of performance improvements that might not be backported into an older version.

Tuned Profiles

Satellite 6 should run with the tuned daemon installed and running with the specific profile that matches its deployment. With RHEL 6 you must install the Tuned package to obtain the performance tuning or equivalent tuning can be done manually. RHEL 7 enables the tuned daemon by default during installation.

```
# service tuned start
# chkconfig tuned on
# tuned-adm profile throughput-performance
```

RHEL 6 (virtual machine)

```
# yum install -y tuned
# service tuned start
# chkconfig tuned on
# tuned-adm profile virtual-guest
```

RHEL 7 (bare-metal):

```
# tuned-adm active
Current active profile: throughput-performance
```

RHEL 7 (virtual machine)

```
# tuned-adm active
Current active profile: virtual-guest
```

If Satellite 6 or a Capsule on RHEL 6 is installed on a virtual machine on Red Hat Enterprise Virtualization, installing rhevm-guest-agent will also deploy Tuned and configure the virtual-guest profile.

It is recommended that Satellite 6 and Capsules on bare-metal run the tuned profile

throughput-performance and if virtualized that they run the virtual-guest profile. If it is not certain the system is currently running the correct profile, check with the tuned-adm active command as shown above. More information about Tuned is located in the Red Hat Enterprise Linux Performance Tuning Guide. Links to the RHEL 7 and RHEL 6 guides are available in Appendix c

4.4 Apache Configuration

KeepAlive Settings

In order to reduce the number of TCP connections and Apache CPU usage, Apache's KeepAlive tunable should be turned on and appropriate values should be set for KeepAliveTimeout and MaxKeepAliveRequests. The recommendation for KeepAliveTimeout is between 2-5 seconds unless there is a latency between Satellite 6 , Capsules or end clients that requires a higher/lower value. The recommendation for MaxKeepAliveRequests is 0 to allow for each connection to request all its content over the same TCP connection. Additionally, there is a reduction in page loading time on the web user interface with KeepAlive on.

Example Satellite 6 Apache configuration tuning:

```
KeepAlive On
MaxKeepAliveRequests 0
KeepAliveTimeout 5
```

Prefork Multi-Processing Module

Apache on Satellite 6 ships with the prefork multi-processing module which scales Apache per process. The default configuration shipped for prefork for Satellite 6 is found in /etc/httpd/conf.d/prefork.conf:

```
<IfModule mpm_prefork_module>
  StartServers       8
  MinSpareServers    5
  MaxSpareServers    20
  ServerLimit        256
  MaxClients         256
  MaxRequestsPerChild 4000
</IfModule >
```

It is likely that Apache's prefork configuration will not require adjustment until after tuning Passenger for the size of the environment. If the environment is large enough and the number of Apache processes is found to be a bottleneck at any point in time, the above values can be adjusted to match the amount of available memory and specific load of the environment

4.5 Passenger Configuration

Passenger configuration is specified within the Apache configuration files and can be used to control the performance, scaling and behavior of Foreman and Puppet.

Global Passenger Configuration Directives
The most important out-of-the box tunable that should be adjusted is the PassengerMaxPoolSize. This should be adjusted to 1.5 * Physical CPU Cores available to the Satellite 6.1 server. This configures the maximum number of processes available for both Foreman and Puppet on Satellite 6.1 and Capsules. PassengerMaxInstancesPerApp can be used to prevent one application from consuming all available Passenger processes.

PassengerMaxRequestQueueSize determines the maximum number of queued requests before Passenger will send a HTTP 503 Service Error to the requester. Depending upon the maximum expected burst of requests, it will be necessary to adjust the Passenger Queue. A queued request consumes an Apache process and setting the queue above Apache's MaxClients/ServerLimit configuration will force queued requests to wait within the ListenBacklog queue in Apache. This will also block Apache from serving any other requests that do not require Foreman or Puppet. It is recommended to adjust PassengerMaxRequestQueueSize to the maximum expected burst in Foreman/Puppet traffic, but below Apache's MaxClients/ServerLimit configuration thus allowing other requests to complete without waiting for Passenger to free up Apache processes such as a client downloading content by running yum install or yum update.

Application Specific Configuration Directives
PassengerMinInstances should be configured to the minimum number of instances required at start up. When a burst of requests comes in, Passenger will spawn additional application processes up to PassengerMaxPoolSize or PassengerMaxInstancesPerApp if set. The preloader handles spawning the new applications and thus should be available all the time to reduce latency spent waiting for a new application process. This can be accomplished by disabling the preloader's time out with PassengerMaxPreloaderIdleTime. Enabling the preloader means more memory will be consumed to keep a preloader process ready.

If the environment has an issue with continuously growing in memory from either Foreman or Puppet, it is recommended to set PassengerMaxRequests such that those processes will be recycled to free up memory. Preventing the Satellite 6 server from swapping is critical to its performance and Scalability.

An example tuned configuration of Passenger for Satellite 6.1 would resemble the following:

Global Passenger configuration: /etc/httpd/conf.d/passenger.conf

```
LoadModule passenger_module modules/mod_passenger.so
```

```
<IfModule mod_passenger.c>
 PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-
4.0.18/lib/phusion_passenger/locations.ini
 PassengerRuby /usr/bin/ruby
 PassengerMaxPoolSize 24
PassengerMaxRequestQueueSize 200
PassengerStatThrottleRate 120
</IfModule>
```

Foreman Passenger application configuration: /etc/httpd/conf.d/05-foreman-ssl.conf

```
  PassengerAppRoot /usr/share/foreman
  PassengerRuby /usr/bin/ruby193-ruby
  PassengerMinInstances 6
  PassengerStartTimeout 90
  PassengerMaxPreloaderIdleTime 0
  PassengerMaxRequests 10000
  PassengerPreStart https://example.com
```

Puppet Passenger application configuration: /etc/httpd/conf.d/25-puppet.conf

```
  PassengerMinInstances 6
  PassengerStartTimeout 90
  PassengerMaxPreloaderIdleTime 0
  PassengerMaxRequests 10000
  PassengerPreStart https://example.com:8140
```

Using the `passenger-status` command the Foreman and Puppet processes spawned by Passenger can be obtained to confirm the `PassengerMaxPoolSize`.

Example `passenger-status` output:

```
#Version : 4.0.18
Date  : 2016-04-29 01:20:35 -0400
Instance: 30548
----------- General information -----------
Max pool size : 24
Processes   : 19
Requests in top-level queue : 0

----------- Application groups -----------
/usr/share/foreman#default:
  App root: /usr/share/foreman
  Requests in queue: 0
  * PID: 31274   Sessions: 0        Processed: 31     Uptime: 10m 56s
      CPU: 0%     Memory  : 264M    Last used: 1m 57s ago
  * PID: 31303   Sessions: 0        Processed: 40     Uptime: 10m 56s
      CPU: 1%     Memory  : 532M    Last used: 1m 42s ago
  * PID: 31338   Sessions: 0        Processed: 30     Uptime: 10m 54s
      CPU: 0%     Memory  : 263M    Last used: 1m 41s ago
```

```
  * PID: 31366   Sessions: 0         Processed: 32     Uptime: 10m 53s
    CPU: 0%      Memory  : 268M     Last used: 1m 48s ago
  * PID: 31394   Sessions: 0         Processed: 61     Uptime: 10m 53s
    CPU: 2%      Memory  : 269M     Last used: 1m 39s ago
  * PID: 31422   Sessions: 0         Processed: 20     Uptime: 10m 53s
    CPU: 0%      Memory  : 262M     Last used: 1m 53s ago
  * PID: 31992   Sessions: 0         Processed: 15     Uptime: 2m 12s
    CPU: 2%      Memory  : 252M     Last used: 1m 59s ago
  * PID: 32070   Sessions: 0         Processed: 15     Uptime: 2m 12s
    CPU: 2%      Memory  : 250M     Last used: 1m 59s ago
  * PID: 32202   Sessions: 0         Processed: 18     Uptime: 2m 12s
    CPU: 6%      Memory  : 253M     Last used: 1m 54s ago
  * PID: 32364   Sessions: 1         Processed: 24     Uptime: 2m 11s
    CPU: 4%      Memory  : 249M     Last used: 2s ago
  * PID: 32516   Sessions: 0         Processed: 6      Uptime: 2m 11s
    CPU: 2%      Memory  : 241M     Last used: 1m 46s ago
  * PID: 32625   Sessions: 0         Processed: 21     Uptime: 2m 10s
    CPU: 5%      Memory  : 244M     Last used: 1m 43s ago
  * PID: 32655   Sessions: 0         Processed: 14     Uptime: 2m 10s
    CPU: 2%      Memory  : 235M     Last used: 1m 50s ago
  * PID: 32697   Sessions: 1         Processed: 12     Uptime: 2m 9s
    CPU: 4%      Memory  : 233M     Last used: 1m 57s ago
  * PID: 32726   Sessions: 0         Processed: 14     Uptime: 2m 9s
    CPU: 2%      Memory  : 220M     Last used: 1m 48s ago
  * PID: 304     Sessions: 1         Processed: 1      Uptime: 2m 8s
    CPU: 1%      Memory  : 216M     Last used: 2m 8s ago
  * PID: 333     Sessions: 0         Processed: 6      Uptime: 2m 8s
    CPU: 2%      Memory  : 218M     Last used: 2m 7s ago
  * PID: 396     Sessions: 0         Processed: 7      Uptime: 2m 7s
    CPU: 2%      Memory  : 222M     Last used: 1m 55s ago
  * PID: 436     Sessions: 0         Processed: 8      Uptime: 2m 7s
    CPU: 2%      Memory  : 222M     Last used: 1m 59s ago
```

View the reported memory usage of Passenger using the passenger-memory-stats command.

Example passenger-memory-stats output:

```
# passenger-memory-stats
Version: 4.0.18
Date   : 2016-04-29 01:21:32 -0400

---------- Apache processes -----------
PID    PPID   VMSize     Private  Name
---------------------------------------
300    30548  193.6 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
367    30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
```

```
368   30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
372   30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
374   30548  193.4 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
375   30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
379   30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
381   30548  193.6 MB   3.0 MB   /usr/sbin/httpd -DFOREGROUND
384   30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
30548  1    191.1 MB   0.6 MB   /usr/sbin/httpd -DFOREGROUND
30629  30548  1071.2 MB  60.0 MB  (wsgi:pulp)   -DFOREGROUND
30630  30548  1135.2 MB  64.4 MB  (wsgi:pulp)   -DFOREGROUND
30631 30548 1135.2 MB 63.2 MB (wsgi:pulp) -DFOREGROUND
30632  30548  460.2 MB   14.1 MB  (wsgi:pulp-cont -DFOREGROUND
30633  30548  460.2 MB   14.1 MB  (wsgi:pulp-cont -DFOREGROUND
30634  30548  524.2 MB   16.1 MB  (wsgi:pulp-cont -DFOREGROUND
30635  30548  786.0 MB   47.1 MB  (wsgi:pulp_forg -DFOREGROUND
30636  30548  850.0 MB   49.1 MB  (wsgi:pulp_forg -DFOREGROUND
30637  30548  786.0 MB   47.1 MB  (wsgi:pulp_forg -DFOREGROUND
30659  30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
30663  30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
31998  30548  193.5 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
32609  30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
32611  30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
32679  30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
32681  30548  193.4 MB   2.7 MB   /usr/sbin/httpd -DFOREGROUND
32682  30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
32685  30548  193.5 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
32756  30548  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
32760  30548  193.6 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
32762  30548  193.6 MB   3.0 MB   /usr/sbin/httpd -DFOREGROUND
32766  30548  193.6 MB   2.8 MB   /usr/sbin/httpd -DFOREGROUND
### Processes: 32
### Total private dirty RSS: 438.06 MB


-------- Nginx processes --------

### Processes: 0
### Total private dirty RSS: 0.00 MB


------ Passenger processes ------
PID   VMSize      Private   Name
--------------------------------
304   1920.1 MB  217.0 MB  Passenger RackApp: /usr/share/foreman
333   1920.2 MB  219.0 MB  Passenger RackApp: /usr/share/foreman
396   1920.3 MB  222.2 MB  Passenger RackApp: /usr/share/foreman
436   1920.4 MB  222.7 MB  Passenger RackApp: /usr/share/foreman
30638  209.8 MB   0.3 MB      PassengerWatchdog
30641  1840.9 MB  3.7 MB      PassengerHelperAgent
```

```
30647  214.1 MB   0.9 MB      PassengerLoggingAgent
30795  676.6 MB   172.0 MB  Passenger AppPreloader: /usr/share/foreman
31274  1917.7 MB  264.4 MB  Passenger RackApp: /usr/share/foreman
31303  2238.8 MB  532.5 MB  Passenger RackApp: /usr/share/foreman
31338  1917.9 MB  263.3 MB  Passenger RackApp: /usr/share/foreman
31366  1919.0 MB  268.2 MB  Passenger RackApp: /usr/share/foreman
31394  1919.1 MB  269.7 MB  Passenger RackApp: /usr/share/foreman
31422  1918.2 MB  263.0 MB  Passenger RackApp: /usr/share/foreman
31992  1918.3 MB  252.6 MB  Passenger RackApp: /usr/share/foreman
32070  1919.3 MB  250.6 MB  Passenger RackApp: /usr/share/foreman
32202  1918.4 MB  253.7 MB  Passenger RackApp: /usr/share/foreman
32364  2190.3 MB  474.5 MB  Passenger RackApp: /usr/share/foreman
32516  1919.6 MB  241.9 MB  Passenger RackApp: /usr/share/foreman
32625  1919.7 MB  244.7 MB  Passenger RackApp: /usr/share/foreman
32655  1919.8 MB  235.9 MB  Passenger RackApp: /usr/share/foreman
32697  1919.9 MB  233.7 MB  Passenger RackApp: /usr/share/foreman
32726  1920.0 MB  220.2 MB  Passenger RackApp: /usr/share/foreman
### Processes: 23
```

All of the relevant Passenger tunables can be found in Appendix E and further documentation can be found in the references.

4.6 Candlepin

Complexity around subscriptions can change the amount of latency required to complete a registration. The process to register involves Candlepin and Foreman and therefore is subject to the number of Foreman processes and Passenger queue size. A method to determine the latency required for a specific environment would be to time subscription-manager registrations such as:

```
# time subscription-manager register --org="Default_Organization"
--activationkey="ak-dev"
```

By timing a specific registration and determining the minimum/average/maximum timings, the capacity of a specific environment can be determined. The default Passenger configuration with Satellite 6.1 allows six concurrent registrations if Foreman consumes all of the processes determined by PassengerMaxPoolSize and all application processes are preloaded. If there is only one process spawned, then additional preloader latency will be added to your registration time. More concurrent registrations experience additional latency due to queuing for an available Foreman process. Any other tasks or workloads that also involve Foreman will also wait on the queue and add delay to any other concurrent registrations.

4.7 Pulp

Pulp handles content management of RPM content and Puppet modules. Pulp is responsible for publishing content views and creating local repositories for Capsules and clients from which to sync content. Pulp's performance and scale to serve content relies on the configuration of Apache and its configuration files.

Worker Concurrency

Pulp's default behavior is to start 8 workers. The workers are responsible for asynchronous tasks such as syncing and publishing content. The number of workers is adjustable in /etc/default/pulp_workers by changing the value of PULP_CONCURRENCY. If many repositories are attempted for syncing at a single point in time, then more workers can consume Satellite 6.1 resources. This can starve other components of Satellite 6.1 and thus it can be necessary to adjust the concurrency level of pulp in an environment with an on-going concurrent workload such as Puppet.

NFS

Red Hat Satellite 6 uses /var/lib/pulp to store and manage repository content. Pulp uses mongodb which has issues with NFS. It is not recommended to run pulp on NFS. Red Hat recommends the usage of high-bandwidth, low-latency storage for the /var/lib/pulp file system. Red Hat Satellite has many operations, that are IO-intensive so usage of high-latency, low-bandwidth storage could potentially have issues with performance degradation.

Store content

It is recommended to mount pulp directory onto a large local partition that you can easily scale . Use Logical Volume Manager (LVM) to create this partition.

4.8 Foreman

Foreman is a ruby application running inside the Passenger application server. Foreman's performance and Scalability are directly affected by the Apache and Passenger configuration. Follow the recommendations discussed in Section 4.11. In addition to provisioning, Foreman processes handle UI and API requests. Turning Apache's KeepAlive on will improve the page load time of the user interface and a properly configured tuned profile will improve the response time of the CLI/API as represented in the commands shown in Section 4.11

4.9 Puppet

Like Foreman, Puppet is a ruby application running inside the Passenger application server. There are several factors in Puppet which affect the overall performance and Scalability of your Satellite 6.1 deployment.

Runinterval – A non-deterministic runinterval that does not distribute the load throughout the interval will cause scaling problems and errors within a Puppet environment. Evenly distributing the load will allow a system to reliably scale and handle more requests with less spikes. Depending upon the scale of an environment, runinterval can be distributed by:

- Puppet splay – Turn on splay for each Puppet client. This adds randomization to runinterval, however this does not accomplish a deterministic runinterval.
- Cronjob – Run each Puppet agent as a cron job rather than a daemon. This makes a runinterval deterministic however at scale this becomes difficult to manage when adding and removing clients.
- Deploy a separate entity to manage when a Puppet agent run occurs.

Passenger – Configure Passenger to allow Puppet to have more processes. This allows for greater concurrency by providing more processes to handle Puppet requests.

Manifest complexity – Measure manifest compilation time and seek to reduce it if possible. Time Puppet runs without caching requests to see the impact each specific manifest in an environment has on Satellite 6.1 and/or Capsules. In order to test a greater number of catalogs rapidly, invoke the Puppet API with a curl command to generate a similar workload and benchmark the specific manifest/catalog. Reducing the complexity of a manifest will reduce the load and improve Scalability.

Other Puppet interactions – Measure other Puppet interactions that a specific environment performs. Other interactions will place load on Satellite 6 and Capsules such as submitting facts, serving files and submitting a report. All these interactions have an additional cost.

Run RHEL 7 – Analysis of Puppet runs on RHEL 7 have shown greater Scalability and improved performance over RHEL 6 on the same exact hardware.

4.10 External Capsules

External Capsules allow a Satellite 6 deployment to scale out and provide services local to the machines that are managed by them.

Advantages of an external Capsule:

- Reduces the number of HTTP requests on Satellite 6.1.
- Places resources closer to end clients to reduce latency in requests.

Factors to consider for when to use an external Capsule:

- Runinterval - Timing between Puppet agent applies and even spread of workload over the entire interval
- Client workload - Amount of work for each Puppet client during a Puppet agent run
- Hardware/Configuration - Amount of available resources for Puppet

The determination of when to use an external Capsule vs an integrated Capsule depends on hardware, configuration, and workload. This should be planned against the Puppet requirements as a number of variables in the Puppet workload will directly affect the Scalability of Satellite 6. In Section 5.5 testing results, Satellite 6 was scaled to 2,000-2,250 clients spread evenly over a 30 minute run-interval. Raising the run-interval will directly increase the capacity but at a cost of increasing the interval between which Puppet applies the configuration. Reducing the run-interval consequently reduces the capacity. If the clients are not spread evenly, a large group of clients can fill the Passenger queue and block other requests while leaving the Satellite 6 server under-utilized at other times. The amount of work each puppet client has to perform in order to complete a puppet run will also change Scalability. Raising the configured number of Puppet processes will improve Scalability if there is physical hardware resources available. Due to these variables it would not be constructive to provide a single one-size-fits all recommendation on when to move to an external Capsule. The best recommendation is the benchmark a specific puppet workload to determine that specific workloads Scalability.

4.11 Apache/Passenger Configuration on capsule:

The same considerations for hardware for Satellite 6 apply directly to a Capsule. A virtualized Capsule provides the advantage of tuning the number of vCPUs and available memory as long as the Capsule is not co-located on a host with virtual machines that over commit the host's resources. Apache and Passenger configuration considerations also apply directly to the Capsule but in the context of Puppet.

Example Capsule Apache configuration tuning:

```
KeepAlive On
MaxKeepAliveRequests 0
```

```
KeepAliveTimeout 5
```

Example Capsule Passenger configuration: /etc/httpd/conf.d/passenger.conf

```
LoadModule passenger_module modules/mod_passenger.so
<IfModule mod_passenger.c>
  PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-
4.0.18/lib/phusion_passenger/locations.ini
  PassengerRuby /usr/bin/ruby
  PassengerMaxPoolSize 6
  PassengerMaxRequestQueueSize 200
  PassengerStatThrottleRate  120
</IfModule>
```

Example Capsule Puppet Passenger configuration tuning: /etc/httpd/conf.d/25-puppet.conf

```
 PassengerMinInstances 2
 PassengerStartTimeout 90
 PassengerMaxPreloaderIdleTime 0
 PassengerMaxRequests 10000
 PassengerPreStart https://example-capsule.com:8140
```

4.12 Scale: Hammer timeout

During scale of Capsules or content hosts or Content views ..etc cause hammer API request timeout on satellite6.1. Set time out as -1 in /etc/hammer/cli.modules.d/foreman.yml to set no timeout

```
:request_timeout: -1 #seconds
```

4.13 Scale: Apache configuration

Validated until 2000-2500 content hosts at scale. During scale Capsules or content hosts or Content views user run short of file descriptors. Recommended to increase Apache open files .

```
 mkdir  -p /etc/systemd/system/httpd.service.d/
cat /etc/systemd/system/httpd.service.d/limits.conf
[Service]
LimitNOFILE=65536
```

And reload daemon and restart Apache

```
systemctl daemon-reload
systemctl restart httpd
```

Same can be validated with

```
[root@satserver ~]# systemctl status httpd | grep 'Main PID:'
Main PID: 11628 (httpd)
```

```
[root@satserver ~]# grep -e 'Limit' -e 'Max open files' /proc/11628/limits
```

Limit	Soft Limit	Hard Limit	Units
Max open files	65536	65536	files

4.14 Postgresql

Postgresql requirements depends on various requirements like number of organizations, environments, registered systems, and content views ..etc. While registering content hosts at scale to satellite server, user need to change buffer_cache in shared_buffers need to set appropriately. Recommended to set 256M for buffer_cache for large scale deployments.

```
shared_buffers = 256MB
```

4.15 Storage media for Database workloads

Pulp, mongodb are good candidates for improved disk performance. Performance improvements are noticed when these works loads are switched from spinning media to ssd.

Red Hat recommends the usage of high-bandwidth, low-latency storage for the pulp and mongodb. Red Hat Satellite has many operations that are IO-intensive so usage of high-latency, low-bandwidth storage could potentially have issues with performance degradation.

4.16 Mongodb

Avoid NFS

MongoDB does not use conventional I/O to access the data files: it uses mmap(). NFS does not perform great with mmap() call, especially with the way that MongoDB uses it (re-mapping all of the data files 10 times per second). It's recommended not to run mongodb on NFS to avoid performance issues.

Size

The storage requirements depends on the number of packages and content views for your Satellite environment. Content view publishing to different versions to consume more monogdb space. Recommend to effective capacity planning based on number of packages and published versions.

4.17 Content View

Red Hat Satellite 6.1 provides users with the ability to create *content views*. Content views act as a snapshot of one or more repositories and/or puppet modules. Content Views are ‘published’ in order to lock their contents in place. Publishing creates a new version of the Content View. Content Views can be promoted & cloned to different life cycle environments (Dev, Test, Production). Content view uses a symbolic link to the Media Library stored in the pulp directory. In addition, each repository in a content view contains metadata about the content belonging to the content view. This means a content view using a minimal number of packages uses a small amount of storage. However, the storage size adds up once you use multiple content views and a large number of packages per view. While each content view is published, it's going to consume more and more storage. To reduce storage consumption, remove old and unused versions of content views. If you are not using an old content view in a life cycle environment, monitor nodes & published directories under pulp.

5 Results

5.1 Tuned profiles

Testing of Tuned on Satellite 6.1 on RHEL 6 shows that throughput-performance profile consistently provided a reduction in latency with Satellite 6.1 API driven hammer commands as well as syncing of content.

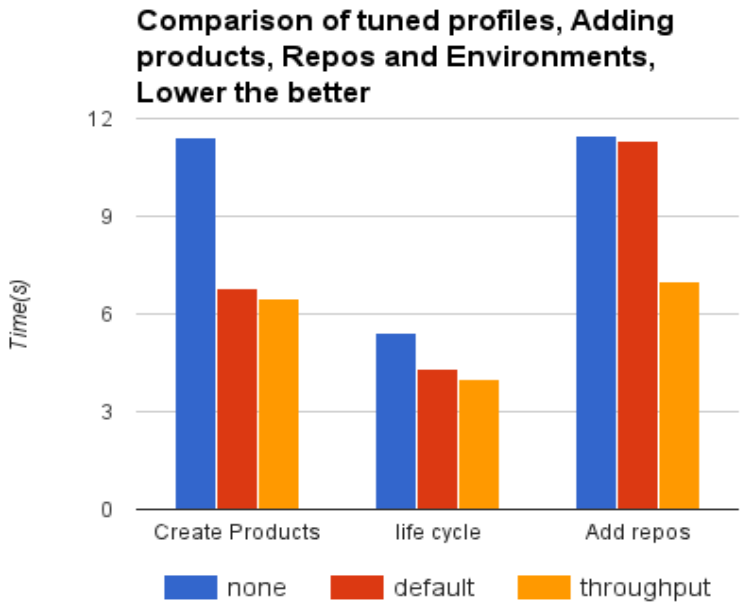

Figure 5.1 Comparison of Tuned profiiles

The above graph shows the improvements in response time to hammer commands by comparing Tuned off, the default profile, and throughput-performance profile on Satellite 6.1 on RHEL 6.

Synchronization of RHEL5 repo to Satellite 6.1 on RHEL 6.x

The above graph shows the improvements in response time to hammer commands by comparing Tuned off, the default profile, and throughput-performance profile on Satellite 6 on RHEL 6.

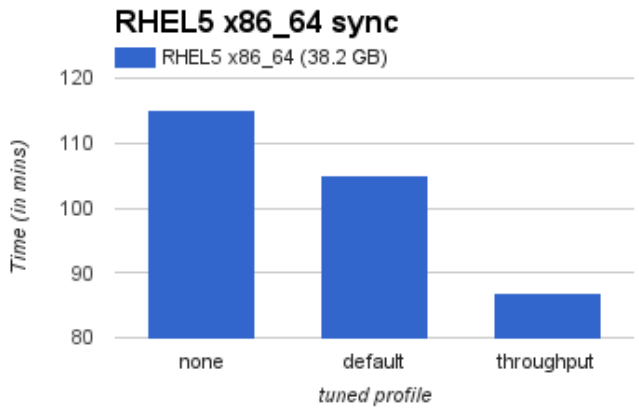

Figure 5.2 RHEL 5 x86_64 repo sync with tuned profiles

The graph above is a comparison between Tuned off, the default profile, and throughput-performance profile on Satellite 6.1 during sequential syncing of content.

5.2 Satellite on RHEL 7

Concurrent synchronization of RHEL5,6,7 repositories to Satellite 6.1 on RHEL 6 & RHEL 7

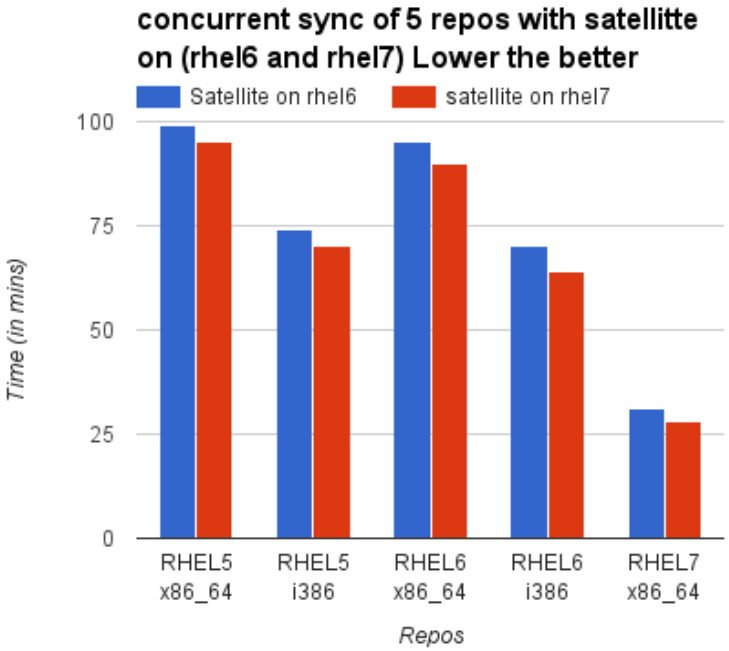

Figure 5.3 Concurrent Sync of 5 repos

The graph above is a comparison between running Satellite 6.1 on RHEL 6 & RHEL 7 during concurrent syncing of content.

5.3 Storage Media preference for Database

Concurrent synchronization of RHEL5,6,7 repos to Satellite 6.1 with pulp & mongodb on SSD, HDD

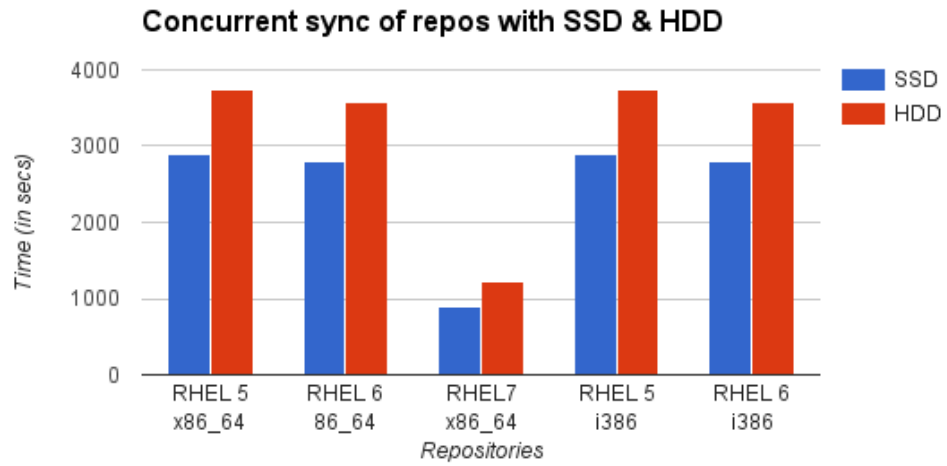

Figure 5.4 Concurrent sync of repos with SSD & HDD

The graph above is a comparison between running Satellite 6.1 on RHEL 7 with pulp and mongdb on SSD during concurrent syncing of content.

5.4 Candlepin Concurrency

Figure 5.5 Concurrent Host Registrations

As shown in the graph above, additional concurrent registrations result in a longer time required to register. Once the number of concurrent registrations rises above the maximum queue size (100 in this example) there can be inconsistencies in what clients Satellite 6.1 shows and what the client subscription-manager reports as registered.

It is also important to note that running subscriptions at the maximum rate leaves no room for other tasks/workloads that require a Foreman process. Avoiding the Passenger queue is essential to successful system registrations at scale as shown in the concurrency test results. The more complex a system registration is, the latency required to complete a registration will be larger and therefore reduce the capacity and rate at which systems can register to Satellite 6.1.

5.5 Pulp Content Syncs

The latency required for a repository to sync in an environment where network bandwidth is not a bottleneck is largely dependent upon the number of files rather than the file size or aggregate size of a repository. This is evident in the graph below, as a repository of the same number of files but only a quarter of the file size requires nearly the same amount of time to sync. This is a especially evident in smaller repositories where the effect of file size has little to no impact on the latency of syncing.

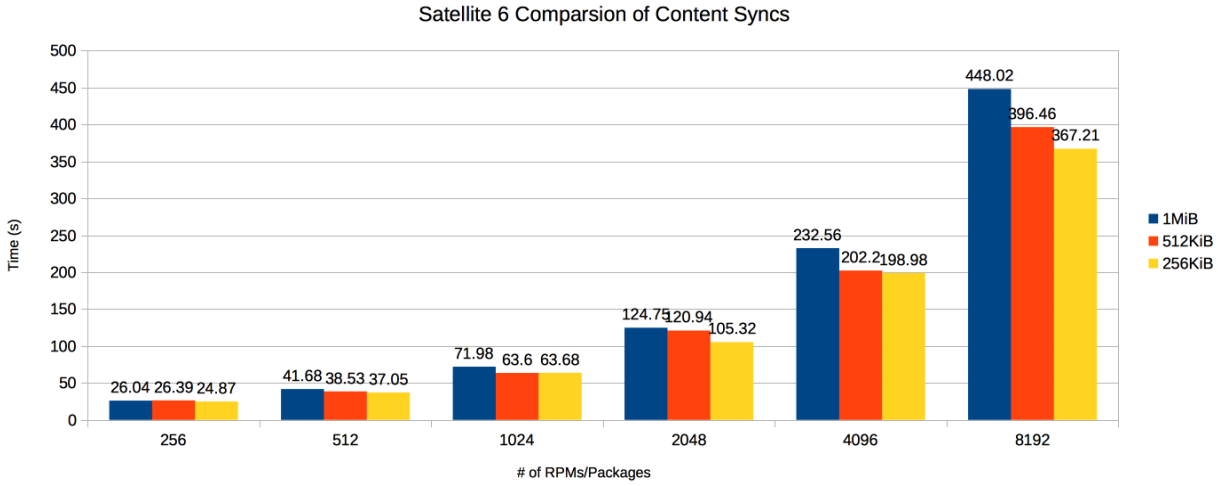

Figure 5.6 Comparison of content syncs

The color of the bar indicates the size of individual packages. The x-axis shows the number of packages. As the number of packages grows, the difference in sync latency due to file size becomes more apparent. At 8192 packages, Pulp synced an aggregate of 8GiB of 1MiB packages in 448.02s and an aggregate of 4GiB of 512KiB packages in 396.46s.

5.6 Puppet Integrated Capsule

Puppet Scalability was tested against Satellite 6's Integrated Capsule by testing specific rates at which multiple Puppet clients can complete a run. Dependent upon the life-cycle of the client and changes to the Puppet configuration, the maximum scalability of an Integrated Capsule will vary. In other words, steady state operations of Puppet checking an existing configuration is less resource intensive than Puppet applying a new configuration.

This test per-provisioned 400 clients and kicked off each with a specific sleep time (Indicated by the color of the line in the legend) between agents generating a steady stream of Puppet clients checking in. The runtime of each Puppet agent was timed and graphed in the below graphs. The initial Puppet agent applying a configuration was timed and the test was run a second time to determine the runtime of Puppet agent at its steady state. The Apache Puppet class was chosen to provide a workload for the catalog compilation.

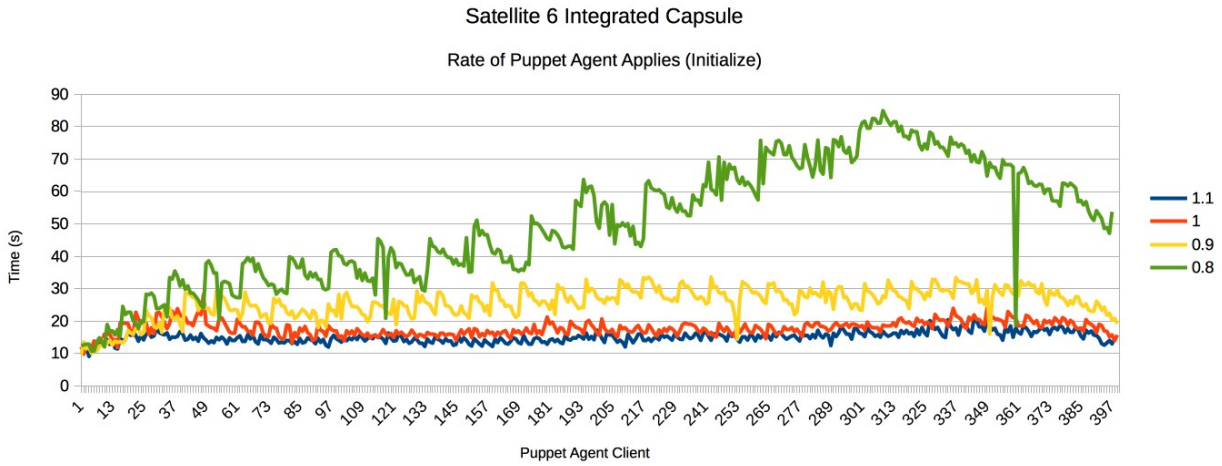

Figure 5.7 Satellite 6 Integrated capsule

For the initial Puppet agent run, a 0.8 second interval between Puppet runs shows a stepped growth in Puppet agent run time. At that rate of growth eventually the number of queued requests will approach the maximum allowable configured from Passenger (PassengerMaxRequestQueueSize) and the agent will receive HTTP 503 service error as a response. At smaller rates between 0.9-1 second, there is no growth which indicated the maximum sustained number of new clients that could attempt to apply this Puppet configuration was between 2,000-1,800 for a run-interval of 30 minutes.

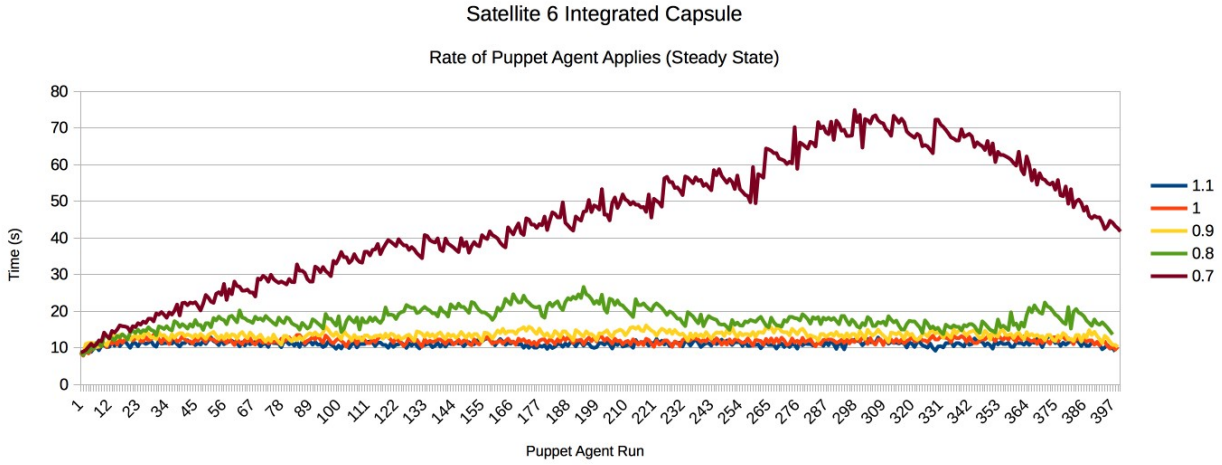

Figure 5.8 Rate of puppet agent Applies

For steady state operations, a 0.7 second interval shows the run time for each successful Puppet agent run growing. This indicates that requests are queuing and the rate at which this

configuration can receive Puppet agent runs has been exceeded. At the 0.8 second interval, Satellite 6 Integrated Capsule scales to 2,250 Puppet agents assuming a 30 minute run-interval per Puppet agent using this specific configuration during steady state operations (No changes to configurations).

It is important to understand that the previous tests were conducted on a default installation of Satellite 6 without tuning the maximum number of Passenger processes. Additionally, it is important to note that running an environment at or near 100% capacity is a recipe for a disaster. If anything adds more latency to each puppet request, there is a risk of the queue growing and HTTP 503 errors occurring when it fills. It would be better to scale out Puppet via external Capsules to allow more room for requests to arrive for Foreman at the Satellite 6 machine.

5.7 Puppet External Capsule

Puppet Scalability for multiple external Capsules was tested through the Puppet API. The workload would upload facts, compile the specific client's Puppet catalog and submit a report. This workload was run on both RHEL 7 and RHEL 6 testbeds to determine improvements. Using a specific manifest built of ten different Puppet classes, Satellite 6 on RHEL 6 was scaled to 16 Capsules each handling 1,500 Puppet catalog compilations distributed over a 30 minute run interval before the response time for the catalog compilation and failures began to occur. The same manifest tested in a RHEL 7 testbed scaled to 20 Capsules at 1500 Puppet catalog compilations per Capsule without failure or significant response time growth. The number of catalog compilations per Capsule had to be raised until failure as the number of Capsules to test against for the RHEL 7 tests were exhausted. Both tests were completed without tuning the Passenger configuration that ships with Satellite 6. For this manifest and hardware, RHEL 7 scaled approximately 36% over what RHEL 6 was able to accomplish. In addition to the scale of catalogs compiling, the latency required to compile the same catalog was significantly reduced.

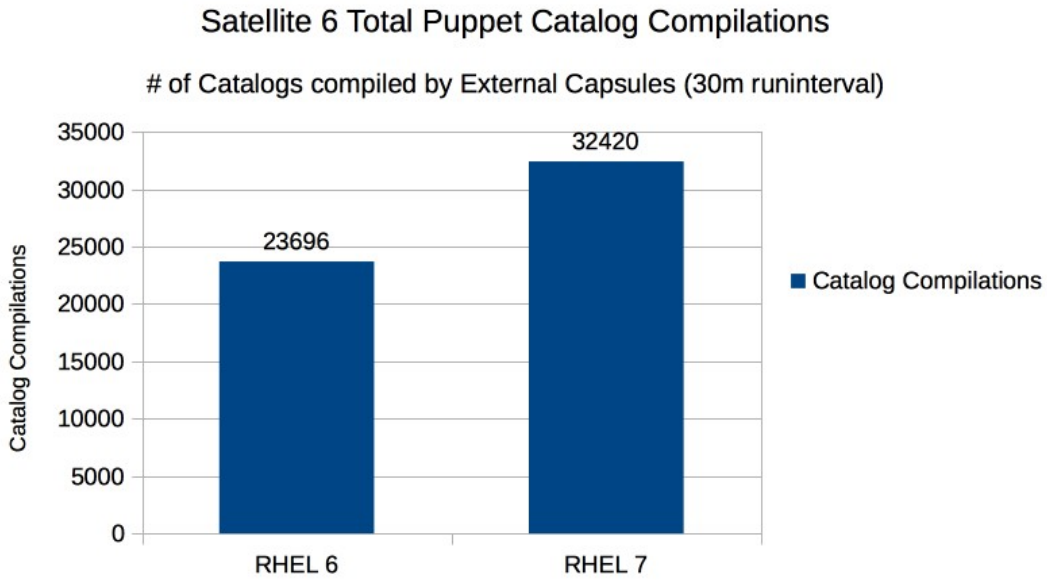

Figure 5.9 number of catalogs compiled by capsule

This workload included submitting facts, compiling a catalog consisting of ten Puppet classes, and submitting a report afterwards.

6 Conclusion

Satellite 6.1 is a robust platform for life cycle management of systems that combines the best of many open source projects. Efficiently planning, tuning and monitoring of system resources in Satellite 6.1 allows the combined platform to extend itself for Scalability and performance.

The goal of this performance brief is to provide basic guidelines and considerations for a performance and scaled Satellite 6.1 environment. Since each Satellite 6.1 deployment will vary in its end goals, it is impossible to provide a one-size-fits-all configuration. Individual tuning will expose the most resources on the tasks which are most important for a specific Satellite 6 deployment. Section 2, Top Performance Considerations, identifies the highest priority items to consider when tuning and deploying Satellite 6 for optimal performance.

7 Recommendations

1. Prefer RHEL 7 to deploy satellite server
2. If satellite is deployed on Bare metal, set tuned profile as Throughput-performance
   If Satellite is deployed on VM, set tuned profile as virtual-guest
3. Set KeepAlive on in apache config to reduce number of TCP connections and cpu usage as mentioned in section 4.4
4. Set PassengerMaxPoolSize to 1.5 X Physical cores available on satellite server to increase maximum number of process available for foreman & Puppet as mentioned in Section 4.5
5. Avoid using NFS for pulp & mongodb to avoid performance impact
6. Leverage external capsules for scale & satellite to manager capsules
7. Set request_timeout to -1 in /etc/hammer/cli.modules.d/foreman.yml to avoid hammer timeouts at scale
8. Increase Apache file descriptors at scale as mentioned in Section 4.13
9. Increase shared_buffer to 256M in /var/lib/pgsql/data/postgresql.conf and restart katello services at Scale.

Appendix A: Revision History

Revision 1.1		Pradeep Surisetty
Initial Release		

Appendix B: Contributors and Reviewers

Contributor	Title	Contribution
Andy Bond	Manager	Review
Tim Wilkinson	Sr. Software Engineer	Review
Mike McCune	Manager, Software Engineering	Review
Rich Jerirdo	Principal Technical Product Marketing Manager	Review
Alex Krzos	Sr. Software Engineer	Review
Deepthi Dharwar	Principal Software Engineer	Review
Brian Riordan	Performance Engineering Director	Review
Douglas Shakshober	Technical Director , Performance Engineering	Review

Appendix C: References

1. https://access.redhat.com/documentation/en-US/Red_Hat_Satellite/
2. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Performance_Tuning_Guide/index.html
3. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/index.html
4. http://httpd.apache.org/docs/2.4/
5. http://httpd.apache.org/docs/2.2/
6. https://www.phusionpassenger.com/documentation/Users%20guide%20Apache.html
7. http://www.candlepinproject.org/docs/candlepin
8. http://www.pulpproject.org/docs/
9. http://theforeman.org/manuals/1.6/index.html
10. https://docs.puppetlabs.com/puppet/
11. https://access.redhat.com/solutions/1375253

Appendix D: Significant Apache Tunables

KeepAlive – Allows multiple requests to be sent over a single TCP connection avoiding the start up / tear down costs of a TCP socket when there are multiple requests occurring rapidly.

KeepAliveTimeout – Adjustment for how long keep alive connections are left open.

MaxKeepAliveRequests – Maximum number of requests allowed per connection when KeepAlive is on.

StartServers – Number of processes created on server startup.

MinSpareServers – Minimum number of idle child server processes.

MaxSpareServers – Maximum number of idle child server processes.

ServerLimit – Sets the maximum number for MaxClients/MaxRequestWorkers.

MaxClients – Maximum number of concurrent requests that will be served. This tunable has been renamed in Apache 2.4 to MaxRequestWorkers however the old name is still supported.

MaxRequestsPerChild – Maximum number of requests a child process will handle before terminating. This tunable is used to prevent a process from continuously growing in memory usage. This tunable has been renamed in Apache 2.4 to MaxConnectionsPerChild however the old name is still supported.

Appendix E: Significant Passenger Tunables

PassengerMaxPoolSize – Maximum number of application processes that can concurrently handle requests.

PassengerMaxInstancesPerApp – Prevent a single application from monopolizing the maximum number of application processes Passenger will spawn.

PassengerMinInstances – Insures a minimum number of application processes are available after an application is first accessed.

PassengerPoolIdleTime – Closes an idle application to conserve memory after a specified

amount of time.

PassengerMaxPreloaderIdleTime – Determines how long the application preloader will exist if idle.

PassengerStartTimeout – If a Passenger application fails to start within the timeout, forcefully kill it.

`PassengerMaxRequests` – Max number of requests before Passenger will restart an application process. This is used as a workaround for memory leak prone applications to prevent an application from consuming too much memory.

`PassengerStatThrottleRate` – Adjusts the rate at which Passenger checks for application startup files and restart.txt.

`PassengerPreStart` – This tunable is used to start an application whenever Apache is restarted.

`PassengerHighPerformance` - Enables Passenger's high performance mode at expense of specific Apache modules (mod_proxy, mod_rewrite, mod_autoindex, others...) from working correctly.

`PassengerMaxRequestQueueSize` – Determines the maximum number of requests that will be queued when all application processes are handling a request. If the queue is full, Apache will return an HTTP 503 Error indicating that the server is too busy to queue the request.