Red Hat Performance Briefs

Performance & Scale Tuning of Satellite 6.2 and capsules

Pradeep Surisetty

Jan Hutar

Archit Sharma

Version 1.0

RHEL 7.2

September 2016

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

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 is constantly evolving and 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 capable of 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 enables 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. Satellite 6.2 is the third major release of the next generation Satellite with a raft of improvements that continue to narrow the gaps in functionality found in Satellite 5 in many critical areas of the product. Satellite 6.2 provides many refinements and new functionality to serve the needs of the Linux System Engineer and Administrator in the following areas:

- Content Management
- Subscription Management
- Provisioning Management
- Configuration management
- Remote Execution (REx) Management
- OpenSCAP Security Management
- Scalability
- Docker 2.0 Container Management

This document provides basic guidelines and considerations for tuning Red Hat Satellite 6.2 for performance and Scalability. Many factors drive the performance of a Satellite 6.2 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 6.2 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 6.2 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/red-hat-satellite/?version=6.2

2 Satellite 6.2 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 Content 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 development 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 through life cycle environments, enabling end-to-end system management. While Satellite 5 used channels and cloning, content views in Satellite 6.2 contain both software and configuration content in one place, greatly simplifying the process of managing the life cycles of systems.
- Integrated with the Red Hat CDN to let users control synchronization of Red Hat content straight from the web UI.
- Distribution and federation of provisioning, configuration, and content delivery via Red Hat Satellite Capsule Server.

2.2 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.

2.3 Provisioning Management

Provision RHEL host systems on bare-metal, on virt fabrics, and on private and public cloud

systems, with dynamic KickStart and seamless hand-off to Puppet Operations.

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

2.4 Configuration Management

Mange RHEL host systems with Puppet Version 3.8, including interoperability with GIT repos, the Puppet Forge, Hiera, R10K, and Puppet Enterprise.

2.5 Remote Execution (REx) Management

Use new and powerful workflows to control one or multiple managed hosts. The new REx feature leverages all aspects of Satellite 6.2, including dynamic templates for scripting commands with smart variables, dynamic host collections, scheduling of remote commands, errata updating, and host services inquiry. All communication to managed hosts occurs over SSH, orchestrated from the nearest Capsule with integrated Key and SUDO management.

2.6 OpenSCAP Security Management

Scan managed hosts for common vulnerabilities and exposures (CVE). OpenSCAP now uses the Capsule for gathering host reports and forwarding them to the Satellite Server, where they are decomposed into discrete database records for scoped search-based reporting.

2.7 Docker 2.0 Container Management

Work in concert with RHEL AtomicOS. Atomic OS trees are now conveyed across lifecycle environments and deployable using the identical framework used for RHEL. After the Docker Compute Resource is established on an Atomic OS or RHEL-based host, Satellite can execute Docker Pull Requests to load containers from its content management system.

2.8 Foreman

Foreman is an open source application used for provisioning and lifecycle 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.

2.9 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 lifecycle.

2.10 Candlepin

Candlepin is a service within Katello that handles subscription management.

2.11 Pulp

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

2.12 Hammer

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

2.13 REST API

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

2.14 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.

3 Top Performance Considerations

1. Deploy on RHEL 7 – Section 5.1
2. Confirm Tuned is running and configured – Section 5.2
3. Apache configuration – Section 5.3
4. Passenger configuration to increase concurrency – Section 5.4
5. Candlepin configuration – Section 5.5
6. Pulp Configuration – Section 5.6
7. Foreman's performance and scalability – Section 5.7
8. Puppet and scalability – Section 5.8
9. Consider deploying external Capsule(s) in lieu of the integrated Capsule – Section 5.9
10. Katello-Agent scalability – Section 5.10
11. Hammer API timeouts related config changes – Section 5.11
12. qpidd and qdrouterd configuration – Section 5.12
13. PostgreSQL tunings to increase concurrent Registrations of hosts – Section 5.13
14. Disk for DB workloads – Section 5.14
15. Storage needs and Network compatibility with MongoDB – Section 5.15
16. Storage requirements while considering Content views – Section 5.16
17. Minimum Hardware Recommendations – Section 5.17
18. Considerations for capacity planning – Section 5.18
19. Remote execution – section 5.19

4 Environment

4.1 Versions Tested

4.1.1 Satellite:

- Satellite-6.2.1-1.3.el7sat.noarch
- kernel-3.10.0-327.el7.x86_64
- katello-3.0.0-11.el7sat.noarch
- foreman-1.11.0.51-1.el7sat.noarch
- candlepin-0.9.54.7-1.el7.noarch
- mongodb-2.6.11-2.el7sat.x86_64
- postgresql-9.2.15-1.el7_2.x86_64
- tfm-rubygem-passenger-4.0.18-22.el7sat.x86_64
- puppet-3.8.6-2.el7sat.noarch
- pulp-server-2.8.3.4-1.el7sat.noarch
- qpid-cpp-server-0.30-11.el7sat.x86_64
- qpid-dispatch-router-0.4-13.el7sat.x86_64
- ruby-2.0.0.598-25.el7_1.x86_64
- tomcat-7.0.54-2.el7_1.noarch
- python-2.7.5-34.el7.x86_64

4.1.2 Capsule:

- satellite-capsule-6.2.1-1.2.el7sat.noarch
- kernel-3.10.0-327.el7.x86_64
- mongodb-2.6.11-2.el7sat.x86_64
- pulp-server-2.8.3.4-1.el7sat.noarch
- puppet-3.8.6-2.el7sat.noarch
- python-2.7.5-34.el7.x86_64
- ruby-2.0.0.598-25.el7_1.x86_64
- qpid-cpp-server-0.30-11.el7sat.x86_64
- qpid-dispatch-router-0.4-13.el7sat.x86_64

4.2 Hardware Considerations

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

4.2.1 CPU

A typical Satellite 6.2 deployment will run many tasks concurrently, which means that more physical CPU cores available to Satellite 6.2 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.

4.2.2 Memory

Adequate memory should be provided. Satellite 6.2 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, 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.

4.2.3 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 and IOPS on the directories most often accessed. Critical directories for IOPS and monitoring are /var/lib/pulp, /var/lib/pgsql, and /var/lib/mongodb.

4.2.4 Network

Consistently in tests, network hardware has not been found to be a bottleneck before CPU or configuration limits have been revealed. The hardware tested included a 10Gb network between Satellite 6.2, Capsules, and an emulated Content Delivery Network (CDN). It is likely that the Internet connection to the CDN will be a bottleneck, but that is outside the scope of this brief.

4.2.5 Server Power Management

Servers usually have default 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 enables 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.

4.2.6 AWS EC2

Part of the testing was performed on AWS EC2 instances. Here are the flavors used for testing:

- Satellite: *c3.4xlarge*
- Capsules: *c3.4xlarge* (but also tested with *m4.large* and it worked fine)

Because both Satellite and Capsule require significant amount of disk space, additional storage was attached.

Satellite:

```
[
{
    "DeviceName": "/dev/sdf",
    "Ebs": {
     "VolumeSize": 100,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 3000,
     "Encrypted": false
    }
},
{
    "DeviceName": "/dev/sdg",
    "Ebs": {
     "VolumeSize": 40,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 1200,
     "Encrypted": false
```

```
    }
},
{
    "DeviceName": "/dev/sdh",
    "Ebs": {
     "VolumeSize": 40,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 1200,
     "Encrypted": false
    }
},
{
    "DeviceName": "/dev/sdi",
    "Ebs": {
     "VolumeSize": 50,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 1500,
     "Encrypted": false
    }
}
]
```

Capsule:

```
[
{
    "DeviceName": "/dev/sdf",
    "Ebs": {
     "VolumeSize": 100,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 3000,
     "Encrypted": false
    }
},
{
    "DeviceName": "/dev/sdg",
    "Ebs": {
     "VolumeSize": 40,
     "DeleteOnTermination": true,
     "VolumeType": "io1",
     "Iops": 1200,
     "Encrypted": false
    }
 }
]
```

5 Tuning

5.1 RHEL 6.X versus RHEL 7.X

Satellite 6.2 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 6.2 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.

5.2 Tuned Profiles

Satellite 6 should run with the tuning daemon **tuned** 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 6.2 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. On bare-metal, it is recommended that Satellite 6 and Capsules run the 'throughput-performance' **tuned** profile. While, if virtualized, they should 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.

5.3 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 to 5 seconds unless there is a latency between Satellite 6.2, Capsules, or end clients that requires a higher or 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. The default configuration for KeepAlive for Satellite 6 is found in /etc/httpd/conf.d/05-foreman-ssl.d/katello.conf :

Example Satellite 6 Apache configuration tuning:

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

Prefork Multi-Processing Module

Apache on Satellite 6 includes the prefork multi-processing module which scales Apache per process. The default configuration 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.

Max open files limit

Increasing max open files for Apache is also required when doing lots of registrations (because of Bug 1328984) or when increasing scale of Capsules, content hosts, and content views. The approach with limits.conf file is valid for RHEL7 only, there is a different way for

RHEL6:

```
# cat /etc/systemd/system/httpd.service.d/limits.conf
[Service]
LimitNOFILE=1000000
# systemctl daemon-reload
# katello-service restart
```

Max open files limit can be validated with:

```
# systemctl status httpd | grep 'Main PID:'

Main PID: 13888 (httpd)
# grep -e 'Limit' -e 'Max open files' /proc/13888/limits
Limit                     Soft Limit           Hard Limit           Units
Max open files            1000000              1000000              files
```

5.4 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 server. This configures the maximum number of processes available for both Foreman and Puppet on Satellite 6.2 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 and 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 and Puppet traffic, but below Apache's MaxClients and 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 continuous growth 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 6.2 is as follows:

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

```
LoadModule passenger_module modules/mod_passenger.so
<IfModule mod_passenger.c>
     PassengerRoot /usr/share/gems/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
```

By using the passenger-status command, the Foreman and Puppet processes spawned by Passenger can be obtained to confirm the PassengerMaxPoolSize.

Example passenger-status output:

```
# passenger-status
Version : 4.0.18
Date    : 2016-07-21 06:45:23 -0400
Instance: 32499
----------- General information -----------
Max pool size : 24
Processes     : 24
Requests in top-level queue : 0
----------- Application groups -----------
/usr/share/foreman#default:
 App root: /usr/share/foreman
 Requests in queue: 0
 * PID: 615     Sessions: 0
Processehttp://weblogs.asp.net/jongalloway/performant-isn-t-a-wordd: 169
    Uptime: 8m 30s
    CPU: 5%      Memory  : 279M    Last used: 3s ago
    [...]
 * PID: 2719    Sessions: 0       Processed: 174     Uptime: 5m 15s
    CPU: 13%     Memory  : 241M    Last used: 3s ago
/etc/puppet/rack#default:
 App root: /etc/puppet/rack
 Requests in queue: 0
 * PID: 15780   Sessions: 1       Processed: 5       Uptime: 22s
    CPU: 1%      Memory  : 34M     Last used: 2s 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   : 2016-07-21 06:46:24 -0400
---------- Apache processes -----------
PID    PPID   VMSize     Private  Name
---------------------------------------
956    32499  193.6 MB   3.0 MB   /usr/sbin/httpd -DFOREGROUND
1152   32499  193.5 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
1153   32499  193.7 MB   3.1 MB   /usr/sbin/httpd -DFOREGROUND
1472   32499  193.6 MB   2.9 MB   /usr/sbin/httpd -DFOREGROUND
1473   32499  193.6 MB   3.1 MB   /usr/sbin/httpd -DFOREGROUND
1771   32499  0.0 MB     ?        [httpd] <defunct>
1778   32499  193.6 MB   3.0 MB   /usr/sbin/httpd -DFOREGROUND
[...]
20577  32499  193.4 MB   2.7 MB   /usr/sbin/httpd -DFOREGROUND
32499  1      191.1 MB   0.5 MB   /usr/sbin/httpd -DFOREGROUND
```

```
32560  32499  1327.5 MB  79.3 MB  (wsgi:pulp)      -DFOREGROUND
32561  32499  1327.5 MB  80.0 MB  (wsgi:pulp)      -DFOREGROUND
32562  32499  1775.5 MB  89.3 MB  (wsgi:pulp)      -DFOREGROUND
32563  32499  524.2 MB   14.1 MB  (wsgi:pulp-cont -DFOREGROUND
32564  32499  460.2 MB   14.1 MB  (wsgi:pulp-cont -DFOREGROUND
32565  32499  460.2 MB   14.1 MB  (wsgi:pulp-cont -DFOREGROUND
32566  32499  786.0 MB   49.1 MB  (wsgi:pulp_forg -DFOREGROUND
32567  32499  850.0 MB   51.1 MB  (wsgi:pulp_forg -DFOREGROUND
32568  32499  786.0 MB   47.1 MB  (wsgi:pulp_forg -DFOREGROUND
### Processes: 175
### Total private dirty RSS: 914.23 MB (?)

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

### Processes: 0
### Total private dirty RSS: 0.00 MB
------ Passenger processes ------
PID    VMSize     Private   Name
-------------------------------
489    676.9 MB   178.3 MB  Passenger AppPreloader: /usr/share/foreman
615    1919.3 MB  280.4 MB  Passenger RackApp: /usr/share/foreman
[...]
2719   1986.4 MB  243.6 MB  Passenger RackApp: /usr/share/foreman
15591  151.9 MB   30.6 MB   Passenger AppPreloader: /etc/puppet/rack
15780  283.9 MB   37.4 MB   Passenger RackApp: /etc/puppet/rack
32569  209.8 MB   0.3 MB    PassengerWatchdog
32572  2290.8 MB  9.2 MB    PassengerHelperAgent
32580  214.1 MB   0.9 MB    PassengerLoggingAgent
### Processes: 29
### Total private dirty RSS: 6644.56 MB
```

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

5.5 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, and maximum

timings, the capacity of a specific environment can be determined. The default Passenger configuration with Satellite 6.2 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 queueing 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.

5.6 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 once, then more workers can consume Satellite 6.2 resources. This can starve other components of Satellite 6.2 and therefore it can be necessary to adjust the concurrency level of Pulp in an environment with a concurrent workload such as Puppet.

NFS

Red Hat Satellite 6.2 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 the Pulp directory onto a large local partition that you can easily scale . Use Logical Volume Manager (LVM) to create this partition.

5.7 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 5.4. 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 and API as represented in the commands shown in Section 5.3.

5.8 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 6.2 deployment.

Runinterval – A non-deterministic run-interval 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, run-interval can be distributed by:

- Puppet splay – Turn on splay for each Puppet client. This adds randomization to the run-interval, however this does not accomplish a deterministic run-interval.
- A cron job – Run each Puppet agent as a cron job rather than a daemon. This makes a run-interval deterministic however at scale this becomes difficult to manage when adding and removing clients.
- Separation – 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 Server and 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 and 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 a load on Satellite Server 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.

5.9 External Capsules

External Capsules allow a Satellite 6.2 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 6.2.
- Provides more CPU and Memory resources for Puppet and Foreman.

- 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 versus 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 6.2 (in Section 5.8 testing results, Satellite 6.2 was scaled). 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 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 to benchmark a specific Puppet workload to determine its scalability.

Hardware Considerations and Apache/Passenger Configuration:

The same considerations for hardware for Satellite 6.2 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
```

5.10 Client Agent Scaling (katello-agent)

First check “Scale: qpidd and qdrouterd configuration” (on both Satellite and Capsule if any). Section Apache configuration: Max open files limitation might be also useful here.

You can increase the content_action_accept_timeout in *Administer -> Settings -> Katello* to more than the default 20 seconds so clients have more time to answer (see https://access.redhat.com/solutions/2016943)

5.11 Scale: Hammer Timeout

During scale of Capsules, content hosts, or Content views hammer API requests can timeout. Set time out to -1 in /etc/hammer/cli.modules.d/foreman.yml to set no timeout.

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

5.12 qpid and qdrouterd Configuration

Max open files limit

When you plan to have a big amount of clients with katello-agent, you need to tune qpidd and qdrouterd settings to handle that load. The approach with limits.conf file is valid for RHEL7 only, there is a different way for RHEL6.

```
# echo 1000000 > /proc/sys/fs/aio-max-nr   # to set it permanently set it
in /etc/sysctl.conf
# cat /etc/systemd/system/qpidd.service.d/limits.conf
[Service]
LimitNOFILE=1000000
# cat /etc/systemd/system/qdrouterd.service.d/limits.conf
[Service]
LimitNOFILE=1000000
# systemctl daemon-reload
# katello-service restart
```

More details can be found in https://access.redhat.com/solutions/1375253

Diskspace consideration

When you use katello-agent extensively, plan storage capacity for /var/lib/qpidd in advance. Currently, in Satellite 6.2 /var/lib/qpidd requires 2MB disk space per a content host (see Bug 1366323).

`mgmt-pub-interval` settings

Also it might happen you are hitting this error in /var/log/messages (RHEL6) or journal (RHEL7):

```
satellite.example.com qpidd[92464]: [Broker] error Channel exception: not-attached: Channel 2 is not attached (/builddir/build/BUILD/qpid-cpp-0.30/src/qpid/amqp_0_10/SessionHandler.cpp:39)
```

```
satellite.example.com qpidd[92464]: [Protocol] error Connection qpid.10.1.10.1:5671-10.1.10.1:53790 timed out: closing
```

This is because qpid maintains management objects for queues, sessions, and connections and recycles them every 10 seconds by default. Meanwhile, the same object with the same ID is created, deleted, and created again. The old mgmt object is not purged yet and qpid raises that error. The workaround is to lower mgmt-pub-interval parameter from the default 10s to something lower (needs tuning). Add it to /etc/qpid/qpidd.conf and restart the qpidd service. See Bug 1335694#c7.

5.13 PostgreSQL Configuration

PostgreSQL requirements depends on various requirements such as the number of organizations, environments, registered systems, and content views. While registering content hosts at scale to Satellite Server, **shared_buffers** needs to be set appropriately. Recommended: 256MB

PostgreSQL configuration: /var/lib/pgsql/data/postgresql.conf

```
shared_buffers = 256MB
```

When registering content hosts at scale, it is recommended to increase max_connections setting (set to 100 by default) as per your needs and HW profile. For example, you might need to set the value to 200 when you are registering 200 content hosts in parallel.

```
max_connections = 200
```

5.14 Storage Media for Database Workloads

Pulp and 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 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.

Directories that should be mounted on high performance storage:

- `/var/lib/pulp` – click here for relevant documentation and recommendations; Plan for expansion over time as this will continue to grow as content is added to Satellite Server.
- `/var/lib/mongodb` – click here for relevant documentation and recommendations; P Plan for expansion over time as this will continue to grow as content is added to Satellite Server.
- `/var/lib/pgsql` – click here for relevant documentation and recommendations.
- `/var/lib/qpidd` – requires 2MB disk space per a content host (see Bug 1366323). Plan for expansion over time as this will continue to grow as content is added to Satellite Server.
- `/var/log` - logs size can grow depending on your use case and your root volume size, consider having a separate volume mounted here

5.15 MongoDB

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

Size
The storage requirements depend on the number of packages and content views for Satellite environment. Content view publishing to different versions consumes more MongoDB space. Therefore effective capacity planning is recommended, based on the number of packages and published versions.

5.16 Content View

Red Hat Satellite 6.2 provides users with the ability to create *content views*. Content views act as a snapshot of one or more repositories or Puppet modules. Content Views are 'published' in order to lock their contents in place. The content of the Content View is cloned and all filters are applied. Publishing creates a new version of the Content View. Content Views can be promoted and cloned to different life cycle environments (Dev, Test, Production). Content view uses a symbolic link to the Media Library stored in the Pulp directory. Additionally, 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 and uses a small amount of storage space. 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 consumes more and more space. To reduce consumption and free storage, remove old and unused versions of content views in a lifecycle environment. Also, monitor nodes and published directories under Pulp.

5.17 Minimal Hardware Recommendations

For minimal HW profile see Installation Guide (in short: 2 CPUs, 12 GB RAM, 4 GB swap). Some basic numbers to expect:

- Sync 7589 pkgs 650 erratas: 15 minutes (syncing directly through local 10 GB network)
- Publish new version of content view with that repo: 2 minutes
- Promote to Library: 1 minute

5.18 Considerations for Capacity Planning

Puppet master is mostly a CPU intensive server. If more CPU intensive modules are required and more CPU and memory is made available, then Puppet would perform better. Also, it is a good idea to use RHEL7 due to performance gain in Ruby 2.0 versus 1.87.

5.19 Remote Execution

Remote execution scales well. We have tested the simple command `date` on 6000 systems and it worked well. Bigger scale might require some tuning or usage of Capsules.

6 Results

6.1 Tuned Profiles

Testing of Tuned on Satellite 6.2 on RHEL 6 shows that throughput-performance profile consistently provided a reduction in latency with Satellite 6.2 API driven hammer commands as well as syncing of content.

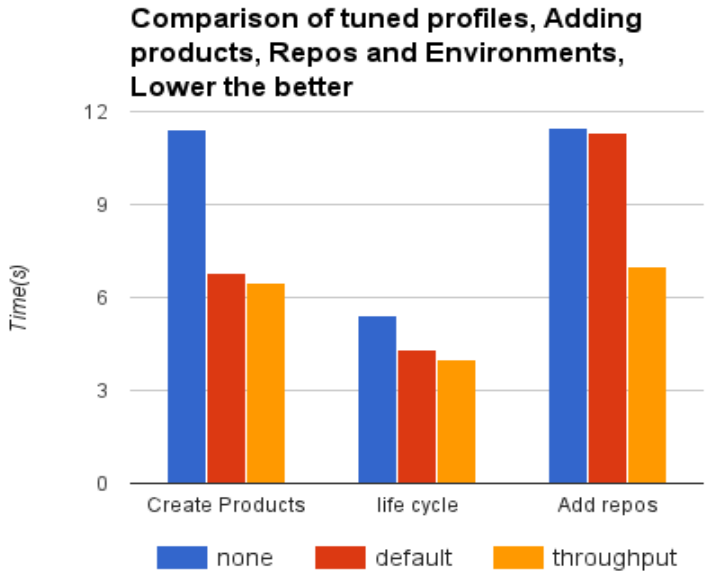

Figure 6.1 Comparison of Tuned Profiles

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 6.2 on RHEL 6.

Synchronization of RHEL5 repo to Satellite 6.2 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.

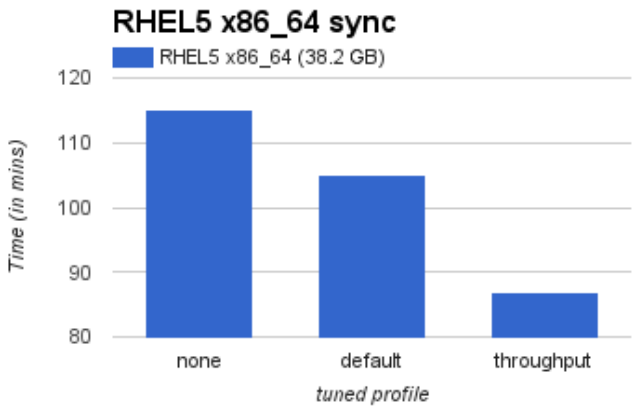

Figure 6.2 RHEL 5 x86_64 Repo sync with tuned Profiles

The graph above is a comparison between Tuned off, the default profile, and throughput-performance profile on Satellite 6.2 during sequential syncing of content.

6.2 Satellite on RHEL 7

Concurrent synchronization of RHEL5, 6, and 7 repositories to Satellite 6.2 on RHEL 6 & RHEL 7

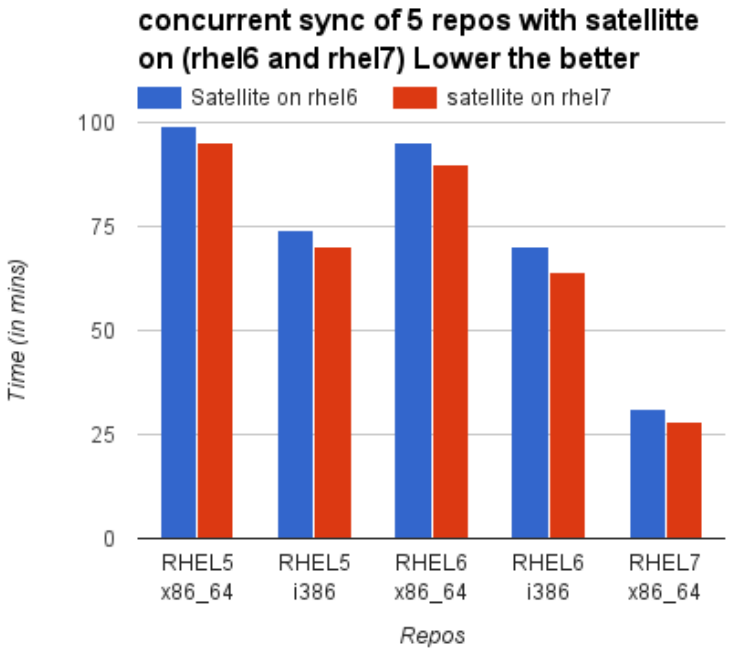

Figure 6.3 Concurrent Sync of 5 Repos

The graph above is a comparison between running Satellite 6.2 on RHEL 6 & RHEL 7 during concurrent syncing of content.

6.3 Storage Media Preference for Database

Concurrent synchronization of RHEL5, 6, and 7 repos to Satellite 6.2 with Pulp & MongoDB on SSD and HDD

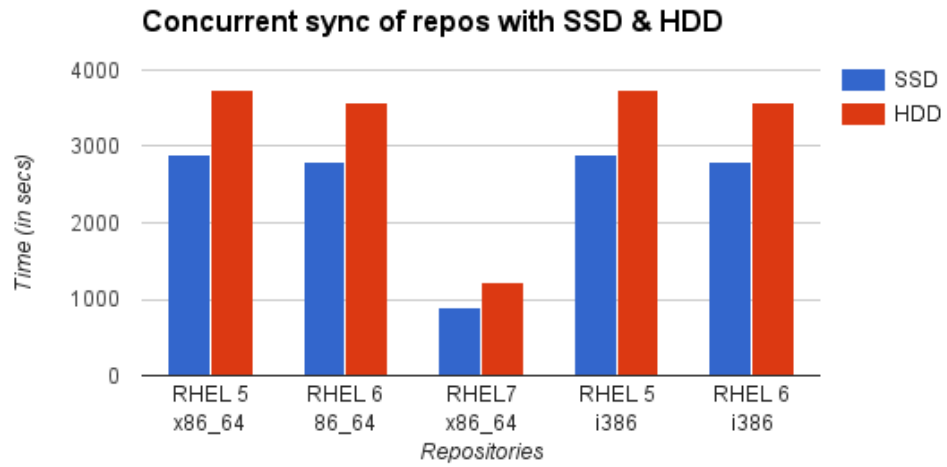

Figure 6.4 Concurrent Sync of Repos with SSD & HDD

The graph above is a comparison between running Satellite 6.2 on RHEL 7 with Pulp and MongDB on SSD during concurrent syncing of content.

6.4 Concurrent Content Host Registrations

With the default configuration, Satellite 6.2 handles about 90 to 100 concurrent registrations in optimal circumstances (no load on the system generated by other software or components, sufficient HW profile). Note that registration is not a synchronous operation, so if you want to register large number of systems, remember to add some delay between each of your bunches. With all the tunings mentioned in the document, Satellite 6.2 can handle up to 200 concurrent registrations.

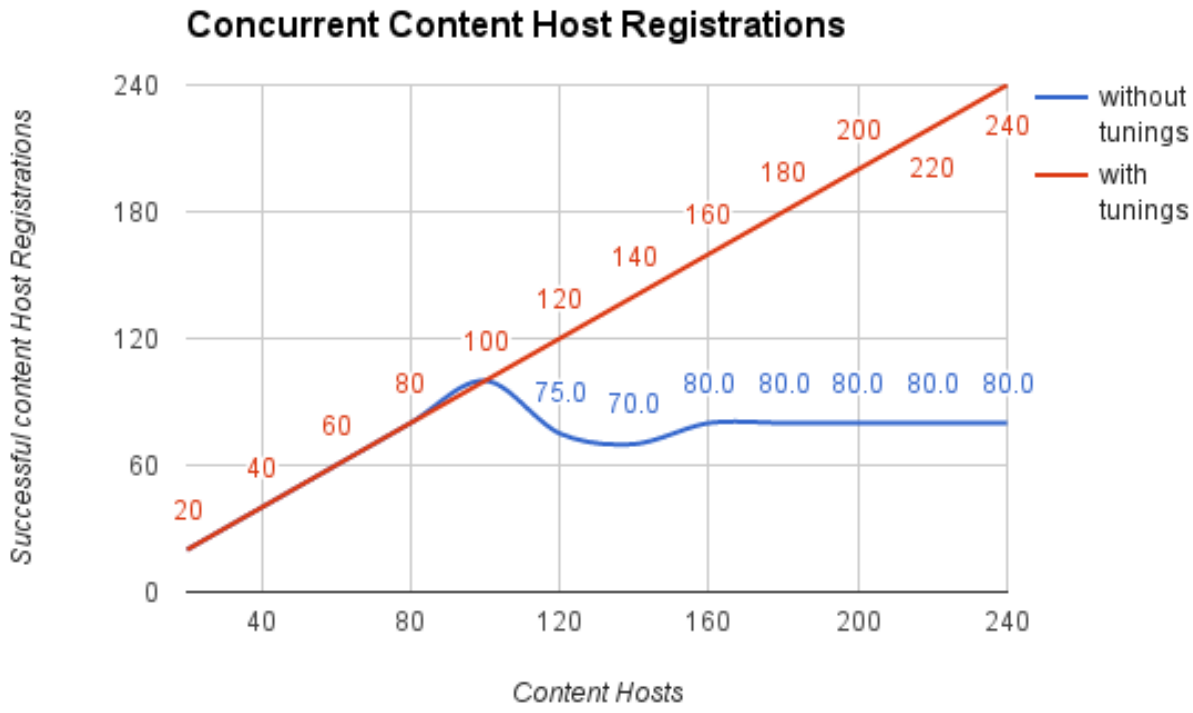

Figure 6.5: Concurrent Content Host Registrations

Testing the concurrency of Candlepin has shown that system registrations scale almost linearly until filling the Passenger queue (in the graph above, we have tested with defaults and with **tuned** configuration):

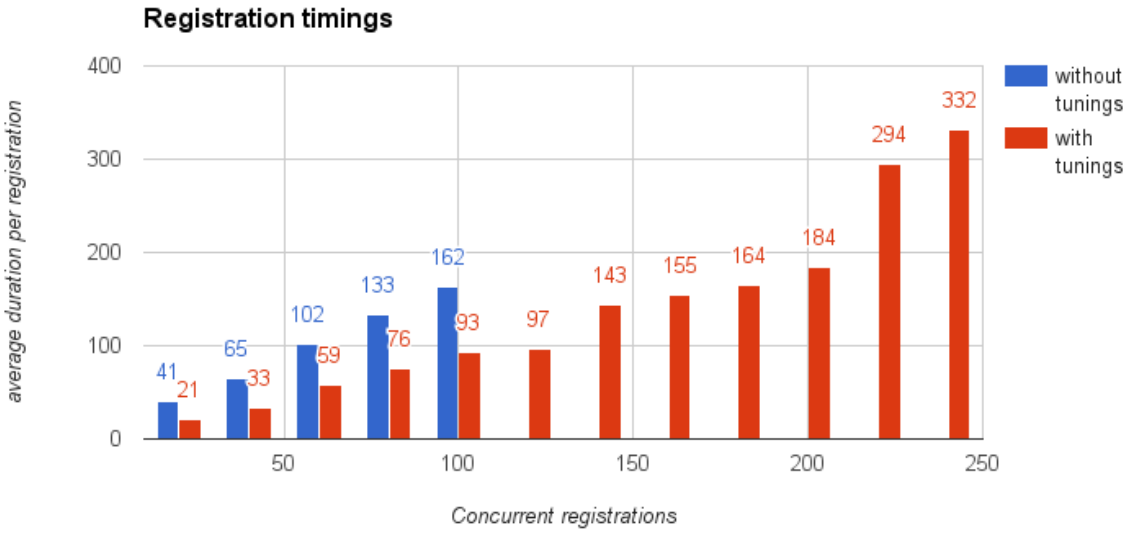

Figure 6.6: Concurrent Registration Timings

The graph does not contain values after 100 concurrent registration attempts (case without tuning) and beyond 240 (case with tuning), because that started to encounter issues. Additional concurrent registrations result in a longer time required to register.
It is also important to note that running subscriptions at the maximum rate leaves no room for other tasks or 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.

6.5 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.

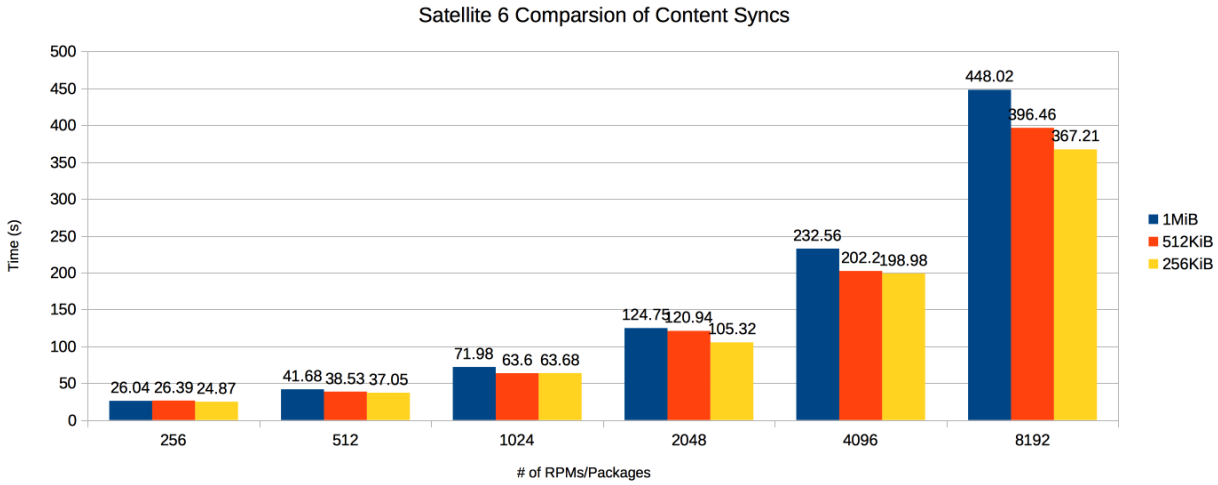

Figure 6.7 Comparison of Content Syncs

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.

Pulp is also installed on the Capsules and used to sync content closer to the end clients. Publishing or promoting a content view to a life cycle environment on a Capsule triggers that Capsule to sync the content in the content view.

6.6 Pupet Integrated Capsule

Puppet scalability against Satellite 6.2 Integrated Capsule depends upon the life- cycle of the client and changes to the Puppet configuration. Steady state operations of Puppet checking an existing configuration is less resource intensive than Puppet applying a new configuration.

When tuning this, check Passenger configuration (especially the PassengerMaxRequestQueueSize directive).
Also, it is important to note that running an environment at or near 100% capacity is strongly discouraged. 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 6.2 machine.

6.7 Puppet External Capsule

A Puppet workload with external Capsule is upload facts, compile the specific client's Puppet catalog and submit a report. RHEL 7 scales better compared to what RHEL 6 was able to accomplish (in both scale of catalog compilation and latency required to compile).

6.8 Concurrent Puppet Agent Runs

When testing with the simplest possible Puppet module, one Capsule in default configuration was able to handle about 320 clients concurrently running "puppet agent". With more concurrency some of the "puppet agent" runs were not served and the client failed to deploy the given module. As a side effect, average run time decreased, but that can not be considered a win when there are these failures.

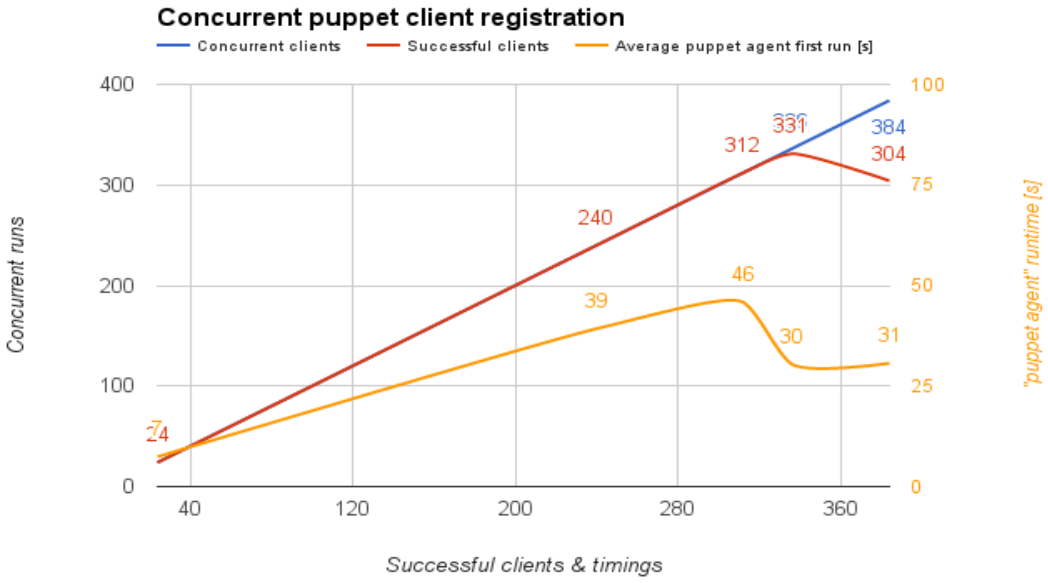

Figure 6.8: Concurrent Puppet Client Registration

6.9 Concurrent Errata Update on Content hosts

With a strong Satellite (36 CPUs, 60 GB RAM, SSD for /var/lib/pulp, /var/lib/pgsql, /var/lib/mongodb and /var/lib/qpidd), 3 Capsules and 6000 clients (2000 per Capsule), we were able to apply errata to 6000 hosts with complete success. Configuration tweaks used on all Satellite and Capsules were:

- Max open files limit for httpd increased (see section *5.3 Apache Configuration*)
- Max open files limit for qpidd and qdrouterd increased (see section *5.12 qpid and qdrouterd configuration*)
- echo 1000000 > /proc/sys/fs/aio-max-nr (see section *5.12 qpid and qdrouterd configuration* again)
- SSD disk mounted as /var/lib/qpidd (see section *5.12 qpid and qdrouterd configuration* again)

6.10 Systems per Capsule Considerations

Using strong systems (36 CPUs, 60 GB RAM, SSD for /var/lib/pulp and /var/lib/mongodb) as a Capsule server, we have tested 2000 clients per Capsule with clients using katello-agent.

7 Conclusion

Satellite 6.2 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 6.2 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 well performing and scaled Satellite 6.2 environment. Since each Satellite 6.2 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 6.2 deployment. Section 3. Top Performance Considerations, identifies the highest priority items to consider when tuning and deploying Satellite 6.2 for optimal performance.

8 Recommendations

Prior to deployment of Satellite 6.2, after assessment of throughput and task latency requirements has been completed, hardware selection should be done keeping in mind:

- Greater number of CPU cores means better Scalability.
- To prevent swapping, sufficient memory should be provided that accounts for all the software components of Satellite, namely Apache, Foreman, Katello, MongoDB, Postgres, Pulp, Puppet, Tomcat, Qpid, and the file system cache.
- Proper partitioning of file system for critical directories such as /var/lib/{pulp,pgsql,mongodb,qpidd} gives efficacious IOPS results, so use Logical Volume Manager (LVM) for this purpose. Additionally, usage of high-bandwidth, low-latency storage is recommended, preferably on SSD. We also recommend adding storage with effective capacity planning based on required content view publishing to different versions, which triggers mongodb to consume more space.
- Network connections to CDN should be robust, but as such, haven't been found to be much of a problem since we used direct 10g network path to CDN for testing.
- To counter performance issues due to a server's default power conservation enable OS Host Power Control in BIOS
- Satellite 6.2 performs best when deployed on a RHEL7 operating system, as it includes many enhancements when compared to RHEL6 (refer to section 5.1). Additionally:
- A tuned profile that's equivalent of 'throughput-performance' on bare-metal and 'virtual-guest' in virtualized environments, is recommended. (refer to results in graphs under section 6.1)
- For Apache, it would be wise to enable KeepAlive tunable, and set KeepAliveTimeout value between 2-5 seconds and MaxKeepAliveRequests to be 0. This in turn, helps Foreman, which handles UI and API requests. Also, depending upon the amount of registrations expected, appropriate apache prefork multi-processing module configurations and max open files limit in limits.conf should suffice.
- For a foreseeable behavior of Foreman and Puppet, depending upon the maximum burst of requests, global Passenger configuration directives as well as those specific to an application, should be set as shown in section 5.4.
- A deterministic runinterval helps the case of performance and scalability, and the same could be achieved by following guidelines enunciated under section 5.8, which also states other tunables relevant to Puppet. The same also affects the use case of an external Capsule vs an integrated Capsule, in addition to the dependability on hardware, configuration, and workload, as stated under section 5.9
- To determine latency required for an environment during completion of registrations, tuning of Candlepin requires recording the time of a subscription-manager registration, as shown in section 5.5 and then, configuring Passenger

appropriately.

- Concurrency level of Pulp can be adjusted in /etc/default/pulp_workers, as it is proportional to the number of available CPUs and is responsible for the time it takes to sync a large number of repositories at once. Also, usage of NFS should be avoided since it creates issues for mongodb (which is used by Pulp).
- For scaling up, following should be the priorities:
  - Apache configurations, as listed under section 5.3
  - For PostgreSQL, set *shared_buffers* to 15% - 25% of memory and *max_connections* to the amount of content hosts being registered concurrently.
  - To plan for large number of clients with katello-agent, follow scaling strategies for katello-agent, hammer API and qpid/qdrouterd, as stated under section 5.10, 5.11 and 5.12 respectively.
- For minimum hardware recommendations, refer to section 5.17

Appendix A: Revision History

Revision 1.0		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
Stephen Wadeley	Senior Technical Writer	Review
Christopher Duryee	Principal Software Engineer	Review
David Caplan	Product Manager	Review
Douglas Shakshober	Technical Director, Performance & Scale Engineering	Review
Brian Riordan	Director, Performance & Scale 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
12. https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server

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.

Appendix F: Significant PostgreSQL Tunables

`max_connections` – Maximum number of client connections allowed

`shared_buffers` – Determines how much memory is dedicated to PostgreSQL to use for caching data