Performance Tuning of Satellite 6 and Capsules

Alex Krzos, Software Engineer

Version 1.0
March 2015
# Table of Contents

1 Executive Summary

2 Top Performance Considerations

3 Environment

3.1 Versions Tested

3.2 Hardware Considerations

3.3 Red Hat Enterprise Linux

3.4 Apache Configuration

3.5 Passenger Configuration

3.6 Candlepin

3.7 Pulp

3.8 Foreman

3.9 Puppet

3.10 External Capsules

3.11 Katello-agent

4 Results

4.1 Tuned Profile

4.2 Apache KeepAlive CPU Usage

4.3 Candlepin Concurrency

4.4 Pulp Content Syncs

4.5 Puppet Integrated Capsule

4.6 Puppet External Capsule

5 Conclusion

Appendix A: Revision History

Appendix B: Contributors and Reviewers

Appendix C: References

Appendix D: Significant Apache Tunables
1 Executive Summary

This paper provides basic guidelines and considerations for tuning Red Hat Satellite 6 for performance and scalability. Many factors drive the performance of a Satellite 6 deployment and testing should be conducted before performing any of the suggested tuning. There is no one-size-fits-all configuration as tunings will vary based on your environmental factors, such as the hardware Satellite 6 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 to meet your needs for life-cycle management of systems.

2 Top Performance Considerations

1. Deploy on RHEL 7 - Section 3.3
2. Consider deploying external Capsule(s) in lieu of the integrated Capsule - Section 3.10
3. Confirm Tuned is running and configured - Section 3.3
4. Adjust Apache KeepAlive tunables on Satellite 6 and Capsule(s) - Section 3.4
5. Adjust Passenger tunables on Satellite 6 and Capsule(s) - Section 3.5
6. Monitor and tune system as changes in environment occur

3 Environment

3.1 Versions Tested

<table>
<thead>
<tr>
<th></th>
<th>Satellite 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>6</td>
</tr>
<tr>
<td>Kernel</td>
<td>2.6.32-504.1.3.el6.x86_64</td>
</tr>
<tr>
<td>Satellite 6</td>
<td>6.0.4</td>
</tr>
<tr>
<td>Apache</td>
<td>2.2.15-39</td>
</tr>
<tr>
<td>Candlepin</td>
<td>0.9.23.1-1</td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>0.90.10-6</td>
</tr>
<tr>
<td>Foreman</td>
<td>1.6.0.47-1</td>
</tr>
<tr>
<td>Katello</td>
<td>1.5.0-30</td>
</tr>
<tr>
<td>Mongodb</td>
<td>2.4.6-2</td>
</tr>
<tr>
<td>Passenger</td>
<td>4.0.18-19</td>
</tr>
<tr>
<td>Postgres</td>
<td>8.4.20-1</td>
</tr>
</tbody>
</table>
Hardware Considerations
Selecting your hardware is the first component of setting up a performant 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.

### 3.3 Red Hat Enterprise Linux

**RHEL 6 vs RHEL 7**

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.

**RHEL 6 (bare-metal):**

```bash
# yum install -y tuned
# service tuned start
# chkconfig tuned on
# tuned-adm profile throughput-performance
```

**RHEL 6 (virtual machine):**

```bash
# yum install -y tuned
```
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 install 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 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.

3.4 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
</IfModule>```
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.

### 3.5 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 server. This configures the maximum number of processes available for both Foreman and Puppet on Satellite 6 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 requestor. 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 startup. 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 would resemble the following:

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

```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**

```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**

```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:**

```
# passenger-status
Version : 4.0.18
Date    : Mon Jan 19 15:49:32 -0500 2015
Instance: 36651
----------- General information -----------
Max pool size : 24
Processes     : 12
Requests in top-level queue : 0
----------- Application groups -----------
/usr/share/foreman#default:
  App root: /usr/share/foreman
  Requests in queue: 0
    * PID: 23186   Sessions: 0       Processed: 141     Uptime: 1h 25m 42s
    CPU: 0%       Memory  : 102M    Last used: 10m 48s
    * PID: 23197   Sessions: 0       Processed: 0       Uptime: 1h 25m 42s
```

www.redhat.com 6 refarch-feedback@redhat.com
CPU: 0%      Memory  : 72M     Last used: 1h 25m 4
* PID: 23207   Sessions: 0       Processed: 0       Uptime: 1h 25m 41s
CPU: 0%      Memory  : 72M     Last used: 1h 25m 4
* PID: 23217   Sessions: 0       Processed: 113     Uptime: 1h 25m 41s
CPU: 0%      Memory  : 101M    Last used: 10m 46s
* PID: 23229   Sessions: 0       Processed: 0       Uptime: 1h 25m 40s
CPU: 0%      Memory  : 72M     Last used: 1h 25m 4
* PID: 23240   Sessions: 0       Processed: 0       Uptime: 1h 25m 40s
CPU: 0%      Memory  : 72M     Last used: 1h 25m 4
* PID: 23200   Sessions: 0       Processed: 16      Uptime: 1h 25m 37s
CPU: 0%      Memory  : 56M     Last used: 12m 59s
* PID: 23204   Sessions: 0       Processed: 0       Uptime: 1h 25m 37s
CPU: 0%      Memory  : 40M     Last used: 1h 25m 3
* PID: 23208   Sessions: 0       Processed: 0       Uptime: 1h 25m 3
CPU: 0%      Memory  : 33M     Last used: 1h 25m 3
* PID: 23212   Sessions: 0       Processed: 0       Uptime: 1h 25m 3
CPU: 0%      Memory  : 33M     Last used: 1h 25m 3
* PID: 23216   Sessions: 0       Processed: 0       Uptime: 1h 25m 3
CPU: 0%      Memory  : 33M     Last used: 1h 25m 3
* PID: 23220   Sessions: 0       Processed: 0       Uptime: 1h 25m 3
CPU: 0%      Memory  : 3M      Last used: 1h 25m 3

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   : Mon Jan 19 15:50:10 -0500 2015
-------- Apache processes --------
PID    PPID   VMSize    Private   Name
--------------------------------------
23090  36651  720.3 MB  38.9 MB  (wsgi:pulp)
23091  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23092  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23093  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23094  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23095  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23096  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23097  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
23098  36651  162.5 MB  2.3 MB   /usr/sbin/httpd
36651  1      160.6 MB  0.8 MB   /usr/sbin/httpd
### Processes: 10
### Total private dirty RSS: 58.23 MB
-------- Nginx processes --------
### Processes: 0
### Total private dirty RSS: 0.00 MB
-------- Passenger processes ------
PID    VMSize     Private   Name
---------------------------------
23071  209.1 MB   0.2 MB    PassengerWatchdog
23074  1524.9 MB  1.4 MB    PassengerHelperAgent
23080  209.2 MB   0.5 MB    PassengerLoggingAgent
All of the relevant Passenger tunables can be found in Appendix E and further documentation can be found in the references.

### 3.6 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 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.

### 3.7 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 an equal number of workers as there are available CPUs. 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 resources. This can starve other components of Satellite 6 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.

### 3.8 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 3.4. 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/API as represented in the commands shown under the graphs displayed in Section 4.1.

### 3.9 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 deployment.

**Runinterval** – A non-deterministic runinterval that does not distribute the load throughout the interval is likely to 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 cronjob 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 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.
3.10 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.
- Provides more CPU/Memory resources for Puppet.
- 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 an 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 4.5 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, then 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.

Hardware Considerations and Apache/Passenger Configuration:

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 overcommit 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

3.11 Katello-agent

By default the Satellite 6 internal and external Capsules have a configuration default that limits the number of connected agents to 225. To scale beyond the 225 predefined limit adjust the max connections allowed by qpid on the Satellite 6 and Capsules.

Edit /etc/sysconfig/qpidd and add the following value:

QPIDD_OPTIONS="--max-connections 2000"

Put the maximum number of planned connections in this option where 2000 was used as an example. Restart the 'qpidd' service to see the changes take effect.
4 Results

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

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.

The graph above is a comparison between Tuned off, the default profile, and throughput-performance profile on Satellite 6 during sequential syncing of content.

4.2 Apache KeepAlive CPU Usage
Tests with a scaled workload of multiple Capsules show that without KeepAlive, Satellite 6 will utilize far more CPU resources due to constant opening of TCP connections instead of
serving multiple files over a single connection to a Capsule. This ties up precious CPU resources and increases latency for any other concurrent tasks which Satellite 6 could be performing.

The above image graphs the user CPU usage on Satellite 6 between KeepAlive On/Off with twenty Capsules syncing a 512MiB repository.

### 4.3 Candlepin Concurrency

Testing the concurrency of Candlepin has shown that system registrations scale linearly until filling the Passenger queue:

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 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.
4.4 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 especially evident in smaller repositories where the effect of file size has little to no impact on the latency of syncing.

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 at the Capsules and used to sync content closer to the end clients. Publishing or promoting a content view to a lifecycle environment on a Capsule triggers that Capsule to sync the content in the content view. Performance of multiple Capsules syncing content from a Satellite 6 server was measured in the following test:

This graph compares RHEL 6 and RHEL 7 syncing the same four repositories to a specific number of Capsules. Percent change is shown between RHEL 6 and RHEL 7.
In this test RHEL 7 has an approximate 21-19% reduction in latency to sync content from 4 to 16 Capsules. As the number of Capsules was scaled between 18 to 20, the percent change decreased to approximately 14.9-14.5%.

4.5 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 pre-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 run time of each Puppet agent was timed and graphed in below graphs. The initial Puppet agent applying a configuration was timed and the test was run a second time to determine the run time of Puppet agent at its steady state. The Apache Puppet class was chosen to provide a workload for the catalog compilation.

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 is between 2,000-1,800 for a run-interval of 30 minutes.
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.

### 4.6 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.

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

![Satellite 6 Capsules Average Catalog Compilation Response Timing](chart.png)

The above response times were averaged between two Capsules, each compiling the specific number of catalogs over a 30 minute interval. Most noticeable is the large improvement in catalog compilation response timing and the slower growth in response timing as more catalogs are compiled in the same interval between Satellite 6 external Capsules on RHEL 7 versus RHEL 6.

### 5 Conclusion

Satellite 6 is a robust platform for lifecycle management of systems that combines the best of many open source projects. Efficiently planning, tuning and monitoring of system resources in Satellite 6 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 performant and scaled Satellite 6 environment. Since each Satellite 6 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. Following Section 2, Top Performance Considerations, provides the highest priority items to consider when tuning and deploying Satellite 6 for best performance.
Appendix A: Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Thursday March 12, 2015</td>
<td>Alex Krzos</td>
</tr>
<tr>
<td>Initial Release</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix B: Contributors and Reviewers

<table>
<thead>
<tr>
<th>Contributor</th>
<th>Title</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy Bond</td>
<td>Sr. Principal Software Engineer</td>
<td>Review</td>
</tr>
<tr>
<td>Brett Thurber</td>
<td>Principal Software Engineer</td>
<td>Review</td>
</tr>
<tr>
<td>Clifford Perry</td>
<td>Sr. Manager, Software Engineering</td>
<td>Review</td>
</tr>
<tr>
<td>Mike McCune</td>
<td>Manager, Software Engineering</td>
<td>Section 3.11</td>
</tr>
<tr>
<td>Nick Dokos</td>
<td>Sr. Software Engineer</td>
<td>Review</td>
</tr>
<tr>
<td>Rich Jerrido</td>
<td>Principal Product Marketing Manager</td>
<td>Review</td>
</tr>
<tr>
<td>Tim Wilkinson</td>
<td>Sr. Software Engineer</td>
<td>Review</td>
</tr>
</tbody>
</table>

Appendix C: References

5. http://httpd.apache.org/docs/2.2/
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.

**PassengerPreloaderIdleTime** – 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.