Satellite 6 Performance Tuning Guide

Red Hat Performance Engineering

Jun 24, 2020

CONTENTS:

CHAPTER

ONE

AUTHORS

Pradeep Surisetty <psuriset@redhat.com>

Jan Hutar <jhutar@redhat.com>

Mike McCune <mmccune@redhat.com>

Sureshkumar Thirugnanasambandan <sthirugn@redhat.com>

Imaanpreet Kaur <ikaur@redhat.com>

Andrew Puch <apuch@redhat.com>

CHAPTER

TWO

LEGAL NOTICE

CHAPTER

THREE

ABSTRACT

The performance tuning guide aims to cover the set of tunings and tips that can be used as a reference to scale up your Red Hat Satellite 6.7 environment.

CHAPTER

FOUR

INTRODUCTION

This document aims to provide the guidelines for tuning Red Hat Satellite 6 for performance and scalability. Although a lot of care has been given to make the content applicable to cover a wide set of use cases, if there is some use case which has not been covered, please feel free to reach out to Red Hat for support for the undocumented use case.

Red Hat Satellite is a complete system management product that enables system administrators to manage the full life cycle of Red Hat product deployments. Red Hat Satellite can manage these deployments across physical, virtual and private clouds. Red Hat Satellite delivers system provisioning, configuration management, software management, subscription management, and does so while maintaining high scalability and security.

For more information on Red Hat Satellite 6, please visit:

https://access.redhat.com/documentation/en/red-hat-satellite/

CHAPTER

FIVE

SYSTEM REQUIREMENTS

For details of Red Hat Satellite 6 hardware and software requirements, please take a look at Preparing your environment for installation, inside the installation guide.

5.1 Quick Tuning Guide

Users who wish to tune their Satellite based on expected managed host counts and hardware allocation can utilize the built in tuning profiles included in Satellite 6.7 and later that are available via the installation routine's new tuning flag:

```
# satellite-installer --help
Usage:
    satellite-installer [OPTIONS]

Options:
[....]
      --tuning INSTALLATION_SIZE  Tune for an installation size. Choices: default,␣
↪medium, large, extra-large, extra-extra-large (default: "default")
```

There are 4 sizes provided based on estimates of the number of managed hosts your Satellite will be hosting.

Name	Number of managed hosts	Recommend RAM	Recommend Cores
default	0-5000	20G	4
medium	5000-10000	32G	8
large	10000-20000	64G	16
extra-large	20000-60000	128G	32
extra-extra-large	60000+	256G+	48+

Instructions for use:

1. Determine the profile you wish to use
2. Run *satellite-installer –tuning large*. This will apply the chose tuning profile.
3. Resume operations

NOTE: The specific tuning settings for each profile can be viewed in the configuration files contained in */usr/share/foreman-installer/config/foreman.hiera/tuning/sizes*

CHAPTER

SIX

TOP PERFORMANCE CONSIDERATIONS

This is the list of things that you can do to improve the performance and scalability of Red Hat Satellite 6:

- Configuring httpd
- Configuring passenger to increase concurrency
- Configure candlepin
- Configure pulp
- Configure Foreman's performance and scalability
- Configure Dynflow
- Deploy external Capsule(s) in lieu of internal capsules
- Configure katello-agent for scalability
- Configure hammer to reduce API timeouts
- Configure qpid and qdrouterd
- Improve PostgreSQL to handle more concurrent loads
- Configure the storage for DB workloads
- Consider storage needs and network for compatibility with MongoDB
- Ensure the storage requirements for Content Views are met
- Ensure the system requirements are met
- Improve the environment for remote execution

CHAPTER

SEVEN

CONFIGURING YOUR ENVIRONMENT FOR PERFORMANCE

7.1 CPU

The more physical cores that are available to Satellite 6.7, the higher throughput can be achieved for the tasks. Some of the Satellite components such as Puppet, MongoDB, PostgreSQL are CPU intensive applications and can really benefit from the higher number of available CPU cores.

7.2 Memory

The higher amount of memory available in the system running Satellite, the better will be the response times for the Satellite operations. Since Satellite uses PostgreSQL and MongoDB as the database solutions, any additional memory coupled with the tunings will provide a boost to the response times of the applications due to increased data retention in the memory.

7.3 Disk

With Satellite doing heavy IOPS due to repository synchronizations, package data retrieval, high frequency database updates for the subscription records of the content hosts, it is advised that Satellite be installed on a high speed SSD drive so as to avoid performance bottlenecks which may happen due to increased Disk reads or writes. Satellite 6 requires disk IO to be at or above 60-80 megabytes per second of average throughput for read operations. Anything below this value can have severe implications for the operation of the Satellite.

7.3.1 Benchmark disk performance

We are working to update foreman-maintain to only warn the users when its internal quick 'fio' benchmark results in numbers below our recommended throughput but will not require a whitelist parameter to continue.

Also working on an updated benchmark script you can run (which will likely be integrated into foreman-maintain in the future) to get a more accurate real-world storage information.

Note:

- One may have to temporarily reduce the RAM in order to run the io benchmark, aka if the box has 256GB that is a lot of pulp space, so add mem=20G kernel option in grub. This is needed because script will execute a series of fio based IO tests against a targeted directory specified in its execution. This test will create a very large file that is double (2x) the size of the physical RAM on this system to ensure that we are not just testing the caching at the OS level of the storage.

- Please bear above in mind when performing benchmark of other filesystems if you have them (like PostgreSQL or MongoDB storage) which might have significantly smaller capacity than Pulp storage and perhaps on different set of storage (SAN, iSCSI, etc).

This test does not use directio and will utilize the OS + caching as normal operations would.

You can find our first version of the script storage-benchmark. To execute just download to your Satellite, *chmod +x* the script and run:

```
# ./storage-benchmark /var/lib/pulp
This test creates a test file that is double (2X) the size of this system's
RAM in GB. This benchmark will create a test file of size:

64 Gigabytes

in the: [/var/lib/pulp/storage-benchmark] location. This is to ensure that the test␣
↪utilizes
a combination of cached and non-cached data during the test.

**** WARNING! Please verify you have enough free space to create a 64 GB
file in this location before proceeding.

Do you wish to proceed? (Y/N) Y


Starting IO tests via the 'fio' command. This may take up to an hour or more
depending on the size of the files being tested. Be patient!

************* Running READ test via fio:

read-test: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B,␣
↪ioengine=psync, iodepth=1
fio-3.1
Starting 1 process
...
```

As noted in the README block in the script: generally you wish to see on average 100MB/sec or higher in the tests below:

- Local SSD based storage should values of 600MB/sec or higher.
- Spinning disks should see values in the range of 100-200MB/sec or higher.

If you see values below this, please open a support ticket for assistance.

Refer this blog for more detailed info.

7.4 Network

The communication between the Satellite and Capsules is impacted by the network performance. A decent network with a minimum jitter and low latency is required to enable hassle free operations such as Satellite and Capsule synchronization (at least make sure it is not causing connection resets, etc).

7.5 Server Power Management

Your server by default is likely to be configured to conserve power. While this is a good approach to keep the max power consumption in check, it also has a side effect of lowering the performance that Satellite may be able to achieve. For a server running Satellite, it is recommended to set the BIOS to enable the system to be run in performance mode to boost the maximum performance levels that Satellite can achieve.

CHAPTER

EIGHT

SATELLITE CONFIGURATION TUNING

Red Hat Satellite as a product comes with a number of components that communicate with each other to produce a final outcome. All these components can be tuned independently of each other to achieve the maximum possible performance for the scenario desired.

8.1 Tuned profile

Red Hat Enterprise Linux 7 enables the tuned daemon by default during installation. On bare-metal, it is recommended that Red Hat Satellite 6 and capsule servers 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:

```
# service tuned start
# chkconfig tuned on
RHEL 7 (bare-metal):
# tuned-adm profile throughput-performance
RHEL 7 (virtual machine)
# tuned-adm profile virtual-guest
```

Transparent Huge Pages is a memory management technique used by the Linux kernel which reduces the overhead of using Translation Lookaside Buffer (TLB) by using larger sized memory pages. Due to databases having Sparse Memory Access patterns instead of Contiguous Memory access patterns, database workloads often perform poorly when Transparent Huge Pages is enabled. To improve performance of MongoDB, Red Hat recommends Transparent Huge Pages be disabled. For details on disabling Transparent Huge Pages, see Red Hat Solution.

8.2 Apache HTTPD Performance Tuning

Apache httpd forms a core part of the Satellite and acts as a web server for handling the requests that are being made through the Satellite Web UI or exposed APIs. To increase the concurrency of the operations, httpd forms the first point where tuning can help to boost the performance of the Satellite.

8.3 Configuring how many processes can be launched by Apache httpd

The version of Apache httpd that ships with Red Hat Satellite 6 by default uses prefork request handling mechanism. With the prefork model of handling the requests, httpd will launch a new process to handle the incoming connection by the client.

When the number of requests to the apache exceed the maximum number of child processes that can be launched to handle the incoming connections, an HTTP 503 Service Unavailable Error is raised by Apache.

Amidst httpd running out of processes to handle the incoming connections can also result in multiple component failure on the Satellite side due to the dependency of components like Passenger, Pulp on the availability of httpd processes.

Based on your expected peak load, you might want to modify the configuration of apache prefork to enable it to handle more concurrent requests.

An example modification to the prefork configuration for a server which may desire to handle 150 concurrent content host registrations to Satellite may look like the configuration file example that follows (see how to use *custom-hiera.yaml* file; this will modify config file /etc/httpd/conf.modules.d/prefork.conf):

```
File: /etc/foreman-installer/custom-hiera.yaml
apache::mod::prefork::serverlimit: 582
apache::mod::prefork::maxclients: 582
apache::mod::prefork::startservers: 10
```

In the above example, the ServerLimit parameter is set only to be able to raise MaxClients value.

The MaxClients (see MaxRequestWorker which is a new name in Apache docs) parameter is being used to set the maximum number of child processes that httpd can launch to handle the incoming requests.

The StartServers parameter defines how many server processes will be launched by default when the httpd process is started.

Note: While accounting for the limits to be set for the ServerLimit, please take a note of the values for PassengerMaxPoolSize, PassengerMaxRequestQueueSize, number of hosts that may run the client registration in parallel and max number of pulp processes that can be launched. The following formula can be used to estimate the value of ServerLimit parameter:

> ServerLimit = PassengerMaxPoolSize + PassengerMaxRequestQueueSize + Amount of content hosts being registered in parallel + number of launchable pulp processes

For example, an adequate value for ServerLimit on a Satellite 6.5 configuration that has PassengerMaxPoolSize set to 24, PassengerMaxRequestQueueSize set to 400 and expecting to register 150 content hosts in parallel with pulp configuration not being modified, can be calculated as shown below:

> ServerLimit = 24 + 400 + 150 + 8 = 582

8.4 Increasing the MaxOpenFiles Limit

With the tuning in place, apache httpd can easily open a lot of file descriptors on the server which may exceed the default limit of most of the linux systems in place. To avoid any kind of issues that may arise as a result of exceeding max open files limit on the system, please create the following file and directory and set the contents of the file as specified in the below given example:

```
File: /etc/systemd/system/httpd.service.d/limits.conf
[Service]
LimitNOFILE=640000
```

Once the file has been edited, the following commands need to be run to make the tunings come into effect::
systemctl daemon-reload foreman-maintain service restart

8.5 Calculating the maximum open files limit for qdrouterd

Calculate the limit for open files in qdrouterd using this formula: (Nx3) + 100, where N is the number of content hosts. Each content host may consume up to three file descriptors in the router, and 100 filedescriptors are required to run the router itself.

The following settings permit Satellite to scale up to 10,000 content hosts.

8.6 qdrouterd settings

Add/Update qpid::router::open_file_limit in custom-hiera.yaml as shown below:

```
File: /etc/foreman-installer/custom-hiera.yaml
qpid::router::open_file_limit: 150100
```

Note The change must be applied via:

```
# satellite-installer
# systemctl daemon-reload
# foreman-maintain service restart
```

8.7 Calculating the maximum open files limit for qpidd

Calculate the limit for open files in qpidd using this formula: (Nx4) + 500, where N is the number of content hosts. A single content host can consume up to four file descriptors and 500 file descriptors are required for the operations of Broker (a component of qpidd).

8.8 qpidd settings

Add/Update qpid::open_file_limit in /etc/foreman-installer/custom-hiera.yaml as shown below:

```
File: /etc/foreman-installer/custom-hiera.yaml
qpid::open_file_limit: 65536
```

Note The change must be applied via:

```
# satellite-installer
# systemctl daemon-reload
# foreman-maintain service restart
```

8.9 Maximum asynchronous input-output (AIO) requests

Increase the maximum number of allowable concurrent AIO requests by increasing the kernel parameter fs.aio-max-nr.1. Edit configuration file /etc/sysctl.conf, setting the value of fs.aio-max-nr to the desired maximum.

fs.aio-max-nr=23456

In this example, 23456 is the maximum number of allowable concurrent AIO requests.

This number should be bigger than 33 multiplied by the maximum number of the content hosts planned to be registered to Satellite. To apply the changes:

sysctl -p

Rebooting the machine also ensures that this change is applied.

8.10 Storage Considerations

Plan to have enough storage capacity for directory /var/lib/qpidd in advance when you are planning an installation that will use katello-agent extensively. In Red Hat Satellite 6, /var/lib/qpidd requires 2MB disk space per content host. See this bug for more details.

8.11 mgmt-pub-interval setting

You might see the following error in /var/log/journal in Red Hat Enterprise Linux 7:

```
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␣
↪Connectionqpid.10.1.10.1:5671-10.1.10.1:53790 timed out: closing
```

This error message appears because qpid maintains management objects for queues, sessions, and connections and recycles them every ten seconds by default. The same object with the same ID is created, deleted, and created again. The old management object is not yet purged, which is why qpid throws this error. Here's a workaround: lower the mgmt-pub-interval parameter from the default 10seconds to something lower. Add it to /etc/qpid/qpidd.conf and restart the qpidd service. See also Bug 1335694 comment 7.

8.12 Passenger Tuning

Passenger is a ruby application server which is used for serving the Foreman related requests to the clients. Passenger executes as a module inside the Apache httpd2 and handles the incoming requests directed towards the use of Foreman API or Satellite UI.

For any Satellite configuration that is supposed to handle a large number of clients or frequent operations, it is important for the Passenger to be tuned appropriately.

The below snippet provides an idea for tuning Passenger (see *how to use custom-hiera.yaml* file; this will modify /etc/httpd/conf.modules.d/passenger_extra.conf file):

```
File: /etc/foreman-installer/custom-hiera.yaml
apache::mod::passenger::passenger_max_pool_size: 48
apache::mod::passenger::passenger_max_request_queue_size: 400
```

In the above example, we set the tuning for two important keys inside Passenger:

PassengerMaxPoolSize: The parameter defines how many ruby application instances can be launched by Passenger once the process has started. To calculate an optimal value for the parameter, multiply the total number of VCPUs available in your deployment by 2 and that is the value for the PassengerMaxPoolSize parameter.

PassengerMaxRequestQueueSize: The PassengerMaxRequestQueueSize parameter defines how many requests can passenger queue for processing. The value for this parameter should never exceed the value of ServerLimit parameter set for the Apache httpd2.

8.13 Foreman Tuning

Foreman is the central application which provides the majority of the Satellite functionality as well as the GUI of Satellite. Under heavy load, the Foreman might need some amount of scaling up so as to provide adequate response times to the incoming requests.

Installer option "–foreman-passenger-min-instances 12" (defaults to 1) can be used to tune Foreman application (that will set "PassengerMinInstances" in /etc/httpd/conf.d/05-foreman-ssl.conf file).

PassengerMinInstances: The configuration key tells how many application instances should be running every time even when no load is being experienced by the application. To calculate an optimal value for the configuration key, divide the value of PassengerMaxPoolSize by 2. One of the repercussions that may be seen with such a tuning is the increased memory usage on the system attributed to the fact of having more foreman instances running during ideal conditions.

8.14 Dynflow Tuning

Dynflow is the workflow management system and task orchestrator which is built as a plugin inside Foreman and is used to execute the different tasks of Satellite in an out-of-order execution manner. Under the conditions when there are a lot of clients checking in on Satellite and running a number of tasks, the Dynflow can take some help from an added tuning specifying how many executors can it launch.

The following configuration snippet provides more information about the tunings involved related to Dynflow:

```
File: /etc/sysconfig/dynflowd:

EXECUTORS_COUNT=2
```

In the above tuning example, we worked with one configuration key:

EXECUTORS_COUNT: The key is used to configure how many executors will be launched by the Dynflow to handle the workflow management and task orchestration jobs. Usually an optimal value for this is in the range of 1-5 with diminishing gains if taken beyond that. Some of the tasks which may see an improvement over with the increased executor count is the ability to handle more number of concurrent package reporting from the content hosts.

8.15 PostgreSQL Tuning

PostgreSQL is the primary SQL based database that is used by Satellite for the storage of persistent context across a wide variety of tasks that Satellite does. The database sees an extensive usage is usually working on to provide the Satellite with the data which it needs for its smooth functioning. This makes PostgreSQL a heavily used process which if tuned can have a number of benefits on the overall operational response of Satellite.

The below set of tunings can be applied to PostgreSQL to improve its response times (see *how to use custom-hiera.yaml* file; this will modify /var/lib/pgsql/data/postgresql.conf file):

```
File: /etc/foreman-installer/custom-hiera.yaml
postgresql::server::config_entries:
  max_connections: 1000
  shared_buffers: 2GB
  work_mem: 8MB
  checkpoint_segments: 32
  autovacuum_vacuum_cost_limit: 2000
```

In the above tuning configuration, there are a certain set of keys which we have altered:

max_connections: The key defines the maximum number of connections that can be accepted by the PostgreSQL processes that are running. An optimal value for the parameter will be equal to the nearest multiple of 100 of the ServerLimit value of Apache httpd2 multiplied by 2. For example, if ServerLimit is set to 582, we can set the max_connections to 1000.

shared_buffers: The shared buffers define the memory used by all the active connections inside postgresql to store the data for the different database operations. An optimal value for this will vary between 2 GB to a maximum of 25% of your total system memory depending upon the frequency of the operations being conducted on Satellite.

work_mem: The work_mem is the memory that is allocated on per process basis for Postgresql and is used to store the intermediate results of the operations that are being performed by the process. Setting this value to 8 MB should be more than enough for most of the intensive operations on Satellite.

checkpoint_segments: The key defines the threshold after which the database should flush the contents WAL. Every segment is usually of 16 MB in size and the total checkpoint size is defined by checkpoint_segments multiplied by 16 MB. Once this much amount of WAL logs are filled, a checkpoint occurs flushing the contents of the WAL to storage.

autovacuum_vacuum_cost_limit: The key defines the cost limit value for the vacuuming operation inside the autovacuum process to clean up the dead tuples inside the database relations. The cost limit defines the number of tuples that can be processed in a single run by the process. An optimal value for this is 2000 based on the general load that Satellite pushes on the PostgreSQL server process.

8.16 Benchmarking raw DB performance

To get a list of the top table sizes in disk space for both Candlepin and Foreman, check postgres-size-report script in satellite-support git repository.

PGbench utility (note you may need to resize PostgreSQL data directory /var/lib/pgsql/ directory to 100GB or what does benchmark take to run) might be used to measure PostgreSQL performance on your system. Use yum install postgresql-contrib to install it. Some resources are:

- https://github.com/RedHatSatellite/satellite-support

Choice of filesystem for PostgreSQL data directory might matter as well:

- https://blog.pgaddict.com/posts/postgresql-performance-on-ext4-and-xfs

Note:

- Never do any testing on production system and without valid backup.
- Before you start testing, see how big the database files are. Testing with a really small database would not produce any meaningful results. E.g. if the DB is only 20G and the buffer pool is 32G, it won't show problems with large number of connections because the data will be completely buffered.

8.17 MongoDb Tuning

Under certain circumstances, mongod consumes randomly high memory (up to 1/2 of all RAM) and this aggressive memory usage limits other processes or can cause OOM killer to kill mongod. In order to overcome this situation, tune the cache size by referring the following steps:

1. Update custom-hiera.yaml:

Edit /etc/foreman-installer/custom-hiera.yaml and add the entry below inserting the value that is 20% of the physical RAM while keeping in mind the guidlines in this case, approximately 6GB for a 32GB server. Please note the formatting of the second line and the indent:

```
mongodb::server::config_data:
 storage.wiredTiger.engineConfig.cacheSizeGB: 6
```

2. Run installer to apply changes:

```
# satellite-installer
```

For more details, please refer to this Kbase article.

8.18 Benchmarking raw performance

To get a size report of MongoDB, use mongo-size-report from satellite-support repository.

Utility used for checking IO speed specific to MongoDB:

- https://www.mongodb.com/blog/post/checking-disk-performance-with-the-mongoperf

For MongoDB benchmark meant to run on (stage) Satellite installs, check mongo-benchmark tool in satellite-support git repository.

Depending on a disk drive type, file system choice (ext4 or xfs) for MongoDB storage directory might be important:

- https://scalegrid.io/blog/xfs-vs-ext4-comparing-mongodb-performance-on-aws-ec2/

Note:

- Never do any testing on production system and without valid backup.

~