Red Hat Satellite 6.4

Tuning Red Hat Satellite

Performance tuning for Red Hat Satellite 6

Last Updated: 2018-10-03

Red Hat Satellite 6.4 Tuning Red Hat Satellite

Performance tuning for Red Hat Satellite 6

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

Pradeep Surisetty
psuriset@redhat.com

Jan Hutar
jhutar@redhat.com

Saurabh Badhwar
sbadhwar@redhat.com

Legal Notice

Abstract

How to tune your Red Hat Satellite 6 environment.

Table of Contents

CHAPTER 1. INTRODUCTION

This document provides guidelines for tuning Red Hat Satellite for performance and scalability. There is no universally-applicable configuration for Red Hat Satellite. What maximizes performance in one situation will not necessarily maximize performance in another.

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, and subscription management, and does so while maintaining high scalability and security.

For more on Red Hat Satellite Server, refer to the documentation:

https://access.redhat.com/documentation/en/red-hat-satellite/?version=6.3

CHAPTER 2. SYSTEM REQUIREMENTS

For details of Red Hat Satellite's hardware and software requirements, see Preparing your environment for installation in the *Installation Guide*.

CHAPTER 3. TOP PERFORMANCE CONSIDERATIONS

This is a list of things that you can do to improve the performance of your installation of Red Hat Satellite.

1. Configure httpd
2. Configure Passenger to increase concurrency
3. Configure Candlepin
4. Configure Pulp
5. Configure Foreman's performance and scalability
6. Configure Puppet for scalability
7. Deploy external Capsule(s) in lieu of the integrated Capsule
8. Configure katello-agent for scalability
9. Configure Hammer to reduce API timeouts
10. Configure qpidd and qdrouterd
11. Improve PostgreSQL tuning to increase concurrent registrations of hosts
12. Configure the storage for DB workloads
13. Consider storage needs and network compatibility with MongoDB
14. Ensure the storage requirements for Content Views are met
15. Ensure the system requirements are met
16. Improve the environment for remote execution

CHAPTER 4. CONFIGURING YOUR ENVIRONMENT FOR PERFORMANCE

4.1. CPU

The more physical CPU cores that are available to Satellite 6.3, the greater the throughput of tasks. It is important to balance throughput and task latency in a way that meets your needs. More CPU cores improve the scale of Red Hat Satellite 6's deployment and should be the first consideration when you choose CPU hardware.

4.2. MEMORY

Provide adequate memory for your Red Hat Satellite installation. When determining how much memory you need, remember that you must provide memory for the following processes: Apache, Foreman, MongoDB, PostgreSQL, Pulp, Puppet, Tomcat, Qpid, and the file system cache. A performant system does not swap even when Apache, Foreman, Puppet, or any other software is scaled to maximum on a single server.

4.3. DISK

Input/Output Operations Per Second (IOPS) and disk capacity should be taken into account when determining what kind of disk to use. The file system can be partitioned over separate disks if necessary to increase capacity and IOPS on the directories that are most often accessed. The critical directories for IOPS and monitoring are **`/var/lib/pulp`**, **`/var/lib/pgsql`**, and **`/var/lib/mongodb`**.

4.4. NETWORK

During testing, network hardware was found consistently not to be a bottleneck before CPU or configuration limits were reached. This result was achieved on a 10Gb network that connected (1) Red Hat Satellite capsule servers and (2) an emulated content delivery network (CDN). The internet connection to the CDN is likely to be a bottleneck, but connections to the internet are beyond the scope of this document.

4.5. SERVER POWER MANAGEMENT

Your server is likely set by default to conserve power. This often diminishes performance. Configure your server's BIOS to allow OS Host Power Control, and Red Hat Enterprise Linux will be able to control your server's power consumption to improve the performance of Red Hat Satellite. Remember that power consumption comes at the expense of performance, and vice versa. Tune your Red Hat Satellite environment to provide the balance that is best for your situation.

CHAPTER 5. TUNING

5.1. CONFIGURING APACHE HTTP SERVER FOR SATELLITE PERFORMANCE

5.1.1. Overview

The Apache HTTP Server is a core component of Satellite. Passenger and Pulp, which are core components of Satellite, depend upon Apache HTTP Server to serve incoming requests. Requests that arrive through the web UI or the Satellite API are received by Apache HTTP Server and then forwarded to the components of Satellite that operate on them.

The version of Apache HTTP Server that ships with Red Hat Satellite 6.3 uses a process-based model of request handling. The Apache HTTP Server spawns a new process each time there is a new incoming request. Each new process takes up space in memory. The maximum number of processes that Apache HTTP Server can spawn is usually governed by its configuration. It is possible to change this number so that Apache HTTP Server can handle a larger number of requests.

When there is a burst of requests at the same time, this can exceed the capacity of Apache HTTP Server to handle requests, which causes new incoming requests to receive HTTP 503 responses.

The two settings that govern how quickly Apache HTTP Server is able to respond to incoming requests during bursts are found in **`/etc/httpd/conf.modules.d/prefork.conf`**. These two settings are:

```
ServerLimit 512
StartServers 10
```

The **`ServerLimit`** parameter defines the maximum number of child processes that Apache HTTP Server is able to spawn to handle new incoming requests. As this setting is increased, the amount of memory used by Apache HTTP Server increases each time a burst of requests arrives at the Satellite and Capsule Servers.

The **`StartServer`** parameter defines the number of child processes launched by Apache HTTP Server when it starts. Increase this number to improve response times for new incoming requests. Increasing this number creates a situation in which requests do not have to wait for new child processes to be spawned before they are responded to.

5.1.2. Tuning KeepAlive settings

Apache HTTP Server uses the **`KeepAlive`** directive to manage TCP connections. When **`KeepAlive`** is set to **`On`**, two variables control how TCP connections are used.

`KeepAliveTimeout` specifies how long a connection should be kept open. Because establishing and re-establishing connections consumes resources, keeping frequently used connections alive improves performance. Conversely, keeping connections alive for too long ties up resources as well, and can cause performance to drop.

Turn on Apache HTTP Server's **`KeepAlive`** tunable and set values for **`KeepAliveTimeout`** and **`MakeKeepAliveRequests`** to reduce the number of TCP connections and usage of the CPU. Red Hat recommends that you set **`KeepAliveTimeout`** to between 2 and 5 seconds unless there is latency between Red Hat Satellite and the Capsules or hosts. Red Hat recommends that **`MaxKeepAliveRequests`** be set to **`0`** to allow each connection to request all its content over the same

TCP connection. The time that it takes to load a page on the web server is lower if **KeepAlive** is on. The default configuration for **KeepAlive** for Red Hat Satellite is found in **/etc/httpd/conf.d/05-foreman-ssl.conf**:

Example Red Hat Satellite 6 Apache HTTP Server configuration tuning:

```
<VirtualHost>
KeepAlive On
MaxKeepAliveRequests 0
KeepAliveTimeout 5
</VirtualHost>
```

5.1.3. Maximum open files limit

Increase the maximum number of files that can be open when doing many registrations or when increasing the scale of Capsules, Content Hosts, and Content Views.

Edit configuration file **/etc/systemd/system/httpd.service.d/limits.conf**, setting the value for **LimitNOFILE**.

NOTE

Create the path and configuration file if they do not exist.

In the following example the maximum number of open files is set to 640000:

```
# cat /etc/systemd/system/httpd.service.d/limits.conf
[Service]
LimitNOFILE=640000
# systemctl daemon-reload
# foreman-maintain service restart
```

The maximum open files limit can be validated with the following command:

```
# 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
```

5.2. PASSENGER CONFIGURATION

5.2.1. Overview

Passenger is a web server and a core component of Red Hat Satellite. Satellite uses Passenger to run Ruby applications such as Foreman and Puppet 3. Passenger integrates with Apache HTTP Server to capture incoming requests and redirects them to the respective components that handle them.

Passenger is involved in Satellite when the GUI is accessed, when the APIs are accessed, and when content hosts are registered. Each request that is serviced by Passenger consumes an Apache HTTP Server process. Passenger queues requests into an application-specific wait queue. The maximum

number of requests that can be queued by Passenger is defined in the Passenger configuration. When running at scale, it might be desirable to increase the number of requests that Passenger can handle concurrently. It might also be desirable to increase the size of the wait queue to accommodate bursts of requests.

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

5.2.2. Tuning Passenger for Foreman-related workloads

NOTE

The tunings in this section apply to **`/etc/httpd/conf.d/passenger.conf`**.

This section applies to instances where your system has 12 or more VCPUs.

For deployments that consist of workloads that do not require large scale and do not require much use of Puppet, increase the pool size for Passenger. The pool size defines how many application processes passenger is able to launch to handle incoming requests.

Set the pool size to two times the number of VCPUs in the system. For example, if your system has 24 VCPUs, set the **`PassengerMaxPoolSize`** to 48.

Increasing the value of **`PassengerMaxPoolSize`** to a value greater than two times the number of VCPUs in the system can cause processes to contend with one another for CPU resources, and this can cause an increase in request timeouts.

5.2.3. Tuning Passenger for Puppet 3-related workloads

NOTE

The tunings in this section apply to **`/etc/httpd/conf.d/passenger.conf`**.

This section applies to instances where your system has 12 or more VCPUs.

For deployments that consist of workloads that are heavily dependent on Puppet 3, increase the pool size for Passenger. The pool size defines how many application processes Passenger is able to launch to handle incoming requests.

Set the pool size to the same number as the number of VCPUs in your system. For example, if your system has 24 VCPUs, set **`PassengerMaxPoolSize`** to 24.

5.2.4. Tuning Passenger for Puppet 4-related workloads

NOTE

The tunings in this section apply to **`/etc/httpd/conf.d/passenger.conf`**.

This section applies to instances where your system has 12 or more VCPUs.

For deployments that consist of workloads that are heavily dependent on Puppet 4, increase the pool size for Passenger. The pool size defines how many application processes Passenger is able to launch to handle incoming requests.

Puppet 4 does not require Passenger to launch its server processes, so the entire Passenger process pool can be dedicated to launching Foreman processes.

Set the pool size to 1.5 multiplied by the number of VCPUs in your system. For example, if your system has 24 VCPUs, set **`PassengerMaxPoolSize`** to 36.

In larger, scaled-out setups in which large numbers of requests can arrive in bursts, set the value of **`PassengerMaxRequestQueueSize`** to 400.

NOTE

The value for **`PassengerMaxRequestQueueSize`** should always be lower than the Apache HTTP Server **`ServerLimit`** value. Increasing this value beyond the **`ServerLimit`** threshold causes other components to become starved of the http-process pool, which leads to increased rejection of requests and action failures.

5.2.5. The "passenger-status" command

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

5.3. CANDLEPIN

Candlepin is a collection of tools that facilitates the management of software subscriptions. It is a part of Katello, which provides a unified workflow and web-based user interface for content and subscriptions. Candlepin provides the component of Katello related to subscriptions.

The complexity of your subscriptions determines how much latency is needed to complete a registration. More latency is required to complete registrations when your configuration has a large number of Foreman processes and a large Passenger queue size.

You can determine how much latency is required to complete a registration by timing your subscription registrations. The following command is used to time subscription registrations:

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

Determine the minimum, average, and maximum times that it takes to complete a subscription registration in order to learn the capacity of the environment against which the timing measurements are taken. In the default Passenger configuration with Red Hat Satellite 6.3, if Foreman consumes all PassengerMaxPoolSize processes and if all application processes are preloaded, then six concurrent registrations are allowed. If there is only one process spawned, then additional preloader latency will be added to registration time. Additional concurrent registrations will experience additional latency due to queuing for an available Foreman process. Any other tasks or workloads involving Foreman will also join the queue and delay any other concurrent registrations.

5.4. PULP

Pulp is a software repository management tool written in Python. Pulp provides complete software repository management and the capability to mirror repositories, the capability to host repositories, and the capability to distribute the contents of those repositories to a large number of consumers.

Pulp manages RPM content, Puppet modules, and container images in Satellite. Pulp also publishes Content Views and creates local repositories from which Capsules and hosts retrieve content. The configuration of the Apache HTTP Server determines how efficiently Pulp REST API requests are handled.

Pulp depends upon **`celery`**, which is responsible for launching Pulp workers, which download data from upstream repositories. Pulp also depends upon Apache HTTP Server to provide access to Pulp's APIs and internal components.

If your Satellite environment requires the concurrent synchronization of a large number of software repositories, increase the number of workers that can be launched by Pulp.

Pulp is a component of Katello.

5.4.1. Storing content

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

5.4.2. Worker concurrency

To adjust the number of tasks that run in parallel, change the value of **`PULP_CONCURRENCY`** in the **`/etc/default/pulp_workers`** file. As Pulp synchronizes more repositories simultaneously, more workers are able to consume Satellite 6.3 resources. However, such configurations can starve other components of Satellite. It is important to experiment with the concurrency level of Pulp in an environment that has a concurrent workload such as Puppet. By default, on a system with less than 8 CPUs, **`PULP_CONCURRENCY`** is set to the number of CPUs. On a system with more than 8 CPUs, it is set to 8.

5.5. MONGODB

MongoDB is a NoSQL database server which is used by Pulp to store the metadata related to the synchronized repositories and their contents. Pulp also uses MongoDB to store information about Pulp tasks and their current state.

5.5.1. Disable Transparent Huge Pages

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 1320153.

5.6. NFS

5.6.1. /var/lib/pulp

Red Hat Satellite 6.3 uses **/var/lib/pulp** to store and manage repository content. Red Hat does not recommend that you run Pulp on NFS. Red Hat recommends instead that you use high-bandwidth, low-latency storage for the **/var/lib/pulp** filesystem. Red Hat Satellite has many operations that are IO-intensive, and that means that the use of high-latency, low-bandwidth storage could degrade the performance of Satellite 6.3.

5.6.2. /var/lib/mongodb

Pulp uses MongoDB. Red Hat recommends that you never use NFS with MongoDB.

5.7. FOREMAN

Foreman is a Ruby application that is spawned by Passenger and does a number of things, among them providing a UI, providing remote execution, running Foreman SCAP scans on content hosts. Foreman is also involved in Content Host Registrations.

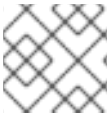
Foreman executes from within Passenger, which dynamically spawns new Foreman processes when new incoming requests arrive.

A single Foreman process services a number of requests before it is recycled by Passenger, releasing the memory that it consumed. The maximum number of Foreman processes that Passenger is able to spawn is governed by the Passenger **PassengerMaxPoolSize** parameter, which is found in **/etc/httpd/conf.d/passenger.conf**:

Foreman runs inside the Passenger application server. Foreman's performance and scalability are affected directly by the configurations of httpd and Passenger. Foreman also processes UI and API requests. Tuning on Apache HTTP Server's **KeepAlive** setting improves the time it takes to load the user-interface page. A properly configured **tuned** profile improves the response time of the CLI and API.

Foreman can be tuned to increase its number of processes, and to set the number of requests that a single process handles before it is recycled.

5.7.1. Tuning Foreman for workloads involving Puppet 3



NOTE

Edit configuration file **/etc/httpd/conf.d/05-foreman-ssl.conf**.

The value **PassengerMinInstances** specifies the minimum active instances of Foreman. The remaining slots are used to spawn Puppet.

Workloads involving Puppet 3

Set **PassengerMinInstances** to 6.

Workloads that do not involve Puppet 3

Set **PassengerMinInstances** to 10.

Workloads that make sparing use of Puppet 3

Set **PassengerMinInstances** to 10.

5.8. DYNFLOW

NOTE

These tunings apply to **`/etc/sysconfig/foreman-tasks`**.

DynFlow is a workflow system and task orchestration engine written in Ruby, and runs as a plugin to Foreman. Foreman uses DynFlow to schedule, plan, and execute queued tasks.

Tuning dynFlow makes possible the control of Ruby's usage of memory. In some cases, when running at scale, dynFlow memory usage can exceed 10 GB.

DynFlow can be configured to limit the memory use of workers and to increase the number of workers that are processing requests.

It is possible in Red Hat Satellite 6.3 to limit the memory used by dynFlow executor.

`EXECUTOR_MEMORY_LIMIT` defines the amount of memory that a single dynFlow executor process can consume before the executor is recycled.

`EXECUTOR_MEMORY_MONITOR_DELAY` defines when the first polling attempt to check the executor memory is made after the initialization of the executor.

`EXECUTOR_MEMORY_MONITOR_INTERVAL` defines how frequently the memory usage of executor is polled.

5.8.1. dynFlow tuning example

NOTE

These tunings apply to **`/etc/sysconfig/foreman-tasks`**.

For a scaled setup involving many operations, set the following executor memory values:

```
EXECUTOR_MEMORY_LIMIT: 3gb
EXECUTOR_MEMORY_MONITOR_DELAY: 3600
EXECUTOR_MEMORY_MONITOR_INTERVAL: 120
```

5.9. POSTGRESQL TUNING

NOTE

The tunings in this section apply to **`/var/lib/pgsql/data/postgresql.conf`**.

PostgreSQL is used by Foreman and Candlepin to store records related to registered content hosts, subscriptions, jobs, and tasks. Over time, PostgreSQL accumulates enough data to cause queries to slow relative to the speeds achievable in a fresh installation.

Increasing the memory to which PostgreSQL has access speeds operation execution.

If you want to increase concurrency, increase the number of connections that PostgreSQL is able to manage concurrently.

5.9.1. PostgreSQL tuning example

NOTE

The tunings in this section apply to **/var/lib/pgsql/data/postgresql.conf**.

The following tunings in **/var/lib/pgsql/data/postgresql.conf** provide performance gains, but also increase memory usage:

```
max_connections = 500
shared_buffers = 512MB
work_mem = 8MB
checkpoint_segments = 32
checkpoint_completion_target = 0.9
```

5.10. PUPPET

Red Hat Satellite 6.3 supports hosts with Puppet version 3.8 or later, and Puppet 4. Puppet 3 is a Ruby application and runs inside the Passenger application server. Puppet 4 runs as a standalone, Java-based, application. Several factors in Puppet affect the overall performance and scalability of Red Hat Satellite.

5.10.1. Run-interval

A non-deterministic run-interval that does not distribute the load throughout the interval causes scaling problems and errors in a Puppet environment. Evenly distributing the load allows a system to scale reliably and to handle more requests with fewer spikes. Run-interval can be distributed in the following ways:

1. Puppet splay - Turn on splay for each Puppet client. This adds randomization to the run-interval. This does not establish a deterministic run-interval.

2. A cron job - Run each Puppet agent as a cron job rather than as a daemon. This makes a run-interval deterministic. At scale, this approach becomes difficult to manage when adding or removing clients.

3. Separation - Deploy a separate entity to manage when a Puppet agent run occurs.

5.10.2. Other Puppet interactions

Measure other Puppet interactions that are performed in your environment. These interactions place a load on Red Hat Satellite Server and its Capsules, and these interactions include submitting facts, serving files, and submitting reports. Each of these interactions imposes a cost on the system.

5.11. EXTERNAL CAPSULES

External Capsules allow a Satellite deployment to scale out and to provide services local to the hosts that are managed by the external Capsules.

5.11.1. Advantages of external Capsules

1. Reduces the number of HTTP requests on Red Hat Satellite

2. Provides more CPU and Memory resources for Puppet and Foreman

3. Places resources closer to end clients and reduces latency in requests

5.11.2. Factors to consider when determining whether to use external Capsules

1. Runinterval — Timing between Puppet agent applies. This spreads the workload evenly over the interval

2. Client workload — the amount of work for each Puppet client during a Puppet agent run

3. Hardware Configuration — the amount of resources available for Puppet

Determining when to use an external Capsule versus an integrated Capsule depends on hardware, configuration, and workload. Plan against the Puppet requirements, because a number of variables in the Puppet workload will directly affect the scalability of Red Hat Satellite. Raising the runinterval directly increases the capacity at the cost of increasing the interval between which Puppet applies the configuration. Reducing the runinterval 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 Red Hat Satellite Server underutilized at other times. The amount of work that each Puppet client has to perform in order to complete a Puppet run changes scalability. Raising the configured number of Puppet processes improves scalability if physical hardware resources are available. The nature of these variables means that it is not constructive to provide a universally-applicable recommendation regarding when it is wise to move to an external Capsule. Benchmark a specific Puppet workload to determine its scalability.

5.11.3. Hardware considerations and httpd/Passenger configuration

Apply the hardware considerations that are listed in this document to Capsules. Virtualized Capsules make it possible to tune the number of vCPUs and available memory as long as the Capsule is not colocated on a machine that hosts virtual machines that overcommit the host's resources. Apache HTTP Server and Passenger configuration considerations also apply to the Capsule in the context of Puppet.

5.11.4. Example Capsule httpd configuration tuning

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

5.11.5. 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
    PassengerStatThrottleRate 120
</IfModule>
```

5.11.6. 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
```

5.12. CLIENT AGENT SCALING (KATELLO-AGENT)

The default timeout value is 20 seconds. If your clients require longer to answer, and return an error message such as **`Host did not respond within 20 seconds`**, increase the **Accept action timeout** value:

1. In the web UI, navigate to **Administer** > **Settings**, and then click the **Content** tab.
2. On the **Accept action timeout** row, click the edit icon in the **Value** column.
3. Enter the required timeout in seconds and click the confirmation icon.

5.13. SCALE: HAMMER TIMEOUT

During the scaling of Capsules, content hosts, or content views, hammer API requests can time out. Add the following line to **`/etc/hammer/cli.modules.d/foreman.yml`** to disable timeout:

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

5.14. QPIDD AND QDROUTERD CONFIGURATION

5.14.1. Max open files limit

To scale up Satellite and Capsules so that they support larger numbers of hosts, it is essential that processes should be able to open the required number of descriptors.

The following tunings are applicable both to Apache HTTP Server and qrouterd:

```
[Service]
LimitNOFILE=640000
```

If your deployment uses **`katello-agent`**, you must tune the file limits for **`qpidd`** and **`qrouterd`**.

5.14.1.1. Calculating the maximum open files limit for qrouterd

Calcluate the limit for open files in **`qrouterd`** 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 file descriptors are required to run the router itself.

5.14.1.2. Calculating the maximum open files limit for qpidd

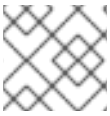
Calculate the limit for open files in **`qpidd`** using this formula: (Nx4) + 500, where N is the number of content hosts. A single content host can consume up to four file descriptors and 500 file descriptors are required for the operations of Broker (a component of **`qpidd`**).

5.14.1.3. Example configurations

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

5.14.1.3.1. qdrouterd settings

Edit configuration file **`/etc/systemd/system/qdrouterd.service.d/limits.conf`**, setting the value of **`LimitNOFILE`** to **`301000`**.



NOTE

Create the path and configuration file if they do not exist.

```
[Service]
LimitNOFILE=301000
```

5.14.1.3.2. qpidd settings

Edit configuration file **`/etc/systemd/system/qpidd.service.d/limits.conf`**, setting the value of **`LimitNOFILE`** to **`40500`**.

NOTE

Create the path and configuration file if they do not exist.

```
[Service]
LimitNOFILE=40500
```

See also Red Hat Solution 1375253 for more detail.

5.14.2. 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=12345
   ```

   In this example, *12345* 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.

1. Apply the changes:

   ```
   # sysctl -p
   ```

   Rebooting the machine also ensures that this change is applied.

See also Red Hat Solution 1425893 for more detail.

5.14.3. Storage consideration

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 6.3, **/var/lib/qpidd** requires 2MB disk space per content host (see Bug 1366323).

The following line is sufficient in new installations:

```
cat /usr/share/katello-installer-
base/modules/qpid/templates/qpidc.conf.erb efp-file-size=256
```

5.14.4. mgmt-pub-interval settings

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 Connection
qpid.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 10 seconds to something lower. Add it to **/etc/qpid/qpidd.conf** and restart the qpidd service. See also Bug 1335694 comment 7.