Red Hat Satellite 6.15

Tuning performance of Red Hat Satellite

Optimize performance for Satellite Server and Capsule

Last Updated: 2024-04-22

Red Hat Satellite 6.15 Tuning performance of Red Hat Satellite

Optimize performance for Satellite Server and Capsule

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

Legal Notice

Abstract

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

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. Because of the enormity of this endeavor, these changes are being updated gradually and where possible. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better.

You can submit feedback by filing a ticket in Bugzilla:

1. Navigate to the Bugzilla website.
2. In the **Component** field, use **Documentation**.
3. In the **Description** field, enter your suggestion for improvement. Include a link to the relevant parts of the documentation.
4. Click **Submit Bug**.

CHAPTER 1. INTRODUCTION TO PERFORMANCE TUNING

This document provides guidelines for tuning Red Hat Satellite 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.

CHAPTER 2. PERFORMANCE TUNING QUICKSTART

You can tune your Satellite Server based on expected host counts and hardware allocation using built in tuning profiles included in Satellite that are available using the installation routine's tuning flag. For more information, see Tuning Satellite Server with Predefined Profiles in *Installing Satellite Server in a connected network environment*.

There are four sizes provided based on estimates of the number of hosts your Satellite manages. You can find the specific tuning settings for each profile in the configuration files contained in **/usr/share/foreman-installer/config/foreman.hiera/tuning/sizes**.

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

Procedure

1. Select an installation size: **default**, **medium**, **large**, **extra-large**, or **extra-extra-large**. The default value is **default**.
2. Run **satellite-installer**:

```
# satellite-installer --tuning "My_Installation_Size"
```

3. Optional: Run a health check. For more information, see Section 5.1, "Applying configurations".
4. Optional: Tune the Ruby app server directly using the Puma Tuning section. For more information, see Section 5.2, "Puma tunings".

CHAPTER 3. SYSTEM REQUIREMENTS FOR TUNING

You can find the hardware and software requirements in Preparing your Environment for Installation in *Installing Satellite Server in a connected network environment* .

CHAPTER 4. DETERMINING HARDWARE AND OS CONFIGURATION

CPU

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

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 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.

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 so as to avoid performance bottlenecks which may happen due to increased disk reads or writes. Satellite 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. Satellite components such as PostgreSQL benefit from using SSDs due to their lower latency compared to HDDs.

Network

The communication between the Satellite Server 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 Server and Capsules synchronization (at least ensure it is not causing connection resets, etc).

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.

4.1. BENCHMARKING DISK PERFORMANCE

We are working to update **satellite-maintain** to only warn the users when its internal quick storage benchmark results in numbers below our recommended throughput.

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

NOTE

- You may have to temporarily reduce the RAM in order to run the I/O benchmark. For example, if your Satellite Server has 256 GiB RAM, tests would require 512 GiB of storage to run. As a workaround, you can add **mem=20G** kernel option in grub during system boot to temporary reduce the size of the RAM. The benchmark creates a file twice the size of the RAM in the specified directory and executes a series of storage I/O tests against it. The size of the file ensures that the test is not just testing the filesystem caching. If you benchmark other filesystems, for example smaller volumes such as PostgreSQL storage, you might have to reduce the RAM size as described above.
- If you are using different storage solutions such as SAN or iSCSI, you can expect a different performance.
- Red Hat recommends you to stop all services before executing this script and you will be prompted to do so.

This test does not use direct I/O and will utilize file caching as normal operations would.

You can find our first version of the script storage-benchmark. To execute it, just download the script to your Satellite, make it executable, and run:

```
# ./storage-benchmark /var/lib/pulp
```

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 give values of 600MB/sec or higher.
- Spinning disks should give values in the range of 100 – 200MB/sec or higher.

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

For more information, see Impact of Disk Speed on Satellite Operations .

4.2. ENABLING TUNED PROFILES

On bare-metal, Red Hat recommends to run the **throughput-performance** tuned profile on Satellite Server and Capsules. On virtual machines, Red Hat recommends to run the **virtual-guest** profile.

Procedure

1. Check if **tuned** is running:

```
# systemctl status tuned
```

2. If **tuned** is not running, enable it:

```
# systemctl enable --now tuned
```

3. Optional: View a list of available **tuned** profiles:

```
# tuned-adm list
```

4. Enable a **tuned** profile depending on your scenario:

```
# tuned-adm profile "My_Tuned_Profile"
```

4.3. DISABLE TRANSPARENT HUGEPAGE

Transparent Hugepage is a memory management technique used by the Linux kernel to reduce the overhead of using the 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 Hugepage is enabled. To improve PostgreSQL and Redis performance, disable Transparent Hugepage. In deployments where the databases are running on separate servers, there may be a small benefit to using Transparent Hugepage on the Satellite Server only.

For more information on how to disable Transparent Hugepage, see How to disable transparent hugepages (THP) on Red Hat Enterprise Linux.

CHAPTER 5. CONFIGURING SATELLITE FOR PERFORMANCE

Satellite comes with a number of components that communicate with each other. You can tune these components independently of each other to achieve the maximum possible performance for your scenario.

You will see no significant performance difference between Satellite installed on Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux 8.

5.1. APPLYING CONFIGURATIONS

In following sections we suggest various tunables and how to apply them. Please always test changing these in non production environment first, with valid backup and with proper outage window as in most of the cases Satellite restart is required.

It is also a good practice to setup a monitoring before applying any change as it will allow you to evaluate effect of the change. Our testing environment might be too far from what you will see although we are trying hard to mimic real world environment.

Changing systemd service files

If you have changed some systemd service file, you need to notify systemd daemon to reload the configuration:

```
# systemctl daemon-reload
```

Restart Satellite services:

```
# satellite-maintain service restart
```

Changing configuration files

If you have changed a configuration file such as **/etc/foreman-installer/custom-hiera.yaml**, rerun installer to apply your changes:

```
# satellite-installer
```

Running installer with additional options

If you need to rerun installer with some new options added:

```
# satellite-installer new options
```

Checking basic sanity of the setup

Optional: After any change, run this quick Satellite health-check:

```
# satellite-maintain health check
```

5.2. PUMA TUNINGS

Puma is a ruby application server which is used for serving the Foreman related requests to the clients. For any Satellite configuration that is supposed to handle a large number of clients or frequent operations, it is important for the Puma to be tuned appropriately.

5.2.1. Puma threads

Number of Puma threads (per Puma worker) is configured using two values: **threads_min** and **threads_max**.

Value of **threads_min** determines how many threads each worker spawns at a worker start. Then, as concurrent requests are coming and more threads is needed, worker will be spawning more and more workers up to **threads_max** limit.

We recommend setting **threads_min** to same value as **threads_max** as having fewer Puma threads lead to higher memory usage on your Satellite Server.

For example, we have compared these two setups using concurrent registrations test:

Satellite VM with 8 CPUs, 40 GiB RAM	Satellite VM with 8 CPUs, 40 GiB RAM
--foreman-foreman-service-puma-threads-min=0	**--foreman-foreman-service-puma-threads-min=16**
--foreman-foreman-service-puma-threads-max=16	**--foreman-foreman-service-puma-threads-max=16**
--foreman-foreman-service-puma-workers=2	**--foreman-foreman-service-puma-workers=2**

Setting the minimum Puma threads to **16** results in about 12% less memory usage as compared to **threads_min=0**.

5.2.2. Puma workers and threads auto-tuning

If you do not provide any Puma workers and thread values with **satellite-installer** or they are not present in your Satellite configuration, the **satellite-installer** configures a balanced number of workers. It follows this formula:

```
min(CPU_COUNT * 1.5, RAM_IN_GB - 1.5)
```

This should be fine for most cases, but with some usage patterns tuning is needed to either limit the amount of resources dedicated to Puma (so other Satellite components can use these) or for any other reason. Each Puma worker consumes around 1 GiB of RAM.

View your current Satellite Server settings

```
# cat /etc/systemd/system/foreman.service.d/installer.conf
```

View the currently active Puma workers

```
# systemctl status foreman
```

5.2.3. Manually tuning Puma workers and threads count

If you decide not to rely on Section 5.2.2, "Puma workers and threads auto-tuning", you can apply custom numbers for these tunables. In the example below we are using 2 workers, 5 and 5 threads:

```
# satellite-installer \
--foreman-foreman-service-puma-workers=2 \
--foreman-foreman-service-puma-threads-min=5 \
--foreman-foreman-service-puma-threads-max=5
```

Apply your changes to Satellite Server. For more information, see Section 5.1, "Applying configurations".

5.2.4. Puma workers and threads recommendations

In order to recommend thread and worker configurations for the different tuning profiles, we conducted Puma tuning testing on Satellite with different tuning profiles. The main test used in this testing was concurrent registration with the following combinations along with different number of workers and threads. Our recommendation is based purely on concurrent registration performance, so it might not reflect your exact use-case. For example, if your setup is very content oriented with lots of publishes and promotes, you might want to limit resources consumed by Puma in favor of Pulp and PostgreSQL.

Name	Number of hosts	RAM	Cores	Recommended Puma Threads for both min & max	Recommended Puma Workers
default	0 – 5000	20 GiB	4	16	4 – 6
medium	5000 – 10000	32 GiB	8	16	8 – 12
large	10000 – 20000	64 GiB	16	16	12 – 18
extra-large	20000 – 60000	128 GiB	32	16	16 – 24
extra-extra-large	60000+	256 GiB +	48+	16	20 – 26

Tuning number of workers is the more important aspect here and in some case we have seen up to 52% performance increase. Although installer uses 5 min/max threads by default, we recommend 16 threads with all the tuning profiles in the table above. That is because we have seen up to 23% performance increase with 16 threads (14% for 8 and 10% for 32) when compared to setup with 4 threads.

To figure out these recommendations we used concurrent registrations test case which is a very specific use-case. It can be different on your Satellite which might have more balanced use-case (not only registrations). Keeping default 5 min/max threads is a good choice as well.

These are some of our measurements that lead us to these recommendations:

	4 workers, 4 threads	4 workers, 8 threads	4 workers, 16 threads	4 workers, 32 threads
Improvement	0%	14%	23%	10%

Use 4 – 6 workers on a default setup (4 CPUs) – we have seen about 25% higher performance with 5 workers when compared to 2 workers, but 8% lower performance with 8 workers when compared to 2 workers – see table below:

	2 workers, 16 threads	4 workers, 16 threads	6 workers, 16 threads	8 workers, 16 threads
Improvement	0%	26%	22%	-8%

Use 8 – 12 workers on a medium setup (8 CPUs) – see table below:

	2 workers, 16 threads	4 workers, 16 threads	8 workers, 16 threads	12 workers, 16 threads	16 workers, 16 threads
Improvement	0%	51%	52%	52%	42%

Use 16 – 24 workers on a 32 CPUs setup (this was tested on a 90 GiB RAM machine and memory turned out to be a factor here as system started swapping – proper *extra-large* should have 128 GiB), higher number of workers was problematic for higher registration concurrency levels we tested, so we cannot recommend it.

	4 workers, 16 threads	8 workers, 16 threads	16 workers, 16 threads	24 workers, 16 threads	32 workers, 16 threads	48 workers, 16 threads
Improvement	0%	37%	44%	52%	too many failures	too many failures

5.2.5. Configuring Puma workers

If you have enough CPUs, adding more workers adds more performance. For example, we have compared Satellite setups with 8 and 16 CPUs:

Table 5.1. satellite-installer options used to test effect of workers count

Satellite VM with 8 CPUs, 40 GiB RAM	Satellite VM with 16 CPUs, 40 GiB RAM
--foreman-foreman-service-puma-threads-min=16	**--foreman-foreman-service-puma-threads-min=16**
--foreman-foreman-service-puma-threads-max=16	**--foreman-foreman-service-puma-threads-max=16**

Satellite VM with 8 CPUs, 40 GiB RAM	Satellite VM with 16 CPUs, 40 GiB RAM
--foreman-foreman-service-puma-workers={2|4|8|16}	**--foreman-foreman-service-puma-workers={2|4|8|16}**

In 8 CPUs setup, changing the number of workers from 2 to 16, improved concurrent registration time by 36%. In 16 CPUs setup, the same change caused 55% improvement.

Adding more workers can also help with total registration concurrency Satellite can handle. In our measurements, setup with 2 workers were able to handle up to 480 concurrent registrations, but adding more workers improved the situation.

5.2.6. Configuring Puma threads

More threads allow for lower time to register hosts in parallel. For example, we have compared these two setups:

Satellite VM with 8 CPUs, 40 GiB RAM	Satellite VM with 8 CPUs, 40 GiB RAM
--foreman-foreman-service-puma-threads-min=16	**--foreman-foreman-service-puma-threads-min=8**
--foreman-foreman-service-puma-threads-max=16	**--foreman-foreman-service-puma-threads-max=8**
--foreman-foreman-service-puma-workers=2	**--foreman-foreman-service-puma-workers=4**

Using more workers and the same total number of threads results in about 11% of speedup in highly concurrent registrations scenario. Moreover, adding more workers did not consume more CPU and RAM but gets more performance.

5.2.7. Configuring Puma DB pool

The effective value of **$db_pool** is automatically set to equal **$foreman::foreman_service_puma_threads_max**. It is the maximum of **$foreman::db_pool** and **$foreman::foreman_service_puma_threads_max** but both have default value 5, so any increase to the max threads above 5 automatically increases the database connection pool by the same amount.

If you encounter **ActiveRecord::ConnectionTimeoutError: could not obtain a connection from the pool within 5.000 seconds (waited 5.006 seconds); all pooled connections were in use** error in **/var/log/foreman/production.log**, you might want to increase this value.

View current db_pool setting

```
# grep pool /etc/foreman/database.yml
  pool: 5
```

5.2.8. Manually tuning db_pool

If you decide not to rely on automatically configured value, you can apply custom number like this:

```
# satellite-installer --foreman-db-pool 10
```

Apply your changes to Satellite Server. For more information, see Section 5.1, “Applying configurations”.

5.3. 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 your Satellite.

5.3.1. Configuring the open files limit for Apache HTTPD

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:

Procedure

1. Set the maximum open files limit in **/etc/systemd/system/httpd.service.d/limits.conf**:

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

2. Apply your changes to Satellite Server. For more information, see Section 5.1, “Applying configurations”.

5.3.2. Tuning Apache httpd child processes

By default, httpd uses event request handling mechanism. When the number of requests to httpd exceeds the maximum number of child processes that can be launched to handle the incoming connections, httpd raises an HTTP 503 Service Unavailable error. Amidst httpd running out of processes to handle, the incoming connections can also result in multiple component failures on your Satellite services side due to the dependency of some components on the availability of httpd processes.

You can adapt the configuration of httpd event to handle more concurrent requests based on your expected peak load.

WARNING

Configuring these numbers in **custom-hiera.yaml** locks them. If you change these numbers using **satellite-installer --tuning=*My_Tuning_Option***, your **custom-hiera.yaml** will overwrite this setting. Set your numbers only if you have specific need for it.

Procedure

1. Modify the number of concurrent requests in **/etc/foreman-installer/custom-hiera.yaml** by changing or adding the following lines:

```
apache::mod::event::serverlimit: 64
apache::mod::event::maxrequestworkers: 1024
apache::mod::event::maxrequestsperchild: 4000
```

The example is identical to running **satellite-installer --tuning=medium** or higher on Satellite Server.

2. Apply your changes to Satellite Server. For more information, see Section 5.1, "Applying configurations".

5.4. DYNFLOW TUNING

Dynflow is the workflow management system and task orchestrator which is a Satellite plugin 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, Dynflow can take some help from an added tuning specifying how many executors can it launch.

For more information about the tunings involved related to Dynflow, see **https://satellite.example.com/foreman_tasks/sidekiq**.

Increase number of Sidekiq workers

Satellite contains a Dynflow service called **dynflow-sidekiq** that performs tasks scheduled by Dynflow. Sidekiq workers can be grouped into various queues to ensure lots of tasks of one type will not block execution of tasks of other type.

Red Hat recommends to increase the number of sidekiq workers to scale the Foreman tasking system for bulk concurrent tasks, for example for multiple content view publications and promotions, content synchronizations, and synchronizations to Capsule Servers. There are two options available:

- You can increase the number of threads used by a worker (worker's concurrency). This has limited impact for values larger than five due to Ruby implementation of the concurrency of threads.
- You can increase the number of workers, which is recommended.

Procedure

1. Increase the number of workers from one worker to three while remaining five threads/concurrency of each:

```
# satellite-installer --foreman-dynflow-worker-instances 3    # optionally, add --foreman-dynflow-worker-concurrency 5
```

2. Optional: Check if there are three worker services:

```
# systemctl -a | grep dynflow-sidekiq@worker-[0-9]
dynflow-sidekiq@worker-1.service       loaded    active   running   Foreman jobs daemon - worker-1 on sidekiq
dynflow-sidekiq@worker-2.service       loaded    active   running   Foreman jobs daemon - worker-2 on sidekiq
dynflow-sidekiq@worker-3.service       loaded    active   running   Foreman jobs daemon - worker-3 on sidekiq
```

For more information, see How to add sidekiq workers in Satellite6? .

5.5. PULL-BASED REX TRANSPORT TUNING

Satellite has a pull-based transport mode for remote execution. This transport mode uses MQTT as its messaging protocol and includes an MQTT client running on each host. For more information, see Transport Modes for Remote Execution in *Managing hosts*.

5.5.1. Increasing host limit for pull-based REX transport

You can tune the **mosquitto** MQTT server and increase the number of hosts connected to it.

Procedure

1. Enable pull-based remote execution on your Satellite Server or Capsule Server:

```
# satellite-installer --foreman-proxy-plugin-remote-execution-script-mode pull-mqtt
```

   Note that your Satellite Server or Capsule Server can only use one transport mode, either SSH or MQTT.

2. Create a config file to increase the default number of hosts accepted by the MQTT service:

```
cat >/etc/systemd/system/mosquitto.service.d/limits.conf <<EOF
[Service]
LimitNOFILE=5000
EOF
```

   This example sets the limit to allow the **mosquitto** service to handle 5000 hosts.

3. Run the following commands to apply your changes:

```
# systemctl daemon-reload
# systemctl restart mosquitto.service
```

5.5.2. Decreasing performance impact of the pull-based REX transport

When Satellite Server is configured with the pull-based transport mode for remote execution jobs using the Script provider, Capsule Server sends notifications about new jobs to clients through MQTT. This notification does not include the actual workload that the client is supposed to execute. After a client receives a notification about a new remote execution job, it queries Capsule Server for its actual workload. During the job, the client periodically sends outputs of the job to Capsule Server, further increasing the number of requests to Capsule Server.

These requests to Capsule Server together with high concurrency allowed by the MQTT protocol can cause exhaustion of available connections on Capsule Server. Some requests might fail, making some child tasks of remote execution jobs unresponsive. This also depends on actual job workload, as some jobs are causing additional load to Satellite Server, making it compete for resources if clients are registered to Satellite Server.

To avoid this, configure your Satellite Server and Capsule Server with the following parameters:

- MQTT Time To Live – Time interval in seconds given to the host to pick up the job before considering the job undelivered

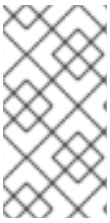
- MQTT Resend Interval – Time interval in seconds at which the notification should be re-sent to the host until the job is picked up or cancelled
- MQTT Rate Limit – Number of jobs that are allowed to run at the same time. You can limit the concurrency of remote execution by tuning the rate limit which means you are going to put more load on Satellite.

Procedure

- Tune the MQTT parameters on your Satellite Server:

```
# satellite-installer \
--foreman-proxy-plugin-remote-execution-script-mqtt-rate-limit My_MQTT_Rate_Limit \
--foreman-proxy-plugin-remote-execution-script-mqtt-resend-interval My_MQTT_Resend_Interval \
--foreman-proxy-plugin-remote-execution-script-mqtt-ttl My_MQTT_Time_To_Live
```

Capsule Server logs are in **/var/log/foreman-proxy/proxy.log**. Capsule Server uses Webrick HTTP server (no httpd or Puma involved), so there is no simple way to increase its capacity.



NOTE

Depending on the workload, number of hosts, available resources, and applied tuning, you might hit the Bug 2244811, which causes Capsule to consume too much memory and eventually be killed, making the rest of the job fail. At the moment there is no universally applicable workaround.

5.6. 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 PostgreSQL authors recommend disabling Transparent Hugepage on servers running PostgreSQL. For more information, see Section 4.3, “Disable Transparent Hugepage”.

You can apply a set of tunings to PostgreSQL to improve its response times, which will modify the **postgresql.conf** file.

Procedure

1. Append **/etc/foreman-installer/custom-hiera.yaml** to tune PostgreSQL:

```
postgresql::server::config_entries:
  max_connections: 1000
  shared_buffers: 2GB
  work_mem: 8MB
  autovacuum_vacuum_cost_limit: 2000
```

   You can use this to effectively tune down your Satellite instance irrespective of a tuning profile.

2. Apply your changes to Satellite Server. For more information, see Section 5.1, “Applying configurations”.

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.
- **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 GiB 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.
- **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. Red Hat recommends setting the value to **2000** as it is for the *medium*, *large*, *extra-large*, and *extra-extra-large* profiles, based on the general load that Satellite pushes on the PostgreSQL server process.

For more information, see BZ1867311: Upgrade fails when checkpoint_segments postgres parameter configured.

5.6.1. Benchmarking raw DB performance

To get a list of the top table sizes in disk space for both Candlepin, Foreman, and Pulp 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 100 GiB or what does benchmark take to run) might be used to measure PostgreSQL performance on your system. Use **dnf install postgresql-contrib** to install it. For more information, see github.com/RedHatSatellite/satellite-support.

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

WARNING

- 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. For example, if the DB is only 20 GiB and the buffer pool is 32 GiB, it won't show problems with large number of connections because the data will be completely buffered.

5.7. REDIS TUNING

Redis is an in-memory data store. It is used by multiple services in Satellite. The Dynflow and Pulp tasking systems use it to track their tasks. Given the way Satellite uses Redis, its memory consumption should be stable.

The Redis authors recommend disabling Transparent Hugepage on servers running Redis. For more about it please see Section 4.3, "Disable Transparent Hugepage".

5.8. CAPSULE CONFIGURATION TUNING

Capsules are meant to offload part of Satellite load and provide access to different networks related to distributing content to clients but they can also be used to execute remote execution jobs. What they cannot help with is anything which extensively uses Satellite API as host registration or package profile update.

5.8.1. Capsule performance tests

We have measured multiple test cases on multiple Capsule configurations:

Capsule HW configuration	CPUs	RAM
minimal	4	12 GiB
large	8	24 GiB
extra large	16	46 GiB

Content delivery use case

In a download test where we concurrently downloaded a 40MB repo of 2000 packages on 100, 200, .. 1000 hosts, we saw roughly 50% improvement in average download duration every time when we doubled Capsule Server resources. For more precise numbers, see the table below.

Concurrent downloading hosts	Minimal (4 CPU and 12 GiB RAM) → Large (8 CPU and 24 GiB RAM)	Large (8 CPU and 24 GiB RAM) → Extra Large (16 CPU and 46 GiB RAM)	Minimal (4 CPU and 12 GiB RAM) → Extra Large (16 CPU and 46 GiB RAM)
Average Improvement	~ 50% (e.g. for 700 concurrent downloads in average 9 seconds vs. 4.4 seconds per package)	~ 40% (e.g. for 700 concurrent downloads in average 4.4 seconds vs. 2.5 seconds per package)	~ 70% (e.g. for 700 concurrent downloads in average 9 seconds vs. 2.5 seconds per package)

When we compared download performance from Satellite Server vs. from Capsule Server, we have seen only about 5% speedup, but that is expected as Capsule Server's main benefit is in getting content closer to geographically distributed clients (or clients in different networks) and in handling part of the load Satellite Server would have to handle itself. In some smaller hardware configurations (8 CPUs and 24 GiB), Satellite Server was not able to handle downloads from more than 500 concurrent clients, while a Capsule Server with the same hardware configuration was able to service more than 1000 and possibly even more.

Concurrent registrations use case

For concurrent registrations, a bottleneck is usually CPU speed, but all configs were able to handle even high concurrency without swapping. Hardware resources used for Capsule have only minimal impact on registration performance. For example, Capsule Server with 16 CPUs and 46 GiB RAM have at most a 9% registration speed improvement when compared to a Capsule Server with 4 CPUs and 12 GiB RAM. During periods of very high concurrency, you might experience timeouts in the Capsule Server to Satellite Server communication. You can alleviate this by increasing the default timeout using the following tunable in **/etc/foreman-installer/custom-hiera.yaml**:

```
apache::mod::proxy::proxy_timeout: 600
```

Remote execution use case

We have tested executing Remote Execution jobs via both SSH and Ansible backend on 500, 2000 and 4000 hosts. All configurations were able to handle all of the tests without errors, except for the smallest configuration (4 CPUs and 12 GiB memory) which failed to finish on all 4000 hosts.

Content sync use case

In a sync test where we synced Red Hat Enterprise Linux 6, 7, 8 BaseOS and 8 AppStream we have not seen significant differences among Capsule configurations. This will be different for syncing a higher number of content views in parallel.