



Red Hat Satellite 6.3

Red Hat Satellite のチューニング

Red Hat Satellite 6 のパフォーマンスチューニング

Red Hat Satellite 6.3 Red Hat Satellite のチューニング

Red Hat Satellite 6 のパフォーマンスチューニング

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

法律上の通知

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution-Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

Red Hat Satellite 6 環境のチューニング方法

目次

第1章 はじめに	4
第2章 システム要件	5
第3章 パフォーマンスに関する最重要の考慮事項	6
第4章 パフォーマンス改善を目的とした環境の設定	7
4.1. CPU	7
4.2. メモリー	7
4.3. ディスク	7
4.4. ネットワーク	7
4.5. サーバーの電源管理	7
第5章 チューニング	8
5.1. APACHE HTTP SERVER で SATELLITE パフォーマンスの設定	8
5.1.1. 概要	8
5.1.2. KeepAlive 設定のチューニング	8
5.1.3. オープンファイル数の上限	9
5.2. PASSENGER の設定	9
5.2.1. 概要	9
5.2.2. Passenger で Foreman 関連のワークロードのチューニング	10
5.2.3. Passenger で Puppet 3 関連のワークロードのチューニング	10
5.2.4. Passenger で Puppet 4 関連のワークロードのチューニング	10
5.2.5. "passenger-status" コマンド	11
5.3. CANDLEPIN	11
5.4. PULP	11
5.4.1. コンテンツの格納	12
5.4.2. ワーカーの同時実行	12
5.5. MONGODB	12
5.5.1. Transparent Huge Pages (THP) の無効	12
5.6. NFS	12
5.6.1. /var/lib/pulp	12
5.6.2. /var/lib/mongodb	12
5.7. FOREMAN	13
5.7.1. Foreman で Puppet 3 に関するワークロードのチューニング	13
5.8. DYNFLOW	13
5.8.1. dynFlow のチューニング例	14
5.9. POSTGRESQL のチューニング	14
5.9.1. PostgreSQL のチューニング例	14
5.10. PUPPET	15
5.10.1. 実行間隔	15
5.10.2. Puppet に関するその他の対話	15
5.11. 外部 CAPSULE	15
5.11.1. 外部 Capsule の利点	15
5.11.2. 外部 Capsule を使用するかどうかを決定する際に検討が必要な要因	16
5.11.3. ハードウェアに関する考慮事項と、httpd および Passenger の設定	16
5.11.4. Capsule の httpd 設定のチューニング例	16
5.11.5. Capsule Passenger 設定例:	16
5.11.6. Capsule Puppet Passenger 設定のチューニング例:	16
5.12. クライアントエージェントのスケーリング (KATELLO-AGENT)	17
5.13. SCALE: HAMMER TIMEOUT	17
5.14. QPIDD および QDROUTERD の設定	17

5.14.1. オープンファイル数の上限	17
5.14.1.1. qrouterd 向けオープンファイル数の上限の計算	17
5.14.1.2. qpidd 向けオープンファイル数の上限の計算	17
5.14.1.3. 設定例	17
5.14.1.3.1. qdrouterd 設定	18
5.14.1.3.2. qpidd 設定	18
5.14.2. 非同期入出力 (AIO) 要求の最大数	18
5.14.3. ストレージの考慮事項	18
5.14.4. mgmt-pub-interval 設定	19

第1章 はじめに

本書は、Red Hat Satellite のパフォーマンスおよびスケーラビリティをチューニングするためのガイドラインを提示します。Red Hat Satellite に普遍的に適用できる設定はありません。ある状況でパフォーマンスを最大限に高められても、別の状況で同じ結果が得られるとは限りません。

Red Hat Satellite は、Red Hat 製品のデプロイメントの完全ライフサイクルをシステム管理者が管理できるように、システム管理を完備した製品です。Red Hat Satellite は、物理、仮想、プライベートクラウドに分散されているデプロイメントを管理できます。Red Hat Satellite は、システムのプロビジョニング、設定管理、ソフトウェア管理、およびサブスクリプション管理を提供し、高レベルのスケーラビリティとセキュリティを維持します。

Red Hat Satellite Server の詳細は、以下のドキュメントを参照してください。

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

第2章 システム要件

Red Hat Satellite のハードウェア要件およびソフトウェア要件の詳細は、『インストールガイド』の「[インストールのための環境準備](#)」を参照してください。

第3章 パフォーマンスに関する最重要の考慮事項

以下の項目は、Red Hat Satellite インストールのパフォーマンスを向上させるためにできることです。

1. `httpd` を設定する
2. 同時実行を増やすように `Passenger` を設定する
3. `Candlepin` を設定する
4. `Pulp` を設定する
5. `Foreman` のパフォーマンスおよびスケーラビリティを設定する
6. `Puppet` のスケーラビリティを設定する
7. 統合 `Capsule` の代わりに外部 `Capsule` をデプロイする
8. `katello-agent` のスケーラビリティを設定する
9. API タイムアウトを減らすように `Hammer` を設定する
10. `qpidd` および `qdrouterd` を設定する
11. ホストの同時実行登録を増やすように `PostgreSQL` のチューニングを向上する
12. ストレージの DB ワークロードを設定する
13. `MongoDB` を使用したストレージのニーズとネットワークの互換性を考慮する
14. コンテンツビューのストレージ要件を満たしていることを確認する
15. システム要件を満たしていることを確認する
16. 環境のリモート実行を向上する

第4章 パフォーマンス改善を目的とした環境の設定

4.1. CPU

Satellite 6.3 で利用できる物理 CPU コアが多くなればなるほど、タスクのスループットはよくなります。ニーズを満たすようにスループットとタスクのレイテンシーのバランスを取ることが重要になります。CPU コアを増やせば Red Hat Satellite 6 のデプロイメントのスケールが向上するため、CPU コアを増やすことが、CPU ハードウェアを選択する際に最初に検討すべき項目となります。

4.2. メモリー

Red Hat Satellite インストールに適切なメモリーを提供します。必要なメモリー量は、Apache、Foreman、MongoDB、PostgreSQL、Pulp、Puppet、Tomcat、Qpid、およびファイルシステムキャッシュのプロセスにメモリーを提供する必要があることを考慮して決定してください。パフォーマンスが高いシステムでは、1 台のサーバーで Apache、Foreman、Puppet などのソフトウェアのスケールが最大になっても、スワップは行われません。

4.3. ディスク

使用するディスクの種類を決定する際は、1 秒あたりの I/O 処理回数 (IOPS: Input/Output Operations Per Second) とディスク容量を考慮する必要があります。もっとも頻繁にアクセスするディレクトリーの容量と IOPS を向上する必要がある場合は、ファイルシステムのパーティションを複数のディスクにわたって作成する必要があります。IOPS と監視に重要なディレクトリーは、`/var/lib/pulp`、`/var/lib/pgsql`、および `/var/lib/mongodb` です。

4.4. ネットワーク

テスト中に、CPU または設定の制限に到達しなければ、ネットワークハードウェアが一貫してボトルネックにならないことが確認されています。これは、(1) Red Hat Satellite capsule サーバーおよび (2) エミュレートしたコンテンツ配信ネットワーク (CDN) に接続した 10Gb ネットワークで実現しています。CDN へのインターネット接続はおそらくボトルネックになりますが、インターネットへの接続は本書の範囲外になります。

4.5. サーバーの電源管理

お使いのサーバーは、おそらくデフォルトで節電するように設定されています。これによりパフォーマンスが低下することがしばしばあります。サーバーの BIOS で、OS がホストの電源制御ができるように設定し、Red Hat Enterprise Linux によるサーバーの電源消費制御を可能にし、Red Hat Satellite のパフォーマンスを向上させます。電源消費とパフォーマンスは、一方を向上すると他方が犠牲になります。状況に応じて最適なバランスを提供できるように、Red Hat Satellite 環境をチューニングします。

第5章 チューニング

5.1. APACHE HTTP SERVER で SATELLITE パフォーマンスの設定

5.1.1. 概要

Apache HTTP Server は、Satellite のコアコンポーネントです。Satellite のコアコンポーネントである Passenger および Pulp は、Apache HTTP Server を利用して受信要求を処理します。Web UI または Satellite API から届く要求は、Apache HTTP Server サーバーが受け、その要求を処理する Satellite のコンポーネントに転送されます。

Red Hat Satellite 6.3 に同梱される Apache HTTP Server のバージョンは、プロセススペースの要求処理モデルを使用します。新しい受信要求が発生するたびに、Apache HTTP Server が新しいプロセスを起動します。各プロセスはメモリー領域を使用します。Apache HTTP Server が起動できるプロセスの最大数は、通常、Apache HTTP Server の設定で制御します。この数値を変更して、Apache HTTP Server が処理できる要求数を増やすことができます。

一度に要求が急激に増えると、Apache HTTP Server が要求を処理できる能力を超えるため、HTTP 503 応答を受ける新しい受信要求が発生します。

要求が急増したときに、Apache HTTP Server が受信要求にどれだけ早く応答できるように制御する設定が、`/etc/httpd/conf.modules.d/prefork.conf` に 2 つあります。

```
ServerLimit 512
StartServers 10
```

ServerLimit パラメーターは、Apache HTTP Server が新規受信要求を処理するために起動する子プロセスの最大数を定義します。この設定を増やすと、Satellite サーバーと Capsule サーバーに到達する要求数が急増するたびに、Apache HTTP Server が使用するメモリー使用量が増えます。

StartServer パラメーターは、Apache HTTP Server が起動する際に、このサーバーが起動する子プロセスの数を定義します。この数を増やすと、新規受信要求に対する応答時間が向上します。この数値を増やすと、要求に応答するのに新規子プロセスが起動するのを待つ必要がなくなります。

5.1.2. KeepAlive 設定のチューニング

Apache HTTP Server は、**KeepAlive** ディレクティブを使用して TCP 接続を管理します。**KeepAlive** を **On** に設定すると、2 つの変数が、TCP 接続の使用方法を制御します。

KeepAliveTimeout は、接続を開いたままにする時間を指定します。再接続を行うとリソースが消費されるため、頻繁に使用する接続は開いたままにするとパフォーマンスが向上します。一方、接続を必要以上に開いたままにするとリソースを過剰に使用するため、パフォーマンスが低下する可能性があります。

Apache HTTP Server の **KeepAlive** パラメーターをオンにし、**KeepAliveTimeout** および **MakeKeepAliveRequests** に値を設定して、TCP 接続数と CPU 使用量を減らします。Red Hat は、Red Hat Satellite と、Capsules またはホストとの間にレイテンシーがある場合を除いて、**KeepAliveTimeout** の値を 2 ~ 5 秒の間に設定することを推奨します。また、**MaxKeepAliveRequests** は 0 に設定し、各接続が同じ TCP 接続ですべてのコンテンツを要求できるようにします。**KeepAlive** をオンにした場合、Web サーバーのページをロードするのにかかる時間は短くなります。Red Hat Satellite における **KeepAlive** のデフォルト設定は `/etc/httpd/conf.d/05-foreman-ssl.d/katello.conf` で確認できます。

Red Hat Satellite 6 Apache HTTP Server 設定のチューニング例:

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

5.1.3. オープンファイル数の上限

登録を多数行う場合、または **Capsule**、コンテンツホスト、およびコンテンツビューのスケールを増やす場合は、オープンファイルの最大数を増やします。

設定ファイル **/etc/systemd/system/httpd.service.d/limits.conf** を編集し、**LimitNOFILE** の値を設定します。



注記

パスおよび設定ファイルがない場合は作成します。

以下の例では、オープンファイルの最大数を **640000** に設定します。

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

以下のコマンドを実行すると、オープンファイルの最大数を確認できます。

```
# 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 の設定

5.2.1. 概要

Passenger は Web サーバーで、Red Hat Satellite のコアコンポーネントです。Satellite は Passenger を使用して、Foreman、Puppet 3 などの Ruby アプリケーションを実行します。Passenger は、Apache HTTP Server と統合され、受信要求を受け取り、その要求を処理する各コンポーネントにリダイレクトします。

Passenger は、GUI にアクセスする際、API にアクセスする際、コンテンツホストを登録する際に、Satellite に関与します。Passenger が処理する各要求は、Apache HTTP Server のプロセスを消費します。Passenger は、各キューを、アプリケーション固有の待機キューに入れます。Passenger がキューに入れることができる要求の最大数は、Passenger の設定で定義します。実行する規模が大きくなる場合は、Passenger が同時に処理できる要求の数を増やすことが望ましい場合があります。また、大量の要求を受け入れられるように、待機キューのサイズを増やしても望ましい場合があります。

Passenger は、Apache HTTP Server 設定ファイルに設定され、Foreman および Puppet のパフォーマンス、スケーリング、および挙動を制御するのに使用されます。

5.2.2. Passenger で Foreman 関連のワークロードのチューニング



注記

本セクションのチューニングは、`/etc/httpd/conf.d/passenger.conf` に適用されます。

大規模である必要がなく、Puppet をあまり使用しないワークロードで構成されるデプロイメントでは、Passenger のプールサイズを増やします。プールサイズは、Passenger が受信要求を処理するために起動できるアプリケーションプロセスの数を定義します。

プールのサイズを、システムの VCPU 数を 2 倍にした数に設定します。たとえば、システムに VCPU が 24 個ある場合は、**MaxPoolSize** を 48 に設定します。

MaxPoolSize を値を、システムの VCPU 数を 2 倍にした数よりも大きくすると、プロセスが CPU リソースに対して互いに競うようになるため、要求のタイムアウトが増える可能性があります。

5.2.3. Passenger で Puppet 3 関連のワークロードのチューニング



注記

本セクションのチューニングは、`/etc/httpd/conf.d/passenger.conf` に適用されます。

Puppet 3 に大きく依存するワークロードで構成されるデプロイメントでは、Passenger のプールサイズを増やします。プールのサイズは、Passenger が受信要求を処理するために起動できるアプリケーションプロセスの数を定義します。

プールのサイズを、システムの VCPU の数に設定します。たとえば、システムに VCPU が 24 個ある場合は、**MaxPoolSize** を 24 に設定します。

5.2.4. Passenger で Puppet 4 関連のワークロードのチューニング



注記

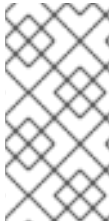
本セクションのチューニングは、`/etc/httpd/conf.d/passenger.conf` に適用されます。

Puppet 4 に大きく依存するワークロードで構成されるデプロイメントでは、Passenger のプールサイズを増やします。プールのサイズは、Passenger が受信要求を処理するために起動できるアプリケーションプロセスの数を定義します。

Puppet 4 では、Passenger がそのサーバープロセスを起動する必要はありません。したがって、Passenger 全体のプロセスプールは、Foreman プロセスの起動に特化しています。

プールのサイズを、システムの VCPU 数に 1.5 を乗算した数に設定します。たとえば、システムに VCPU が 24 個ある場合は、**MaxPoolSize** を 36 に設定します。

要求が急増するような、大規模なスケールアウト設定では、**PassengerMaxRequestQueueSize** を 400 に設定します。



注記

PassengerMaxRequestQueueSize の値は、常に **Apache HTTP Server** の **ServerLimit** 値よりも低くする必要があります。この値を **ServerLimit** のしきい値よりも大きくすると、その他のコンポーネントで、**http-process** プールが不足するため、要求が拒否される可能性が高くなり、動作に失敗します。

5.2.5. "passenger-status" コマンド

passenger-status コマンドで、**Passenger** が起動した **Foreman** プロセスおよび **Puppet** プロセスを取得して、**PassengerMaxPoolSize** を確認できます。

5.3. CANDLEPIN

Candlepin は、ソフトウェアサブスクリプションの管理を容易にするツールコレクションで、コンテンツおよびサブスクリプションに、統一されたワークフローと、**Web** ベースのユーザーインターフェースを提供する **Katello** の一部です。**Candlepin** は、サブスクリプションに関連する **Katello** のコンポーネントを提供します。

登録を完了するのに必要なレイテンシーは、サブスクリプションの複雑さによって決まります。設定に含まれる **Foreman** プロセスの数が多く、**Passenger** のキューのサイズが大きくなると、登録を完了するのに必要なレイテンシーは大きくなります。

サブスクリプションの登録にかかる時間を測定すれば、登録を完了するのに必要なレイテンシーを決定できます。以下のコマンドを実行して、サブスクリプションの登録にかかる時間を測定します。

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

時間測定が行われる環境の能力を知るために、サブスクリプションの登録を完了するまでの最小時間、平均時間、最大時間を決定します。**Red Hat Satellite 6.3** を使用したデフォルトの **Passenger** 設定では、**Foreman** が **PassengerMaxPoolSize** プロセスをすべて消費し、すべてのアプリケーションプロセスをプリロードした場合は、同時に 6 個の登録が可能になります。起動したプロセスが 1 つしかない場合は、さらなるプリロードのレイテンシーが登録時間に追加されます。追加の同時登録により、利用可能な **Foreman** プロセスのキューが原因で、余計なレイテンシーが発生します。**Foreman** に関するその他のタスクまたはワークロードがキューに入り、その他の同時登録を遅らせます。

5.4. PULP

Pulp は、**Python** で記述されるソフトウェアリポジトリ管理ツールです。**Pulp** は、完全ソフトウェアリポジトリ管理、リポジトリをミラーリングする機能、リポジトリをホストする機能、多くのコンシューマーにこのリポジトリのコンテンツを配信する機能を提供します。

Pulp は、**Satellite** の **RPM** コンテンツ、**Puppet** モジュール、およびコンテナイメージを管理します。また、**Pulp** はコンテンツビューを公開し、**Capsule** およびホストがコンテンツを取得するローカルリポジトリを作成します。**Apache HTTP Server** の設定は、**Pulp** の **REST API** 要求がどのように効率的に処理されるかを決定します。

Pulp は、**celery** を利用して、アップストリームリポジトリからデータをダウンロードする **Pulp** ワーカーを起動します。**Pulp** は、**Apache HTTP Server** を利用して、**Pulp** の **API** や内部コンポーネントへのアクセスを提供します。

Satellite 環境で、多数のソフトウェアリポジトリを同時に同期する必要がある場合は、Pulp が起動できるワーカーの数を増やします。

Pulp は Katello のコンポーネントです。

5.4.1. コンテンツの格納

Pulp ディレクトリーを、簡単にスケールできる大きなローカルパーティションにマウントします。論理ボリュームマネージャー (LVM) を使用してこのパーティションを作成します。

5.4.2. ワーカーの同時実行

並行して実行するタスクの数を調整するには、`/etc/default/pulp_workers` ファイルの **PULP_CONCURRENCY** の値を変更します。Pulp はより多くのリポジトリを同時に同期し、さらに多くのワーカーが Satellite 6.3 リソースを消費できます。ただし、このように設定すると、その他の Satellite コンポーネントが足りなくなる可能性があります。Puppet などの同時実行ワークロードが含まれる環境で、Pulp の同時実行レベルを実現することが重要です。デフォルトでは、CPU の数が 8 より少ないシステムでは、**PULP_CONCURRENCY** を CPU の数に設定します。CPU の数が 8 より多い場合は、8 に設定します。

5.5. MONGODB

MongoDB は、Pulp によって使用される NoSQL データベースサーバーで、同期したりポジトリとそのコンテンツに関するメタデータを格納します。また、Pulp は MongoDB を使用して、Pulp のタスクと現在のステータスに関する情報を格納します。

5.5.1. Transparent Huge Pages (THP) の無効

Transparent Huge Page (THP) は、Linux カーネルが使用するメモリー管理技術で、より大きいメモリーページを使用して、トランスレーションルックアサイドバッファ (TLB: Translation Lookaside Buffer) を使用する際のオーバーヘッドを減らします。データベースで連続するメモリーアクセスパターンの代わりにスパースメモリーアクセスパターンを使用しているため、THP が有効な場合にデータベースのワークロードのパフォーマンスがしばしば低下します。

MongoDB のパフォーマンスを向上するために、Red Hat は THP を無効にすることを推奨します。THP を無効にする方法は「[Red Hat Enterprise Linux 7 で transparent hugepage \(THP\) を無効にする](#)」を参照してください。

5.6. NFS

5.6.1. /var/lib/pulp

Red Hat Satellite 6.3 は `/var/lib/pulp` を使用してリポジトリのコンテンツを格納し、管理します。Red Hat は、Pulp を NFS で実行することは推奨しません。代わりに、`/var/lib/pulp` ファイルシステムに、高帯域幅で低レイテンシーのストレージを使用することをお勧めします。Red Hat Satellite には、IO を大量に使用する操作が多数あるため、高レイテンシーで低帯域幅のストレージを使用すると、Satellite 6.3 のパフォーマンスが低下することがあります。

5.6.2. /var/lib/mongodb

Pulp は MongoDB を使用します。Red Hat は、MongoDB と NFS を一緒に使用しないことを推奨します。

5.7. FOREMAN

Foreman は、Passenger が起動する Ruby アプリケーションで多くのことを行いますが、たとえば UI やリモート実行を提供し、コンテンツホストで Foreman SCAP スキャンを実行します。Foreman は、コンテンツホストの登録にも関係します。

Foreman は Passenger 内から実行し、新たに受信要求が到着したら、動的に新規 Foreman プロセスを起動します。

1つの Foreman プロセスが、Passenger によって再利用される前に多くの要求に対応し、消費したメモリーを解放します。Passenger が起動できる Foreman プロセスの最大数は、Passenger の `MaxPoolSize` パラメーター (`/etc/httpd/conf.d/passenger.conf` 設定ファイル) によって制御されます。

Foreman は Passenger アプリケーションサーバー内で実行します。Foreman のパフォーマンスおよびスケーラビリティは、httpd および Passenger の設定の直接の影響を受けます。Foreman は、UI および API の要求も処理します。Apache HTTP Server の `KeepAlive` 設定をオンにすると、ユーザーインターフェースページのロードにかかる時間が短くなります。`tuned` プロファイルを適切に設定すると、CLI および API の応答時間が向上します。

Foreman では、プロセス数を増やし、1つのプロセスが再利用される前に処理できる要求数を設定できます。

5.7.1. Foreman で Puppet 3 に関するワークロードのチューニング



注記

設定ファイル `/etc/httpd/conf.d/05-foreman-ssl.conf` を編集します。

`PassengerMinInstances` 値は、Foreman でアクティブなインスタンス数の下限を設定できます。残りのスロットは Puppet を起動するために使用されます。

Puppet 3 に関するワークロード

`PassengerMinInstances` を 6 に設定します。

Puppet 3 には関係ないワークロード

`PassengerMinInstances` を 10 に設定します。

Puppet 3 の使用を最小限にとどめるワークロード

`PassengerMinInstances` を 10 に設定します。

5.8. DYNFLOW



注記

このチューニングは `/etc/sysconfig/foreman-tasks` に適用されます。

DynFlow は、Ruby で記述されたワークフローシステムおよびタスクのオーケストレーションエンジンで、Foreman へのプラグインとして実行します。Foreman は DynFlow を使用して、キューに登録されたタスクのスケジュール、計画、および実行を行います。

dynFlow をチューニングすると、**Ruby** のメモリ使用量が制御できるようになります。状況によっては、実行する規模が大きくなると、**dynFlow** のメモリ使用量は **10 GB** を超えます。

DynFlow で、ワーカーが使用するメモリ量を制限し、要求を処理するワーカー数を増やすように設定できます。

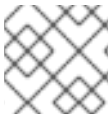
Red Hat Satellite 6.3 で、**dynFlow** エグゼキューターが使用するメモリの量を制限できます。

EXECUTOR_MEMORY_LIMIT は、**dynFlow** エグゼキューターが再利用される前に、そのエグゼキューターの 1 プロセスが消費できるメモリ量を定義します。

EXECUTOR_MEMORY_MONITOR_DELAY は、エグゼキューターの初期化後に、エグゼキューターのメモリを確認する最初のポーリングをいつ行うかを定義します。

EXECUTOR_MEMORY_MONITOR_INTERVAL は、エグゼキューターのメモリ使用量をポーリングする頻度を定義します。

5.8.1. dynFlow のチューニング例



注記

このチューニングは **/etc/sysconfig/foreman-tasks** に適用されます。

多くの操作に関する設定をスケーリングした場合は、以下のように、エグゼキューターメモリの値を設定します。

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

5.9. POSTGRESQL のチューニング



注記

本セクションのチューニングは、**/var/lib/pgsql/data/postgresql.conf** に適用されます。

PostgreSQL は **Foreman** および **Candlepin** によって使用され、登録したコンテンツホスト、サブスクリプション、ジョブ、およびタスクに関するレコードを格納します。時間が経つにつれ、**PostgreSQL** のデータがたまるため、新規にインストールした状態と比較して、クエリーの速度が遅くなります。

IPostgreSQL のアクセス先にメモリを増やすと、操作実行が加速します。

同時発生を増やす場合は、**PostgreSQL** が同時に管理できる接続数を増やします。

5.9.1. PostgreSQL のチューニング例



注記

本セクションのチューニングは、**/var/lib/pgsql/data/postgresql.conf** に適用されます。

`/var/lib/pgsql/data/postgresql.conf`に以下のチューニングを行うとパフォーマンスが向上しますが、メモリーの使用量も増えます。

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

5.10. PUPPET

Red Hat Satellite 6.3 は、Puppet バージョン 3.8 以降、および Puppet 4 を使用するホストをサポートします。Puppet 3 は Ruby アプリケーションで、Passenger アプリケーションサーバー内で実行します。Puppet 4 は、スタンドアロンで、Java ベースのアプリケーションとして実行します。Puppet の複数の要因が、Red Hat Satellite 全体のパフォーマンスとスケーラビリティに影響を及ぼします。

5.10.1. 実行間隔

全期間を通じてロードを分散しない非決定論的な実行間隔により、Puppet 環境ではスケーリング問題およびエラーが発生します。負荷を均一に分散すると、システムが確実にスケールできるようになり、より多くの要求が処理できるようになり、急増する回数が少なくなります。実行間隔は、以下の方法で分散できます。

1. **Puppet スプレー** - 各 Puppet クライアントに対してスプレーをオンにします。これにより、実行間隔にランダム化が追加され、決定論的な実行間隔は確立されなくなります。
2. **cron ジョブ** - 各 Puppet エージェントを、デーモンではなく cron ジョブとして実行します。これにより、実行間隔が決定論的になります。規模が大きくなると、クライアントの追加時または削除時に管理するのが難しくなります。
3. **分離** - Puppet エージェント実行が発生したときに管理する別のエンティティをデプロイします。

5.10.2. Puppet に関するその他の対話

お使いの環境で実行するその他の Puppet の対話を測定します。たとえば、ファクトの送信、ファイルの処理、レポートの送信などの対話により、Red Hat Satellite Server とその Capsule に負荷がかかります。各対話により、システムにコストがかかります。

5.11. 外部 CAPSULE

外部 Capsule を使用すると、Satellite デプロイメントをスケールアウトできるようになり、外部 Capsules が管理するホストにローカルのサービスを提供します。

5.11.1. 外部 Capsule の利点

1. Red Hat Satellite における HTTP 要求の数が減ります。
2. Puppet および Foreman に提供する CPU およびメモリーリソースが増えます。
3. リソースを配置する場所をエンドクライアントに近づけて、要求のレイテンシーを短くします。

5.11.2. 外部 Capsule を使用するかどうかを決定する際に検討が必要な要因

1. 実行間隔: Puppet エージェントを適用するタイミング。これにより、ワークロードがその期間内で均一に広がります。
2. クライアントのワークロード: Puppet エージェントの実行時に、各 Puppet クライアントのワーク量
3. ハードウェア設定: Puppet で利用できるリソースの量

外部 Capsule と統合 Capsule のどちらを使用するかは、ハードウェア、設定、およびワークロードによって異なります。多くの Puppet ワークロード変数は、Red Hat Satellite のスケーラビリティに直接影響するため、Puppet 要件に対して計画します。実行間隔を長くすると、能力が高くなりますが、Puppet が設定を適用する間隔が長くなります。実行間隔を短くすると能力が低くなります。クライアントが均一に広がっていないと、大規模なクライアントグループが Passenger のキューを満たし、その他の要求をブロックします。一方、Red Hat Satellite Server が十分に活用されなく期間も発生します。Puppet の実行を完了するために実行する必要がある各 Puppet クライアントのワーク量は、スケーラビリティを変更します。物理ハードウェアリソースが利用できる場合は、設定した Puppet プロセス数を増やすと、スケーラビリティが向上します。この変数の在り方は、外部 Capsule に移行するタイミングを計るために、普遍的に適用できる推奨事項を提供するものではありません。そのスケーラビリティを決定するための Puppet ワークロードを、ベンチマークに従って評価します。

5.11.3. ハードウェアに関する考慮事項と、httpd および Passenger の設定

本書に記載したハードウェアの考慮事項を Capsules に適用します。ホストのリソースをオーバーコミットする仮想マシンをホストするマシンと同じ場所に Capsule を設定している場合を除いて、仮想化 Capsule で、vCPU の数と利用可能なメモリーをチューニングできます。Apache HTTP Server および Passenger の設定に関する考慮事項は、Puppet 関連の Capsule にも適用できます。

5.11.4. Capsule の httpd 設定のチューニング例

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

5.11.5. Capsule Passenger 設定例:

/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. Capsule Puppet Passenger 設定のチューニング例:

/etc/httpd/conf.d/25-puppet.conf:

```
PassengerMinInstances 2
```

```

PassengerStartTimeout 90
PassengerMaxPreloaderIdleTime 0
PassengerMaxRequests 10000
PassengerPreStart https://example-capsule.com:8140

```

5.12. クライアントエージェントのスケーリング (KATELLO-AGENT)

クライアントが応答するまでに必要な時間を 20 秒より長くする必要がある場合は、**管理** → **設定** → **Katello** の GUI で、**content_action_accept_timeout** をデフォルトの 20 秒より大きくします (Red Hat ナレッジベースソリューション「[Errata apply fails with "RuntimeError: Host did not respond within 20 seconds. Is katello-agent installed and running on the goferd Host?"](#)」を参照)。

5.13. SCALE: HAMMER TIMEOUT

Capsule、コンテンツホスト、またはコンテンツビューのスケーリング時に、hammer API 要求がタイムアウトになる場合があります。`/etc/hammer/cli.modules.d/foreman.yml` に以下の行を追加すると、タイムアウトが無効になります。

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

5.14. QPIDD および QDROUTERD の設定

5.14.1. オープンファイル数の上限

Satellite および Capsule をスケールアップして、対応するホストの数を増やすには、プロセスが、必要なだけ記述子を開くことができるようにする必要があります。

以下のチューニングは、Apache HTTP Server および `qrouterd` に適用されます。

```

[Service]
LimitNOFILE=640000

```

デプロイメントで **katello-agent** を使用する場合は、**qpidd** および **qrouterd** でファイルの制限をチューニングする必要があります。

5.14.1.1. qrouterd 向けオープンファイル数の上限の計算

`qrouterd` で数式 $(N \times 3) + 100$ を使用して、オープンファイルの制限を計算します。 N はコンテンツホストの数です。コンテンツホスト 1 つに対して、ルーターで使われるファイル記述子の数は最大 3 つ、そしてルーターを実行するのに必要なファイル記述子の数は最大 100 となります。

5.14.1.2. qpidd 向けオープンファイル数の上限の計算

`qpidd` で数式 $(N \times 4) + 500$ を使用して、オープンファイルの制限を計算します。 N はコンテンツホストの数です。コンテンツホスト 1 つに対して使われるファイル記述子の数は最大 4 つ、そして Broker (`qpidd` のコンポーネント) の操作に必要なファイル記述子の数は最大 500 となります。

5.14.1.3. 設定例

以下の設定では、Satellite のコンテンツホストを 10,000 まで増やすことができます。

5.14.1.3.1. qdrouterd 設定

設定ファイル `/etc/systemd/qdrouterd.service.d/limits.conf` を編集し、`LimitNOFILE` の値を **301000** に設定します。



注記

パスおよび設定ファイルがない場合は作成します。

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

5.14.1.3.2. qpidd 設定

設定ファイル `/etc/systemd/system/qpidd.service.d/limits.conf` を編集し、`LimitNOFILE` の値を **40500** に設定します。



注記

パスおよび設定ファイルがない場合は作成します。

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

詳細は、Red Hat ナレッジベースソリューション「[Satellite 6 Remote Agent \(katello-agent\) Scaling / Performance Tuning](#)」を参照してください。

5.14.2. 非同期入出力 (AIO) 要求の最大数

カーネルパラメーター `fs.aio-max-nr` を増やすと、同時に要求できる AIO の最大数を増やすことができます。

1. 設定ファイル `/etc/sysctl.conf` を編集し、`fs.aio-max-nr` の値を、希望する最大値に設定します。

```
fs.aio-max-nr=12345
```

この例では、同時に要求できる AIO の最大数は **12345** です。

この数値は、**Satellite** に登録する予定のコンテンツホストの最大数に **33** を乗算した数よりも大きくする必要があります。

1. 変更を適用します。

```
# sysctl -p
```

マシンを再起動すると、この変更が適用されます。

詳細は、Red Hat ナレッジベースソリューション「[Deleting a content host, pulp raises error "Value for replyText is too large"](#)」を参照してください。

5.14.3. ストレージの考慮事項

katello-agent を広範囲に使用するインストールを計画している場合は、前もって **/var/lib/qpidd** ディレクトリーに対して十分なストレージ容量を計画します。Red Hat Satellite 6.3 では、1つのコンテンツホストに対して **/var/lib/qpidd** のディスク容量が 2MB 必要になります ([Bugzilla 1366323](#) を参照)。

新規インストールでは以下のように設定すれば十分です。

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

5.14.4. mgmt-pub-interval 設定

Red Hat Enterprise Linux 7 の **/var/log/journal** に、以下のエラーが出力される場合があります。

```
satellite.example.com qpidd[92464]: [Broker] error Channel exception: not-  
attached: Channel 2 is not attached  
(/builddir/build/BUILD/qpidd-cpp-  
0.30/src/qpidd/amqp_0_10/SessionHandler.cpp: 39)  
  
satellite.example.com qpidd[92464]: [Protocol] error Connection  
qpidd.10.1.10.1:5671-10.1.10.1:53790 timed out: closing
```

このエラーメッセージは、**qpidd** が、キュー、セッション、接続に対する管理オブジェクトを維持し、デフォルトで 10 秒ごとに再利用するために表示されます。同じ ID を持つ同じオブジェクトが作成され、再度作成されます。古い管理オブジェクトは削除されていないため、**qpidd** によってこのエラーが発生します。この問題を回避するには、**/etc/qpidd/qpidd.conf** で、**mgmt-pub-interval** パラメーターの設定を 10 秒よりも短くし、**qpidd** サービスを再起動します。[Bugzilla 1335694](#) のコメント 7 も併せて参照してください。