Red Hat Satellite 6.12

Red Hat Satellite のパフォーマンスチューニング

Red Hat Satellite のパフォーマンスを最適化するためのガイド

Last Updated: 2024-04-13

Red Hat Satellite 6.12 Red Hat Satellite のパフォーマンスチューニング

Red Hat Satellite のパフォーマンスを最適化するためのガイド

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

法律上の通知

概要

このガイドの目的は、Red Hat Satellite 環境をスケールアップする際の参考資料として役立つ、一連のチューニング方法やヒントについて説明することです。

目次

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

Bugzilla でチケットを作成することでフィードバックを送信できます。

1. Bugzilla のWeb サイトに移動します。
2. **Component** フィールドで、**Documentation** を使用します。
3. **Description** フィールドに、ドキュメントの改善に関するご意見を記入してください。ドキュメントの該当部分へのリンクも追加してください。
4. **Submit Bug** をクリックします。

第1章 パフォーマンスチューニングの概要

このドキュメントでは、Red Hat Satellite をチューニングしてパフォーマンスとスケーラビリティーを確保するためのガイドラインを提供します。幅広いユースケースに適用できるように、細心の注意を払って内容を作成していますが、取り上げられていないユースケースがある場合には、Red Hat までお気軽にお問い合わせいただき、サポートを受けてください。

第2章 パフォーマンスチューニングのクイックスタート

インストールルーチンのチューニングフラグを使用して利用可能な、Satellite に組み込まれているチューニングプロファイルを使用すると、予想される管理対象ホスト数とハードウェアの割り当てに基づいて、Satellite Server をチューニングできます。詳細は、**接続されたネットワーク環境での Satellite Server のインストール**の 定義済みプロファイルによる Satellite Server のチューニング を参照してください。

Satellite が管理する管理対象ホストの数の見積もりに基づいて、4 つのサイズが提供されます。各プロファイルの具体的なチューニング設定は、**/usr/share/foreman-installer/config/foreman.hiera/tuning/sizes** に含まれている設定ファイルにあります。

名前	マネージドホスト数	推奨 RAM	推奨コア数
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 以上	256 GiB 以上	48 以上

手順

1. **default**、**medium**、**large**、**extra-large**、または **extra-extra-large** から、インストール環境のサイズを選択します。デフォルト値は **default** です。
2. **satellite-installer** を実行します。

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

3. オプション: ヘルスチェックを実行します。詳細は、「設定の適用」 を参照してください。
4. オプション: 「Puma のチューニング」セクションを使用して、Ruby アプリケーションサーバーを直接チューニングします。詳細は、「Puma のチューニング」 を参照してください。

第3章 チューニングのシステム要件

ハードウェアとソフトウェアの要件については、**オンラインネットワーク環境での Satellite Server のインストール** 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4章 ハードウェアと OS の設定の決定

CPU

Satellite で使用できる物理コアが多いほど、タスクのスループットを向上させることができます。Puppet や PostgreSQL などの一部の Satellite コンポーネントは CPU を大量に使用するアプリケーションであり、利用可能な CPU コアの数が多いほどメリットが得られます。

メモリー

Satellite を実行しているシステムで利用可能なメモリーの量が多いほど、Satellite の操作の応答時間が向上します。Satellite はデータベースソリューションとして PostgreSQL を使用しているため、メモリー追加とチューニングを組み合わせると、メモリー内に保持されるデータが増加するため、アプリケーションの応答時間が短縮されます。

ディスク

Satellite は、リポジトリーの同期、パッケージデータの取得、コンテンツホストのサブスクリプションレコードの頻繁なデータベース更新により、大量の IOPS を処理します。そのため、ディスクの読み書きの増加により発生する可能性のあるパフォーマンスのボトルネックを回避するために、Satellite は高速の SSD にインストールすることを推奨します。Satellite では、読み取り操作の平均スループットが 1 秒あたり 60 - 80 メガバイト以上のディスク IO が必要です。この値を下回ると、Satellite の操作に重大な影響を与える可能性があります。SSD は HDD と比較してレイテンシーが低いため、PostgreSQL などの Satellite コンポーネントは、SSD を使用することでメリットがあります。

ネットワーク

Satellite Server と Capsule 間の通信は、ネットワークパフォーマンスの影響を受けます。Satellite Server と Capsule の同期などの操作を手間なく実行できるようにするには、レイテンシーが低く、ジッターを最小限に抑えた適切なネットワークが必要です (少なくとも、接続のリセットなどが発生しないようにします)。

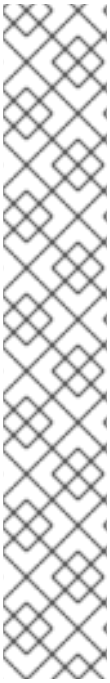
サーバーの電源管理

デフォルトでは、サーバーは電力を節約するように設定されている可能性があります。これは最大消費電力を抑えるには良い方法ですが、Satellite が達成できるパフォーマンスが低下するという副作用もあります。Satellite を実行しているサーバーの場合、システムをパフォーマンスモードで実行できるように BIOS を設定して、Satellite が達成できる最大パフォーマンスレベルを上げることを推奨します。

4.1. ディスクパフォーマンスのベンチマーク

Red Hat は、内部のクイックストレージベンチマークの結果が推奨スループットを下回った場合にのみユーザーに警告するように、**satellite-maintain** の更新に取り組んでいます。

また、より正確な実際のストレージ情報を取得するために実行可能なベンチマークスクリプトの更新にも取り組んでいます (このスクリプトは、今後 **satellite-maintain** に統合される見込みです)。



注記

- I/O ベンチマークを実行するには、RAM を一時的に縮小する必要がある場合があります。たとえば、Satellite Server に 256 GiB の RAM がある場合、テストを実行するには 512 GiB のストレージが必要です。回避策として、システムの起動中に grub で **mem=20G** のカーネルオプションを追加して、RAM のサイズを一時的に減らすことができます。ベンチマークは、指定されたディレクトリーに RAM の 2 倍のサイズのファイルを作成し、それに対して一連のストレージ I/O テストを実行します。このようなサイズのファイルを使用することで、このテストでファイルシステムのキャッシュ以外もテストされるようにしています。PostgreSQL ストレージなどの小規模なボリュームなど、他のファイルシステムのベンチマークを行う場合は、上記のように RAM のサイズを減らす必要がある場合があります。
- SAN や iSCSI などの異なるストレージソリューションを使用している場合は、想定されるパフォーマンスは異なります。
- Red Hat では、このスクリプトを実行する前にすべてのサービスを停止することを推奨します。

このテストはダイレクト I/O を使用せず、通常の操作と同様にファイルのキャッシュを使用します。

storage-benchmark というスクリプトの最初のバージョンを こちら で提供しています。このスクリプトを実行するには、スクリプトを Satellite にダウンロードし、実行可能にして、次を実行します。

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

スクリプトの README ブロックに記載されているように、通常、以下のテストで平均 100 MB/秒以上が得られる必要があります。

- ローカル SSD ベースのストレージでは、600 MB/秒以上の値が得られる必要があります。
- 回転ディスクでは、100 - 200 MB/秒以上の値が得られる必要があります。

値がこれを下回る場合は、サポートチケットを作成してサポートを受けてください。

詳細は、Impact of Disk Speed on Satellite Operations を参照してください。

4.2. TUNED プロファイルの有効化

Red Hat Enterprise Linux 8 は、インストール時にデフォルトで tuned デーモンを有効にします。ベアメタルでは、Satellite Server および Capsule に対して、tuned プロファイルの **throughput-performance** を実行することを推奨します。仮想マシンでは、**virtual-guest** プロファイルを実行することを推奨します。

手順

1. **tuned** が実行されているかどうかを確認します。

```
# systemctl status tuned
```

2. **tuned** が実行されていない場合は、有効にします。

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

3. オプション: 使用可能な **tuned** プロファイルのリストを表示します。

```
# tuned-adm list
```

4. シナリオに応じて、**tuned** プロファイルを有効にします。

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

4.3. TRANSPARENT HUGE PAGE の無効化

Transparent Huge Page は、Linux カーネルで使用されるメモリー管理手法です。より大きなサイズのメモリーページを使用することで、Translation Lookaside Buffer (TLB) を使用する際のオーバーヘッドを削減します。連続メモリーアクセスパターンではなく、スパースメモリーアクセスパターンをデータベースで使用するため、Transparent Huge Page が有効な場合、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。PostgreSQL と Redis のパフォーマンスを向上させるには、Transparent Huge Page を無効にします。データベースが別々のサーバーで実行されているデプロイメントでは、Satellite Server でのみ Transparent Huge Page を使用すると、若干メリットが得られる場合があります。

Transparent Huge Page を無効にする方法の詳細は、How to disable transparent hugepages (THP) on Red Hat Enterprise Linux を参照してください。

第5章 パフォーマンスのための SATELLITE の設定

Satellite には、相互に通信する多くのコンポーネントが付属しています。これらのコンポーネントをそれぞれ別個にチューニングすることで、シナリオに応じた最大限のパフォーマンスを実現できます。

Red Hat Enterprise Linux 7 にインストールされた Satellite と Red Hat Enterprise Linux 8 にインストールされた Satellite の間に大きなパフォーマンスの違いは見られません。

5.1. 設定の適用

次のセクションでは、さまざまな調整パラメーターとその適用方法を提案します。ほとんどの場合、Satellite の再起動が必要となるため、有効なバックアップと適切な停止期間を使用して、先に非実稼働環境でこれらの変更を常にテストしてください。

また、変更の効果を評価できるよう、変更を適用する前にモニタリングを設定することを推奨します。実際の環境を模倣するよう努めていますが、当社のテスト環境は、お客様の環境と大きく異なっている可能性があります。

systemd サービスファイルの変更

一部の systemd サービスファイルを変更した場合は、systemd デーモンに設定をリロードするよう通知する必要があります。

```
# systemctl daemon-reload
```

Satellite サービスを再起動します。

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

設定ファイルの変更

/etc/foreman-installer/custom-hiera.yaml などの設定ファイルを変更した場合は、インストーラーを再実行して変更を適用します。

```
# satellite-installer
```

追加オプションを指定してインストーラーを実行する

いくつかの新しいオプションを追加してインストーラーを再実行する必要がある場合は、以下を実行します。

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

セットアップの基本的な健全性を確認する

オプション: 変更を行った後は、次の簡単な Satellite ヘルスチェックを実行します。

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

5.2. PUMA のチューニング

Puma は、Foreman 関連のリクエストをクライアントに提供するために使用される ruby アプリケーションサーバーです。多数のクライアントまたは頻繁な操作を処理することが想定されている Satellite 設定では、Puma を適切にチューニングすることが重要です。

5.2.1. Puma スレッド数

Puma ワーカーごとの Puma スレッドの数は、**threads_min** と **threads_max** という 2 つの値を使用して設定されます。

threads_min の値は、ワーカーの起動時に各ワーカーが生成するスレッドの数を決定します。その後、同時リクエストが発生し、より多くのスレッドが必要になると、ワーカーは上限である **thread_max** に達するまでさらに多くのワーカーを生成します。

Puma スレッドが少ないと Satellite Server のメモリー使用量が増えるため、**threads_min** を **thread_max** と同じ値に設定することを推奨します。

例として、同時登録テストを使用して、次の 2 つの設定を比較しました。

8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン	8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン
--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**

Puma の最小スレッド数を **16** に設定すると、**thread_min=0** と比較してメモリー使用量が約 12% 少なくなります。

5.2.2. Puma のワーカー数とスレッド数の自動チューニング

Puma のワーカーとスレッドの値を **satellite-installer** で指定しなかった場合、またはそれらが Satellite 設定に存在しない場合、**satellite-installer** はバランスの取れたワーカー数を設定します。satellite-installer は次の式に従います。

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

ほとんどの場合はこれで問題ありませんが、使用パターンによっては、(他の Satellite コンポーネントがリソースを使用できるように) Puma 専用のリソースの量を制限するなどの理由で、チューニングが必要になる場合があります。各 Puma ワーカーは約 1 GiB の RAM を消費します。

現在の Satellite Server 設定を表示する

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

現在アクティブな Puma ワーカーを表示する

```
# systemctl status foreman
```

5.2.3. Puma のワーカー数とスレッド数の手動チューニング

「Puma のワーカー数とスレッド数の自動チューニング」 を利用しない場合は、これらの調整パラメーターにカスタム数値を適用できます。以下の例では、2 つのワーカーと、最小 5 つ、最大 5 つのスレッドを使用します。

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

変更を Satellite Server に適用します。詳細は、 「設定の適用」 を参照してください。

5.2.4. 推奨される Puma のワーカー数とスレッド数

さまざまなチューニングプロファイルのスレッドとワーカーの設定を推奨するために、Red Hat はさまざまなチューニングプロファイルを使用して、Satellite で Puma チューニングテストを実施しました。このテストで使用した主なテストは、さまざまなワーカー数とスレッド数を使用した、次の組み合わせによる同時登録です。当社の推奨値は、純粋に同時登録のパフォーマンスに基づいているため、正確なユースケースを反映していない可能性があります。たとえば、セットアップが非常にコンテンツ指向で、パブリッシュとプロモートが多い場合は、Pulp と PostgreSQL を優先して Puma によって消費されるリソースを制限することもできます。

名前	マネージドホスト数	RAM	コア	最小と最大の両方に推奨される Puma スレッド数	推奨される Puma ワーカー数
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 以上	256 GiB 以上	48 以上	16	20 - 26

ここではワーカー数のチューニングがより重要であり、場合によっては最大 52% のパフォーマンス向上が見られました。インストーラーはデフォルトで最小/最大 5 スレッドを使用しますが、上記の表のすべてのチューニングプロファイルでは、16 スレッドを使用することを推奨します。これは、4 スレッドでのセットアップと比較して、16 スレッドで最大 23% のパフォーマンス向上 (8 で 14%、32 で 10%) が見られたためです。

Red Hat はこの推奨値を明らかにするために、非常に特殊なユースケースである同時登録テストケースを使用しました。これは、(登録だけでなく) よりバランスの取れたユースケースを持つ Satellite では異なる場合があります。デフォルトの最小/最大 5 スレッドを維持することも良い選択です。

この推奨値の根拠となった測定値の一部を以下に示します。

	4 ワーカー、4 スレッド	4 ワーカー、8 スレッド	4 ワーカー、16 スレッド	4 ワーカー、32 スレッド
向上率	0%	14%	23%	10%

default セットアップ (4 CPU) では 4 - 6 つのワーカーを使用します。ワーカー 5 つの場合は、ワーカー 2 つと比較した場合、約 25% パフォーマンスが向上しました。しかし、ワーカー 8 つの場合は、ワーカー 2 つと比較した場合、パフォーマンスが 8% 低下しました。以下の表を参照してください。

	2 ワーカー、16 スレッド	4 ワーカー、16 スレッド	6 ワーカー、16 スレッド	8 ワーカー、16 スレッド
向上率	0%	26%	22%	-8%

medium セットアップ (8 CPU) では 8 - 12 個のワーカーを使用します。以下の表を参照してください。

	2 ワーカー、16 スレッド	4 ワーカー、16 スレッド	8 ワーカー、16 スレッド	12 ワーカー、16 スレッド	16 ワーカー、16 スレッド
向上率	0%	51%	52%	52%	42%

32 CPU のセットアップでは 16 - 24 個のワーカーを使用します (これは 90 GiB RAM マシンでテストされました。システムがスワッピングを開始し、メモリーがここでの 1 つの要因となっていることが判明したためです。適切な **extra-large** には 128 GiB が必要です)。ワーカー数をそれ以上に増やすと、高い同時登録レベルでテストした際に問題が発生したため、このような設定は推奨できません。

	4 ワーカー、16 スレッド	8 ワーカー、16 スレッド	16 ワーカー、16 スレッド	24 ワーカー、16 スレッド	32 ワーカー、16 スレッド	48 ワーカー、16 スレッド
向上率	0%	37%	44%	52%	失敗が多すぎる	失敗が多すぎる

5.2.5. Puma ワーカー数の設定

CPU が十分にある場合は、ワーカーを追加するとパフォーマンスが向上します。例として、8 CPU と 16 CPU の Satellite セットアップを比較しました。

表5.1 ワーカー数の効果をテストするために使用した satellite-installer のオプション

8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン	16 CPU、40 GiB RAM を搭載した Satellite 仮想マシン
--foreman-foreman-service-puma-threads-min=16	**--foreman-foreman-service-puma-threads-min=16**

8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン	16 CPU、40 GiB RAM を搭載した Satellite 仮想マシン
--foreman-foreman-service-puma-threads-max=16	**--foreman-foreman-service-puma-threads-max=16**
--foreman-foreman-service-puma-workers={2|4|8|16}	**--foreman-foreman-service-puma-workers={2|4|8|16}**

8 CPU のセットアップでは、ワーカー数を 2 から 16 に変更すると、同時登録時間が 36% 向上しました。16 CPU のセットアップでは、同じ変更で 55% の向上が見られました。

ワーカーを追加すると、Satellite が処理できる同時登録の総数も増やすことができます。私たちの測定では、2 つのワーカーを使用したセットアップで最大 480 の同時登録を処理できましたが、ワーカーを追加すると状況が改善されました。

手動チューニング

ワーカーの数を 2 に、スレッドの数を 5 に設定できます。

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

5.2.6. Puma スレッド数の設定

スレッド数が多いほど、ホストを並行して登録する時間を短縮できます。例として、次の 2 つの設定を比較しました。

8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン	8 CPU、40 GiB RAM を搭載した Satellite 仮想マシン
--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**

総スレッド数はそのままにして、使用するワーカー数を増やすと、同時登録のシナリオで約 11% の速度向上が得られます。さらに、ワーカーを追加しても CPU と RAM の消費量は増加しませんでしたが、パフォーマンスは向上しました。

5.2.7. Puma DB プールの設定

$db_pool の実効値は、自動的に **$foreman::foreman_service_puma_threads_max** と等しくなるように設定されます。これは **$foreman::db_pool** と **$foreman::foreman_service_puma_threads_max** の最大値ですが、どちらもデフォルト値が 5 であるため、最大スレッド数が 5 を超えると、データベース

接続プールが自動的に同じ量だけ増加します。

/var/log/foreman/production.log に、**ActiveRecord::ConnectionTimeoutError: could not obtain a connection from the pool within 5.000 seconds (waited 5.006 seconds); all pooled connections were in use** というエラーが見られたら、この値を増やすことを推奨します。

現在の db_pool 設定を表示

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

5.2.8. db_pool の手動チューニング

自動的に設定される値を利用しない場合は、次のようにカスタムの数値を適用できます。

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

変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

5.3. APACHE HTTPD のパフォーマンスチューニング

Apache httpd は Satellite のコア部分を形成し、Satellite Web UI または公開された API を介して行われるリクエストを処理する Web サーバーとして機能します。操作の同時実行性を高めるために、httpd は、チューニングが Satellite のパフォーマンスを向上させるのに役立つ最初のポイントを形成します。

5.3.1. Apache httpd で起動できるプロセスの数を設定する

デフォルトでは、HTTPD はプリフォークリクエスト処理メカニズムを使用します。リクエストを処理するプリフォークモデルを使用して、httpd はクライアントによる着信接続を処理するための新しいプロセスを起動します。

apache へのリクエストの数が、着信接続を処理するために起動できる子プロセスの最大数を超えると、httpd によって HTTP 503 サービス使用不可エラーが発生します。処理するプロセスが不足している httpd の中で、着信接続は、Pulp などのコンポーネントが httpd プロセスの可用性に依存しているため、Satellite 側で複数のコンポーネントの障害を引き起こす可能性もあります。

HTTPD プリフォークの設定を調整して、予想されるピーク負荷に基づいて、より多くの同時要求を処理できます。

Satellite への 150 の同時コンテンツホスト登録を処理することを望む可能性があるサーバーのプリフォーク設定の変更例は、次の設定ファイルの例のようになります **custom-hiera.yaml** ファイルの使用方法を参照してください。これにより設定ファイル **/etc/httpd/conf.modules.d/prefork.conf** が変更されます):

/etc/foreman-installer/custom-hiera.yaml を変更できます:

```
apache::mod::prefork::serverlimit: 582
apache::mod::prefork::maxclients: 582
apache::mod::prefork::startservers: 10
```

- **ServerLimit** パラメーターを設定して、MaxClients 値を上げます。
  詳細については、httpd ドキュメントの ServerLimit ディレクティブ を参照してください。

- **MaxClients** パラメーターを設定して、httpd が着信要求を処理するために起動できる子プロセスの最大数を制限します。
  詳細については、httpd ドキュメントの MaxRequestWorkers ディレクティブ を参照してください。

5.3.2. Apache HTTPD のオープンファイル数上限の設定

チューニングを行うと、Apache httpd はサーバー上で多くのファイル記述子を簡単に開くことができます。これは、ほとんどの Linux システムのデフォルトの制限を超える可能性があります。システムの最大オープンファイル数の上限を超えた結果として発生する可能性のある問題を回避するには、次のファイルとディレクトリーを作成し、以下の例で指定されているようにファイルの内容を設定してください。

手順

1. **/etc/systemd/system/httpd.service.d/limits.conf** で最大オープンファイル数の上限を設定します。

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

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

5.3.3. Apache Httpd の子プロセスのチューニング

デフォルトでは、httpd はイベント要求処理メカニズムを使用します。httpd へのリクエストの数が、着信接続を処理するために起動できる子プロセスの最大数を超えると、httpd で HTTP 503 Service Unavailable エラーが発生します。httpd で処理するプロセスが不足すると、コンポーネントで httpd プロセスを使用できるかどうかに左右されるので、Satellite サービス側の着信接続も複数のコンポーネントで問題が発生します。

httpd イベントの設定を調整して、予想されるピーク負荷に基づいてより多くの同時リクエストを処理できます。

警告

これらの番号を **custom-hiera.yaml** で設定すると、ロックされます。**satellite-installer --tuning=My_Tuning_Option** を使用してこれらの数値を変更すると、**custom-hiera.yaml** によってこの設定が上書きされます。固有の要件がある場合にのみ、数値を設定してください。

手順

1. 次の行を変更または追加して、/etc/foreman-installer/custom-hiera.yaml の同時リクエストの数を変更します。

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

この例は、Satellite Server で **Satellite-installer --tuning=medium** 以上を実行する場合と同じです。

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

5.4. QDROUTERD と QPID のチューニング

5.4.1. qdrouterd の最大オープンファイル数上限の計算

多数のコンテンツホストを備えた **katello-agent** インフラストラクチャーを使用するデプロイメントでは、qdrouterd の最大オープンファイル数を増やす必要がある場合があります。

qdrouterd のオープンファイル数の上限は、**(N x 3) + 100** という式で計算します。N はコンテンツホストの数です。各コンテンツホストはルーターで最大 3 つのファイル記述子を消費する可能性があり、ルーター自体を実行するには 100 個のファイル記述子が必要です。

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。

手順

1. **/etc/foreman-installer/custom-hiera.yaml** で最大オープンファイル数の上限を設定します。

```
qpid::router::open_file_limit: "My_Value"
```

デフォルト値は **150100** です。

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

5.4.2. qpidd の最大オープンファイル数上限の計算

多数のコンテンツホストを備えた **katello-agent** インフラストラクチャーを使用するデプロイメントでは、qpidd の最大オープンファイル数を増やす必要がある場合があります。

qpidd のオープンファイル数の上限は、**(N x 4) + 500** という式で計算します。N はコンテンツホストの数です。1 つのコンテンツホストは最大 4 つのファイル記述子を使用する可能性があり、Broker (qpidd のコンポーネント) の操作には 500 個のファイル記述子が必要です。

手順

1. **/etc/foreman-installer/custom-hiera.yaml** で最大オープンファイル数の上限を設定します。

```
qpid::open_file_limit: "My_Value"
```

デフォルト値は **65536** です。

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

5.4.3. 最大非同期入出力リクエストの設定

多数のコンテンツホストを備えた **katello-agent** インフラストラクチャーを使用するデプロイメントでは、許容される同時 AIO リクエストの最大数を増やす必要がある場合があります。カーネルパラメーター **fs.aio-max-nr** の値を増やすことで、許容される同時 AIO リクエストの最大数を増やすことができます。

手順

1. **/etc/sysctl.d** 内のファイルで、**fs.aio-max-nr** の値を目的の最大値に設定します。

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

この数値が、Satellite に登録する予定のコンテンツホストの最大数に 33 を乗算した値より大きいことを確認してください。

2. 変更を適用します。

```
# sysctl -p
```

3. オプション: この変更が確実に適用されるように、Satellite Server を再起動します。

5.4.4. ストレージに関する考慮事項

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、事前に **/var/lib/qpidd** に十分なストレージ領域を確保してください。Satellite Server では、1 つのコンテンツホストに対して **/var/lib/qpidd** のディスク領域が 2 MiB 必要になります。

5.4.5. QPID mgmt-pub-interval パラメーターの設定

Red Hat Enterprise Linux 7 のジャーナルに次のエラーが表示される場合があります (**journalctl** コマンドを使用してアクセスします)。

```
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
```

このエラーメッセージが表示されるのは、qpid がキュー、セッション、および接続の管理オブジェクトを保持し、それらをデフォルトで 10 秒ごとに再利用するためです。同じ ID を持つ同じオブジェクトが作成、削除され、再作成されます。古い管理オブジェクトは削除されていないため、qpid でこのようなエラーが発生します。

手順

1. **/etc/foreman-installer/custom-hiera.yaml** で **mgmt-pub-interval** パラメーターを設定します。

```
qpid::mgmt_pub_interval: 5
```

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。
詳細は、BZ 1335694 を参照してください。

5.5. DYNFLOW のチューニング

Dynflow は、ワークフロー管理システムおよびタスクオーケストレーターです。これは Satellite のプラグインであり、Satellite のさまざまなタスクをアウトオブオーダー実行方式で実行するために使用されます。多くのクライアントが Satellite にチェックインし、多数のタスクを実行している状況では、追加のチューニングを行って起動できるエグゼキューターの数を指定することで、Dynflow のパフォーマンスを若干向上できます。

Dynflow に関連するチューニングの詳細は、**https://satellite.example.com/foreman_tasks/sidekiq** を参照してください。

Sidekiq ワーカーの数を増やす

Satellite には、Dynflow によってスケジュールされたタスクを実行する **dynflow-sidekiq** と呼ばれる Dynflow サービスが含まれています。Sidekiq ワーカーをさまざまなキューにグループ化して、あるタイプのタスクの多くが他のタイプのタスクの実行をブロックしないようにすることができます。

Red Hat は、複数のコンテンツビューの公開とプロモーション、コンテンツの同期、Capsule Server への同期など、一括同時タスク用に Foreman タスクシステムを拡張するために、sidekiq ワーカーの数を増やすことを推奨します。次の 2 つの方法を使用できます。

- ワーカーが使用するスレッドの数 (ワーカーの同時実行数) を増やすことができます。Ruby にはスレッドの同時実行性が実装されているため、値を 5 より大きくしても、効果は限定的です。
- ワーカーの数を増やすことができます。こちらの方法を推奨します。

手順

1. ワーカー数を 1 から 3 に増やします。各ワーカーのスレッド数/同時実行数は 5 つのままにします。

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

2. オプション: ワーカーサービスが 3 つあるかどうかを確認します。

```
# 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
```

詳細は、How to add sidekiq workers in Satellite6? を参照してください。

5.6. プルベースの REX トランスポート調整

Satellite には、リモート実行用のプルベースのトランスポートモードがあります。このトランスポートモードは、メッセージングプロトコルとして MQTT を使用し、各ホストで実行される MQTT クライアントを含みます。詳細は、**ホストの管理** の リモート実行のトランスポートモード を参照してください。

5.6.1. プルベースの REX トランスポートのホスト制限の増加

mosquitto MQTT サーバーを調整して、接続するホストの数を増やすことができます。

手順

1. Satellite Server または Capsule Server でプルベースのリモート実行を有効にします。

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

Satellite Server または Capsule Server が使用できる転送モードは、SSH または MQTT のいずれか 1 つのみであることに注意してください。

2. MQTT サービスによって受け入れられるデフォルトのホスト数を増やすための設定ファイルを作成します。

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

この例では、**mosquitto** サービスが 5000 ホストを処理できるように制限を設定します。

3. 次のコマンドを実行して、変更を適用します。

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

5.6.2. プルベースの REX トランスポートのパフォーマンスへの影響の軽減

Satellite Server が Script プロバイダーを使用してリモート実行ジョブのプルベースのトランスポートモードで設定されている場合、Capsule Server は新しいジョブに関する通知を MQTT 経由でクライアントに送信します。この通知には、クライアントが実行するはずの実際のワークロードは含まれません。クライアントは、新しいリモート実行ジョブに関する通知を受け取ると、実際のワークロードについて Capsule Server に問い合わせます。ジョブ中、クライアントは定期的にジョブの出力を Capsule Server に送信するため、Capsule Server へのリクエストの数がさらに増加します。

Capsule Server へのこれらのリクエストと、MQTT プロトコルによって許可される高い同時実行性により、Capsule Server で使用可能な接続が枯渇する可能性があります。一部のリクエストが失敗し、リモート実行ジョブの一部の子タスクが応答しなくなる場合があります。これは、実際のジョブのワークロードにも依存します。これは、一部のジョブが Satellite Server に追加の負荷を発生させ、クライアントが Satellite Server に登録されている場合にリソースに対して競合するためです。

これを回避するには、Satellite Server と Capsule Server を次のパラメーターで設定します。

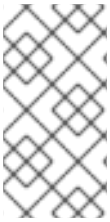
- MQTT Time To Live – ジョブが配信されていないとみなされる前にホストにジョブを取得するために与えられる時間間隔 (秒単位)
- MQTT 再送信間隔 – ジョブが選択されるかキャンセルされるまで、ホストに通知を再送信する時間間隔 (秒)
- MQTT レート制限 – 同時に実行できるジョブの数。レート制限を調整することでリモート実行の同時実行性を制限できます。これは、Satellite により多くの負荷をかけることを意味します。

手順

- Satellite サーバーの MQTT パラメーターを調整します。

```
# 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 のログは **/var/log/foreman-proxy/proxy.log** にあります。Capsule Server は Webrick HTTP サーバー (httpd や Puma は関与しません) を使用するため、その容量を増やす簡単な方法はありません。



注記

ワークロード、ホストの数、利用可能なリソース、および適用されたチューニングによっては、Bug 2244811 が発生する可能性があります。これにより、Capsule が大量のメモリーを消費し、最終的に強制終了となり、残りのジョブが失敗します。現時点では、普遍的に適用できる回避策はありません。

5.7. POSTGRESQL のチューニング

PostgreSQL は、Satellite が実行するさまざまなタスクの永続的なコンテキストを保存するために、Satellite によって使用される SQL ベースの主要なデータベースです。このデータベースは、Satellite のスムーズな動作に必要なデータを提供するために、通常、広範に使用されています。そのため、PostgreSQL は頻繁に使用されるプロセスであり、PostgreSQL をチューニングすることで、Satellite の全体的な動作の応答に多くの利点がもたらされます。

PostgreSQL の作成者は、PostgreSQL を実行しているサーバーで Transparent Huge Page を無効にすることを推奨しています。詳細は、「Transparent Huge Page の無効化」 を参照してください。

PostgreSQL に一連のチューニングを適用することで、応答時間を改善できます。適用すると、**postgresql.conf** ファイルが変更されます。

手順

1. **/etc/foreman-installer/custom-hiera.yaml** を追加して PostgreSQL をチューニングします。

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

   これを使用すると、チューニングプロファイルに関係なく、Satellite インスタンスを効果的にチューニングできます。

2. 変更を Satellite Server に適用します。詳細は、「設定の適用」 を参照してください。

上記のチューニング設定には、変更した特定のキーのセットがあります。

- **max_connections**: このキーは、実行中の PostgreSQL プロセスが受け入れることができる接続の最大数を定義します。
- **shared_buffers**: 共有バッファーは、さまざまなデータベース操作のデータを格納するために PostgreSQL 内のすべてのアクティブな接続によって使用されるメモリーを定義します。このキーの最適な値は、Satellite で実行される操作の頻度に応じて、2 GiB からシステムメモリー合計の 25% までの間で変動します。
- **work_mem**: work_mem は、PostgreSQL のプロセスごとに割り当てられるメモリーであり、プロセスによって実行されている操作の中間結果を格納するために使用されます。この値を 8 MB に設定すれば、Satellite でのほとんどの集中的な操作に十分対応できます。

- **autovacuum_vacuum_cost_limit**: このキーは、データベースリレーション内のデッドタプルをクリーンアップする autovacuum プロセス内のバキューム操作のコスト制限値を定義します。コスト制限は、プロセスによる 1 回の実行で処理できるタプルの数を定義します。Red Hat は、**medium**、**large**、**extra-large**、および **extra-extra-large** のプロファイルと同様に、Satellite が PostgreSQL サーバープロセスに加える一般的な負荷に基づいて、この値を **2000** に設定することを推奨します。

詳細は、BZ1867311: Upgrade fails when checkpoint_segments postgres parameter configured を参照してください。

5.7.1. 生の DB パフォーマンスのベンチマーク

Candlepin、Foreman、Pulp 用のディスク領域の上位テーブルサイズのリストを取得するには、satellite-support git リポジトリーの postgres-size-report スクリプトを確認してください。

PGbench ユーティリティーを使用して、システムの PostgreSQL パフォーマンスを測定できます (場合によっては、PostgreSQL データディレクトリー **/var/lib/pgsql** のサイズを 100 GiB またはベンチマークの実行に必要なサイズに変更する必要があります)。このユーティリティーは、**dnf install postgresql-contrib** を使用してインストールします。詳細は、github.com/RedHatSatellite/satellite-support を参照してください。

PostgreSQL データディレクトリーのファイルシステムの選択も重要になる場合があります。

警告

- 有効なバックアップがない状態で、実稼働システムでテストを実行しないでください。
- テストを開始する前に、データベースファイルのサイズを確認してください。非常に小さなデータベースでテストしても、意味のある結果は得られません。たとえば、DB がわずか 20 GiB で、バッファープールが 32 GiB の場合、データが完全にバッファリングされるため、接続の数が多くても問題が発生することはありません。

5.8. REDIS のチューニング

Redis はインメモリーデータストアです。これは、Satellite の複数のサービスで使用されます。Dynflow および Pulp タスクシステムは、これを使用してタスクを追跡します。Satellite が Redis を使用する方法を考えると、そのメモリー消費量は安定しているはずです。

Redis の作成者は、Redis を実行しているサーバーで Transparent Huge Page を無効にすることを推奨しています。詳細は、「Transparent Huge Page の無効化」を参照してください。

5.9. CAPSULE 設定のチューニング

Capsule は、Satellite の負荷の一部をオフロードし、クライアントへのコンテンツの配信に関連するさまざまなネットワークへのアクセスを提供することを目的としていますが、リモート実行ジョブの実行にも使用できます。ホスト登録やパッケージプロファイルの更新など、Satellite API を広範囲に使用するものについては、サポートできません。

5.9.1. Capsule のパフォーマンステスト

Red Hat は、複数の Capsule 設定で複数のテストケースを測定しました。

Capsule HW 設定	CPU	RAM
minimal	4	12 GiB
large	8	24 GiB
extra large	16	46 GiB

コンテンツ配信のユースケース

ダウンロードテストでは、2000 個のパッケージがある 40 MB リポジトリーを、100 台、200 台、(中略。以下 100 ずつ増加)、1000 台のホストで同時にダウンロードしました。Capsule Server のリソースを 2 倍にするたびに、平均ダウンロード時間が約 50% 向上しました。より正確な数値については、以下の表を参照してください。

同時ダウンロードホスト	Minimal (4 CPU および 12 GiB RAM) → Large (8 CPU および 24 GiB RAM)	Large (8 CPU および 24 GiB RAM) → Extra Large (16 CPU および 46 GiB RAM)	Minimal (4 CPU および 12 GiB RAM) → Extra Large (16 CPU および 46 GiB RAM)
平均向上率	約 50% (例: 700 台での同時ダウンロードの場合に、1 パッケージあたり平均 9 秒が4.4 秒に向上)	約 40% (例: 700 台での同時ダウンロードの場合に、1 パッケージあたり平均 4.4 秒が2.5 秒に向上)	約 70% (例: 700 台での同時ダウンロードの場合に、1 パッケージあたり平均 9 秒が2.5 秒に向上)

Satellite Server と Capsule Server のダウンロードパフォーマンスを比較したところ、約 5% の速度向上しか見られませんでした。しかし、Capsule Server の主な利点は、コンテンツを地理的に分散したクライアント (または異なるネットワーク内のクライアント) に近づけることや、Satellite Server 自体が処理する必要がある負荷の一部を処理することにあります。小規模なハードウェア設定 (8 CPU および 24 GiB) の Satellite Server は、500 を超える同時クライアントからのダウンロードを処理できませんでしたが、同じハードウェア設定の Capsule Server は、1000 以上のクライアントにサービスを提供できました。

同時登録のユースケース

同時登録では、通常、ボトルネックは CPU 速度ですが、すべての設定で、スワッピングせずに高い同時実効性を実現することができました。Capsule に使用されるハードウェアリソースは、登録パフォーマンスに最小限の影響しか与えません。たとえば、16 個の CPU と 46 GiB の RAM を備えた Capsule Server は、4 個の CPU と 12 GiB の RAM を備えた Capsule Server と比較して、最大で 9% しか登録速度が向上しません。同時実行性が非常に高くなると、Capsule Server から Satellite Server への通信でタイムアウトが発生する可能性があります。これを軽減するには、**/etc/foreman-installer/custom-hiera.yaml** で以下の調整可能パラメーターを使用し、デフォルトのタイムアウト時間を延長します。

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

リモート実行のユースケース

500 台、2000 台、4000 台のホストで SSH と Ansible バックエンドの両方を介してリモート実行ジョブを実行することをテストしました。4000 台すべてのホストで完了できなかった最小の設定 (4 CPU と 12 GiB メモリー) を除いて、すべての設定ですべてのテストをエラーなしで処理できました。

コンテンツ同期のユースケース

Red Hat Enterprise Linux 6、7、8 BaseOS、および 8 AppStream を同期した同期テストでは、Capsule 設定間で大きな違いは見られませんでした。これは、より多くのコンテンツビューを並行して同期する場合とは異なります。