Red Hat JBoss Operations Network 3.3

Configuring JON Servers and Agents

Red Hat JBoss Operations Network 3.3 およびそのパッチリリース

Last Updated: 2020-08-30

Red Hat JBoss Operations Network 3.3 Configuring JON Servers and Agents

Red Hat JBoss Operations Network 3.3 およびそのパッチリリース

Jared Morgan
jmorgan@redhat.com

Zach Rhoads
zach@redhat.com

Ella Deon Ballard
dlackey@redhat.com

法律上の通知

Copyright © 2017 Red Hat.

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, the Red Hat 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 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

本ガイドを参照して、製品に関連するプラットフォーム、エージェント、およびストレージノードを設定する方法を説明します。

目次

第1章 JBOSS OPERATIONS NETWORK について 4
1.1. JBOSS ON エージェント 4
1.2. JBOSS ON サーバーについて 4
1.3. メトリクスストレージノードについて 5

第2章 一般的な管理 7
2.1. JBOSS ON ファイルの場所 7
2.2. デフォルトのサーバー、エージェント、およびストレージノードポート 12
2.3. RHQCTL コントロールスクリプト 13
2.4. JBOSS ON サーバーの起動 16
2.5. JBOSS ON エージェントの起動 20
2.6. ストレージノードの起動 25

第3章 パフォーマンスを向上するための JBOSS ON の調整 27
3.1. インベントリーのベースライン 27
3.2. 監視およびアラートベースライン 29
3.3. エージェント JVM メモリーサイズの調整 30
3.4. エージェント JVM ヘルスチェック設定の変更 30
3.5. サーバー JVM の調整 31
3.6. 多数のエージェントのサーバーチューニング 32
3.7. データベースおよびディスク領域 34

第4章 サーバーエージェント通信の SSL 接続の設定 35
4.1. 暗号化の設定 35
4.2. サーバーおよびエージェント間のクライアント認証の設定 38
4.3. SSL の問題のトラブルシューティング 43

第5章 高可用性とエージェントのロードバランシングの使用 47
5.1. エージェントサーバー通信とサーバーの可用性について 47
5.2. アフィニティーグループの作成 52
5.3. サーバーをメンテナンスモードに 56
5.4. 高可用性クラウドからのサーバーの削除 57
5.5. パーティションイベントの管理 58

第6章 サーバーの設定 63
6.1. JBOSS ON サーバーのデバッグロギングの有効化 63
6.2. JBOSS ON SERVER の URL の変更 65
6.3. RHQ-SERVER.PROPERTIES での JBOSS ON SERVER 設定の編集 67
6.4. サーバー設定の同期 83

第7章 エージェントの設定 91
7.1. エージェントの登録および再登録 91
7.2. エージェントの削除 95
7.3. エージェントのコマンドプロンプトの使用 97
7.4. エージェントおよびリソースのシステムユーザーとの対話 100
7.5. ROOT 以外のユーザーとしてエージェントの実行 104
7.6. エージェントのデバッグモードの有効化 105
7.7. エージェントの IP アドレスの変更 108
7.8. エージェントのリソースとしての管理 108
7.9. エージェントのクエット時間（タイムアウト時間）の設定 110
7.10. エージェント更新の設定 111
7.11. エージェントの永続設定の管理 113
7.12. エージェント JVM の管理 118

7.13. 共有ディレクトリーまたはアカウントを使用した複数のエージェントのインストール 118
7.14. 検出スキャン間隔の設定 120
7.15. エージェントのサーバーフェイルオーバー一覧の表示 121
7.16. サーバーを検出またはポーリングするエージェントの設定 123
7.17. エージェントのスロットリング 125
7.18. コマンドの GUARANTEED DELIVERY の設定 126
7.19. エージェント通信の設定 127

第8章 JBOSS ON に関連付けられたデータベースの管理 129
8.1. JBOSS ON からの SQL コマンドの実行 129
8.2. データベースパスワードの変更 129
8.3. JBOSS ON SERVER のデータベース設定の編集 130

第9章 ストレージノードのデプロイおよび管理 132
9.1. HIGH-SPEED METRICS STORAGE 132
9.2. ストレージノードのデプロイおよびアンデプロイ 134
9.3. ストレージノードのメトリクスおよび状態の表示 141
9.4. クラスターポートの変更 143
9.5. ストレージノードアラートの表示 145
9.6. ストレージノードのデバッグモードの有効化 147
9.7. ストレージノードのヒープの管理 148
9.8. JBOSS ON サーバーの CONTAINERSANDRA STORAGE ノードのスケーリング 149
9.9. ストレージノードの容量の調整によるメトリクス受信（ストレージノードの要求スロットリング） 151
9.10. メトリクスストレージデータベースのバックアップおよび復元 153

付録A ドキュメント履歴 162

第1章 JBOSS OPERATIONS NETWORK について

JBoss ON の主な用途は、管理者がシステムを表示するための単一アクセスを可能にすることです。機能は機能的に、JBoss ON はシステムの **インベントリー** を開発および監視する手段を提供します。プラットフォーム、アプリケーション、サービスなどの **管理リソース** はすべて、IT 環境が複雑であってもインベントリーに含まれ、整理されます。

JBoss ON では、インストールされた **サーバー** のすべての操作を一元化します。JBoss ON サーバーは、ローカルにインストールされた JBoss ON **エージェント** と通信します。これはプラットフォームやサービスと直接対話して、監視などのローカルタスクを実行します。JBoss ON が管理できるリソースタイプと実行可能な操作は、JBoss ON でロードされるサーバーおよびエージェントプラグインによって決まります。

サーバー、エージェント、プラグイン、およびリソース間の関係は、JBoss ON を定義するものです。

バグを報告します。

1.1. JBOSS ON エージェント

JBoss ON エージェントは JBoss ON が管理するすべてのマシンにデプロイされます。エージェントは、リソース自体と中央の JBoss ON サーバーとの間の仲介です。

エージェントは、設定変更、更新されたパッケージ、アラートの新たな設定、JBoss ON サーバーから操作などの更新を受信し、それらのタスクをリソース上で実行します。また、エージェントはサーバーに転送するリソースから情報を収集します。これにより、サーバーはリソースのアラート、メトリクス、および可用性の情報を処理することができます。

エージェントはサーバーから独立しているため、サーバーが停止したり、リソースがネットワーク接続が失われた場合でも、監視タスクを続行し、リソースに関する情報を収集できます。

各リソースは階層で編成され、プラットフォーム、サーバー、およびサービスの関係を表示します。マシンごとにエージェントは 1 つだけ必要です。プラットフォームがリソースとして管理されると、インストールされているアプリケーションまたはサービスのサブセットをリソースとして追加できます。すべては、同じローカルエージェントを使用します。

バグを報告します。

1.2. JBOSS ON サーバーについて

JBoss ON は中央サーバーを中心に構築されます。サーバーは 2 つの必須機能を実行します。

- リソースとリソースグループの両方の設定を保存します。
- エージェントが収集するデータを整理して応答します。

JBoss ON サーバーは、管理者が操作環境を管理するための中心的な場所です。サーバーは、ベースラインの設定およびアプリケーションのプロビジョニング、アラートと通知の定義、および操作の開始に使用されます。エージェントがリソースからサーバーに情報を送信すると、サーバーは監視タスクも実行できます（メトリクスとレポートを提供することにより）。また、アラートや起動操作を送信することでイベントに応答することもできます。

JBoss ON サーバーによって使用されるデータは、バックエンド SQL データベースに保存されます。これらのデータには以下が含まれます。

- リソースのインベントリー

- 設定済みのグループ
- データのモニタリング
- 設定データ
- リソースで利用可能なコンテンツ
- ユーザーおよびアクセス制御情報

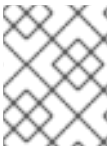
JBoss ON サーバーは、JBoss ON と対話するために使用されるグラフィカルユーザーインターフェースをホストします。

JBoss ON サーバーの非常に重要な機能の 1 つは、バックエンドデータベースおよび JBoss ON エージェントのみと通信することです。JBoss ON サーバーが同じバックエンドデータベースを使用している場合は、サーバーまたはデータベースに追加の設定がなくても、フェイルオーバーとスケーラビリティーが可能になります。JBoss ON エージェントは、実際にサーバー間でエージェント負荷を分散し、詳細な設定なしにエージェントサーバーフェイルオーバーを提供する、推奨される JBoss ON サーバーのリストを使用するように設定できます。

バグを報告します。

1.3. メトリクスストレージノードについて

メトリクスの収集およびアラートの生成は、リソース集約型と言えます。これにより、バックエンドデータベース上でのほぼコンスタント書き込み操作が可能になります。これにより、パフォーマンスの低下に遭遇する前に収集できるメトリクスの数の許容しきい値(minute per minute)を作成します。



注記

毎分の メトリクスの許容しきい値は、内部のパフォーマンステストに基づいています。このしきい値は、システムリソースと全体的な負荷によって異なります。

JBoss ON は 2 つのデータベースを使用して情報を保存します。1 つは、JBoss ON サーバーおよびエージェント、すべてのリソースインベントリーデータ、リソース設定、およびその他のデータに関するすべての設定を格納する中央リレーショナルデータベース（PostgreSQL または Oracle）です。他のデータベースは、すべての数値監視データを保存する分散データベース（ストレージノードのクラスター）です。つまり、収集したすべてのメトリクスです。

メトリクスストレージノードは、独自の専用マシンにインストールできます。これにより、データベースに対する書き込みパフォーマンスが大幅に向上します（これにより、モニタリングのパフォーマンスが向上します）。

- 専用 CPU
- 利用可能な物理メモリーがさらに必要です。
- 高速ディスク
- より多くのディスク領域

これは、JBoss ON サーバーとは別のマシンにリレーショナルデータベースをインストールする場合と同じパフォーマンス上の考慮事項です。2 つのデータベースを使用することで、書き込み集約型のリソース集約型のメトリックストレージをリソース集中型の構成データ（Dributing snapshots やバンドル設定など）から移動することもできます。

さらに、分散データベースは、クラスター内の複数のノードを使用して拡張できます。負荷に応じてノードを追加することは、管理者にとって重要な管理ツールです。1分あたりに収集される | メトリックのハードウェア駆動な制限に遭遇する代わりに、パフォーマンスを向上させるために追加のノードを追加できます。

ストレージノードクラスターは JBoss ON によって作成および管理されます（メトリクスストレージノード管理のオーバーヘッドも最小限に抑えられます）。**rhqctl install --storage** コマンドを使用して、ストレージノードがシステムにインストールされている。ストレージノードには、常にコンパニオンエージェントが必要です。

バグを報告します。

第2章 一般的な管理

ここでは、JBoss Operations Network サーバーおよびエージェントの設定、ファイル、およびオプションについて説明します。

バグを報告します。

2.1. JBOSS ON ファイルの場所

ここでは、JBoss ON サーバーおよびエージェントによる共通のファイルおよびディレクトリーについて説明します。これらのファイルの基本リファレンスにより、JBoss ON の管理およびトラブルシューティングが容易になります。

バグを報告します。

2.1.1. サーバーファイルの場所

すべての JBoss Operations Network サーバーは、ユーザー定義の単一のサーバールートディレクトリーにインストールされます。ドキュメントおよび例では、serverRoot という名前 **です**。そのサーバールートディレクトリー内のディレクトリーレイアウトは、サーバーごとに同じです。

```
                    serverRoot
                       |
                      jon
                       |
     -------------------------------------------------------
     |        |    |    |     |      |      |      |
alert-scripts/  bin/  etc/  EULA  jbossas/  LICENSE  logs/  plugins/
```

JBoss ON サーバーを管理するのに最も一般的に使用されるディレクトリーおよびファイルは、に記載されてい 表2.1「サーバーディレクトリーおよびファイル」 ます。サーバールートはインストールとプラットフォームごとに異なりますが、JBoss ON サブディレクトリーのレイアウトはすべてのプラットフォームで同じです。

表2.1 サーバーディレクトリーおよびファイル

設定エリア	ディレクトリーまたはファイルの場所	description
設定ディレクトリー	**serverRoot**/bin/	サーバー起動スクリプト、PID ファイル、設定ファイルが含まれます。
スクリプトの起動	**serverRoot**/bin/rhqctl	サーバー、ストレージノード、およびローカルエージェントの起動、停止、インストール、およびアップグレードに使用される管理スクリプト。
サーバー設定ファイル	**serverRoot**/bin/rhq-server.properties	JBoss ON データベースに保存されていないすべてのサーバー設定の設定ファイル。

設定エリア	ディレクトリーまたはファイルの場所	description
ストレージノードの設定ファイル	**serverRoot**/bin/rhq-storage.properties	JBoss ON データベースに保存されていないすべてのストレージノード設定の設定ファイル。
パスワードハッシュスクリプト	**serverRoot**/bin/rhq-encode-password.{sh|bat}	移行したサーバーでは、**rhq-server.properties** ファイルで使用するデータベースパスワードのエンコード形式が生成されます。
SNMP ファイル	**serverRoot**/etc/RHQ-mib.txt	SNMP トラップの設定に使用する JBoss ON MIB ファイル。
ログファイル	**serverRoot**/logs/	JBoss ON サーバーのログファイルは、このディレクトリーに自動的に作成されます。現在のログには名前が付けられ **rhq-server-log4j.log**ます。古いログファイルの名前は **rhq-server-log4j.log#** で、それよりも古いログファイルの数が高くなります。
カスタムプラグインのデプロイメントディレクトリー	**serverRoot**/plugins/	カスタムプラグインファイルが JBoss ON サーバーによって自動的に検出およびポーリングされるためにカスタムプラグインファイルをドロップできるディレクトリー。
JBoss AS ディレクトリー	**serverRoot**/jbossas/	必要な JBoss AS クライアントおよびサーバーライブラリーがすべて含まれています。[a]
サーバー JAR ファイル	**serverRoot**/jbossas/default/deploy/rhq.ear/	JBoss ON サーバー、Web インターフェース、およびクライアントによって使用される JAR ファイルをすべて格納します。
サーバー側のプラグインディレクトリー	**serverRoot**/jbossas/default/deploy/rhq.ear/rhq-serverplugins/	デフォルトの JBoss ON サーバー側のプラグインのすべての JAR ファイルが含まれます。
エージェントプラグインディレクトリー	**serverRoot**/jbossas/default/deploy/rhq.ear/rhq-downloads/rhq-plugins/	デフォルトの JBoss ON エージェントプラグインのすべての JAR ファイルが含まれます。

設定エリア	ディレクトリーまたはファイルの場所	description
サーバー側のプラグインディレクトリー	**serverRoot**/jbossas/default/deploy/rhq.ear/rhq-serverplugins/	デフォルトの JBoss ON サーバー側のプラグインのすべての JAR ファイルが含まれます。
エージェントパッケージディレクトリー	**serverRoot**/jbossas/default/deploy/rhq.ear/rhq-downloads/rhq-agent/	JBoss ON エージェントのスナップショットパッケージが含まれます。
Web インターフェースディレクトリー	**serverRoot**/jbossas/default/work/jboss.web/localhost/	Web インターフェースをレンダリングするためのファイルを保持するディレクトリーが含まれます。
[a] このディレクトリーのライブラリーおよびファイルの大半は JBoss ON と直接関係しません。		

バグを報告します。

2.1.2. エージェントファイルの場所

サーバーと同様に、JBoss ON エージェントはユーザー定義のルートディレクトリーにインストールされます。すべてのエージェントファイルとディレクトリーは、そのルートディレクトリー内の **rhq-agent**/ ディレクトリーの下にあります。

```
                 serverRoot
                     |
                rhq-agent/
                     |
     -----------------------------------
     |    |    |    |    |    |
    bin/  conf/  data/  lib/  logs/  plugins/
```

表2.2 エージェントディレクトリーとファイル

設定エリア	ディレクトリーまたはファイルの場所	description
スクリプトの起動	**serverRoot**/rhq-agent/bin/	エージェントの起動スクリプトが含まれます。
設定ファイル	**serverRoot**/rhq-agent/conf/agent-configuration.xml	基本的なエージェント設定用の設定ファイル。
ライブラリーファイル	**serverRoot**/rhq-agent/lib/	リソースを監視するためにエージェントによって使用されるライブラリーが含まれます。

設定エリア	ディレクトリーまたはファイルの場所	description
スクリプトの起動	**serverRoot**/rhq-agent/logs/	JBoss ON エージェントのログファイルは、このディレクトリーに自動的に作成されます。現在のログには名前が付けられ **agent.log** ます。古いログファイルの名前は **agent.log#** で、それよりも古いログファイルの数が高くなります。
プラグインディレクトリー	**serverRoot**/rhq-agent/plugins/	リソースの管理に使用するエージェントによって使用されるプラグイン（リソース設定の編集など）が含まれます。

バグを報告します。

2.1.3. ストレージノードファイルの場所

ストレージノードがサーバーのディレクトリーにインストールされている **/opt/jon/jon-server-3.3.0.GA**。

```
                 serverRoot
                     |
               jon-server-3.3.0.GA/
                     |
      ----------------------------------------
      |     |    |          |     |      |
     bin/  conf/  interface/  lib/   pylib/  tools/
```

また、すべてのメトリクスデータとバックアップデータを格納する個別のディレクトリーがあります。

```
          /opt/jon
              |
           rhq-data/
              |
      ---------------------
      |           |     |
     commit_log/  data/  saved_caches/
```

表2.3 ストレージディレクトリーおよびファイル

設定エリア	ディレクトリーまたはファイルの場所	description

設定エリア	ディレクトリーまたはファイルの場所	description
分散データベーススクリプト	**serverRoot**/rhq-storage/bin/	分散データベースノードの管理に使用するスクリプトが含まれています。これらは管理者ではなくストレージノードを管理する JBoss ON エージェントによって最も頻繁に呼び出されます。
PID ファイル	**serverRoot**/rhq-storage/bin/cassandra.pid	ノードプロセスの PID ファイル。
設定ファイル	**serverRoot**/rhq-storage/conf/rhq-storage-auth.conf	ストレージノードの基本設定用の設定ファイル。
ロギング設定ファイル	**serverRoot**/rhq-storage/conf/log4j-server.properties	ストレージノードの log4j の設定ファイル。
ライブラリーファイル	● **serverRoot**/rhq-agent/lib/ ● **serverRoot**/rhq-agent/pylib/ ● **serverRoot**/rhq-agent/tools/	ストレージノードによって使用されるライブラリーが含まれます。
データファイル	**serverRoot**/rhq-data/	分散ストレージデータベースノートのデータ、キャッシュファイル、およびバックアップの親ディレクトリー。
データファイル	**serverRoot**/rhq-data/commit_log/	ストレージデータベースに書き込まれる前に書き込み操作を保存するバイナリーコミットログが含まれます。
データファイル	**serverRoot**/rhq-data/saved_caches/	**data**/ サブディレクトリー内の各データベースのデータベースキャッシュファイルが含まれます。

設定エリア	ディレクトリーまたはファイルの場所	description
データファイル	**serverRoot**/rhq-data/data/	ノードのデータファイルとスナップショットが含まれます。キースペースとテーブル別に整理されます。には **rhq**、、**system**、およびの 4 つのキースペース **system_auth**があります **system_traces**。
データファイル	**serverRoot**/rhq-data/data/rhq/	すべてのメトリクステーブルのデータが含まれます。これは、各レベルのアグリゲーション（1 時間、6 時間、および 24 時間）のすべての新規(raw)メトリクスおよびテーブルです。これには、収集されるメトリクスのバージョンのスキーマおよびインデックス情報も含まれます。
データファイル	**serverRoot**/rhq-data/data/system/	分散データベースノードに関する情報が含まれます。
データファイル	**serverRoot**/rhq-data/data/system_auth/	分散データベースに設定された認証情報、ユーザー、およびパーミッションに関する情報が含まれます。
データファイル	**serverRoot**/rhq-data/data/system_traces/	データベースセッションおよびイベントに関する情報が含まれています。

バグを報告します。

2.2. デフォルトのサーバー、エージェント、およびストレージノードポート

他のサーバーおよびサービスと同様に、JBoss ON サーバーとエージェントはシステムポート経由で接続して相互に通信します。JBoss ON は、複数の異なる接続にポートを使用します。

サーバーからデータベースへの通信

サーバーがそのデータベースに接続できる必要があります。データベースのポート番号は、データベースのタイプとデータベースの特定の設定により異なります。

サーバーからエージェント通信

サーバーは、エージェントに設定された 1 つのポートでエージェントに接続します。このポートは、サーバーとエージェント間の標準および SSL 通信の両方に使用されます。

エージェント対サーバー通信

同じポートを使用している場合でも、エージェントは複数の JBoss ON サーバーと通信できます（各サーバーが別のホストにあるため）。 エージェントは、設定された接続(transport)メソッドに

応じて標準ポートまたは SSL ポートを使用して JBoss ON サーバーに接続します。エージェントは単一のポートの使用のみを試行します。

サーバーからストレージノードへの通信

JBoss ON サーバーはエージェントから監視データを受け取り、その情報を利用可能なストレージノードに転送します。サーバーは、クライアント(CQL)ポートでストレージノードに接続します。

ストレージノードからストレージノードの通信

データの整合性と可用性を確保するために、監視データはストレージノード間で定期的に同期されます。ノードは、**ゴシップ** ポートと呼ばれるクラスターのポートでデータを相互に送信します。

注記

サーバーは直接相互に対話しないため、サーバー間のリンク用のポートはありません。

JBoss ON 接続のデフォルトポート番号はに記載されてい 表2.4「デフォルトのポート」 ます。JBoss ON サービスまたは異なる値は、任意の JBoss ON サービスに対して変更できます。

表2.4 デフォルトのポート

接続タイプ	ポート番号
Server to agent	16163
エージェント対サーバー（標準）	7080
エージェント対サーバー（セキュア）	7443
Server to database	5432（デフォルトの PostgreSQL ポート）
Server to storage node	9142
ストレージノードからストレージノードへ	7100

バグを報告します。

2.3. RHQCTL コントロールスクリプト

JBoss Operations Network には、サーバーおよびストレージノードの基本的なライフサイクル管理に使用される制御スクリプトがあります。サーバーコンソールを開き、インスタンスの起動および停止、およびコンポーネントのインストールまたはアップグレードを行うことができます。

rhqctl スクリプトには、サブコマンドとオプションがあります。

```
rhqctl [command] [[options]
```

オプションは通常、、、、またはにタスクを実行するローカルインスタンスを指定 **--server --agent**し **--storage**ます。

例2.1 特定のサービスのインストール

この **install** コマンドは、JBoss ON のサーバー、ストレージノード、およびエージェントを同時に設定します。

3 つの管理サービスをすべて同じシステム（同じ親ディレクトリーから）で実行することが推奨されますが、ストレージノードとは別のマシンで JBoss ON サーバーを実行するのに有益な環境があります。その他の場合は、異なるタイミングで異なるサービスをインストールする必要がある場合があります。

install コマンドには、サービスごとにオプションがあります。このオプションを使用すると、そのサービスだけがインストールされ、他のサービスは除外されます。

たとえば、これにより、サーバー、ストレージノード、およびエージェントが 3 つの異なるコマンド呼び出しにインストールされます。

```
[root@server bin]# ./rhqctl install --storage --start
[root@server bin]# ./rhqctl install --server --start
[root@server bin]# ./rhqctl install --agent --start
```

サーバーまたはノードのインストール後には、**rhqctl** コマンドを使用してインスタンスを起動および停止します。したがって、start/stop コマンドは最も一般的なコマンドです。オプションを指定せずにコマンドを実行すると、そのコマンドはシステム上のすべての JBoss ON インスタンスに対して実行されます。

```
[root@server bin]# ./rhqctl start
```

オプションを使用すると、そのインスタンスにのみそのコマンドが実行されます。

```
[root@server bin]# ./rhqctl start --server --agent
```

表2.5 rhqctl コマンド

コマンド	description
コンソール	サービスとしてではなく、フォアグラウンドでサーバーまたはストレージノードを起動します。
install	サーバー、ストレージノード、またはエージェントインスタンスをインストールします。
restart	JBoss ON サービスを再起動します。サービスの種別が指定されていない場合は、すべてのローカルサービスが再起動されます。
start	JBoss ON サービスを開始します。サービスの種別が指定されていない場合は、すべてのローカルサービスが開始されます。

コマンド	description
status	JBoss ON サービスのステータスを表示します。サービスの種別が指定されていない場合は、すべてのローカルサービスのステータスが表示されます。
stop	JBoss ON サービスを停止します。サービスの種別が指定されていない場合は、ローカルサービスがすべて停止します。
アップグレード	すべての JBoss ON サービスをアップグレードします。これは、サーバーの移行後に過去のメトリックデータを移行するために再実行することができます。

表2.6 rhqctl オプション

オプション	description	許可するコマンド
--start	すべてのサービスを起動します。	● アップグレード ● install
--server	ローカルサーバーの指定コマンドを実行します。このパラメーターを指定すると、コマンドはサーバーに対してのみ実行され、他のコンポーネントは無視されます（明示的に指定されていない限り）。	● コンソール ● restart ● start ● stop ● status ● install
--agent	エージェントに対して指定されたコマンドを実行します。このパラメーターを指定すると、コマンドはエージェントに対してのみ実行され、他のコンポーネントは無視されます（明示的に指定されていない限り）。	● restart ● start ● stop ● status ● install

オプション	description	許可するコマンド
--storage	ストレージデータベースノードの指定コマンドを実行します。これが指定されている場合、コマンドはストレージデータベースノードに対してのみ実行され、その他のコンポーネントは明示的に指定されていない限り実行されます。	• コンソール • restart • start • stop • status • install
--storage-data-root-dir **directory**	ストレージデータが保存されるディレクトリーを変更します。デフォルトでは、ストレージノードのディレクトリーはになり ***serverRoot*/jon-server-3.3.0.GA/rhq-data**/ます。	• install
--from-server-dir **directory**	アップグレードするサーバーへのディレクトリーパスを指定します。	• アップグレード
--from-agent-dir **directory**	アップグレードするエージェントへのディレクトリーパスを指定します。アップグレードスクリプトは、というサブディレクトリーに、サーバーと同じディレクトリーにエージェントがインストールされていることを前提としてい**rhq-agent**/ます。このオプションは、エージェントの場所が異なる場合に必要です。	• アップグレード
--run-data-migrator	既存の JBoss ON SQL データベースから新しい監視ストレージデータベースにデータを移行します。upgrade コマンドは、監視データベースを作成します。このオプションを使用しないと、以前のモニタリングデータは失われます。	• アップグレード

バグを報告します。

2.4. JBOSS ON サーバーの起動

JBoss ON サーバーは JBoss ON インストールに含まれるカスタマイズされた JBoss AS サーバーであるため、JBoss ON サーバーを起動すると JBoss インスタンスが起動します。

JBoss ON サーバーは手作業で起動することも、システムサービスとして起動および実行するように設定することもできます。

バグを報告します。

2.4.1. JBoss ON Server(Basic)の起動

JBoss ON サーバープロセスは、serverRoot/**bin**/ ディレクトリーで **rhqctl** スクリプトを実行して開始します。

重要

JBoss ON サーバーは、コマンドラインまたは UI のリソース操作（他の EAP リソースとは異なり） **から** 起動または再起動できません。あるサーバーの JBoss ON UI または CLI を使用して、クラウドで別の JBoss ON サーバーを起動できますが、起動操作または再起動操作を実行できません。

サーバーを起動する最も簡単な方法は、**start** コマンドでスクリプトを実行することです。これにより、サーバープロセスが開始され、スクリプトから終了します。

```
serverRoot/bin/rhqctl start --server
Trying to start the RHQ Server...
RHQ Server (pid 27547) is starting
```

rhqctl スクリプトは実行中、JBoss EAP サーバーインスタンスで使用する JVM に関連する特定の環境変数を検索します。環境変数の完全なリストはスクリプト自体で指定されています。インストール情報に基づくデフォルトが使用されるため、ほとんどの環境変数をリセットする必要はありません。

注記

The **RHQ_JAVA_HOME** サーバーを起動するには、Red Hat Enterprise Linux システムで環境変数を設定する必要があります。これは、のように一般的な値に設定でき **/usr**/ます。

サーバーをコンソールモードで起動することもできます。これは、サーバープロセスに関する詳細情報をターミナルにプリントし、サーバーが実行されている限り、スクリプトを開いたままにします。

```
[root@server ~]# ./rhqctl console --server
20:28:44,120 INFO  [org.jboss.modules] JBoss Modules version 1.2.2.Final-redhat-1
Starting RHQ Server in console...
JAVA_OPTS already set in environment; overriding default settings with values: -
....
```

バグを報告します。

2.4.2. JBoss ON Server をサービスとして実行

以下の例を使用して、JBoss ON サーバー、ストレージノード、およびエージェントを Red Hat Enterprise Linux で SysV スタイルの init サービスとして管理し、実行できます。

1. initalization scipt を作成し **/etc/init.d/jboss-on**ます。

```
#!/bin/sh
```

```
#
# Example Red Hat JBoss Operations Network init script
#
# chkconfig: - 95 20
# description: Red Hat JBoss Operations Network rhqctl services
# processname: standalone.sh
#
#
# This is an example script that can be used with a SysV style
# init service which starts a JBoss ON server, storage node, and
# agent using the rhqctl script provided by the JBoss ON server
# installation.

prog="`basename "$0"`"

# If available, the following files will be sourced, in order, for
# configuration. This means that if the same value is defined in a later
# file, it will override the one specified in an earlier one:
#
#   /etc/jboss-on.conf
#   /etc/jboss-on/${prog}.conf
#
# The following environment variables can be defined:
#
#   RHQ_HOME -- Defaults to /opt/jboss/jboss-on
#   RHQ_SERVER_HOME -- Defaults to RHQ_HOME/jon-server-*
#   RHQ_AGENT_HOME -- Defaults to RHQ_HOME/rhq-agent
#   RHQ_JAVA_HOME -- Defaults to /usr
#   RHQ_USER -- Defaults to jonadmin
#

# If available, source environment variables for the service
# Global defaults/settings
[ -r "/etc/jboss-on.conf" ] && . "/etc/jboss-on.conf"
# Settings specific to this service
[ -r "/etc/jboss-on/${prog}.conf" ] && . "/etc/jboss-on/${prog}.conf"

# If not yet set, use some reasonable defaults:
[ -z "${RHQ_HOME}" ] && RHQ_HOME='/opt/jboss/jboss-on'
[ -z "${RHQ_SERVER_HOME}" ] && RHQ_SERVER_HOME="`eval echo
"${RHQ_HOME}/jon-server-"*`"
[ -z "${RHQ_AGENT_HOME}" ] && RHQ_AGENT_HOME="${RHQ_HOME}"'/rhq-agent'
[ -z "${RHQ_JAVA_HOME}" ] && RHQ_JAVA_HOME='/usr'
[ -z "${RHQ_USER}" ] && RHQ_USER=jonadmin

# We have to export these variables so that they are available to child
# processes.
export RHQ_SERVER_HOME
export RHQ_AGENT_HOME
export RHQ_JAVA_HOME


# This is a convenient function that invokes rhqctl
rhqctl() {
 RHQCTL_SCRIPT="${RHQ_SERVER_HOME}/bin/rhqctl"
 [ ! -x "${RHQCTL_SCRIPT}" ] && {
```

```
  echo >&2 "${RHQCTL_SCRIPT}'" was not found or is not executable.'
  exit 2
 }
 if [ -n "${RHQ_USER}" -a "$(whoami)" != "${RHQ_USER}" ]; then
  CMD="su -m ${RHQ_USER} -c '\"${RHQCTL_SCRIPT}\" $@'"
 else
  CMD="\"${RHQCTL_SCRIPT}\" $@"
 fi
 eval ${CMD}
}

# We invoke rhqctl differently if start or stop is used so we need this
# case statement to determine what should happen.
case "$1" in
start)
 # Because rhqctl is so chatty/verbose we redirect standard output to
 # null so we do not see it during system start and stop.
 # Additionally, we invoke it in the background as it can block for
 # some time during start.
 rhqctl "$@" >/dev/null &
 ;;
stop)
 # Because rhqctl is so chatty/verbose we redirect standard output to
 # null so we do not see it during system start and stop.
 rhqctl "$@" >/dev/null
 ;;
*)
 # Assuming we are not starting or stopping, we will display complete
 # output.
 rhqctl "$@"
 ;;
esac
```

2. ファイルに対する権限を変更し、以下を実行します。

```
sudo chmod +x /etc/init.d/jboss-on
```

3. initalization スクリプトを chkconfig に追加します。

```
sudo chkconfig --add /etc/init.d/jboss-on
```

4. スクリプトがシステムの起動およびシャットダウン時に停止することを意図していることを示します。

```
sudo chkconfig jboss-on on
```

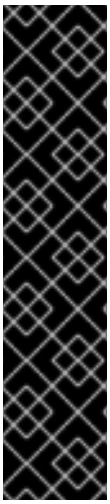
5. 、、**RHQ_HOME**、およびのデフォルト値を上書きするには **RHQ_SERVER_HOME**、、**RHQ_USER**、**RHQ_AGENT_HOME RHQ_JAVA_HOME** および、**/etc/jboss-on.conf** または特定の値 **/etc/jboss-on/<SERVICE NAME>.conf** でサービス設定ファイルを作成します。

```
RHQ_HOME=/usr/share/jboss-on
RHQ_USER=jbosson
```

バグを報告します。

2.5. JBOSS ON エージェントの起動

JBoss ON エージェントは手作業で起動することも、システムサービスとして起動して実行するように設定できます。



重要

エージェントの設定は、ユーザーがエージェントを実行しているユーザーによって決まります。エージェントが 1 人のユーザーとして実行され、その後別のユーザーとして実行される場合、設定に異なるバッキングストアを使用するため、エージェントは 2 回目とは異なる設定を持ちます。

つまり、あるユーザーがエージェントのインストール時にエージェントを設定するのに使用する場合、同じユーザーを使用してエージェントを後に実行する必要があります。そうしないと、エージェントは設定を失い、新しいユーザーで再設定する必要があります。

エージェント設定バッキングストアについては、を参照してください 「エージェントの永続設定の管理」。

バグを報告します。

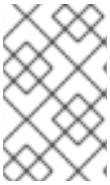
2.5.1. 管理システムでの JBoss ON エージェントの起動

エージェントが起動し、エージェントの **bin**/ ディレクトリー内のスクリプトを使用して実行されます。サーバー起動スクリプト（サーバープロセスが開始され、スクリプトが終了し、エージェントスクリプトは開いたままになります）とは異なり、必要に応じてさらに入力を受け入れるプロンプトが表示されます。（通常、スクリプトを開始してバックグラウンドで実行できます。）

```
/opt/rhq-agent/bin/rhq-agent.sh

RHQ 3.3.0-SNAPSHOT [cda7569] (Tue Apr 13 13:39:16 EDT 2017)
>
```

ほとんどの場合、JBoss ON エージェントは追加のオプションや設定なしで実行できます。**rhq-agent.sh** スクリプトで使用できるすべてのオプションは、に記載されてい 「エージェントプロンプトコマンド」 ます。追加の設定オプションは、**rhq-agent-env.sh** スクリプトファイルを編集して設定できます。



注記

JBoss ON エージェントの起動時にエラーが発生した場合は、でエージェント起動スクリプトを実行して以前のエージェント設定 **--cleanconfig** を消去し、新規の起動を開始します。

バグを報告します。

2.5.2. サーバーマシンにインストールされているエージェントの起動

エージェントがサーバーまたはストレージノードと同じシステムにインストールされている場合、エージェントはサーバーとストレージノードと同じ制御スクリプト(**rhqctl**)で制御されます。エージェントを起動するには、スクリプトで **--agent** オプションを使用します。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --agent
```

これにより、エージェントコンソールを開くのではなく、バックグラウンドでエージェントが起動します。

デフォルトでは、エージェントの init スクリプトはローカルシステムアカウント（デフォルト**または .\LocalSystem**）として実行されます。の説明に従って、**rhq-agent-env.bat** スクリプトを編集して ssytem ユーザーアカウントを変更でき「エージェントを Windows サービスとして設定」 ます。サーバーマシンのエージェントは、**rhqctl** コマンドを使用して起動し、停止します。

バグを報告します。

2.5.3. エージェントを Windows サービスとして設定

1. **rhq-agent-env.bat** スクリプトを編集し、init スクリプトが実行されるようにシステムユーザーを定義する環境変数を設定します。以下の 2 つのオプションがあります。

   - **RHQ_AGENT_RUN_AS** ユーザーアカウント名を明示的に設定します。これは、Windows ユーザーアカウント名 **DOMAIN\username** の形式と一致する必要があります。

   - **RHQ_AGENT_RUN_AS_ME** エージェントを現在のユーザーとして実行するように強制します。\%USERNAME% 形式を使用します。両方の環境変数が定義されている場合、この変数は上書きされます。 **RHQ_AGENT_RUN_AS**.

   注記

   設定前 **RHQ_AGENT_RUN_AS_ME** または **RHQ_AGENT_RUN_AS**指定のユーザーに、サービスを開始するパーミッションがあることを確認します。必要に応じて、適切な権限をユーザーに割り当てます。割り当て権限は、Windows ドキュメントで説明されています。

   変数が設定されていない場合、エージェントの init スクリプトがシステムユーザーとして実行されます。

   その他の利用可能な環境変数は、**rhq-agent-wrapper.bat** スクリプトのコメントに一覧表示され、定義されます。

2. **rhq-agent-wrapper.bat** スクリプトを実行して init スクリプトをサービスとしてインストールします。**install** コマンドを使用して init スクリプトをインストールします。

3. プロンプトが表示されたら、サービスが実行されるシステムユーザーのパスワードを入力します。

エージェントサービスは、Windows システムの起動時に自動的に起動します。このサービスは、Windows サービス管理ツールから起動または停止できます。

また、およびコマンドを使用して、**rhq-agent-wrapper.bat** スクリプトからエージェントサービスを起動 **start** および停止することもでき **stop** ます。この **status** コマンドは、エージェントの init スクリプトがサービスとしてインストールされているかどうか、および実行しているかどうかを表示します。この **remove** コマンドは、エージェント init スクリプトをサービスとして削除します。

JBoss ON エージェント Windows スクリプトは Java Wrapper Service を使用してサービスを制御します。設定ファイル agentRoot には、**\bin\wrapper\rhq-agent-wrapper.conf**サービス設定プロパティーが含まれます。これらは標準のラッパーサービスプロパティーです。詳細は http://wrapper.tanukisoftware.org/doc/english/properties.html を参照してください。

サービスをカスタマイズする一般的なプロパティーは複数あります。

- ***wrapper.app.parameter.#*** エージェントに渡すコマンドラインオプションを設定します。これらのオプションは、に記載されているオプションと同じです 「エージェントのコマンドプロンプトの使用」。各オプションには独自の設定プロパティーが必要です。プロパティーは数値順に配置し、最初の 2 つのプロパティー（***wrapper.app.parameter.1*** および ***wrapper.app.parameter.2***）が予約される必要があります。で始まり ***wrapper.app.parameter.3***ます。

- ***wrapper.java.additional.#*** **-D** およびオプションに対応する、仮想マシンに直接渡される追加の JVM **-X** オプションを設定します。数値にインクリメントする必要もあります。***wrapper.java.additional.1*** 常にログファイルファイルを指定します。

- ***wrapper.ntservice.starttype*** サービスを開始するタイミングを設定します。デフォルトはで **AUTO_START**、システムの起動時にサービスを起動します。サービスを手動で起動するには、の値はになり **DEMAND_START**ます。

バグを報告します。

2.5.4. エージェントをデーモンまたは init.d サービスとして実行

重要

エージェントは、サービスとして起動する際に設定を求めるプロンプトを出しません。エージェントは事前に設定されたか、または一度起動して、設定が入力されている必要があります。どちらのオプションも 『『インストールガイド』』 で説明されています。

エージェントの設定は、ユーザーがエージェントを実行しているユーザーによって決まります。エージェントが 1 人のユーザーとして実行され、その後別のユーザーとして実行される場合、設定に異なるバッキングストアを使用するため、エージェントは 2 回目とは異なる設定を持ちます。

つまり、あるユーザーがエージェントのインストール時にエージェントを設定するのに使用する場合、同じユーザーを使用してエージェントを後に実行する必要があります。そうしないと、エージェントは設定を失い、新しいユーザーで再設定する必要があります。

エージェント設定バッキングストアについては、を参照してください 「エージェントの永続設定の管理」。

エージェントを設定（または事前設定）すると、エージェントは 2 つの方法で起動できます。**rhq-agent.sh** スクリプトはエージェントを起動し、コマンドコンソールを開きます。**rhq-agent-wrapper.sh** スクリプトは、エージェントデーモンを起動して終了します。どちらの方法でも、**rhq-agent-env.sh** スクリプトファイルを使用して追加の環境変数を設定できます。

デーモンは、システムサービスとして起動して実行できます。Red Hat Enterprise Linux では、これを設定するには、を使用 **/etc/init.d** してインストールし **chkconfig**ます。Solaris およびその他の Unix システムでは、他のシステムツールを使用してサービスを設定し **/etc/init.d**、その他の Unix システムツールを使用してサービスを設定することで行われます。

1. エージェントが完全にセットアップされていることを確認します。

2. **rhq-agent-env.sh** ファイルを開きます。

3. エージェントのディレクトリー、JDK ディレクトリー、および PID **bin** ディレクトリーに必要な環境変数のコメントを解除し、設定します（エージェントユーザーが書き込み可能でなければなりません）。

```
RHQ_AGENT_HOME=agentRoot/rhq-agent/bin/
export RHQ_JAVA_HOME=/usr
PIDFILEDIR=/var/run
```

注記

の設定時 **PIDFILEDIR** Red Hat Enterprise Linux で、 **rhq-agent-wrapper.sh** スクリプトファイルの ***pidfile*** 設定を編集します。ラッパースクリプトの値はにより使用され **chkconfig**ます。

4. オプションの環境変数のいずれかを設定します。
   - **RHQ_AGENT_DEBUG** デバッグロギングを有効にします。
   - **RHQ_AGENT_JAVA_EXE_FILE_PATH** Java 実行ファイルを指定します。
   - **RHQ_AGENT_JAVA_OPTS** 設定をエージェント JVM に渡します。
   - **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** 追加の Java オプションを JVM に渡します。
5. root でシステムにログインします。

重要

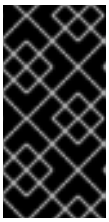
残りの手順では、Red Hat Enterprise Linux でエージェント init スクリプトをサービスとして設定する方法を説明します。その他の Unix システムについては、特定のプラットフォームに対応する同様の手順に従います。

6. ラッパースクリプトが実行可能であることを確認します。

```
[root@server rhq-agent]# chmod a+x agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh
```

7. **rhq-agent-wrapper.sh** ファイルのシンボリックリンクです **/etc/init.d/**。例：

```
# ln -s agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh /etc/init.d/rhq-agent-wrapper.sh
```



重要

Solaris では、エージェントスクリプトファイルのシンボリックリンクでは、起動が必要 **rhq-agent-wrapper.sh**です。一部の Solaris インストール ***readlink*** で ***readlink*** は、デフォルトでは提供されません。Solaris ユーザーは、Sunfreeware などのソース ***readlink*** からダウンロードする必要があります。

8. **rhq-agent-wrapper.sh** に登録し **chkconfig**ます。

```
# /sbin/chkconfig --add rhq-agent-wrapper.sh
```

9. システムの起動時にエージェントサービスを実行でき、システムのシャットダウン時に正常に停止できるようにします。

```
# /sbin/chkconfig rhq-agent-wrapper.sh on
```

システムの起動時にエージェントサービスを起動しない場合は、スクリプトをオフに **chkconfig** します。

```
# /sbin/chkconfig rhq-agent-wrapper.sh off
```

バグを報告します。

2.5.5. カスタムコマンドを使用したエージェントの起動

エージェントをサービスとして実行するように設定すると「エージェントをデーモンまたは init.d サービスとして実行」、エージェントはカスタム start コマンドで開始するように設定できます。これは主に **sudo**、**su** またはを使用してエージェントを起動し、エージェントを別のユーザーとして実行できるようにするために使用されます。

start コマンドは、**rhq-agent-env.sh** ファイルの他のエージェントプロパティーで定義されます。設定には、start コマンド自体とパスワードプロンプトを有効にする設定の 2 つの部分があります。

```
RHQ_AGENT_START_COMMAND="su -m test -c '${RHQ_AGENT_HOME}/bin/rhq-agent.sh'"
RHQ_AGENT_PASSWORD_PROMPT=true
```

start コマンドを設定すると、エージェントを起動するためにコマンドラインで渡されるコマンドが上書きされます。

パスワードプロンプトは必要ありません。設定は **sudo**、およびエージェントのユーザー設定によって異なります。必要な場合は、ユーザーがパスワードを入力するか、またはパスワードを **RHQ_AGENT_PASSWORD** パラメーターに設定できるようにパスワードプロンプトを有効にする必要があります。それ以外の場合は、開始プロセスがハングします。

定義した start コマンドが無効な場合、エージェントは起動できません。コマンドのフォーマットに加えて、ディレクトリー名は影響を受けます。名前にスペースや特殊文字がある場合は、コマンドの実行をエスケープする必要があります。

バグを報告します。

2.5.6. エージェントおよび JVM の再起動

エージェントの JVM プロセスをダウンせずにエージェントを再起動できます。また、エージェントとその JVM を再起動することもできます。

エージェントは、JBoss ON サーバーによって管理されるプラグインコンテナーを使用して管理されます。コンテナーは、すべてのエージェントのライフサイクルを読み込み、管理します。プラグインコンテナーを再起動すると、JVM を破棄せずにエージェントとそのコンポーネントを再起動します。

1. 上部のナビゲーションバーで **Resources** メニューを選択し、**Servers** メニュー項目を選択します。

2. 一覧でエージェントリソースをクリックします。

3. **Operations** タブをクリックします。

4. **Restart** タスクを選択し、起動します。

エージェントとその JVM の両方を再起動できます（例： ランチャースクリプトや JVM オプションが編集されている場合などに便利です）。

1. 上部のナビゲーションバーで **Resources** メニューを選択し、**Servers** メニュー項目を選択します。
2. 一覧でエージェントリソースをクリックします。
3. エージェントの下にあるランチャースクリプト子リソースに移動します。
4. ランチャースクリプトリソースの **Operations** タブをクリックします。
5. **Restart** タスクを選択し、起動します。

バグを報告します。

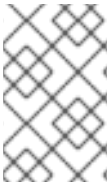
2.6. ストレージノードの起動

JBoss ON のストレージノードプロセスは、JBoss ON サーバーの serverRoot **ディレクトリー/bin/** で **rhqctl** スクリプトを実行して開始します。

ストレージノードを起動する最も簡単な方法は、**start** コマンドでスクリプトを実行することです。これにより、ノードプロセスが開始され、スクリプトから終了します。

```
[root@server bin]# ./rhqctl start storage
22:03:01,362 INFO  [org.jboss.modules] JBoss Modules version 1.2.2.Final-redhat-1
 INFO 22:03:03,138 Logging initialized
 INFO 22:03:03,177 JVM vendor/version: OpenJDK 64-Bit Server VM/1.6.0_24
...
 INFO 22:03:51,215 Node /10.16.46.34 state jump to normal
 INFO 22:03:51,223 Startup completed! Now serving reads.
```

rhqctl スクリプトは、その実行中、特にストレージノードの JVM に関連する特定の環境変数を検索します。Java 設定の完全なリストは **/opt/jon/jon-server-3.3.0.GA/rhq-storage/conf/cassandra-jvm.properties** ファイルに記載されています。



注記

The **RHQ_JAVA_HOME** サーバーを起動するには、Red Hat Enterprise Linux システムで環境変数を設定する必要があります。これは、のように一般的な値に設定でき **/usr/**ます。

ストレージノードはコンソールモードで起動することもできます。これにより、ストレージノードプロセスに関する詳細情報がターミナルに出力され、ノードが実行している限りスクリプトが開いたままになります。

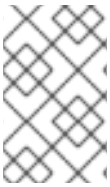
```
[root@server ~]# ./rhqctl console --server
20:28:44,120 INFO  [org.jboss.modules] JBoss Modules version 1.2.2.Final-redhat-1
Starting RHQ Storage in console...
JAVA_OPTS already set in environment; overriding default settings with values: -
....
```

バグを報告します。

第3章 パフォーマンスを向上するための JBOSS ON の調整

JBoss Operations Network デプロイメントはすべて一意で、管理される環境、インベントリー内のリソース数およびタイプ、個別のリソースの設定、メトリクスコレクションやアラート定義などの JBoss ON 機能の設定により異なります。環境におけるこれらの違いは、JBoss ON のサーバー、エージェント、およびバックエンドデータベースのパフォーマンスが異なります。これにより、「ユニバーサル」のベースラインや期待されるパフォーマンスの定義が困難になる可能性があります。

しかし（各デプロイメントの制限あり）、JBoss ON 設定の一般的な動作特性と潜在的なチューニングを特定でき、サーバーとエージェントのパフォーマンスを向上させることができます。



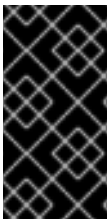
注記

PostgreSQL または Oracle データベースを調整してパフォーマンスを向上させることができます。これは、JBoss ON 管理の範囲外です。データベースのチューニング情報は、該当する製品ドキュメントを参照してください。

バグを報告します。

3.1. インベントリーのベースライン

インベントリーベースラインは、**一度にインベントリーにインポートできるリソースの数に基づいて** 決定されました。エージェントでは、すべてのリソースが同時に検出およびインポートされると、エージェントが最初に並行してインストールされます。サーバーでは、既存のエージェントが多数のインベントリーに追加されると、アップグレード操作が並行して実行されます。



重要

これらのベースラインは、テスト環境での汎用 JBoss ON デプロイメントと、その環境を反映した情報です。これは参照用です。これは推奨事項として意図されていません。これは、異なるハードウェア、リソース、およびその他の要素を持つ実際の JBoss ON デプロイメントを反映しない可能性があります。

サーバーベースライン

サーバーは通常、メモリー不足のエラーに遭遇する前に、100 以上のエージェントでアップグレードできます。これらのエラーが発生した場合は、で説明しているようにサーバー設定を調整 「多数のエージェントのサーバーチューニング」 して、スレッドプールと同時実行制限を増やし、より多くのリクエストの処理を可能にします。

エージェントインポートのベースライン

システムのリソースは、2 つの異なる方法で編成できます。1 つの方法は、階層のレベルが少なく、各レベルが非常に幅広い階層であるフラット階層です。基本的には、多くの子リソースを持つ親リソースはほとんどありません。

```
                platform
                   |
  ------------------------------------------------
  |      |      |      |      |    |
server1  server2  server3  server4  server5  ...
  |
```

```
    -------------------------------------------------------
   |         |         |         |         |     |
service1   service2   service3   service4   service5   ...
```

階層が非常に深い場合、親リソースには直接子はほとんどありませんが、子サービスのレベルはあります。これは、EAP 6 や多くのサブシステムとサービスを持つ同様のアプリケーションでは特に一般的です。

```
        platform
          |
  ----------------------
  |        |        |
server1   server2   server3
  |
  -------------
  |         |
service1   service2
   |
   ------------
   |         |
serviceA   serviceB
   |
   ------------
   |         |
serviceI   serviceII
   |
   ------------
   |         |
serviceX   serviceY
   |
   ---
   |
service...
```

リソースの階層は、親リソースと子リソースを再帰的にどのように再帰するかにより、大きな記述をインポートする際にエージェントがどのように実行するかに影響します。

表3.1 エージェントインポートのパフォーマンス

リソースの階層	テスト済みインポート時間	注記
階層： 10 トップレベルのサーバー、10 個の中間サーバー、750 子サービス(10x10x750) （合計リソースは 75,000）	2 時間 46 分	エージェントのヒープサイズを 2GB に増やすと、インポート時間が 2 時間 34 分に減ります。また、通常のエージェントメモリー設定で多数のリソースをインポートする際に発生する可能性のあるメモリー不足のリスクも軽減されました。
フラット階層： 100 トップレベルのサーバー、750 子サービス (10x10x750) （合計リソースは 75,000）	2 時間 14 分	

ガベージコレクションの後、インポート操作中のメモリーフットプリントは平均で 665.3MB でした。インポートが完了すると、メモリーフットプリントは 586.2MB になります。

エージェントおよび EAP 6 リソースベースライン

単一エージェントは、40 から 50 個の JBoss Enterprise Application Platform 6 インスタンスを管理できます。

注記

単一のエージェントで管理できる EAP 6 リソースの数は、基盤システムのハードウェア、各 EAP インスタンスが管理する web コンテキストの数、Web コンテキストのサイズ、その他の要素に大きく依存します。

バグを報告します。

3.2. 監視およびアラートベースライン

理論的には、収集できるメトリクスの数や発生する可能性のあるアラートの数に制限はありません。

実際には、IT 環境には、監視とアラート設定の両方を制限するという制約があります。

- データベースのパフォーマンス（ほとんどの環境では主要な要素）
- ネットワーク帯域幅

JBoss ON のアラートおよびモニタリング設定はリソースの数、メトリクス数、収集頻度、アラート数により異なるため、JBoss ON のアラートおよびモニタリング設定にはハード制限はありません。

経験上、パフォーマンスのしきい値は以下のとおりです。

- 最大メトリクスを毎分収集できます。
- 最大 9000 個のアラートを 1 日あたり最大 4000 に発生する（毎分約 70 個）

メトリクスの収集およびアラートの実装方法を計画します。メトリクススケジュールを有効にし、コレクションの頻度を設定する際に、リソースの優先順位を決定し、それらのリソースから必要な情報を設定します。次に、これらの優先順位に基づいて必要なアラートを計画します。

監視およびアラートのクリアストラテジーは、重要な情報を収集している最中にパフォーマンスを維持するのに役立ちます。

注記

収集したメトリクスの量を拡張するために、追加のストレージノードをストレージクラスターに追加できます。

ストレージノードは、JBoss ON の設計の一部として永続的に行うか、既存のノード上のアラート状態を使用して新規ノードのデプロイメントをトリガーして動的に作成できます。

追加ノードの作成については、を参照してください 9章*ストレージノードのデプロイおよび管理*。

バグを報告します。

3.3. エージェント JVM メモリーサイズの調整

エージェントが多数のリソースを管理する場合、JVM のデフォルト設定でメモリーが不足します。これにより、**メモリーがしきい値を超え、エージェントログに記録されないようなエラーが発生する可能性** があり、エージェントは自動的に再起動されます。通常、これはエージェントのヒープサイズが低く設定されているため発生しますが、perm gen の値が低い場合にも関連します。

エージェントのメモリー設定を変更する場合は、**rhq-agent-env.sh** ファイル **RHQ_AGENT_JAVA_OPTS** で適切な JVM 設定を設定します。

1. エージェントを停止します。たとえば、エージェントがサービスとして実行している場合は、以下を実行します。

```
[root@server ~]# service rhq-agent-wrapper.sh stop
```

2. この **rhq-agent-env.sh** ファイルを開き、エージェントが使用する必要な環境変数を設定します。

```
[root@server ~]# vim agentRoot/rhq-agent/bin/rhq-agent-env.sh
```

3. エージェント JVM のヒープサイズの最小および最大範囲のおよび **-Xms -Xmx** パラメーターで **RHQ_AGENT_JAVA_OPTS** 値を設定します。例：

```
RHQ_AGENT_JAVA_OPTS="-Xms1024m -Xmx1024m -XX:PermSize=256M -
XX:MaxPermSize=256M -Djava.net.preferIPv4Stack=true"
```

4. 必要に応じて、**-XX:PermSize** および **-XX:MaxPermSize** を使用して perm gen サイズを設定します。

5. エージェントプロセスを再起動して、新しい設定を読み込みます。たとえば、エージェントがサービスとして実行している場合は、以下を実行します。

```
[root@server ~]# service rhq-agent-wrapper.sh start
```

バグを報告します。

3.4. エージェント JVM ヘルスチェック設定の変更

エージェントに含まれる複数の典型的な問題の 1 つにメモリーエラーがないことがあります。エージェント JVM ヒープのサイジングは困難です。エージェントはシステムリソースの過剰消費を実行できず、システム全体のパフォーマンスに影響を及ぼす可能性がありますが、メトリックの収集やドリフトの監視などの集中的なタスクを実行するのに十分なメモリーが必要です。

JVM ヘルスチェックは JVM を定期的にスキャンし、メモリーしきい値を確認します。利用可能なメモリーが指定の時点を下回る場合、JVM プロセスは再起動します。これにより、エージェントがメモリー不足の状態を防ぎ、不明な状態でシャットダウンできます。

注記

JVM のヘルスチェックは利点がありますが、利用できません。メモリーがスキャン間隔よりも速く使い切られる場合があります。

エージェントのヘルスチェックには、以下の 3 つのパラメーターがあります。

- スキャンの間隔。 **rhq.agent.vm-health-check.interval-msecs**
- ヒープしきい値 **rhq.agent.vm-health-check.low-heap-mem-threshold**
- heap 以外のメモリーしきい値 **rhq.agent.vm-health-check.low-nonheap-mem-threshold**

ヘルスチェックの設定を変更するには、以下を実行します。

1. エージェントプロンプトを開きます。**-n** オプションを使用してプロンプトを開かずにプロンプトを開きます。

```
[root@server ~]# agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. 編集する希望の名前とその新しい値 **setconfig** で送信します。

```
> setconfig rhq.agent.vm-health-check.interval-msecs=3000
Set preference: rhq.agent.vm-health-check.interval-msecs=3000
> setconfig rhq.agent.vm-health-check.low-heap-mem-threshold=1.1
Set preference: rhq.agent.vm-health-check.low-heap-mem-threshold=1.1
> setconfig rhq.agent.vm-health-check.low-nonheap-mem-threshold=1.1
Set preference: rhq.agent.vm-health-check.low-nonheap-mem-threshold=1.1
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --agent
```

バグを報告します。

3.5. サーバー JVM の調整

デフォルトでは、JBoss ON サーバーは、モードに割り当てられたヒープサイズと、*nix システム制限 (1024)を並行する設定済みのスレッド制限で実行されます。しかし、JBoss ON サーバーが PostgreSQL などの他のリソース集中型アプリケーションと同じシステムで実行している場合は、JBoss ON サーバーに十分なシステムリソースがあることを確認するために JVM を調整する必要があります。それ以外の場合は、サーバーがメモリー不足のエラーが発生する可能性があります。

サーバーの JVM を調整する際に、ヒープサイズを増やし、システムスレッドの制限を、他のアプリケーション用の十分なシステムリソースを残す一方で、メモリーエラーが解決されるほど大きな値に増やします。

1. root で、システムのユーザースレッド制限を増やします。

```
[root@server ~]# ulimit -u 4096
```

2. **rhq-server-env.sh** スクリプトを開き、新しい JVM 設定を設定します。

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server-env.sh
```

3. Java オプションを変更して、ヒープサイズ **-Xms** を最小および **-Xmx** 最大まで増やします。例：

```
RHQ_SERVER_JAVA_OPTS="-Xms512M -Xmx1024M -XX:MaxPermSize=128M -
Djava.net.preferIPv4Stack=true -Dorg.jboss.resolver.warning=true"
```

4. サーバーを再起動して、新しいヒープ設定を読み込みます。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --server
```

バグを報告します。

3.6. 多数のエージェントのサーバーチューニング

JBoss ON サーバーに多数のエージェントがインベントリーに多数ある場合（リソース数や監視スケジュールなどの設定により、これは 100 エージェントまたは 1,000 台を超えるエージェントの数により）、デフォルト設定ではアップグレードができません。サーバーがすべてのデータを正しく読み込むことができないので、データのサイズは十分です。

通常の運用中に同様のパフォーマンス問題が発生する可能性は低くなりますが、

最も大きな症状の 1 つは、エージェント要求のタイムアウトが頻繁に発生することです。

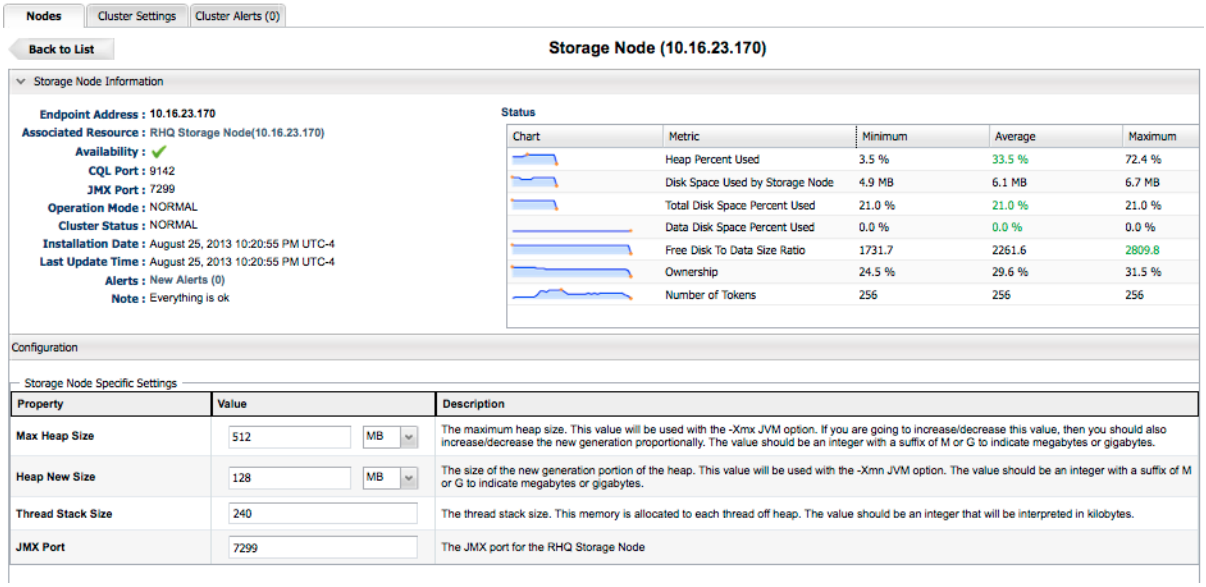
問題はメモリーに関連する問題ではなく、スレッドの問題です。JBoss ON サーバーを上回るエージェントリクエストの数により、プロセスは全体的に遅くなります。

サーバー設定には、パフォーマンスを向上させるために調整できる部分が 3 つあります。ストレージノードのメモリー設定の増加、EJB プールの増加、同時実行制限のリセット、許可されたエージェント接続数を増やすために調整できます。

1. ストレージノードのメモリー使用量のデフォルトサイズを増やします。これは、約 1,000 以上のノードでのみ必要です。

   この設定は、ストレージノードの管理 UI で確認できます。ストレージノードの JVM ヒープサイズを変更するには、新しい値を入力して Save ボタンをクリックします。設定の変更がディスクに適用され、ストレージノードが再起動されます。

   図3.1 ストレージノードの構成設定



2. EJB プールを増やします。

   a. サーバーの **standalone-full.xml** プロファイルを開きます。

```
[root@server ~]# vim /opt/jon/jon-server-3.3.0.GA/jbossas/standalone/configuration/standalone-full.xml
```

b. **strict-max-pool** キーを変更してプールサイズを増やします。デフォルトは 20 です。例：

```
<strict-max-pool name="slsb-strict-max-pool" max-pool-size="2000" instance-acquisition-timeout="1" instance-acquisition-timeout-unit="MINUTES"/>
```

注記

このオプションを選択し、**standalone-full.xml** ファイルが変更された場合、管理者は JON アップグレードプロセス外で維持する必要があります。パッチと更新は、ファイルをデフォルト設定に戻します。

3. コンカレンシー制限を引き上げ、同時にサーバーと通信できるエージェントの数を増やします。

a. **rhq-server.properties** ファイルを開きます。

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties
```

b. 通信関連のパラメーターにはブロックがあります。同時実行制限は、パラメーターおよび **concurrency-limit rhq.communications.global-concurrency-limit** パラメーターで設定されます。Web UI 接続およびダウンロードには、その他の通信制限があります。異なる通信パラメーターについては、を参照してください 「コンカレンシー制限の設定」。

例：

```
rhq.server.startup.web.max-connections=1000
rhq.server.agent-downloads-limit=45
rhq.server.client-downloads-limit=5
rhq.communications.global-concurrency-limit=200
rhq.server.concurrency-limit.inventory-report=25
rhq.server.concurrency-limit.availability-report=25
rhq.server.concurrency-limit.inventory-sync=25
rhq.server.concurrency-limit.content-report=25
rhq.server.concurrency-limit.content-download=25
rhq.server.concurrency-limit.measurement-report=25
rhq.server.concurrency-limit.measurement-schedule-request=25
rhq.server.concurrency-limit.configuration-update=25
```

4. サーバーを再起動して、新しい設定を読み込みます。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl restart --server
```

注記

サーバーは再起動するまで新しい設定を使用しません。

バグを報告します。

3.7. データベースおよびディスク領域

特定の JBoss ON 機能はストレージ要件に大きく影響します。JBoss ON データベースにコンテンツの格納に関連するもの（設定が分散スナップショット、バンドルバージョン、WAR などのコンテンツベースのリソースなど）を使用することで、ストレージ要件が増加します。

JBoss ON はコンテンツのすべてのバージョンを保存します。したがって、バックエンドデータベース（Oracle または PostgreSQL）をホストするシステムには、ドリフト監視、コンテンツ更新、バンドルを使用して、すべてのリソースのすべてのバージョンを保存するのに十分なディスク領域が必要です。また、データベース自体にコンテンツに適したテーブル空間が必要です。

必要な領域を算出する際に、すべてのアーティファクト（バンドル、Web アプリケーション、監視ディレクトリー）のサイズを見積もり、各アーティファクトのバージョン数を計算します。少なくとも **2 倍の領域が利用可能です。PostgreSQL と Oracle の** 両方で、vacuum、圧縮、およびバックアップおよび復元などのクリーンアップ操作を実行するには、PostgreSQL と Oracle の両方に 2 倍のデータベースサイズが必要です。

バグを報告します。

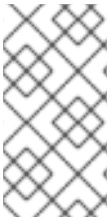
第4章 サーバーエージェント通信の SSL 接続の設定

デフォルトでは、JBoss ON サーバーおよび JBoss ON エージェントは **クリアで** 相互に通信するため、通信トラフィックはすべて暗号化されず、認証はいずれのエンドも実行されません。

JBoss ON は一部のリソース種別で設定変更を実施できるため、特にサーバーをクリアして実行すると、ネットワークのセキュリティー上の考慮事項が考えられます。JBoss ON は、暗号化または認証なしで実行する必要があります。JBoss ON がテストされている場合や、すべての JBoss ON サーバーおよびエージェントが完全にセキュアなネットワークにデプロイされ、ファイアウォールや VPN によるアクセスに制限され、信頼できる担当者に制限されている場合にのみ実行する必要があります。

JBoss ON は SSL/TLS を使用して、2 つの方法でエージェントとサーバー間の接続をセキュア化します。

- **暗号化** は、セッション中にエージェントとサーバー間で送信されたデータを特別にエンコードします。
- **認証** は SSL サーバー証明書およびクライアント証明書を使用して、サーバーに接続する前にエージェントのアイデンティティーを検証します。その逆も同様です。



注記

サーバーが使用する基本的な認証メカニズムがあります。このメカニズムでは、登録済みエージェントの特定および認証に使用されるエージェントにセキュリティートークンを割り当てます。ただし、このトークンメカニズムは、JBoss ON ネットワークを侵入から保護するために強力な認証スキームと見なされないでください。

暗号化の設定は非常にシンプルで、サーバーとエージェント間で適切なトランスポートメカニズムを有効にする必要があります。これにより、攻撃者がデータを傍受したり、中間者攻撃を設定し、正当な JBoss ON サーバーと正当な JBoss ON エージェントとの間で通信またはデータを傍受しないようにします。

認証は、攻撃者が JBoss ON エージェントの「不正な」インストールを防ぎ、JBoss ON システムに登録し、不正エージェントがネットワークにアクセスできるようにすることで、保護のもう 1 つの層を追加します。認証の設定は暗号化のみを使用する場合よりも複雑ですが、特にネットワーク設定に脆弱性がある場合など、追加の保護のために実装することが推奨されます。

バグを報告します。

4.1. 暗号化の設定

暗号化を設定するために必要なのは、JBoss ON サーバーおよびエージェント設定ファイルで SSL トランスポートコネクターを有効にすることです。SSL には、2 つのトランスポートオプションが **sslservlet** あり **sslsocket**ます。

JBoss ON サーバーには、暗号化に使用できるデフォルトの証明書があり、エージェントは自己署名証明書を生成することができるため、暗号化のみで追加の SSL 証明書を生成またはインストールする必要はありません。

1. 最初に、JBoss ON サーバーで SSL 暗号化を有効にします。

   a. JBoss ON サーバーをシャットダウンします。

```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop
```

b. JBoss ON サーバーの ***serverRoot*/jon-server-3.3.0.GA/bin/rhq-server.properties** ファイルを開きます。

c. SSL を使用するよう **rhq.communications.connector.*** 設定を編集します。認証に推奨される **sslsocket** トランスポートメソッドを使用するには、**rhq.communications.connector.transport** メソッドを更新し、ソケットに使用するポート番号を設定し、トランスポートパラメーター設定で指定されているサーブレットを削除します。

```
rhq.communications.connector.transport=sslsocket
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=55555
rhq.communications.connector.transport-params=
```

sslservlet トランスポートメソッドを使用するには、メソッドを変更するだけで十分です **rhq.communications.connector.transport**。

```
rhq.communications.connector.transport=sslservlet
rhq.communications.connector.bind-address=
rhq.communications.connector.bind-port=
rhq.communications.connector.transport-params=/jboss-remoting-servlet-
invoker/ServerInvokerServlet
```

d. 暗号化のみを設定する場合は、証明書ベースの認証が無効になっていることを確認してください。

```
rhq.server.tomcat.security.client-auth-mode=false
rhq.server.client.security.server-auth-mode-enabled=false
```

e. 必要に応じて、使用するセキュアなプロトコルを定義します。デフォルトは TLS（通常は問題ありません）ですが、SSL に設定できます。

```
rhq.server.tomcat.security.secure-socket-protocol=TLSv1,TLSv1.1,TLSv1.2
rhq.server.client.security.secure-socket-protocol=TLS
```

f. 変更を保存し、JBoss ON サーバーを再起動します。

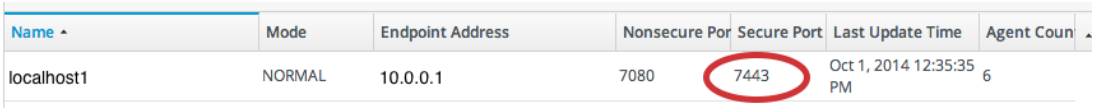
```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl start
```

g. 設定で指定された終了ポイントアドレスとポート番号が JBoss ON のサーバーに設定されていることを確認します。

i. トップメニューの **Administration** タブをクリックします。

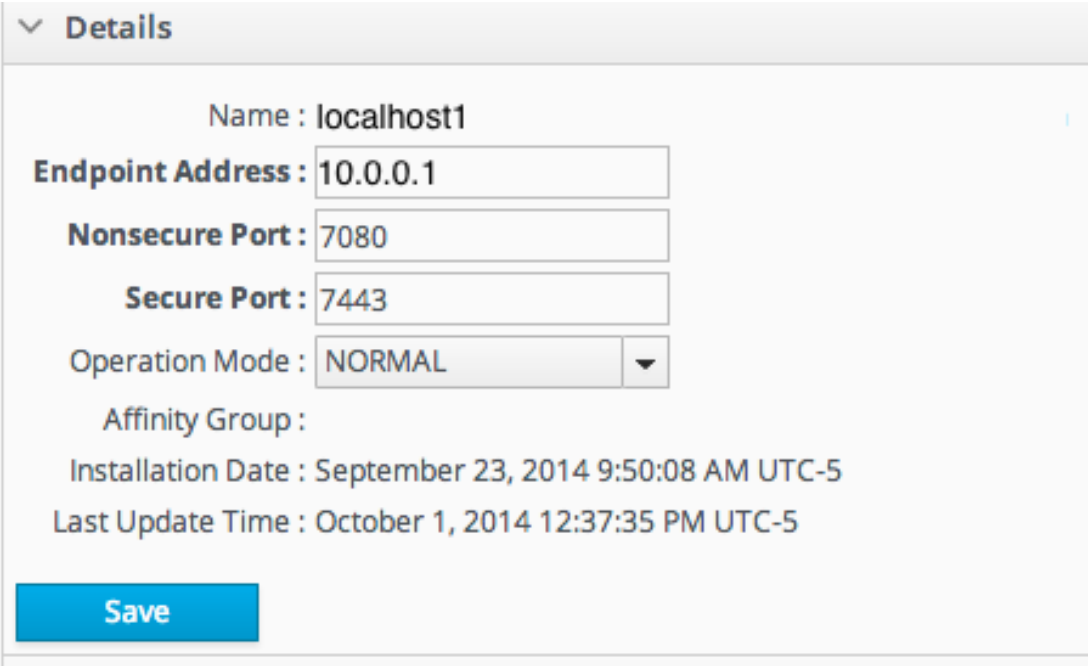
ii. 左側の **Topology** ボックスで **Servers** 項目を選択します。

iii. **Secure Port** 列のポート番号を確認します。



iv. 値が間違っている場合は、サーバーの名前をクリックして編集ページを開きます。

v. サーバー情報の **Edit** 下にあるをクリックして、必要に応じて終了ポイントアドレスまたはポートをリセットします。



2. 次に、エージェントで SSL 暗号化を有効にします。

注記

ここでは、エージェント設定ファイルを編集してエージェント設定を編集する方法を説明します。エージェント設定は、エージェント起動スクリプトの高度な設定モードに移動して編集することもできます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig --setup --advanced
```

a. エージェントの設定ファイルを開きます。

```
vim agentRoot/rhq-agent/conf/agent-configuration.xml
```

b. トランスポートプロトコルをに変更し **sslsocket**ます。

```
<entry key="rhq.communications.connector.transport"       value="sslsocket" />
```

c. サーバーの設定に一致するように、適切なサーバー接続情報を設定します。以下の設定では、sslsocket を使用して、エージェントが server.example.com ポート 55555 の JBoss ON サーバーに接続できるようになりました。

```
<entry key="rhq.agent.server.transport" value="sslsocket" />
<entry key="rhq.agent.server.bind-port" value="55555" />
<entry key="rhq.agent.server.bind-address" value="server.example.com" />
<entry key="rhq.agent.server.transport-params" value="" />
```

sslservlet を使用する場合、設定は以下のようになります。

```
<entry key="rhq.agent.server.transport" value="sslservlet" />
<entry key="rhq.agent.server.bind-port" value="7443" />
<entry key="rhq.agent.server.bind-address" value="server.example.com" />
<entry key="rhq.agent.server.transport-params" value="/jboss-remoting-servlet-invoker/ServerInvokerServlet" />
```

d. 暗号化のみを設定する場合は、証明書ベースの認証が無効になっていることを確認してください。これらのパラメーターはコメントアウトすることも、明示的に認証をオフにするように設定することもできます。

```
<entry key="rhq.communications.connector.security.client-auth-mode"       value="none" />
<entry key="rhq.agent.client.security.server-auth-mode-enabled" value="false" />
```

e. 必要に応じて、エージェントの追加プロトコル設定を定義します。これは、サーバーが TLS 以外のトランスポートプロトコルを使用するように設定されている場合に必要です。

```
<entry key="rhq.communications.connector.security.secure-socket-protocol" value="TLS" />
<entry key="rhq.agent.client.security.secure-socket-protocol"   value="TLS" />
```

f. **--cleanconfig** オプションを使用して新しい設定を読み込んで、エージェントを終了し、再起動します。

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig
```

バグを報告します。

4.2. サーバーおよびエージェント間のクライアント認証の設定

認証 は、何かのアイデンティティーを検証するプロセスです。証明書ベースの認証では、エンティティーはそのエンティティーを識別するために使用される信頼されたソースから証明書ファイルを取得し、SSL 接続を開始する際にそのエンティティーを特定します。これにより、SSL 接続に関与する唯一の参加者が、そのユーザーであると判断されます。

JBoss ON に証明書ベースの認証を設定するには、複数の手順を実行する必要があります。暗号化を有効にするには、JBoss ON のサーバーおよびエージェントに対して証明書を発行および保存する必要があります。また、信頼できないクライアントからのメッセージを拒否するようサーバーおよびエージェントを設定する必要があります。

JBoss ON の SSL 認証は **双方向** です。エージェントはサーバーに対して認証するよう設定され、サーバーはエージェントに対して認証するように設定されます。

注記

サーバーまたはエージェントのみが認証する必要がある一方向認証を設定できます。最善のセキュリティーは双方向認証を使用したものです。これはここで説明する設定です。

JBoss ON には、SSL 接続を許可するトランスポートメソッドが 2 つ **sslservlet** あり **sslsocket**ます。

以下の手順では **sslsocket**、サーバーエージェント SSL 接続に特別なポートが使用される一方で、GUI 接続にデフォルトの指定ポートを使用できるようにします。

sslservlet は組み込みの Tomcat サーバーを利用しますが、GUI ユーザーがサーバーへの認証と、エージェントの証明書ベースの認証を有効にする必要があります。

これを行うには、ブラウザーが使用するセキュアで認証されていない https コネクターを提供する JON Server の Tomcat 設定に別の Web コネクターを追加する必要があります。

この新しい Web コネクターを追加したら、ユーザーはそのコネクターのポート（例： https://your-server-hostname:9443）に接続し、証明書の認証なしに **https** を使用して JON GUI にアクセスできます。

これにより、元のセキュアな Tomcat コネクターは、証明書が認証されたエージェント要求を受け入れるために解放されます。

このような Web コネクターは、にある JBoss EAP CLI を使用して追加でき ***JBOSS_HOME*/bin/jboss-cli.\[sh,bat]**ます。

たとえば、ポート 9443 に新しい Web コネクターを追加するには、最初に JBoss CLI を使用してそのポート 9443 の新しいソケットバインディングを作成し、その新しいソケットバインディングを使用して新しい Web コネクターを作成します。

手順4.1（オプション）新しい Tomcat Web コネクターの追加

1. ```
   jboss-cli.sh --controller=127.0.0.1:6999 --connect --command='/socket-binding-group=standard-sockets/socket-binding=httpsbrowser/:add(port=9443)'
   ```

2. ```
   jboss-cli.sh --controller=127.0.0.1:6999 --connect --command='/subsystem=web/connector=httpsbrowser/:add(socket-binding=httpsbrowser,scheme=https,protocol=HTTP/1.1,secure=true,enabled=true)'
   ```

3. ```
   jboss-cli.sh  --controller=127.0.0.1:6999 --connect --command='/subsystem=web/connector=httpsbrowser/ssl=configuration:add(name=ssl,verify-client=false,key-alias=RHQ,password=${VAULT::restricted::rhq.server.tomcat.security.keystore.password::5fb458952ebdaa86aa0b4e8d3eac5d13},certificate-key-file=${jboss.server.config.dir}/rhq.keystore,certificate-file=${jboss.server.config.dir}/rhq.keystore'
   ```

4. JBoss ON サーバーの再起動

   ```
   rhqctl restart --server
   ```

注記

上記の手順では、JON のキーストアにある自己署名証明書を使用する場合に SSL 設定を作成します。証明書（特に自己署名なしの証明書）を使用することが推奨されます。独自のキーストア情報を使用するには、上記の手順（具体的には key-alias、password、certificate-key-file、および certificate-file の値）で変更します。

手順4.2 サーバーとエージェント間のクライアント認証の設定

1. にあるように暗号化を有効にします。これは 「暗号化の設定」、クライアント認証が無効にならない ようにします。
   ```

2. SSL ソケット接続は、ユーザー定義のポートで行われます。必要に応じて、ファイアウォールまたは VPN を開き、そのポートへのアクセスを許可します。

3. 各 JBoss ON サーバーおよびエージェントの SSL 証明書を生成します。例：

```
keytool -genkey -dname "CN=server1.example.com"  -keystore server1-keystore.dat -validity 3650 -alias server1 -keyalg RSA -storetype JKS -keypass secret -storepass secret
```

これにより、以下の特徴を持つ自己署名証明書が作成されます。

- サーバーのホスト名と同じ共通名(CN)値 **server1.example.com**。この **-dname** 値は、SSL 接続の初期ステップ（SSL ハンドシェイク）において、証明書を発行したアイデンティティーと同じものであることを確認するため、ホスト名と同じ値である必要があります。つまり、CN のホスト名と、証明書を示すサーバーまたはエージェントのホスト名が一致します。
- 「キーストアファイル」 **server1-keystore.dat**
- 有効期間（3650 日）
- エイリアス **server1**
- RSA の鍵アルゴリズム
- キーストアの JKS 形式で保存
- のキーおよびストレージのパスワード **secret**

組織では、証明書の生成または取得にすでに方法がある場合があります。この例では、他のユーティリティーを使用することも **certutil**できます **keytool**。他のユーティリティーも使用できます。この **keytool** ドキュメントは、http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html の Oracle-Sun サイトから入手でき ます。

4. 各自己署名証明書を単一のトラストストアファイルに配置します。

   a. 各キーストアから自己署名証明書をエクスポートします。

```
keytool -export -keystore server1-keystore.dat -alias server1 -storetype JKS -storepass secret -file server1-cert
```

   b. すべての証明書を単一のトラストストアファイルにインポートします。

```
keytool -import -keystore truststore.dat -alias server1 -storetype JKS -file server1-cert -noprompt -keypass secret -storepass secret
```

   -alias は、トラストストアのインポートされた証明書に付与する名前です。便宜上、これは元のキーストアファイルのエイリアスと同じです。

重要

エクスポートしたすべてのサーバー証明書とエージェント証明書を、同じトラストストアファイル **に** インポートします。

c. を使用して証明書を一覧表示し、すべての証明書が正常 **keytool** にインポートされたことを確認します。

```
keytool -list -keystore truststore.dat -storepass secret -storetype JKS

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entries

server1, Feb 25, 2017, trustedCertEntry,
Certificate fingerprint (MD5): 24:D9:8A:50:BA:1B:26:08:DC:44:A8:2A:9E:8A:43:D9
```

5. キーストアとトラストストアファイルをすべての JBoss ON およびサーバーマシンおよびエージェントマシンに分散します。キーストアは、証明書の CN のホスト名に一致するマシンにのみ配布し、キーストアを誤ったマシンに追加すると、SSL 接続が失敗します。

   a. サーバーでは、キーストア(**server1-keystore.dat**)を JBoss Operations Network サーバーに埋め込まれた JBoss AS サーバーの **jon-server-*version*/jbossas/standalone/configuration/** ディレクトリーにコピーします。**server1-keystore.dat** に変更し **keystore.dat**ます。

   b. サーバーでは、トラストストアを組み込み JBoss AS サーバーの ***serverRoot*/jon-server-3.3.0.GA/jbossas/standalone/configuration/** ディレクトリーにコピーします。このファイルに名前を付けてください **truststore.dat**。

   c. エージェントの場合は、キーストアを ***agentRoot*/rhq-agent/conf** ディレクトリーにコピーします。***agentRoot*/rhq-agent/conf** ディレクトリーの証明書ファイルは、自動更新後も保持されます。

6. JBoss ON サーバーをシャットダウンします。

```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop
```

7. JBoss ON サーバーの **rhq-server.properties** ファイルを開きます。

```
vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties
```

8. ***rhq.communications.connector.security.client-auth-mode*** パラメーターを **need** およびにに設定し、sslsocket を使用する際にクライアント認証を有効 ***rhq.server.client.security.server-auth-mode-enabled*** にし **true**ます。sslservlet set パラメーターを **true**、および ***rhq.server.tomcat.security.client-auth-mode*** パラメーターがと使用する場合 ***rhq.server.client.security.server-auth-mode-enabled*** **true**。

   キーストアおよびトラストストアファイルに関する情報を設定します。

   受信メッセージ（sslsocket 使用時のエージェント間通信）の設定はすべて ***rhq.communications.connector.security.**** パラメーターで設定されます。送信メッセージの設定は、***rhq.server.client.security.**** パラメーターで設定されます。

```
# Server-side SSL Security Configuration (for incoming messages from agents)
# These are used when secure transports other than sslservlet are used
rhq.communications.connector.security.secure-socket-protocol=TLS
rhq.communications.connector.security.keystore.file=${jboss.server.config.dir}/keystore.dat
rhq.communications.connector.security.keystore.algorithm=SunX509
```

```
rhq.communications.connector.security.keystore.type=JKS
rhq.communications.connector.security.keystore.password=secret
rhq.communications.connector.security.keystore.key-password=secret
rhq.communications.connector.security.keystore.alias=server1
rhq.communications.connector.security.truststore.file=${jboss.server.config.dir}/truststore.dat
rhq.communications.connector.security.truststore.algorithm=SunX509
rhq.communications.connector.security.truststore.type=JKS
rhq.communications.connector.security.truststore.password=secret
rhq.communications.connector.security.client-auth-mode=need

...

# Client-side SSL Security Configuration (for outgoing messages to agents)
rhq.server.client.security.secure-socket-protocol=TLS
rhq.server.client.security.keystore.file=${jboss.server.config.dir}/keystore.dat
rhq.server.client.security.keystore.algorithm=SunX509
rhq.server.client.security.keystore.type=JKS
rhq.server.client.security.keystore.password=secret
rhq.server.client.security.keystore.key-password=secret
rhq.server.client.security.keystore.alias=server1
rhq.server.client.security.truststore.file=${jboss.server.config.dir}/truststore.dat
rhq.server.client.security.truststore.algorithm=SunX509
rhq.server.client.security.truststore.type=JKS
rhq.server.client.security.truststore.password=secret
rhq.server.client.security.server-auth-mode-enabled=true
```

9. ファイルを保存し、サーバーを再起動します。

```
serverRoot/jon-server-3.3.0.GA/bin/rhqctl start
```

10. エージェント設定ファイルで、セキュアな接続に関連する行のコメントを解除します。これらのパラメーターは、**rhq.communications.connector.security.*** とで始まり、**rhq.agent.client.security.*** エージェント間の通信とサーバー間の通信はそれぞれで始まります。

    適切な値を入力します。

```
<entry key="rhq.communications.connector.security.secure-socket-protocol" value="TLS" />
<entry key="rhq.communications.connector.security.keystore.file"
value="conf/keystore.dat" />
<entry key="rhq.communications.connector.security.keystore.algorithm"     value="SunX509"
/>
<entry key="rhq.communications.connector.security.keystore.type"          value="JKS" />
<entry key="rhq.communications.connector.security.keystore.password"      value="rhqpwd"
/>
<entry key="rhq.communications.connector.security.keystore.key-password"
value="rhqpwd" />
<entry key="rhq.communications.connector.security.keystore.alias"         value="rhq" />
<entry key="rhq.communications.connector.security.truststore.file"
value="conf/truststore.dat" />
<entry key="rhq.communications.connector.security.truststore.algorithm"   value="SunX509"
/>
<entry key="rhq.communications.connector.security.truststore.type"        value="JKS" />
<entry key="rhq.communications.connector.security.truststore.password"    value="" />
<entry key="rhq.communications.connector.security.client-auth-mode"       value="need" />
```

```
<entry key="rhq.agent.client.security.secure-socket-protocol"   value="TLS" />
<entry key="rhq.agent.client.security.keystore.file"            value="conf/keystore.dat" />
<entry key="rhq.agent.client.security.keystore.algorithm"       value="SunX509" />
<entry key="rhq.agent.client.security.keystore.type"            value="JKS" />
<entry key="rhq.agent.client.security.keystore.password"        value="rhqpwd" />
<entry key="rhq.agent.client.security.keystore.key-password"    value="rhqpwd" />
<entry key="rhq.agent.client.security.keystore.alias"           value="rhq" />
<entry key="rhq.agent.client.security.truststore.file"          value="conf/truststore.dat" />
<entry key="rhq.agent.client.security.truststore.algorithm"     value="SunX509" />
<entry key="rhq.agent.client.security.truststore.type"          value="JKS" />
<entry key="rhq.agent.client.security.truststore.password"      value="" />
<entry key="rhq.agent.client.security.server-auth-mode-enabled" value="true" />
```

注記

ここでは、エージェント設定ファイルを編集してエージェント設定を編集する方法を説明します。エージェント設定は、エージェント起動スクリプトの高度な設定モードに移動して編集することもできます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh --cleanconfig --setup --advanced
```

バグを報告します。

4.3. SSL の問題のトラブルシューティング

SSL の接続問題の最も一般的なシンボリックリンクは、JBoss ON サーバーへの接続を確立できないため、起動時にエージェントがハングする点です。SSL 問題が発生したかどうかを確認するには、さまざまなエリアがあります。

バグを報告します。

4.3.1. 一般的な SSL 接続の問題

SSL の問題は単に接続の問題です。これは、エージェントまたはサーバー設定に問題があることを示しています。設定が正常であることを確認するための一般的なエリアがいくつかあります。

- エージェントとサーバーのホスト名の両方がサーバー証明書のホスト名で解決可能であることを確認してください。
- サーバーのセキュアなポートに指定されたポート番号が実際にサーバーに設定されたポート番号であることを確認してください。**Administration > High Availability > Servers** ページを確認して、パブリックエンドポイントのアドレスとポートが正しいことを確認します。UI でサーバー定義を編集して、SSL 設定と同じにします。

図4.1 サーバーのホスト名およびポート設定

Name	Mode	Endpoint Address	Nonsecure Por	Secure Port	Last Update Time	Agent Coun
localhost1	NORMAL	10.0.0.1	7080	7443	Oct 1, 2014 12:35:35 PM	6

この値が SSL 接続に設定された値と同じ値と一致しない場合、エージェントはサーバーと通信できません。

- エージェントとサーバーのホスト名の両方がサーバー証明書のホスト名で解決可能であることを確認してください。
- agent-server 通信に使用されるすべての証明書が適切なエイリアスを持つ必要なキーストアに保存されていることを確認します。
- パスワードがキーストアにアクセスするよう適切に設定されていることを確認します。
- 通信が TLS を使用するように設定されていることを確認します。
- サーバーおよびエージェント設定（特に割り当てられたトランスポート（ソケットまたはサーブレット））オプションを検証します。の設定例は次のとおりです 「SSL 設定例」。
- クライアント認証が必要で、サーバーが **sslservlet** transport オプションを使用している場合、JBoss ON UI に接続するすべてのユーザーに、クライアント認証を使用してサーバー UI に接続できるように、JBoss ON UI にインストールされたユーザー証明書があることを確認します。エージェント証明書と同様に、ユーザー証明書はサーバーのキーストアに保存する必要があり 「サーバーおよびエージェント間のクライアント認証の設定」 ます。

  ユーザーがクライアント認証を使用して接続できない場合は、の **sslsocket** 代わりにサーバーを変更し **sslservlet**ます。

バグを報告します。

4.3.2. SSL デバッグの有効化

エージェントで詳細なロギングを有効にすると、エージェントログにさらに詳細な SSL 通信メッセージが返され、接続の問題の診断に役立ちます。

1. エージェントの環境変数ファイルを開きます。これにより、デバッグログ設定を含む、エージェントが実行される JVM の設定の一部を定義します。

   ```
   vim agentRoot/rhq-agent/bin/rhq-agent-env.sh
   ```

2. デバッグ環境変数を設定する **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** 行を追加します。

   ```
   RHQ_AGENT_ADDITIONAL_JAVA_OPTS="-Djavax.net.debug=all"
   ```

3. エージェントを再起動します。

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

バグを報告します。

4.3.3. SSL 設定例

以下の例は、異なる暗号化シナリオおよび認証設定シナリオのサーバー設定ファイルとエージェント設定ファイルの両方で正しい設定を示しています。

注記

以下の例は最小限の設定のみを示し、デフォルトのキーストアおよびトラストストアの使用を想定しています。関連するキーストアおよびトラストストアプロパティーを適切な値で更新する必要があります。

例4.1暗号化のみ： サーバー(sslservlet)およびエージェント(sslsocket)

サーバーの設定	エージェントの設定
rhq.communications.connector.transport=sslservlet rhq.communications.connector.bind-address= rhq.communications.connector.bind-port= rhq.communications.connector.transport-params=/jboss-remoting-servlet-invoker/ServerInvokerServlet rhq.server.tomcat.security.client-auth-mode=false rhq.server.client.security.server-auth-mode-enabled=false	<entry key="rhq.communications.connector.transport" value="sslsocket" /> <entry key="rhq.agent.server.transport" value="sslservlet" /> <entry key="rhq.agent.server.bind-port" value="7443" />

エージェント設定は、**sslservlet** またはのいずれかになるように **サーバーの接続** 情報を定義し **sslsocket**ます。エージェントは、受信メッセージしか受信できません **sslsocket**。

例4.2 暗号化のみ： サーバー(sslsocket)およびエージェント(sslsocket)

サーバーの設定	エージェントの設定
rhq.communications.connector.transport=sslsocket rhq.communications.connector.bind-address= rhq.communications.connector.bind-port=7800 rhq.communications.connector.transport-params= rhq.server.tomcat.security.client-auth-mode=false rhq.server.client.security.server-auth-mode-enabled=false	<entry key="rhq.agent.server.transport" value="sslsocket" /> <entry key="rhq.agent.server.bind-port" value="7800" /> <entry key="rhq.agent.server.transport-params" value="" />

エージェント設定 **はサーバーの接続** 情報を定義するため、サーバーの **rhq-server.properties** ファイルの設定と一致する必要があります。

例4.3 暗号化およびクライアント認証： サーバー(sslservlet)およびエージェント(sslsocket)

サーバーの設定	エージェントの設定
rhq.communications.connector.transport=sslservlet rhq.communications.connector.bind-address= rhq.communications.connector.bind-port= rhq.communications.connector.transport-params=/jboss-remoting-servlet-invoker/ServerInvokerServlet rhq.server.tomcat.security.client-auth-mode=true rhq.server.client.security.server-auth-mode-enabled=true	<entry key="rhq.communications.connector.transport" value="sslsocket" /> <entry key="rhq.agent.server.transport" value="sslservlet" /> <entry key="rhq.agent.server.bind-port" value="7443" /> <entry key="rhq.communications.connector.security.client-auth-mode" value="need" /> <entry key="rhq.agent.client.security.server-auth-mode-enabled" value="true" />

例4.4 暗号化およびクライアント認証： サーバー(sslsocket)およびエージェント(sslsocket)

サーバーの設定	エージェントの設定
rhq.communications.connector.transport=sslsocket rhq.communications.connector.bind-address= rhq.communications.connector.bind-port=55555 rhq.communications.connector.transport-params= rhq.communications.connector.security.client-auth-mode=need rhq.server.client.security.server-auth-mode-enabled=true	<entry key="rhq.agent.server.transport" value="sslsocket" /> <entry key="rhq.agent.server.bind-port" value="55555" /> <entry key="rhq.agent.server.transport-params" value="" /> <entry key="rhq.communications.connector.security.client-auth-mode" value="need" /> <entry key="rhq.agent.client.security.server-auth-mode-enabled" value="true" />

バグを報告します。

第5章 高可用性とエージェントのロードバランシングの使用

JBoss ON サーバーと高可用性は、同じ中央データベースを使用するすべての JBoss ON サーバーがクラウドで相互作用することを意味します。これにより、サーバーがメンテナンスのためにオフラインにする必要がある場合にサーバー間でシームレスなフェイルオーバーが可能になり、負荷分散エージェントとリソース操作に最適な方法が提供されます。

クラウドで複数の JBoss ON サーバーを使用すると、エージェントは通常の通信に使用するサーバーの設定を定義できます。この優先順位(**affinity**)は、エージェントサーバー通信を負荷分散し、全体的なパフォーマンスを向上させる方法です。

バグを報告します。

5.1. エージェントサーバー通信とサーバーの可用性について

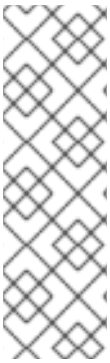
5.1.1. エージェントおよびサーバー通信

高可用性を使用するかどうかの計画の一部は、エージェントとサーバーが相互に通信する方法を理解することです。

エージェントとサーバーには双方向通信があります。エージェントは、現在の監視データ、設定設定、リソース、およびその他の現在のデータをサーバーに送信します。サーバーは、設定更新、アラート定義、ドリフト定義、およびその他の設定をエージェントに送信します。

エージェントを最初にインストールすると、エージェントの設定で接続するサーバーのホスト名または IP アドレスの入力が求められます。これは、（JBoss ON デプロイメントにあるすべてのサーバー）登録サーバーです。次に、エージェント登録の一環として、利用可能な JBoss ON サーバーの一覧を受け取ります。その一覧の最初のサーバーは、エージェントが最も定期的に通信を試み、リストの他のサーバーを順番に試行するサーバーです（その後で 「エージェントおよびサーバーフェイルオーバー」）。その最初のサーバーは、登録サーバーであるか、異なるサーバーである場合もありますが、問題はありません。

エージェントが接続するサーバーには、優先度が若干ありますが、エージェントがサーバーに接続するサーバーやサーバーがどのエージェントと通信できるかには制限はありません。すべてのサーバーは任意の時点で任意のエージェントと通信でき、その逆も同様です。



注記

通信は双方向である必要があるため、すべてのサーバーがすべてのエージェントからアクセスできる必要があり、すべてのエージェントがすべてのサーバーからアクセスできるようにする必要があります。サーバーまたはエージェントのインストール時に設定されるサーバーおよびエージェントのホスト名および IP アドレスが解決可能な場合は、重要です。

ただし、サーバーは **相互に通信しないため、サーバー間** で相互のホスト名または IP アドレスを解決することはできません。

バグを報告します。

5.1.2. サーバーの可用性： 単一クラウドで複数のサーバー

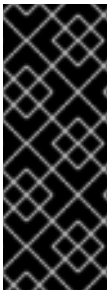
多くのデプロイメントでは、単一の JBoss ON サーバーがあり、すべてのエージェントがそのサーバーと通信します。ただし、複数のサーバーが有益な環境は複数あります。

- エージェントの負荷を処理する問題があり、メトリクスの評価、アラートまたはイベントの生成、またはリソースの可用性の報告に影響を及ぼします。これは、エージェントの数により必ずしも生じる訳ではありません。ネットワーク品質またはその他の要素に関連する可能性があります。

- 複数のデータセンターまたはサーバーへのエージェントの論理グループを持つ地理的に分散環境が必要です。

同じデプロイメントの複数の JBoss ON サーバーが同じバックエンドデータベースを使用するように設定されます。新しい JBoss ON サーバーがデータベースに追加されると、そのサーバーは **JBoss ON サーバー高可用性クラウド** に自動的に追加されます。

高可用性設定は、必ずしも多数の JBoss ON エージェントを意味するわけではありません。複数のサーバー **が中央データベースの負荷に影響を及ぼさない** ため、パフォーマンスには大きな影響はありません。高可用性の目的は、JBoss ON デプロイメント全体が応答性と可用性、フォールトトレランスを必要とすることです。そのため、複数のサーバーが必要になります。これは、比較的少ないエージェントでも当てはまります。



重要

比較的簡単な高可用性サーバークラウドに JBoss ON サーバーを追加することは可能ですが、バックエンドデータベースに影響する可能性があるため、注意して実行する必要があります。各 JBoss ON サーバーは同時データベース接続を制限しますが、クラウド全体の接続の合計数に制限はありません。2 番目のサーバーを追加すると、エージェントの数は同じでも、潜在的なデータベース接続を 2 倍にできます。サーバーが追加されると、接続が増えると線形になります。

基本的には、高可用性は JBoss ON デプロイメント全体に対して、基本的なフェイルオーバーと冗長性を提供する方法です。すべてのサーバーは同じデータベースバックエンドを使用するため、すべて同じエージェントとインベントリーにアクセスでき、データの監視、リソース履歴、およびその他の情報にアクセスできます。つまり、すべての JBoss ON サーバーが基本的に同じであることを意味します。

JBoss ON サーバーは、高可用性クラウドに簡単に追加および削除できます。また、メンテナンスモードに切り替えてサーバーを一時的に削除することもできます。

高可用性を計画する際に必要となる事項がいくつかあります。

1. すべてのサーバーは、同じバージョンの JBoss ON を実行している必要があります。

2. すべてのサーバーに一意な名前を付ける必要があります。この文字列はサーバーのインストール時に定義されます。

3. 各サーバーは、高可用性サーバークラウドに対して実行される **すべて** の JBoss ON エージェントによって解決可能な一意のエンドポイント（ホスト名または IP アドレス）を定義する必要があります。

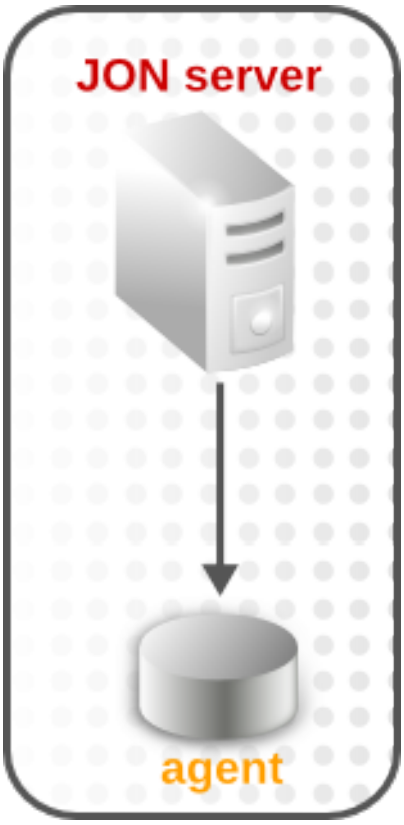
4. **オプション。**サーバーで同時実行制限を調整して、データベースへの負荷が過剰に発生し、パフォーマンスを低下させないようにします。

バグを報告します。

5.1.3. エージェントおよびサーバーのパーティション： エージェントロードの配布

1 台のサーバーしかない場合、エージェントディストリビューションは非常にシンプルです。すべてのエージェントがその 1 つのサーバーと通信します。

図5.1 単一のサーバー上にすべて



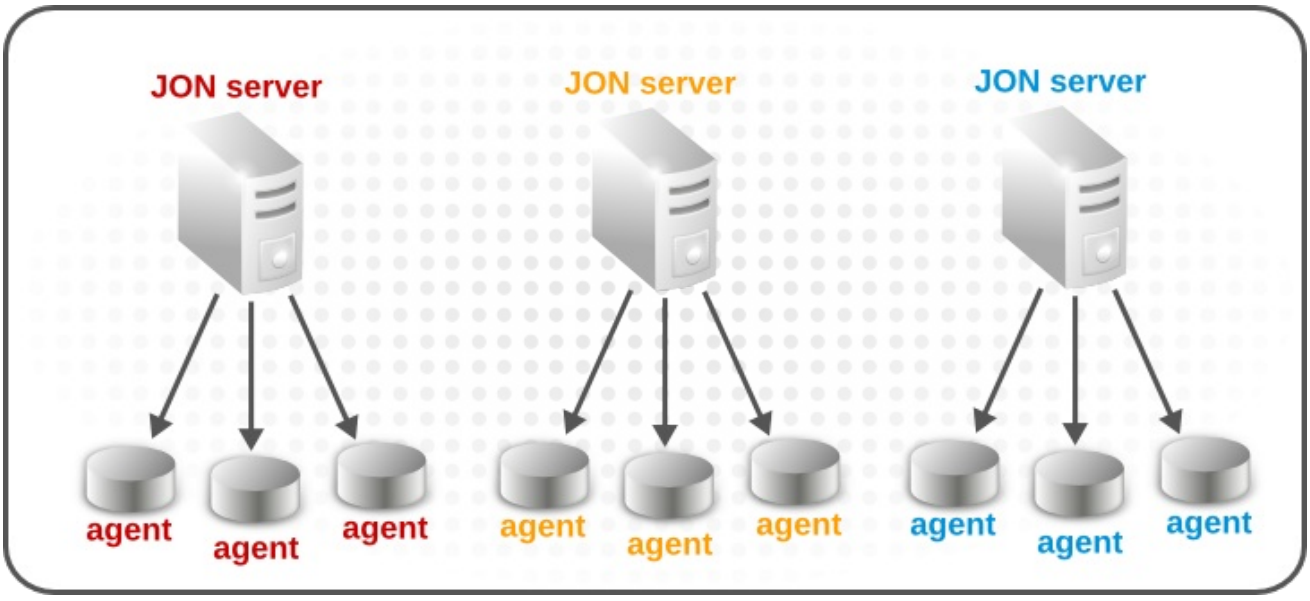
ただし、高可用性が導入されると、エージェントはサーバーと通信するサーバーを選択できます。すべてのサーバーは、エージェントに送信される利用可能なサーバーの一覧に含まれますが、エージェントは常にリスト内の最初のサーバー（その **プライマリーサーバー）への接続を試みます。**

この一覧はラウンドロビンパターンで生成されるため、サーバーリストはエージェントごとに若干異なります。例：

> A1: S1, S2, S3
> A2: S2, S3, S1
> A3: S3, S1, S2

これにより、サーバー全体でエージェントが非常に均等に分散されます。ディストリビューションは **パーティション** です。

図5.2 partitions: エージェントが分散された複数のサーバーを読み込む



サーバーが高可用性クラウドに追加されると、agent-server リストが更新されます。エージェントロードディストリビューションの変更は、パーティションイベントと呼ば **れます**。

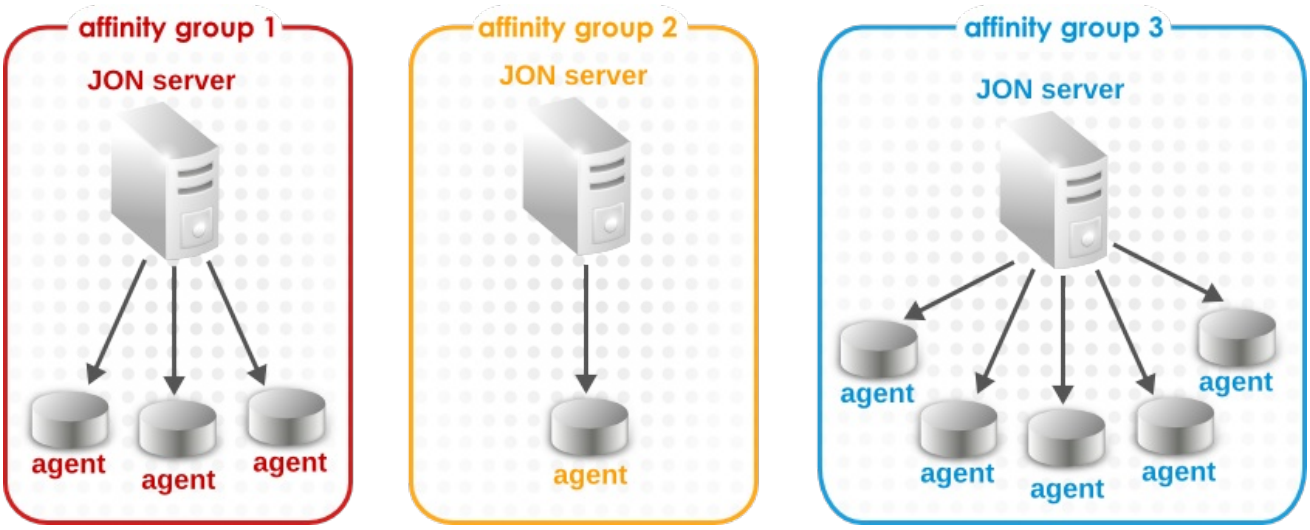
バグを報告します。

5.1.4. エージェントおよび推奨サーバー： アフィニティーおよびロードバランシング

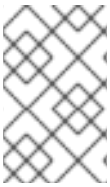
エージェント負荷の性質的な分布は、非常にランダムで、均等のディストリビューションを作成します。ただし、一部のネットワーク環境では、無作為の分布が妥当ではないり、最適な効率が得られる場合もあります。たとえば、同じリージョンまたはファシリティーにあるサーバーおよびエージェントは、はるかに距離にあるサーバーおよびエージェントよりも早く通信できます。この場合、管理者はランダムなサーバーエージェント関係ではなく、明示的な適切なサーバーエージェント関係を優先します。

JBoss ON には server-agent **affinity** の概念があります。アフィニティーは、エージェントが通信するサーバーに対する定義された優先順位です。アフィニティーグループは基本的に手動パーティションです。これはサーバーおよびエージェントのグループであり、エージェントは最初にアフィニティーグループでサーバーと選択的に通信します。

図5.3 エージェントロードのアフィニティー設定



アフィニティーグループは、どのサーバーと通信するかについてエージェントに対して緩やかな優先順位を作成します。ハードルールが作成されたり、エージェントが通信できるサーバーを制限するサーバーも制限されます。

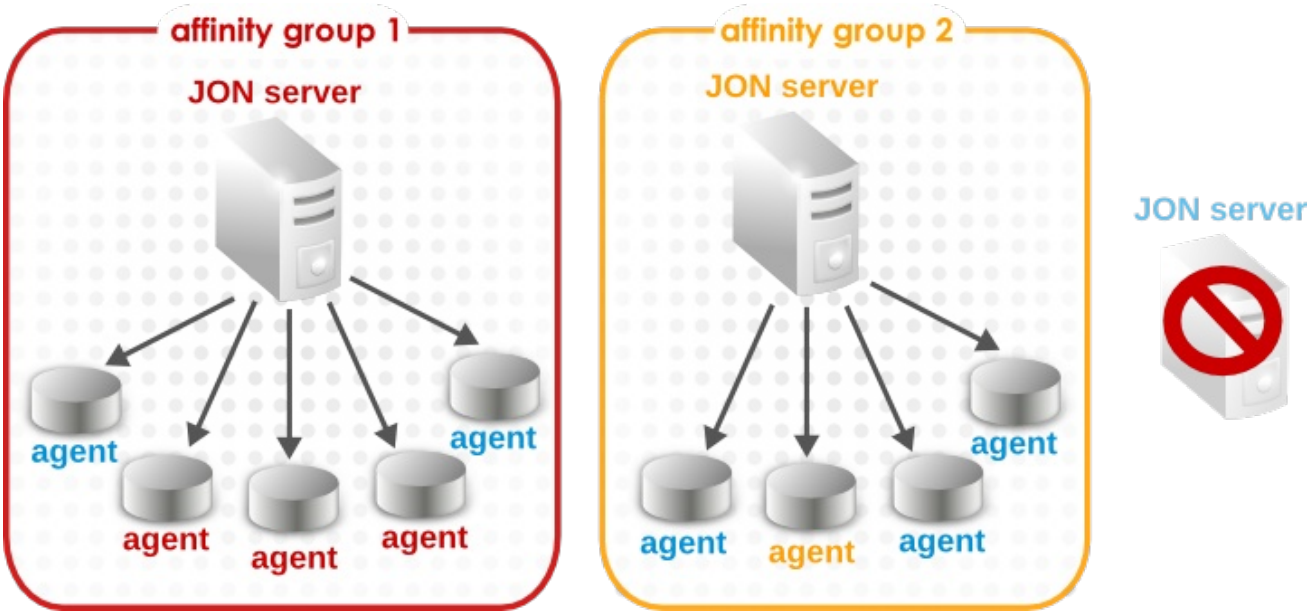


注記

アフィニティーグループは、エージェントからサーバーへの **一方向の優先** 度を定義します。アフィニティー設定に関係なく、サーバーは JBoss ON トポロジーのすべてのエージェントに接続できます。

エージェントのプライマリーサーバーが利用できない場合、エージェントはアフィニティーグループ内の他のサーバーを通過してラウンドロビンを試みます。これらのサーバーがない場合や、アフィニティーグループに他のサーバーがない場合、フェイルオーバーのリストに従って JBoss ON の高可用性クラウドのすべてのサーバーを繰り返し処理します。これは、アフィニティーグループのすべてのエージェントで当てはまるため、最終的にあるグループ内のエージェントは他の JBoss ON サーバー間で均等に分散されます。

図5.4 アフィニティーによるフェイルオーバー



アフィニティーグループは、以下の 3 つの利点を提供します。

- **物理またはネットワーク効率。** 一般的に、特定のエージェントサーバー接続が他よりも明確に効率的に実行される場合は、これらの接続を優先するようにアフィニティーを定義することが理にかなっています。これには、同じデータセンター、地理的なグループ化、またはネットワークトポロジーに同じサーバーとエージェントを共存させる可能性があります。

- **論理組織。** 運用効率以外に、管理責任やビジネスユニットの割り当てなど、特定のエージェントとサーバーをグループ化する組織上の理由が考えられます。

- **ホットバックアップ。** 特定のマシンがフェイルオーバーの目的で特に必要でない限り、エージェント負荷を割り当てない必要がある場合があります。この場合、すべてのエージェントを利用可能なサーバーのサブセットにアフィニティーを割り当てる必要があります。一部のサーバーは、通常の操作で関連付けられたエージェントなしで残す必要があります。

バグを報告します。

5.1.5. エージェントおよびサーバーフェイルオーバー

そのエージェントで利用可能なサーバーを特定するために各エージェントに提供されるサーバーの中央リストがあります。これは **フェイルオーバーの一覧** です。新しいサーバーがクラウドに参加すると、そのサーバーがリストに追加され、リストがエージェントに更新されます。

エージェントの一覧にあるサーバーが、その **プライマリーサーバー** と最も頻繁に通信します。エージェントがそのサーバーに接続できない場合は、利用可能なサーバーを見つけるまで、リスト内の次のサーバーへの接続を試みます。

エージェントは定期的に（時間ごとに）定期的にチェックし、プライマリーサーバーがオンラインに戻るかどうかを確認し、復元するとすぐにそのサーバーに切り替えます。

エージェントの定期的なディストリビューションでは、エージェントは、提供されたフェイルオーバーリストに応じて、利用可能なすべてのサーバーを（比較的）ランダムな順序で実行します。エージェントがアフィニティーグループに属する場合は、最初にそのアフィニティーグループ内のすべてのサーバーを試行し、フェイルオーバー一覧で設定される順序でアフィニティーグループ外のサーバーに移動します。

バグを報告します。

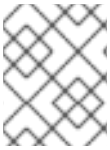
5.2. アフィニティーグループの作成

Affinity Group は、JBoss ON エージェントが管理する JBoss ON サーバーの設定を設定します。アフィニティーグループは、絶対要件ではなく、エージェントを管理するサーバーのみを設定します。すべてのエージェントは JBoss ON サーバークラウド内で管理されるため、JBoss ON サーバーは現在の負荷とパフォーマンスに基づいて JBoss ON エージェントを理論的に管理できます。

重要

エージェントには、高可用性のアフィニティーのみが優先されます。つまり、エージェントには、通信を試みるサーバーが優先されることを意味します。JBoss ON は双方向通信を使用するため、サーバーもエージェントと通信します。サーバーは、パーティションやエージェントアフィニティー設定に関係なく、そのエージェントのアフィニティーグループにサーバーがない場合やサーバーがエージェントを管理しない場合でも、JBoss ON のエージェントと通信できます。

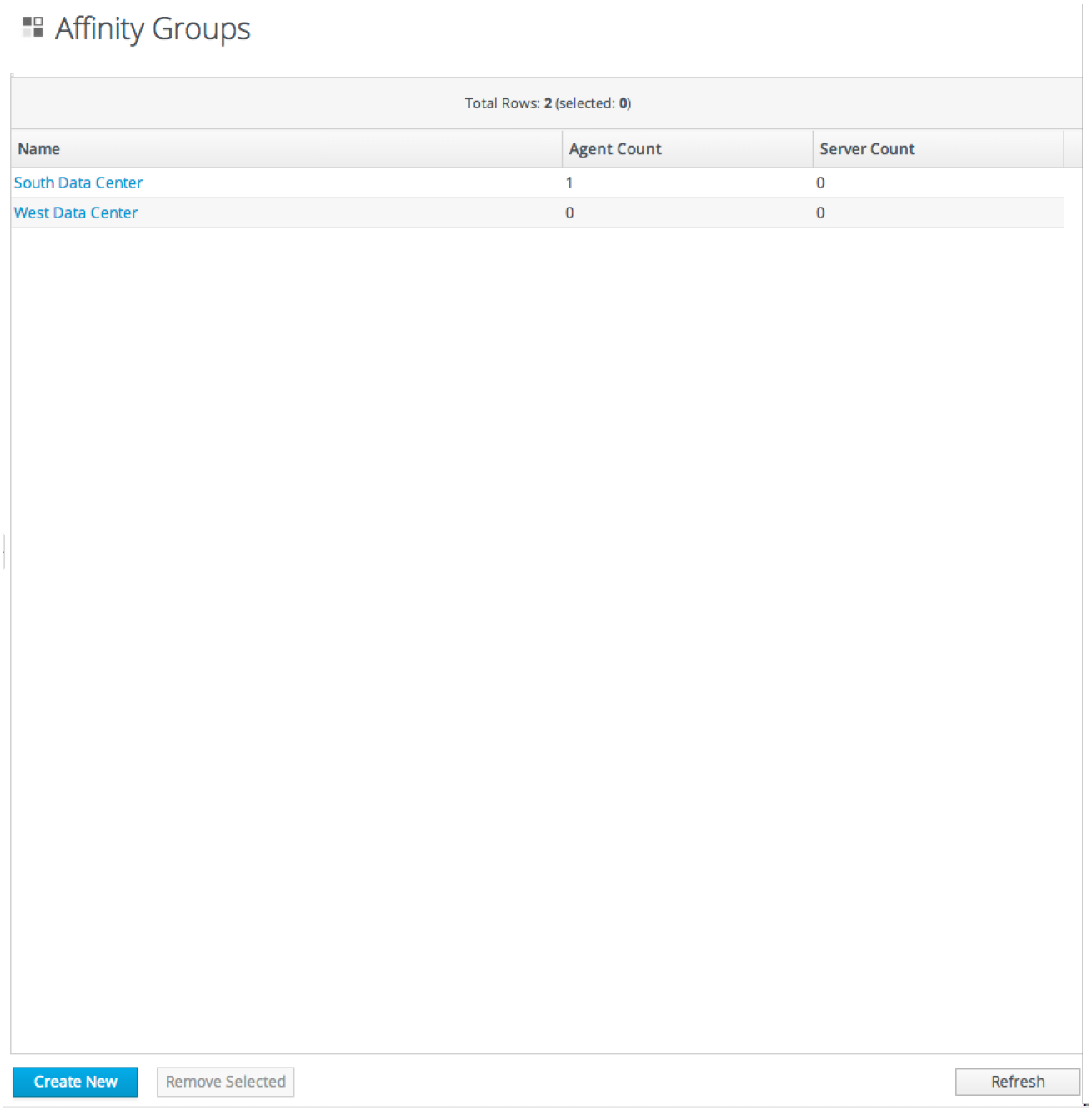
アフィニティーグループページには、各アフィニティーグループに割り当てられたエージェントおよびサーバーの数が表示されます。



注記

エージェントとサーバーは単一のアフィニティーグループにのみ属することができます。

図5.5 アフィニティーグループの一覧表示

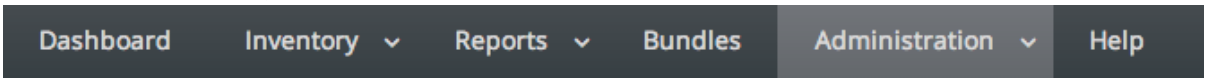


手順5.1 アフィニティーグループの作成

注記

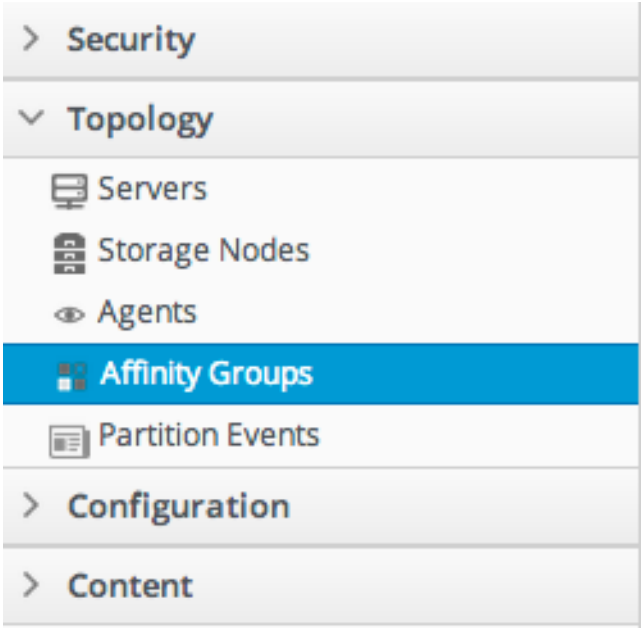
アフィニティーグループを編集するには、その名前をクリックして、新しいアフィニティーグループの作成と同じようにします。

1. 「」をクリックします。 **Administration**



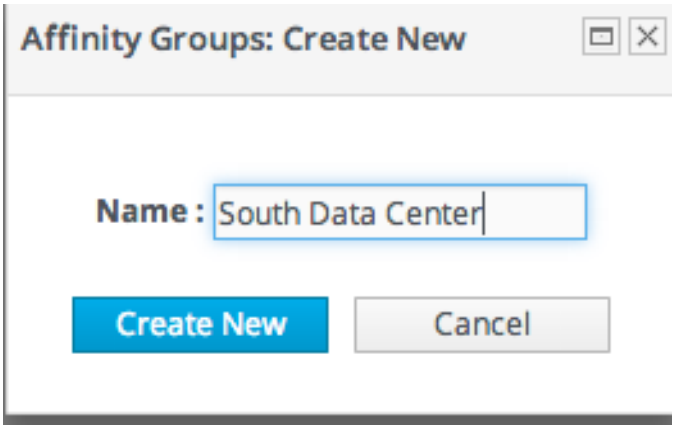
トップメニューのタブ。

2. をクリックし ます。

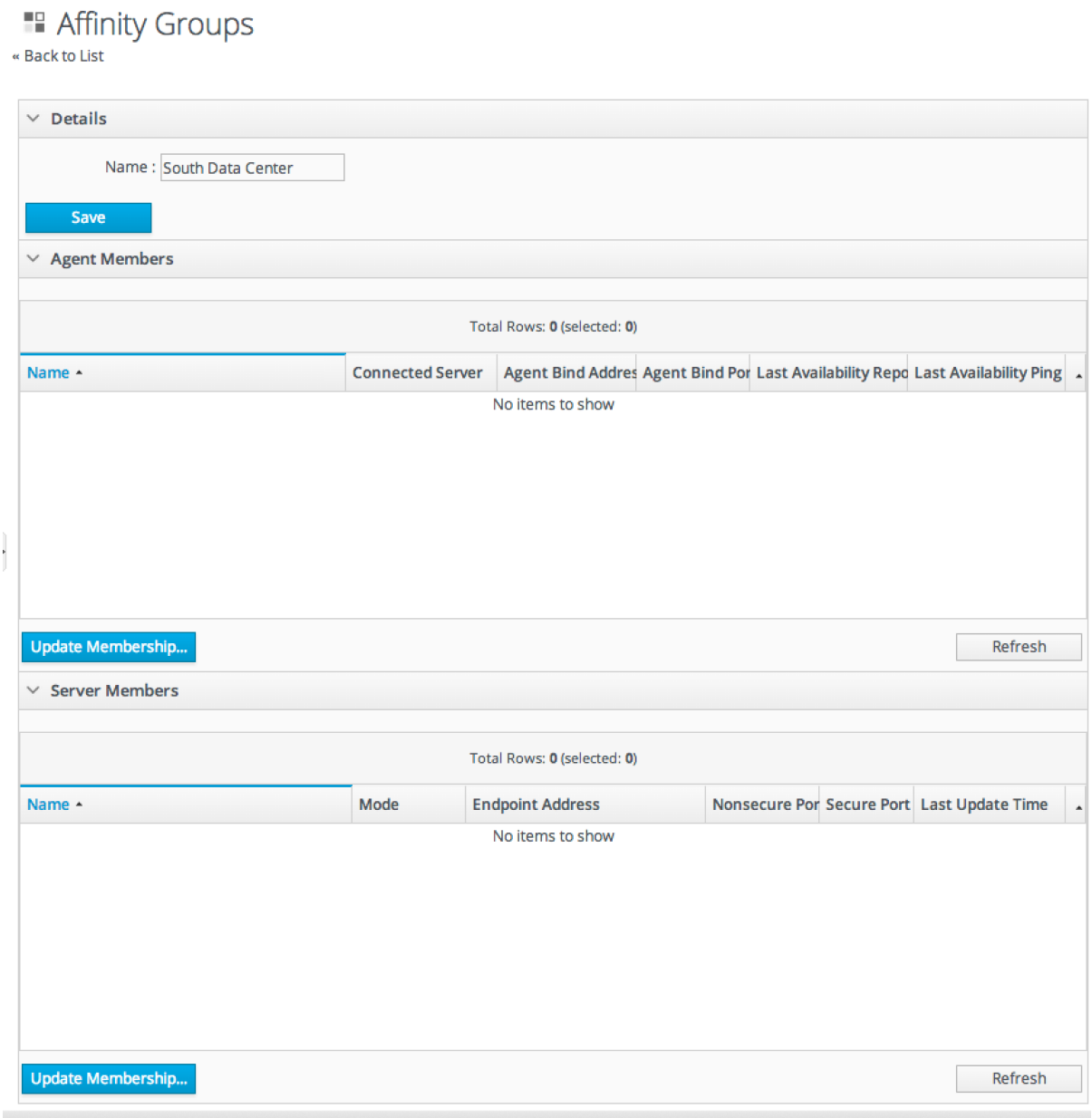


3. **Create New** ボタンをクリックします。

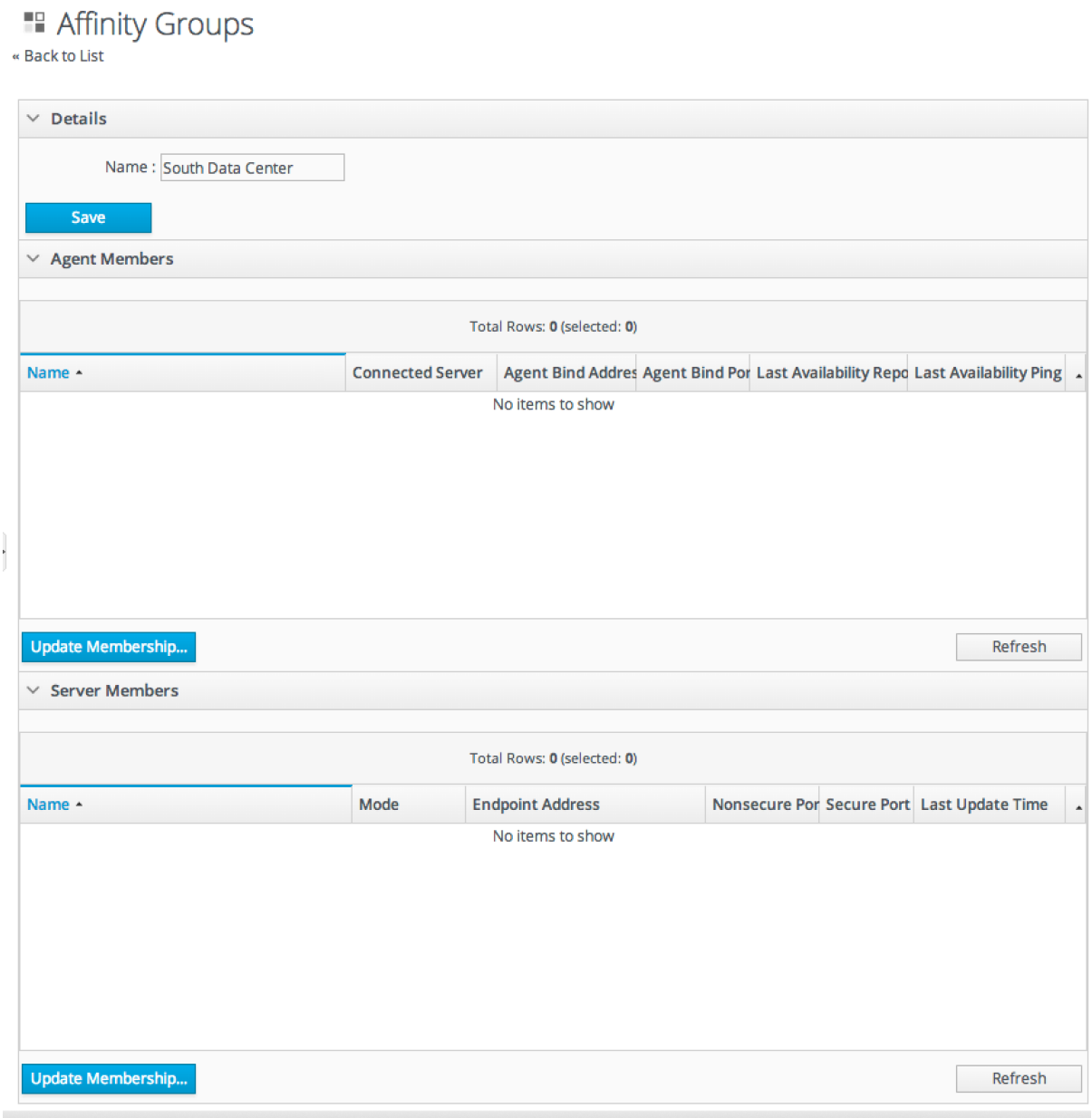
4. 新しいアフィニティーグループの名前を入力し、をクリックし **Create New**ます。



5. **Agent Members** グループで、エージェント名のチェックボックスをクリックしてグループに追加し **Update Membership**ます。



6. **Server Members** グループで、サーバー名のチェックボックスをクリックしてグループに追加し、をクリックし **Update Membership**ます。



7. をクリックし **Return to Affinity Group Link**ます。

   サーバーとエージェントの両方がアフィニティーグループに追加されると、グループが完全に設定されます。

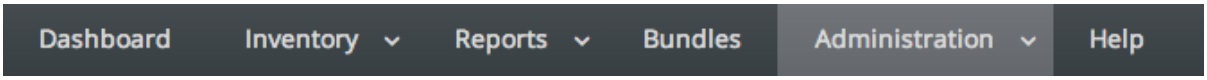
バグを報告します。

5.3. サーバーをメンテナンスモードに

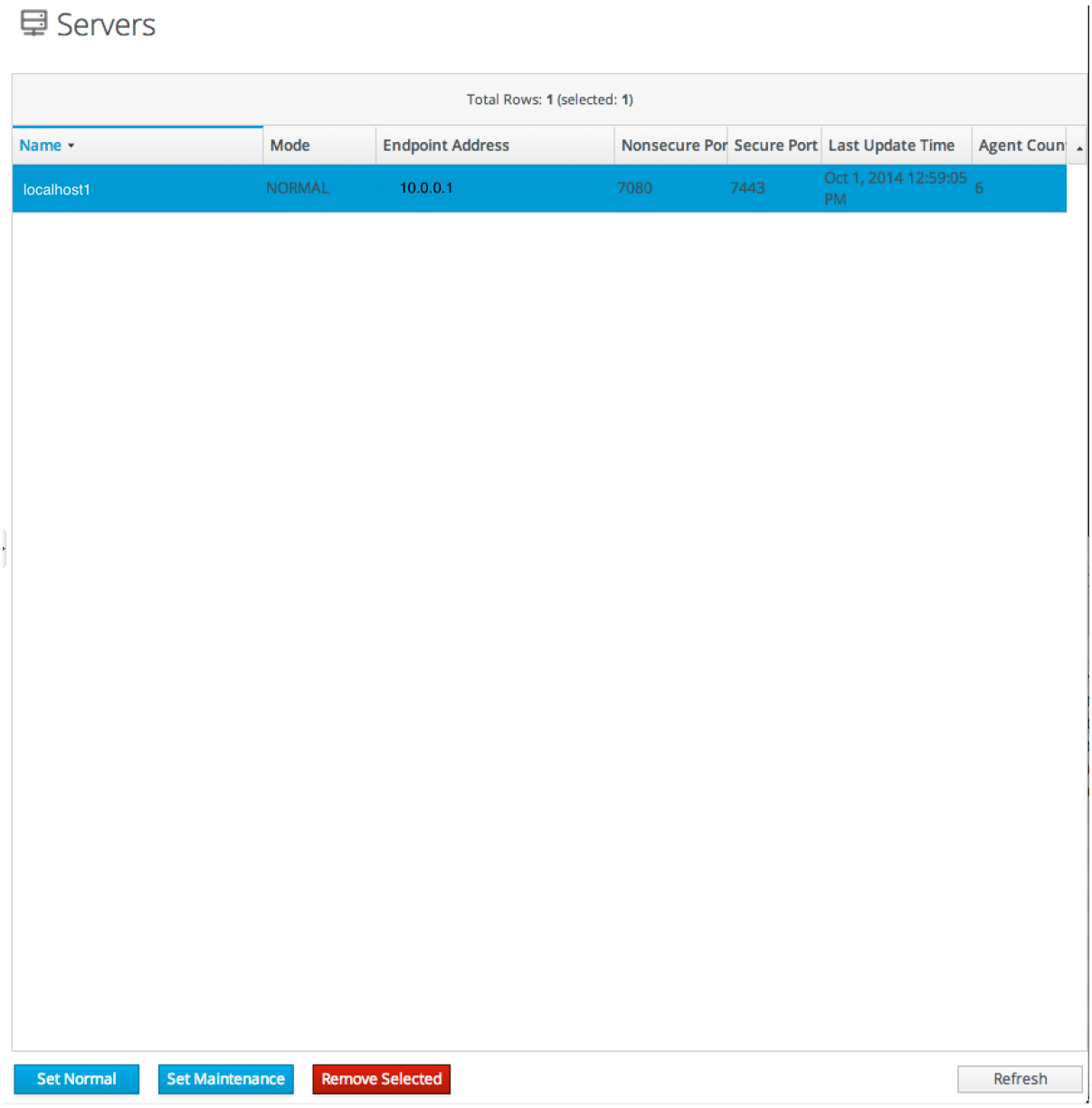
JBoss ON サーバーをメンテナンスモードにすると、一時的に高可用性クラウドからサーバーが削除されるため、エージェントの操作は処理されなくなります。これは、アップグレード時にサーバーがオフラインの場合や、サービスが中断した場合に役立ちます。

手順5.2 サーバーでのメンテナンスモードの設定

1. トップメニューの **Administration** タブをクリックします。



2. をクリックし ます。
3. 行をクリックして JBoss ON サーバーを選択し、メンテナンスモードに切り替えます。
4. **Set Maintenance** ボタンをクリックします。



JBoss ON サーバーは、**SET NORMAL** ボタンをクリックして高可用性クラウドに戻すことができます。エージェントは、段階的にサーバーに移行します。

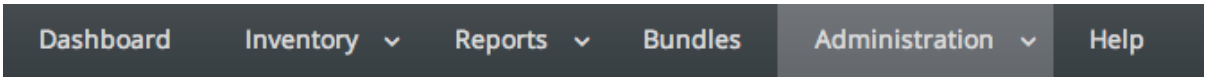
バグを報告します。

5.4. 高可用性クラウドからのサーバーの削除

メンテナンスモードの JBoss ON サーバーは、高可用性クラウドから永続的に削除できます。

手順5.3

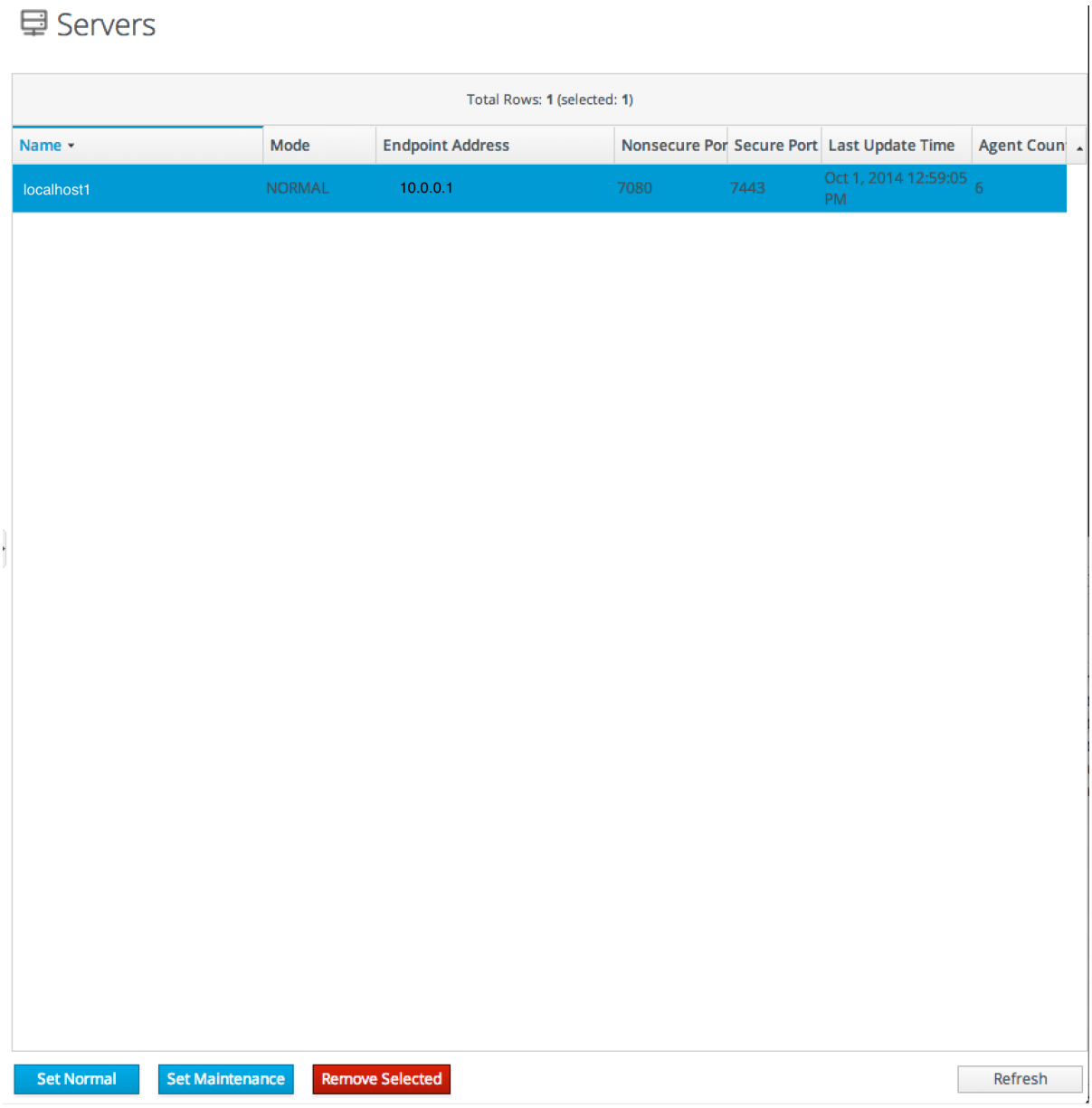
1. トップメニューの **Administration** タブをクリックします。



2. をクリックし ます。

3. 削除する JBoss ON サーバーの行を強調表示し、をクリックしてクラウドから **Set Maintenance** 削除します。

4. 画面を更新すると、をクリックし、削除する JBoss ON サーバーの行を強調表示し、をクリックし **Remove Selected**ます。



バグを報告します。

5.5. パーティションイベントの管理

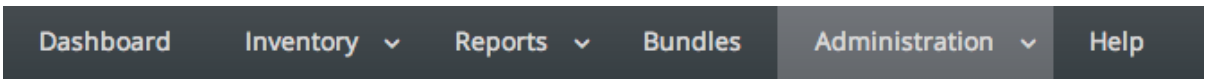
エージェントがサーバーによって管理されるように割り当てられると、**パーティション** になります。**パーティションイベントは**、パーティション設定に変更が加えられるたびに発生するログメッセージのようになります。

バグを報告します。

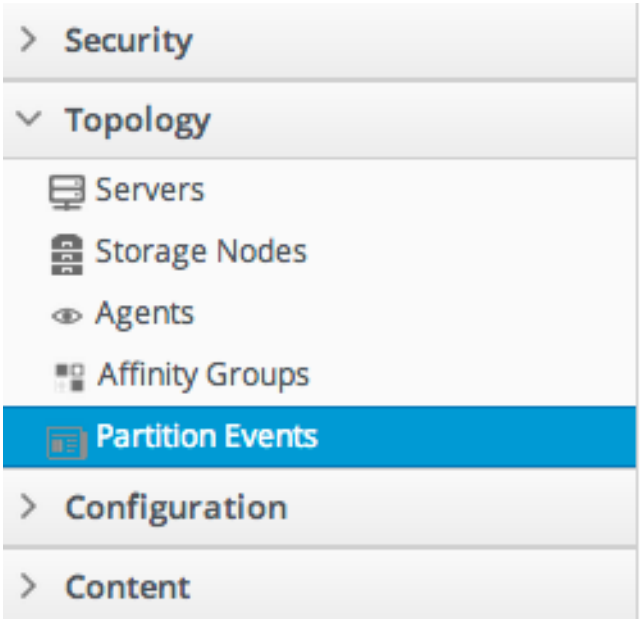
5.5.1. パーティションイベントの表示

パーティションイベントログは、高可用性設定でアクセスされます。

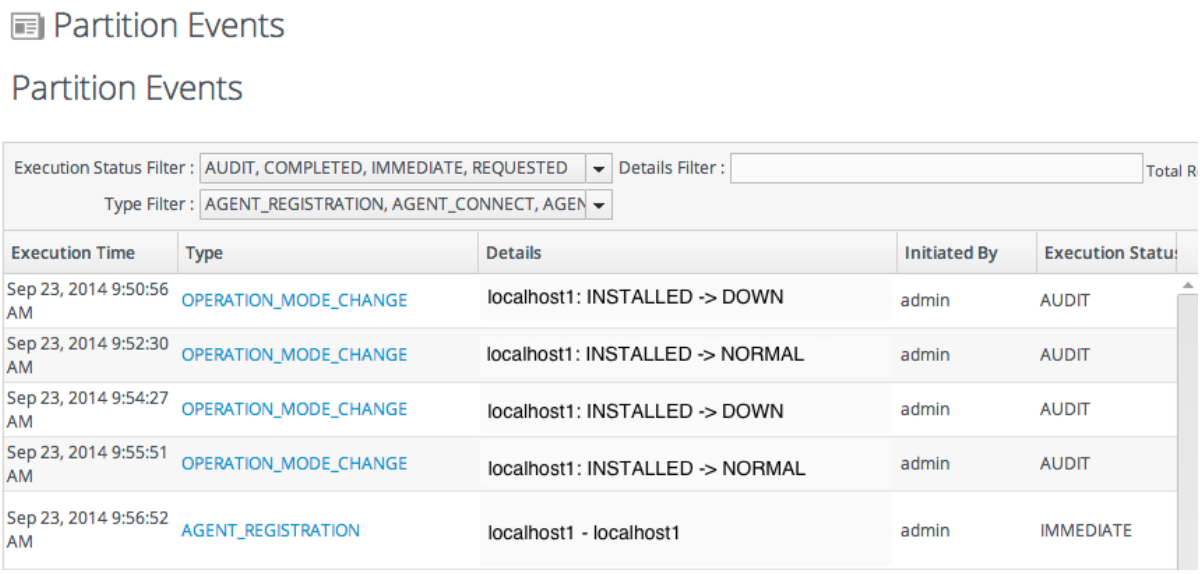
1. トップメニューの **Administration** タブをクリックします。



2. 左側の **Topology** メニューテーブルで、**Partition Events** 項目を選択します。



3. パーティションイベントページには、記録されたすべてのイベントが表示されます（表 5.2「パーティションイベントエントリー」 では、別のフィールドを説明します）。 パーティションイベントのタイプ名をクリックすると、パーティションの割り当て方法に関する詳細情報を含むレコードが開きます。



パーティションイベントログをフィルタリングして、イベントレコードのタイプ、ステータス、または詳細に一致するエントリーを表示できます。

記録されるパーティションイベントには、基本的に 4 つのカテゴリーがあります。

- アフィニティーグループの変更
- エージェントイベント
- サーバーイベント
- パーティションの変更

記録されたすべてのイベントがに一覧表示され 表5.1「パーティションイベントのタイプ」 ます。

表5.1 パーティションイベントのタイプ

パーティションイベント	description
アフィニティーグループの変更	
AFFINITY_GROUP_CHANGE	アフィニティーグループのエージェントまたはサーバー割り当ての変更を登録するか、またはグループが追加されたことを登録します。
AFFINITY_GROUP_DELETE	JBoss ON 設定から削除されたアフィニティーグループを登録します。
AGENT_AFFINITY_GROUP_ASSIGN	エージェントがアフィニティーグループに追加されたことを登録します。
AGENT_AFFINITY_GROUP_REMOVE	エージェントがアフィニティーグループから削除されたことを登録します。
SERVER_AFFINITY_GROUP_ASSIGN	サーバーがアフィニティーグループに追加されたことを登録します。
SERVER_AFFINITY_GROUP_REMOVE	サーバーがアフィニティーグループから削除されたことを登録します。
エージェントイベント	
AGENT_CONNECT	は、JBoss ON エージェントが起動したことを示しています。
AGENT_SHUTDOWN	は、JBoss ON エージェントが停止したことを示しています。
AGENT_LEAVE	は、JBoss ON エージェントが JBoss ON 設定から永続的に削除されたことを示しています。
AGENT_REGISTRATION	新しい JBoss ON エージェントが JBoss ON 設定に追加されました。
サーバーイベント	

パーティションイベント	description
SERVER_DELETION	は、JBoss ON サーバーが JBoss ON 設定から永続的に削除されたことを示しています。
SERVER_COMPUTE_POWER_CHANGE	
OPERATION_MODE_CHANGE	は、サーバーが停止するか、起動しているか、または新規インストールされたことを示しています。タイプは、モード（など **server.example.com: DOWN --> NORMAL**）による移行方法も示します。
パーティションの変更	
SYSTEM_INITIATED_PARTITION	は、JBoss ON がサーバーの再パーティションを開始したことを示しています。
ADMIN_INITIATED_PARTITION	は、JBoss ON ユーザーがサーバーの再パーティションを開始したことを示しています。

表5.2 パーティションイベントエントリー

フィールド	description
実行時間	パーティションイベントの時間。
type	は、パーティションイベントタイプを表示します。これは、エージェントパーティションに影響するシステムで発生したことを示します。
詳細情報	パーティションイベントに関する詳細情報を提供します。指定されている情報は、パーティションイベントタイプによって異なります。
開始元	は、パーティションイベントの生成アクションを開始したユーザーの名前を示しています。JBoss ON サーバーによって開始されたイベントは、によって開始されたことを示してい **admin**ます。

フィールド	description
実行状態	は、ステータスの説明の低さを示しています。ステータスの 4 つの設定があります。 ● **Audit** は、変更が加えられたが、パーティショントポロジーに影響するイベントではないことを示しています。 ● **即時**、パーティションがイベント時に変更されたことを示します。 ● **Requested** は、パーティションの変更が要求され、JBoss ON サーバークラウドジョブの次回実行されるまで遅延されたことを示しています（通常は 1 分）。パーティションの再パーティション要求は、通常、要求されたステータスを持ちます。 ● 変更が**完了し** たことを示します。

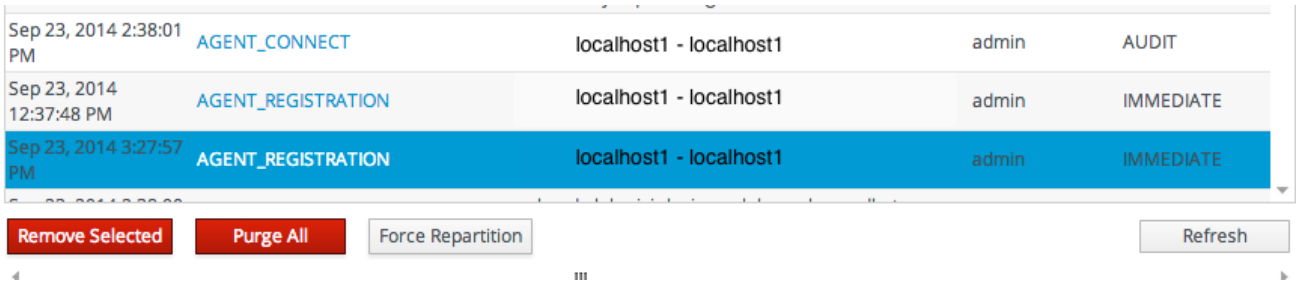
バグを報告します。

5.5.2. パーティションイベントの削除

パーティションイベントレコードを削除する方法は 2 つあります。

- 個々のレコードを選択して、 **REMOVE SELECTED**
- をクリックしてすべてのイベント **PURGE ALL** を削除します。

図5.6 パーティションイベントの削除



バグを報告します。

第6章 サーバーの設定

JBoss ON の設定は、設定に応じて 2 つの項目のいずれかで編集されます。

- JBoss ON GUI での

注記

JBoss ON UI で編集できる設定は JBoss ON UI で編集 **する必要** があります。

- **rhq-server.properties** 設定ファイルで

その他の設定は、JBoss ON サーバーで使用されるデータベースバックエンドに保存されます。

バグを報告します。

6.1. JBOSS ON サーバーのデバッグロギングの有効化

デバッグモードは、サーバーログへの開始スクリプトによって発生または発生したデバッグメッセージを記録します。

デバッグメッセージはログファイルにあり ***serverRoot*/jon-server-3.3.0.GA/logs/server.log**ます。

バグを報告します。

6.1.1. デバッグ環境変数の設定

場合によっては、JBoss ON サーバーランチャースクリプトからデバッグメッセージが必要な場合もあります。これには、環境変数を設定する必要があります。 **RHQ_SERVER_DEBUG** 任意の値を設定します。ランチャーの開始時にこの変数を設定すると、スクリプトはデバッグメッセージを出力します。

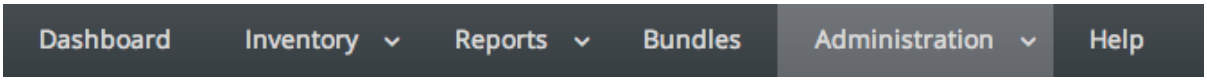
デバッグロギングを有効にする最も簡単な方法は、 **RHQ_SERVER_DEBUG** サーバーを起動する前の値への環境変数。

バグを報告します。

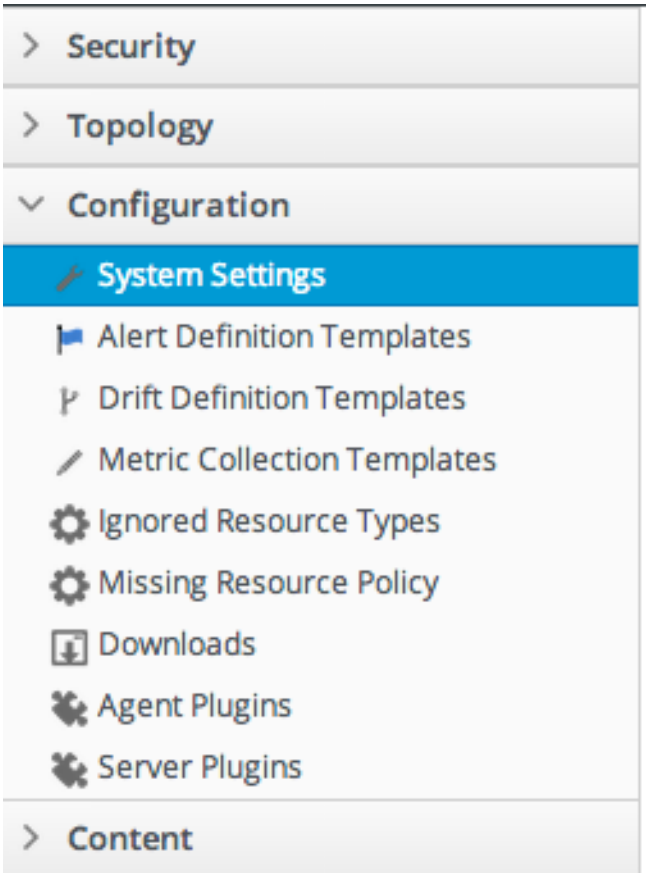
6.1.2. 現在のサーバー状態のログへのダンプ

サーバー設定の現在の状態の記録があると、デバッグや監査に役立ちます。ビルド番号、データベース情報、測定スケジュールなどの現在のサーバー詳細は、即座にサーバーログにエクスポートできます。

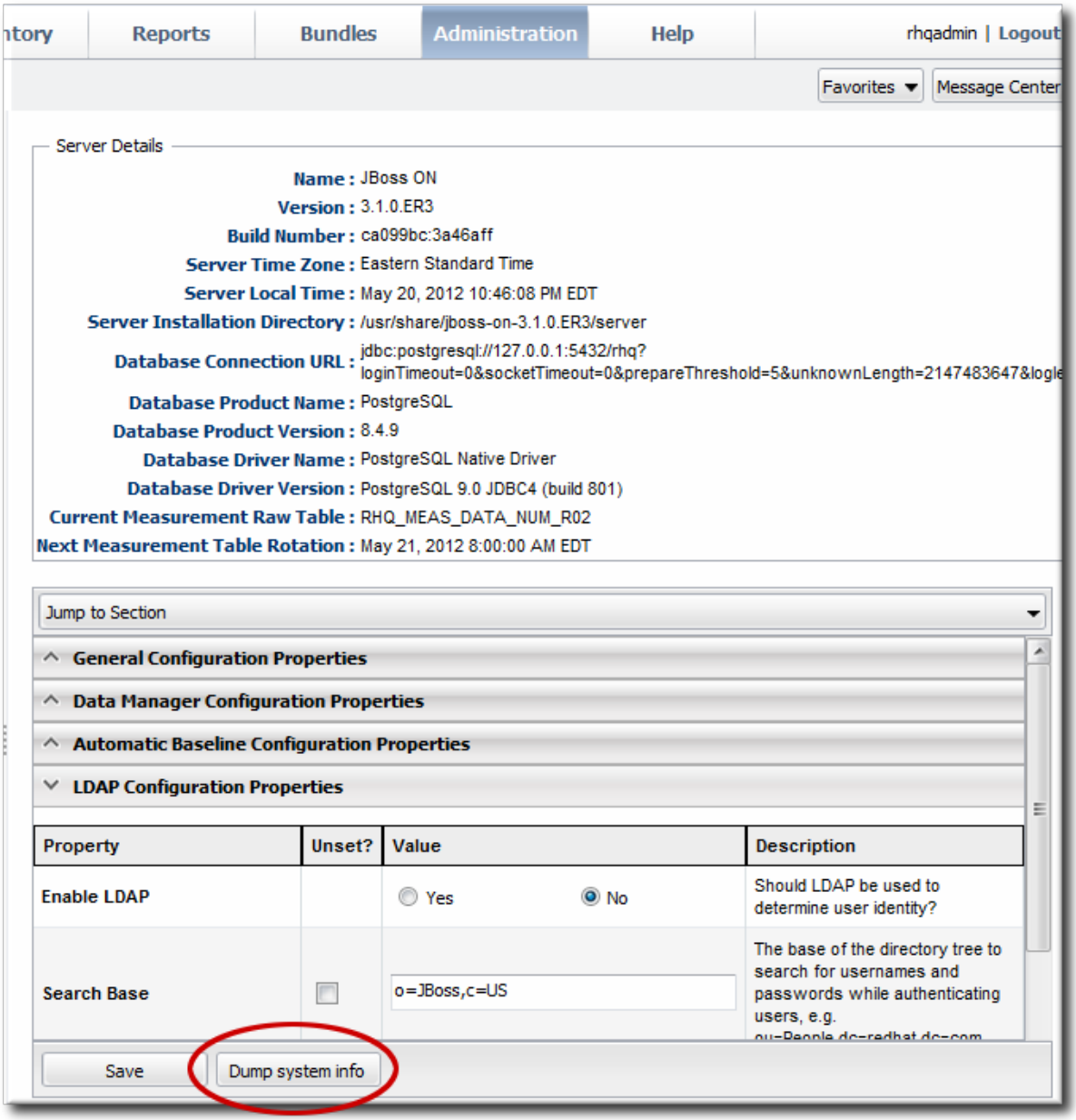
1. トップメニューの **Administration** タブをクリックします。



2. 左側の **Configuration** メニューテーブルで、**System Settings** 項目を選択します。



3. メインウィンドウでサーバー設定の下部までスクロールし、**Dump system info** ボタンをクリックします。



4. 現在のサーバー設定と詳細はすべてサーバーログに出力されます。

```
2012-05-14 19:44:28,587 INFO  [SystemInfoManager] SystemInformation: ********
CAM_LDAP_BIND_PW: [- non null -]
AlertDefinitionCount: [2]
CAM_LDAP_BASE_DN: [o=JBoss,c=US]
AVAILABILITY_PURGE: [31536000000]
CAM_JAAS_PROVIDER: [false]
BuildNumber: [ca099bc:3a46aff]
ServerCount: [27]
DATABASE_DRIVER_NAME: [PostgreSQL Native Driver]
RESOURCE_GENERIC_PROPERTIES_UPGRADE: [false]
... 8< ...
```

バグを報告します。

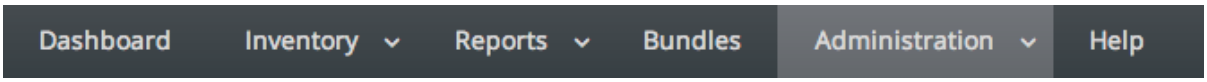
6.2. JBOSS ON SERVER の URL の変更

サーバー URL は、以下の 2 つの方法でサーバーを特定し、接続するために使用される URL です。

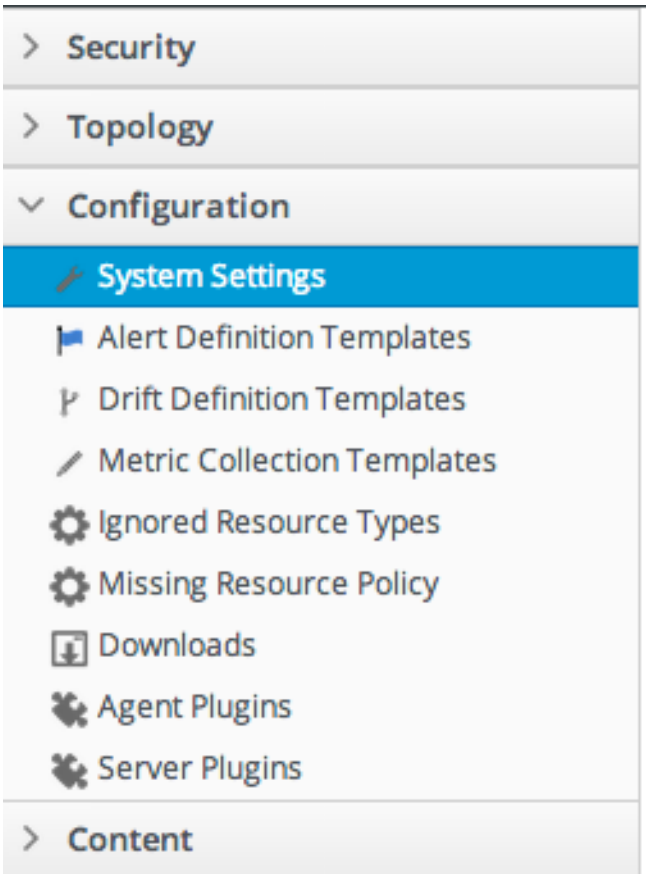
- GUI への接続
- アラートの詳細（アラートのメール通知に含まれる）

JBoss ON がプロキシーサーバーを介してインターネットに接続しない限り、サーバー URL を変更する必要はありません。

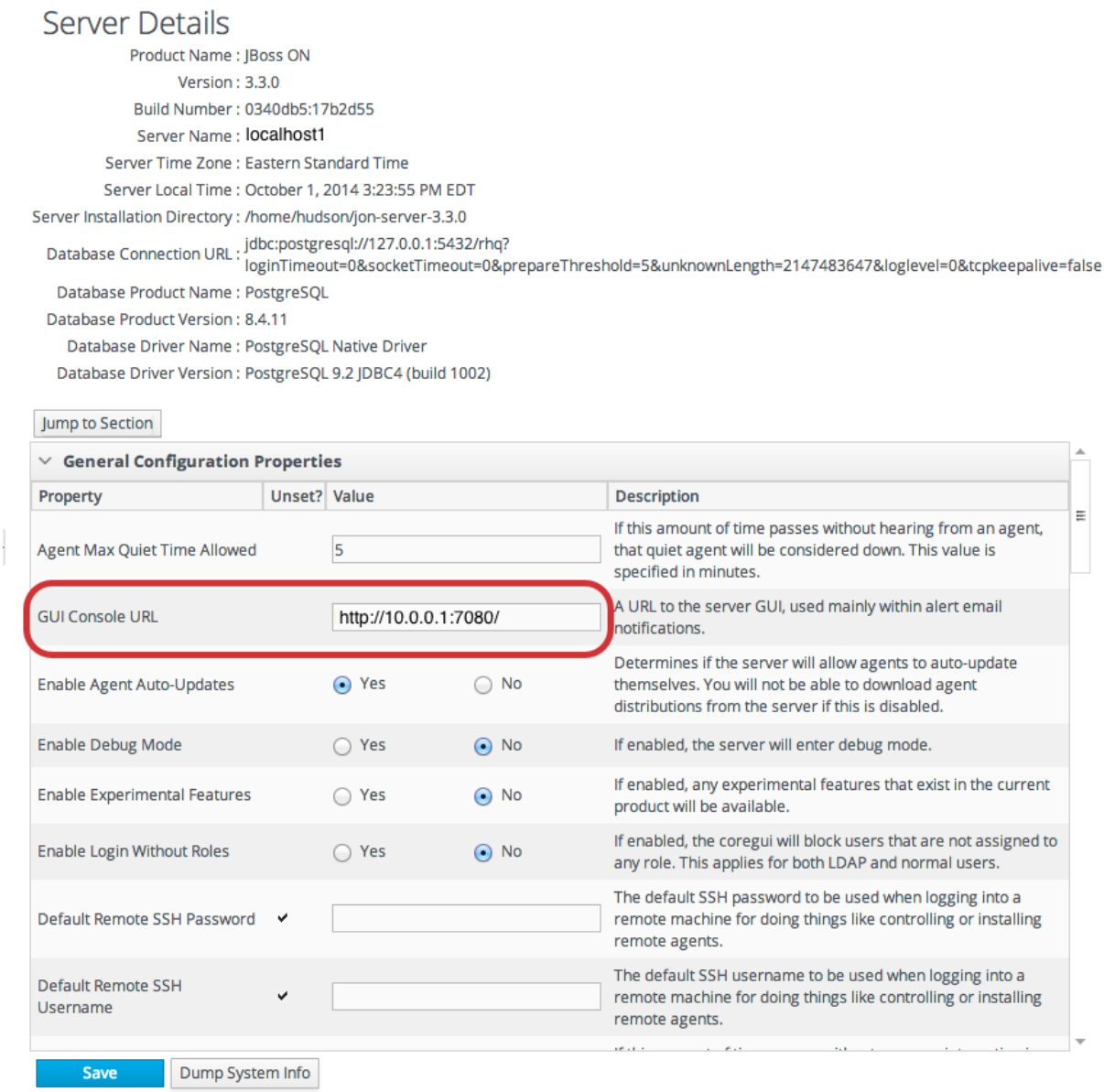
1. トップメニューの **Administration** タブをクリックします。



2. 左側の **Configuration** メニューテーブルで、**System Settings** 項目を選択します。



3. メインの作業領域の **JON General Configuration Properties** セクションまでスクロールします。
4. **GUI Console URL** フィールドのホスト名または IP アドレスを変更します。



5. をクリックし **Save**ます。

バグを報告します。

6.3. RHQ-SERVER.PROPERTIES での JBOSS ON SERVER 設定の編集

JBoss ON サーバー設定プロパティーは、***serverRoot*/jon-server-3.3.0.GA/bin** ディレクトリー内の **rhq-server.properties** 設定ファイルまたは JBoss ON データベースのいずれかに保存されます。設定ファイルには、リッスンする TCP/IP ポートやホスト名や IP アドレスなど、JBoss ON サーバーの基本情報の多くが含まれています。

サーバーの起動時に JBoss ON サーバー設定が **rhq-server.properties** ファイルからロードされます。初期設定は、JBoss ON プログラムの設定時にインストーラーによって定義されます。

注記

JBoss ON サーバーの起動 **rhq-server.properties** 時に設定プロパティーがロードされるため、新しい設定がロードされるように、設定ファイルの変更後に JBoss ON サーバーを再起動する必要があります。

バグを報告します。

6.3.1. インストール時のプロパティー設定

サーバー名、ポート、使用するデータベースなどのサーバープロパティーを定義する方法は、サーバーインスタンスの設定時に設定されます。

データベースプロパティーの変更については、を参照してください *8章JBoss ON に関連付けられたデータベースの管理*。

title

データベースタイプ

これにより、JBoss ON サーバーで使用されるデータベースのタイプまたはベンダーが設定されます。PostgreSQL または Oracle のいずれかになります。

データベース接続 URL

これにより、データベースへの接続時に JBoss ON サーバーが使用する JDBC URL（例：**jdbc:postgresql://localhost:5432/rhq、jdbc: oracle:oci:@localhost:1521:orcl）が提供されます。**

データベース JDBC ドライバークラス

これにより、JBoss ON サーバーがデータベースとの通信に使用する JDBC ドライバーの完全修飾クラス名（例： **oracle.jdbc.driver.OracleDriver** ）が提供されます。

データベース XA データソースクラス

これにより、JBoss ON サーバーがデータベースとの通信に使用する JDBC ドライバーの完全修飾クラス名（例： **org.postgresql.xa.PGXADataSource または oracle.jdbc.xa.client.OracleXADatasource）が提供されます** 。

データベースユーザー名

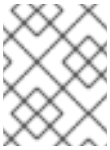
これにより、データベースにログインする際に JBoss ON サーバーが使用するユーザーの名前が指定されます。このユーザーはすでにデータベースに存在している必要があります。基本的なインストール手順によると、これは特別に作成された **rhqadmin** ユーザーです（JBoss ON のスーパーユーザーには関連しません）。

データベースのパスワード

これにより、データベースにログインする際に JBoss ON サーバーによって使用されるデータベースユーザーのパスワードが提供されます。このパスワードは、**rhq-server.properties** ファイルのハッシュに保存されます。データベースにパスワードを変更すると、手動でパスワードをハッシュして **rhq-server.properties** ファイルにコピーする必要があります。これは、で説明してい 「データベースパスワードの変更」 ます。

サーバー名

これにより、JBoss ON サーバーの一意の名前が設定されます。デフォルトはシステムのホスト名ですが、JBoss ON サーバークラウド内で一意である限り、任意の文字列にすることができます。

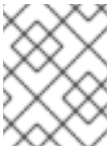


注記

他のサーバープロパティーとは異なり、これは JBoss ON UI でのみ管理され、その **rhq-server.properties** ファイルには管理されません。

サーバーのパブリックアドレス

これにより、サーバーに使用するパブリック IP アドレスが提供されます。このアドレスは、このサーバーへのアクセスを必要とするすべてのエージェントによって認識される必要があるアドレスです。デフォルトでは、ローカルホストのパブリック IP アドレスの値です。パブリック IP アドレスは HTTP/HTTPS ポートとともに使用され、エージェントの高可用性エンドポイントを提供します。



注記

他のサーバープロパティーとは異なり、これは JBoss ON UI でのみ管理され、その **rhq-server.properties** ファイルには管理されません。

HTTP ポート

これにより、JBoss ON GUI がセキュアでない HTTP リクエストをリッスンするポートが設定されます。これは、JBoss ON GUI URL のポート番号です（例： http :// localhost : 7080 ）。これは、高可用性のエンドポイントとして使用されるセキュアでないポートでもあります。

セキュアな HTTPS ポート

これにより、JBoss ON GUI がセキュアな HTTPS リクエストに対してリッスンするポートが設定されます。これは、JBoss ON GUI URL で使用されるポート番号です（例： https :// localhost : 7443 ）。これは、高可用性のエンドポイントとして使用されるセキュアなポートでもあります。

サーバーバインドアドレス

これにより、JBoss ON GUI コンソールの IP アドレス（他のサービスなど）がバインドされます。これは、のような JBoss ON GUI URL のホストです **server.example.com** **http://server.example.com:7080**。

Email SMTP ホスト名

これにより、JBoss ON サーバーによって使用される SMTP サーバーのホスト名が設定されます。電子メールは、主にアラート通知のために送信されます。

アドレスからのメール

これは、JBoss ON サーバーが送信したすべてのメールの **From** ヘッダーに使用するアドレスを設定します。

バグを報告します。

6.3.2. 通信の設定

JBoss ON サーバーは、サーバーおよびエージェントが接続する方法を定義して特定し、他のクライアント（JBoss ON GUI にアクセスするユーザーなど）がサーバーに接続できる方法を定義して、エージェントと通信するように設定されます。これらの通信エンドポイントには、サーバーのホスト名または IP アドレスの特定、異なる種類のメッセージに対してサーバーがリッスンするポート、サーバーへのアクセスに使用するプロトコルが含まれます。ユーザー設定可能なサーバー接続パラメーターはすべて、サーバープロパティー一覧に記載されています。

サーバーの接続またはトランスポートメソッドは、サーブレット（HTTP および HTTPS）またはソケット(HTTPS)を介して実装されます。**servlet** (HTTP)および **sslservlet** (HTTPS)は、JBoss ON サーバーに組み込まれた Tomcat サーバー経由でトラフィックを転送します。

重要

サーバーがトランスポートを使用している場合 **servlet** または **sslservlet**、エージェントの通信は Tomcat コネクター上で行われます。これ ***rhq.communications.connector.bind-port*** は無視され、Tomcat コネクターポートを使用してエージェントからサーバーにメッセージを送信します。デフォルトでは、Tomcat コネクターポートは 7080（サーブレット）または 7443(sslservlet)です。

注記

サーブレットベースのトランスポートは、Tomcat コネクターインフラストラクチャーを活用して、エージェントと GUI リクエストの両方を処理します。ただし、サーブレットを使用すると、エージェントと GUI クライアントが同じ接続メソッドを使用するように制限されます。証明書ベースの SSL 接続の場合、サーブレットでは、保存されたブラウザー証明書を使用してユーザーが GUI に対して認証する必要があります。SSL の場合は、エージェントと GUI セッションに異なる認証方法を可能にするソケット接続を使用することが推奨されます。

SSL ソケット「サーバーおよびエージェント間のクライアント認証の設定」 の設定については、を参照してください。

一般的な設定では、ユーザーがサーバーにアクセスするのに使用できるポート番号を設定します。

```
# General Properties
rhq.server.startup.web.http.port=7080
rhq.server.startup.web.https.port=7443
```

その他の接続設定を追加して、サーバーへの受信接続（エージェントからサーバーへのメッセージ）とアウトバウンド接続（サーバーからエージェントへのメッセージ）の SSL 接続を設定できます。例：

```
rhq.server.tomcat.security.client-auth-mode=true
rhq.server.tomcat.security.secure-socket-protocol=TLSv1,TLSv1.1,TLSv1.2
rhq.server.tomcat.security.algorithm=SunX509
rhq.server.tomcat.security.keystore.alias=RHQ
rhq.server.tomcat.security.keystore.file=conf/rhq.keystore
rhq.server.tomcat.security.keystore.password=RHQManagement
rhq.server.tomcat.security.keystore.type=JKS
rhq.server.tomcat.security.truststore.file=conf/rhq.truststore
rhq.server.tomcat.security.truststore.password=RHQManagement
rhq.server.tomcat.security.truststore.type=JKS
```

JBoss ON サーバー通信の 3 番目の部分では、相互との通信に使用する JBoss ON サーバーおよびエージェントの接続エンドポイントの情報に対する制御が強化されます。これらはサーバーの **トランスポート** パラメーターです。JBoss ON エージェントとポートはどちらも URL に似た呼び出し文字列を使用して、同じリモーティングフレームワークを使用します。これらの接続文字列の形式は以下のとおりです。

```
protocol://server:port/transportParm1=value1&transportParam2=value2
```

重要

ほとんどの通信では、JBoss ON サーバーはサーブレットプロトコルまたは sslservlet プロトコルを使用する必要があります。ソケットがトランスポートパラメーターを渡すことができる唯一のインスタンス。それ以外の場合は、ソケットおよび sslsocket はサポートされません。

トランスポート設定では、最初にサーバーとエージェントが通信に共同で定義し、使用するコネクター（**エンドポイント**と呼ばれる）を設定します。つまり、同じ情報がサーバーとエージェントの設定ファイルにある必要があります。リモーティング URL の各部分は、個別のサーバー設定パラメーターを使用して構築されます。

標準サーバー設定には、トランスポートメソッド、サーバー IP アドレス、エージェントポート、URL に追加するパラメーターの 4 つの部分があります。例：

```
rhq.communications.connector.transport=servlet
rhq.communications.connector.bind-address=192.168.1.2
rhq.communications.connector.bind-port=16163
rhq.communications.connector.transport-params=/jboss-remoting-servlet-
invoker/ServerInvokerServlet
```

この標準設定はマージされ、この URL を作成します。

```
servlet://192.168.1.2:16163/jboss-remoting-servlet-invoker/ServerInvokerServlet
```

登録の送信や可用性レポートなど、サーバーとの通信を送信する方法を認識できるように、同じエンドポイント定義を持つ対応するエントリーもエージェント設定に一覧表示されます。

```
RHQ Server IP Address=192.168.1.2
RHQ Server Port=16163
RHQ Server Transport Protocol=servlet
RHQ Server Transport Parameters=/jboss-remoting-servlet-invoker/ServerInvokerServlet
```

例6.1 基本的なサーバーエージェントトランスポートの例

IP アドレスが指定されたサーバー **192.168.0.10** は、16163 の標準エージェントポートを介してエージェントに接続します。サーバー設定には、サーバーアドレス(***rhq.communications.connector.bind-address***)、エージェント通信ポート（）、および接続プロトコル(***rhq.communications.connector.bind-portrhq.communications.connector.transport***)を定義するために、以下の設定があります。

```
rhq.communications.connector.transport=servlet
rhq.communications.connector.bind-address=192.168.0.10
rhq.communications.connector.bind-port=16163
rhq.communications.connector.transport-params=enableTcpNoDelay=true&backlog=200
```

コネクション URL は以下のようになります。

```
servlet://192.169.0.10:16163/enableTcpNoDelay=true&backlog=200
```

JBoss ON エージェント設定には、サーバー設定に一致する対応するエントリーが含まれます。

```
RHQ Server IP Address=192.168.0.10
```

```
RHQ Server Port=16163
RHQ Server Transport Protocol=socket
RHQ Server Transport Parameters=enableTcpNoDelay=true&backlog=200
```

JBoss ON サーバーがメッセージを処理する方法により、トランスポートパラメーターは受信メッセージと送信メッセージの両方の関連情報を渡できます。これらのトランスポートパラメーターは、標準の Web URL パラメーターと同様に、ampersands(&)とともに strung になります。

server および client トランスポートパラメーターはどちらも同じ URL に渡されます。JBoss ON サーバーは、現在の操作に関連するパラメーターのみを処理します。たとえば 例6.1「基本的なサーバーエージェントトランスポートの例」、設定は 2 つのトランスポートパラメーター **enableTcpNoDelay**（クライアント）と **backlog** （サーバー）を設定します。JBoss ON サーバーがメッセージを受信した場合（通信サーバーとして機能する場合）、**backlog** パラメーターを使用し、ignore **enableTcpNoDelay** は送信（クライアント）メッセージにのみ実行される **enableTcpNoDelay** ため無視します。

rhq-server.properties の一般的なサーバー接続パラメーター

jboss.bind.address [1][2]

バインドする JBoss ON GUI コンソールの IP アドレス（他のサービスなど）を指定します。これは、のような JBoss ON GUI URL のホストです **server.example.com** **http://server.example.com:7080**。

rhq.server.startup.web.http.port [1][2]

JBoss ON GUI がセキュアでない HTTP リクエストをリッスンするポートを指定します。これは、JBoss ON GUI URL のポート番号です（例： http :// localhost : 7080 ）。これは、高可用性のエンドポイントとして使用されるセキュアでないポートでもあります。

rhq.server.startup.web.https.port [1][2]

JBoss ON GUI がセキュアな HTTPS リクエストをリッスンするポートを提供します。これは、JBoss ON GUI URL のポート番号です（例： https :// localhost : 7443 ）。これは、高可用性のエンドポイントとして使用されるセキュアなポートでもあります。

rhq.server.startup.keystore.filename [2]

JBoss ON GUI は HTTPS 経由のブラウザーリクエストを受け入れることができます。JBoss ON GUI をリモートブラウザーに認証する場合は、SSL 証明書をキーストアファイルに配置する必要があります。この設定はキーストアファイルの場所を参照します。このファイル名は、***<JBoss ON server Install Dir>*/jbossas/standalone/configuration** ディレクトリーに対する相対パスである必要があることに注意してください。JBoss ON サーバーには、デフォルトのキーストアファイルに自己署名証明書が同梱されます。

rhq.server.startup.keystore.password [2]

キーストアファイルのパスワード。これは、JBoss ON GUI がキーストアファイルにアクセスして、証明書をブラウザークライアントに提供できるようにするためです。

rhq.server.startup.keystore.sslprotocol [2]

ブラウザーが JBoss ON GUI との通信に使用するプロトコル。

rhq.server.maintenance-mode-at-start

サーバーをメンテナンスモード(true)で起動するか、またはシャットダウン時(false)モードでサーバーを起動するかどうかを設定します。デフォルトは false です。

rhq-server.properties SSL サーバー接続パラメーター

rhq.communications.connector.security.secure-socket-protocol（エージェントからサーバーへ）, rhq.server.client.security.secure-socket-protocol（サーバーからエージェント）

この JBoss ON サーバーと通信する際にエージェントが使用する安全なプロトコル。

rhq.communications.connector.security.keystore.file(agent to server) , rhq.server.client.security.keystore.file（サーバーからエージェント）

JBoss ON サーバーをエージェントに対して認証する証明書が含まれるキーストアファイル。

rhq.communications.connector.security.keystore.algorithm (agent to server) , rhq.server.client.security.keystore.algorithm (server to agent)

rhq.communications.connector.security.keystore.type (agent to server) , rhq.server.client.security.keystore.type (server to agent)

keystore.para のファイル形式

rhq.communications.connector.security.keystore.password (agent to server) , rhq.server.client.security.keystore.password (server to agent)

キーストアファイルへのアクセスに使用されるパスワード。

rhq.communications.connector.security.keystore.key-password (agent to server) , rhq.server.client.security.keystore.key-password (server to agent)

キーストア内のキーへのアクセスを取得するために使用されるパスワード。

rhq.communications.connector.security.keystore.alias (agent to server) , rhq.server.client.security.keystore.alias (server to agent)

キーストア内の JBoss ON サーバーのキーを識別するエイリアス。

rhq.communications.connector.security.truststore.file（サーバーにエージェント）, rhq.server.client.security.truststore.file（エージェントへのサーバー）

この JBoss ON サーバーが信頼する証明書が含まれるトラストストアファイル。JBoss ON サーバーが JBoss ON エージェントを認証する必要がある場合、これを設定する必要があります。それ以外の場合は不要です。このトラストストアには、この JBoss ON サーバーと通信する必要があるすべての JBoss ON エージェントの証明書が含まれます。『着信クライアント認証モードを参照してください』。

rhq.communications.connector.security.truststore.algorithm (agent to server) , rhq.server.client.security.truststore.algorithm（エージェントへのサーバー）

rhq.communications.connector.security.truststore.type (agent to server) , rhq.server.client.security.truststore.type (server to agent)

トラストストアファイルのファイル形式。

rhq.communications.connector.security.truststore.password (agent to server) , rhq.server.client.security.truststore.password (server to agent)

トラストストアファイルへのアクセスに使用されるパスワード。

rhq.communications.connector.security.client-auth-mode（エージェントからサーバーへ）, rhq.server.client.security.server-auth-mode-enabled（エージェントへのサーバー）

JBoss ON サーバーがメッセージを送信する JBoss ON エージェントを認証する必要があるかどうかを示します。サーバーがセキュアな接続を使用し、トラストストアのすべての JBoss ON エージェントの信頼済み証明書がない場合、これを none に設定します。有効な値は、**true** またはです **false**。

- **rhq.communications.connector.security.client-auth-mode**: **none**、**want** または **need**
- **rhq.server.client.security.server-auth-mode-enabled**: **true** または **false**

rhq-server.properties トランスポート接続サーバーのパラメーター

rhq.communications.connector.transport

JBoss ON エージェントがメッセージを JBoss ON サーバーに転送する方法を定義します。許可される値は servlet または sslservlet です。エージェントリクエストは JBoss ON サーバー Web アプリケーション層（セキュアな Tomcat Connector など）を経由します。sslservlet では、エージェントが web アプリケーションレイヤーを通過するルートだけでなく、セキュアな Tomcat Connector 経由でセキュア化されます。受信エージェントメッセージ認証に使用されるキーストアは、で設定したものと同じです ***rhq.communications.connector.security.keystore.file***。

注記

このトランスポート設定は、エージェントがその接続方法のみを超えることを制限し**ませ** ん。デフォルトでは、JBoss ON サーバーはサーブレットトラフィックと sslservlet トラフィックの両方を許可する通信コネクターを常にデプロイします。この設定では、サーバーにメッセージを送信するときに使用するトランスポートを決定するようエージェントに指示します。サーバーがトランスポートをサーブレットに設定し、エージェントが sslservlet 経由でサーバーと通信するよう設定されている場合、エージェントが送信するメッセージは sslservlet 経由で行われます。

rhq.communications.connector.bind-address

これは、サーバーの JBoss/Remoting ロケーター URL に配置されるアドレスです。これは、JBoss ON サーバーがそのコネクターをバインドするエンドポイントを定義します。また、すべてのエージェントがサーバーへの接続に使用できるパブリックエンドポイントアドレスも表しています。

rhq.communications.connector.bind-port

JBoss ON サーバーがバインドするエンドポイントと、すべてのエージェントがサーバーへの接続に使用できるパブリックアドレスを定義します。これはインストーラーのビューからは表示されなくなりますが、これは **rhq-server.properties** ファイルに表示されます。この値は空白にすることができます。サーバーに設定されたトランスポートに応じて、サーバーは HTTP または HTTPS ポートのいずれかに設定します。

rhq.communications.connector.transport-params

追加のトランスポートパラメーターを定義します。JBoss ON サーバーは、JBoss ON エージェントからの受信メッセージを受け入れるコネクターに設定されます。可能なトランスポートパラメーターはすべてに記載されてい 表6.1「トランスポートパラメーター」 ます。

rhq.communications.multicast-detector.enabled

true の場合、JBoss ON サーバーは JBoss ON エージェントの自動検出を試行し、マルチキャスト検出を使用してオフラインに移行します。これを機能させるには、ネットワークがマルチキャストトラフィックをサポートする必要があります。

rhq.communications.multicast-detector.bind-address

マルチキャスト検出が直接バインドするアドレス。マルチキャスト検出を有効にしていない場合は、これは使用されません。

rhq.communications.multicast-detector.multicast-address

マルチキャスト検出がメッセージをブロードキャストするアドレス。マルチキャスト検出を有効にしていない場合は、これは使用されません。

rhq.communications.multicast-detector.port

マルチキャスト検出がメッセージをブロードキャストするポート。マルチキャスト検出を有効にしていない場合は、これは使用されません。

表6.1 トランスポートパラメーター

トランスポートパラメーター	description	着信メッセージまたは出力メッセージの場合
serverBindAddress	サーバーソケットがリクエストをリッスンするアドレス。デフォルトは空の値で、サーバーのソケットが InvokerLocator URL（ホスト）が提供するホストにバインドされることを示します。	incoming
serverBindPort	要求をリッスンするポート。	incoming
timeout	ソケットのタイムアウト値。サーバー側のデフォルトは 60000（1 分）です。timeout パラメーターを設定すると、その値はクライアント側にも渡されます（下記参照）。	incoming
backlog	指定のタイミングで許可される未承認着信接続の優先数。実際の数は指定されたバックログよりも大きくなる可能性があります。 キューが満杯になると、追加の接続リクエストが拒否されます。0 よりも大きい正の値である必要があります。0 以下の値が渡された場合、デフォルト値が想定されます。デフォルト値は 200 です。	incoming
numAcceptThreads	クライアント接続を受け入れるために存在するスレッドの数。デフォルトは 1 です。	incoming

トランスポートパラメーター	description	着信メッセージまたは出力メッセージの場合
maxPoolSize	クライアント要求を処理するサーバースレッドの数。デフォルトは 300 です。	incoming
socket.check_connection	クライアントからサーバーに単一のバイト ping を送信し、サーバーから接続を再度使用する前に、呼び出し元が接続の確認を試みるかどうかを示します。この設定は、クライアントとサーバーの両方で設定する必要があります。デフォルト値は false です。	incoming
clientConnectAddress	クライアントがサーバー側のソケットへの接続に使用する IP アドレスまたはホスト名。これは、クライアントが外部から別の IP アドレスまたはホスト名に送信されるルーターを通過する場合に必要です。clientConnectAddress または serverBindAddress が指定されていない場合は、ローカルホストのアドレスが使用されます。	送信
clientConnectPort	クライアントがサーバー側のソケットへの接続に使用するポート。これは、クライアントが外部に送信されたリクエストを内部的に異なるポートに転送するルーターを通過する場合に必要です。	送信
timeout	ソケットのタイムアウト値。クライアント側のデフォルトは 1800000（または 30 分）です。	送信
enableTcpNoDelay	クライアントソケットの TCP_NODELAY をオンまたはオフにするかを示します。TCP_NODELAY は特定の目的で使用されており、Nagle バッファーアルゴリズムを無効にします。これは、即時に応答を取らずに頻繁に発生する情報バーストルメントを送信するアプリケーションに対してのみ設定する必要があります。デフォルトは false です。	送信

トランスポートパラメーター	description	着信メッセージまたは出力メッセージの場合
clientMaxPoolSize	クライアント側の最大ソケット接続数。これは基本的に、ソケットクライアントインボーダーから実行できる同時クライアント呼び出しの最大数と同じです。デフォルトは 50 です。	送信
numberOfRetries	プールからソケットを取得する再試行回数。これは基本的に、タイムアウトが発生する前に、プールからクライアントソケット接続を取得するまでクライアントが待機する秒数と一致します。最大再試行に到達すると、CannotConnectException がスローされます。デフォルトは 30 です。	送信
numberOfCallRetries	呼び出しの再試行回数。numberOfRetries には関連性がありません。これは、プールからクライアントソケット接続をすでに受け取った後にこのプレイに関連します。ただし、プール内で待機中にソケット接続がタイムアウトになる可能性があります。接続チェックはデフォルトでは行われないため、接続が切断され、新しい接続の取得が試行されます。これは、numberOfCallRetries が多数ある場合に発生します（デフォルトは 3 に設定されます）。ただし、(numberOfCallsRetries - 2)に到達すると、接続プール全体がプールのすべての接続がタイムアウトし、無効であることが前提でフラッシュされ、新しい接続を作成して開始します。これに失敗すると、MarshalException がスローされます。	送信
socket.check_connection	クライアントからサーバーに単一のバイト ping を送信し、サーバーから接続を再度使用する前に、呼び出し元が接続の確認を試みるかどうかを示します。この設定は、クライアントとサーバーの両方で設定する必要があります。デフォルトでは false の場合に使用します。	送信

バグを報告します。

6.3.3. コンカレンシー制限の設定

JBoss ON は、多数のエージェントを処理できます（おそらく 100 個）。多くのエージェントが同時にサーバーと通信しようとすると、JBoss ON サーバーがメッセージでいっぱいになる可能性があります。これは、しばらく停止した後に JBoss ON サーバーが再起動した場合に発生する可能性があります。JBoss ON エージェントが JBoss ON サーバーが復帰したことを検出すると、すべてがメッセージのバックログの送信を試みます。

JBoss ON サーバーは、一度に処理できる同時メッセージ数に設定可能な制限を課して、サーバーのフラッドリスクを軽減できます。この制限を経過したメッセージはすべてドロップされ、エージェントは後で送信するよう要求されます。

コンカレンシー関連のパラメーターはすべてに記載されてい 表6.2「rhq-server.properties Parameters for Concurrency Limits」 ます。

同時実行制限は、エージェント接続の数を制限するだけでなく、GUI およびその他の Web 接続への接続数も制限します。コンカレンシー制限を制御する主なパラメーターは 3 つあります。

- サーバーへの受信メッセージの合計数に対するグローバル制限***rhq.communications.global-concurrency-limit***。

  許可されるエージェント接続の合計数です。特定のメッセージタイプには、その他の同時実行制限があります。これは、コンテンツのダウンロード、インベントリーの同期、その他のリソース集約型または再帰的なエージェント操作のパフォーマンスの調整に役立ちます。この同時実行制限は特定のメッセージタイプにのみ適用され、これらの制限は互いに独立して評価されます。グローバル同時実行制限は、**すべて** のエージェント接続の合計上限です。これは、他の同時実行制限の合計がより高い場合でも、効果的な同時実行制限です。

- 許可される同時 Web 接続の合計数の制限(***rhq.server.startup.web.max-connections***)。

  これにより、HTTP または HTTPS 接続を使用して JBoss ON サーバーに接続するクライアント接続がカウントされます。これには、当然ながら Web GUI 接続が含まれますが、（デフォルト） **servlet** または **ssslservlet** トランスポートを使用するすべてのエージェント接続も含まれます。

  Web 接続の制限は、セキュアでない HTTP リクエストと HTTPS リクエストの両方で同じですが、異なるプールに対して HTTP および HTTPS 接続数がカウントされるように、制限は加算されます。許可される最大接続の **合計** は、の ***rhq.server.startup.web.max-connections*** 値に関係なく実際には 2 倍になります。たとえば、300 を設定すると、合計 600 の同時 Web 接続に対して HTTP リクエストが許可され、300 HTTPS リクエストが許可されます。

- エージェント(***rhq.server.agent-downloads-limit***)および他のクライアント(***rhq.server.client-downloads-limit***)からダウンロードする回数に制限されます。

例6.2 コンカレンシー制限

```
rhq.server.startup.web.max-connections=200
rhq.server.agent-downloads-limit=45
rhq.server.client-downloads-limit=5
rhq.communications.global-concurrency-limit=30
```

表6.2 rhq-server.properties Parameters for Concurrency Limits

パラメーター	description
rhq.server.startup.web.max-connections	GUI への接続とエージェントによる接続の両方を含め、同時に作成できる Web 接続の数に制限を設定します。 **注記** エージェント要求が Web 接続上でルーティングされる場合、***rhq.communications.global-concurrency-limit*** 値が Web 接続の制限よりも若干小さいことを確認してください。それ以外の場合は、エージェントの負荷が高い場合に、GUI ユーザーが JBoss ON UI へのアクセスをブロックする可能性がありました。 Web 接続の制限は HTTP および HTTPS（セキュア）リクエストの両方で同じであるため、許可される最大接続の **合計** は、この設定とは 2 回になります。たとえば、最大 Web 接続が 300 に設定されている場合、合計 600 の同時 Web 接続に対して 300 の HTTP リクエストが許可され、300 の HTTPS リクエストが許可されます。
rhq.communications.global-concurrency-limit	サーバーに送信されるエージェントメッセージの合計数を設定します。これは、GUI 要求ではなく、影響を受ける受信エージェントメッセージのみに影響します。このグローバル同時実行制限を 300 に設定すると、メッセージの種類に関係なく、一度に 300 を超えるエージェントメッセージを処理できます。 他の同時実行制限の合計がこのグローバル制限よりも大きい場合でも、グローバル制限よりも多くのメッセージが処理されないため、このグローバル制限に制限されます。 この値は、許可されている Web 接続の数よりも若干小さい値である必要があります。これにより、エージェントの負荷が高い場合に GUI への Web 接続がブロックされないようにします。
rhq.server.concurrency-limit.inventory-report	インベントリーレポートは、エージェントが起動するとエージェントから送信され、定期的に送信されます。エージェントマシンのリソース数によっては、インベントリーレポートが大きくなる可能性があります。
rhq.server.concurrency-limit.availability-report	可用性レポートは、通常 60 秒ごとにエージェントから定期的に送信されます。通常、可用性レポートは非常に小さくなりますが、送信頻度が高くなるため、大量の生じます。

パラメーター	description
rhq.server.concurrency-limit.inventory-sync	インベントリーの同期は、エージェントがサーバーのインベントリーを同期する必要がある場合に発生します。通常、エージェントは起動時に同期します。エージェントによって管理されるリソースの数により、通常、インベントリーの同期の一部としてフローされるトラフィックは、大きくなります。
rhq.server.concurrency-limit.content-report	コンテンツレポートは、検出されたコンテンツ（ソフトウェアのインストールパッケージなど）に関する情報が含まれている場合を除き、インベントリーレポートと似ています。このレポートは、エージェントが検出し、管理しているインストール済みのソフトウェアの数によって大きくなります。
rhq.server.concurrency-limit.content-download	コンテンツダウンロードは、エージェントのリソースで、通常はパッケージをインストールするためにパッケージバージョンのコンテンツを要求する必要がある場合に発生します。
rhq.server.concurrency-limit.measurement-report	エージェントが測定収集を完了すると、測定レポートが定期的にサーバーに送信されます。測定レポートの数とサイズは、収集される測定の数と頻度によって異なります。エージェントが収集する必要のあるスケジュール測定値が大きくなり、より頻繁に測定レポートが送信されるため、レポートが大きくなるほど増加します。
rhq.server.concurrency-limit.measurement-schedule-request	インベントリーの同期と同様に、測定スケジュールリクエストはサーバーに対して最新の測定スケジュールのセットを要求するエージェントに送信されます。

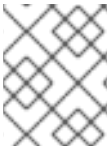
バグを報告します。

6.3.4. メール通知の SMTP サーバーの設定

各 JBoss ON サーバーは特定の SMTP サーバーと通信します。SMTP サーバーは **rhq-server.properties** ファイルに定義されます。デフォルトの設定は、ローカルの JBoss ON サーバーホストを参照します。

```
# Email
rhq.server.email.smtp-host=localhost
rhq.server.email.smtp-port=25
rhq.server.email.from-address=rhqadmin@localhost
```

これらの設定を編集して、異なる SMTP サーバーまたはメールアカウントを使用できます。



注記

SMTP 設定が正しく、サーバーが電子メールを正常に送信できることを確認するには、にあるテストメールページに移動し **http://*server*/portal/admin/test/email.jsp** ます。

表6.3 rhq-server.properties パラメーター for SMTP

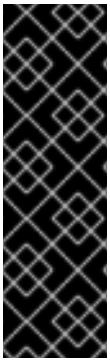
パラメーター	description
rhq.server.email.smtp-host	JBoss ON サーバーによって使用される SMTP サーバーのホスト名を設定します。
rhq.server.email.smtp-port	JBoss ON サーバーによって使用される SMTP サーバーのポートを設定します。
rhq.server.email.from-address	JBoss ON サーバーによって送信されたすべてのメールの **From** ヘッダーに使用するアドレスを設定します。

バグを報告します。

6.3.5. サーバーをサイレントインストール

rhq-server.properties ファイルの一部のオプションは、Web ベースのインストーラーからではなく、ファイルからサーバー設定を読み込むようにインストールプロセスに指示します。

```
# Auto-Install Pre-Configuration Settings
rhq.autoinstall.enabled=true
rhq.autoinstall.database=auto
rhq.autoinstall.public-endpoint-address=
```



重要

自動インストーラーオプションは、JBoss ON サーバーを最初にインストールした時に一度だけ評価されます。この初期設定後、自動インストーラーが無効になります。これらのプロパティーはサーバーのセットアップ後に無視され、既存のインスタンスの再インストールには使用できません。

サーバーを再インストールするには、最初にサーバーインストールディレクトリーを削除してから、元の JBoss ON サーバーアーカイブを展開し、新規であるかのようにサーバーをインストールします。

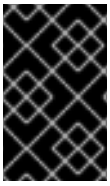
表6.4 rhq-server.properties パラメーターのサイレントインストール

パラメーター	description
rhq.autoinstall.enabled	**rhq-server.properties** ファイル(**true**)から設定を読み込むかどうか、または Web ベースのインストーラー(**false**)から設定を読み込むかどうかをインストールプロセスに指示します。

パラメーター	description
rhq.autoinstall.database	データベーススキーマを読み込んだり、追加する方法をインストールプロセスに指示します。以下の 3 つのオプションがあります。 ● **auto** データを上書きすることなく、新規インストールまたは既存のスキーマのアップグレード用に新しいスキーマを作成します。 ● **overwrite** データベースを上書きし、新しい空のスキーマを作成します。 ● **skip** データベースの作成や更新が行われないように、データベースプロセス全体をスキップします。
rhq.autoinstall.public-endpoint-address	サーバーに使用する IP アドレスまたはホスト名を設定します。値の指定がない場合、サーバーは起動時に独自の値を検出し、設定します。

バグを報告します。

6.3.6. サーバー設定における機密情報の保護



重要

デフォルトでは、サーバー設定ファイルのすべてのパスワードは、アップグレードおよびインストール時に保護されます。このプロセスは、ユーザーが追加のプロパティーを保護する場合にのみ必要です。

JBoss ON は、サーバー設定ファイルのほぼすべてのプロパティー（特に **{RHQ_SERVER_HOME}/jbossas/standalone/configuration/standalone-full.xml** および **{RHQ_SERVER_HOME}/bin/rhq-server.properties** ファイル）をサポートします。JBoss ON は、プロパティーを難読化するための **{RHQ_SERVER_HOME}/bin/** ディレクトリーに **rhq-encode-value.sh** および **rhq-encode-value.bat** スクリプトを提供します。

バグを報告します。

6.3.6.1. standalone-full.xml

たとえば **standalone-full.xml**、プロパティー値はパスワード vault メカニズムによりエンコードされ、すべてのパスワードはインストール時にデフォルトで保護されます。このパスワード vault メカニズムは JBoss Enterprise Application Platform 6 の機能として提供されます。また、JBoss ON は JBoss ON 内で使用するために同じパスワード vault メカニズムの変更済み実装を使用してこの機能を提供します。パスワード vault メカニズムの詳細は、JBoss Enterprise Application Platform 6 の『管理および設定ガイド』の「パスワード vaults for Sensitive Strings」を参照してください。

バグを報告します。

6.3.6.2. rhq-server-properites

たとえば **rhq-server.properties**、プロパティー値は **RESTRICTED::** 形式を使用してエンコードされ、すべてのパスワードはインストール時にデフォルトで保護されます。

バグを報告します。

6.3.6.3. エンコーディングでの rhq-encode-value スクリプトの使用

rhq-encode-value スクリプトを呼び出すと、必要なプロパティーと値をエンコードするように求められます。

```
> ./rhq-encode-value.sh
Property rhq.autoinstall.server.admin.password [y/n]: n
Property rhq.server.database.password [y/n]: n
Property: rhq.protect.property
Value: 1234
*** !!! WARNING !!!
*** Both standalone-full.xml and rhq-server.properties need to be updated if a property from rhq-
server.properties is used in standalone-full.xml
*** !!! WARNING !!!
***
***
*** Encoded password for rhq-server.properties:
***    rhq.protect.property=RESTRICTED::-299a94df3b478ca8
***
*** Encoded password for standalone-full.xml with vault with password as default value:
***    ${VAULT::restricted::rhq.protect.property::-299a94df3b478ca8}
***
*** Encoded password for standalone-full.xml with vault without default:
***    ${VAULT::restricted::rhq.protect.property:: }
***
*** Encoded password for agent-configuration.xml:
***    <entry key="rhq.protect.property" value="RESTRICTED::-299a94df3b478ca8" />
***
*** Please consult the documentation for additional help.
```

スクリプトの実行後、から生成された値はサーバー設定ファイルにコピーされ、貼り付ける **rhq-encode-value** 必要があります。

重要

にある一部のプロパティー **standalone-full.xml** も存在し **rhq-server.properites**ます。両方のファイルにあるプロパティーを保護する場合は、サーバーを再起動する前に、両方の場所が同じエンコードされたバリアントで更新されていることを確認します。部分的な更新により、サーバーが使用できなくなる可能性があります。

バグを報告します。

6.4. サーバー設定の同期

JBoss ON サーバーは、異なる環境でであっても、多くの同じ設定を共有できます。たとえば、異なる JBoss ON サーバーによって開発環境、ステージング環境、および実稼働環境が 3 つすべて管理されても、サーバーは同様のメトリクステンプレートと設定を使用します。

個別の環境と類似した環境の管理を簡素化するために、JBoss ON はサーバーの設定をエクスポートし、その設定を別のサーバーにインポートできます。

設定を管理する権限を持つユーザーは、サーバー設定をエクスポートできます。データには、以下の 2 つのカテゴリーがあります。

- アラート、イベント、モニタリングメトリックの保存期間、ベースライン計算スケジュール、LDAP サーバー設定など、システム設定。
- 各リソースタイプのメトリクスコレクション設定。

この情報は、gz 形式の XML ファイルにダンプされたエクスポートされます。このファイルは、別のサーバーにインポートする前に簡単に編集できます。

注記

サーバー設定の同期は、サーバーが異なるバックエンドデータベースを使用する場合にのみ必要です。データベース（高可用性クラウド）を共有するサーバーはすでに設定を共有しています。

インポートおよびエクスポート操作は JBoss ON CLI でのみ行われます。詳細は、カスタマーポータルの JBoss ON API ドキュメントを参照してください。

バグを報告します。

6.4.1. サーバー設定のエクスポート

1. JBoss ON CLI にログインします。

```
[root@server bin]# installDir/bin/rhq-cli.sh -u rhqadmin -p rhqadmin
```

2. データをデータベースオブジェクトにエクスポートします。

```
rhqadmin@localhost:7080$ var ex = SynchronizationManager.exportAllSubsystems();
```

3. そのオブジェクトをエクスポートファイルに変換します。ファイル拡張子は、エクスポート形式が GZIP の XML ファイルである **.xml.gz** ためです。

```
rhqadmin@localhost:7080$ saveBytesToFile(ex.exportFile, 'export.xml.gz');
```

注記

サーバーデータをエクスポートするために、ユーザーには管理設定のパーミッションが必要です。

バグを報告します。

6.4.2. サーバー設定のインポート

サーバー設定は XML ファイルにエクスポートされます。管理者は、このファイルを編集して他の JBoss ON サーバーにインポートされる情報の種類を制御できるため、インポートプロセスに多くの適応性があります。ファイルがインポートされると、最初に一連の検証テストを実行して、設定データを

実際にサーバーにインポートできるようにします。次に、2 つのクラス **または** 同期（システム設定用およびメトリクステンプレート用）を使用して、データのインポートに使用します。

インポートプロセスは管理者が変更できるため、一般的なインポートシナリオがいくつかあります。

- 設定データは、すべてのデフォルト設定を使用して直接サーバーにインポートされます。
- XML ファイルを編集して、設定値がターゲット JBoss ON サーバーに適用されるようにできます。
- インポートされるデータ要素を変更する同期の動作が変更されます。

バグを報告します。

6.4.2.1. XML インポートファイルの編集

すべてのデータは、1 つの XML ファイルにダンプされます。これには、システム設定、各リソースタイプのメトリック設定、処理命令が含まれます。

設定エントリーはすべて 2 つの大きな **<entities>** 要素で定義されます。

メトリクステンプレートは各メトリクスを個別の **<entity>** 要素に個別に一覧表示し、その名前、リソースタイプ、およびプラグインによって識別されるメトリクス自体を要素の引数として識別します。エンティティー ID は JBoss ON データベースのテンプレートを特定しますが、サーバー間で ID を照合する必要がないため、インポート時には無視されます。

```
<entities id="org.rhq.enterprise.server.sync.MetricTemplateSynchronizer">
 <entity>
  <data>
   <metricTemplate
    enabled="false"
    defaultInterval="300000"
    perMinute="false"
    metricName="trap_count"
    resourceTypePlugin="snmptrapd"
    resourceTypeName="SnmpTrapd"
    referencedEntityId="10001">
   </metricTemplate>
  </data>
 </entity>
 .....
```

一方、システム設定はすべて 1 つの **<entity>** 要素で定義され、各設定パラメーターはエントリーのキーとして指定されます。これらのキーすべてがターゲットサーバーにインポートされているわけではありません。インポートされたキーは、シンクラーの設定によって異なります。

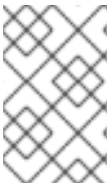
```
<entities id="org.rhq.enterprise.server.sync.SystemSettingsSynchronizer">
 <entity>
  <data>
   <systemSettings referencedEntityId="0">
    <entry key="CAM_BASE_URL">http://10.16.65.121:7080/</entry>
    <entry key="CAM_DATA_PURGE_6H">2678400000</entry>
    <entry key="CAM_LDAP_BIND_DN"></entry>
    .....
   </systemSettings>
```

```
  </data>
 </entity>
</entities>
```

バグを報告します。

6.4.2.2. シンchronizer 設定の変更

JBoss ON は **同期** を使用して、メトリクススケジュールなどの要素を JBoss ON サーバーにインポートし、サーバーに適用する方法を設定します。同期には、すべてのインポート操作にプログラムで設定変更を適用するデフォルトテンプレートがあります。また、エクスポートされた XML ファイルに同期する設定があり、その特定のインポート操作に適用されます。



注記

XML ファイルのカスタム設定は、プログラムによるテンプレートの設定を上書きします。CLI コマンドで渡されるプログラムによる設定は、XML ファイルの設定を上書きします。

特定の同期ツールの設定を出力するには、の同期名を指定します。**SynchronizationManager.getImportConfigurationDefinition()**.例：

```
rhqadmin@localhost:7080$ var configDef =
SynchronizationManager.getImportConfigurationDefinition('org.rhq.enterprise.server.sync.SystemSettingsSynchronizer')
```

両方の同期機能の設定をすべて出力するには、次のコマンドを実行します。

```
rhqadmin@localhost:7080$ var configDefs =
SynchronizationManager.importConfigurationDefinitionOfAllSynchronizers
rhqadmin@localhost:7080$ configDef = configDefs.get(0)

rhqadmin@localhost:7080$
pretty.print(configDef.configurationDefinition.defaultTemplate.configuration)
```

バグを報告します。

6.4.2.2.1. XML ファイルのシンchronizer 設定の変更

同期設定をカスタマイズする最も簡単な方法は、エクスポートされた XML ファイルの設定を変更することです。設定およびメトリクスの同期は、リソースプラグイン設定と非常によく似ている XML 要素を使用します。シンクサーのルート要素はで **<default-configuration>**、設定はその要素内の properties として一覧表示されます。

設定同期には、最も単純な設定があります。これには単一の **<ci:simple-property>** 要素があり、インポートする設定の一覧は **<ci:simple-property>** 要素の ***value=*** フラグに指定されます。

```
<default-configuration>
   <ci:simple-property value="AGENT_MAX_QUIET_TIME_ALLOWED,
ENABLE_AGENT_AUTO_UPDATE, ENABLE_DEBUG_MODE,
ENABLE_EXPERIMENTAL_FEATURES,
CAM_DATA_PURGE_1H, CAM_DATA_PURGE_6H, CAM_DATA_PURGE_1D,
CAM_DATA_MAINTENANCE, DATA_REINDEX_NIGHTLY, RT_DATA_PURGE, ALERT_PURGE,
EVENT_PURGE,
```

```
TRAIT_PURGE, AVAILABILITY_PURGE, CAM_BASELINE_FREQUENCY,
CAM_BASELINE_DATASET" type="string" name="propertiesToImport">
      <c:description>The names of the properties that should be imported. Note that these are the
INTERNAL names as used in the RHQ database</c:description>
   </ci:simple-property>
</default-configuration>
```

注記

設定の値は、サーバー設定の JBoss ON データベースで使用される名前です。

メトリクススケジュールはリソースごとに異なるため、メトリクススケジュールの設定はより複雑です。メトリクススケジュールは、以下の 3 つの方法（または組み合わせ）で定義できます。

- プロパティー(**<ci:simple-property>**)で定義されている **<ci:list-property>** リストメンバーと値の一覧が含まれる簡単なリスト。

```
<default-configuration>
   <ci:list-property name="my-list">
      <c:simple-property name="element" type="string"/>
      <ci:values>
         <ci:simple-value value="a"/>
         <ci:simple-value value="b"/>
         <ci:simple-value value="c"/>
      </ci:values>
   </ci:list-property>
</default-configuration>
```

- 値のマップ。簡単なリストと、プロパティー（）と対応する値のリスト(**<ci:simple-property>**)を使用します。ただし**<ci:simple-value>**、各値は名前に基づく単一の指定されたプロパティーに対応します。

```
<default-configuration>
   <ci:map-property name="my-map">
      <c:simple-property name="prop1" type="integer"/>
      <c:simple-property name="prop2" type="string"/>
      <c:simple-property name="prop3" type"boolean"/>
      <ci:values>
         <ci:simple-value property-name="prop1" value="1"/>
         <ci:simple-value property-name="prop2" value="abc"/>
         <ci:simple-value property-name="prop3" value="true"/>
      </ci:values>
   </ci:map-property>
</default-configuration>
```

- マップの一覧であるテーブル。マップの各セットは、行の 1 つのテーブルを指定します。

```
<default-configuration>
   <ci:list-property name="table">
      <c:map-property name="row">
         <c:simple-property name="column1" type="integer"/>
         <c:simple-property name="column2" type="boolean"/>
         <c:simple-property name="column3" type="string"/>
      </c:map-property>
```

```
        <ci:values>
          <ci:map-value>
            <ci:simple-value property-name="column1" value="1"/>
            <ci:simple-value property-name="column2" value="true"/>
            <ci:simple-value property-name="column3" value="a"/>
          </ci:map-value>
          <ci:map-value>
            <ci:simple-value property-name="column1" value="2"/>
            <ci:simple-value property-name="column2" value="true"/>
            <ci:simple-value property-name="column3" value="b"/>
          </ci:map-value>
          <ci:map-value>
            <ci:simple-value property-name="column1" value="3"/>
            <ci:simple-value property-name="column2" value="false"/>
            <ci:simple-value property-name="column3" value="c"/>
          </ci:map-value>
        </ci:values>
      </ci:list-property>
    </default-configuration>
```

たとえば、これはマップを使用して、JBoss AS 5 サーバーの空きメモリーメトリックのメトリックスケジュールのみをインポートします。

```
<default-configuration>
  <ci:simple-property value="false" type="boolean" name="updateAllSchedules" />
  <ci:list-property name="metricUpdateOverrides">
    <c:map-property summary="false" required="true" readOnly="false"
name="metricUpdateOverride">
      <c:simple-property type="string" summary="false" required="true" readOnly="false"
name="metricName" />
      <c:simple-property type="string" summary="false" required="true" readOnly="false"
name="resourceTypeName" />
      <c:simple-property type="string" summary="false" required="true" readOnly="false"
name="resourceTypePlugin" />
      <c:simple-property type="boolean" summary="false" required="true" readOnly="false"
name="updateSchedules" />
    </c:map-property>
    <ci:values>
      <ci:map-value>
        <ci:simple-value name="metricName" value="MCBean|ServerInfo|*|freeMemory"/>
        <ci:simple-value name="resourceTypeName" value="JBoss AS Server"/>
        <ci:simple-value name="resourceTypePlugin" value="JBossAS5"/>
        <ci:simple-value name="updateSchedules" value="true"/>
      </ci:map-value>
    </ci:values>
  </ci:list-property>
</default-configuration>
```

すべてのメトリクススケジュールを更新するには、**<ci:simple-property>** 要素をに設定し **name="updateAllSchedules"** ます。

単一のメトリクススケジュールを更新するには、property 要素の名前をに設定し、**updateSchedules** property の値を **metricUpdateOverride** 設定し **true** ます。

バグを報告します。

6.4.2.2.2. プログラムによるシンクレーション設定の変更

設定を変更するには、デフォルトの新しいインスタンスを作成し、**setValue** 設定オブジェクトを使用してリストからキーを追加または削除します。設定同期では、インポートするキー名が一覧表示されます。

```
configurationObject.getSimple('propertiesToImport').setValue(defaultSettingsToImport + ', keyName')
```

メトリクススケジュールでは、プロパティーのリストまたはプロパティーマップに基づいて、リソースタイプごとのメトリクススケジュールを一覧表示します。

```
var update = new PropertyMap('metricUpdateOverrides')
update.put(new PropertySimple('propertyName', 'resourcePluginName'))
```

1. デフォルトの定義を取得します。

```
rhqadmin@localhost:7080$ var systemSettingsImportConfigurationDefinition =
SynchronizationManager.getImportConfigurationDefinition('org.rhq.enterprise.server.sync.SystemSettingsSynchronizer')
```

2. 新しい設定インスタンスを作成します。

```
rhqadmin@localhost:7080$ var configurationObject =
systemSettingsImportConfigurationDefinition.configurationDefinition.defaultTemplate.createConfiguration()

rhqadmin@localhost:7080$ var systemSettingsImportConfiguration = new
ImportConfiguration(systemSettingsImportConfigurationDefinition.synchronizerClassName,
configurationObject)
```

3. 新しいインスタンスの設定を変更します。

   たとえば、サーバー設定の同期の場合、以下を実行します。

```
rhqadmin@localhost:7080$ var defaultSettingsToImport =
configurationObject.getSimple('propertiesToImport').stringValue

rhqadmin@localhost:7080$
configurationObject.getSimple('propertiesToImport').setValue(defaultSettingsToImport + ',
CAM_BASE_URL')
```

   メトリクステンプレート同期の場合：

```
configurationObject.getSimple('updateAllSchedules').setBooleanValue(true)
var updateList = new PropertyList('metricUpdateOverrides')
var update = new PropertyMap('metricUpdateOverride')
update.put(new PropertySimple('metricName', 'MCBean|ServerInfo|*|freeMemory'))
update.put(new PropertySimple('resourceTypeName', 'JBossAS Server'))
update.put(new PropertySimple('resourceTypePlugin', 'JBossAS5'))
update.put(new PropertySimple('updateSchedules', 'true'))

updateList.add(update)

configurationObject.put(updateList)
```

-

バグを報告します。

6.4.2.3. 設定のインポート

1. JBoss ON CLI にログインします。

```
[root@server bin]# installDir/bin/rhq-cli.sh -u rhqadmin -p rhqadmin
```

2. 設定を含む XML ファイルをインポートします。

```
rhqadmin@localhost:7080$ var data = getFileBytes('export.xml.gz');
rhqadmin@localhost:7080$  SynchronizationManager.importAllSubsystems(ex, null);
```

null パラメーターは、インポートプロセスで XML ファイルのデフォルト設定が使用されるか、またはデフォルトが XML にない場合、ターゲットサーバーに定義された設定を使用することを意味します。 で代替の設定が構築されている場合は 「シンchronizer 設定の変更」、代わりにプログラムで指定できます。例：

```
rhqadmin@localhost:7080$ var configsToImport = new java.util.ArrayList()
rhqadmin@localhost:7080$ configsToImport.add(systemSettingsImportConfiguration);
rhqadmin@localhost:7080$ configsToImport.add(metricTemplatesImportConfiguration);
rhqadmin@localhost:7080$ SynchronizationManager.importAllSubsystems(ex,
configToImport);
```

バグを報告します。

---

[1] これらの設定により、JBoss ON サーバーインスタンスの特定の IP アドレスおよびポートが設定されます。ファイアウォールで異なる設定が失敗した場合は、これらのパラメーターを変更することができます。

[2] この値の変更を反映するには、JBoss ON サーバーを再起動する必要があります。

第7章 エージェントの設定

エージェントは、**rhq-agent.sh** スクリプトから開くエージェントプロンプトで設定および管理できます。

バグを報告します。

7.1. エージェントの登録および再登録

エージェントが JBoss ON サーバーに登録すると、エージェント名は一意のリソースキーとして使用され、エージェントを特定します。さらに、サーバーは、登録トークンまたは **セキュリティートークンとして使用するためにエージェントに送信するランダムな文字列を生成します。**

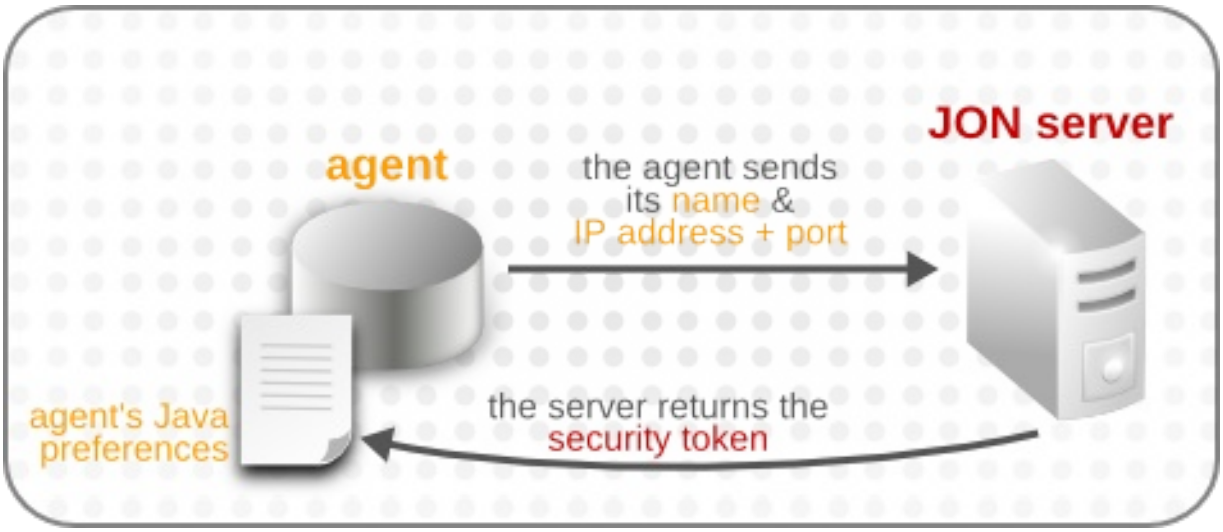
バグを報告します。

7.1.1. セキュリティートークンとエージェント登録について

JBoss ON エージェントが起動すると、JBoss ON サーバーに登録され、サーバー情報を送信します。JBoss ON サーバーは、指定のエージェント名、IP アドレス、およびポート番号を基にエントリーを作成します。

JBoss ON サーバーは、エージェント名と IP アドレスとポート番号のペアに関連 **する無作為に生成された文字列であるセキュリティートークン** も作成します。

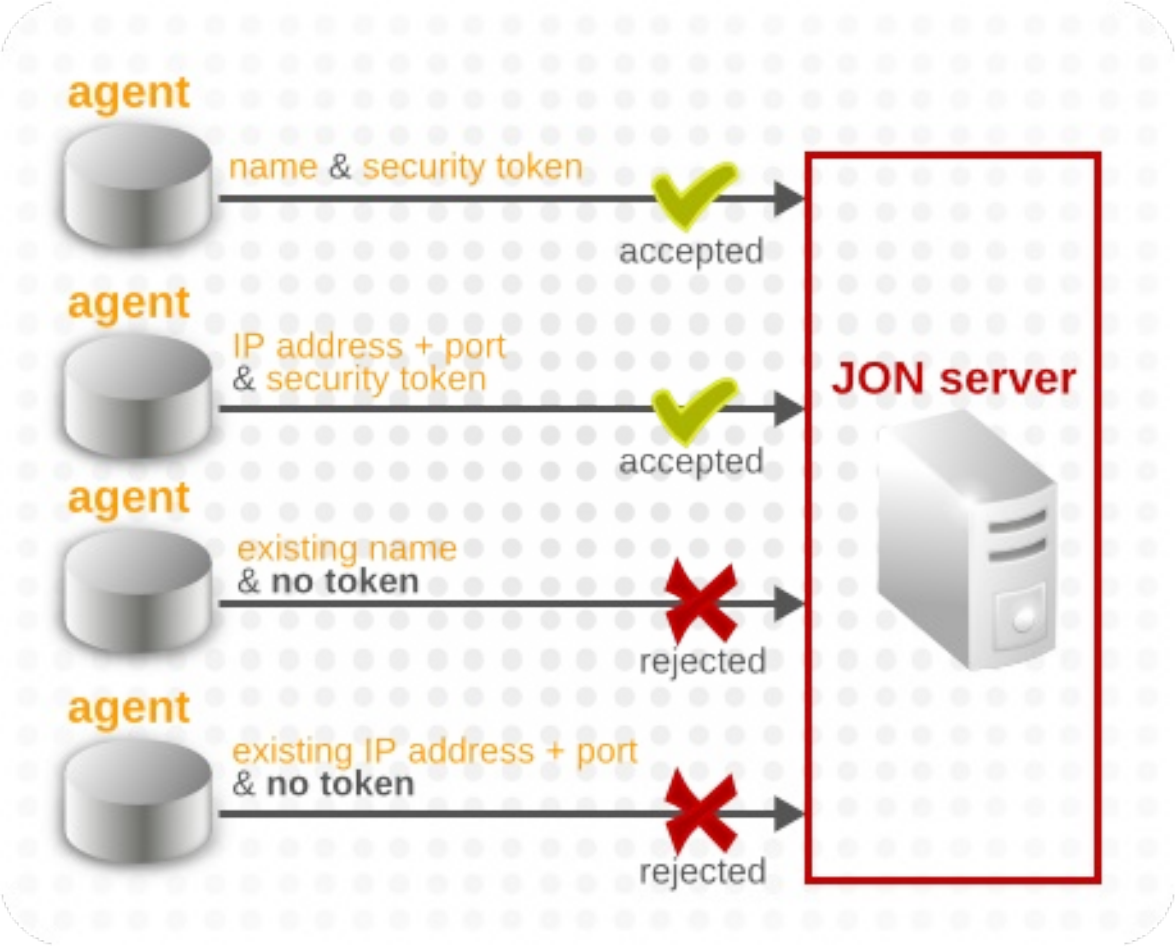
図7.1 エージェント登録



擬似認証の形式として再起動すると、エージェントはそのセキュリティートークンをサーバーに送信します。JBoss ON サーバーは、一意のリソースキー（エージェントの名前）と、そのセキュリティートークンをエージェントアイデンティティーを検証する方法として使用します。

JBoss ON サーバーは、エージェントが起動してサーバーに登録するたびにエージェント名とセキュリティートークンを関連付けます。エージェントが提供する情報が、そのエージェントに対して JBoss ON サーバーが提供する情報と一致しない場合、エージェントの接続試行を拒否します。

図7.2 異なるエージェント接続の試行



そのため、JBoss ON サーバーがエージェントの登録情報への変更を受け入れるタイミングについて、いくつかのルールがあります。

- エージェントは、対応するセキュリティートークン **なし** で既存のエージェント名で登録することはできません。

  既存のエージェント名でエージェントを登録するには、の説明に従って、最初に対応するセキュリティートークンをインストールする必要があり 「失われたセキュリティートークンの再インストール」 ます。

- エージェントは、対応するセキュリティートークンがなく、元のエージェント名を **使用** すること **なく**、既存の IP アドレス/ポートの組み合わせで登録することはできません。

  つまり、基本的にエージェントの名前を変更することはできません。エージェントが既存の IP アドレス/ポートの組み合わせに登録されている場合は、元のセキュリティートークンと元の名前の両方を使用する必要があります。これにより、エージェントの元のアイデンティティーが再確立され、あるエージェントが別のエージェントのアイデンティティーを効果的に破棄するのを防ぎます。

- エージェント名に対応するセキュリティートークンがある場合は、エージェントは既存の名前と新しい IP アドレス/ポートの組み合わせで登録 **でき** ます。

  再登録時にエージェント名を変更できず、エージェント IP アドレス、エージェントポート、または両方を変更することができます。これは、既存のエージェントが新しい IP アドレスまたはポートで再登録する必要があるクラウド、仮想、または DHCP 環境における一般的かつ便利なシナリオです。

注記

セキュリティートークンはエージェントの Java 設定に保存されます。このセキュリティートークンは、エージェントが再起動されたり、アンインストールされたり、その設定が消去された場合でも永続し **--cleanconfig**ます。これにより、エージェントは簡単に再登録できます。

バグを報告します。

7.1.2. 失われたセキュリティートークンの再インストール

セキュリティートークンがエージェントの設定から誤って削除されると、エージェントはサーバーと通信できなくなります。エラーの **認証に失敗したと、試行に失敗** します。

失われたセキュリティートークンは、エージェントの設定に手動で追加できます。

1. エージェントを停止します。
2. セキュリティーパーミッションの管理を持つユーザーとして Web UI にログインします。
3. **Administration** タブをクリックし、左側の **Topology** セクションの下にある **Agents** リンクをクリックします。
4. 一覧からエージェントを選択し、その名前をクリックして詳細ページを開きます。

5. セキュリティートークンをコピーします。
6. エージェントを再起動して、**-D** オプションを使用して ***rhq.agent.security-token*** プロパティーをセキュリティートークンに設定します。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -Drhq.agent.security-token=abcd1234
```

バグを報告します。

7.1.3. 新しいセキュリティートークンを使用したエージェントの再インストール

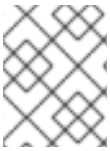
エージェントは、完全に新しい設定で再インストールおよび再登録できます。エージェントの設定には、エージェントの（ローカル）永続設定、エージェントインベントリー（および関連するリソースデータ）、およびサーバーインベントリーのプラットフォームエントリーの 3 つの点があります。エージェントが正常に再インストールできるようにするには、ローカルマシンの設定と JBoss ON サーバーのエージェントおよびリソース設定の両方をクリアする必要があります。

- エージェントの永続化された Java 設定はパージされる必要があります。
- エージェントのインベントリーは、リソースの履歴および設定と共にパージする必要があります。
- エージェントは JBoss ON インベントリーから削除する必要があります。これには、JBoss ON トポロジー(「エージェントの削除」)からエージェントを削除するか、または platform エントリーを削除します。
- エージェントの元の識別情報（名前、IP アドレス、およびポート）は変更できます。

エージェントを再インストールするには、以下を実行します。

1. 元のエージェントインスタンスが正しく削除されていることを確認します。
   a. エージェントプロセスを停止します。
   b. JBoss ON サーバーインベントリーから platform エントリーを削除します。
2. **--fullcleanconfig** オプションを指定してエージェントを再起動します。これにより、エージェントに新しいセキュリティートークンと新しい設定が登録されます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh --fullcleanconfig
```



注記

JBoss ON インベントリーからエージェントが削除されない場合、エージェントに無効なセキュリティートークンがあるというエラーで再インストールに失敗します。

バグを報告します。

7.1.4. 元のセキュリティートークンを使用したエージェント設定のクリーニング

別の再登録パスは、**そのセキュリティートークン以外** のエージェント設定を消去します。エージェントはその既存のセキュリティートークンを使用してサーバーを登録するため、基本的に再登録の代わりに登録内容を更新します。

この場合、元のエージェント設定のほとんどが保持されます。

- エージェントの永続化された Java 設定は消去されます。
- エージェントのインベントリーと、リソース履歴および設定が保存されています。
- （プラットフォームエントリー経由）エージェントは JON インベントリーに残ります。
- エージェントの名前はそのままにする必要があります（ただし、IP アドレスまたはポート番号は変更できます）。

次に主なアクションはエージェント設定が更新され、エージェントエントリー自体が保持されます。

エージェント設定を消去するには、**--cleanconfig** オプションを指定してエージェントを再起動します。これにより、（ **conf/agent-configuration.xml** ファイルから）エージェントが新しい設定に登録され、以前のセキュリティートークンが再利用されます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh --fullcleanconfig
```

注記

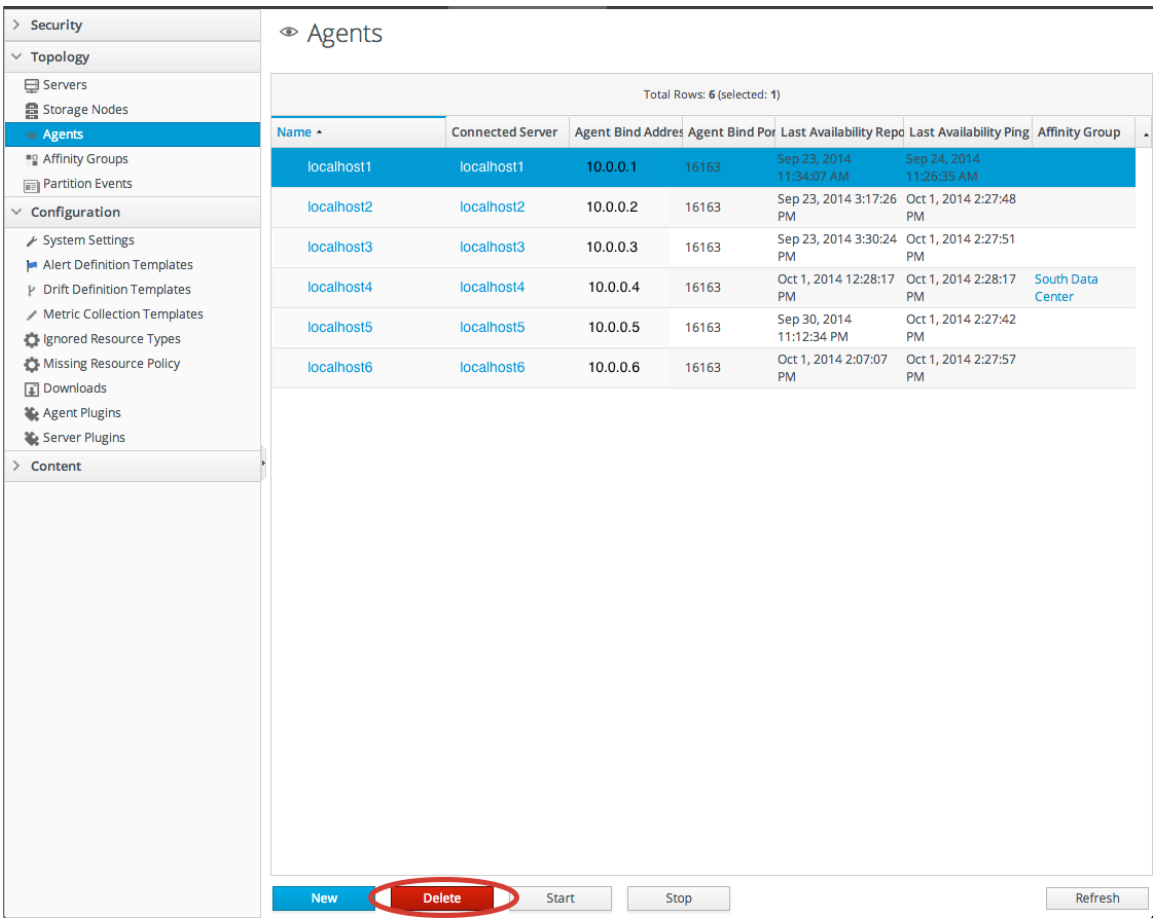
エージェント名が異なる場合は、既存のセキュリティートークンを指定の（新しい）エージェント名で検証できないため、再登録の試行に失敗します。

バグを報告します。

7.2. エージェントの削除

7.2.1. 管理対象プラットフォームからのエージェントの削除

1. エージェントサービスを停止します。
2. JBoss ON トポロジーからエージェントを削除します。
   a. JBoss ON UI で、トップメニューの **Administration** タブをクリックします。
   b. 左側のメニューの **Topology** セクションで **Agents** 項目を選択します。
   c. インストール済みエージェントの一覧で、削除するエージェントの行を選択します。



   d. ページ下部の **Delete** ボタンをクリックします。
   e. エージェントを削除する必要があることを確認します。
3. エージェントがシステムサービスとして設定されている場合は、関連するファイルを削除します。Windows では、**rhq-agent.bat remove** コマンドで実行できます。Linux では、**init.d**/ ファイルを削除して更新し **chkconfig**ます。

4. エージェントのインストールディレクトリーを削除します。

バグを報告します。

7.2.2. JBoss ON Server マシンでのエージェントの削除

1. エージェントサービスを停止します。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop --agent
```

2. エージェントを削除します。

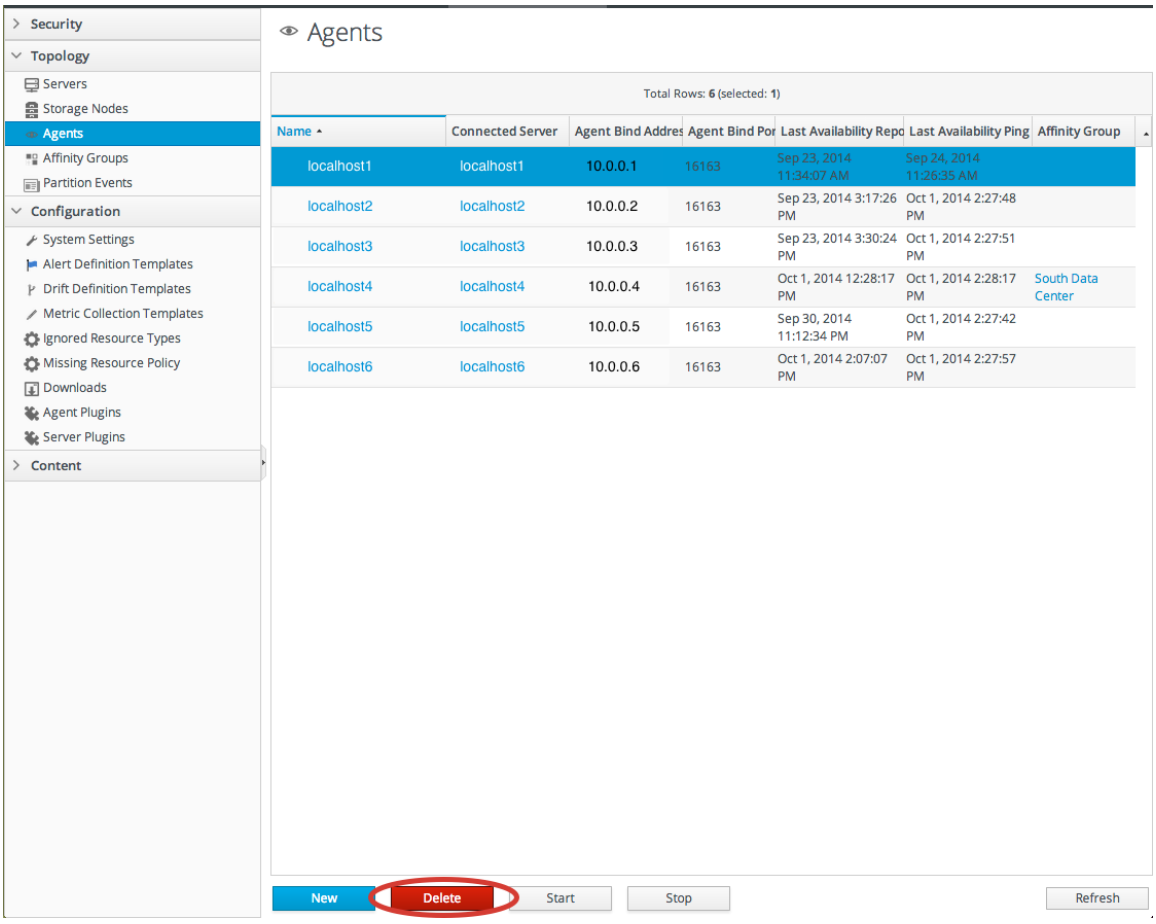
```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl remove --agent
```

3. JBoss ON トポロジーからエージェントを削除します。

   a. JBoss ON UI で、トップメニューの **Administration** タブをクリックします。

   b. 左側のメニューの **Topology** セクションで **Agents** 項目を選択します。

   c. インストール済みエージェントの一覧で、削除するエージェントの行を選択します。



   d. ページ下部の **Delete** ボタンをクリックします。

   e. エージェントを削除する必要があることを確認します。

バグを報告します。

7.3. エージェントのコマンドプロンプトの使用

エージェントがターミナルで起動すると（エージェントプロセスを開始して）、スクリプトはエージェントコマンドプロンプトを起動します。エージェントプロンプトを使用してエージェントを管理するには、設定の確認、一部のタスクの実行、またはエージェントの設定の編集を行います。

バグを報告します。

7.3.1. エージェントのコマンドプロンプトを開く

エージェントの起動スクリプトを実行すると、エージェントコマンドプロンプトが開きます。

```
$ rhq-agent.sh
```

バグを報告します。

7.3.2. エージェント起動オプション

エージェント管理によっては、**rhq-agent.sh** 開始スクリプトでオプションを渡すことで実行できます。これらは主に、入力ファイルまたは渡されたパラメーターを使用して外部設定を読み込んでサーバーに永続的な設定オプションを渡すことを目的としています。これらのオプションはに記載されてい 表7.1「rhq-agent.sh スクリプトのオプション」 ます。

表7.1 rhq-agent.sh スクリプトのオプション

短い引数	長い引数	description
-a	--advanced	基本的な開始モードではなく、設定モードでエージェントスクリプトを実行します。
-c	--config=**filename**	ファイルシステムまたはクラスパスのエージェント設定ファイルを指定します。
-d	--daemon	エージェントをデーモンモードで実行します。つまり、stdin から追加のコマンドが読み取られません。
-D**name**[=**value**]		エージェント設定の設定を上書きし、システムプロパティーを設定します。
-e	--console=**type**	コンソール入力の読み取り時に使用する実装を指定します。使用できる 3 つの値は jline、sigar、および java です。
-g	--purgeplugins	すべてのプラグインを削除し、すべてのプラグインを再ダウンロードするよう強制します。

短い引数	長い引数	description
-h	--help	ヘルプメッセージを開きます。
-i	--input=**filename**	入力に使用するスクリプトファイルを指定します。
-l	--cleanconfig	既存の設定およびデータファイルをすべて消去します。これにより、エージェントは保持されるエージェントセキュリティートークンを**除き**、空白の設定で起動します。
-L	--fullcleanconfig	既存の設定ファイルおよびデータファイルを消去します。これにより、エージェントはセキュリティートークンのパージなど、完全にクリーンなスレートで起動します。
-n	--nostart	エージェントプロセスを開始せずに、エージェントスクリプトを実行します。
-o	--output=**filename**	スクリプトからすべての出力を書き込むファイルを指定します。ただし、ログメッセージはエージェントログに常に書き込まれます。
-p	--pref=**preferences_name**	エージェント設定に使用する Java プリファレンスノードを指定します。
-s	--setup	エージェントがセットアップに関する質問を強制します。
-t	--nonative	エージェントがネイティブシステムを無効にするように強制します。
-u	--purgedata	永続インベントリーおよびその他のデータファイルをパージします。
--		エージェントがオプションの処理を停止します。

バグを報告します。

7.3.3. エージェントプロンプトコマンド

エージェントは、エージェントプロンプトを介して対話的に渡される、または起動スクリプトの起動時に渡すことのできる入力ファイルから渡されるプロンプトコマンドを処理します。エージェントプロンプトコマンド（にリストされている 表7.2「エージェントプロンプトコマンド」）は、リソースの管理（可用性、検出の実行、監視情報の確認）、エージェント自体の管理（サーバーへの登録、プラグインの読み込み、構成設定の表示または再読み込みなど）に使用できます。

表7.2 エージェントプロンプトコマンド

prompt コマンド	description
avail	従来のリソースの可用性を提供します。
config	エージェントの設定を管理します。
debug	エージェントのデバッグに役立つ機能を提供します。
検出	サーバースキャン検出の実行をプラグインに要求します。
ダウンロード	JBoss ON サーバーからファイルをダウンロードします。
dumpspool	コマンドの spool ファイルにあるエントリーを表示します。
exit	エージェントの通信サービスをシャットダウンし、エージェントを強制終了します。
フェイルオーバー	高可用性サーバーのフェイルオーバーリストを表示または更新します。
gc	ガベージコレクターを呼び出してメモリーを解放するのに役立ちます。
getconfig	1 つ、複数のエージェント設定設定、またはすべてのエージェント設定設定設定を表示します。
help	指定コマンドのヘルプを表示します。
識別	リモートサーバーの特定を要求します。
インベントリー	現在のリソースのインベントリーに関する情報を提供します。
log	ログメッセージの設定を行います。
metrics	は、エージェントメトリクスを表示します。
ネイティブ	ネイティブシステム情報にアクセスします。

prompt コマンド	description
pc	プラグインコンテナーおよびデプロイされたすべてのプラグインを開始して停止します。
PING	JBoss ON サーバーを ping します。
piql	PIQL クエリーを実行して、実行中のプロセスを検索します。
plugins	エージェントプラグインを、サーバーからの最新バージョンで更新します。
quit	エージェントプロンプトを終了します（エージェントを停止せずに）。
登録	このエージェントを JBoss ON サーバーに登録します。
スケジュール	指定されたリソースの測定スケジュール情報を取得します。
sender	コマンド送信を開始または停止するためにコマンドセンダーを制御します。
setconfig	エージェント設定の優先度を設定します。
設定	一連の質問を行い、エージェントの設定を設定します。
shutdown	エージェントを強制終了せずに、すべての通信サービスをシャットダウンします。
sleep	エージェントプロンプトを一定期間スリープ状態にします。
start	リモート要求を許可できるように、エージェントがサービスを起動します。
timer	別のプロンプトコマンドを実行するのにかかる時間。
update	エージェントの更新機能を提供します。
version	エージェントのバージョン情報を表示します。

バグを報告します。

7.4. エージェントおよびリソースのシステムユーザーとの対話

エージェントは特定のシステムユーザーとして実行されるため、JBoss ON によって管理される JBoss や Apache などのサーバーも実行できます。検出などの多くのエージェント管理タスクに関する一般的

な仮定は、エージェントユーザーがリソースユーザーと同じであることです。ユーザーが異なる場合は、リソースの検出および管理方法に影響を及ぼす可能性があります。

JBoss ON が管理するサーバーの一般的なタイプは以下のとおりです。

- JBoss EAP サーバー
- PostgreSQL データベース
- Tomcat サーバー
- Apache サーバー
- 汎用 JVM

JBoss ON エージェントが開始する一部の管理操作では、エージェントのシステムユーザーは関与することはありません。たとえば、JBoss EAP プラグインは JBoss EAP 自体が管理する認証メカニズムを使用して EAP インスタンスに接続するため、システム ACL やユーザーパーミッションは必要ありません。ユーザーが JBoss EAP インスタンスにアクセスできる限り、すべてが機能します。

Agent and Resource Users のスプレッドシート

PostgreSQL

監視および検出には影響しません。

設定の表示および編集には、エージェントユーザーに PostgreSQL 設定ファイルへの読み取り/書き込み権限が必要です。

apache

監視および検出には影響しません。

設定の表示および編集には、エージェントユーザーに Apache 設定ファイルへの読み取り/書き込み権限が必要です。

Tomcat

同じユーザーを使用する必要があります。またはエージェントを検出できません。

JMX サーバーまたは JVM

JMX リモーティングを使用する場合は、ユーザーごとに問題ありません。異なるユーザーやアタッチ API では検出できません。

JBoss AS/EAP

ユーザーはすべて問題ありませんが、run.jar で読み取り権限が必要で、run.jar のすべての先祖ディレクトリーに対して実行および検索パーミッションが必要になります。

バグを報告します。

7.4.1. エージェントユーザー

通常、エージェントは管理リソースと同じユーザーとして実行され、これが設定の最もクリーンなオプションであることを前提としています。

エージェントインストーラーの JAR ファイルから JBoss ON エージェントをインストールする場合、エージェントのインストールファイルを所有するシステムユーザーとグループは JAR をインストールしたユーザーと同じです。したがって、特別なシステムユーザーを作成または選択でき、そのユーザーがエージェントをインストールできます。

バグを報告します。

7.4.2. エージェントユーザーおよび検出

エージェントは、PID やプロセス、起動スクリプトなどの特定の共通プロパティーを検索してリソースを検出します。

このエージェントがリソースユーザーとして上位の権限を持っているかどうかは必ずしも問題ありません。

ほとんどのリソースでは、エージェントはそのリソースの設定への読み取りアクセスを必要とします。Apache や Postgres などのリソースでは、エージェントがリソース設定を読み取ることができる限り、リソースを検出できます。

他のリソースに関して、agent ユーザーには、非常に特殊なパーミッションが必要です。

- JBoss EAP リソースでは、エージェントにファイルに対する読み取り権限が必要です。さらに、**run.jar** ファイルへのパス内のすべてのディレクトリーに対して実行および検索のパーミッションが必要になり **run.jar** ます。
- Tomcat サーバーは、JBoss ON エージェントと Tomcat サーバーが同じユーザーとして稼働している場合にのみ検出できます。エージェントが root として実行されていても、Tomcat サーバーがエージェントとは異なるユーザーとして実行されている場合は検出できません。
- JVM または JMX サーバーが JMX リモーティングで実行されている場合は、エージェントが別のユーザーとして実行されているかどうかを検出できます。ただし、attach API を使用して実行している場合は、リソースを検出するためのエージェントと同じユーザーとして実行する必要があります。

バグを報告します。

7.4.3. ユーザーおよび管理タスク

エージェントが実行しているシステムユーザーは、いくつかの一般的なエージェントタスクに影響します。

- 検出
- アプリケーションのデプロイ
- スクリプトの実行
- 起動、停止、および再起動の操作の実行
- JBoss ON UI を使用した子リソースの作成
- リソース設定の表示および編集

実行するタスクと、権限または承認のためにリソースまたはオペレーティングシステムの制限に基づいて、その操作を実行する必要があるタスクを決定することが重要になります。

一部のアクション： 検出、アプリケーションのデプロイ、または子リソースの作成 - エージェントユーザーにパーミッションを付与するシステム ACL を設定するだけで十分です。これは、 で説明してい「root 以外のユーザーとしてエージェントの実行」 ます。

操作の実行やスクリプトの実行には、エージェントユーザー以外のユーザーとしてタスクを実行する必要があります。これはを使用して実行でき **sudo**ます。

いずれの方法でも、JBoss ON ユーザーに、操作の実行に必要なすべてのシステムパーミッションを付与することが目的です。

バグを報告します。

7.4.4. JBoss ON 操作での sudo の使用

使用の時間は、サービス **sudo** の起動やプロセス、リソースユーザーが所有するスクリプトなど、長時間実行される操作に使用されます。スクリプトを実行するユーザーは、適切な承認とパーミッションを持つため、リソースユーザーと同じである必要があります。

ユーザーは実際に同じであることや、JBoss ON ユーザーに指定コマンドへの **sudo** 権限を付与できます。

エージェントユーザーのパーミッションを昇格する場合は、以下の 2 つの項目が true である必要があります。

- パスワードのプロンプトなしでユーザーとの対話を必要としません。
- エージェントが変数をスクリプトに渡すことができるはずです。

リソーススクリプトを設定するには **sudo**、以下を行います。

1. JBoss ON エージェントユーザーに特定のスクリプトまたはコマンドへの **sudo** 権限を付与します。たとえば、**jbossadmin** ユーザーとしてスクリプトを実行するには、以下を実行します。

   ```
   [root@server ~]# visudo

   jbosson-agent     hostname=(jbossadmin)  NOPASSWD: /opt/jboss-eap/jboss-as/bin/*myScript*.sh
   ```

   NOPASSWD オプションを使用すると、パスワードを要求せずにコマンドが実行されます。

   重要

   JBoss ON は、EAP インスタンスの開始時に、コマンドライン引数を start スクリプトで渡します。これは、**sudoers** エントリーに完全なコマンドラインスクリプト（引数を含む）を含めるか、ラッパースクリプトの **sudo -u user** コマンドを使用するか、スクリプト接頭辞のいずれかを指定して実行できます。

   2 つ目のオプションには、簡単な **sudoers** エントリーがあります。

2. 使用するラッパースクリプトを作成または編集します。リソースのスクリプトを直接呼び出す代わりに、スクリプトの実行 **sudo** に使用するラッパースクリプトを呼び出します。

注記

EAP 起動スクリプトでは、別のラッパースクリプトを作成する代わりに、接続設定にスクリプト接頭辞を設定できます。

```
/usr/bin/sudo -u jbosson-agent
```

たとえば、開始スクリプトラッパーの場合は、**start-myScript.sh**以下を行います。

```
#!/bin/sh
# start-myScript.sh
# Helper script to execute start-myConfig.sh as the user jbosson-agent
#
sudo -u jbosson-agent /opt/jboss-eap/jboss-as/bin/start-myConfig.sh
```

3. スクリプトで渡す引数または設定を使用して、開始 **run.sh** スクリプトを作成します。たとえば、**start-myConfig.sh**以下のようになります。

```
nohup ./run.sh -c MyConfig -b jonagent-host 2>&1> jboss-MyConfig.out &
```

バグを報告します。

7.5. ROOT 以外のユーザーとしてエージェントの実行

一部のリソース情報にアクセスするには、エージェントがリソース自体への root アクセスを持っている必要があります。ただし、セキュリティー上、多くの管理者は、エージェントプロセスを root として実行しません。

Red Hat Enterprise Linux では、エージェントを root 以外のユーザーとして実行している間に、エージェントを特定リソースにアクセスを許可できます。これは、リソースのローカルディレクトリーまたはファイルにローカル **アクセス制御ルール** を設定することで行います。

注記

この例では、PostgreSQL データベースの ACL を設定します。**setfacl** コマンドで指定するディレクトリーおよびファイルは、リソースタイプによって異なります。

1. root でシステムにログインします。

2. システムに acl パッケージがインストールされていることを確認します。

```
# rpm -q acl
acl-2.2.39-6.el5
```

acl オプションはファイルシステムに適用する必要があります。これには、**/etc/fstab** ファイルを編集するか、を使用し **tune2fs**ます。例：

```
# vim /etc/fstab

LABEL=/          /          ext3    defaults,acl    1 1
...
```

次に、ファイルシステムを再マウントします。

```
# mount -o remount /
```

3. 必要に応じて、エージェントに使用するシステムユーザーを作成します。

```
useradd jbosson-agent
```

4. PostgreSQL の場合、エージェントは **postgresql.conf** ファイルにアクセスできる必要があります。PostgreSQL ディレクトリーを開きます。

```
# cd /var/lib/pgsql
```

5. エージェントユーザーに **postgresql.conf** ファイルの読み取りおよび書き込みアクセスを付与します。例：

```
# setfacl -m u:jbosson-agent:rw $PGDATA/postgresql.conf
```

6. 次に、**data**/ ディレクトリーへのアクセスを agent ユーザーに付与します。例：

```
# setfacl -m u:jbosson-agent:x $PGDATA
```

7. **getfacl** コマンドを使用して、新しい ACL が正しく追加されたことを確認します。

```
# getfacl .
# file: .
# owner: postgres
# group: postgres
user::rwx
user:jbosson-agent:--x
group::---
mask::--x
other::---
```

バグを報告します。

7.6. エージェントのデバッグモードの有効化

JBoss ON サーバーなどの JBoss ON エージェントはロギングに使用 **log4j** します。エージェントのパフォーマンスまたはサーバーエージェント通信をトラブルシューティングするには、エージェントのデバッグロギングを有効にし、**log4j** デバッグログを有効にします。

ログファイルは ***agentRoot*/rhq-agent/logs** ディレクトリーにあります。

バグを報告します。

7.6.1. 環境変数の使用

デバッグロギングを有効にする最も簡単な方法は、 **RHQ_AGENT_DEBUG** エージェントを起動する前の任意の値への環境変数。エージェントを起動すると、ランチャースクリプトとエージェント自体の両方がデバッグメッセージを出力します。

サービスラッパーを使用して JBoss ON エージェントが Microsoft Windows で実行している場合は、設定 **RHQ_AGENT_DEBUG** サービスをインストールします。

```
rhq-agent-wrapper.bat install
```

バグを報告します。

7.6.2. log4j の優先度の設定

log4j カテゴリーはロギングレベルの **優先順位** をサポートします。つまり、エージェントのさまざまな領域をログレベルごとに設定できます。

注記

は設定しません。 **RHQ_AGENT_DEBUG log4j.xml** ファイルに優先順位を設定する場合の環境変数。環境変数によりこの **log4j.xml** 設定が上書きされます。

カテゴリーのデバッグロギングを有効にするには、priority の値を **DEBUG**以下のように変更します。

1. エージェント **log4j** ファイルを開きます。

```
# vim agentRoot/rhq-agent/conf/log4j.xml
```

2. カテゴリーの ***priority*** 要素をリセットします。デフォルトでは、エージェント設定には、送受信サーバーエージェント通信とベース両方のログ記録があります。 **org.rhq** クラス。オプションで、プラグインクラスローダーと JBoss リモーティング通信に対してロギングを有効にできます。

```
<!-- ================ -->
<!-- Limit categories -->
<!-- ================ -->

<!-- RHQ -->
<category name="org.rhq">
  <priority value="INFO"/>
</category>

<!-- RHQ outgoing command tracing  - set to TRACE to trace commands sent by the agent
-->
<category
name="org.rhq.enterprise.communications.command.client.OutgoingCommandTrace">
    <priority value="NONE"/>
    <appender-ref ref="COMMANDTRACE"/>
</category>
...
```

3. エージェントを再起動して、新しい設定を読み込みます。

log4j ファイル形式の詳細は Apache log4j ドキュメントを参照してください。

バグを報告します。

7.6.3. Agent debug プロンプトコマンドの使用

デバッグロギングは、エージェントコマンドプロンプト(「エージェントのコマンドプロンプトを開く」)で **debug** コマンドを使用して有効にできます。

--enable オプションを使用すると、**log4j** デバッグログが有効になります。

```
> debug --enable
log4j:WARN No appenders could be found for logger
(org.rhq.core.pc.measurement.MeasurementCollectorRunner).
log4j:WARN Please initialize the log4j system properly.
Switched to log file [log4j-debug.xml]. Root log level is [DEBUG]
started>
```

サーバーエージェント通信層専用のデバッグロギングを有効にするには、**--comm** オプションを true に設定します。

```
> debug --comm=true
Agent-server communications tracing has been enabled.
You may set the following, additional configuration settings
to collect more detailed trace data. You can set these
using the setconfig prompt command. Please refer to the
documentation for more information on these settings. The
values you see here are the current settings:
  rhq.trace-command-config=true
  rhq.trace-command-response-results=256
  rhq.trace-command-size-threshold=99999
  rhq.trace-command-response-size-threshold=99999
```

この **debug** コマンドは、**--threaddump** オプションを使用してサーバーおよびシステム管理ハンドラーに対してすべてのエージェントスレッドを確認することもできます。これにより、スレッドが実行されているか、スレッドごとに、エージェントが直面しているかどうかに関わらず、各スレッドの情報を出力します。例：

```
> debug --threaddump
"DestroyJavaVM" Id=47 RUNNABLE


"RHQ Agent Prompt Input Thread" Id=46 RUNNABLE


"EventManager.sender-2" Id=49 TIMED_WAITING on
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@17d7c01
        at sun.misc.Unsafe.park(Native Method)
        -  waiting on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@17d7c01
        at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226)
        at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSy
nchronizer.java:2081)
        at java.util.concurrent.DelayQueue.take(DelayQueue.java:193)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolEx
ecutor.java:688)
        at
java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolEx
ecutor.java:681)
```

```
        at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1043)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1103)
...
```

バグを報告します。

7.7. エージェントの IP アドレスの変更

エージェントの IP アドレスは、設定設定で ***rhq.communications.connector.bind-address*** 設定されます。これは、サーバーソケットの起動時にエージェントがバインドする IP アドレスです。これは、エージェントがサーバーから受信メッセージをリッスンするのに使用するサイトです。

注記

agent-configuration.xml ファイルの編集は行わないでください。初期設定が完了したらエージェントはこのファイルを使用しないため、エージェントによるファイルの変更は自動的に読み込まれません。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. ***rhq.communications.connector.bind-address*** 設定の優先 **setconfig** 度と新しい値でを送信します。

   ```
   > setconfig rhq.communications.connector.bind-address=1.2.3.4
   ```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

バグを報告します。

7.8. エージェントのリソースとしての管理

エージェントはリソースとして JBoss ON インベントリーに追加できるため、その動作とメトリクスを監視して適切に機能し、他のリソースと同様にアラートや操作を起動できます。

重要

エージェントがシャットダウンした場合、JBoss ON GUI を使用して起動コマンドを実行するのに有効なエージェントがないため、このエージェントを再起動することはできません。エージェントを再起動するには、エージェントリソース自体ではなく、ランチャースクリプトの子リソースで再起動操作を使用します。

shutdown 操作は、エージェントプロセスがデーモンとして実行している場合は強制終了します。エージェントがコマンドプロンプトとして実行している場合は、shutdown 操作はエージェントを停止しますが JVM ではなく、プロンプトコマンドはエージェントのコマンドプロンプトで実行できます。

エージェントがインベントリーにインポートされると、複数の子リソースも自動的に追加されます。これらはに記載されてい エージェントのリソース ます。

エージェントのリソース

エージェント自体

重要

通常、agent リソースの操作は agent プロセスディレクトリーには影響を与えません。これらのオプションは、JVM 設定やプロセス、JRE オプションに対する制御は提供しません。JVM の制御は、エージェントリソースではなく、エージェント子リソースを介して行われます。

エージェントおよびその内部コンポーネントの監視、設定、および制御機能を提供します。これらの設定は **agent-configuration.xml** ファイルで定義された設定に対応し、Java 設定としてエージェントマシンで永続化されます。

エージェント測定サブシステム

エージェントの測定収集およびレポートコンポーネントにデータを提供します。

エージェント JVM

エージェントを実行している JVM の詳細な監視および管理（クラスローダー、スレッド、およびメモリー管理サブシステムを含む）を提供します。これは子サーバーです。

エージェント環境の設定スクリプト

エージェントランチャースクリプトの起動時にサーバーを設定する設定済みの環境変数。

エージェントプラグインコンテナー

組み込みプラグインコンテナーにビューを提供し、プラグインコンテナーに直接関連する管理データを提供します。プラグインコンテナーはエージェント内で実行され、これらのプラグインの実行に必要なすべての管理プラグインおよびインフラストラクチャーのデプロイメントを処理します。

Java サービスラッパーランチャー(Windows)

Java サービスラッパーを制御します。これは、エージェントを Windows サービスとしてインストールして実行するサードパーティーのライブラリーです。Java サービスラッパーには、読み取り専用ファイルである 1 つの主要設定 **rhq-agent-wrapper.conf** ファイルがあります。これにより、エージェントが適切に起動および動作するために必要な設定の基本設定を定義します。

設定の追加グループは、エージェントの環境をカスタマイズできます。環境グループは、一般的な

環境スクリプトで定義される環境変数に加えて、メインの設定で使用される環境変数を定義します。Includes グループは、すべてのラッパー設定を定義します。デバッグを設定したり、新しい JVM オプションをエージェント JVM に渡す以外に、これらのグループはほとんど編集しないでください。

エージェントランチャースクリプト(UNIX)

エージェントを制御します。エージェントがランチャースクリプトで生成されたバックグラウンドデーモンプロセスとして実行している場合は、ランチャースクリプトは停止または再起動します。追加の設定はありません。ランチャースクリプトは、環境セットアップスクリプトで設定されます。

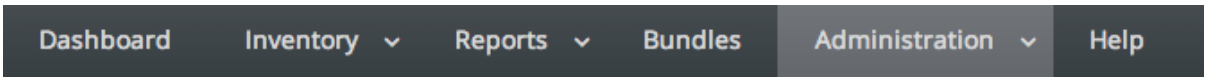
バグを報告します。

7.9. エージェントのクエット時間（タイムアウト時間）の設定

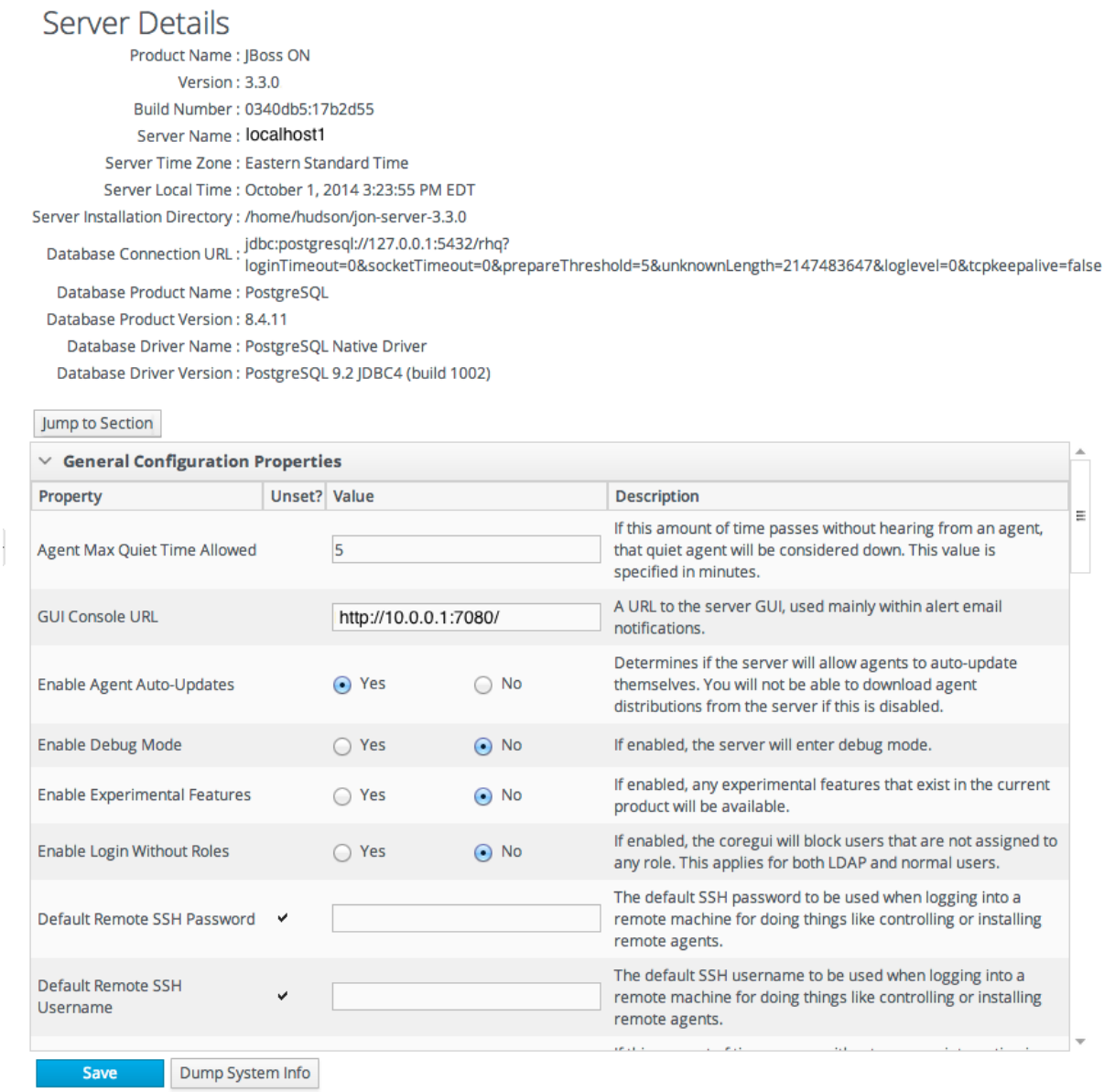
JBoss ON サーバーは、エージェントがダウンしているとみなすまで一定期間待機し、エージェントから応答します。デフォルト設定では、エージェントは毎分サーバーにハートビートを送信します。エージェントが 5 分間サイレントである場合、サーバーはエージェントがダウンしていることを認識し、プラットフォームとすべての子に利用不可のマークを付けます。

この設定では、エージェントの **quiet 時間** は JBoss ON 全体の設定です。クラウドのすべての JBoss ON サーバーは同じコア設定を使用します。

1. トップメニューの **Administration** タブをクリックします。



2. 左側の **Configuration** メニューテーブルで、**System Settings** 項目を選択します。
3. メインの作業領域の **JON General Configuration Properties** セクションまでスクロールします。



4. を希望 **Agent Max Quiet Time Allowed** する間隔に変更し、サーバーがエージェントハートビートを待機してから、エージェントをダウンとしてマークします。

5. **Save** ボタンをクリックします。変更は、すべてのサーバーに即座に適用されます。

バグを報告します。

7.10. エージェント更新の設定

JAR ファイルからエージェントがインストールされると、エージェントがサーバーからバージョン更新を自動的に受信できるようにする設定プロパティーがあります。つまり、サーバーが更新されるたびに、そのサーバーが管理するすべてのエージェントがサーバーと同じバージョンに自動的に更新されます。（サーバーとエージェントが同じ JBoss ON バージョンとして実行している必要があるため、これは有益です。）

単一のエージェントの場合、これはエージェント設定ファイルで設定されます。

```
<entry key="rhq.agent.agent-update.enabled" value="true" />
```

true を指定すると、エージェントはサーバーから更新を受信できます。

重要

この値は、RPM によりインストールされるすべてのエージェントについて false に設定されます。**RPM からエージェントがインストールされている場合は、エージェントの更新設定は常に false である必要があります。**

1 つのエージェントに対するエージェントの更新設定は、設定プロパティーを編集してリセットできます。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. ***rhq.agent.agent-update.enabled*** 設定設定の新しい値を使用して **setconfig** コマンドを実行します。

```
> setconfig rhq.agent.agent-update.enabled=false
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

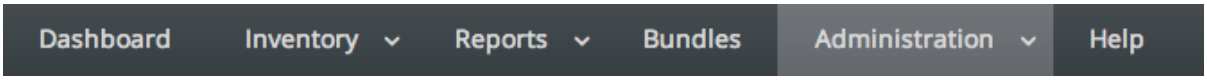
```
agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

agentRoot/rhq-agent/bin/rhq-agent.sh
```

すべてのエージェントが RPM を使用してインストールされている場合や、すべてのエージェントの自動アップグレードを防ぐための環境的な理由がある場合、JBoss ON サーバークラウドで自動アップグレードを無効にすることができます。つまり、エージェント設定に関係なく、JBoss ON サーバーは更新パッケージをエージェントが利用できるようにしません。

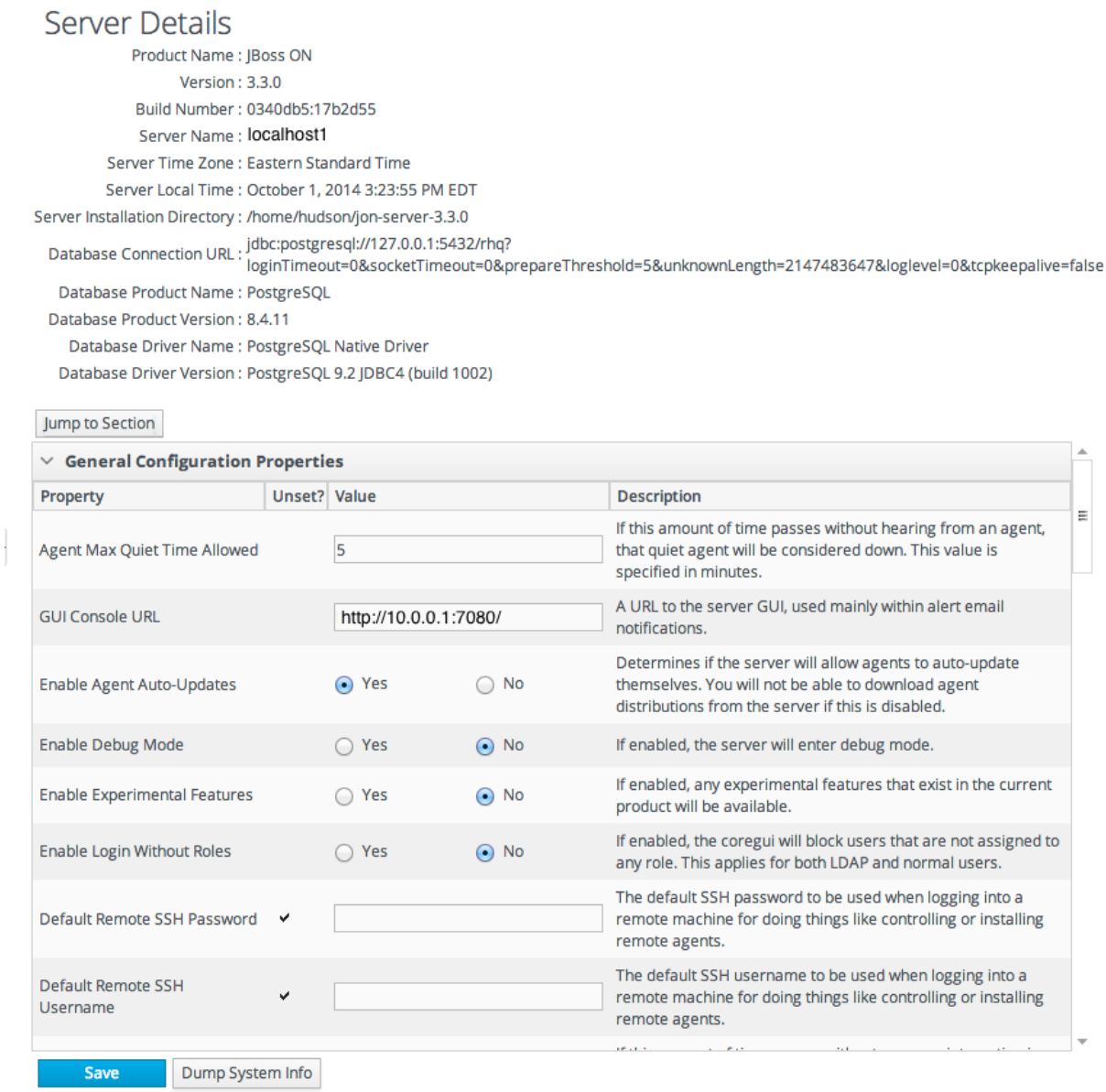
エージェント更新サーバー設定を変更するには、以下を実行します。

1. トップメニューの **Administration** タブをクリックします。



2. 左側の **Configuration** メニューテーブルで、**System Settings** 項目を選択します。

3. メインの作業領域の **JON General Configuration Properties** セクションまでスクロールします。



4. **Enable Agent Auto-Updates** ラジオボタンをに設定し **No**ます。これにより、サーバーがインストールされたエージェントに新しいバイナリーを送信できなくなります。

5. **Save** ボタンをクリックします。変更は、すべてのサーバーに即座に適用されます。

バグを報告します。

7.11. エージェントの永続設定の管理

エージェントは Java プラットフォームの Java 設定を使用して設定を保存します。一般的な Java 設定については、Java ドキュメントで説明されてい http://download.oracle.com/javase/1.5.0/docs/guide/preferences/index.html ます。JBoss ON は、ユーザーの設定をバッキングストアのルートノードに保存します。

バッキングストアの場所は、システムによって異なります。

- Windows では、バッキングストアは Windows レジストリーにあります。
- Linux および Unix システムでは、バッキングストアは、のエージェントユーザーのホームディレクトリーにあり ~/**.java**ます。

重要

エージェントの設定は、ユーザーがエージェントを実行しているユーザーによって決まります。エージェントが 1 人のユーザーとして実行され、その後別のユーザーとして実行される場合、設定に異なるバッキングストアを使用するため、エージェントは 2 回目とは異なる設定を持ちます。

たとえば、エージェントがという名前のシステムユーザーが設定されている場合 **jsmith**、永続化された設定はになり **~jsmith/.java**ます。次にエージェントが root ユーザーとしてバックグラウンドサービスとして実行されるように設定されている場合、エージェントはその設定を探し **~root/.java**、異なる設定を見つけます。

つまり、あるユーザーがエージェントのインストール時にエージェントを設定するのに使用する場合、同じユーザーを使用してエージェントを後に実行する必要があります。そうしないと、エージェントは設定を失い、新しいユーザーで再設定する必要があります。

エージェントは、バッキングストアから実行するのに使用する設定を取得します。エージェントがバッキングストアを初期化する必要がある場合や、エージェントが開始した状態で新しい設定を読み込む必要がある場合にのみ **--cleanconfig**、**agent-configuration.xml** ファイルの設定を読み込みます。

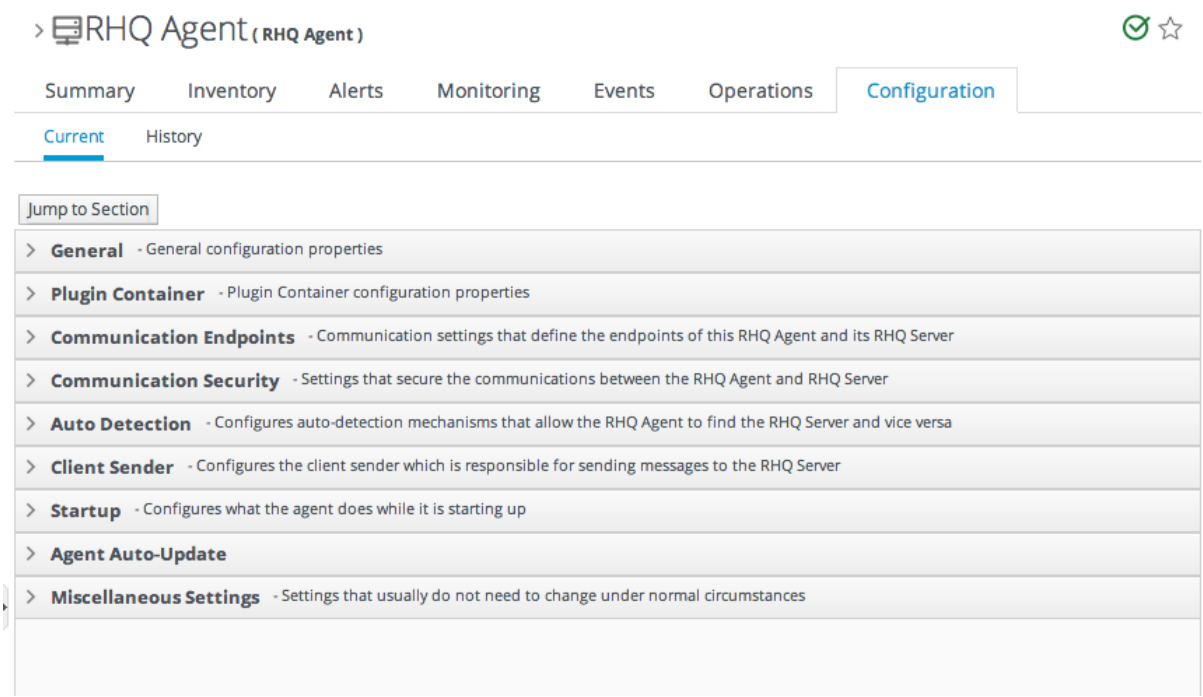
バグを報告します。

7.11.1. 永続設定の表示

エージェント設定は Java **設定** で構成されており、各 JBoss ON ユーザーに対して永続化されます。設定が永続化される方法はオペレーティングシステムによって異なります。Windows は、Unix がユーザーのホームディレクトリーに保持する一方で、レジストリーに設定を保存します。

エージェント設定は、**rhq-agent-env.sh** ファイルを介して設定およびロードできるいくつかのパラメーターを除き、最初に設定し、その後にデータベースに永続化される際に読み込まれます。エージェントの永続化設定は、以下の複数の方法で表示できます。

1. JBoss ON インベントリーにエージェントがある場合は、完全な設定が **Configuration** タブから表示され、各設定エリアを表示する折りたたみのテーブルが表示されます。



2. 設定は、エージェントの **getconfig** または **config** prompt コマンドで返すこともできます。これらのコマンドは、エージェントがコマンドプロンプトから実行しているか、エージェントリソースの JBoss ON UI で **Execute Command Prompt** 操作を実行している場合はターミナルから実行できます。

```
> getconfig
rhq.agent.agent-update.enabled=true
rhq.agent.client.command-preprocessors=org.rhq.enterprise.agent.
SecurityTokenCommandPreprocessor: org.rhq.enterprise.agent.
ExternalizableStrategyCommandPreprocessor
rhq.agent.client.command-spool-file.compressed=true
rhq.agent.client.command-spool-file.name=command-spool.dat
rhq.agent.client.command-spool-file.params=10000000:75
rhq.agent.client.command-timeout-msecs=600000
rhq.agent.client.max-concurrent=5
rhq.agent.client.max-retries=10
rhq.agent.client.queue-size=50000
rhq.agent.client.queue-throttling=200:2000
rhq.agent.client.retry-interval-msecs=15000
rhq.agent.client.send-throttling=100:1000
rhq.agent.client.server-polling-interval-msecs=60000
rhq.agent.configuration-schema-version=5
rhq.agent.configuration-setup-flag=true
rhq.agent.data-directory=data
rhq.agent.disable-native-system=false
rhq.agent.name=localhost.localdomain
rhq.agent.plugins.directory=plugins
...
```

3. エージェント設定は Java 設定で永続化されるため、Java 設定を調べるツールを使用して、永続化された設定を表示できます。

警告

サードパーティーツールを使用する優先度の値は変更しないでください。エージェントの優先度が低い値に設定すると、エージェントを完全に無効にできます。

バグを報告します。

7.11.2. Persisted 設定の優先順位の変更(Agent Preferences)

エージェントの設定は、最初に起動時にセットアッププロンプトに入力 **agent-configuration.xml** した値で読み取られ、上書きされます。エージェントは最初に設定されると、設定がパージされ、リロードされない限り、エージェントはその設定を保持し、**agent-configuration.xml** 再び参照しないようになります。ほとんどの設定変更は **rhq-agent-env.sh** ファイルに対して行われ、エージェントが起動するたびに読み込まれます。

エージェントプロンプトで **setconfig** コマンドを使用して、永続化された設定を変更できます（設定ファイルを編集せずに）。

1. エージェントプロンプトを開きます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh
```

2. 編集する希望の名前とその新しい値 **setconfig** で送信します。プリファレンス名は、**agent-configuration.xml** ファイル内のエントリー名になります。例：

```
> setconfig rhq.agent.client.max-concurrent=20
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

```
agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

agentRoot/rhq-agent/bin/rhq-agent.sh
```

バグを報告します。

7.11.3. 永続的な設定の上書き

Java バッキングストアおよびエージェントの **agent-configuration.xml** ファイルの設定は、**-D** オプション、設定パラメーター名、およびエージェントの起動時に新しい値を使用して上書きできます。

たとえば、起動時にエージェントが JBoss ON サーバーを検出するまでの一時的な値を設定するには**rhq.agent.wait-for-server-at-startup-msecs**、引数 start コマンドでこの引数を渡します。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -Drhq.agent.wait-for-server-at-startup-msecs=90000
```

バグを報告します。

7.11.4. エージェント設定における機密情報の保護

重要

デフォルトでは、**{RHQ_AGENT_HOME}/conf/agent-configuration.xml** ファイルのすべてのパスワードはアップグレードおよびインストール時に保護されます。このプロセスは、ユーザーが追加のプロパティーをエンコードする必要がある場合にのみ必要です。

JBoss ON では、**{RHQ_AGENT_HOME}/conf/agent-configuration.xml** ファイル内のほぼすべてのプロパティーを保護することがサポートされます。JBoss ON は、プロパティーを難読化するための **{RHQ_SERVER_HOME}/bin/** ディレクトリーに **rhq-encode-value.sh** および **rhq-encode-value.bat** スクリプトを提供します。たとえば **agent-configuration.xml**、プロパティー値は **RESTRICTED::** 形式を使用してエンコードされ、すべてのパスワードはインストール時にデフォルトで保護されます。

バグを報告します。

7.11.4.1. エンコーディングでの rhq-encode-value スクリプトの使用

注記

rhq-encode-value スクリプトはサーバーおよびエージェント設定ファイルのエンコーディングに使用できますが、JBoss ON はサーバーレベルでの **rhq-encode-value** スクリプトのみを提供します。**agent-configuration.xml** ファイルの値をエンコードするユーザーは、JBoss ON サーバーレベルで **rhq-encode-value** スクリプトを呼び出す必要があります。

rhq-encode-value スクリプトを呼び出すと、必要なプロパティーと値をエンコードするように求められます。

```
> ./rhq-encode-value.sh
Property rhq.autoinstall.server.admin.password [y/n]: n
Property rhq.server.database.password [y/n]: n
Property: rhq.protect.property
Value: 1234
*** !!! WARNING !!!
*** Both standalone-full.xml and rhq-server.properties need to be updated if a property from rhq-
server.properties is used in standalone-full.xml
*** !!! WARNING !!!
***
***
*** Encoded password for rhq-server.properties:
***    rhq.protect.property=RESTRICTED::-299a94df3b478ca8
***
*** Encoded password for standalone-full.xml with vault with password as default value:
***    ${VAULT::restricted::rhq.protect.property::-299a94df3b478ca8}
***
*** Encoded password for standalone-full.xml with vault without default:
***    ${VAULT::restricted::rhq.protect.property:: }
***
*** Encoded password for agent-configuration.xml:
***    <entry key="rhq.protect.property" value="RESTRICTED::-299a94df3b478ca8" />
***
*** Please consult the documentation for additional help.
```

スクリプトの実行後は、から生成された値をコピーし、貼り付ける **rhq-encode-value** 必要があり**agent-configuration.xml**ます。

バグを報告します。

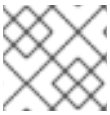
7.12. エージェント JVM の管理

エージェントは Java 仮想マシンで実行され、その動作の特性を **rhq-agent-env.sh** ファイルに定義して JVM に渡すことができます。

JVM オプションを設定する引数は 2 つあります。

- **RHQ_AGENT_JAVA_OPTS** デフォルトの JVM 設定のいずれかをリセットします。
- **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** デフォルト設定を変更せずに JVM 設定を追加します。

JVM 設定の詳細は、http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp および その他の Sun JVM のドキュメントを参照してください。



注記

JVM 設定を変更した後にエージェントを再起動して、新しい設定を読み込む。

バグを報告します。

7.13. 共有ディレクトリーまたはアカウントを使用した複数のエージェントのインストール

複数のシステムで実行している複数のエージェントは、同じシステムユーザーアカウントを共有できます。異なるシステムで JBoss ON エージェントに同じユーザーを使用し 、 そのシステムユーザーがすべて同じ共有ホームディレクトリーを使用する場合、デフォルトで同じエージェント設定の場所と優先度が共有されます。エージェントが Java プリファレンスを使用する方法により、エージェントが相互の設定を上書きするのを防ぐために特別なエージェント設定が必要になります。

同じドメインユーザーが JBoss ON エージェントに使用されると、Windows システムで同様の状況が発生する可能性があります。この場合、Java プリファレンスはドメインユーザーが使用し、ローカルユーザーのプロファイルに読み込まれるレジストリーキーに保存されます。同じドメインユーザーを使用するエージェントが複数ある場合、それらは相互のレジストリーキーを上書きします。

設定後のすべてのエージェント設定は、の説明に従って Java 設定ノードに保管され 「エージェントの永続設定の管理」 ます。デフォルト設定では、ノード名はで **default**、ノードの場所は agentUserHomeDir になり **ます/.java/.userPrefs/rhq-agent/default**。

同じファイル共有を使用して複数のエージェントがインストールされている場合は、すべて同じデフォルトノードと場所の使用を試みます。

複数のエージェントが同じ Java 設定ノードの使用を試みると、各エージェントは設定時に以前のエージェントの設定を上書きします。つまり、最新のエージェントの設定のみが保存されるので、最新のエージェントしか起動できません。独自の設定が見つからないため、以前のエージェントの起動に失敗します。

preferences ノードは 2 つの設定で一意に識別されます。

- その名前は、エージェント設定として定義されます。

- その場所（Java オプション自体）

同じホームディレクトリーで複数のエージェントを実行するには、preferreds ノードは各エージェントに対して一意に識別される必要があります。これを行う方法は複数あります。

- エージェント設定ファイルの直接編集
- 明示的な Java オプションの設定

バグを報告します。

7.13.1. 設定ファイルの編集

エージェントを最初に設定すると、エージェント設定ノードの名前は **agent-configuration.xml** ファイルに設定され、そこからロードされます。ノードの場所は、ノード名の設定から派生します。

1. 新しいノード名を使用するように **agent-configuration.xml** ファイルを編集します。

```
[rhquser@server ~]$ vim agentRoot/rhq-agent/conf/agent-configuration.xml

<node name="agent01-node">
```

2. 次に、編集した設定ファイルを読み込む **--config** オプションと、特定のノードの場所を指定する **--prefs** オプションを指定してエージェントを起動します。

```
[rhquser@server ~]$ agentRoot/rhq-agent/bin/rhq-agent.sh --prefs=agent01-node --
config=agent-configuration.xml
```

重要

agent-configuration.xml ファイルを編集してカスタム Java 設定ノードを指定すると、エージェントが再起動するたびに、**--prefs** オプションを使用してノードの場所をエージェントに渡す必要があります。

バグを報告します。

7.13.2. Java オプションの設定

agent-configuration.xml ファイルを編集するとノード名は設定されません。エージェントを起動するたびにノードの場所が渡される必要があります。

rhq-agent-env.sh ファイルに Java オプションを設定すると、 **--prefs** オプションを渡すか編集したり、設定を再読み込みすることなく、Java プリファレンスノード情報が一度設定され、永続化されるため、エージェントをサービスとして再起動できます。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

```
[rhquser@server ~]$ agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. **setconfig** コマンドを使用して、プリファレンスノードで **RHQ_AGENT_ADDITIONAL_JAVA_OPTS** 値を設定します。例：

–

```
> setconfig RHQ_AGENT_ADDITIONAL_JAVA_OPTS="-
Djava.util.prefs.userRoot=agentUserHomeDir/.java/.userPrefs/rhq-agent/agent01-node"
```

優先順位ノードは、など別の名前を持つユーザー設定ディレクトリーに配置するか **agent01-node**、共有またはファイルシステムがマウントされていない場所など **/etc/agent-preferences**、完全に異なる場所に配置することができます。

3. エージェントプロセスを再起動して、新しい設定を読み込みます。たとえば、エージェントがサービスとして実行している場合は、以下を実行します。

```
[rhquser@server ~]$ service rhq-agent-wrapper.sh stop

[rhquser@server ~]$ service rhq-agent-wrapper.sh start
```

注記

また、エージェントを停止し、**rhq-agent-env.sh** ファイルを直接編集してから、エージェントを再起動することもできます。

重要

このファイルは JBoss ON エージェントの更新時に上書きされるため、rhq-agent ランタイムディレクトリーに Java オプションを設定しないでください。rhq-agent ランタイムのデフォルトの場所はです **$USERHOME/.java/.userPrefs/rhq-agent/default**。

バグを報告します。

7.14. 検出スキャン間隔の設定

エージェントはプラットフォームを定期的にスキャンして、検出キューに追加する新しいサーバーまたはサービスを、後でインベントリーに検索します。スキャン間隔を設定するさまざまなパラメーターがあります。

- で設定したサーバーのスキャン間隔 ***rhq.agent.plugins.server-discovery.period-secs***。デフォルトは 900 秒（15 分）です。
- で設定したサービスのスキャン間隔 ***rhq.agent.plugins.service-discovery.period-secs***。デフォルトは 86400 秒（24 時間）です。
- で設定した低レベルの子サービスのスキャン間隔 ***rhq.agent.plugins.child-discovery.delay-secs***。デフォルトは 5 秒です。

  親リソースが検出されるとすぐに子が検出されます。ただし、下位レベルの子は、その後の検出スキャンで検出されます。これにより、初期インポートと次の検出スキャンの間の遅延が設定されます。

これらは **agent-configuration.xml** ファイルに設定されるため、変更を反映するには、設定を適切にリロードする必要があります。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. **setconfig** コマンドを使用して、検出スキャン間隔をリセットします。プリファレンス名は、**agent-configuration.xml** ファイル内のエントリー名になります。例：

```
> setconfig rhq.agent.plugins.server-discovery.period-secs=600

> setconfig rhq.agent.plugins.service-discovery.period-secs=1440

> setconfig rhq.agent.plugins.child-discovery.delay-secs=60
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。たとえば、エージェントがサービスとして実行している場合は、以下を実行します。

```
[root@server ~]# service rhq-agent-wrapper.sh stop

[root@server ~]# service rhq-agent-wrapper.sh start
```

バグを報告します。

7.15. エージェントのサーバーフェイルオーバー一覧の表示

JBoss ON エージェントは、管理のためにサーバーに割り当てるために高可用性に自動的に組み込まれます。エージェントサーバーの設定は、アフィニティーグループ(「アフィニティーグループの作成」)を使用して割り当てられます。エージェントの高可用性設定には、アフィニティーグループ、サーバーが現在管理されているサーバー、およびフェイルオーバーに使用できるサーバーが表示されます。

エージェントが接続する最初のサーバーは **agent-configuration.xml** ファイルで定義され、これはエージェントが初期登録リクエストを送信するサーバーです。登録後、エージェントは高可用性クラウドに参加し、監視情報、リソースの変更などの更新をクラウド内のサーバーに送信します。登録時に、エージェントは最初のアフィニティーグループ割り当てを取得します。プライマリーサーバーが登録サーバーと異なる場合、エージェントはプライマリーサーバーと通信を切り替えます。

高可用性サーバークラウドは、エージェントが正常に動作した後にサーバーとエージェント間の関係を定義するのに役立ちます。

エージェントが更新を送信するサーバーのグループは、アフィニティーグループを定義することで緩やかに制限されます。affinity グループは、エージェントがアクセスするサーバーの一覧を作成します。この一覧は順序付けられ、最初のサーバーエントリーはエージェントが接続するプライマリーサーバーになります。プライマリーサーバーが利用できない場合、エージェントはリスト内の他のサーバー間を順番にサイクルします。これにより、JBoss ON のパフォーマンスを中断せずに、エージェントは高可用性クラウドで定義されたサーバーに接続できます。

エージェントがフェイルオーバー一覧でサーバーに接続できない場合、エージェントは一時的に通信を停止し、そのメッセージをスプールします。しばらくすると、プライマリーサーバーからフェイルオーバーリストが再度実行されます。

エージェントは常に、プライマリーサーバーに接続していることを常に確認します。1時間後、デフォルトでは接続をチェックして、サーバーが使用しているサーバーがプライマリーサーバーであることを確認します。そうでない場合は、エージェントはそのプライマリーサーバーに再接続を試みます。

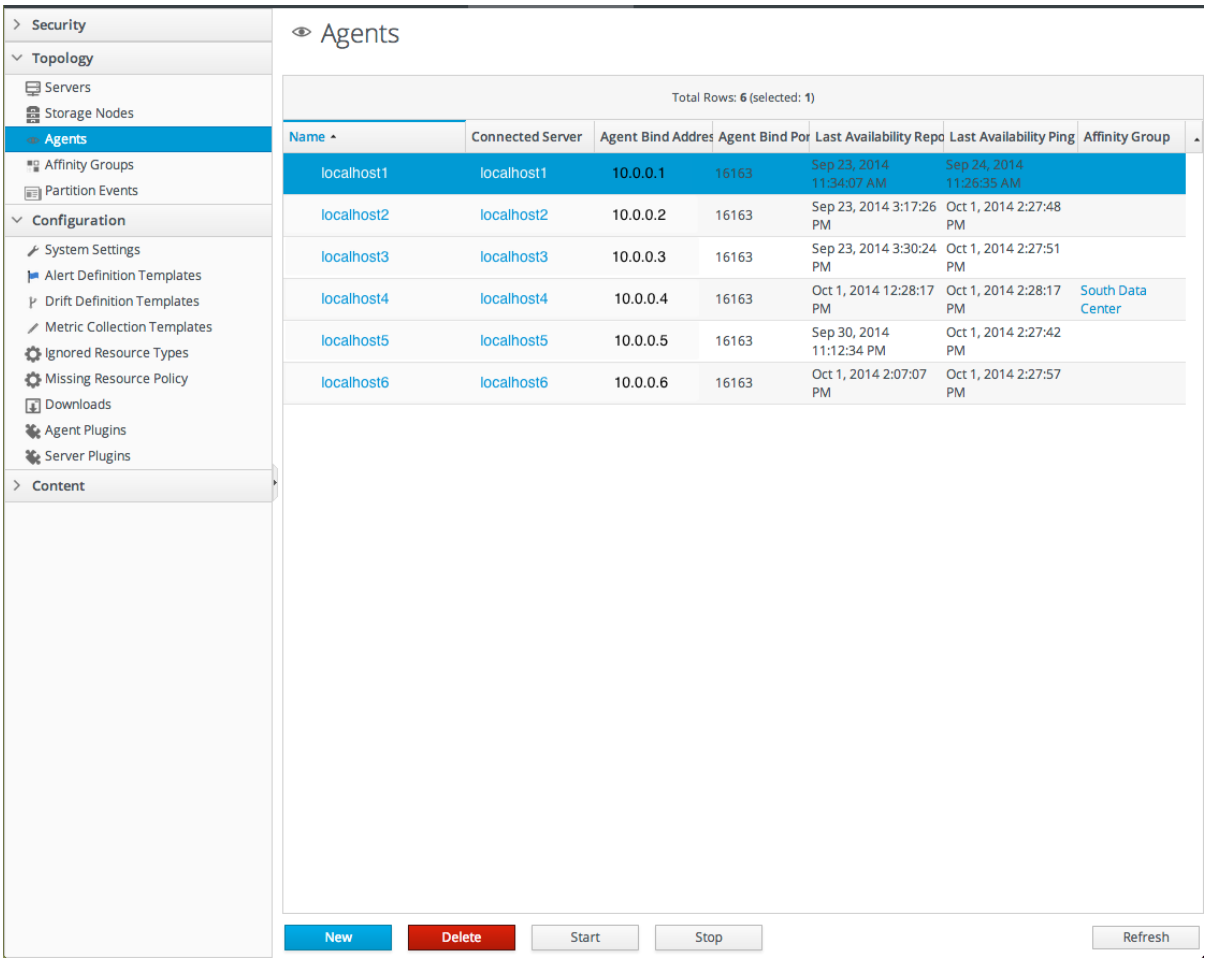
エージェントの実際のフェイルオーバーリストはサーバーによって生成され、サーバーのアフィニティーグループ設定で編集されます。エージェントがサーバーをポーリングして設定変更を行うと、新規サーバーまたはエージェントなどのアフィニティーグループへの変更が、サーバーの優先度が変更されたり、新しいグループ割り当ても毎時送信されます。

エージェントコマンドプロンプトからエージェントのフェイルオーバー一覧を表示するには、次のコマンドを実行します。

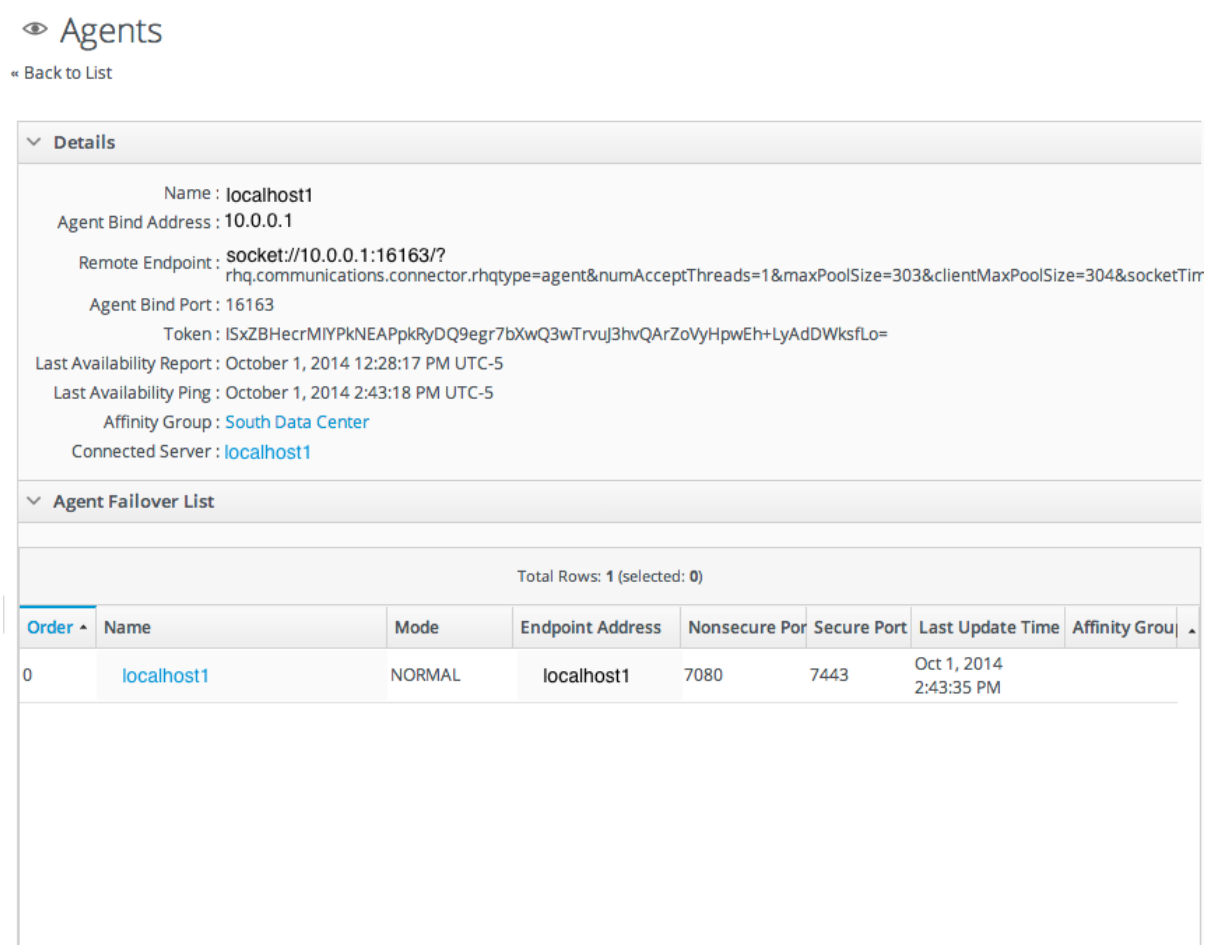
```
> failover --list localhost.localdomain:7080/7443 server2.example.com:7080/7443
1.2.34.56:7080/7443
```

UI からフェイルオーバー一覧を表示するには、次のコマンドを実行します。

1. トップメニューの **Administration** タブをクリックします。
2. をクリックし ます。
3. エージェントの高可用性ページには、高可用性に関連する項目など、エージェントに関する情報が表示されます。



   - エージェントが現在接続している JBoss ON サーバー（または、最後に接続されたサーバー）。
   - 最後のエージェントの可用性レポートがサーバーに送信された時間。
   - エージェントが割り当てられているアフィニティーグループ。
4. エージェントの名前をクリックします。これにより、エージェントのサーバーフェイルオーバーの一覧が開きます。上記の最初のサーバーはエージェントのプライマリーサーバーです。他のすべてのサーバーは高可用性クラウドで利用できます。接続されているサーバーは、通常、プライマリーサーバーがオフラインでない限り、プライマリーサーバーでもあります。



バグを報告します。

7.16. サーバーを検出またはポーリングするエージェントの設定

エージェントは JBoss ON サーバーと通信する必要があります。これは、マルチキャスト検出を使用して、主要な JBoss ON サーバーがオンラインになるか、またはオフラインになったかを監視するか、または JBoss ON サーバーを間隔でポーリングしてサーバーがオンラインであるかどうかを確認することで実行できます。

これらのポーリングメソッドは排他的ではありません。これらのポーリングメソッドはいずれも設定でき、エージェントはいずれのメソッドも使用でき、サーバーのポーリングに利用できるようになります。

サーバーをポーリングすると、サーバーがオフラインになり、サーバーがオンラインに戻ると、エージェントはコマンドおよびデータの送信を停止できます。エージェントでサーバーのポーリングが有効になっていない場合、エージェントは常にサーバーがオンラインであることを確認し、その情報をサーバーに送信します。サーバーがダウンした場合、エージェントは繰り返し **接続が拒否** されたエラーを記録します。これは（サーバーが長時間停止している場合）、エージェントのログが非常に大きくなる可能性があります。

バグを報告します。

7.16.1. JBoss ON サーバーをポーリングするための設定

最も簡単な設定では、エージェントのポーリング間隔を設定します。この方法では、エージェントは事前に定義された間隔でサーバーを ping します。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. ***rhq.agent.client.server-polling-interval-msecs*** 設定と値（ミリ秒単位）で送信 **setconfig** します。この値をゼロにするか、負の値を設定するとサーバーのポーリングが無効になります。

```
> setconfig rhq.agent.client.server-polling-interval-msecs=500
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

```
agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

agentRoot/rhq-agent/bin/rhq-agent.sh
```

バグを報告します。

7.16.2. マルチキャスト検出の設定

マルチキャスト検出は JBoss の Remoting フレームワークを使用します。これにより、サーバーが数秒以内にサーバーの行時または停止時に常にエージェントを検出できます。Remoting フレームワークを使用する場合は、マルチキャストトラフィックのサポートが必要になります。そうでない場合は、エージェントはサーバーを検出できません。これには、簡易ポーリングよりも多くの設定パラメーターがあります。

- サーバー検出とマルチキャストトラフィックの両方を有効にするように設定（***rhq.agent.server-auto-detection*** および ***rhq.communications.multicast-detector.enabled***それぞれ）。
- サーバー通信(***rhq.communications.multicast-detector.default-time-delay***)間の待機間隔。サーバーがその間隔よりも長い場合は、サーバーはオフラインと見なされます。
- Await（ハートビート）は、エージェント独自のメッセージの間隔(***rhq.communications.multicast-detector.heartbeat-time-delay***)です。この値は JBoss ON サーバーのハートビート間隔(***rhq.communications.multicast-detector.default-time-delay***)より短くする必要があります。そうしないと、不要なメッセージやネットワークトラフィックが発生します。

マルチキャスト検出を有効にするには、以下を実行します。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

```
agentRoot/rhq-agent/bin/rhq-agent.sh -n
```

2. をマルチキャスト設定 **setconfig** で送信します。time-delay の値はミリ秒単位です。

```
> setconfig rhq.agent.server-auto-detection=true
> setconfig rhq.communications.multicast-detector.enabled=true
> setconfig rhq.communications.multicast-detector.default-time-delay=75000
> setconfig rhq.communications.multicast-detector.heartbeat-time-delay=60000
```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

```
agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

agentRoot/rhq-agent/bin/rhq-agent.sh
```

バグを報告します。

7.17. エージェントのスロットリング

エージェント設定によっては、エージェントがアクセスできるリソース数と、一度に実行できるタスク数を制御します。エージェントのスロットリングには、2 つの目的があります。これは、ホスト上の（ホストマシンでのパフォーマンスを向上できる）ホストのリソース数を制限し、エージェントがデータを過負荷または一元化してサーバーを過負荷または一元化しないようにします。

に記載されている複数の異なる設定を使用して、エージェントパフォーマンスのさまざまな要素を調整表7.3「エージェント操作のスロットリング用のエージェントパラメーター」できます。これらの設定は相互に独立して動作しますが、他の値を考慮した後に設定を行うと、より効果的になる場合があります。たとえば、キューサイズは max-concurrent 設定を増やさない限り、コマンドの timeout 期間よりも大きく設定する必要があります。これらの値のいずれかを変更すると、これらの値をすべて調整することとは異なります。

表7.3 エージェント操作のスロットリング用のエージェントパラメーター

パラメーター	description
rhq.agent.client.queue-size	JBoss ON サーバーへ送信するためにエージェントがキューに格納できるコマンドの最大数を設定します。数値が大きいほど、エージェントが使用できるメモリー数が増え、これをゼロ(0)に設定するとキューのサイズは無制限になります。この値を 0 に設定すると、サーバーが長時間オフラインになると、マシンよりも多くのコマンドをキューに入れることができます。
rhq.agent.client.max-concurrent	エージェントが一度にサーバーに送信できるメッセージの最大数を設定します。数値が大きいほど、エージェントはより迅速にキューを処理できますが、これによりエージェントがより多くの CPU サイクルを使用するよう設定することもできます。
rhq.agent.client.command-timeout-msecs	エージェントがコマンドを中止するまで、JBoss ON ON サーバーからの応答を待つ時間を設定します。時間がかかる間隔により、一部のコマンドの完了に必要な時間を設定できますが、他のメッセージが処理を待機していることを示すこともできます。
rhq.agent.client.retry-interval-msecs	コマンドを再試行する前にエージェントが待機する時間を設定します。保証された配信タグを持つコマンドのみが再試行されます。

パラメーター	description
rhq.agent.client.send-throttling	エージェントがコマンドの送信を停止するまでにエージェントが送信できない数に制限を設定します。この設定は、スロットリング可能なコマンドのみに影響します。このコマンドは、頻繁にサーバーに送信されるコマンドで、メトリクスコレクションなどの多数の数値で送信されます。送信スロットリングは、メッセージがサーバーに送信されるのを防ぎます。 このパラメーターは、コマンドの **timeout_milliseconds 形式のコマンド数と quiet 期間の両方を** 設定します。たとえば、50 コマンドの制限と、未使用期間の 10000 ミリ秒を **50:10000** 設定します。
rhq.agent.client.queue-throttling	指定された時間内にデキューできるコマンドの量を制限します。これは **バースト期間** です。バースト期間に許可されているよりも多くのコマンドをデキューしようとすると、次のバーストが始まるまでデキュー要求はブロックされます。 送信スロットリングと同様に、このパラメーターはコマンドの **timeout_milliseconds** 形式で、コマンドの数と quiet 期間の両方を設定します。たとえば、50 コマンドの制限と、未使用期間の 10000 ミリ秒を **50:10000** 設定します。 キュースロットリングは、できるだけ迅速にコマンドを実行して、エージェントが CPU を起動しないようにします。キュースロットリングは、エージェントが必要とする CPU の量を減らす 1 つの方法です。 キューのスロットリング値を設定する場合は、追加のコマンドをキューにキューに追加する余分な値にキューサイズを設定してください。

バグを報告します。

7.18. コマンドの GUARANTEED DELIVERY の設定

エージェントとサーバー間の ping などのコマンドの多くは、JBoss ON の機能には重要ではありません。これらは **揮発性コマンドです**。揮発性コマンドが一度送信されます。失敗したコマンドが失敗すると、エージェントがログに記録され、次のコマンドが処理されます。

ただし、重要なコマンドは JBoss ON サーバーに送信され、正常に処理される必要があります。エージェントは、これらのコマンドが確実に配信されることを確認する必要があります。これらはコマンドを **保証し** ます。エージェントは、コマンドが可能な限りサーバーに到達するように保証します（JVM のクラッシュなどの外部イベントはコマンドを送信しないようにします）。保証された **コマンドは、エージェントがシャットダウンした場合でもコマンド spool ファイル** に残ります。これにより、次にエージェントが起動するたびにロードおよびキューに置かれ、サーバーに配信されます。

保証された配信に関連する 4 つのパラメーターがあります。

- 失敗したコマンド(***rhq.agent.client.retry-interval-msecs***)の再送信をエージェントが試行する頻度を設定する時間間隔
- spool ファイル(***rhq.agent.client.command-spool-file.name***)のファイル名
- spool ファイル(***rhq.agent.client.command-spool-file.params***)を設定する設定。この設定は、**max_file_size:purge_percentage** の形式になります。ファイルサイズはバイト単位で定義されます。ファイルがファイルサイズに達すると、パージ操作により、パーセンテージに関係なくファイルをトリミングします。したがって、ファイルが 100 KB(100000)に設定され、パージ率が 90 に設定されている場合、ファイルはパージ操作後に 90 KB に戻ります。パージ操作は最初に未使用領域を圧縮し、最も古いものからコマンドのパージを開始します。
- spool ファイルを圧縮できるオプションの設定(***rhq.agent.client.command-spool-file.compressed***)。spool ファイルを圧縮するとサイズが 30 から MTU に縮小できますが、場合によってはエージェントのパフォーマンスに悪影響を及ぼす可能性があります（保証されたコマンドがすべて送信される前にエージェントがシャットダウンした場合など）。

配信の保証はデフォルトで設定されるため、エージェントは重要なコマンドを再送信し、スプールファイルを圧縮できます。

```
rhq.agent.client.command-spool-file.compressed=true
rhq.agent.client.command-spool-file.name=command-spool.dat
rhq.agent.client.command-spool-file.params=10000000:75
rhq.agent.client.retry-interval-msecs=15000
```

保証された配信設定のいずれかを変更するには、以下を実行します。

1. エージェントプロンプトを開きます。たとえば、エージェントプロセスがすでに実行している場合は、**-n** オプションで **rhq-agent.sh** スクリプトを再度実行してプロンプトを開くことができます。

   ```
   agentRoot/rhq-agent/bin/rhq-agent.sh -n
   ```

2. 新しい保証された配信設定 **setconfig** でを送信します。

   ```
   > setconfig rhq.agent.client.command-spool-file.compressed=true
   rhq.agent.client.command-spool-file.name=my-spool.dat
   rhq.agent.client.command-spool-file.params=25000000:67
   rhq.agent.client.retry-interval-msecs=25000
   ```

3. エージェントプロセスを再起動して、新しい設定を読み込みます。

   ```
   agentRoot/rhq-agent/bin/rhq-agent-wrapper.sh stop

   agentRoot/rhq-agent/bin/rhq-agent.sh
   ```

バグを報告します。

7.19. エージェント通信の設定

JBoss ON エージェントとサーバーの両方で同じ基盤となる通信サービスが使用されます。エージェントサーバー通信に使用される接続の種類は、エージェント設定により定義され、これらの設定を変更して編集できます。エージェントは通信に 2 つの設定を使用します。

- エージェントがサーバー(***rhq.agent.server.transport***)および追加のトランスポートパラメーター（）との通信に使用するプロトコルを定義するパラメーター。***rhq.agent.server.transport-params***
- パラメーター。エージェントがサーバー(***rhq.communications.connector.transport***)からの着信通信を予想し、オプションのトランスポートパラメーター(***rhq.communications.connector.transport-params***)を定義します。

JBoss ON のサーバーおよびエージェントは、JBoss Remoting フレームワークで構築される通信層を使用します。エージェントでは、以下の 4 つの異なるトランスポートタイプがサポートされます。

- サーブレット（エージェントからサーバーとの通信のみ）
- sslservlet
- ソケット（サーバーからエージェント通信のみ）
- sslsocket

注記

JBoss ON サーバーとは異なり、JBoss ON エージェントはサーブレットコンテナーをホストしません。これは、サーバー間の通信にサーブレットを使用できないことを意味します。これらの接続はソケットを使用します。エージェント間の接続のみがサーブレットを使用します。

エージェントとサーバー間の接続の動作は、トランスポートパラメーターを設定して制御できます。エージェントとサーバー間の接続は、URL のように概観する文字列で定義されます。この基本形式は、以下の基本形式です。

```
protocol://hostname:port/?param1=value&param2=value
```

例：

```
socket://server.example.com:16163/?
serverBindAddress=127.0.0.1&serverBindPort=16163&numAcceptThreads=3&maxPoolSize=303&client
MaxPoolSize=304&socketTimeout=60000&enableTcpNoDelay=true&backlog=200
```

サーバーおよびエージェントには、トランスポートパラメーターを設定できる ***rhq.communications.connector.transport-params*** 構成設定があります。これらのパラメーターは URL の最後に追加され、サーバー側およびクライアント側の動作の両方を設定できます。たとえば、**backlog** パラメーターは JBoss ON サーバーによって使用されます。この URL では、サーバーはバックログの値を 200 に設定しますが、この設定はクライアントであるためエージェントによって無視されます。同様に、**enableTcpNoDelay** パラメーターはサーバーに接続するときにエージェントによって使用されますが、サーバー自体は無視されます。

利用可能なすべてのトランスポートパラメーターの詳細は、で JBoss Remoting のドキュメントを参照してください http://docs.jboss.org/jbossremoting/2.5.4.SP4/guide/html/chapter-configuration.html。

バグを報告します。

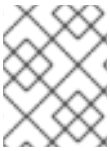
第8章 JBOSS ON に関連付けられたデータベースの管理

JBoss ON サーバーで使用される Oracle または PostgreSQL データベースを管理するために、基本的なタスクが複数あります。

バグを報告します。

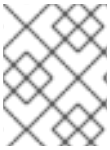
8.1. JBOSS ON からの SQL コマンドの実行

SQL コマンドは、JBoss ON サーバーがデータに使用する任意のデータベースの JBoss ON Web UI で実行できます。



注記

データベース管理ページは、通常の JBoss ON GUI からアクセスできません。管理者は、JBoss ON UI の admin エリアを手動で移動する必要があります。



注記

ログインしている JBoss ON ユーザーでも、SQL コマンドの実行には **データベースに対する** 適切なユーザー権限が必要です。

1. 場所を指定して管理ページを開き **coregui/#Test/ServerAccess/SQL**ます。例：

   ```
   http://server.example.com:7080/coregui/#Test/ServerAccess/SQL
   ```

2. JBoss ON Oracle または PostgreSQL データベースインスタンスに適した SQL コマンドを入力します。複数のコマンドがある場合は、**Continue if statements fail?** チェックボックスが選択されていることを確認します。このようにして、いずれかのコマンドが失敗しても、他のコマンドが送信されます。それ以外の場合は、最初の失敗時にシリーズが終了します。

3. **Execute SQL** ボタンをクリックします。

バグを報告します。

8.2. データベースパスワードの変更

JBoss ON サーバーは、データベースインスタンスにデータベースユーザーとして接続します。これは、**rhq-server.properties** ファイルで指定される認証情報を使用して自動的に行われます。データベースのパスワードは、**rhq-server.properties** ファイルに保存される前にインストーラーによって自動的にエンコードされるため、データベースのパスワードへの不正アクセスに対して追加の保護が提供されます。

そのデータベースユーザーアカウントのパスワードが変更された可能性があります。この変更は JBoss ON ではなくデータベースで常に実行されるため、JBoss ON が引き続き機能するには、**rhq-server.properties** ファイルのパスワードを手動でエンコードし、更新する必要があります。

1. データベース内の JBoss ON ユーザー（デフォルト**rhqadmin** では）のパスワードを変更します。

2. **rhq-encode-password.sh** スクリプトを使用してパスワードをエンコードします。

```
serverRoot/bin/rhq-encode-password.sh mypassword
Encoded password: 1d31b70b3650168f79edee9e04977e34
```

JBoss ON は、データベースのパスワードを **rhq-server.properties** ファイルにエンコードされた形式で格納します。そのため、新しいデータベースを **rhq-server.properties** ファイルに追加する前に適切にエンコードし、サーバーが正しく読み取られるようにする必要があります。

3. **rhq-server.properties** ファイル内の ***rhq.server.database.password*** 値を編集して、新しいエンコードされたパスワード値になるようにします。

```
vim serverRoot/bin/rhq-server.properties

rhq.server.database.password=1d31b70b3650168f79edee9e04977e34
```

バグを報告します。

8.3. JBOSS ON SERVER のデータベース設定の編集

JBoss ON サーバーは、エージェントやリソースなどのほとんどのデータをインベントリーおよびプラグイン設定に保存するために常にバックエンドデータベースに接続されます。データベースに接続するためのパラメーターはで定義されてい **rhq-server.properties**ます。

例8.1 PostgreSQL データベースのデフォルト設定

```
# Database
rhq.server.database.connection-url=jdbc:postgresql://127.0.0.1:5432/rhq
rhq.server.database.driver-class=org.postgresql.Driver
rhq.server.database.xa-datasource-class=org.postgresql.xa.PGXADataSource
rhq.server.database.user-name=rhqadmin
rhq.server.database.password=1eeb2f255e832171df8592078de921bc
rhq.server.database.type-mapping=PostgreSQL
rhq.server.database.server-name=127.0.0.1
rhq.server.database.port=5432
rhq.server.database.db-name=rhq
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

表8.1 rhq-server.properties パラメーターによるデータベース設定

パラメーター	description
rhq.server.database.type-mapping	JBoss ON サーバーで使用されるデータベースのタイプまたはベンダーを指定します。PostgreSQL または Oracle10g（Oracle10g は Oracle データベースのバージョン 10、11、および 12 のいずれかに使用されます）。
rhq.server.database.connection-url	JBoss ON サーバーがデータベースに接続するときに使用する JDBC URL（例：**jdbc:postgresql://localhost:5432/rhq**、**jdbc:oracle:oci:@localhost:1521:orcl** など）。

パラメーター	description
rhq.server.database.driver-class	JBoss ON サーバーがデータベースとの通信に使用する JDBC ドライバーの完全修飾クラス名（例：**oracle.jdbc.driver.OracleDriver** ）。
rhq.server.database.xa-datasource-class	JBoss ON サーバーがデータベースとの通信に使用する JDBC ドライバーの完全修飾クラス名（例：**org.postgresql.xa.PGXADataSource または oracle.jdbc.xa.client.OracleXADatasource**）。
rhq.server.database.user-name	データベースにログインする際に JBoss ON サーバーが使用するユーザーの名前。
rhq.server.database.password	データベースにログインする際に JBoss ON サーバーによって使用されるデータベースユーザーのパスワード。このパスワードは、**rhq-server.properties** ファイルのハッシュに保存されます。データベースにパスワードを変更すると、手動でパスワードをハッシュして **rhq-server.properties** ファイルにコピーする必要があります。これは、で説明してい「データベースパスワードの変更」ます。
rhq.server.database.server-name	データベースが置かれているサーバー名。これは、接続 URL のサーバーと一致する必要があります。現在、これは PostgreSQL に接続するときにのみ使用されます。
rhq.server.database.port	データベースがリッスンしているポート。これは、接続 URL のポートと一致する必要があります。現在、これは PostgreSQL に接続するときにのみ使用されます。
rhq.server.database.db-name	データベースの名前。これは、接続 URL にある名前に一致する必要があります。現在、これは PostgreSQL に接続するときにのみ使用されます。
rhq.server.quartz.driverDelegateClass	サーバーとデータベース間の接続に使用される Quartz ドライバー。この値はインストーラーによって設定され、JBoss ON 情報を保存するために使用されるデータベースの種類によって異なります。PostgreSQL の場合、これはです。**org.quartz.impl.jdbcjobstore.PostgreSQLDelegate**Oracle の場合、これは **org.quartz.impl.jdbcjobstore.oracle.OracleDelegate**.

バグを報告します。

第9章 ストレージノードのデプロイおよび管理

未加工のメトリックデータと集約されたメトリックデータは、専用のスケーラブルな分散データベースに保存されます。JBoss ON サーバーと同様に、クラスター内には複数のストレージノードが存在することがあります（サーバーとは別にインストール）。クラウドからノードを簡単に追加およびドロップできるため、環境のニーズに応じてメトリクスストレージを動的に拡張できます。

バグを報告します。

9.1. HIGH-SPEED METRICS STORAGE

メトリクスの収集およびアラートの生成は、リソース集約型と言えます。これにより、バックエンドデータベース上でのほぼコンスタント書き込み操作が可能になります。これにより、パフォーマンスの低下に遭遇する前に収集できるメトリクスの数の許容しきい値(minute per minute)を作成します。

注記

毎分の メトリクスの許容しきい値は、内部のパフォーマンステストに基づいています。このしきい値は、システムリソースと全体的な負荷によって異なります。

JBoss ON は 2 つのデータベースを使用して情報を保存します。1 つは、JBoss ON サーバーおよびエージェント、すべてのリソースインベントリーデータ、リソース設定、およびその他のデータに関するすべての設定を格納する中央リレーショナルデータベース（PostgreSQL または Oracle）です。他のデータベースは、すべての数値監視データを保存する分散データベース（ストレージノードのクラスター）です。つまり、収集したすべてのメトリクスです。

分散データベースは、クラスター内の複数のノードを使用して拡張できます。負荷に応じてノードを追加することは、管理者にとって重要な管理ツールです。1 分あたりに収集される │ メトリックのハードウェア駆動な制限に遭遇する代わりに、パフォーマンスを向上させるために追加のノードを追加できます。

JBoss ON によって少なくとも 1 つのストレージノードを作成して管理する必要があります（これにより、外部ツールが必要ないため、メトリクスストレージノードの管理オーバーヘッドが最小限に抑えられます）。

重要

JBoss ON インストールでの負荷や需要に関係なく、JBoss ON サーバーはメンテナンスモードに移行します。

- ストレージノードは、ストレージ要求のバックログを処理します（ピーク時）。
- メモリーからディスクにデータの転送
- データの集約または圧縮を実行します。

ほとんどのインスタンスでは、システムは中断せずに復旧するように設計されています。ただし、システムの需要や負荷が増大すると、システム停止の長さも増えます。これらの停止の長さが割り込み操作が開始されたら、負荷を配布するために追加のストレージノードをインストールする必要があります。ストレージノードにストレージ容量が十分ではない場合にも、追加のストレージノードが必要です。追加のストレージノードのインストールに関する詳細は、を参照してください 「追加ノードのインストール」。

メトリクスデータを処理する通信のパスは複数あり、すべて並行して動作します。

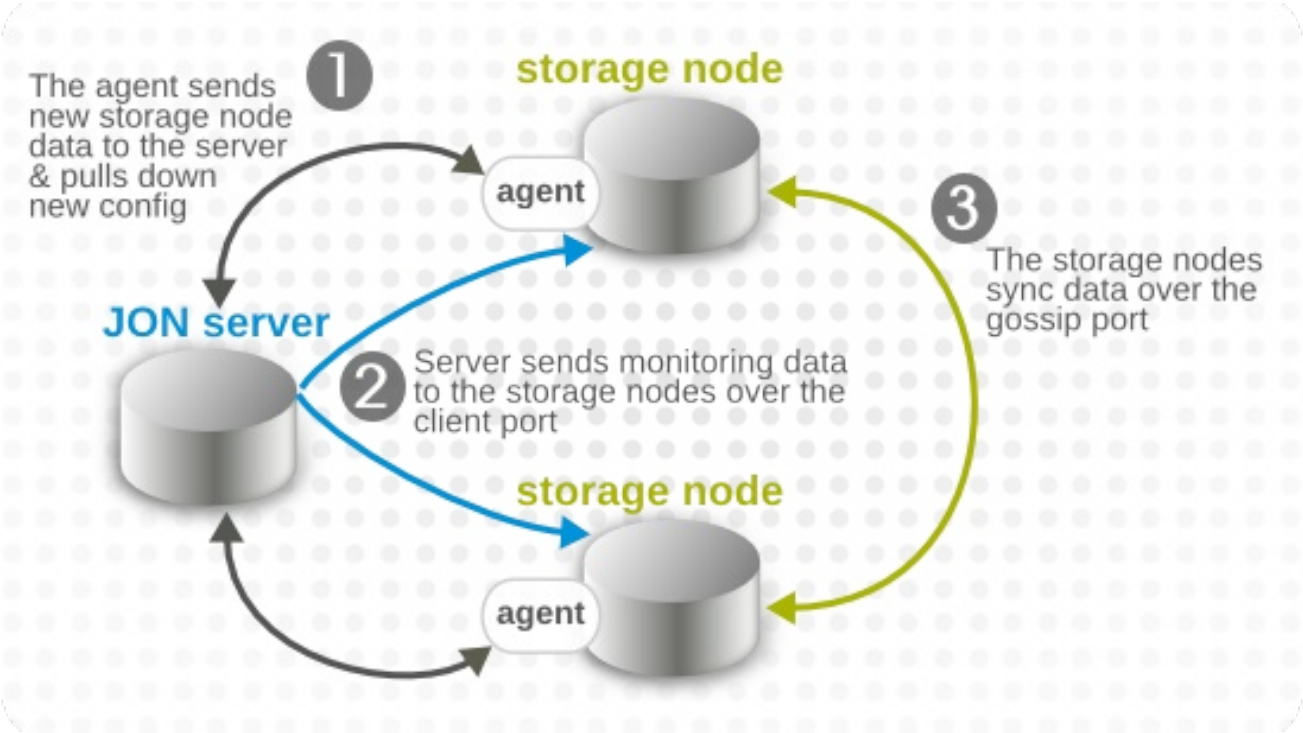
1. エージェントはストレージノード設定を JBoss ON サーバーに送信します。その後、JBoss ON サーバーは更新されたストレージクラスター情報をストレージノードに関連付けられたすべてのエージェントに送信します。

   その後、コンパニオンエージェントは、新しいノードの **rhq-storage-auth.conf**ホスト名または IP アドレスでストレージクラスター設定を更新します。（同様に、ノードが削除されると、サーバーは各エージェントに情報を送信し、エージェントはローカルストレージノードの **rhq-storage-auth.conf** ファイルのリストからホスト名または IP アドレスを削除します）。

2. サーバーは、（ストレージノードに関連するものだけでなく）すべてのエージェントから監視データを受信し、その情報を利用可能なストレージノードに送信して保存します。

3. ストレージノードは、高可用性と整合性を確保するために、相互に監視データを複製します。

図9.1 サーバー、エージェント、およびメトリクスストレージノードの通信



ノードのクラスター通信には 3 つの要素が必要です。

- に保管されているすべてのストレージノードのホスト名または IP アドレス **rhq-storage-auth.conf**
- JBoss ON サーバーがストレージノード（クライアントポート）との通信に使用する一般的な**ポート**番号。
- 相互にデータの同期に使用するクラスター内の他のストレージノード用の共通のポート番号（**ゴシップポート**）

メトリクスストレージは、以下の 2 つの方法でデータのバックアップを作成することで、データの可用性と整合性を提供します。

- ストレージノード間のデータの複製（ゴシップポート上）
- ローカルスナップショットの取得およびローカルデータのバックアップ

バグを報告します。

9.2. ストレージノードのデプロイおよびアンデプロイ

JBoss ON 環境では、複数のメトリクスストレージノードが存在する可能性があります。JBoss ON サーバーと同様に、高可用性クラウドで動作するのと同様に、複数のストレージノードが同じ環境にデプロイされ、クラスターで機能するように相互に通信します。

ノードを管理者がクラスターに追加および削除したり、環境の変更に対応して JBoss ON スクリプトを動的に使用したりできます。

注記

コンパニオンエージェントは、サーバーノード通信を有効にするために、ストレージノードと共に常にインストールされます。

デフォルトのクラスター設定では、ノードが JBoss ON サーバーにインストールおよび設定されるとすぐにデプロイされます。これらは、実際には 2 つの手順です。

1. ビットはローカルシステムにインストールされ、ストレージノードが JBoss ON サーバーに登録されます。
2. 新しいノード情報はクラスターにデプロイされます。

環境のプロビジョニングおよびモニタリングの要件により、ノードを自動にデプロイするか、または手動で判断できるかどうか。

警告

ノード一覧をクラスター設定にデプロイし、許可 さ れるホストはストレージクラスター内のデータにアクセスできます。

許可されたホスト一覧を変更できないように、**rhq-storage-auth.conf** ファイルへのアクセスを制限して、攻撃者がクラスターおよび保存されたデータにアクセスできるようにします。

バグを報告します。

9.2.1. ストレージノードの要件

ノードをデプロイするには、2 つの重要な要件があります。

- すべてのストレージノードのホスト名または IP アドレス、JBoss ON サーバーおよびエージェントのホスト名およびバインドアドレスはすべて DNS で完全に解決できる必要があります。ストレージノード、サーバー、またはエージェントの IP アドレスおよびホスト名が DNS で適切にフォーマットされていない場合は、異なる JBoss ON コンポーネント間の通信はすべて失敗します。

- ファイアウォールは、ストレージノードで使用される 2 つのポートでの通信を許可する必要があります。デフォルトでは、ポートはサーバー/クライアントの 9142 と 7100、ゴシップポートの場合はそれぞれ 9142 と 7100 です。

バグを報告します。

9.2.2. サーバーを使用したストレージノードのインストール

デフォルトでは、サーバーインストールスクリプトはストレージノードとエージェントもインストールします。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --start
```

追加オプションは必要ありません。

バグを報告します。

9.2.3. サーバーのインストール前のストレージノードのインストール

サーバーをインストールする前に複数のストレージノードを作成してから、事前にインストールしたノードでサーバーをインストールできます。これは、ストレージデータベースが別個の専用のマシンにある場合にも便利です。

警告

これは高度な設定です。クラスター内のストレージノードまたはノードが適切に設定されていない場合は、クラスターが適切に機能しない可能性があります。

警告

ノード一覧をクラスター設定にデプロイし、許可 さ れるホストはストレージクラスター内のデータにアクセスできます。

許可されたホスト一覧を変更できないように、**rhq-storage-auth.conf** ファイルへのアクセスを制限して、攻撃者がクラスターおよび保存されたデータにアクセスできるようにします。

重要

すべてのストレージノードは、同じクライアント(CQL)とゴシップポートを使用する必要があります。

さらに、全ストレージノードシステムのホスト名および IP アドレスは、DNS で完全に解決できるか、各システムの **hosts** ファイルで設定する必要があります。

1. 使用するノードおよびクラスター設定情報を判別します。
   - ノードをホストする各システムのホスト名または IP アドレスを特定します。
   - クラスターが通信に使用する 2 つのポートを定義します（デフォルトでは、9142 および 7100）。
2. **ストレージノードをインストールする前に、ノード** とクラスター情報をすべて指定してストレージプロパティーファイルを編集します。

   ```
   [root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-storage.properties
   ```

   たとえば、この ***rhq.storage.seeds*** パラメーターで設定される 3 つのノードを設定します。

   ```
   rhq.storage.cql-port=9142
   rhq.storage.gossip-port=7100
   rhq.storage.seeds=192.68.0.0, 192.68.0.1, 192.68.0.2
   start=false
   ```

3. コンパニオンエージェントを使用して、各システムにストレージノードをインストールします。サーバーが **インストールされていない場合でも、JBoss ON サーバーの** IP アドレスが必要です。

   この時点で、ストレージノードやエージェントは起動しないでください。インストールスクリプトに --start オプションを使用しないでください。

   ```
   [root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --storage --agent-
   preference="rhq.agent.server.bind-address=192.68.0.2"
   ```

4. 各ストレージノードについて、ローカル **rhq-storage-auth.conf** ファイルを編集します。これにより、クラスター内のすべてのストレージノードのホスト名または IP アドレスが 1 行ごとに一覧表示されます。

   ```
   [root@server ~]# vim serverRoot/jon-server-3.3.0.GA/rhq-storage/conf/rhq-storage-auth.conf

   192.68.0.0
   192.68.0.1
   192.68.0.2
   ```

   サーバーの設定後、ローカルエージェントはノードのホスト名または IP アドレスで **rhq-storage-auth.conf** ファイルを更新し、クラスターからデプロイされ、削除されます。

5. 各ノードを起動します。

   ```
   [root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --storage
   ```

6. サーバーをインストールする前に、**rhq-server.properties** ファイルを編集してストレージノードの接続情報を追加します。

   rhq.storage.nodes パラメーターに一覧表示されるカンマ区切りの各ストレージノードを追加します。次に、クライアントとゴシップポートの値を追加します。

   ```
   [root@server ~]# vim serverRoot/jon-server-3.3.0.GA/bin/rhq-server.properties
   ```

```
rhq.storage.nodes=192.68.0.0,192.68.0.1,192.68.0.2
rhq.storage.cql-port=9142
rhq.storage.gossip-port=7100
```

7. サーバーとエージェントをインストールします。**--server** および **--agent** オプションを指定すると、これら 2 つのコンポーネントのみがインストールされます。ストレージデータベースは除外されます。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --server --agent --start
```

既存の JBoss ON エージェントをアップグレードする場合は、**--use-remote-storage-note** オプションを指定してアップグレードスクリプトを実行し、ストレージノードをインストールするのではなくプロパティーファイルからストレージデータベース情報をロードします。

```
[root@server]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl upgrade --use-remote-storage-node=true
```

バグを報告します。

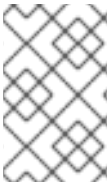
9.2.4. 追加ノードのインストール

新しいストレージノードを作成する際には、ストレージノードとコンパニオンエージェントの 2 つのコンポーネントが常にインストールされます。ノードをセットアップするのに必要な唯一の情報は JBoss ON サーバーの IP アドレスです。エージェントはサーバーに登録してから、ストレージノードのホスト名または IP アドレスを送信します。その後、サーバーは他のエージェント間でその情報を配布し、クラスターに伝播されます。

クラスターがカスタムクライアントおよびゴシップポートを使用している場合は、インストールスクリプトを実行する前に正しいポートで **rhq-storage.properties** ファイルを編集します。

--agent-preference オプションを指定してインストールスクリプトを実行し、サーバーバインドアドレスを指定します。例：

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl install --storage --agent-preference="rhq.agent.server.bind-address=0.0.0.0"
```

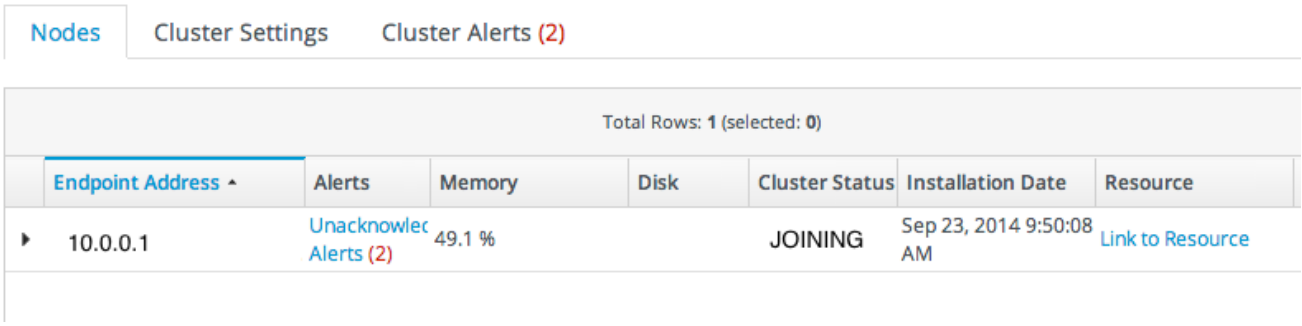


注記

Linux にインストールする場合は、root で **rhqctl** コマンドを実行する必要があります。Windows では、オプションを使用してコマンドプロンプトを開く必要があり **Run as Administrator**ます。

新規ノード情報は既存ノード間で伝播されるため、デプロイメント操作が完了するまで数分かかる場合があります。deploy 操作が完了するまで、ノードには **Joining** のステータスが表示されます。

図9.2 クラスターの参加



警告

ノード一覧をクラスター設定にデプロイし、許可 さ れるホストはストレージクラスター内のデータにアクセスできます。

許可されたホスト一覧を変更できないように、**rhq-storage-auth.conf** ファイルへのアクセスを制限して、攻撃者がクラスターおよび保存されたデータにアクセスできるようにします。

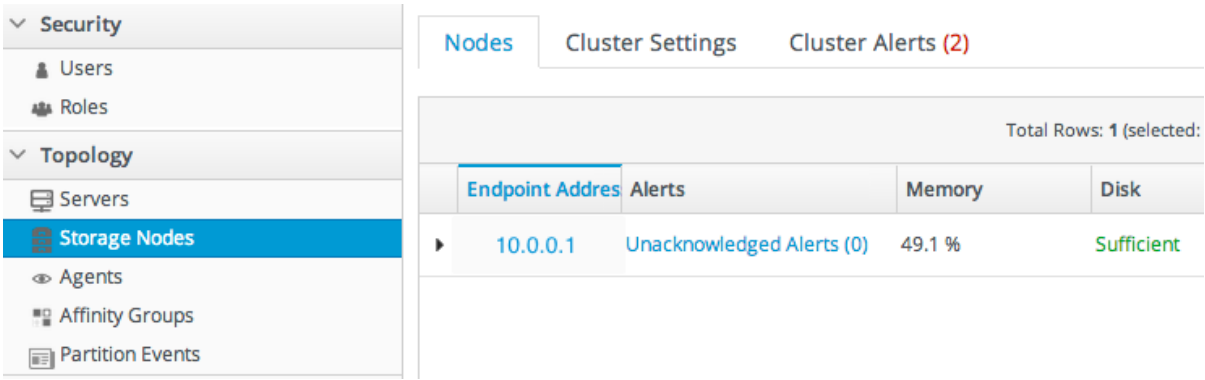
バグを報告します。

9.2.5. ノードの自動デプロイメントの設定

適切なポートおよびクラスター設定でノードを作成すると、即座にクラスターに参加するか、基本的にスタンバイ状態で待ってから、後で手動でデプロイできます。これは、クラスター全体の設定で決定されます。

これは、マシンがプロビジョニングされ、（仮想環境など）繰り返しオフラインにする場合や、ノードが特定の期間中のみオンラインになる必要がある場合に便利です。

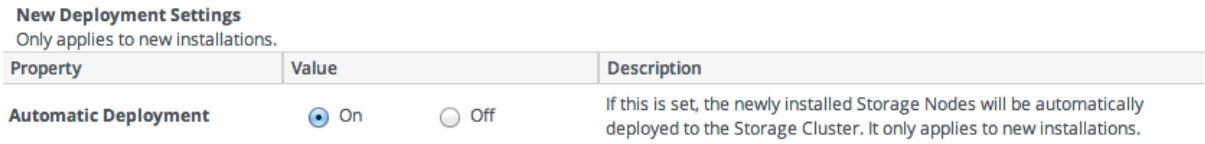
1. トップナビゲーションバーの **Administration** タブをクリックします。
2. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。



3. **Cluster Settings** タブを開きます。

4. **Automatic Deployment** オプションは、ストレージノードを JBoss ON にインストールされるとすぐにクラスターにデプロイするか、または手動でデプロイするのを待機するかどうかを設定します。

   デフォルトでは、ノードを自動的にデプロイします。これは、環境に他のプロビジョニングルールがある場合に変更できます。



5. ページの **Save** 下部をクリックします。

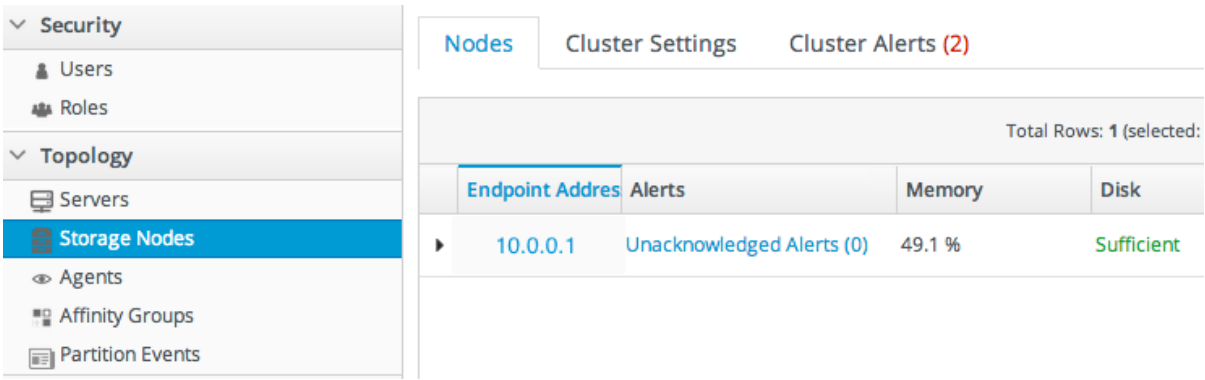
バグを報告します。

9.2.6. ノードの手動によるデプロイ

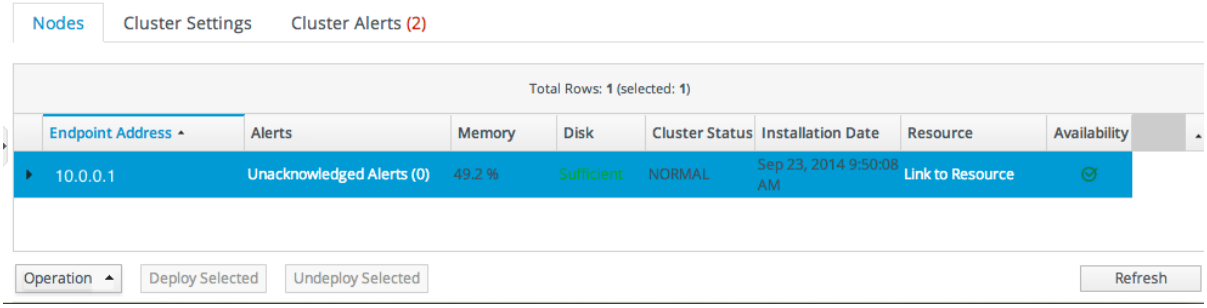
デフォルトでは、新しいストレージノードがインストールされると、自動的にクラスターにデプロイされます。ただし、自動デプロイメントが無効の場合は、新規ノードが作成後に手動でデプロイされる必要があります。

> **警告**
>
> ノード一覧をクラスター設定にデプロイし、許可 さ れるホストはストレージクラスター内のデータにアクセスできます。
>
> 許可されたホスト一覧を変更できないように、**rhq-storage-auth.conf** ファイルへのアクセスを制限して、攻撃者がクラスターおよび保存されたデータにアクセスできるようにします。

1. **rhqctl install** コマンドを使用してノードをインストールします。
2. トップナビゲーションバーの **Administration** タブをクリックします。
3. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。



4. **Nodes** タブで、デプロイするノードの行を選択します。



5. **Deploy** ボタンをクリックします。

ストレージノードリソースで **Deploy** 操作を実行して、ノードをデプロイすることもできます。

注記

ストレージノードは、JBoss ON CLI とスクリプトを使用してデプロイできます。これは、インフラストラクチャーに新しいシステムをプロビジョニングする場合に便利です。

例：

```
// deploy a storage node
nodes = StorageNodeManager.findStorageNodesByCriteria(StorageNodeCriteria());
node = nodes.get(0); StorageNodeManager.deployStorageNode(node);
```

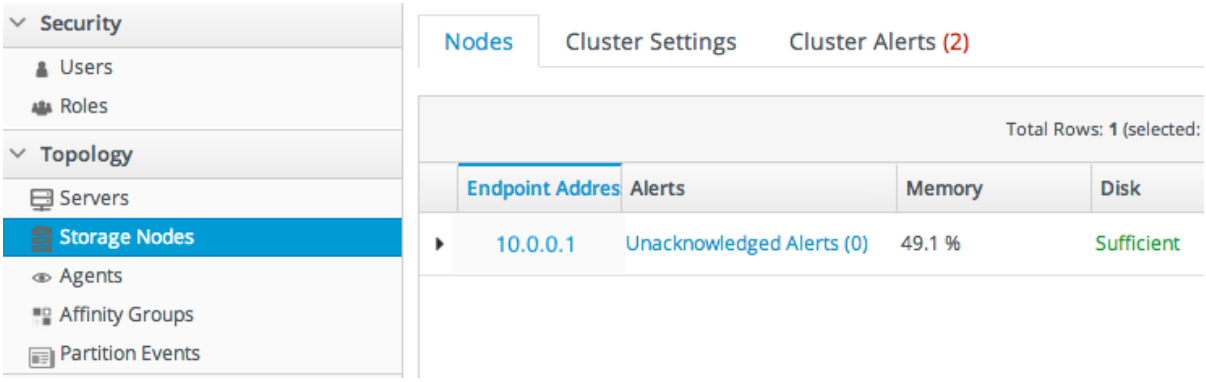
これは、インフラストラクチャーの要求に応じてストレージノードを動的に追加する際にキーとなることができます。ノードを事前にインストールしてから、必要に応じてホットデプロイおよび削除することができます。

バグを報告します。

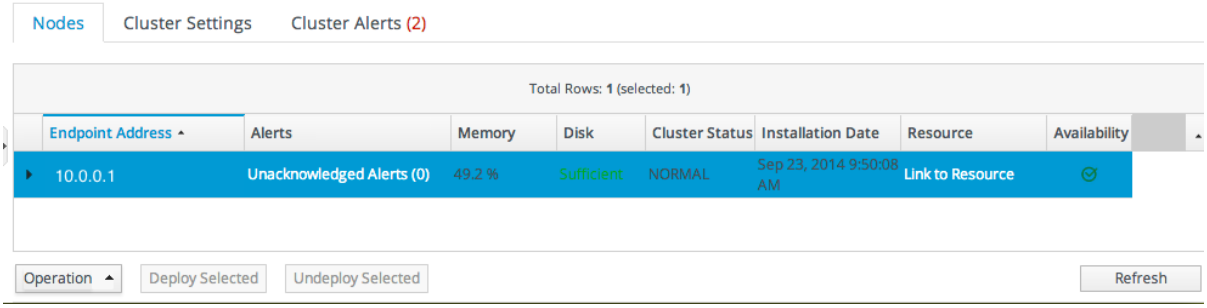
9.2.7. ノードの削除

ストレージノードをアンデプロイすると、クラスターからリソースが削除され、インベントリーからリソースが削除され、ストレージノードのビットがマシンからアンインストールされます。

1. トップナビゲーションバーの **Administration** タブをクリックします。
2. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。

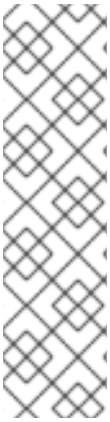


3. **Nodes** タブで、削除するノードの行を選択します。複数の行を選択するには、**Ctrl** キーを保持し、希望の行をクリックします。



4. **Undeploy Selected** ボタンをクリックして操作を確認します。

また、ストレージノードリソースで **Undeploy** 操作を実行してノードを削除することもできます。



注記

ストレージノードは、JBoss ON CLI およびスクリプトを使用して削除することもできます。これは、インフラストラクチャーからシステムのプロビジョニングおよび削除を行う場合に便利です。

例：

```
// undeploy a storage node
nodes = StorageNodeManager.findStorageNodesByCriteria(StorageNodeCriteria());
node = nodes.get(0); StorageNodeManager.undeployStorageNode(node);
```

バグを報告します。

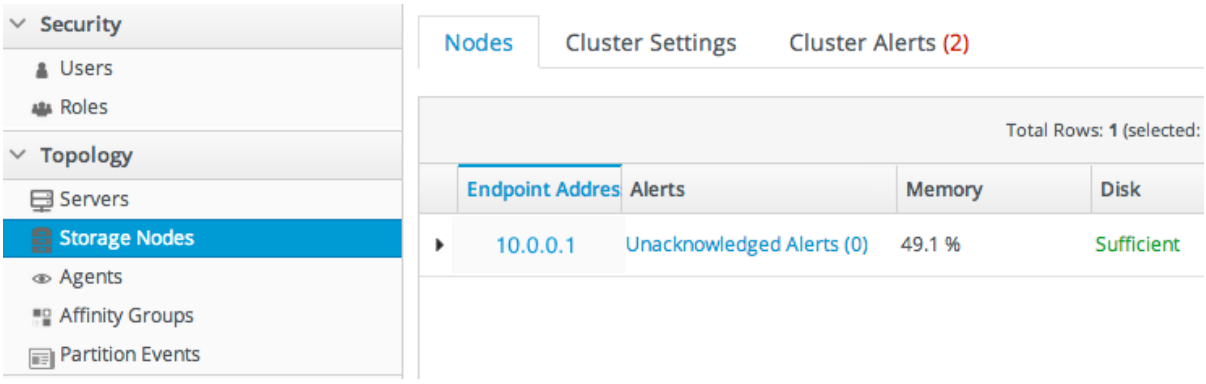
9.3. ストレージノードのメトリクスおよび状態の表示

ストレージノードには、メモリー、キャッシュ操作、データ操作、ロギング、クライアント要求に関連するサービスに対する子リソースが多数あります。ノードのパフォーマンスに不可欠なメトリックは、複数の子リソース（特に JVM メモリーサービスとデータベース管理ストレージサービス）に分散されます。

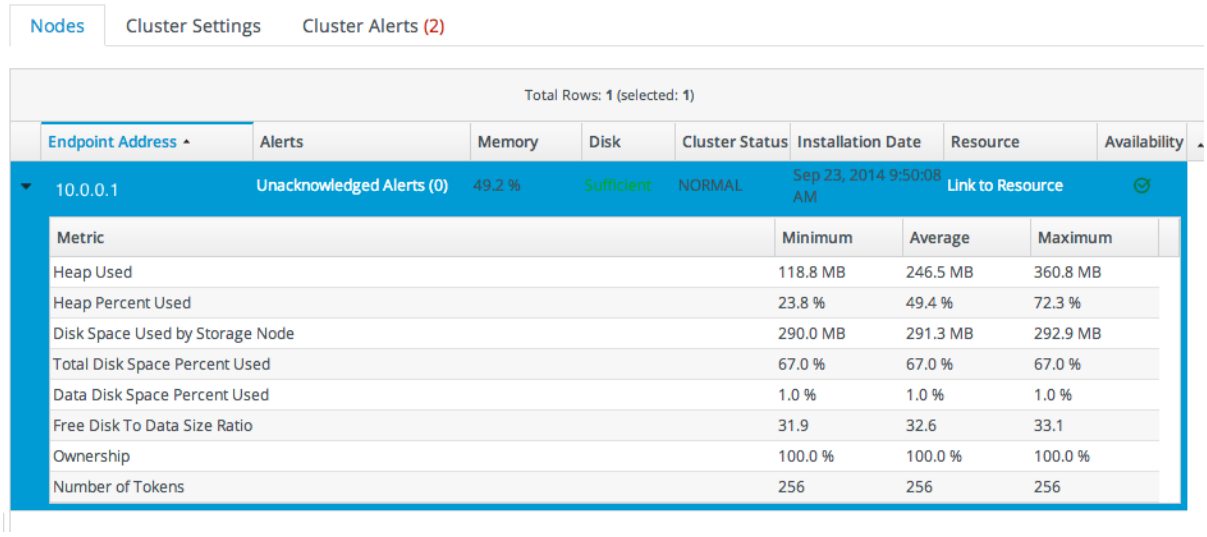
この **Nodes** タブは、すべてのノードサービスで収集される最も重要なメトリクスの概要を表示するため、ノードのパフォーマンスに優れたスナップショットが提供されます。また、**クラスター内のノードの追加情報も提供し** ます。これにより、サービスメトリクスには表示されないノードの通信の問題を示すことができます。

ノードのメトリクスおよび状態を表示するには、以下を実行します。

1. トップナビゲーションバーの **Administration** タブをクリックします。
2. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。

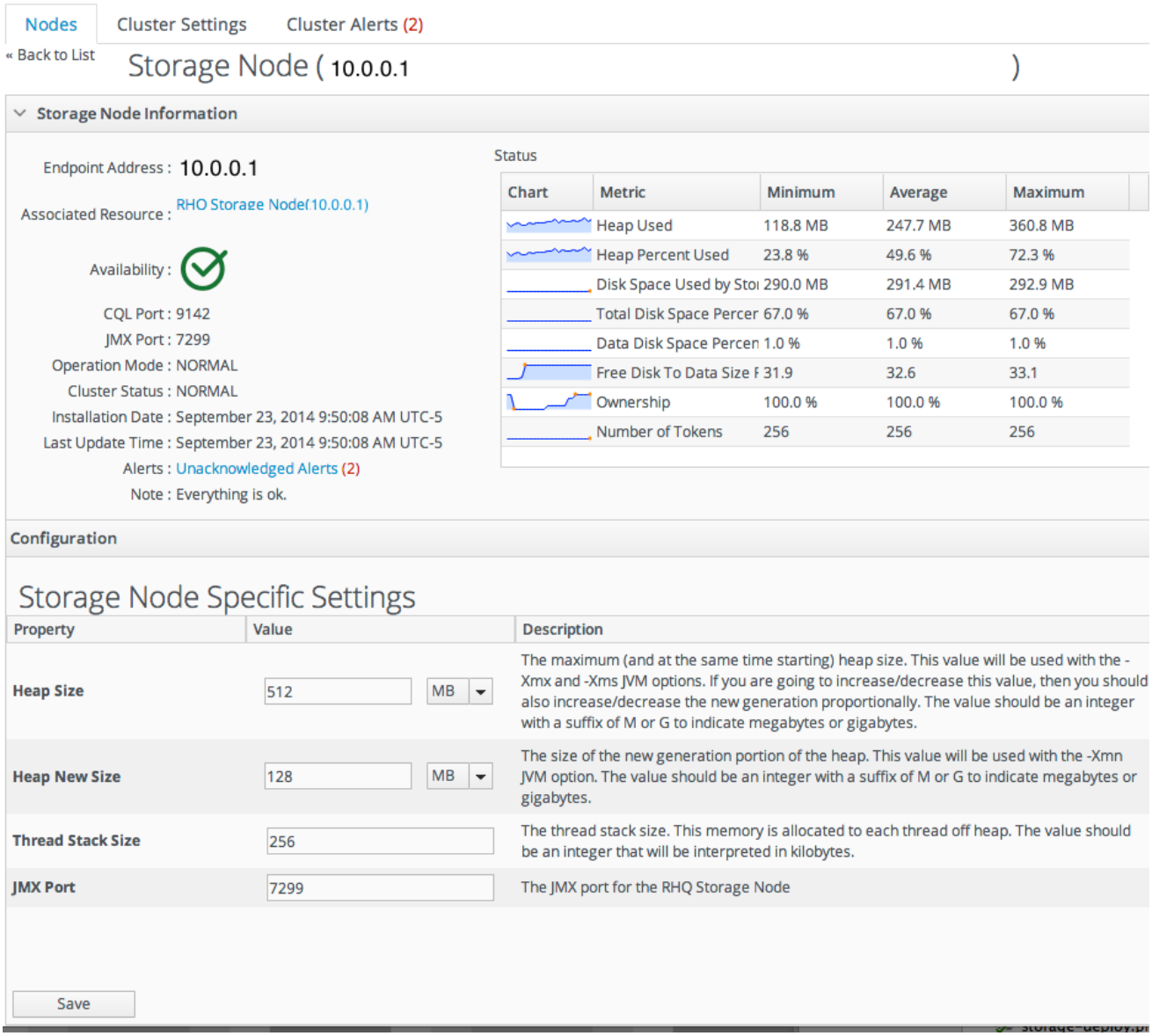


3. **Nodes** タブには、各ノードの承認されていないアラートの数が表示されます。



ノードのホスト名または IP アドレスをクリックすると、すべてのメトリクスおよび状態データ、現在の設定、およびノードがリッスンするポートが含まれる詳細ページが開きます。

図9.3 ストレージノードの詳細



バグを報告します。

9.4. クラスターポートの変更

メトリクスストレージノードクラスターは、クラスターノードを特定して通信できるように共有情報に依存します。JBoss ON サーバーは設定されたポートを使用してストレージノードと通信し、agent-storage 設定を管理します。

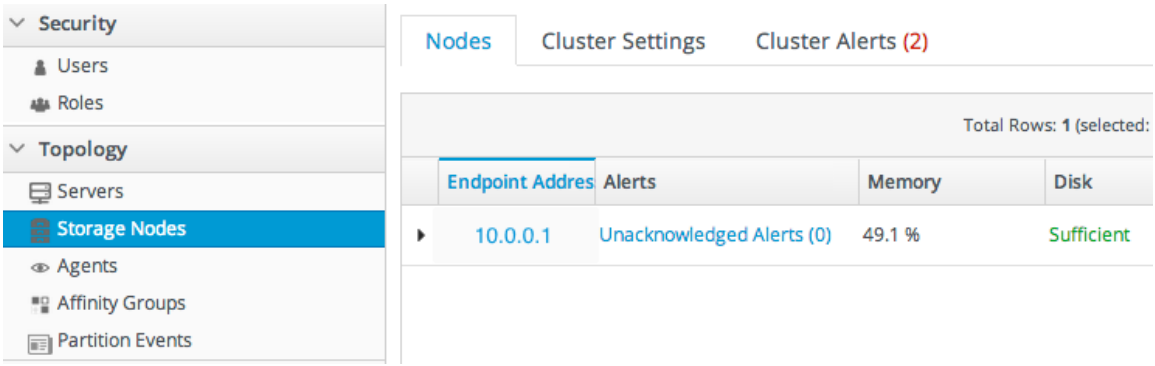
クラスターの設定では、すべてのメンバーが同じポート（クライアントポート）上のサーバーと通信し、同じ **ポート（ゴシップポート）**で相互に通信している必要があります。

ポート設定は、JBoss ON の設定（サーバーおよびエージェントによって使用される）と個別のノードで 2 つの場所で変更する必要があります。

1. サーバーで使用されるポート設定を変更します。

   a. トップナビゲーションバーの **Administration** タブをクリックします。

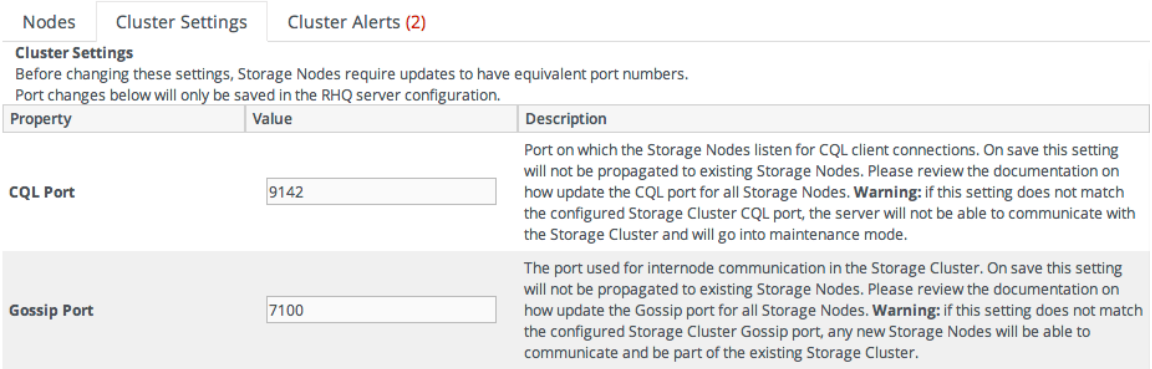
   b. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。



c. **Cluster Settings** タブを開きます。

d. ポート番号を編集します。

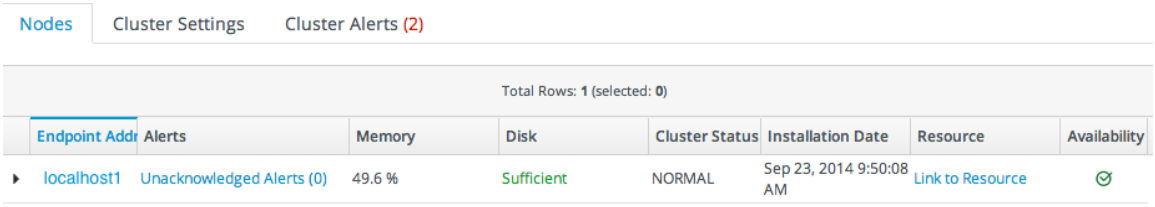
- JBoss ON サーバーがすべてのストレージノードと通信し、メトリクスデータを送信するために使用するクライアントポートを **CQL Port** 設定します。
- は、ノードが相互に接続し、メトリクスデータを複製するのに使用するポートを **Gossip Port** 設定します。



e. ページの **Save** 下部をクリックします。

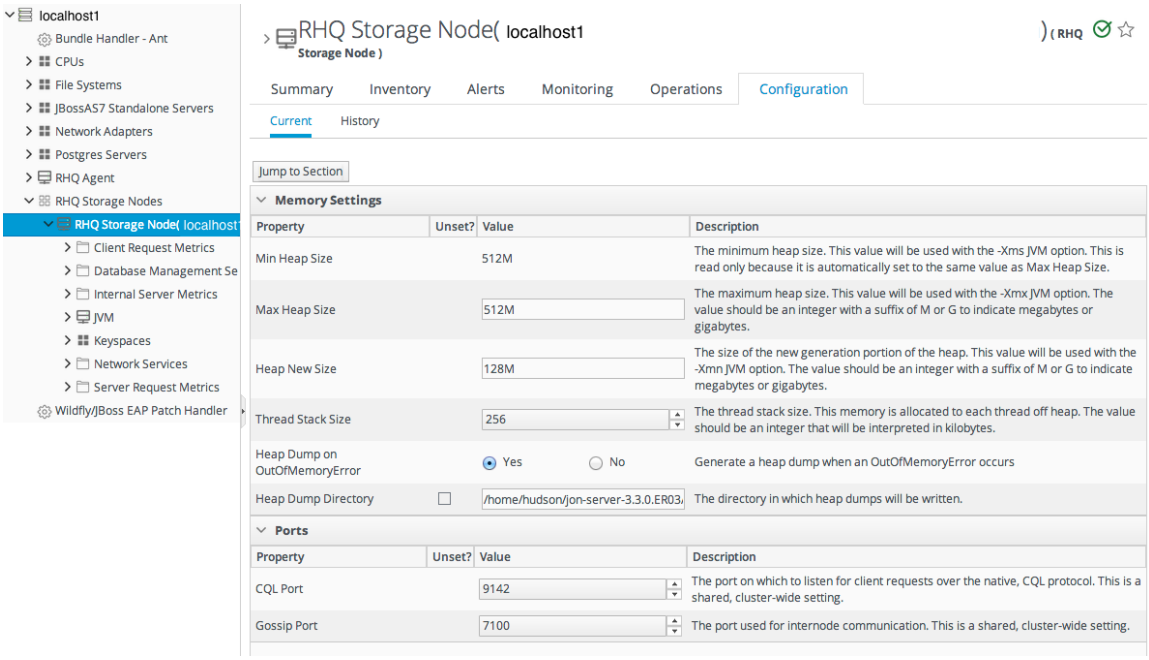
2. 各ノードのポート設定を変更します。

a. ストレージ管理エリアの **Nodes** タブで、をクリックし **Link to Resource**ます。これにより、ノードのリソースページが開きます。



b. ストレージノードの **Configuration** タブを開きます。

c. ポート番号を編集します。



- JBoss ON サーバーがすべてのストレージノードと通信し、メトリクスデータを送信するために使用するクライアントポートを **CQL Port** 設定します。
- は、ノードが相互に接続し、メトリクスデータを複製するのに使用するポートを **Gossip Port** 設定します。

d. ページの **Save** 下部をクリックします。

3. 各ノードを再起動します。
4. クライアント(CQL)ポートが変更された場合は、各 JBoss ON サーバーも再起動します。

バグを報告します。

9.5. ストレージノードアラートの表示

ストレージノードは、メトリクス情報を保存する弾力的な方法を提供します。これにより、より多くのメトリクスをより頻繁に収集できるようになります。特定のインフラストラクチャー、リソース数、およびメトリクス数に適切なストレージノード数を維持するには、追加のストレージノードが必要であるかどうかを示すために、現在のノードの実行方法を常に認識する必要があります。

追加のメトリクスストレージノードをデプロイするか、またはメトリクスコレクションスケジュールを調整する必要があることを示します。

- JVM ヒープは最大しきい値に達し、パフォーマンスが低下します。
- ストレージノードは、システムで多くのディスク容量の使用を開始します。

ヒープサイズ（他の JVM パラメーターと併せて）はノードごとに設定でき、ディスク領域の上限はシステムに対して相対的ですが、大規模な環境や集中的なメトリクスコレクションでは、ストレージノードはハードウェアの上限を生じさせる可能性があります。

ストレージノードリソースには、他のリソースと同様に直接メトリクスまたはアラート定義が設定されていません。代わりに、すべてのノードに事前定義された 4 つのアラートが事前に定義され、ノードのスロットの自動モニタリングを追加するタイミングを決定するのに役立ちます。

- ヒープ使用率が高いため、メモリーエラーやパフォーマンスの低下につながる可能性があります。集中的なルールが、一時的なメモリーの急増を防ぐために配置されます。
- ディスク使用率が高くなると、圧縮やその他のルーチン操作で問題が発生する可能性があります。

  ディスク上のデータファイルの圧縮により、1 つのディスクファイルに圧縮がマージされるため、圧縮操作は特に重要です。これにより、ディスク領域が解放され、読み取りパフォーマンスが向上します。この操作に失敗すると、パフォーマンスが低下する可能性があります。

  ディスク使用率が高くなるためにアラートは、いずれかの条件が満たされる場合に発生します。

  - ストレージノードデータのサイズは、合計ディスク容量の 50% を超えています。
  - 使用されるディスク容量 **の** 合計は、合計ディスク容量の 75% を超えています（ストレージノードが使用するディスク容量に関係なく）。
  - ストレージノードデータとの空きディスク容量の割合は 1.5 未満です。これは、ストレージノードが使用するディスク領域で除算される空きディスク容量を使用して算出されます。50 MB の空き領域があり、ストレージノードが 35MB のディスクを使用している場合、比率は 50/35 または 1.42 になります。低すぎるため、アラートがトリガーされます。

  一時的な使用の際のアラートを防ぐために、分離ルールが実装されます。
- スナップショットの障害 - ローカルルーチンのバックアップ操作に失敗したことを意味します。
- メンテナンス操作に失敗すると、ノードの deploy または undeploy 操作のいずれかが失敗していました。利用不可のリソースなどの根本的な原因はどれでも対処でき、その後操作を再実行できます。

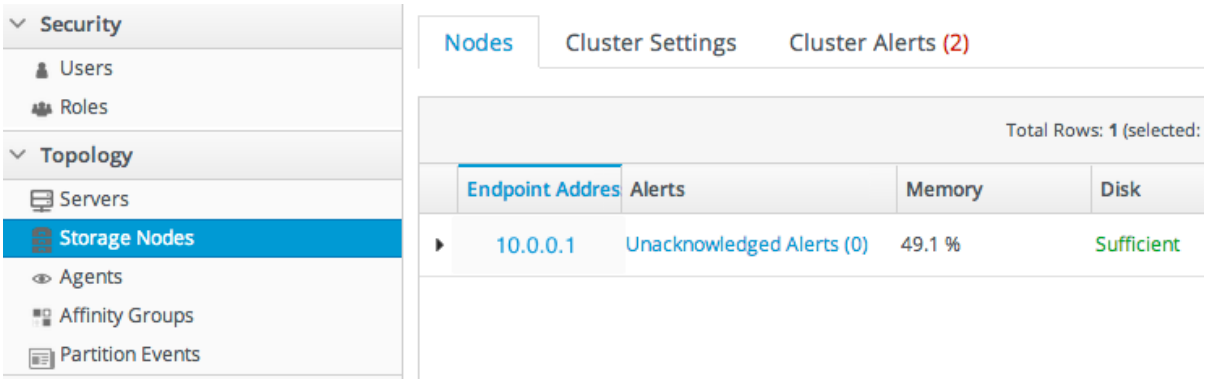
事前定義された各アラートは、ストレージノードの子リソースに対して設定されます。

表9.1 アラートのストレージリソース

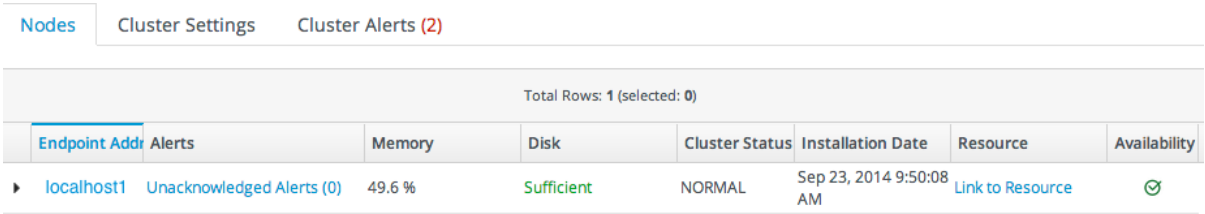
Alert	親リソース	リソースタイプ	範囲からアドレス
ハイヒープ使用	Cassandra Server JVM	Memory サブシステム	storge ノードの JVM 設定でヒープサイズを編集します。
ディスク使用率が高い	データベース管理サービス	ストレージサービス	ノードをホストするシステムのディスク容量を増やします。
スナップショットの障害	データベース管理サービス	ストレージサービス	
メンテナンス操作の失敗		ストレージノード	クラウドで利用できないストレージノード（更新を防ぐ）

ストレージクラスターのアラートを表示するには、以下を実行します。

1. トップナビゲーションバーの **Administration** タブをクリックします。
2. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。



3. **Nodes** タブには、各ノードの承認されていないアラートの数が表示されます。



4. アラートのリストを表示するには、**Cluster Alerts** タブを開きます。

   すべてのアラートは、トリガーした状況、影響を受けるリソース、およびアラートの時間と共に一覧表示されます。

バグを報告します。

9.6. ストレージノードのデバッグモードの有効化

1. ストレージノードマシンで、ストレージノードを停止します。

```
[root@node ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop --storage
```

2. ストレージノードの **log4j-server.properties** ファイルを開きます。

```
[root@server ~]# vim serverRoot/jon-server-3.3.0.GA/rhq-storage/conf/log4j-server.properties
```

3. ログしきい値プロパティー(***log4j.appender.R.Threshold***)および apache プロパティー(***log4j.logger.org.apache.cassandra***)の行を追加または編集し、デバッグロギングを有効にします。

```
log4j.appender.R.Threshold=DEBUG
log4j.logger.org.apache.cassandra=DEBUG
```

4. ストレージノードを再起動します。

```
[root@node ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --storage
```

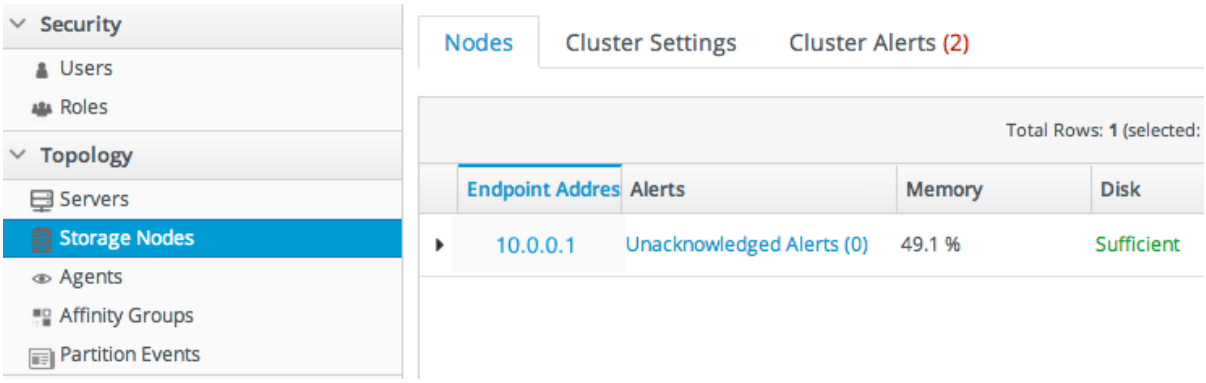
バグを報告します。

9.7. ストレージノードのヒープの管理

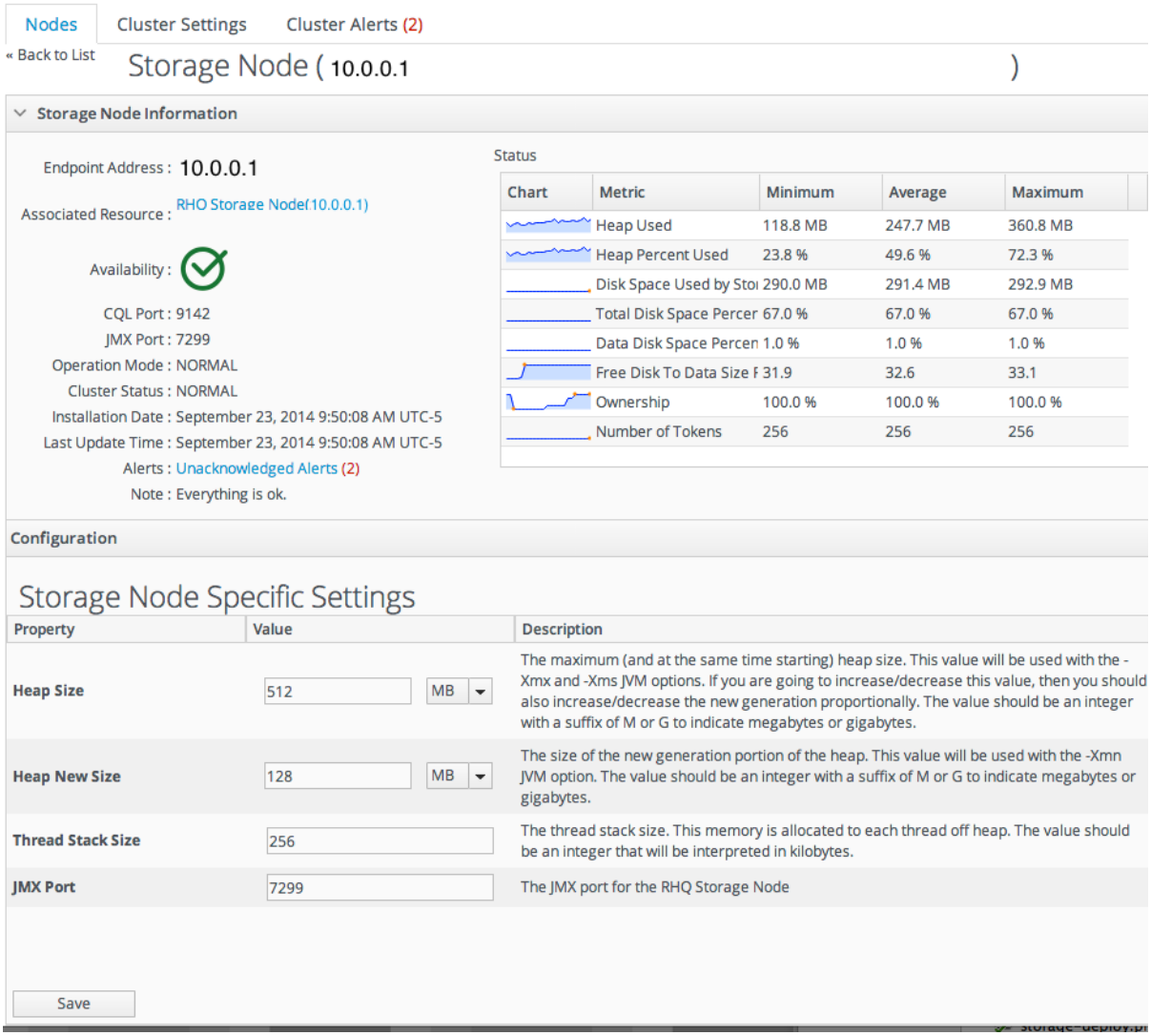
ストレージノードのパフォーマンスに影響を与える最も一般的な問題の 1 つがヒープ不足です。これは、ガベージコレクションなどの通常のデータベース操作に影響を及ぼす可能性があります。これにより、ストレージノードプロセスがメモリー不足のエラーで失敗する可能性があります。

デフォルトでは、ストレージノードの JVM は最大ヒープサイズ 512MB で設定されます。これは、大規模な環境では不十分な場合もあります。ヒープサイズは、ノードの設定でリセットできます。

1. トップナビゲーションバーの **Administration** タブをクリックします。
2. 左側の **Topology** エリアで **Storage Nodes** 項目を選択します。



3. **Nodes** タブで、編集するストレージノードのホスト名または IP アドレスをクリックします。
4. この **Configuration** エリアで **Max Heap Size** 設定をリセットします。



5. 推奨されるように、ヒープサイズに比例して **Heap New Size** 設定を調整します。
6. ページ下部の **Save** ボタンをクリックします。
7. ストレージノードを再起動します。たとえば、ストレージノードマシンで以下を行います。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --storage
```

バグを報告します。

9.8. JBOSS ON サーバーの CONTAINERSANDRA STORAGE ノードのスケーリング

重要

JBoss ON インストールでの負荷や需要に関係なく、JBoss ON サーバーはメンテナンスモードに移行します。

- ストレージノードは、ストレージ要求のバックログを処理します（ピーク時）。
- メモリーからディスクにデータの転送
- データの集約または圧縮を実行します。

ほとんどのインスタンスでは、システムは中断せずに復旧するように設計されています。ただし、システムの需要や負荷が増大すると、システム停止の長さも増えます。これらの停止の長さが割り込み操作が開始されたら、負荷を配布するために追加のストレージノードをインストールする必要があります。ストレージノードにストレージ容量が十分ではない場合にも、追加のストレージノードが必要です。追加のストレージノードのインストールに関する詳細は、を参照してください 「追加ノードのインストール」。

containersandra ベースの JBoss ON Storage Node をスケーリングすると、メトリックの読み取りおよび書き込み要求に対応するストレージノードの容量を改善するなど、パフォーマンスと信頼性が向上します。ストレージノードをスケーリングする方法は複数あります。以下のとおりです。

- の **heap_max** および **heap_new** プロパティーを増やして、JBoss ON Storage ノードの JVM で利用可能なメモリーを増やし ***RHQ_SERVER_HOME*/rhq-storage/conf/cassandra-jvm.properties**ます。例：

  ```
  heap_max=-Xmx1024M
  heap_new=-Xmn384M
  ```

  の説明にあるように、ヒープサイズは JBoss ON UI で変更することもでき 「ストレージノードのヒープの管理」 ます。

  重要

  ホストマシンが最大ヒープに設定された値を処理するのに十分な物理メモリーがあることを確認します。そうでないと、ホストマシンはメモリーのスワップを使用して、ストレージノードの処理速度を遅くします。

- の **max_tenuring_threshold** プロパティーを増やして、新しい生成領域で持続する時間オブジェクトの長さを増やし ***RHQ_SERVER_HOME*/rhq-storage/conf/cassandra-jvm.properties**ます。例：

  ```
  max_tenuring_threshold="-XX:MaxTenuringThreshold=4"
  ```

  警告

  JVM の最大しきい値が 15 を超える場合、JVM ガベージコレクションは正常に機能しません。

- の説明に従って、複数のマシンおよびディスクに処理を分散するために、追加の JBoss ON Storage ノードをストレージクラスターに追加し「追加ノードのインストール」ます。
- パフォーマンスを向上させるために、JBoss ON Storage ノードの **rhq-data** ディレクトリーを専用ディスクに再配置します。
- JBoss ON Storage ノードの **rhq-data** および **commit_log** ディレクトリーを専用の物理ディスクに移動します。
- の **write_request_timeout_in_ms** プロパティーを設定して、書き込みのタイムアウト値を増やし ***RHQ_SERVER_HOME*/rhq-storage/conf/cassandra.yaml**ます。例：

```
write_request_timeout_in_ms: 40000
```

注記

書き込み操作のタイムアウトを増やすことは推奨されません。

バグを報告します。

9.9. ストレージノードの容量の調整によるメトリクス受信（ストレージノードの要求スロットリング）

JBoss ON ストレージノードはリクエストの容量（メトリクスの容量）を自動的に調整しますが、手動の調整が必要になる場合があります。手動の調整は、**rhq-server.properties** ファイルまたは JBoss ON GUI 内でストレージクライアント **設定** を調整することで実行できます。

9.9.1. ストレージノード要求の制限設定（ストレージクライアント設定）

rhq.storage.request.limit.topology-delta=30000

トポロジー差分は、追加のストレージノードが追加されると要求の制限が 1 分あたりに増加します。たとえば、既存のストレージノードに 1 分あたりのリクエストの制限がある場合、新規ストレージノードを追加すると 1 分あたり 60,000 要求に制限を増加します。ストレージノードの追加に関する詳細は、を参照してください「追加ノードのインストール」。

rhq.storage.request.limit.timeout-dampening=30000

タイムアウトの Dampener は、タイムアウトが連続して無視される期間（ミリ秒単位）を定義します（デフォルトは │ms）。この timeout Dampening プロパティーでは、タイムアウトが発生すると、すぐにそれに続くタイムアウトがあることを前提としています。

rhq.storage.request.limit.timeout-delta=0.2

（JBoss ON バージョン 3.3.1）タイムアウトの差は、リクエストのタイムアウトが発生したときに要求の制限で比例の減少（0 ~ 1 の間）を定義します。ここでは、要求のタイムアウトが発生するのは、ストレージノードのハードウェアが現在の要求制限で要求を処理するのに十分な容量がないことが前提です。たとえば、最初の制限が 1 分あたりのリクエスト数で、リクエストタイムアウトが発生すると、制限は 1 分あたり 20% から 2 ミリ秒に減少します（0.2 * ✓ リクエストの 1 分あたりのリクエスト数 = │ リクエストの減少）。したがって、新しいリクエスト制限は 2 つのリクエストに制限されます。

rhq.storage.request.limit.min=5000

（JBoss ON バージョン 3.3.1 から非推奨）。リクエストの最小制限を **rhq.storage.request.limit.min** 設定します。ストレージ要求の制限は、タイムアウトの数に関係なく、この値を下回ることができません。

rhq.storage.request.limit.warmup-period=3

（JBoss ON 3.3.1 更新によって導入される）ホットアップ期間は、ストレージ要求のタイムアウト発生時の増加（分単位）を設定します。

rhq.storage.request.limit.max-warmup-counter=10

（JBoss ON 3.3.1 更新で導入される）最大ヒートアップカウンターは、連続するストレージ要求のタイムアウト後に最大モニタリング時間を設定します。最大ヒュームアップ時間は、Wareft up カウンターとヘルスアップ期間の機能です。たとえば、で **warmup-period=3** 乗算した場合に **max-warmup-counter=10** は、最大浮動時間（30 分）を指定します。

ストレージ要求の制限のホットアップ設定の使用

タイムアウトが発生すると、ストレージ要求の制限の半分が設定されます。ストレージ要求の制限は、設定済みのストレージ要求の制限の 100% に返され、まずはホットアップ時間に相当し **rhq.storage.request.limit.warmup-period**ます。（アカウンティング後 **rhq.storage.request.limit.timeout-dampening**）ウェイクアップ時間帯に後続のタイムアウトが発生すると、最大ウェイトアップ時間まで増加します。最大ヒートアップ時間は、で指定され **rhq.storage.request.limit.max-warmup-counter** ます **rhq.storage.request.limit.warmup-period** （デフォルトの最大ホットアップ時間は 30 分）。

9.9.2. 「ストレージノードの要求制限設定の手動変更」 rhq-server.properties

ストレージノードの要求制限設定は、ストレージクライアント **設定** **<JBoss_ON_install_directory>/bin/rhq-server.properties** で手動で変更できます。

RHQ ストレージ要求の制限および JBoss ON バージョン 3.3.1 で追加さ **<JBoss_ON_install_directory>/bin/rhq-server.properties.new** れた。ストレージ要求の制限のホットアップ設定（**warmup-period** および **max-warmup-counter**）を変更するには、にあるように設定を手動で追加するか、「ストレージノード要求の制限設定（ストレージクライアント設定）」 または **<JBoss_ON_install_directory>/bin/rhq-server.properties** ファイルからコピーする必要があり **<JBoss_ON_install_directory>/bin/rhq-server.properties.new**ます。

9.9.3. JBoss ON GUI を使用したストレージノードの要求のスロットリングの調整

JBoss ON GUI で Storage Node Request Throtling 設定を変更するには、以下を実行します。

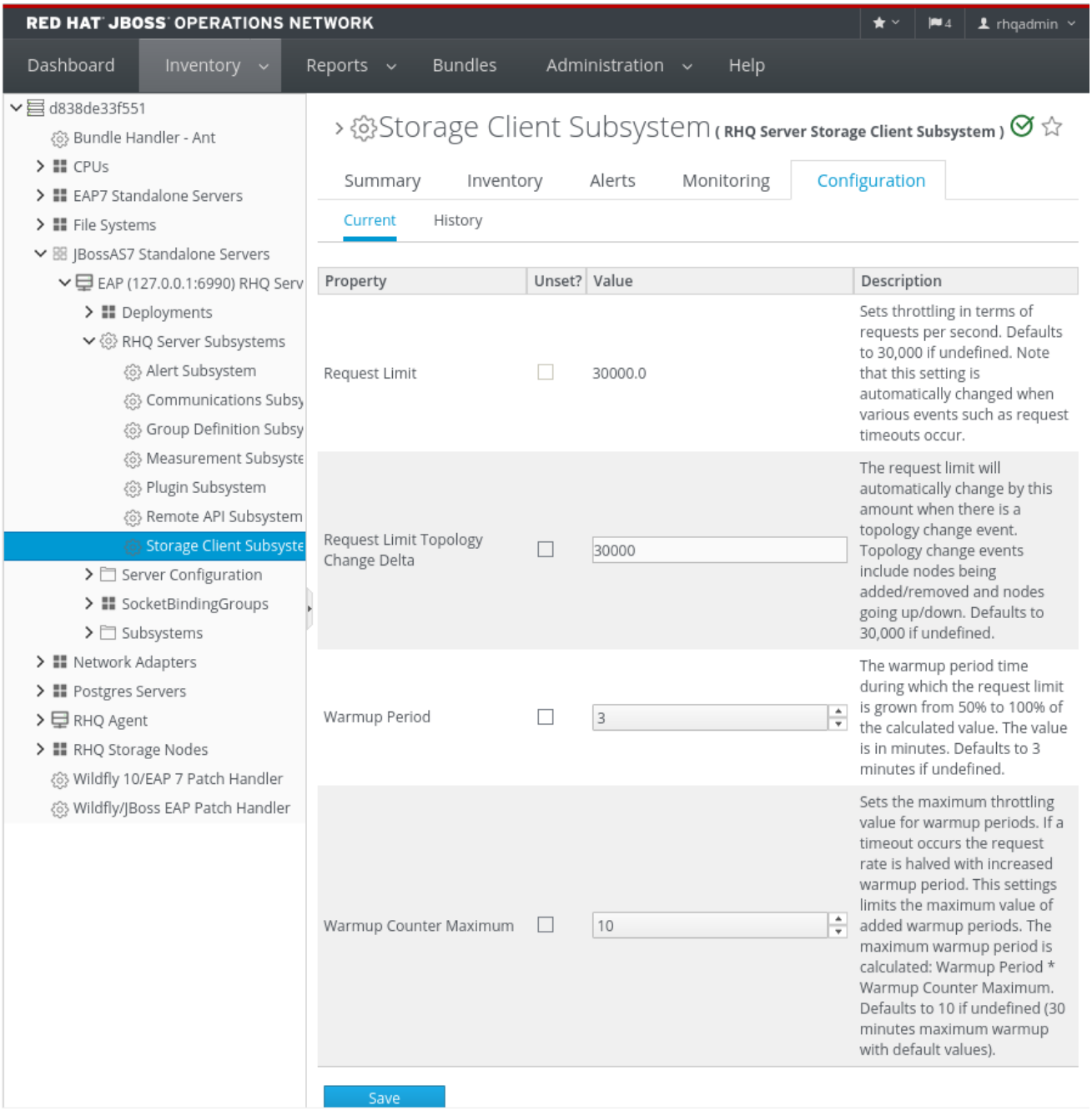
1. **Inventory** タブを選択します。
2. **Resources** メニューから、を選択し **All Resources**ます。
3. を開き **Storage Client Subsystem**、開く名前を選択します。

4. **Storage Client Subsystem** ページから **Configuration** タブを選択します。

5. プロパティーに変更を加えます。デフォルト値を使用するには **Unset?** チェックボックスを選択するか、カスタムの値を使用するには **Unset?** チェックボックスの選択を解除します。

6. ページの **Save** 下部をクリックします。

図9.4 JBoss ON GUI 内のストレージノードのリクエストスロットリング設定の例



バグを報告します。

9.10. メトリクスストレージデータベースのバックアップおよび復元

ストレージノードには、データの管理およびアーカイブを行うためのフローが定義されます。スケジュールされた **スナップショット** は、すべてのノードデータ（メトリクスおよびノード設定の両方）をバックアップします。これにより、クラスターが破損した場合にデータを復元できます。

バグを報告します。

9.10.1. ストレージデータファイルについて

ノードのデータベースファイルはすべて **rhq-data**/ ディレクトリーに保存されます。同じルートディレクトリーにあるのは JBoss ON サーバー（など **/opt/jon/jon-server-3.3.0.GA**）です。このディレクトリーは、3 つのサブディレクトリーに分けられます。

- データベースに書き込まれていない書き込みデータのコミットログ。
- アーカイブのキャッシュディレクトリー。
- 監視データ（集計および raw）、監視データのスナップショットアーカイブ、およびノード（システム）データの両方を含むデータディレクトリー。

```
        rhq-data/
          |
  ---------------------
  |          |     |
commit_log/  data/  saved_caches/
```

commit_log/ ディレクトリーには、書き込みがデータベースに書き込まれる前に書き込みを保存するバイナリーコミットログが含まれます。

saved_caches/ ディレクトリーには、ストレージノード内のメジャーテーブルのアーカイブ（メトリクステーブルとストレージノード設定の両方）が含まれます。

data/ ディレクトリーには、最も重要なデータが含まれます。このディレクトリーはキースペース別に編成されます。キースペースには、システム関連の設定、認証、およびスキーマの詳細とともに、生および集計されたメトリックデータが含まれます。

の下にあるキースペースディレクトリーには **snapshot**/ サブディレクトリーが **data**/ 含まれます。**snapshot**/ ディレクトリーは、親キースペースにあるデータの特定の時点および関連するデータファイルのスナップショットまたはバックアップを表します。

バグを報告します。

9.10.2. ストレージノードのスナップショット

JBoss ON は、監視およびシステムデータのスナップショットまたはバックアップを作成する機能を提供します。

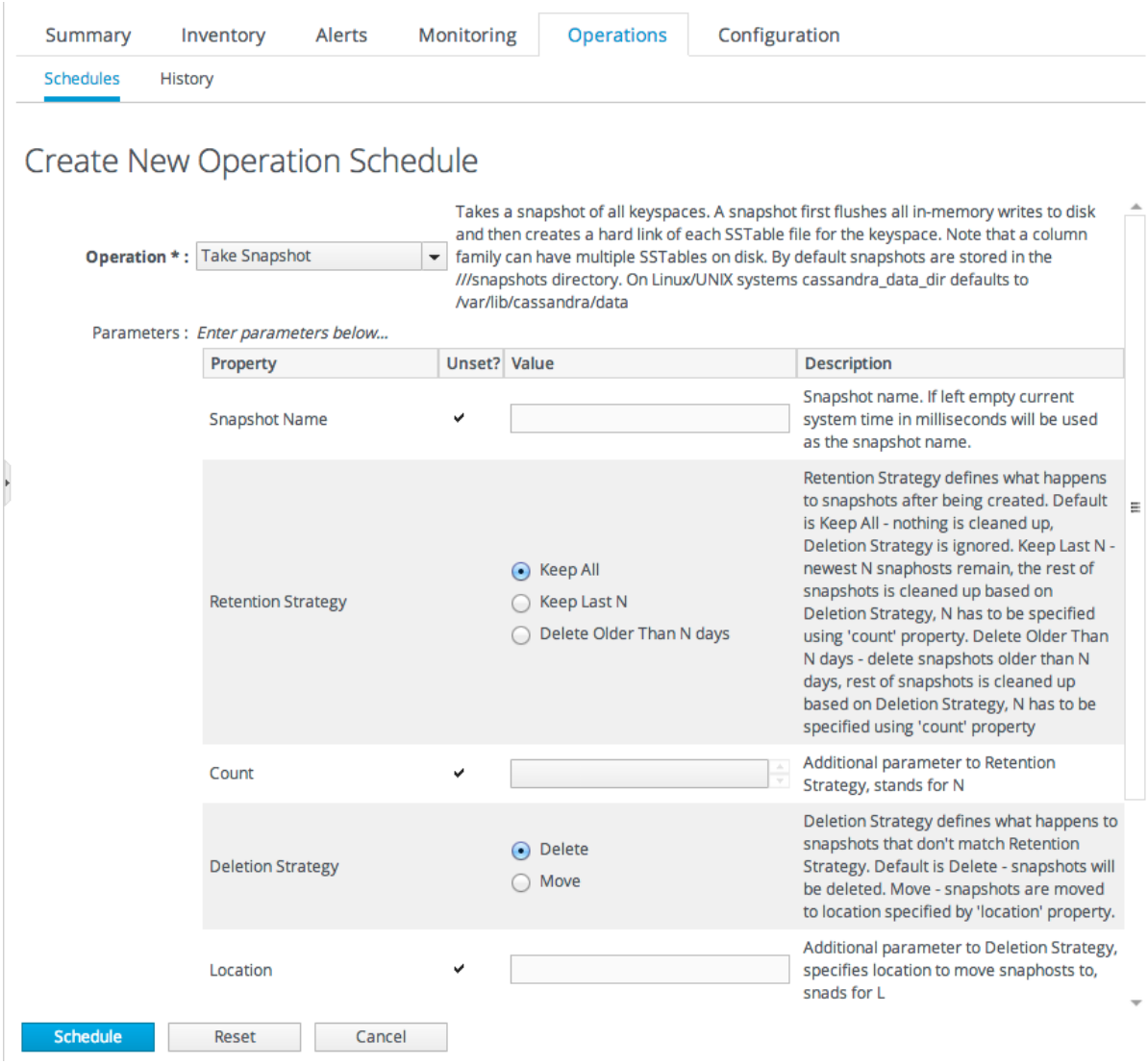
バグを報告します。

9.10.2.1. 個別のスナップショットの作成

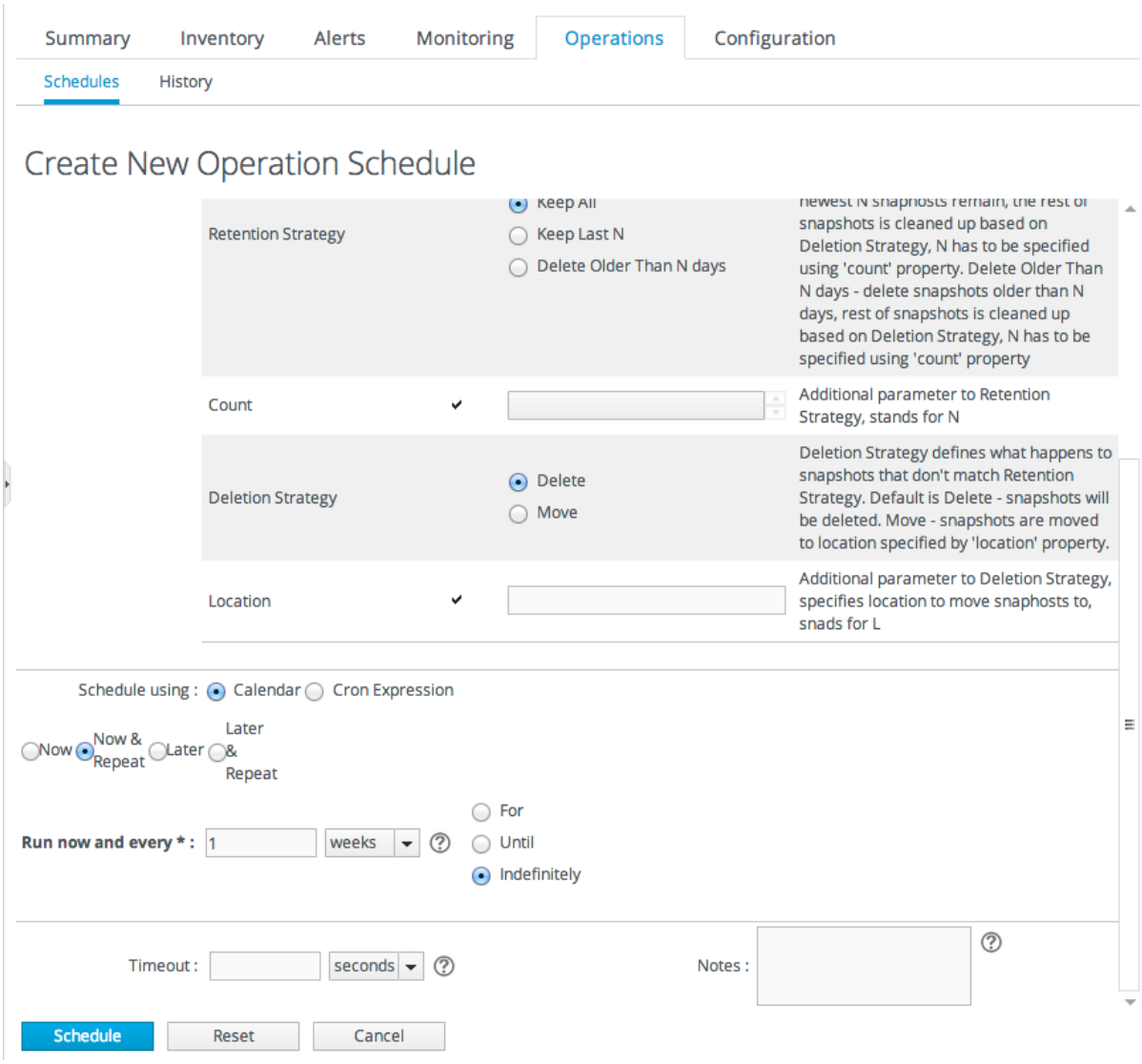
スナップショットは、以下の手順を使用してスケジュールできます。

1. 上部ナビゲーションバー **Inventory** をクリックし、左側のナビゲーションバーの **Resources** セクション **Servers - Top Level Imports** にあるをクリックします。
2. 必要なストレージノードをクリックします。
3. **Operations** タブをクリックし、下部の **New** ボタンをクリックします。
4. **Operation** ドロップダウンメニュー **Take Snapshot** から選択します。

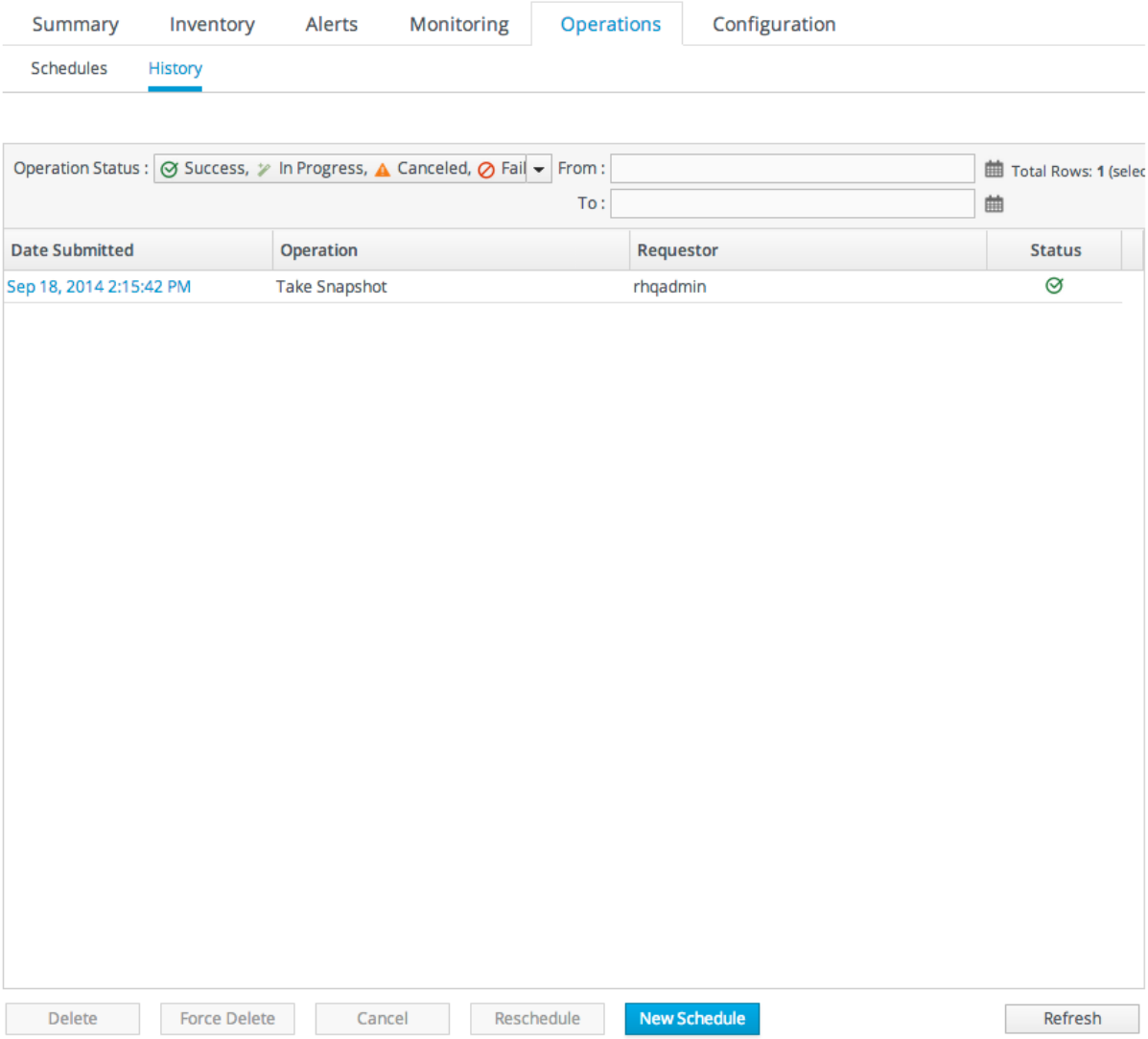
5. 必要なスナップショットおよび保持オプションを入力します。



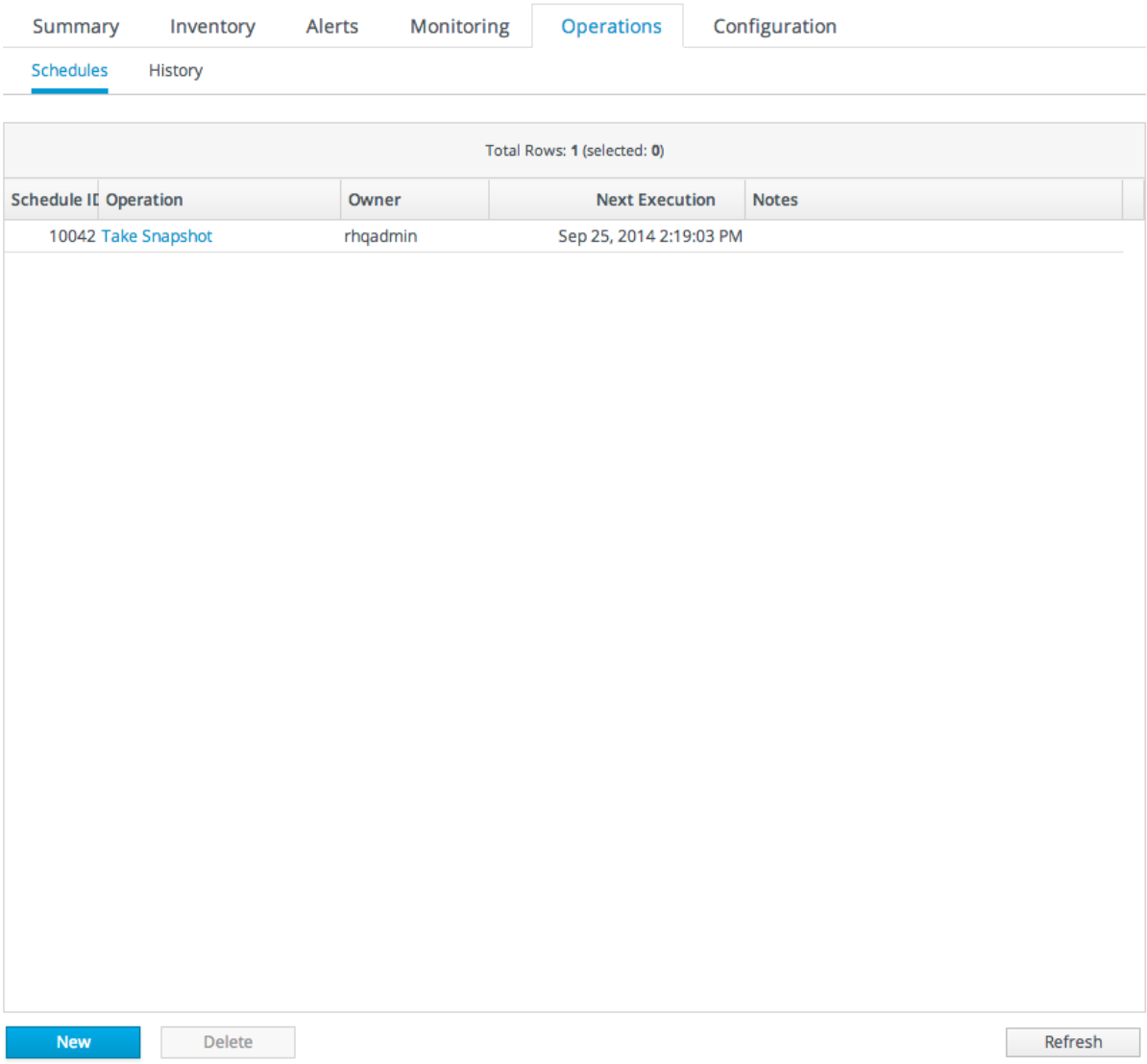
6. **オプション。**スナップショットは後でスケジュールしたり、指定したサイクルで繰り返すこともできます。デフォルトでは、スナップショットは「Now」にスケジュールされます。



7. **Schedule** をクリックし、スナップショットをスケジュールします。その後、ユーザーは操作履歴セクションに取得され、スナップショットのステータスが表示されます。



8. スナップショットが後にスケジュールされている場合や、繰り返しに設定された場合には、そのスナップショットも **Schedules** サブタブに表示されます。



バグを報告します。

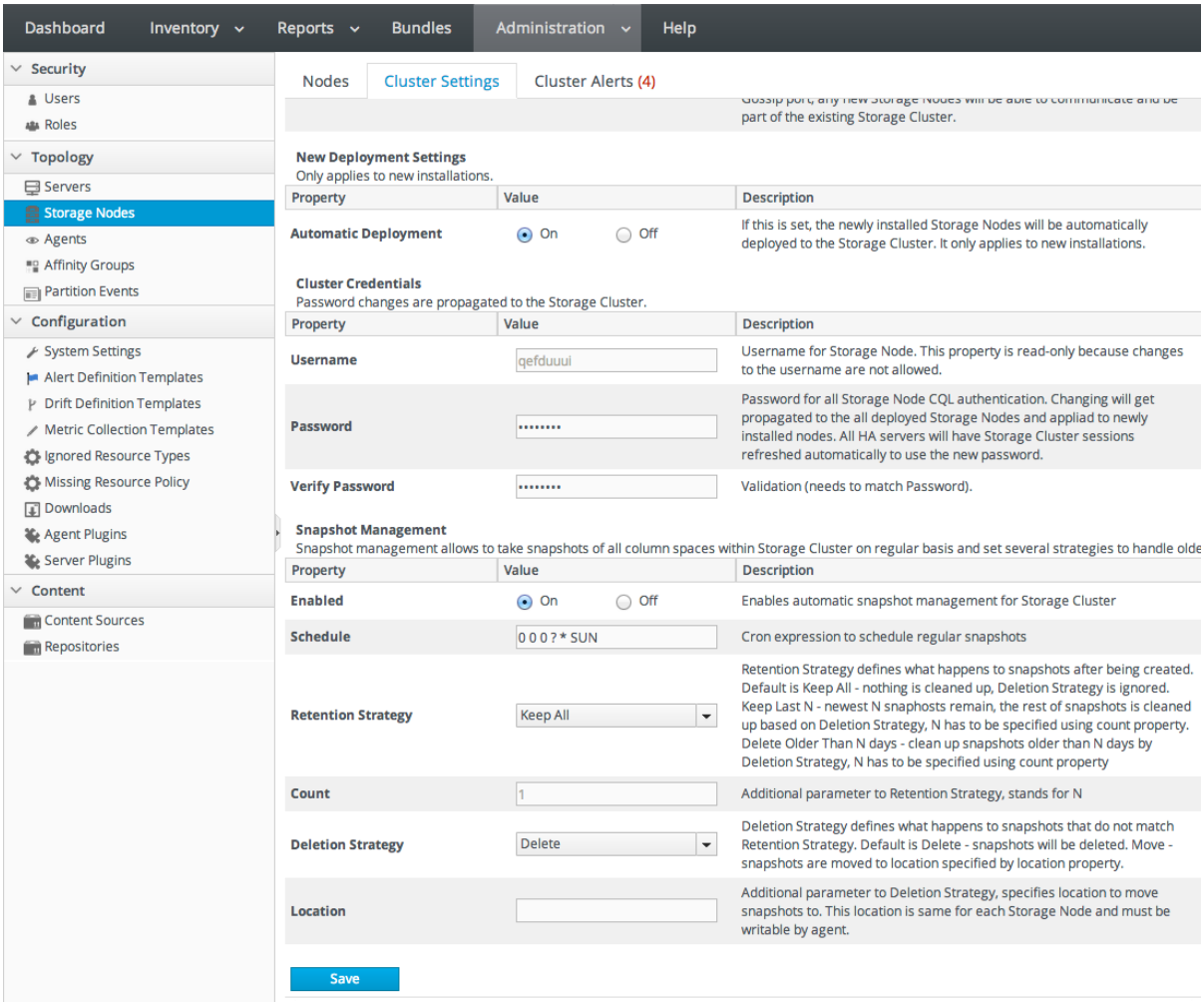
9.10.2.2. ストレージノードクラスターのスナップショットの再作成のスケジューリング

各ストレージノードでスナップショットの繰り返し作成以外に、ストレージノードのクラスターレベルで繰り返しスナップショットをスケジュールすることもできます。

ストレージノードクラスターのスナップショットをスケジュールするには、以下を実行します。

1. 上部ナビゲーションバー **Adminstration** をクリックし、左側のナビゲーションバーの **Topology** セクション **Storage Nodes** にあるをクリックします。

2. **Cluster Settings** タブをクリックし、**Snapshot Management** セクションまでスクロールします。



3. **Enabled** プロパティーをに設定すると、スナップショットの繰り返しを有効にすることができ **On**ます。

4. 必要なスケジュールおよび保持ストラテジーを設定します。

5. をクリックし **Save**ます。

注記

ストレージノードのクラスターレベルでスケジュールされるスナップショットは、個別にスケジュールされたスナップショットに加えて、各ストレージノードの **Schedules** サブタブも表示されます。スナップショットのスケジュールがストレージノードのクラスターレベルで更新される場合、これは各ストレージノードの **Schedules** サブタブにも反映されます。ストレージノードでスケジュールされた繰り返しスナップショットが無効にされると、各ストレージノードの **Schedules** サブタブから削除され、**History** サブタブには表示されません。

重要

スナップショットは CPU とディスク集約型です。使用率が高い時間にスナップショットを実行すると、パフォーマンスが低下します。

バグを報告します。

9.10.3. クラスターの復元

注記

復元プロセスにより、クラスター全体が以前の状態に復元されます。

これは、単一のノードを復元する予定ではありません。すべてのデータはすべてのノード間で複製されるため、障害が発生しているノードを削除し、ノードの復元を試みるよりも簡単かつ安全です。

重要

すべての手順は、クラスター内のすべてのノードで実行する必要があります。

1. ストレージクラスター内のすべてのノードをシャットダウンします。すべてのストレージマシンで stop コマンドを実行します。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl.sh stop --storage
```

2. 各ノードの **commit_log**/ ディレクトリーを削除します。

```
[root@server ~]# rm * /opt/jon/rhq-data/data/commit_log/*
```

3. **スナップショットファイルを除く、以下のディレクトリーにあるすべてのファイルを** 削除します。

   - /opt/jon/rhq-data/data/rhq/metrics_index/
   - /opt/jon/rhq-data/data/rhq/one_hour_metrics/
   - /opt/jon/rhq-data/data/rhq/raw_metrics/
   - /opt/jon/rhq-data/data/rhq/schema_version/
   - /opt/jon/rhq-data/data/rhq/six_hour_metrics/
   - /opt/jon/rhq-data/data/rhq/twenty_four_hour_metrics/

```
[root@server ~]# cd /opt/jon/rhq-data/data/rhq/
[root@server rhq]# rm metrics_index/*.* one_hour_metrics/*.* raw_metrics/*.*
schema_version/*.* six_hour_metrics/*.* twenty_four_hour_metrics/*.*
```

4. スナップショットディレクトリーからディレクトリーにすべてのファイルをコピーし **metrics_index** ます。

```
[root@server rhq]# cp metrics_index/snapshots/timestamp/* /opt/jon/rhq-
data/data/rhq/metrics_index
```

   残りのディレクトリーに対してコマンドを繰り返します。

   - /opt/jon/rhq-data/data/rhq/metrics_index/
   - /opt/jon/rhq-data/data/rhq/one_hour_metrics/
   - /opt/jon/rhq-data/data/rhq/raw_metrics/

- /opt/jon/rhq-data/data/rhq/schema_version/
- /opt/jon/rhq-data/data/rhq/six_hour_metrics/
- /opt/jon/rhq-data/data/rhq/twenty_four_hour_metrics/

5. 各ストレージノードを再起動します。

```
[root@server ~]# serverRoot/jon-server-3.3.0.GA/bin/rhqctl start --storage
```

バグを報告します。

付録A ドキュメント履歴

改訂 3.3.7-3 **Tue 11 Apr 2017** **Conscot ムムー**

さまざまなバグ修正を再構築します。

改訂 3.3.7-2 **Mon 05 Sep 2016** **Conscot ムムー**

BZ-1370474: Procedure 4.1 に示されている設定ファイルで誤った値を修正します。

BZ-1369081: 複数の rhq-server.properties エントリーで正しくない値が修正されました。

BZ-1370498: SSL 設定例に設定が欠落している。キーストア/トラストストアの更新に関する注意点を追加

BZ-1370489: クライアント認証の有効化で明確化されたテキスト

改訂 3.3.2-13 **Tue 23 Feb 2016** **Conscot ムムー**

BZ-1323876: 「Setting up Client Authentication Between Servers and Agents」の章

改訂 3.3.2-12 **Tue 23 Feb 2016** **Conscot ムムー**

BZ-1279557 および BZ-1279556: Updated list secure-socket-protocols

bz-1214414: DSA アルゴリズムを使用して、RSA アルゴリズムを使用するコマンドを RSA に更新します。

bz-1187400: 手順で SSL トランスポートコネクターオプションの使用。

改訂 3.3.2-11 **Thu 02 Jul 2015** **Jared イタリア**

JBoss ON UI のヘルプ > ドキュメントメニューにリンクしていた HTML Anchors が修正されました。

プリフェイスの問題が削除されました。

改訂 3.3.2-1 **Tue 28 Apr 2015** **Jared イタリア**

JBoss ON 3.3.2 リリースの準備

改訂 3.3.1-1 **Wed Feb 18 2015** **Jared イタリア**

JBoss ON 3.3.1 向け準備、リリース

改訂 3.3-32 **Mon Nov 17 2014** **Jared イタリア**

『3.3 Release Notes for Customer Reported Defects』を参照してください。

https://bugzilla.redhat.com/show_bug.cgi?id=1095805: JVM パラメーターの問題を修正しました。「サーバー JVM の調整」

https://bugzilla.redhat.com/show_bug.cgi?id=1118072: 手順で 2 つの手順を変更して、必要な追加のテーブルを反映させます。「クラスターの復元」

https://bugzilla.redhat.com/show_bug.cgi?id=1128770: の説明に従って、-g --purgeplugins オプションを追加しました。「エージェント起動オプション」

https://bugzilla.redhat.com/show_bug.cgi?id=1148601: High Availability から Topology に変更し、でメニュー選択に変換してください。「エージェントのサーバーフェイルオーバー一覧の表示」

https://bugzilla.redhat.com/show_bug.cgi?id=1148576: 新しいスクリーンショットを反映するように手順を更新しました。「アフィニティーグループの作成」。また、このセクションで copy-edit および XML tweak も行います。

https://bugzilla.redhat.com/show_bug.cgi?id=1148584: の手順に従って、チェックボックスではなく強調表示する行が表示されています。「高可用性クラウドからのサーバーの削除」

https://bugzilla.redhat.com/show_bug.cgi?id=1148581: 順序付けリストを手順、図の変更、および更新された手順テキストに変換します。「サーバーをメンテナンスモードに」

https://bugzilla.redhat.com/show_bug.cgi?id=1111311: 「メール通知の SMTP サーバーの設定」セクションの email.jsp パスの問題を修正しました。