Red Hat OpenStack Platform 16.0

監視ツール設定ガイド

OpenStack のロギングおよび監視ツールについてのガイド

Last Updated: 2022-09-13

Red Hat OpenStack Platform 16.0 監視ツール設定ガイド

OpenStack のロギングおよび監視ツールについてのガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本ガイドでは、Red Hat OpenStack Platform 環境でのロギングと監視の設定方法について説明します。

目次

第1章 はじめに

監視ツールは、オペレーターが OpenStack 環境を維持管理するのに役立つオプションのツールセットです。これらのツールは、以下の機能を果たします。

- 集中ロギング: OpenStack 環境の全コンポーネントからのログを 1 つの場所に収集することができます。すべてのノードとサービスにわたって問題を特定することができます。また、オプションで Red Hat にログデータをエクスポートして、問題を診断するサポートを受けることもできます。

- 可用性監視: OpenStack 環境内の全コンポーネントを監視して、いずれかのコンポーネントが現在停止中または機能していない状態かどうかを判断します。また、問題が特定された時にシステムが警告を送信するように設定することも可能です。

第2章 監視アーキテクチャー

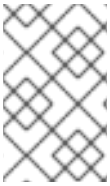
監視ツールは、クライアントが Red Hat OpenStack Platform オーバークラウドノードにデプロイされる、クライアント/サーバーモデルを使用します。Rsyslog サービスは、クライアント側の集中ロギング(CL)および collectd を提供し、有効な sensubility プラグインはクライアント側の可用性監視(AM)を提供します。

2.1. 集中ロギング

Red Hat OpenStack Platform 環境では、デバッグおよび管理を簡素化するために、1 つの場所に全サービスからログを収集します。これらのログは、syslog や監査ログファイル等のオペレーティングシステム、RabbitMQ や MariaDB 等のインフラストラクチャーコンポーネント、および Identity や Compute 等の OpenStack サービスから収集されます。

集中ロギングのツールチェーンは、以下のコンポーネントで構成されます。

- ログ収集エージェント (Rsyslog)
- データストア (Elasticsearch)
- API/プレゼンテーション層 (Kibana)



注記

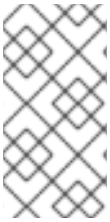
Red Hat OpenStack Platform director は、集中ロギング向けのサーバー側のコンポーネントはデプロイしません。Red Hat は、Elasticsearch データベースおよび Kibana を含むサーバー側のコンポーネントはサポートしません。

2.2. 可用性監視

可用性監視により、OpenStack 環境全体にわたる全コンポーネントのハイレベルな機能性を一元的に監視することができます。

可用性監視のツールチェーンは、複数のコンポーネントで構成されます。

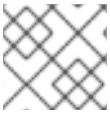
- 監視エージェント (有効な collectd-sensubility プラグイン)
- 監視リレー/プロキシー (RabbitMQ)
- 監視コントローラー/サーバー (Sensu サーバー)
- API/プレゼンテーション層 (Uchiwa)



注記

Red Hat OpenStack Platform director は、サーバー側の可用性監視のコンポーネントはデプロイしません。Red Hat では、Uchiwa、Sensu Server、Sensu API plus、RabbitMQ、監視ノードで実行する Redis インスタンスなどのサーバー側のコンポーネントはサポートしていません。

以下の図は、可用性監視のコンポーネントとそれらの対話を示しています。



注記

青で示した項目は Red Hat のサポート対象コンポーネントです。

図2.1 ハイレベルでの可用性監視のアーキテクチャー

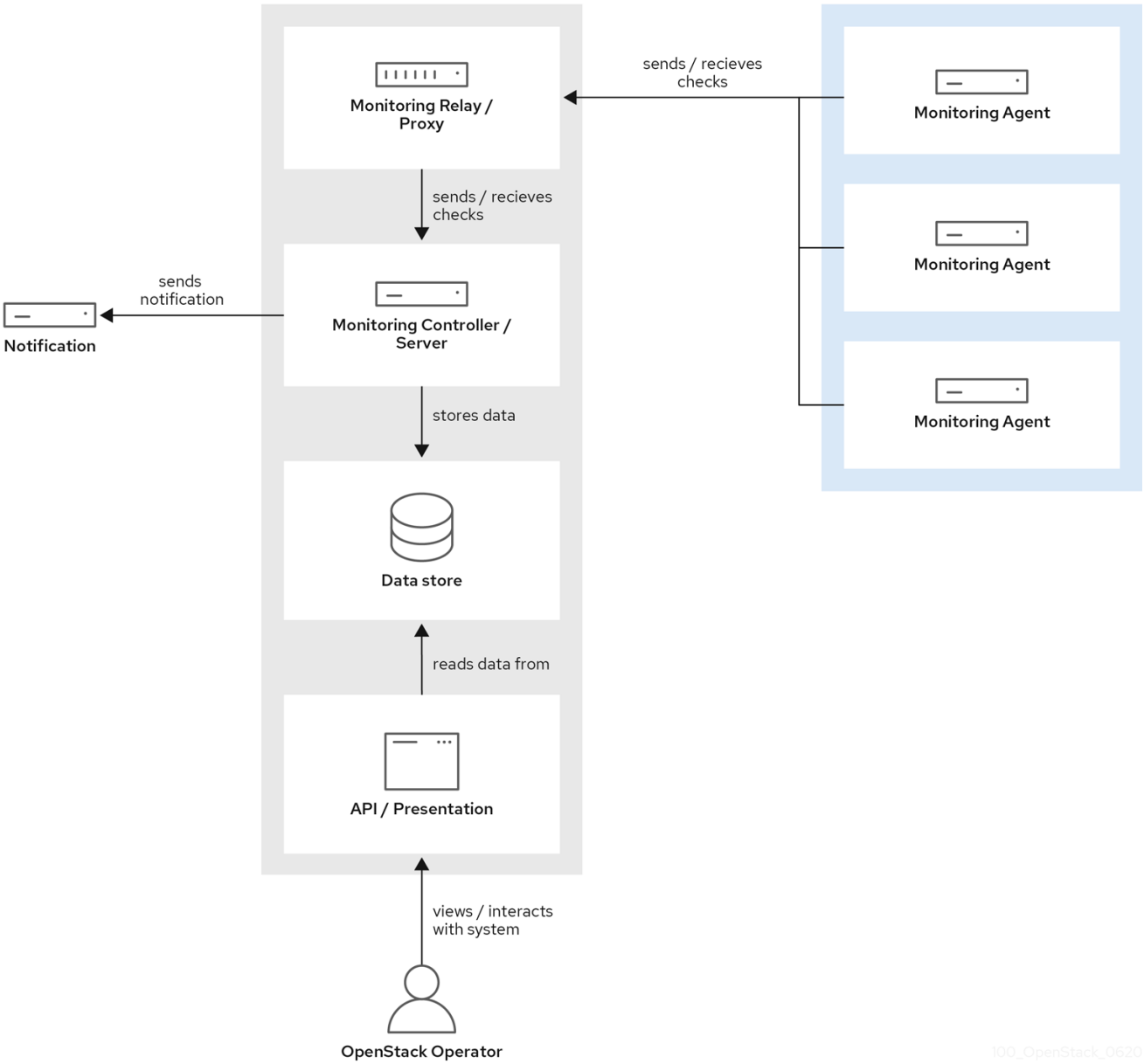


図2.2 Red Hat OpenStack Platform の単一ノードデプロイメント

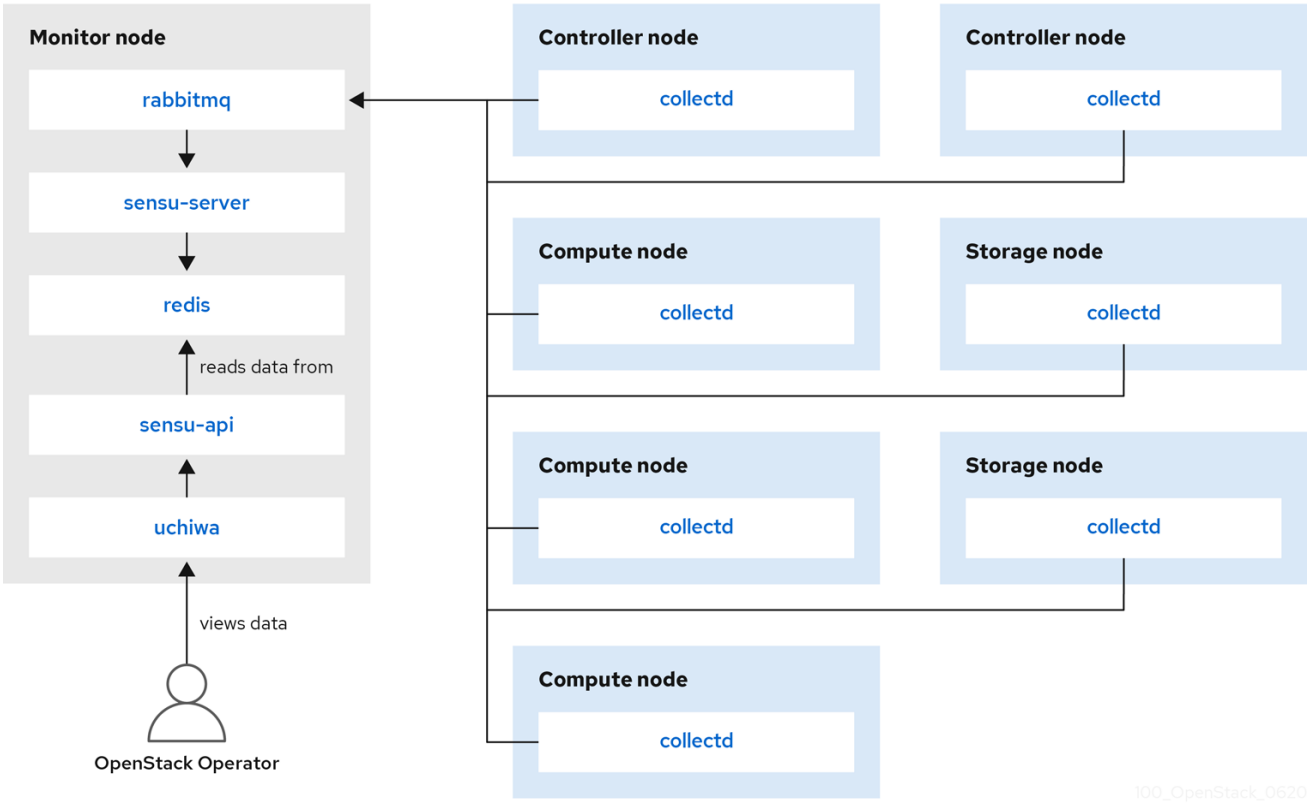
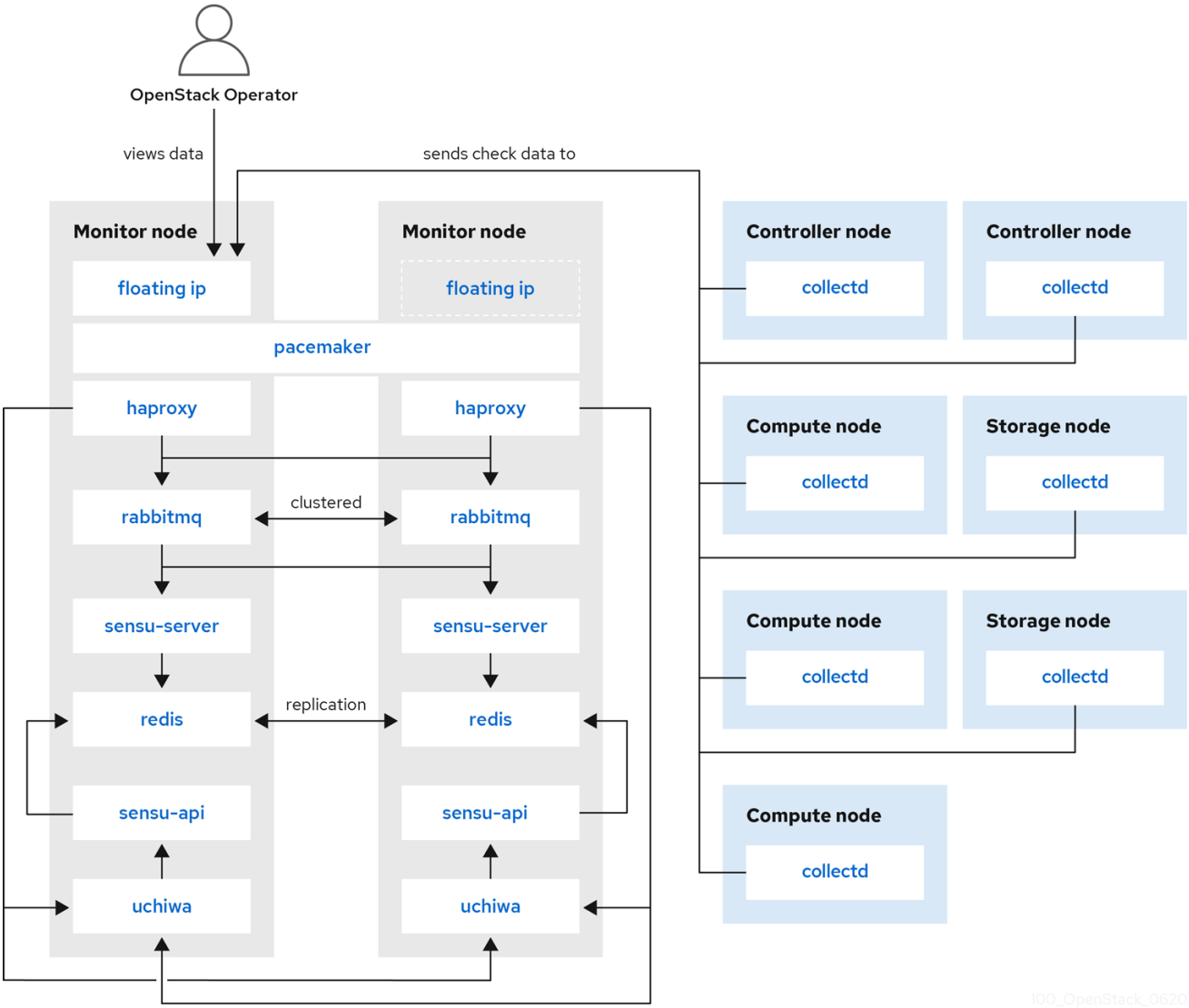


図2.3 Red Hat OpenStack Platform の HA デプロイメント



第3章 クライアント側のツールのインストール

オーバークラウドをデプロイする前には、各クライアントに適用する構成の設定を決定する必要があります。Heat テンプレートコレクションからサンプルの環境ファイルをコピーし、ご使用の環境に応じてファイルを変更します。

3.1. 集中ロギングのクライアントパラメーターの設定

詳細は、「 デプロイメント時の集中ロギングの有効化 」を参照してください。

3.2. 監視クライアントパラメーターの設定

監視ソリューションは、定期的にシステム情報を収集し、データ収集エージェントを使用してさまざまな方法で値を保管し、監視するメカニズムを提供します。Red Hat は、collectd をコレクションエージェントとしてサポートします。collectd-sensubility は collectd の機能拡張であり、RabbitMQ を介して Sensu サーバー側と通信します。Service Telemetry Framework (STF) を使用してデータを保存し、続いてシステムの監視、パフォーマンスのボトルネックの特定、将来のシステム負荷の予測を行うことができます。詳細は、『 Service Telemetry Framework 』ガイドを参照してください。

collectd および collectd-sensubility を設定するには、以下の手順を完了します。

1. ホームディレクトリーに **/home/templates/custom** などの **config.yaml** を作成し、**MetricsQdrConnectors** パラメーター が STF サーバー側をポイントするように設定します。

```
MetricsQdrConnectors:
  - host: qdr-normal-sa-telemetry.apps.remote.tld
    port: 443
    role: inter-router
    sslProfile: sslProfile
    verifyHostname: false
MetricsQdrSSLProfiles:
  - name: sslProfile
```

2. **config.yaml** ファイルで、**CollectdExtraPlugins** の下に必要なプラグインを一覧表示します。**ExtraConfig** セクションにパラメーターを指定することもできます。デフォルトでは、collectd のプラグインには **cpu**、**df**、**disk**、**hugepages**、**interface**、**load**、**memory**、**processes**、**tcpconns**、**unixsock**、および **uptime** があります。追加のプラグインは、**CollectdExtraPlugins** パラメーターを使用して追加できます。また、以下に示す **ExtraConfig** オプションを使用して、**CollectdExtraPlugins** の設定情報を追加することもできます。たとえば、**virt** プラグインを有効にし、接続文字列とホスト名の形式を設定するには、以下の構文を使用します。

```
parameter_defaults:
  CollectdExtraPlugins:
    - disk
    - df
    - virt

  ExtraConfig:
    collectd::plugin::virt::connection: "qemu:///system"
    collectd::plugin::virt::hostname_format: "hostname uuid"
```

注記

unixsock プラグインは削除しないでください。削除すると、collectd コンテナーは正常ではないと永続的に指定されます。

3. collectd-sensubility を有効にするには、以下の環境設定を **config.yaml** ファイルに追加します。

```
parameter_defaults:
  CollectdEnableSensubility: true

  # Use this if there is restricted access for your checks by using the sudo command.
  # The rule will be created in /etc/sudoers.d for sensubility to enable it calling restricted commands via sensubility executor.
  CollectdSensubilityExecSudoRule: "collectd ALL = NOPASSWD: <some command or ALL for all commands>"

  # Connection URL to Sensu server side for reporting check results.
  CollectdSensubilityConnection: "amqp://sensu:sensu@<sensu server side IP>:5672//sensu"

  # Interval in seconds for sending keepalive messages to Sensu server side.
  CollectdSensubilityKeepaliveInterval: 20

  # Path to temporary directory where the check scripts are created.
  CollectdSensubilityTmpDir: /var/tmp/collectd-sensubility-checks

  # Path to shell used for executing check scripts.
  CollectdSensubilityShellPath: /usr/bin/sh

  # To improve check execution rate use this parameter and value to change the number of goroutines spawned for executing check  scripts.
  CollectdSensubilityWorkerCount: 2

  # JSON-formatted definition of standalone checks to be scheduled on client side. If you need to schedule checks
  # on overcloud nodes instead of Sensu server, use this parameter. Configuration is compatible with Sensu check definition.
  # For more information, see https://docs.sensu.io/sensu-core/1.7/reference/checks/#check-definition-specification
  # There are some configuration options which sensubility ignores such as: extension, publish, cron, stdin, hooks.
  CollectdSensubilityChecks:
    example:
      command: "ping -c1 -W1 8.8.8.8"
      interval: 30

  # The following parameters are used to modify standard, standalone checks for monitoring container health on overcloud nodes.
  # Do not modify these parameters.
  # CollectdEnableContainerHealthCheck: true
  # CollectdContainerHealthCheckCommand: <snip>
  # CollectdContainerHealthCheckInterval: 10
  # The Sensu server side event handler to use for events created by the container health check.
  # CollectdContainerHealthCheckHandlers:
```

```
#   - handle-container-health-check
# CollectdContainerHealthCheckOccurrences: 3
# CollectdContainerHealthCheckRefresh: 90
```

4. オーバークラウドをデプロイします。overcloud deploy コマンドに、**config.yaml**、**collectd-write-qdr.yaml** ファイル、および **qdr-*.yaml** ファイルのいずれかを含めます。以下に例を示します。

```
$ openstack overcloud deploy
-e  /home/templates/custom/config.yaml
-e tripleo-heat-templates/environments/metrics/collectd-write-qdr.yaml
-e tripleo-heat-templates/environments/metrics/qdr-form-controller-mesh.yaml
```

5. オプション： オーバークラウドの RabbitMQ 監視を有効にするには、overcloud deploy コマンドに **collectd-read-rabbitmq.yaml** ファイルを追加します。YAML ファイルの詳細は、「YAML ファイル」を参照してください。

3.3. YAML ファイル

collectd を設定する際に、**オーバークラウドのデプロイ** コマンドに以下の YAML ファイルを追加することができます。

- **collectd-read-rabbitmq.yaml**: python-collect-rabbitmq を有効にし、オーバークラウドの RabbitMQ インスタンスを監視するように設定します。
- **collectd-write-qdr.yaml**: collectd が QPID ディスパッチルーターを使用して Telemetry および通知データを送信できるようにします。
- **qdr-edge-only.yaml**: QPID ディスパッチルーターのデプロイメントを有効にします。各オーバークラウドノードでは、エッジモードで 1 つのローカルの qdrouterd サービスが実行および操作されます。たとえば、受信データを定義された MetricsQdrConnectors に直接送信します。
- **qdr-form-controller-mesh.yaml**: QPID ディスパッチルーター(QDR)のデプロイメントを有効にします。各オーバークラウドノードでは、1 つのローカルの qdrouterd サービスが実行され、メッシュトポロジーを形成します。たとえば、コントローラーで実行される QDR は、定義された MetricsQdrConnectors に接続して内部ルーターモードで動作し、他のノード種別で実行される QDR は、エッジモードでコントローラー上で実行される内部ルーターに接続されます。

第4章 OPENSTACK PLATFORM の監視

Sensu のスタックインフラストラクチャーについては、https://docs.sensu.io/sensu-core/1.7/overview/architecture/ の Sensu のドキュメントを参照してください。

Red Hat は、**osops-tools-monitoring-oschecks** パッケージで、check スクリプトのセットを提供しています。check スクリプトの大半は、OpenStack コンポーネントへの API 接続のみをチェックします。ただし、特定のスクリプトは、OpenStack Compute (nova)、OpenStack Block Storage (cinder)、OpenStack Image (glance)、OpenStack Networking (neutron) を対象とする追加の OpenStack リソースのテストも実行します。たとえば、OpenStack Identity (keystone) API check は、**keystone** の実行時に以下の結果を返します。

```
OK: Got a token, Keystone API is working.
```

第5章 SENSU クライアントインストールの検証

1. 各オーバークラウドノードで **sensu-client** のステータスを確認します。

```
# podman ps | grep sensu-client
```

2. エラーログ (**/var/log/containers/sensu/sensu-client.log**) で問題があるかどうかを確認します。

3. 各オーバークラウドノードに、監視サーバーの IP アドレスを設定する **/var/lib/config-data/puppet-generated/sensu/etc/sensu/conf.d/rabbitmq.json** ファイルがあることを確認します。

第6章 ノードの状態の確認

Uchiwa ダッシュボードがデプロイされている場合には、そのダッシュボードを Sensu サーバーと共に使用して、ノードの状態を確認することができます。

1. Uchiwa ダッシュボードにログインして、**Data Center** タブをクリックし、データセンターが稼働中であることを確認します。

   ```
   http://<SERVER_IP_ADDRESS>/uchiwa
   ```

2. 全オーバークラウドノードが **Connected** の状態であることを確認します。

3. オーバークラウドノードの1台を適時に再起動し、そのノードのステータスを Uchiwa ダッシュボードで確認します。再起動の完了後には、ノードが Sensu サーバーに正常に再接続されて check の実行を開始するかどうかを確認します。

第7章 OPENSTACK サービスの状態の確認

以下の例では、**openstack-ceilometer-central** サービスの監視をテストします。

1. **openstack-ceilometer-central** サービスが実行中であることを確認します。

```
docker ps -a | grep ceilometer
```

2. Uchiwa ダッシュボードに接続して、正常な **ceilometer** check があり、**ceilometer** JSON ファイルで定義されているとおりに実行されていることを確認します。