Red Hat OpenStack Platform 16.1

Monitoring Tools Configuration Guide

A guide to OpenStack logging and monitoring tools
Abstract

This guide provides information on configuring logging and monitoring for a Red Hat OpenStack Platform environment.
Table of Contents

CHAPTER 1. INTRODUCTION .............................................................................. 3

CHAPTER 2. MONITORING ARCHITECTURE .................................................... 4
  2.1. CENTRALIZED LOGGING ................................................................. 4
  2.2. AVAILABILITY MONITORING .......................................................... 4

CHAPTER 3. INSTALLING THE CLIENT-SIDE TOOLS ....................................... 8
  3.1. SETTING CENTRALIZED LOGGING CLIENT PARAMETERS .................. 8
  3.2. SETTING MONITORING CLIENT PARAMETERS .................................... 8
  3.3. COLLECTING DATA THROUGH AMQ INTERCONNECT ...................... 9
  3.4. COLLECTD PLUGIN CONFIGURATIONS ............................................ 10
    3.4.1. Configuring the pcie_errors plug-in ........................................... 10
    3.4.2. Configuring the mcelog plug-in ................................................ 11
  3.5. YAML FILES .................................................................................. 11

CHAPTER 4. MONITOR THE OPENSTACK PLATFORM ...................................... 12

CHAPTER 5. VALIDATE THE SENSU CLIENT INSTALLATION .......................... 13

CHAPTER 6. REVIEW THE STATE OF A NODE .............................................. 14

CHAPTER 7. REVIEW THE STATE OF AN OPENSTACK SERVICE ..................... 15
CHAPTER 1. INTRODUCTION

Monitoring tools are an optional suite of tools designed to help operators maintain an OpenStack environment. The tools perform the following functions:

- Centralized logging: Allows you gather logs from all components in the OpenStack environment in one central location. You can identify problems across all nodes and services, and optionally, export the log data to Red Hat for assistance in diagnosing problems.

- Availability monitoring: Allows you to monitor all components in the OpenStack environment and determine if any components are currently experiencing outages or are otherwise not functional. You can also configure the system to alert you when problems are identified.
CHAPTER 2. MONITORING ARCHITECTURE

Monitoring tools use a client-server model with the client deployed onto the Red Hat OpenStack Platform overcloud nodes. The Rsyslog service provides client-side centralized logging (CL) and the collectd with enabled sensubility plugin provides client-side availability monitoring (AM).

2.1. CENTRALIZED LOGGING

In your Red Hat OpenStack environment, collecting the logs from all services in one central location simplifies debugging and administration. These logs come from the operating system, such as syslog and audit log files, infrastructure components such as RabbitMQ and MariaDB, and OpenStack services such as Identity, Compute, and others.

The centralized logging toolchain consists of the following components:

- Log Collection Agent (Rsyslog)
- Data Store (Elasticsearch)
- API/Presentation Layer (Kibana)

**NOTE**

Red Hat OpenStack Platform director does not deploy the server-side components for centralized logging. Red Hat does not support the server-side components, including the Elasticsearch database and Kibana.

2.2. AVAILABILITY MONITORING

With availability monitoring, you have one central place to monitor the high-level functionality of all components across your entire OpenStack environment.

The availability monitoring toolchain consists of several components:

- Monitoring Agent (collectd with enabled sensubility plugin)
- Monitoring Relay/Proxy (RabbitMQ)
- Monitoring Controller/Server (Sensu server)
- API/Presentation Layer (Uchiwa)

**NOTE**

Red Hat OpenStack Platform director does not deploy the server-side components for availability monitoring. Red Hat does not support the server-side components, including Uchiwa, Sensu Server, the Sensu API plus RabbitMQ, and a Redis instance running on a monitoring node.

The availability monitoring components and their interactions are laid out in the following diagrams:

**NOTE**

Items shown in blue denote Red Hat-supported components.
Figure 2.1. Availability monitoring architecture at a high level

- Monitoring Relay / Proxy
  - sends/receives checks
  - sends notification
- Monitoring Controller / Server
  - stores data
- Data store
  - reads data from
- API / Presentation
  - views/interacts with system
- OpenStack Operator
  - interacts with system
- Monitoring Agent
- Monitoring Agent
- Monitoring Agent
Figure 2.2. Single-node deployment for Red Hat OpenStack Platform

**MONITOR NODE**
- rabbitmq
- sensu-server
- redis
- sensu-api
- uchiwa

**Controller Nodes**
- sensu-agent

**Compute Nodes**
- sensu-agent

**Storage Nodes**
- sensu-agent

**OpenStack Operator**
views data

reads data from
Figure 2.3. HA deployment for Red Hat OpenStack Platform
CHAPTER 3. INSTALLING THE CLIENT-SIDE TOOLS

Before you deploy the overcloud, you need to determine the configuration settings to apply to each client. Copy the example environment files from the heat template collection and modify the files to suit your environment.

3.1. SETTING CENTRALIZED LOGGING CLIENT PARAMETERS

For more information, see Enabling centralized logging during deployment in the Logging, Monitoring, and Troubleshooting guide.

3.2. SETTING MONITORING CLIENT PARAMETERS

The monitoring solution collects system information periodically and provides a mechanism to store and monitor the values in a variety of ways using a data collecting agent. Red Hat supports collectd as a collection agent. You can use Service Telemetry Framework (STF) to store the data, and in turn, monitor systems, find performance bottlenecks, and predict future system load.

To configure collectd, complete the following steps:

1. Create `config.yaml` in your home directory, for example, `/home/templates/custom`, and configure the `MetricsQdrConnectors` parameter to point to the Service Telemetry Framework server side:

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

2. In the `config.yaml` file, list the plug-ins you want under `CollectdExtraPlugins`. You can also provide parameters in the `ExtraConfig` section. By default, collectd comes with the `cpu`, `df`, `disk`, `hugepages`, `interface`, `load`, `memory`, `processes`, `tcpconns`, `unixsock`, and `uptime` plug-ins. You can add additional plug-ins using the `CollectdExtraPlugins` parameter. You can also provide additional configuration information for the `CollectdExtraPlugins` using the `ExtraConfig` option as shown. For example, to enable the `virt` plug-in, and configure the connection string and the hostname format, use the following syntax:

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

   ExtraConfig:
     collectd::plugin::virt::connection: "qemu:///system"
     collectd::plugin::virt::hostname_format: "hostname uuid"
   ```
3. Optional: To collect metric and event data through AMQ Interconnect, add the line `MetricsQdrExternalEndpoint: true` to the `config.yaml` file:

```yaml
parameter_defaults:
  MetricsQdrExternalEndpoint: true
```

4. Deploy the overcloud. Include `config.yaml`, `collectd-write-qdr.yaml`, and one of the `qdr-*.yaml` files in your `overcloud deploy` command. For example:

```
$ 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. Optional: To enable overcloud RabbitMQ monitoring, include the `collectd-read-rabbitmq.yaml` file in your `overcloud deploy` command.

Additional resources

- For more information about the YAML files, see Section 3.5, "YAML files".
- For more information about collectd plug-ins, see Section 3.4, "Collectd plug-in configurations".
- For more information about Service Telemetry Framework, see the Service Telemetry Framework guide.

### 3.3. COLLECTING DATA THROUGH AMQ INTERCONNECT

To subscribe to the available AMQ Interconnect addresses for metric and event data consumption, create an environment file to expose AMQ Interconnect for client connections, and deploy the overcloud.

**NOTE**

The Service Telemetry Operator simplifies the deployment of all data ingestion and data storage components for single cloud deployments. To share the data storage domain with multiple clouds, see Configuring multiple clouds in the Service Telemetry Framework guide.

**Procedure**

1. Log on to the Red Hat OpenStack Platform undercloud as the `stack` user.

2. Create a configuration file called `data-collection.yaml` in the `/home/stack` directory.

3. To enable external endpoints, add the line `MetricsQdrExternalEndpoint: true` to the `data-collection.yaml` file:
To enable collectd and AMQ Interconnect, add the following files to your Red Hat OpenStack Platform director deployment:

- the `data-collection.yaml` environment file
- the `qdr-form-controller-mesh.yaml` file that enables the client side AMQ Interconnect to connect to the external endpoints

Optional: To collect Ceilometer and collectd events, include `ceilometer-write-qdr.yaml` and `collectd-write-qdr.yaml` file in your `openstack overcloud deploy` command.

6. Deploy the overcloud.

Additional resources
- For more information about the YAML files, see Section 3.5, “YAML files”.

### 3.4. COLLECTD PLUG-IN CONFIGURATIONS

There are multiple configuration possibilities of Red Hat OpenStack Platform director. You can configure multiple collectd plug-ins to suit your environment.

To view a complete list of collectd plug-ins, see collectd plug-ins in the Service Telemetry Framework guide.

#### 3.4.1. Configuring the pcie_errors plug-in

Use the pcie_errors plug-in to poll PCI config space for baseline and Advanced Error Reporting (AER) errors, and to parse syslog for AER events. Errors are reported through notifications.

**Procedure**

- Add the `pcie_errors` plug-in configuration to `config.yaml`:

```yaml
parameter_defaults:
  CollectdExtraPlugins:
    - pcie_errors
```

Additional resources
- For more information about configuring collectd plug-ins, see Setting monitoring client parameters.
3.4.2. Configuring the mcelog plug-in

Use the mcelog plug-in to send notifications and statistics relevant to Machine Check Exceptions when they occur. Configure mcelog to run on the platform in daemon mode and ensure that logging capabilities are enabled.

Procedure

- Add the mcelog plug-in configuration to config.yaml:

```yaml
parameter_defaults:
  CollectdExtraPlugins: mcelog
  CollectdEnableMcelog: true
```

Additional resources

- For more information about configuring collectd plug-ins, see Setting monitoring client parameters.

3.5. YAML FILES

You can include the following YAML files in your overcloud deploy command when you configure collectd:

- **collectd-read-rabbitmq.yaml**: Enables and configures python-collect-rabbitmq to monitor the overcloud RabbitMQ instance.

- **collectd-write-qdr.yaml**: Enables collectd to send telemetry and notification data through AMQ Interconnect.

- **qdr-edge-only.yaml**: Enables deployment of AMQ Interconnect. Each overcloud node has one local qdrouterd service running and operating in edge mode. For example, sending received data straight to defined MetricsQdrConnectors.

- **qdr-form-controller-mesh.yaml**: Enables deployment of AMQ Interconnect. Each overcloud node has one local qdrouterd service forming a mesh topology. For example, AMQ Interconnect routers on controllers operate in interior router mode, with connections to defined MetricsQdrConnectors, and AMQ Interconnect routers on other node types connect in edge mode to the interior routers running on the controllers.
See the Sensu documentation for further details about the Sensu stack infrastructure: https://docs.sensu.io/sensu-core/1.7/overview/architecture/

Red Hat supplies a set of check scripts in the `osops-tools-monitoring-oschecks` package. The majority of the check scripts only check the API connection to the OpenStack component. However, certain scripts also perform additional OpenStack resource tests for OpenStack Compute (nova), OpenStack Block Storage (cinder), OpenStack Image (glance), and OpenStack Networking (neutron). For example, the OpenStack Identity (keystone) API check returns the following result when `keystone` is running:

```
OK: Got a token, Keystone API is working.
```
CHAPTER 5. VALIDATE THE SENSU CLIENT INSTALLATION

1. Check the status of the `sensu-client` on each overcloud node:
   
   ```bash
   # podman ps | grep sensu-client
   ```

2. Review the error log for any issues: `/var/log/containers/sensu/sensu-client.log`

3. Verify that each overcloud node has the `/var/lib/config-data/puppet-generated/sensu/etc/sensu/conf.d/rabbitmq.json` file that sets the IP address of the monitoring server.
CHAPTER 6. REVIEW THE STATE OF A NODE

If you have a deployment of the Uchiwa dashboard, you can use it with the Sensu server to review the state of your nodes:

1. Login to the Uchiwa dashboard and click the **Data Center** tab to confirm that the Data Center is operational.

   http://<SERVER_IP_ADDRESS>/uchiwa

2. Check that all overcloud nodes are in a **Connected** state.

3. At a suitable time, reboot one of the overcloud nodes and review the rebooted node’s status in the Uchiwa dashboard. After the reboot completes, verify that the node successfully re-connects to the Sensu server and starts executing checks.
CHAPTER 7. REVIEW THE STATE OF AN OPENSTACK SERVICE

This example tests the monitoring of the openstack-ceilometer-central service.

1. Confirm that the openstack-ceilometer-central service is running:
   
   ```bash
   docker ps -a | grep ceilometer
   ```

2. Connect to the Uchiwa dashboard and confirm that a successful ceilometer check is present and running as defined in the ceiling JSON file.