



Red Hat OpenStack Platform 16.2

Monitoring Tools Configuration Guide

A guide to OpenStack logging and monitoring tools

Red Hat OpenStack Platform 16.2 Monitoring Tools Configuration Guide

A guide to OpenStack logging and monitoring tools

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2021 Red Hat, Inc.

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

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

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

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

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, 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.

Abstract

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

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	3
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
CHAPTER 1. INTRODUCTION TO RED HAT OPENSTACK PLATFORM MONITORING TOOLS	5
1.1. SUPPORT STATUS OF MONITORING COMPONENTS	5
CHAPTER 2. MONITORING ARCHITECTURE	6
2.1. CENTRALIZED LOGGING	6
2.2. AVAILABILITY MONITORING	6
CHAPTER 3. INSTALLING THE CLIENT-SIDE TOOLS	10
3.1. SETTING CENTRALIZED LOGGING CLIENT PARAMETERS	10
3.2. SETTING MONITORING CLIENT PARAMETERS	10
3.3. COLLECTING DATA THROUGH AMQ INTERCONNECT	12
3.4. COLLECTD PLUG-IN CONFIGURATIONS	13
3.4.1. amqp1	13
3.4.2. cpu	14
3.4.3. ovs_stats	15
3.4.4. mcelog	15
3.4.5. pcie_errors	16
3.4.6. virt	16
3.4.7. write_http	17
3.5. YAML FILES	18

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. INTRODUCTION TO RED HAT OPENSTACK PLATFORM MONITORING TOOLS

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.

1.1. SUPPORT STATUS OF MONITORING COMPONENTS

Use this table to view the support status of monitoring components in Red Hat OpenStack Platform.

Table 1.1. Support status

Component	Fully supported since	Deprecated in	Removed since	Note
Aodh	OSP 9	OSP 15		Used for auto scaling in 16.1
Ceilometer	OSP 4			Used for auto scaling in 16.1
Collectd	OSP 11			
Gnocchi	OSP 9	OSP 15		
Panko	OSP 11	OSP 12, not installed by default since OSP 14	OSP 16.1	Required for Cloudforms until 16.1
osops-tools-monitoring-oscheck		OSP 14		

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.

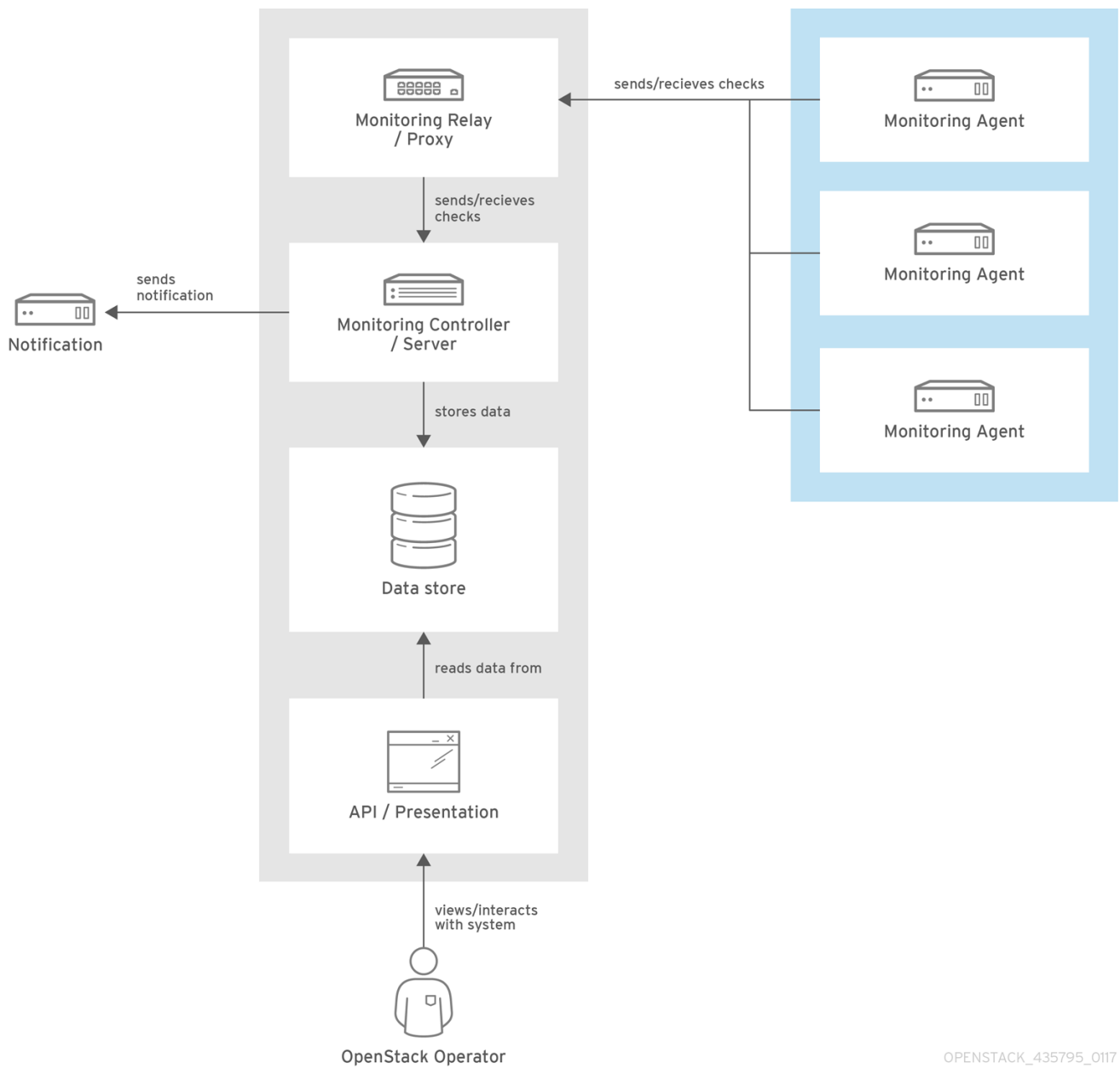
The availability monitoring components and their interactions are laid out in the following diagrams:



NOTE

Items shown in blue denote Red Hat-supported components.

Figure 2.1. Availability monitoring architecture at a high level



OPENSTACK_435795_0117

Figure 2.2. Single-node deployment for Red Hat OpenStack Platform

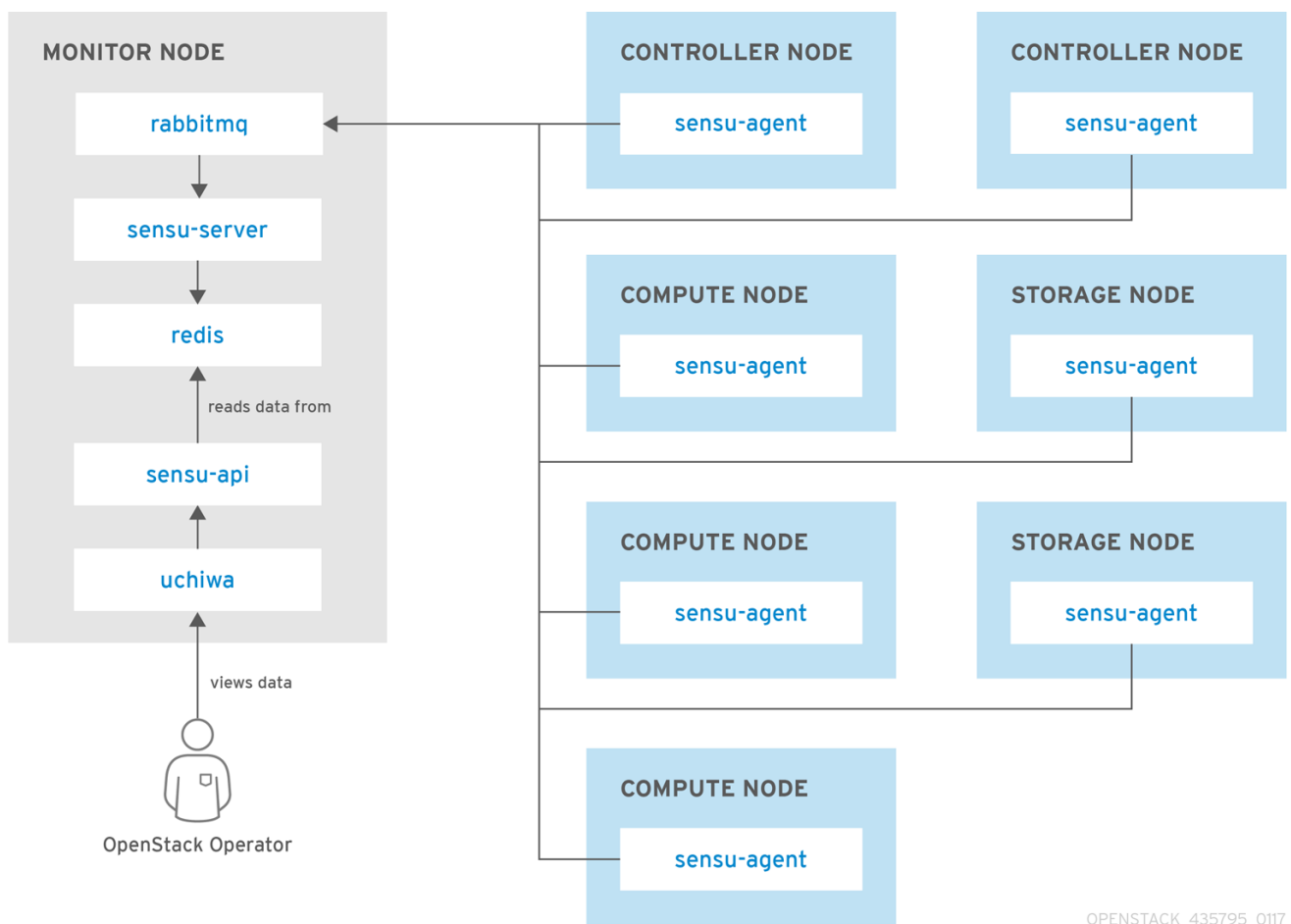
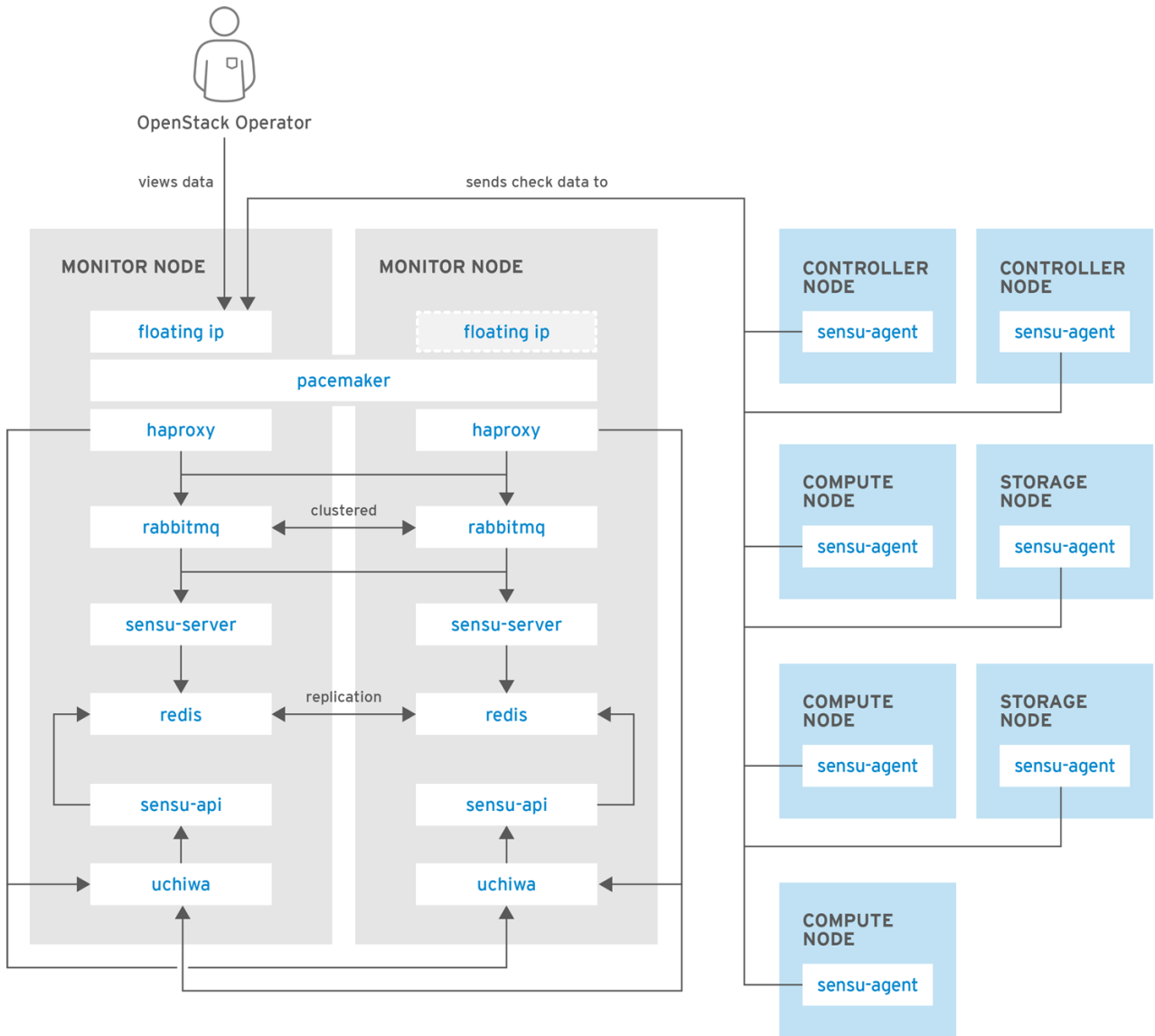


Figure 2.3. HA deployment for Red Hat OpenStack Platform



OPENSTACK_435795_017

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. Collectd-sensubility is an extension of collectd and communicates with Sensu server side through RabbitMQ. You can use Service Telemetry Framework (STF) to store the data, and in turn, monitor systems, find performance bottlenecks, and predict future system load. For more information about Service Telemetry Framework, see the [Service Telemetry Framework](#) guide.

To configure collectd and collectd-sensubility, complete the following steps:

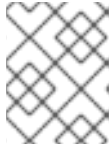
1. Create **config.yaml** in your home directory, for example, `/home/templates/custom`, and configure the **MetricsQdrConnectors** parameter to point to STF server side:

```
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 to use 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. For example, to enable the **virt** plug-in, and configure the connection string and the hostname format, use the following syntax:

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

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

**NOTE**

Do not remove the **unixsock** plug-in. Removal results in the permanent marking of the collectd container as unhealthy.

- Optional: To collect metric and event data through AMQ Interconnect, add the line **MetricsQdrExternalEndpoint: true** to the **config.yaml** file:

```
parameter_defaults:
  MetricsQdrExternalEndpoint: true
```

- To enable collectd-sensubility, add the following environment configuration to the **config.yaml** file:

```
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
```

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

```
$ 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
```

6. Optional: To enable overcloud RabbitMQ monitoring, include the **collectd-read-rabbitmq.yaml** file in the **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.



WARNING

It is not possible to switch between QDR mesh mode and QDR edge mode, as used by the Service Telemetry Framework (STF). Additionally, it is not possible to use QDR mesh mode if you enable data collection for STF.

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 **MetricsQdrExternalEndpoint: true** parameter to the **data-collection.yaml** file:

```
parameter_defaults:
  MetricsQdrExternalEndpoint: true
```

4. 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

```
openstack overcloud deploy <other arguments>
  --templates /usr/share/openstack-tripleo-heat-templates \
  --environment-file <...other-environment-files...> \
  --environment-file /usr/share/openstack-tripleo-heat-
  templates/environments/metrics/qdr-form-controller-mesh.yaml \
  --environment-file /home/stack/data-collection.yaml
```

5. Optional: To collect Ceilometer and collectd events, include **ceilometer-write-qdr.yaml** and **collectd-write-qdr.yaml** file in your **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 many configuration possibilities of Red Hat OpenStack Platform director. You can configure multiple collectd plug-ins to suit your environment. Each documented plug-in has a description and example configuration. Some plug-ins have a table of metrics that you can query for from Grafana or Prometheus, and a list of options you can configure, if available.

Additional resources

To view a complete list of collectd plug-in options, see [collectd plug-ins](#) in the *Service Telemetry Framework* guide.

3.4.1. amqp1

The amqp1 plug-in writes values to an amqp1 message bus, for example AMQ Interconnect.

Example configuration

```
Parameter_defaults:
```

```
CollectdExtraPlugins:
- amqp1
ExtraConfig:
collectd::plugin::amqp1::send_queue_limit: 50
```

3.4.2. cpu

Use the **cpu** plug-in to monitor the amount of time spent by the CPU in various states, for example, executing user code, executing system code, waiting for IO-operations, and being idle. The **cpu** plug-in does not collect percentages. It collects *jiffies*, which are units of scheduling. On many Linux systems, there are approximately 100 jiffies in one second, but this does not mean that you get a percentage value. Depending on system load, hardware, whether or not the system is virtualized, and other factors, there can be more or less than 100 jiffies in one second. There is no guarantee that all states add up to 100, which is a requirement for percentages.

Table 3.1. cpu metrics

Name	Description	Query
idle	Amount of idle time	collectd_cpu_total{...,type_instance=idle}
interrupt	CPU blocked by interrupts	collectd_cpu_total{...,type_instance=interrupt}
nice	Amount of time running low priority processes	collectd_cpu_total{...,type_instance=nice}
softirq	Amount of cycles spent in servicing interrupt requests	collectd_cpu_total{...,type_instance=waitirq}
steal	The percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor	collectd_cpu_total{...,type_instance=steal}
system	Amount spent on system level (kernel)	collectd_cpu_total{...,type_instance=system}
user	Jiffies used by user processes	collectd_cpu_total{...,type_instance=user}
wait	CPU waiting on outstanding I/O request	collectd_cpu_total{...,type_instance=wait}

Options

- collectd::plugin::cpu::reportbystate
- collectd::plugin::cpu::valuespercentage
- collectd::plugin::cpu::reportbycpu
- collectd::plugin::cpu::reportnumcpu

- `collectd::plugin::cpu::reportgueststate`
- `collectd::plugin::cpu::subtractgueststate`
- `collectd::plugin::cpu::interval`

Example configuration

```
parameter_defaults:
  CollectdExtraPlugins:
    - cpu
  ExtraConfig:
    collectd::plugin::cpu::reportbystate: true
```

Additional resources

For more information about configuring the **cpu** plug-in, see [cpu plug-in](#).

3.4.3. ovs_stats

Use the **ovs_stats** plug-in to collect statistics of OVS connected interfaces. This plug-in uses the OVSDDB management protocol (RFC7047) monitor mechanism to get statistics from OVSDDB.

Options

- `collectd::plugin::ovs_stats::address`
- `collectd::plugin::ovs_stats::bridges`
- `collectd::plugin::ovs_stats::port`
- `collectd::plugin::ovs_stats::socket`

Example configuration

```
parameter_defaults:
  CollectdExtraPlugins:
    - ovs_stats
  ExtraConfig:
    collectd::plugin::ovs_stats:
```

Additional resources

- For more information about configuring the **ovs_stats** plug-in, see [ovs_stats upstream](#).

3.4.4. mcelog

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.

Example configuration

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

Additional resources

- For more information about configuring the **mcelog** plug-in, see [mcelog upstream](#).

3.4.5. pcie_errors

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.

Options

- `collectd::plugin::pcie_errors::reportbystate`
- `collectd::plugin::pcie_errors::source`
- `collectd::plugin::pcie_errors::access`
- `collectd::plugin::pcie_errors::reportmasked`
- `collectd::plugin::pcie_errors::persistentnotifications`

Example configuration

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

Additional resources

- For more information about configuring the **pcie_errors** plug-in, see [pcie_errors upstream](#).

3.4.6. virt

Use the **virt** plug-in to collect CPU, disk, network load and other metrics for virtual machines on the host. Metrics are collected through the **libvirt** API.

Options

- `collectd::plugin::virt::connection`
- `collectd::plugin::virt::refresh_interval`
- `collectd::plugin::virt::domain`
- `collectd::plugin::virt::block_device`
- `collectd::plugin::virt::interface_device`
- `Collectd::plugin::virt::ignore_selected`

- `collectd::plugin::virt::plugin_instance_format`
- `collectd::plugin::virt::hostname_format`
- `collectd::plugin::virt::interface_format`
- `collectd::plugin::virt::extra_stats`
- `collectd::plugin::virt::interval`

Example configuration

```
ExtraConfig:
  collectd::plugin::virt::plugin_instance_format: name
```

Additional resources

For more information about configuring the **virt** plug-in, see https://collectd.org/documentation/manpages/collectd.conf.5.shtml#plugin_virt

3.4.7. write_http

Use the **write_http** output plug-in to submit values to an HTTP server by using POST requests and encoding metrics with JSON or using the **PUTVAL** command.

Options

- `collectd::plugin::write_http::url`
- `collectd::plugin::write_http::username`
- `collectd::plugin::write_http::password`
- `collectd::plugin::write_http::verifypeer`
- `collectd::plugin::write_http::verifyhost`
- `collectd::plugin::write_http::cacert`
- `collectd::plugin::write_http::capath`
- `collectd::plugin::write_http::clientkey`
- `collectd::plugin::write_http::clientcert`
- `collectd::plugin::write_http::clientkeypass`
- `collectd::plugin::write_http::header`
- `collectd::plugin::write_http::sslversion`
- `collectd::plugin::write_http::format`
- `collectd::plugin::write_http::attribute`
- `collectd::plugin::write_http::ttl`

- `collectd::plugin::write_http::prefix`
- `collectd::plugin::write_http::metrics`
- `collectd::plugin::write_http::notifications`
- `collectd::plugin::write_http::storerates`
- `collectd::plugin::write_http::buffersize`
- `collectd::plugin::write_http::lowspeedlimit`
- `collectd::plugin::write_http::timeout`
- `collectd::plugin::write_http::loghttperror`

Example configuration

```
parameter_defaults:
  CollectdExtraPlugins:
    - write_http
  ExtraConfig:
    collectd::plugin::write_http::nodes:
      collectd:
        url: "http://collectd.tld.org/collectd"
        metrics: true
        header: "X-Custom-Header: custom_value"
```

Additional resources

- For more information about configuring the **write_http** plug-in, see [write_http upstream](#).

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.

Additional resources

For more information about configuring collectd, see [Section 3.2, “Setting monitoring client parameters”](#).