



# Red Hat OpenStack Platform 16.2

## Operational Measurements

Tracking physical and virtual resources, and collecting metrics



# Red Hat OpenStack Platform 16.2 Operational Measurements

---

Tracking physical and virtual resources, and collecting metrics

OpenStack Team  
rhos-docs@redhat.com

## Legal Notice

Copyright © 2023 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> 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<sup>®</sup> 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

Use operational tools to help you measure and maintain your Red Hat OpenStack Platform environment.

## Table of Contents

<b>MAKING OPEN SOURCE MORE INCLUSIVE</b> .....	<b>4</b>
<b>PROVIDING FEEDBACK ON RED HAT DOCUMENTATION</b> .....	<b>5</b>
<b>CHAPTER 1. INTRODUCTION TO OPERATIONAL MEASUREMENTS</b> .....	<b>6</b>
1.1. TELEMETRY ARCHITECTURE	6
1.1.1. Support status of monitoring components	7
1.2. DATA COLLECTION IN RED HAT OPENSTACK PLATFORM	8
1.2.1. Ceilometer	8
1.2.1.1. Publishers	9
Configuring publisher parameters	9
1.2.2. collectd	10
1.3. STORAGE WITH GNOCCHI	10
1.3.1. Archive policies: Storing both short and long-term data in a time-series database	10
1.3.2. Indexer driver	11
1.3.3. Gnocchi Metric-as-a-Service terminology	11
1.4. DISPLAYING METRIC DATA	12
1.4.1. Using and connecting Grafana to display data	12
<b>CHAPTER 2. PLANNING FOR OPERATIONAL MEASUREMENTS</b> .....	<b>13</b>
2.1. CEILOMETER MEASUREMENTS	13
2.2. COLLECTD MEASUREMENTS	13
2.3. PLANNING FOR DATA STORAGE	13
2.4. PLANNING AND MANAGING ARCHIVE POLICIES	14
2.4.1. Metrics	14
2.4.1.1. Creating a metric	15
2.4.2. Creating custom measures	15
2.4.3. Default archive policies	15
2.4.4. Calculating the size of a time-series aggregate	16
2.4.5. Metricd workers	17
2.4.6. Creating an archive policy	17
2.4.7. Managing archive policies	17
2.4.8. Creating an archive policy rule	17
2.5. VERIFYING THE RED HAT OPENSTACK PLATFORM DEPLOYMENT	17
<b>CHAPTER 3. MANAGING ALARMS</b> .....	<b>19</b>
3.1. VIEWING EXISTING ALARMS	19
3.2. CREATING AN ALARM	20
3.3. EDITING AN ALARM	21
3.4. DISABLING AN ALARM	21
3.5. DELETING AN ALARM	22
3.6. EXAMPLE: MONITORING THE DISK ACTIVITY OF INSTANCES	22
3.7. EXAMPLE: MONITORING CPU USE	23
3.8. VIEWING ALARM HISTORY	26
<b>CHAPTER 4. INSTALLING AND CONFIGURING THE LOGS SERVICE</b> .....	<b>27</b>
4.1. THE CENTRALIZED LOG SYSTEM ARCHITECTURE AND COMPONENTS	27
4.2. ENABLING CENTRALIZED LOGGING WITH ELASTICSEARCH	27
4.3. CONFIGURING LOGGING FEATURES	28
4.3.1. Configurable logging parameters	28
4.4. OVERRIDING THE DEFAULT PATH FOR A LOG FILE	29
4.5. MODIFYING THE FORMAT OF A LOG RECORD	30

4.6. TESTING THE CONNECTION BETWEEN RSYSLOG AND ELASTICSEARCH	30
4.7. SERVER-SIDE LOGGING	30
4.8. TRACEBACKS	30
4.9. LOCATION OF LOG FILES FOR OPENSTACK SERVICES	31
4.9.1. Bare Metal Provisioning (ironic) log files	31
4.9.2. Block Storage (cinder) log files	31
4.9.3. Compute (nova) log files	32
4.9.4. Dashboard (horizon) log files	32
4.9.5. Identity Service (keystone) log files	33
4.9.6. Image Service (glance) log files	33
4.9.7. Networking (neutron) log files	33
4.9.8. Object Storage (swift) log files	34
4.9.9. Orchestration (heat) log files	35
4.9.10. Shared Filesystem Service (manila) log files	35
4.9.11. Telemetry (ceilometer) log files	35
4.9.12. Log files for supporting services	36
4.9.13. aodh (alarming service) log files	36
4.9.14. gnocchi (metric storage) log files	36
<b>CHAPTER 5. COLLECTD PLUGINS</b> .....	<b>37</b>
collectd::plugin::aggregation	37
collectd::plugin::amqp1	38
collectd::plugin::apache	39
collectd::plugin::battery	41
collectd::plugin::bind	41
collectd::plugin::ceph	42
collectd::plugins::cgroups	43
collectd::plugin::connectivity	43
collectd::plugin::contrack	44
collectd::plugin::contextswitch	44
collectd::plugin::cpu	44
collectd::plugin::cpufreq	46
collectd::plugin::csv	46
collectd::plugin::df	46
collectd::plugin::disk	47
collectd::plugin::hugepages	48
collectd::plugin::interface	49
collectd::plugin::load	49
collectd::plugin::mcelog	50
collectd::plugin::memcached	50
collectd::plugin::memory	51
collectd::plugin::ntpd	51
collectd::plugin::ovs_stats	52
collectd::plugin::processes	53
collectd::plugin::smart	53
collectd::plugin::swap	54
collectd::plugin::tcpconns	54
collectd::plugin::thermal	55
collectd::plugin::uptime	55
collectd::plugin::virt	56
collectd::plugin::vmem	56
collectd::plugin::write_http	57
collectd::plugin::write_kafka	57



## 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.

## Providing documentation feedback in Jira

Use the [Create Issue](#) form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: [Create Issue](#)
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

# CHAPTER 1. INTRODUCTION TO OPERATIONAL MEASUREMENTS

You can use the components of the Telemetry service in the Red Hat OpenStack Platform (RHOSP) environment to track physical and virtual resources, and collect metrics such as CPU usage and resource availability in your deployment with a data collection daemon that stores the aggregates on a Gnocchi back end.

You can use the availability and performance monitoring tools to measure and maintain your RHOSP environment. These tools perform the following functions:

## Availability monitoring

Monitors all components in the RHOSP environment and determines if any components are currently experiencing outages or are not functional. You can also configure the system to alert you when problems are identified.

## Performance monitoring

Collects system information periodically and provides a mechanism to store and monitor the values by using a data collecting daemon. This daemon stores the data it collects, for example operating system and log files. It also makes the data available over the network. You can use statistics gathered from the data to monitor systems, find performance bottlenecks, and predict future system load.

## 1.1. TELEMETRY ARCHITECTURE

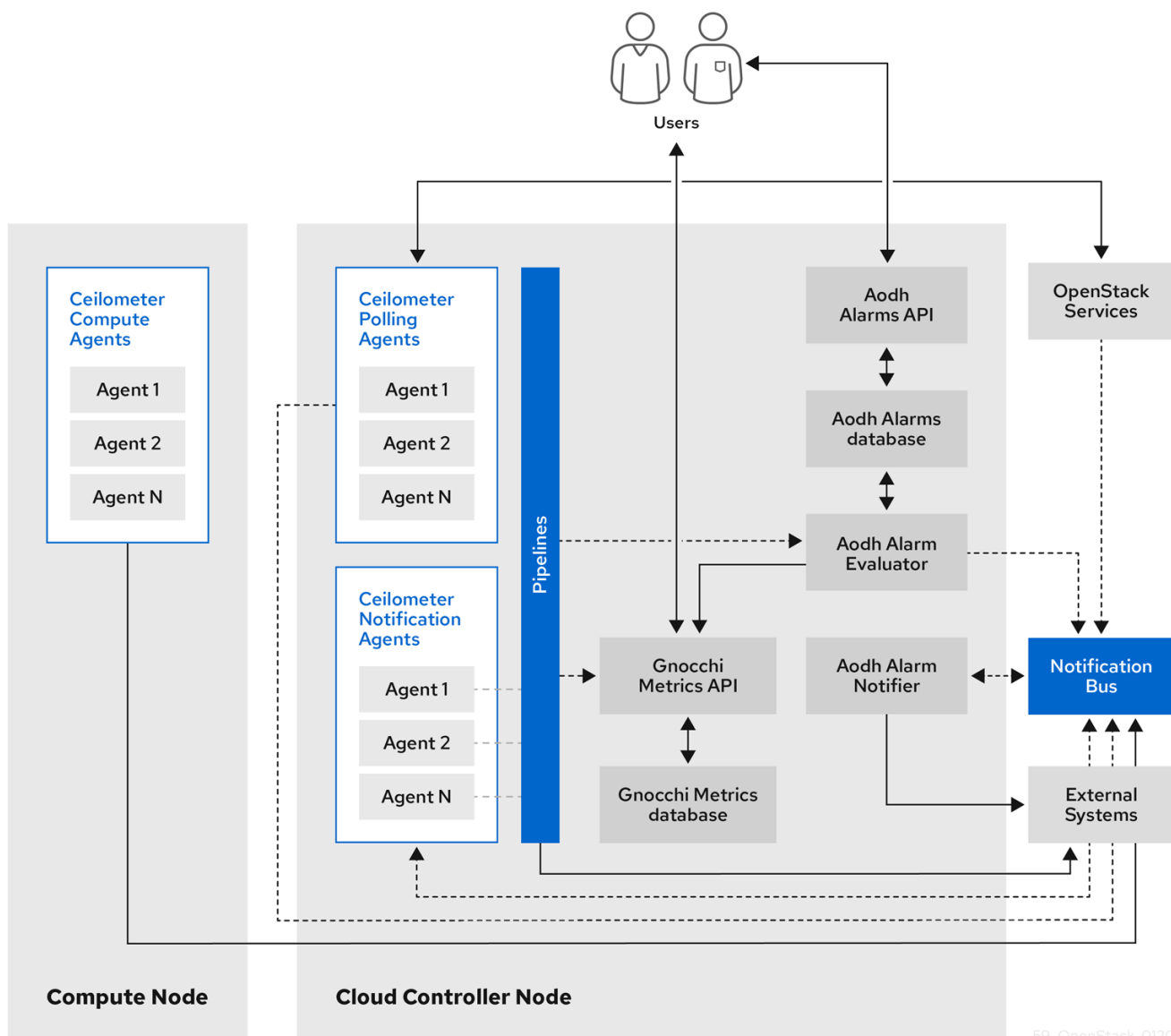
Red Hat OpenStack Platform (RHOSP) Telemetry provides user-level usage data for OpenStack-based clouds. You can use the data for customer billing, system monitoring, or alerts. You can configure Telemetry components to collect data from notifications sent by existing RHOSP components, such as Compute usage events, or by polling RHOSP infrastructure resources, such as libvirt. Telemetry publishes collected data to various targets including data stores and message queues.

Telemetry consists of the following components:

- **Data collection:** Telemetry uses Ceilometer to gather metric and event data. For more information, see [Section 1.2.1, "Ceilometer"](#).
- **Storage:** Telemetry stores metric data in Gnocchi and event data in Panko. For more information, see [Section 1.3, "Storage with Gnocchi"](#).
- **Alarm service:** Telemetry uses the Alarming service (Aodh) to trigger actions based on defined rules against metric or event data collected by Ceilometer.

After you collect the data, you can use a third-party tool to display and analyze metric data, and you can use the Alarming service to configure alarms for events.

Figure 1.1. Telemetry architecture



### 1.1.1. Support status of monitoring components

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

Table 1.1. Support status

Component	Fully supported since	Deprecated in	Removed since	Note
Aodh	RHOSP 9	RHOSP 15		Supported for the autoscaling use case.

Component	Fully supported since	Deprecated in	Removed since	Note
Ceilometer	RHOSP 4			Supported for collection of metrics and events for RHOSP in the autoscaling and Service Telemetry Framework (STF) use cases.
Collectd	RHOSP 11	RHOSP 17.1		Supported for collection of infrastructure metrics for STF.
Gnocchi	RHOSP 9	RHOSP 15		Supported for storage of metrics for the autoscaling use case.
Panko	RHOSP 11	RHOSP 12, not installed by default since RHOSP 14	RHOSP 17.0	
QDR	RHOSP 13	RHOSP 17.1		Supported for transmission of metrics and events data from RHOSP to STF.

## 1.2. DATA COLLECTION IN RED HAT OPENSTACK PLATFORM

Red Hat OpenStack Platform (RHOSP) supports two types of data collection:

- Ceilometer for the OpenStack component-level monitoring. For more information, see [Section 1.2.1, "Ceilometer"](#).
- collectd for infrastructure monitoring. For more information, see [Section 1.2.2, "collectd"](#).

### 1.2.1. Ceilometer

Ceilometer is the default data collection component of the OpenStack Telemetry service that provides the ability to normalize and transform data across all of the current OpenStack core components. Ceilometer collects metering and event data relating to OpenStack services. Collected data is accessible to users based on the deployment configuration.

The Ceilometer service uses three agents to collect data from Red Hat OpenStack Platform (RHOSP) components:

- **A compute agent (ceilometer-agent-compute):** Runs on each Compute node and polls for resource utilization statistics. This agent is the same as polling agent **ceilometer-polling** running with parameter **--polling namespace-compute**.
- **A central agent (ceilometer-agent-central):** Runs on a central management server to poll for resource utilization statistics for resources that are not tied to instances or Compute nodes. You can start multiple agents to scale services horizontally. This is the same as the polling agent **ceilometer-polling** that operates with the parameter **--polling namespace-central**.
- **A notification agent (ceilometer-agent-notification):** Runs on a central management server and consumes messages from the message queues to build event and metering data. Data publishes to defined targets. Gnocchi is the default target. These services use the RHOSP notification bus to communicate.

The Ceilometer agents use publishers to send data to the corresponding end points, for example Gnocchi. You can configure this information in the **pipeline.yaml** file.

### Additional resources

- For more information about publishers, see [Section 1.2.1.1, “Publishers”](#).

#### 1.2.1.1. Publishers

The Telemetry service provides several transport methods to transfer the data collected to an external system. The consumers of this data are different, for example, monitoring systems, for which data loss is acceptable, and billing systems, which require reliable data transportation. Telemetry provides methods to fulfil the requirements of both system types. You can use the publisher component of the service to save the data into persistent storage through the message bus or to send it to one or more external consumers. One chain can contain multiple publishers.

The following publisher types are supported:

- **Gnocchi (default):** When the Gnocchi publisher is enabled, measurement and resource information is pushed to Gnocchi for time-series optimized storage. Ensure that you register Gnocchi in the Identity service as Ceilometer discovers the exact path through the Identity service.
- **Panko:** You can store event data from Ceilometer in panko, which provides an HTTP REST interface to query system events in Red Hat OpenStack Platform. Panko has been deprecated.

### Configuring publisher parameters

You can configure the multi-publisher for each data point within the Telemetry service, allowing the same technical meter or event to be published multiple times to multiple destinations, each potentially using a different transport method.

### Procedure

1. Create a YAML file to describe the possible publisher parameters and defaults, for example, **ceilometer-publisher.yaml**. Insert the following parameters in **parameter\_defaults**:

```
parameter_defaults:
  ManagePipeline: true
  ManageEventPipeline: true
  EventPipelinePublishers:
```

```
- gnocchi://?archive_policy=high
PipelinePublishers:
- gnocchi://?archive_policy=high
```

2. Deploy the overcloud. Include the modified YAML files in the **openstack overcloud deploy** command. In the following example, replace **<environment\_files>** with the other YAML files that you want to include in the deployment:

```
$ openstack overcloud deploy
--templates \
-e /home/custom/ceilometer-publisher.yaml
-e <environment_files>
```

### Additional resources

- For more information about parameters, see [Telemetry parameters](#) in the *Overcloud Parameters* guide and [Heat parameters](#) in the *Advanced Overcloud Customization* guide.

## 1.2.2. collectd

Performance monitoring 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 the collectd daemon as a data collection agent. This daemon stores the data in a time-series database. One of the Red Hat supported databases is called Gnocchi. You can use this stored data to monitor systems, find performance bottlenecks, and predict future system load.

### Additional resources

- For more information about Gnocchi, see [Section 1.3, "Storage with Gnocchi"](#).

## 1.3. STORAGE WITH GNOCCHI

Gnocchi is an open-source time-series database. It stores metrics at a very large scale and provides access to metrics and resources to operators and users. Gnocchi uses an archive policy to define which aggregations to compute and how many aggregates to retain; and an indexer driver to store the index of all resources, archive policies, and metrics.

### 1.3.1. Archive policies: Storing both short and long-term data in a time-series database

An archive policy defines which aggregations to compute and how many aggregates to retain. Gnocchi supports different aggregation methods, such as minimum, maximum, average, Nth percentile, and standard deviation. These aggregations are computed over a period of time called granularity and retained for a specific timespan.

The archive policy defines how the metrics are aggregated and for how long they are stored. Each archive policy is defined as the number of points over a timespan.

For example, if your archive policy defines a policy of 10 points with a granularity of 1 second, the time-series archive keeps up to 10 seconds, each representing an aggregation over 1 second. This means that the time series, at a maximum, retains 10 seconds of data between the more recent point and the older point.

The archive policy also defines which aggregate methods are used. The default is set to the parameter **default\_aggregation\_methods** whose values by default are set to mean, min, max, sum, std, count. So, depending on the use case, the archive policy and the granularity vary.

### Additional resources

- For more information about archive policies, see Planning and managing archive policies.

### 1.3.2. Indexer driver

The indexer is responsible for storing the index of all resources, archive policies, and metrics along with their definitions, types, and properties. It is also responsible for linking resources with metrics. Red Hat OpenStack Platform director installs the indexer driver by default. You need a database to index all the resources and metrics that Gnocchi handles. The supported driver is MySQL.

### 1.3.3. Gnocchi Metric-as-a-Service terminology

This table contains definitions of the commonly used terms for the Metric-as-a-Service features.

**Table 1.2. Metric-as-a-Service terminology**

Term	Definition
Aggregation method	A function used to aggregate multiple measures into an aggregate. For example, the min aggregation method aggregates the values of different measures to the minimum value of all the measures in the time range.
Aggregate	A data point tuple generated from several measures according to the archive policy. An aggregate is composed of a timestamp and a value.
Archive policy	An aggregate storage policy attached to a metric. An archive policy determines how long aggregates are kept in a metric and how aggregates are aggregated (the aggregation method).
Granularity	The time between two aggregates in an aggregated time series of a metric.
Measure	An incoming data point tuple sent to the Time series database by the API. A measure is composed of a timestamp and a value.
Metric	An entity storing aggregates identified by an UUID. A metric can be attached to a resource using a name. How a metric stores its aggregates is defined by the archive policy to which the metric is associated.
Resource	An entity representing anything in your infrastructure that you associate a metric with. A resource is identified by a unique ID and can contain attributes.
Time series	A list of aggregates ordered by time.
Timespan	The time period for which a metric keeps its aggregates. It is used in the context of archive policy.

## 1.4. DISPLAYING METRIC DATA

You can use the following tools to display and analyze metric data:

- **Grafana:** An open source metric analytics and visualization suite. Grafana is most commonly used for visualizing time series data for infrastructure and application analytics.
- **Red Hat CloudForms:** An infrastructure management platform that IT departments use to control the self-service abilities of users to provision, manage, and ensure compliance across virtual machines and private clouds.

### Additional resources

- For more information about Grafana, see [Section 1.4.1, “Using and connecting Grafana to display data”](#).
- For more information about Red Hat Cloudforms, see the [product documentation](#).

### 1.4.1. Using and connecting Grafana to display data

You can use third-party software, for example Grafana, to view a graphical representation of the metrics gathered and stored.

Grafana is an open source metric analytics, monitoring, and visualization suite. To install and configure Grafana, see the official [Grafana documentation](#).



## CHAPTER 2. PLANNING FOR OPERATIONAL MEASUREMENTS

The resources that you monitor depend on your business requirements. You can use Ceilometer or collectd to monitor your resources.

- For more information on collectd measurements, see [Section 2.2, “Collectd measurements”](#).
- For more information on Ceilometer measurements, see [Section 2.1, “Ceilometer measurements”](#).

### 2.1. CEILOMETER MEASUREMENTS

For a full list of Ceilometer measures, see

<https://docs.openstack.org/ceilometer/train/admin/telemetry-measurements.html>

### 2.2. COLLECTD MEASUREMENTS

The following measurements are useful collectd metrics:

- disk
- interface
- load
- memory
- processes
- tcpconns

### 2.3. PLANNING FOR DATA STORAGE

Gnocchi stores a collection of data points, where each data point is an aggregate. The storage format is compressed using different techniques. As a result, to calculate the size of a time-series database, you must estimate the size based on the worst-case scenario.



#### WARNING

The use of Red Hat OpenStack Platform (RHOSP) Object Storage (swift) for time series database (Gnocchi) storage is only supported for small and non-production environments.

#### Procedure

1. Calculate the number of data points:  
number of points = timespan / granularity

For example, if you want to retain a year of data with one-minute resolution, use the formula:

$$\text{number of data points} = (365 \text{ days} \times 24 \text{ hours} \times 60 \text{ minutes}) / 1 \text{ minute} \\ \text{number of data points} = 525600$$

2. Calculate the size of the time-series database:  
size in bytes = number of data points X 8 bytes

If you apply this formula to the example, the result is 4.1 MB:

$$\text{size in bytes} = 525600 \text{ points} \times 8 \text{ bytes} = 4204800 \text{ bytes} = 4.1 \text{ MB}$$

This value is an estimated storage requirement for a single aggregated time-series database. If your archive policy uses multiple aggregation methods (min, max, mean, sum, std, count), multiply this value by the number of aggregation methods you use.

### Additional resources

- [Section 1.3.1, "Archive policies: Storing both short and long-term data in a time-series database"](#)
- [Section 2.4, "Planning and managing archive policies"](#)

## 2.4. PLANNING AND MANAGING ARCHIVE POLICIES

An archive policy defines how you aggregate the metrics and for how long you store the metrics in the time-series database. An archive policy is defined as the number of points over a timespan.

If your archive policy defines a policy of 10 points with a granularity of 1 second, the time series archive keeps up to 10 seconds, each representing an aggregation over 1 second. This means that the time series retains, at a maximum, 10 seconds of data between the more recent point and the older point. The archive policy also defines the aggregate method to use. The default is set to the parameter **default\_aggregation\_methods**, where the default values are set to **mean, min, max, sum, std, count**. So, depending on the use case, the archive policy and the granularity can vary.

To plan an archive policy, ensure that you are familiar with the following concepts:

- Metrics. For more information, see [Section 2.4.1, "Metrics"](#).
- Measures. For more information, see [Section 2.4.2, "Creating custom measures"](#).
- Aggregation. For more information, see [Section 2.4.4, "Calculating the size of a time-series aggregate"](#).
- Metricd workers. For more information, see [Section 2.4.5, "Metricd workers"](#).

To create and manage an archive policy, complete the following tasks:

1. Create an archive policy. For more information, see [Section 2.4.6, "Creating an archive policy"](#).
2. Manage an archive policy. For more information, see [Section 2.4.7, "Managing archive policies"](#).
3. Create an archive policy rule. For more information, see [Section 2.4.8, "Creating an archive policy rule"](#).

### 2.4.1. Metrics

Gnocchi provides an object type called *metric*. A metric is anything that you can measure, for example, the CPU usage of a server, the temperature of a room, or the number of bytes sent by a network interface. A metric has the following properties:

- A UUID to identify it
- A name
- The archive policy used to store and aggregate the measures

### Additional resources

- For terminology definitions, see [Gnocchi Metric-as-a-Service terminology](#).

#### 2.4.1.1. Creating a metric

##### Procedure

1. Create a resource. Replace `<resource_name>` with the name of the resource:

```
$ openstack metric resource create <resource_name>
```

2. Create the metric. Replace `<resource_name>` with the name of the resource and `<metric_name>` with the name of the metric:

```
$ openstack metric metric create -r <resource_name> <metric_name>
```

When you create the metric, the archive policy attribute is fixed and unchangeable. You can change the definition of the archive policy through the **archive\_policy** endpoint.

#### 2.4.2. Creating custom measures

A measure is an incoming datapoint tuple that the API sends to Gnocchi. It is composed of a timestamp and a value. You can create your own custom measures.

##### Procedure

- Create a custom measure:

```
$ openstack metric measures add -m <MEASURE1> -m <MEASURE2> .. -r
<RESOURCE_NAME> <METRIC_NAME>
```

#### 2.4.3. Default archive policies

By default, Gnocchi has the following archive policies:

- low
  - 5 minutes granularity over 30 days
  - aggregation methods used: **default\_aggregation\_methods**
  - maximum estimated size per metric: 406 KiB

- medium
  - 1 minute granularity over 7 days
  - 1 hour granularity over 365 days
  - aggregation methods used: **default\_aggregation\_methods**
  - maximum estimated size per metric: 887 KiB
- high
  - 1 second granularity over 1 hour
  - 1 minute granularity over 1 week
  - 1 hour granularity over 1 year
  - aggregation methods used: **default\_aggregation\_methods**
  - maximum estimated size per metric: 1 057 KiB
- bool
  - 1 second granularity over 1 year
  - aggregation methods used: last
  - maximum optimistic size per metric: 1 539 KiB
  - maximum pessimistic size per metric: 277 172 KiB

#### 2.4.4. Calculating the size of a time-series aggregate

Gnocchi stores a collection of data points, where each point is an aggregate. The storage format is compressed using different techniques. As a result, calculating the size of a time-series is estimated based on a worst case scenario, as shown in the following example.

##### Procedure

1. Use this formula to calculate the number of points:

number of points = timespan / granularity

For example, if you want to keep a year of data with one-minute resolution:

number of points = (365 days X 24 hours X 60 minutes) / 1 minute

number of points = 525600

2. To calculate the point size in bytes, use this formula:

size in bytes = number of points X 8 bytes

size in bytes = 525600 points X 8 bytes = 4204800 bytes = 4.1 MB

This value is an estimated storage requirement for a single aggregated time-series. If your archive policy uses multiple aggregation methods - min, max, mean, sum, std, count - multiply this value by the number of aggregation methods you use.

### 2.4.5. Metricd workers

You can use the metricd daemon to processes measures, create aggregates, store measures in aggregate storage and delete metrics. The metricd daemon is responsible for most CPU usage and I/O jobs in Gnocchi. The archive policy of each metric determines how fast the metricd daemon performs. Metricd checks the incoming storage for new measures periodically. To configure the delay between each check, you can use the **[metricd]metric\_processing\_delay configuration** option.

### 2.4.6. Creating an archive policy

#### Procedure

- Create an archive policy. Replace <archive-policy-name> with the name of the policy and <aggregation-method> with the method of aggregation.

```
# openstack metric archive policy create <archive-policy-name> --definition <definition> \
--aggregation-method <aggregation-method>
```



#### NOTE

<definition> is the policy definition. Separate multiple attributes with a comma (,). Separate the name and value of the archive policy definition with a colon (:).

### 2.4.7. Managing archive policies

- To delete an archive policy:

```
openstack metric archive policy delete <archive-policy-name>
```

- To view all archive policies:

```
# openstack metric archive policy list
```

- To view the details of an archive policy:

```
# openstack metric archive-policy show <archive-policy-name>
```

### 2.4.8. Creating an archive policy rule

An archive policy rule defines a mapping between a metric and an archive policy. This gives users the ability to predefine rules so an archive policy is assigned to metrics based on a matched pattern.

#### Procedure

- Create an archive policy rule. Replace <rule-name> with the name of the rule and <archive-policy-name> with the name of the archive policy:

```
# openstack metric archive-policy-rule create <rule-name> /
--archive-policy-name <archive-policy-name>
```

## 2.5. VERIFYING THE RED HAT OPENSTACK PLATFORM DEPLOYMENT

You can use the **openstack metric** command to verify a successful deployment.

### Procedure

- Verify the deployment:

```
(overcloud) [stack@undercloud-0 ~]$ openstack metric status
+-----+
| Field                               | Value |
+-----+
| storage/number of metric having measures to process | 0 |
| storage/total number of measures to process      | 0 |
+-----+
```

## CHAPTER 3. MANAGING ALARMS

You can use the Telemetry Alarming service (aodh) to trigger actions based on defined rules against metric or event data collected by Ceilometer or Gnocchi.

Alarms can be in one of the following states:

### ok

The metric or event is in an acceptable state.

### firing

The metric or event is outside of the defined ok state.

### insufficient data

The alarm state is unknown. This can be for several reasons, for example, there is no data for the requested granularity, the check has not been executed yet, and so on.

## 3.1. VIEWING EXISTING ALARMS

You can view existing Telemetry alarm information and list the meters assigned to a resource to check the current state of the metrics.

### Procedure

1. List the existing Telemetry alarms:

```
# openstack alarm list
+-----+-----+-----+-----+
+-----+-----+-----+
| alarm_id           | type           | name           | state         |
| severity | enabled |
+-----+-----+-----+-----+
+-----+-----+-----+
| 922f899c-27c8-4c7d-a2cf-107be51ca90a | gnocchi_aggregation_by_resources_threshold |
| iops-monitor-read-requests | insufficient data | low   | True   |
+-----+-----+-----+-----+
+-----+-----+-----+

```

2. To list the meters assigned to a resource, specify the UUID of the resource. For example:

```
# openstack metric resource show 22592ae1-922a-4f51-b935-20c938f48753
| Field           | Value |
+-----+-----+
| created_by_project_id | 1adaed3aaa7f454c83307688c0825978 |
| created_by_user_id   | d8429405a2764c3bb5184d29bd32c46a |
| creator              | d8429405a2764c3bb5184d29bd32c46a:1adaed3aaa7f454c83307688c0825978 |
| ended_at             | None |
| id                   | 22592ae1-922a-4f51-b935-20c938f48753 |
| metrics              | cpu: a0375b0e-f799-47ea-b4ba-f494cf562ad8 |
|                      | disk.ephemeral.size: cd082824-dfd6-49c3-afdf-6bfc8c12bd2a |
|                      | disk.root.size: cd88dc61-ba85-45eb-a7b9-4686a6a0787b |
|                      | memory.usage: 7a1e787c-5fa7-4ac3-a2c6-4c3821eaf80a |
|                      | memory: ebd38ef7-cdc1-49f1-87c1-0b627d7c189e |

```

```

|          | vcpus: cc9353f1-bb24-4d37-ab8f-d3e887ca8856          |
| original_resource_id | 22592ae1-922a-4f51-b935-20c938f48753          |
| project_id          | cdda46e0b5be4782bc0480dac280832a          |
| revision_end        | None                                         |
| revision_start      | 2021-09-16T17:00:41.227453+00:00          |
| started_at          | 2021-09-16T16:17:08.444032+00:00          |
| type                | instance                                    |
| user_id             | f00de1d74408428cadf483ea7dbb2a83          |
+-----+-----+

```

## 3.2. CREATING AN ALARM

Use the Telemetry Alarming service (aodh) to create an alarm that triggers when a particular condition is met, for example, when a threshold value is reached. In this example, the alarm activates and adds a log entry when the average CPU utilization for an individual instance exceeds 80%.

### Procedure

1. Archive policies are pre-populated during the deployment process and you rarely need to create a new archive policy. However, if there is no configured archive policy, you must create one. To create an archive policy that creates metrics for 5s \* 86400 points (5 days), use the following command:

```

# openstack archive-policy create <name> \
  -d granularity:5s,points:86400 \
  -b 3 -m mean -m rate:mean

```

+ Replace *<name>* with the name of the archive policy.



### NOTE

Ensure that you set the value of the evaluation period for the Telemetry Alarming service to an integer greater than 60. The Ceilometer polling interval is linked to the evaluation period. Ensure that you set the Ceilometer polling interval value to a number between 60 and 600 and ensure that the value is greater than the value of the evaluation period for the Telemetry Alarming service. If the Ceilometer polling interval is too low, it can severely impact system load.

2. Create an alarm and use a query to isolate the specific ID of the instance for monitoring purposes. The ID of the instance in the following example is 94619081-abf5-4f1f-81c7-9cedaa872403.



### NOTE

To calculate the threshold value, use the following formula:  $1,000,000,000 \times \{\text{granularity}\} \times \{\text{percentage\_in\_decimal}\}$

```

# openstack alarm create \
  --type gnocchi_aggregation_by_resources_threshold \
  --name cpu_usage_high \
  --granularity 5 \
  --metric cpu \
  --threshold 48000000000 \

```



```

--aggregation-method rate:mean \
--resource-type instance \
--query '{"=": {"id": "94619081-abf5-4f1f-81c7-9cedaa872403"}}' --alarm-action 'log://'
+-----+
| Field          | Value                                     |
+-----+
| aggregation_method | rate:mean                               |
| alarm_actions     | [u'log://']                             |
| alarm_id         | b794adc7-ed4f-4edb-ace4-88cbe4674a94    |
| comparison_operator | eq                                       |
| description      | gnocchi_aggregation_by_resources_threshold alarm rule |
| enabled         | True                                     |
| evaluation_periods | 1                                       |
| granularity      | 5                                       |
| insufficient_data_actions | []                                     |
| metric          | cpu                                       |
| name            | cpu_usage_high                           |
| ok_actions      | []                                       |
| project_id      | 13c52c41e0e543d9841a3e761f981c20      |
| query           | '{"=": {"id": "94619081-abf5-4f1f-81c7-9cedaa872403"}}' |
| repeat_actions  | False                                    |
| resource_type   | instance                                  |
| severity       | low                                       |
| state          | insufficient data                         |
| state_timestamp | 2016-12-09T05:18:53.326000             |
| threshold      | 48000000000.0                           |
| time_constraints | []                                       |
| timestamp      | 2016-12-09T05:18:53.326000             |
| type           | gnocchi_aggregation_by_resources_threshold |
| user_id       | 32d3f2c9a234423cb52fb69d3741dbbc      |
+-----+

```

### 3.3. EDITING AN ALARM

When you edit an alarm, you increase or decrease the value threshold of the alarm.

#### Procedure

- To update the threshold value, use the **openstack alarm update** command. For example, to increase the alarm threshold to 75%, use the following command:

```
# openstack alarm update --name cpu_usage_high --threshold 75
```

### 3.4. DISABLING AN ALARM

You can disable and enable alarms.

#### Procedure

- Disable the alarm:

```
# openstack alarm update --name cpu_usage_high --enabled=false
```

### 3.5. DELETING AN ALARM

Use the **openstack alarm delete** command to delete an alarm.

#### Procedure

- To delete an alarm, enter the following command:

```
# openstack alarm delete --name cpu_usage_high
```

### 3.6. EXAMPLE: MONITORING THE DISK ACTIVITY OF INSTANCES

This example demonstrates how to use an alarm that is part of the Telemetry Alarming service to monitor the cumulative disk activity for all the instances contained within a particular project.

#### Procedure

- Review the existing projects and select the appropriate UUID of the project that you want to monitor. This example uses the **admin** tenant:

```
$ openstack project list
+-----+-----+
| ID                | Name  |
+-----+-----+
| 745d33000ac74d30a77539f8920555e7 | admin |
| 983739bb834a42ddb48124a38def8538 | services |
| be9e767afd4c4b7ead1417c6dfedde2b | demo   |
+-----+-----+
```

- Use the project UUID to create an alarm that analyses the **sum()** of all read requests generated by the instances in the **admin** tenant. You can further restrain the query by using the **--query** parameter:

```
# openstack alarm create \
--type gnocchi_aggregation_by_resources_threshold \
--name iops-monitor-read-requests \
--metric disk.read.requests.rate \
--threshold 42000 \
--aggregation-method sum \
--resource-type instance \
--query '{"=": {"project_id": "745d33000ac74d30a77539f8920555e7"}}'
+-----+-----+
| Field                | Value                                |
+-----+-----+
| aggregation_method   | sum                                  |
| alarm_actions        | []                                   |
| alarm_id             | 192aba27-d823-4ede-a404-7f6b3cc12469 |
| comparison_operator  | eq                                   |
| description          | gnocchi_aggregation_by_resources_threshold alarm rule |
| enabled              | True                                 |
| evaluation_periods   | 1                                    |
| granularity          | 60                                   |
| insufficient_data_actions | []                                   |
```

```

| metric          | disk.read.requests.rate          |
| name           | iops-monitor-read-requests      |
| ok_actions     | []                                |
| project_id     | 745d33000ac74d30a77539f8920555e7 |
| query         | {"=": {"project_id": "745d33000ac74d30a77539f8920555e7"}} |
| repeat_actions | False                            |
| resource_type  | instance                          |
| severity       | low                               |
| state          | insufficient data                |
| state_timestamp | 2016-11-08T23:41:22.919000      |
| threshold      | 42000.0                          |
| time_constraints | []                                |
| timestamp      | 2016-11-08T23:41:22.919000      |
| type           | gnocchi_aggregation_by_resources_threshold |
| user_id       | 8c4aea738d774967b4ef388eb41fef5e |
+-----+-----+

```

### 3.7. EXAMPLE: MONITORING CPU USE

To monitor the performance of an instance, examine the Gnocchi database to identify which metrics you can monitor, such as memory or CPU usage.

#### Procedure

1. To identify the metrics you can monitor, enter the **openstack metric resource show** command with an instance UUID:

```
$ openstack metric resource show --type instance 22592ae1-922a-4f51-b935-20c938f48753
```

```

+-----+-----+
| Field          | Value                              |
+-----+-----+
| availability_zone | nova                                |
| created_at      | 2021-09-16T16:16:24+00:00          |
| created_by_project_id | 1adaed3aaa7f454c83307688c0825978 |
| created_by_user_id | d8429405a2764c3bb5184d29bd32c46a |
| creator        | d8429405a2764c3bb5184d29bd32c46a:1adaed3aaa7f454c83307688c0825978 |
| deleted_at     | None                                |
| display_name   | foo-2                              |
| ended_at       | None                                |
| flavor_id      | 0e5bae38-a949-4509-9868-82b353ef7ffb |
| flavor_name    | workload_flavor_0                  |
| host           | compute-0.redhat.local             |
| id             | 22592ae1-922a-4f51-b935-20c938f48753 |
| image_ref      | 3cde20b4-7620-49f3-8622-eeacbd43d49 |
| launched_at    | 2021-09-16T16:17:03+00:00          |
| metrics        | cpu: a0375b0e-f799-47ea-b4ba-f494cf562ad8 |
|                | disk.ephemeral.size: cd082824-dfd6-49c3-afdf-6bfc8c12bd2a |
|                | disk.root.size: cd88dc61-ba85-45eb-a7b9-4686a6a0787b |
|                | memory.usage: 7a1e787c-5fa7-4ac3-a2c6-4c3821eaf80a |
|                | memory: ebd38ef7-cdc1-49f1-87c1-0b627d7c189e |
|                | vcpus: cc9353f1-bb24-4d37-ab8f-d3e887ca8856 |
| original_resource_id | 22592ae1-922a-4f51-b935-20c938f48753 |
| project_id     | cdda46e0b5be4782bc0480dac280832a |

```

```

| revision_end      | None |
| revision_start   | 2021-09-16T17:00:41.227453+00:00 |
| server_group     | None |
| started_at       | 2021-09-16T16:17:08.444032+00:00 |
| type             | instance |
| user_id          | f00de1d74408428cadf483ea7dbb2a83 |
+-----+-----+

```

In this result, the metrics value lists the components you can monitor with the Alarming service, for example **cpu**.

- To monitor CPU usage, use the **cpu** metric:

```

$ openstack metric show --resource-id 22592ae1-922a-4f51-b935-20c938f48753 cpu
+-----+-----+
| Field          | Value |
+-----+-----+
| archive_policy/name | ceilometer-high-rate |
| creator        |      |
d8429405a2764c3bb5184d29bd32c46a:1adaed3aaa7f454c83307688c0825978 |
| id            | a0375b0e-f799-47ea-b4ba-f494cf562ad8 |
| name          | cpu |
| resource/created_by_project_id | 1adaed3aaa7f454c83307688c0825978 |
|
| resource/created_by_user_id | d8429405a2764c3bb5184d29bd32c46a |
|
| resource/creator |      |
d8429405a2764c3bb5184d29bd32c46a:1adaed3aaa7f454c83307688c0825978 |
| resource/ended_at | None |
| resource/id      | 22592ae1-922a-4f51-b935-20c938f48753 |
| resource/original_resource_id | 22592ae1-922a-4f51-b935-20c938f48753 |
|
| resource/project_id | cdda46e0b5be4782bc0480dac280832a |
| resource/revision_end | None |
| resource/revision_start | 2021-09-16T17:00:41.227453+00:00 |
| resource/started_at | 2021-09-16T16:17:08.444032+00:00 |
| resource/type     | instance |
| resource/user_id  | f00de1d74408428cadf483ea7dbb2a83 |
| unit             | ns |
+-----+-----+

```

The archive\_policy defines the aggregation interval for calculating the std, count, min, max, sum, mean values.

- Inspect the currently selected archive policy for the **cpu** metric:

```

$ openstack metric archive-policy show ceilometer-high-rate
+-----+-----+
| Field          | Value |
+-----+-----+
| aggregation_methods | rate:mean, mean |
| back_window      | 0 |
| definition       | - timespan: 1:00:00, granularity: 0:00:01, points: 3600 |
|                  | - timespan: 1 day, 0:00:00, granularity: 0:01:00, points: 1440 |

```

```

|           | - timespan: 365 days, 0:00:00, granularity: 1:00:00, points: 8760 |
| name      | ceilometer-high-rate          |
+-----+

```

- Use the Alarming service to create a monitoring task that queries **cpu**. This task triggers events based on the settings that you specify. For example, to raise a log entry when the CPU of an instance spikes over 80% for an extended duration, use the following command:

```

$ openstack alarm create \
  --project-id 3cee262b907b4040b26b678d7180566b \
  --name high-cpu \
  --type gnocchi_resources_threshold \
  --description 'High CPU usage' \
  --metric cpu \
  --threshold 800,000,000.0 \
  --comparison-operator ge \
  --aggregation-method mean \
  --granularity 300 \
  --evaluation-periods 1 \
  --alarm-action 'log://' \
  --ok-action 'log://' \
  --resource-type instance \
  --resource-id 22592ae1-922a-4f51-b935-20c938f48753

+-----+
| Field          | Value                               |
+-----+
| aggregation_method | rate:mean                           |
| alarm_actions     | ['log:']                             |
| alarm_id         | c7b326bd-a68c-4247-9d2b-56d9fb18bf38 |
| comparison_operator | ge                                   |
| description      | High CPU usage                       |
| enabled         | True                                 |
| evaluation_periods | 1                                    |
| granularity      | 300                                  |
| insufficient_data_actions | []                                  |
| metric          | cpu                                  |
| name            | high-cpu                             |
| ok_actions      | ['log:']                             |
| project_id      | cdda46e0b5be4782bc0480dac280832a   |
| repeat_actions  | False                                |
| resource_id     | 22592ae1-922a-4f51-b935-20c938f48753 |
| resource_type   | instance                             |
| severity        | low                                  |
| state           | insufficient data                    |
| state_reason     | Not evaluated yet                    |
| state_timestamp  | 2021-09-21T08:02:57.090592          |
| threshold       | 800000000.0                          |
| time_constraints | []                                    |
| timestamp       | 2021-09-21T08:02:57.090592          |
| type            | gnocchi_resources_threshold         |
| user_id         | f00de1d74408428cadf483ea7dbb2a83   |
+-----+

```

- comparison-operator: The ge operator defines that the alarm triggers if the CPU usage is greater than or equal to 80%.

- **granularity:** Metrics have an archive policy associated with them; the policy can have various granularities. For example, 5 minutes aggregation for 1 hour + 1 hour aggregation over a month. The granularity value must match the duration described in the archive policy.
- **evaluation-periods:** Number of granularity periods that need to pass before the alarm triggers. For example, if you set this value to 2, the CPU usage must be over 80% for two polling periods before the alarm triggers.
- `[u'log://']`: When you set **alarm\_actions** or **ok\_actions** to `[u'log://']`, events, for example, the alarm is triggered or returns to a normal state, are recorded to the aodh log file.



#### NOTE

You can define different actions to run when an alarm is triggered (`alarm_actions`), and when it returns to a normal state (`ok_actions`), such as a webhook URL.

## 3.8. VIEWING ALARM HISTORY

To check if a particular alarm has been triggered, you can query the alarm history and view the event information.

### Procedure

- Use the **openstack alarm-history show** command:

```
$ openstack alarm-history show 1625015c-49b8-4e3f-9427-3c312a8615dd --fit-width
+-----+-----+-----+
+-----+
| timestamp           | type           | detail
| event_id            |                |
+-----+-----+-----+
+-----+
| 2017-11-16T05:21:47.850094 | state transition | {"transition_reason": "Transition to ok due
to 1 samples inside threshold, most recent: 0.0366665763", "state": "ok"}
3b51f09d-ded1-4807-b6bb-65fdc87669e4 |
+-----+-----+-----+
+-----+
```

## CHAPTER 4. INSTALLING AND CONFIGURING THE LOGS SERVICE

Red Hat OpenStack Platform (RHOSP) writes informational messages to specific log files; you can use these messages for troubleshooting and monitoring system events. The log collection agent Rsyslog collects logs on the client side and sends these logs to an instance of Rsyslog that is running on the server side. The server-side Rsyslog instance redirects log records to Elasticsearch for storage.



### NOTE

You do not need to attach the individual log files to your support cases manually. The **sosreport** utility gathers the required logs automatically.

### 4.1. THE CENTRALIZED LOG SYSTEM ARCHITECTURE AND COMPONENTS

Monitoring tools use a client-server model with the client deployed onto the Red Hat OpenStack Platform (RHOSP) overcloud nodes. The Rsyslog service provides client-side centralized logging (CL).

All RHOSP services generate and update log files. These log files record actions, errors, warnings, and other events. In a distributed environment like OpenStack, collecting these logs in a central location simplifies debugging and administration.

With centralized logging, there is one central place to view logs across your entire RHOSP environment. 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 (Grafana)



### 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 Grafana. RHOSP 16.2 only supports rsyslog with Elasticsearch version 7.

### 4.2. ENABLING CENTRALIZED LOGGING WITH ELASTICSEARCH

To enable centralized logging, you must specify the implementation of the **OS::TripleO::Services::Rsyslog** composable service.



### NOTE

The Rsyslog service uses only Elasticsearch as a data store for centralized logging.

#### Prerequisites

- Elasticsearch is installed on the server side.

## Procedure

- Add the file path of the logging environment file to the **overcloud deploy** command with any other environment files that are relevant to your environment and deploy, as shown in the following example:

```
openstack overcloud deploy \
<existing_overcloud_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/logging-environment-
rsyslog.yaml
```

Replace **<existing\_overcloud\_environment\_files>** with the list of environment files that are part of your existing deployment.

## 4.3. CONFIGURING LOGGING FEATURES

To configure logging features, modify the **RsyslogElasticsearchSetting** parameter in the **logging-environment-rsyslog.yaml** file.

### Procedure

1. Copy the **tripleo-heat-templates/environments/logging-environment-rsyslog.yaml** file to your home directory.
2. Create entries in the **RsyslogElasticsearchSetting** parameter to suit your environment. The following snippet is an example configuration of the **RsyslogElasticsearchSetting** parameter:

```
parameter_defaults:
  RsyslogElasticsearchSetting:
    uid: "elastic"
    pwd: "yourownpassword"
    skipverifyhost: "on"
    allowunsignedcerts: "on"
    server: "https://log-store-service-
telemetry.apps.stfcloudops1.lab.upshift.rdu2.redhat.com"
    serverport: 443
```

### Additional resources

- For more information about the configurable parameters, see [Section 4.3.1, “Configurable logging parameters”](#).

### 4.3.1. Configurable logging parameters

This table contains descriptions of logging parameters that you use to configure logging features in Red Hat OpenStack Platform (RHOSP). You can find these parameters in the **tripleo-heat-templates/deployment/logging/rsyslog-container-puppet.yaml** file.

Table 4.1. Configurable logging parameters



Parameter	Description
<b>RsyslogElasticsearchSetting</b>	Configuration for <b>rsyslog-elasticsearch</b> plugin. For more information, see <a href="https://www.rsyslog.com/doc/v8-stable/configuration/modules/omelasticsearch.html">https://www.rsyslog.com/doc/v8-stable/configuration/modules/omelasticsearch.html</a> .
<b>RsyslogElasticsearchTlsCACert</b>	Contains the content of the CA cert for the CA that issued the Elasticsearch server cert.
<b>RsyslogElasticsearchTlsClientCert</b>	Contains the content of the client cert for doing client cert authorization against Elasticsearch.
<b>RsyslogElasticsearchTlsClientKey</b>	Contains the content of the private key corresponding to the cert <b>RsyslogElasticsearchTlsClientCert</b> .

## 4.4. OVERRIDING THE DEFAULT PATH FOR A LOG FILE

If you modify the default containers and the modification includes the path to the service log file, you must also modify the default log file path. Every composable service has a **<service\_name>LoggingSource** parameter. For example, for the nova-compute service, the parameter is **NovaComputeLoggingSource**.

### Procedure

- To override the default path for the nova-compute service, add the path to the **NovaComputeLoggingSource** parameter in your configuration file:

```
NovaComputeLoggingSource:
  tag: openstack.nova.compute
  file: /some/other/path/nova-compute.log
```



### NOTE

For each service, define the **tag** and **file**. Other values are derived by default.

- You can modify the format for a specific service. This passes directly to the Rsyslog configuration. The default format for the **LoggingDefaultFormat** parameter is `/(? <time>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d+) (?<pid>\d+) (?<priority>\S+) (? <message>.*)$/` Use the following syntax:

```
<service_name>LoggingSource:
  tag: <service_name>.tag
  path: <service_name>.path
  format: <service_name>.format
```

The following snippet is an example of a more complex transformation:

```
ServiceLoggingSource:
  tag: openstack.Service
  path: /var/log/containers/service/service.log
  format: multiline
```

```

format_firstline: '/^\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}.\d{3} \d+ \S+ \S+ \[(req-\S+ \S+ \S+
\S+ \S+ \S+|-)\]'
```

```

format1: '/^(?<Timestamp>\S+ \S+) (?<Pid>\d+) (?<log_level>\S+) (?
<python_module>\S+) (\[(req-(?<request_id>\S+) (?<user_id>\S+) (?<tenant_id>\S+) (?
<domain_id>\S+) (?<user_domain>\S+) (?<project_domain>\S+)-|-)\])? (?<Payload>.*)?
$/'
```

## 4.5. MODIFYING THE FORMAT OF A LOG RECORD

You can modify the format of the start of the log record for a specific service. This passes directly to the Rsyslog configuration.

The default format for the Red Hat OpenStack Platform (RHOSP) log record is ('^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}(\.[0-9]+ [0-9]+)? (DEBUG|INFO|WARNING|ERROR) ').

### Procedure

- To add a different regular expression for parsing the start of log records, add **startmsg.regex** to the configuration:

```

NovaComputeLoggingSource:
  tag: openstack.nova.compute
  file: /some/other/path/nova-compute.log
  startmsg.regex: "^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}(\.[0-9]+ \+[0-9]+)? [A-
Z]+ \|[a-z]+\|"
```

## 4.6. TESTING THE CONNECTION BETWEEN RSYSLOG AND ELASTICSEARCH

On the client side, you can verify communication between Rsyslog and Elasticsearch.

### Procedure

- Navigate to the Elasticsearch connection log file, **/var/log/rsyslog/omelasticsearch.log** in the Rsyslog container or **/var/log/containers/rsyslog/omelasticsearch.log** on the host. If this log file does not exist or if the log file exists but does not contain logs, there is no connection problem. If the log file is present and contains logs, Rsyslog has not connected successfully.



### NOTE

To test the connection from the server side, view the Elasticsearch logs for connection issues.

## 4.7. SERVER-SIDE LOGGING

If you have an Elasticsearch cluster running, you must configure the **RsyslogElasticsearchSetting** parameter in the **logging-environment-rsyslog.yaml** file to connect Rsyslog that is running on overcloud nodes. To configure the **RsyslogElasticsearchSetting** parameter, see <https://www.rsyslog.com/doc/v8-stable/configuration/modules/omelasticsearch.html>

## 4.8. TRACEBACKS

When you encounter an issue and you start troubleshooting, you can use a traceback log to diagnose the issue. In log files, tracebacks usually have several lines of information, all relating to the same issue.

Rsyslog provides a regular expression to define how a log record starts. Each log record usually starts with a timestamp and the first line of the traceback is the only line that contains this information. Rsyslog bundles the indented records with the first line and sends them as one log record.

For that behaviour configuration option **startmsg.regex** in `<Service>LoggingSource` is used. The following regular expression is the default value for all `<service>LoggingSource` parameters in director:

```
startmsg.regex='^[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}(. [0-9]+ [0-9]+)?
(DEBUG|INFO|WARNING|ERROR)'
```

When this default does not match log records of your added or modified **LoggingSource**, you must change **startmsg.regex** accordingly.

## 4.9. LOCATION OF LOG FILES FOR OPENSTACK SERVICES

Each OpenStack component has a separate logging directory containing files specific to a running service.

### 4.9.1. Bare Metal Provisioning (ironic) log files

Service	Service name	Log path
OpenStack Ironic API	openstack-ironic-api.service	/var/log/containers/ironic/ironic-api.log
OpenStack Ironic Conductor	openstack-ironic-conductor.service	/var/log/containers/ironic/ironic-conductor.log

### 4.9.2. Block Storage (cinder) log files

Service	Service name	Log path
Block Storage API	openstack-cinder-api.service	/var/log/containers/cinder-api.log
Block Storage Backup	openstack-cinder-backup.service	/var/log/containers/cinder/backup.log
Informational messages	The cinder-manage command	/var/log/containers/cinder/cinder-manage.log
Block Storage Scheduler	openstack-cinder-scheduler.service	/var/log/containers/cinder/scheduler.log
Block Storage Volume	openstack-cinder-volume.service	/var/log/containers/cinder/volume.log

### 4.9.3. Compute (nova) log files

Service	Service name	Log path
OpenStack Compute API service	openstack-nova-api.service	/var/log/containers/nova/nova-api.log
OpenStack Compute certificate server	openstack-nova-cert.service	/var/log/containers/nova/nova-cert.log
OpenStack Compute service	openstack-nova-compute.service	/var/log/containers/nova/nova-compute.log
OpenStack Compute Conductor service	openstack-nova-conductor.service	/var/log/containers/nova/nova-conductor.log
OpenStack Compute VNC console authentication server	openstack-nova-consoleauth.service	/var/log/containers/nova/nova-consoleauth.log
Informational messages	nova-manage command	/var/log/containers/nova/nova-manage.log
OpenStack Compute NoVNC Proxy service	openstack-nova-novncproxy.service	/var/log/containers/nova/nova-novncproxy.log
OpenStack Compute Scheduler service	openstack-nova-scheduler.service	/var/log/containers/nova/nova-scheduler.log

### 4.9.4. Dashboard (horizon) log files

Service	Service name	Log path
Log of certain user interactions	Dashboard interface	/var/log/containers/horizon/horizon.log

The Apache HTTP server uses several additional log files for the Dashboard web interface, which you can access by using a web browser or command-line client, for example, keystone and nova. The log files in the following table can be helpful in tracking the use of the Dashboard and diagnosing faults:

Purpose	Log path
All processed HTTP requests	/var/log/containers/httpd/horizon_access.log
HTTP errors	/var/log/containers/httpd/horizon_error.log
Admin-role API requests	/var/log/containers/httpd/keystone_wsgi_admin_access.log

Purpose	Log path
Admin-role API errors	/var/log/containers/httpd/keystone_wsgi_admin_error.log
Member-role API requests	/var/log/containers/httpd/keystone_wsgi_main_access.log
Member-role API errors	/var/log/containers/httpd/keystone_wsgi_main_error.log

**NOTE**

There is also **/var/log/containers/httpd/default\_error.log**, which stores errors reported by other web services that are running on the same host.

#### 4.9.5. Identity Service (keystone) log files

Service	Service name	Log Path
OpenStack Identity Service	openstack-keystone.service	/var/log/containers/keystone/keystone.log

#### 4.9.6. Image Service (glance) log files

Service	Service name	Log path
OpenStack Image Service API server	openstack-glance-api.service	/var/log/containers/glance/api.log
OpenStack Image Service Registry server	openstack-glance-registry.service	/var/log/containers/glance/registry.log

#### 4.9.7. Networking (neutron) log files

Service	Service name	Log path
OpenStack Neutron DHCP Agent	neutron-dhcp-agent.service	/var/log/containers/neutron/dhcp-agent.log
OpenStack Networking Layer 3 Agent	neutron-l3-agent.service	/var/log/containers/neutron/l3-agent.log
Metadata agent service	neutron-metadata-agent.service	/var/log/containers/neutron/metadata-agent.log

Service	Service name	Log path
Metadata namespace proxy	n/a	/var/log/containers/neutron/neutron-ns-metadata-proxy- <i>UUID</i> .log
Open vSwitch agent	neutron-openvswitch-agent.service	/var/log/containers/neutron/openvswitch-agent.log
OpenStack Networking service	neutron-server.service	/var/log/containers/neutron/server.log

#### 4.9.8. Object Storage (swift) log files

OpenStack Object Storage sends logs to the system logging facility only.



#### NOTE

By default, all Object Storage log files go to **/var/log/containers/swift/swift.log**, using the local0, local1, and local2 syslog facilities.

The log messages of Object Storage are classified into two broad categories: those by REST API services and those by background daemons. The API service messages contain one line per API request, in a manner similar to popular HTTP servers; both the frontend (Proxy) and backend (Account, Container, Object) services post such messages. The daemon messages are less structured and typically contain human-readable information about daemons performing their periodic tasks. However, regardless of which part of Object Storage produces the message, the source identity is always at the beginning of the line.

Here is an example of a proxy message:

```
Apr 20 15:20:34 rhv-a24c-01 proxy-server: 127.0.0.1 127.0.0.1 20/Apr/2015/19/20/34 GET
/v1/AUTH_zaitcev%3Fformat%3Djson%26marker%3Dtestcont HTTP/1.0 200 - python-swiftclient-
2.1.0 AUTH_tk737d6... - 2 - txc454fa8ea4844d909820a-0055355182 - 0.0162 - -
1429557634.806570053 1429557634.822791100
```

Here is an example of ad-hoc messages from background daemons:

```
Apr 27 17:08:15 rhv-a24c-02 object-auditor: Object audit (ZBF). Since Mon Apr 27 21:08:15 2015:
Locally: 1 passed, 0 quarantined, 0 errors files/sec: 4.34 , bytes/sec: 0.00, Total time: 0.23, Auditing
time: 0.00, Rate: 0.00
Apr 27 17:08:16 rhv-a24c-02 object-auditor: Object audit (ZBF) "forever" mode completed: 0.56s.
Total quarantined: 0, Total errors: 0, Total files/sec: 14.31, Total bytes/sec: 0.00, Auditing time: 0.02,
Rate: 0.04
Apr 27 17:08:16 rhv-a24c-02 account-replicator: Beginning replication run
Apr 27 17:08:16 rhv-a24c-02 account-replicator: Replication run OVER
Apr 27 17:08:16 rhv-a24c-02 account-replicator: Attempted to replicate 5 dbs in 0.12589 seconds
(39.71876/s)
Apr 27 17:08:16 rhv-a24c-02 account-replicator: Removed 0 dbs
Apr 27 17:08:16 rhv-a24c-02 account-replicator: 10 successes, 0 failures
```

### 4.9.9. Orchestration (heat) log files

Service	Service name	Log path
OpenStack Heat API Service	openstack-heat-api.service	/var/log/containers/heat/heat-api.log
OpenStack Heat Engine Service	openstack-heat-engine.service	/var/log/containers/heat/heat-engine.log
Orchestration service events	n/a	/var/log/containers/heat/heat-manage.log

### 4.9.10. Shared Filesystem Service (manila) log files

Service	Service name	Log path
OpenStack Manila API Server	openstack-manila-api.service	/var/log/containers/manila/api.log
OpenStack Manila Scheduler	openstack-manila-scheduler.service	/var/log/containers/manila/scheduler.log
OpenStack Manila Share Service	openstack-manila-share.service	/var/log/containers/manila/share.log



#### NOTE

Some information from the Manila Python library can also be logged in **/var/log/containers/manila/manila-manage.log**.

### 4.9.11. Telemetry (ceilometer) log files

Service	Service name	Log path
OpenStack ceilometer notification agent	ceilometer_agent_notification	/var/log/containers/ceilometer/agent-notification.log
OpenStack ceilometer central agent	ceilometer_agent_central	/var/log/containers/ceilometer/central.log
OpenStack ceilometer collection	openstack-ceilometer-collector.service	/var/log/containers/ceilometer/collector.log
OpenStack ceilometer compute agent	ceilometer_agent_compute	/var/log/containers/ceilometer/compute.log

### 4.9.12. Log files for supporting services

The following services are used by the core OpenStack components and have their own log directories and files.

Service	Service name	Log path
Message broker (RabbitMQ)	rabbitmq-server.service	/var/log/rabbitmq/rabbit@short_hostname.log /var/log/rabbitmq/rabbit@short_hostname-sasl.log (for Simple Authentication and Security Layer related log messages)
Database server (MariaDB)	mariadb.service	/var/log/mariadb/mariadb.log
Virtual network switch (Open vSwitch)	openvswitch-nonetwork.service	/var/log/openvswitch/ovsdb-server.log /var/log/openvswitch/ovs-vswitchd.log

### 4.9.13. aodh (alarming service) log files

Service	Container name	Log path
Alarming API	aodh_api	/var/log/containers/httpd/aodh-api/aodh_wsgi_access.log
Alarm evaluator log	aodh_evaluator	/var/log/containers/aodh/aodh-evaluator.log
Alarm listener	aodh_listener	/var/log/containers/aodh/aodh-listener.log
Alarm notification	aodh_notifier	/var/log/containers/aodh/aodh-notifier.log

### 4.9.14. gnocchi (metric storage) log files

Service	Container name	Log path
Gnocchi API	gnocchi_api	/var/log/containers/httpd/gnocchi-api/gnocchi_wsgi_access.log
Gnocchi metricd	gnocchi_metricd	/var/log/containers/gnocchi/gnocchi-metricd.log
Gnocchi statsd	gnocchi_statsd	/var/log/containers/gnocchi/gnocchi-statsd.log



## CHAPTER 5. COLLECTD PLUGINS

You can configure multiple collectd plugins depending on your Red Hat OpenStack Platform (RHOSP) 16.2 environment.

The following list of plugins shows the available heat template **ExtraConfig** parameters that you can set to override the default values. Each section provides the general configuration name for the **ExtraConfig** option. For example, if there is a collectd plugin called **example\_plugin**, the format of the plugin title is **collectd::plugin::example\_plugin**.

Reference the tables of available parameters for specific plugins, such as in the following example:

```
ExtraConfig:
collectd::plugin::example_plugin::<parameter>: <value>
```

Reference the metrics tables of specific plugins for Prometheus or Grafana queries.

### collectd::plugin::aggregation

You can aggregate several values into one with the **aggregation** plugin. Use the aggregation functions such as **sum**, **average**, **min**, and **max** to calculate metrics, for example average and total CPU statistics.

**Table 5.1. aggregation parameters**

Parameter	Type
host	String
plugin	String
plugininstance	Integer
agg_type	String
typeinstance	String
sethost	String
setplugin	String
setplugininstance	Integer
settypeinstance	String
groupby	Array of Strings
calculatesum	Boolean
calculatenum	Boolean
calculateaverage	Boolean

Parameter	Type
calculateminimum	Boolean
calculatemaximum	Boolean
calculatestddev	Boolean

### Example configuration:

Deploy three aggregate configurations to create the following files:

1. **aggregator-calcCpuLoadAvg.conf**: average CPU load for all CPU cores grouped by host and state
2. **aggregator-calcCpuLoadMinMax.conf**: minimum and maximum CPU load groups by host and state
3. **aggregator-calcMemoryTotalMaxAvg.conf**: maximum, average, and total for memory grouped by type

The aggregation configurations use the default **cpu** and **memory** plugin configurations.

```
parameter_defaults:
  CollectdExtraPlugins:
    - aggregation

  ExtraConfig:
    collectd::plugin::aggregation::aggregators:
      calcCpuLoadAvg:
        plugin: "cpu"
        agg_type: "cpu"
        groupby:
          - "Host"
          - "TypeInstance"
        calculateaverage: True
      calcCpuLoadMinMax:
        plugin: "cpu"
        agg_type: "cpu"
        groupby:
          - "Host"
          - "TypeInstance"
        calculatemaximum: True
        calculateminimum: True
      calcMemoryTotalMaxAvg:
        plugin: "memory"
        agg_type: "memory"
        groupby:
          - "TypeInstance"
        calculatemaximum: True
        calculateaverage: True
        calculatsum: True
```

```
collectd::plugin::amqp1
```

Use the **amqp1** plugin to write values to an amqp1 message bus, for example, AMQ Interconnect.

**Table 5.2. amqp1 parameters**

Parameter	Type
manage_package	Boolean
transport	String
host	String
port	Integer
user	String
password	String
address	String
instances	Hash
retry_delay	Integer
send_queue_limit	Integer
interval	Integer

Use the **send\_queue\_limit** parameter to limit the length of the outgoing metrics queue.



#### NOTE

If there is no AMQP1 connection, the plugin continues to queue messages to send, which can result in unbounded memory consumption. The default value is 0, which disables the outgoing metrics queue.

Increase the value of the **send\_queue\_limit** parameter if metrics are missing.

#### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - amqp1

  ExtraConfig:
    collectd::plugin::amqp1::send_queue_limit: 5000
```

**collectd::plugin::apache**

Use the **apache** plugin to collect Apache data from the **mod\_status** plugin that is provided by the Apache web server. Each instance provided has a per-**interval** value specified in seconds. If you provide the **timeout** interval parameter for an instance, the value is in milliseconds.

**Table 5.3. apache parameters**

Parameter	Type
instances	Hash
interval	Integer
manage-package	Boolean
package_install_options	List

**Table 5.4. apache instances parameters**

Parameter	Type
url	HTTP URL
user	String
password	String
verifypeer	Boolean
verifyhost	Boolean
cacert	AbsolutePath
sslciphers	String
timeout	Integer

### Example configuration:

In this example, the instance name is **localhost**, which connects to the Apache web server at [http://10.0.0.111/mod\\_status?auto](http://10.0.0.111/mod_status?auto). You must append **?auto** to the end of the URL to prevent the status page returning as a type that is incompatible with the plugin.

```
parameter_defaults:
  CollectdExtraPlugins:
    - apache

  ExtraConfig:
    collectd::plugin::apache::instances:
      localhost:
        url: "http://10.0.0.111/mod_status?auto"
```

## Additional resources

For more information about configuring the **apache** plugin, see [apache](#).

### collectd::plugin::battery

Use the **battery** plugin to report the remaining capacity, power, or voltage of laptop batteries.

Table 5.5. battery parameters

Parameter	Type
values_percentage	Boolean
report_degraded	Boolean
query_state_fs	Boolean
interval	Integer

## Additional resources

For more information about configuring the **battery** plugin, see [battery](#).

### collectd::plugin::bind

Use the **bind** plugin to retrieve encoded statistics about queries and responses from a DNS server, and submit those values to collectd.

Table 5.6. bind parameters

Parameter	Type
url	HTTP URL
memorystats	Boolean
opcodes	Boolean
parsetime	Boolean
qtypes	Boolean
resolverstats	Boolean
serverstats	Boolean
zonemaintstats	Boolean
views	Array
interval	Integer

Table 5.7. bind views parameters

Parameter	Type
name	String
qtypes	Boolean
resolverstats	Boolean
cacherrsets	Boolean
zones	List of strings

**Example configuration:**

```

parameter_defaults:
  CollectdExtraPlugins:
    - bind

  ExtraConfig:
    collectd::plugins::bind:
      url: http://localhost:8053/
      memorystats: true
      opcodes: true
      parsetime: false
      qtypes: true
      resolverstats: true
      serverstats: true
      zonemaintstats: true
    views:
      - name: internal
        qtypes: true
        resolverstats: true
        cacherrsets: true
      - name: external
        qtypes: true
        resolverstats: true
        cacherrsets: true
    zones:
      - "example.com/IN"

```

**collectd::plugin::ceph**

Use the **ceph** plugin to gather data from ceph daemons.

Table 5.8. ceph parameters

Parameter	Type
daemons	Array

Parameter	Type
longrunavglatency	Boolean
convertspecialmetrictypes	Boolean
package_name	String

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::ceph::daemons:
      - ceph-osd.0
      - ceph-osd.1
      - ceph-osd.2
      - ceph-osd.3
      - ceph-osd.4
```



#### NOTE

If an Object Storage Daemon (OSD) is not on every node, you must list the OSDs.

When you deploy collectd, the **ceph** plugin is added to the Ceph nodes. Do not add the **ceph** plugin on Ceph nodes to **CollectdExtraPlugins** because this results in a deployment failure.

### Additional resources

For more information about configuring the **ceph** plugin, see [ceph](#).

### collectd::plugins::cgroups

Use the **cgroups** plugin to collect information for processes in a cgroup.

Table 5.9. cgroups parameters

Parameter	Type
ignore_selected	Boolean
interval	Integer
cgroups	List

### Additional resources

For more information about configuring the **cgroups** plugin, see [cgroups](#).

### collectd::plugin::connectivity

Use the connectivity plugin to monitor the state of network interfaces.

**NOTE**

If no interfaces are listed, all interfaces are monitored by default.

**Table 5.10. connectivity parameters**

Parameter	Type
interfaces	Array

**Example configuration:**

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::connectivity::interfaces:
      - eth0
      - eth1
```

**Additional resources**

For more information about configuring the **connectivity** plugin, see [connectivity](#).

**collectd::plugin::conntrack**

Use the **conntrack** plugin to track the number of entries in the Linux connection-tracking table. There are no parameters for this plugin.

**collectd::plugin::contextswitch**

Use the **ContextSwitch** plugin to collect the number of context switches that the system handles. The only parameter available is **interval**, which is a polling interval defined in seconds.

**Additional resources**

For more information about configuring the **contextswitch** plugin, see [contextswitch](#).

**collectd::plugin::cpu**

Use the **cpu** plugin to monitor the time that the CPU spends in various states, for example, idle, executing user code, executing system code, waiting for IO-operations, and other states.

The **cpu** plugin collects *jiffies*, not percentage values. The value of a jiffy depends on the clock frequency of your hardware platform, and therefore is not an absolute time interval unit.

To report a percentage value, set the Boolean parameters **reportbycpu** and **reportbystate** to **true**, and then set the Boolean parameter **valuespercentage** to true.

This plugin is enabled by default.

**Table 5.11. cpu metrics**

Name	Description	Query
idle	Amount of idle time	<b>collectd_cpu_total{...,type_instance='idle'}</b>



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

Table 5.12. cpu parameters

Parameter	Type	Defaults
reportbystate	Boolean	true
valuespercentage	Boolean	true
reportbycpu	Boolean	true
reportnumcpu	Boolean	false
reportgueststate	Boolean	false
subtractgueststate	Boolean	true
interval	Integer	120

**Example configuration:**

```
parameter_defaults:
  CollectdExtraPlugins:
    - cpu
```

ExtraConfig:  
 collectd::plugin::cpu::reportbystate: `true`

### Additional resources

For more information about configuring the `cpu` plugin, see [cpu](#).

### collectd::plugin::cpufreq

Use the `cpufreq` plugin to collect the current CPU frequency. There are no parameters for this plugin.

### collectd::plugin::csv

Use the `csv` plugin to write values to a local file in CSV format.

**Table 5.13. csv parameters**

Parameter	Type
datadir	String
storerates	Boolean
interval	Integer

### collectd::plugin::df

Use the `df` plugin to collect disk space usage information for file systems.

This plugin is enabled by default.

**Table 5.14. df metrics**

Name	Description	Query
free	Amount of free disk space	<code>collectd_df_df_complex{...,type_instance="free"}</code>
reserved	Amount of reserved disk space	<code>collectd_df_df_complex{...,type_instance="reserved"}</code>
used	Amount of used disk space	<code>collectd_df_df_complex{...,type_instance="used"}</code>

**Table 5.15. df parameters**

Parameter	Type	Defaults
devices	Array	<code>[]</code>
fstypes	Array	<code>['xfs']</code>

Parameter	Type	Defaults
ignoreselected	Boolean	true
mountpoints	Array	[]
reportbydevice	Boolean	true
reportinodes	Boolean	true
reportreserved	Boolean	true
valuesabsolute	Boolean	true
valuespercentage	Boolean	false

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::df::fstypes: ['tmpfs','xfs']
```

### Additional resources

For more information about configuring the **df** plugin, see [df](#).

### collectd::plugin::disk

Use the **disk** plugin to collect performance statistics of hard disks and, if supported, partitions.



#### NOTE

The **disk** plugin monitors all disks by default. You can use the **ignoreselected** parameter to ignore a list of disks. The example configuration ignores the *sda*, *sdb*, and *sdc* disks, and monitors all disks not included in the list.

This plugin is enabled by default.

**Table 5.16. disk parameters**

Parameter	Type	Defaults
disks	Array	[]
ignoreselected	Boolean	false
udevnameattr	String	<undefined>

**Table 5.17. disk metrics**

Name	Description
merged	The number of queued operations that can be merged together, for example, one physical disk access served two or more logical operations.
time	The average time an I/O-operation takes to complete. The values might not be accurate.
io_time	Time spent doing I/Os (ms). You can use this metric as a device load percentage. A value of 1 second matches 100% of load.
weighted_io_time	Measure of both I/O completion time and the backlog that might be accumulating.
pending_operations	Shows queue size of pending I/O operations.

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::disk::disks: ['sda', 'sdb', 'sdc']
    collectd::plugin::disk::ignoreselected: true
```

### Additional resources

For more information about configuring the **disk** plugin, see [disk](#).

### collectd::plugin::hugepages

Use the hugepages plugin to collect hugepages information.

This plugin is enabled by default.

**Table 5.18. hugepages parameters**

Parameter	Type	Defaults
report_per_node_hp	Boolean	true
report_root_hp	Boolean	true
values_pages	Boolean	true
values_bytes	Boolean	false
values_percentage	Boolean	false

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::hugepages::values_percentage: true
```

### Additional resources

- For more information about configuring the **hugepages** plugin, see [hugepages](#).

### collectd::plugin::interface

Use the **interface** plugin to measure interface traffic in octets, packets per second, and error rate per second.

This plugin is enabled by default.

Table 5.19. interface parameters

Parameter	Type	Default
interfaces	Array	[]
ignoreselected	Boolean	false
reportinactive	Boolean	true

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::interface::interfaces:
      - lo
    collectd::plugin::interface::ignoreselected: true
```

### Additional resources

- For more information about configuring the **interfaces** plugin, see [interfaces](#).

### collectd::plugin::load

Use the **load** plugin to collect the system load and an overview of the system use.

This plugin is enabled by default.

Table 5.20. plugin parameters

Parameter	Type	Default
report_relative	Boolean	true

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::load::report_relative: false
```

### Additional resources

- For more information about configuring the **load** plugin, see [load](#).

### collectd::plugin::mcelog

Use the **mcelog** plugin to send notifications and statistics that are relevant to Machine Check Exceptions when they occur. Configure **mcelog** to run in daemon mode and enable logging capabilities.

Table 5.21. mcelog parameters

Parameter	Type
Mcelogfile	String
Memory	Hash { <b>mcelogclientsocket</b> [string], <b>persistentnotification</b> [boolean] }

### Example configuration:

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

### Additional resources

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

### collectd::plugin::memcached

Use the **memcached** plugin to retrieve information about memcached cache usage, memory, and other related information.

Table 5.22. memcached parameters

Parameter	Type
instances	Hash
interval	Integer

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - memcached

  ExtraConfig:
```

```
collectd::plugin::memcached::instances:
  local:
    host: "%{hiera('fqdn_canonical')}"
    port: 11211
```

### Additional resources

- For more information about configuring the **memcached** plugin, see [memcached](#).

### collectd::plugin::memory

Use the **memory** plugin to retrieve information about the memory of the system.

This plugin is enabled by default.

Table 5.23. memory parameters

Parameter	Type
Defaults	valuesabsolute
Boolean	true
valuespercentage	Boolean

### Example configuration:

```
parameter_defaults:
  ExtraConfig:
    collectd::plugin::memory::valuesabsolute: true
    collectd::plugin::memory::valuespercentage: false
```

### Additional resources

- For more information about configuring the **memory** plugin, see [memory](#).

### collectd::plugin::ntpd

Use the **ntpd** plugin to query a local NTP server that is configured to allow access to statistics, and retrieve information about the configured parameters and the time sync status.

Table 5.24. ntpd parameters

Parameter	Type
host	Hostname
port	Port number (Integer)
reverselookups	Boolean
includeunitid	Boolean

Parameter	Type
interval	Integer

**Example configuration:**

```
parameter_defaults:
  CollectdExtraPlugins:
    - ntpd

  ExtraConfig:
    collectd::plugin::ntpd::host: localhost
    collectd::plugin::ntpd::port: 123
    collectd::plugin::ntpd::reverselookups: false
    collectd::plugin::ntpd::includeunitid: false
```

**Additional resources**

- For more information about configuring the **ntpd** plugin, see [ntpd](#).

**collectd::plugin::ovs\_stats**

Use the **ovs\_stats** plugin to collect statistics of OVS-connected interfaces. The **ovs\_stats** plugin uses the OVSDb management protocol (RFC7047) monitor mechanism to get statistics from OVSDb.

**Table 5.25. ovs\_stats parameters**

Parameter	Type
address	String
bridges	List
port	Integer
socket	String

**Example configuration:**

The following example shows how to enable the **ovs\_stats** plugin. If you deploy your overcloud with OVS, you do not need to enable the **ovs\_stats** plugin.

```
parameter_defaults:
  CollectdExtraPlugins:
    - ovs_stats

  ExtraConfig:
    collectd::plugin::ovs_stats::socket: '/run/openvswitch/db.sock'
```

**Additional resources**

- For more information about configuring the **ovs\_stats** plugin, see [ovs\\_stats](#).



## collectd::plugin::processes

The **processes** plugin provides information about system processes. If you do not specify custom process matching, the plugin collects only the number of processes by state and the process fork rate.

To collect more details about specific processes, you can use the **process** parameter to specify a process name or the **process\_match** option to specify process names that match a regular expression. The statistics for a **process\_match** output are grouped by process name.

Table 5.26. plugin parameters

Parameter	Type	Defaults
processes	Array	<undefined>
process_matches	Array	<undefined>
collect_context_switch	Boolean	<undefined>
collect_file_descriptor	Boolean	<undefined>
collect_memory_maps	Boolean	<undefined>

### Additional resources

- For more information about configuring the **processes** plugin, see [processes](#).

## collectd::plugin::smart

Use the **smart** plugin to collect SMART (self-monitoring, analysis and reporting technology) information from physical disks on the node. You must also set the parameter

**CollectdContainerAdditionalCapAdd** to **CAP\_SYS\_RAWIO** to allow the **smart** plugin to read SMART telemetry. If you do not set the **CollectdContainerAdditionalCapAdd** parameter, the following message is written to the collectd error logs:

**smart plugin: Running collectd as root, but the CAP\_SYS\_RAWIO capability is missing. The plugin's read function will probably fail. Is your init system dropping capabilities?.**

Table 5.27. smart parameters

Parameter	Type
disks	Array
ignoreselected	Boolean
interval	Integer

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - smart
```

```
CollectdContainerAdditionalCapAdd: "CAP_SYS_RAWIO"
```

### Additional information

- For more information about configuring the **smart** plugin, see [smart](#).

### collectd::plugin::swap

Use the **swap** plugin to collect information about the available and used swap space.

Table 5.28. swap parameters

Parameter	Type
reportbydevice	Boolean
reportbytes	Boolean
valuesabsolute	Boolean
valuespercentage	Boolean
reportio	Boolean

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - swap

  ExtraConfig:
    collectd::plugin::swap::reportbydevice: false
    collectd::plugin::swap::reportbytes: true
    collectd::plugin::swap::valuesabsolute: true
    collectd::plugin::swap::valuespercentage: false
    collectd::plugin::swap::reportio: true
```

### collectd::plugin::tcpconns

Use the **tcpconns** plugin to collect information about the number of TCP connections inbound or outbound from the configured port. The local port configuration represents ingress connections. The remote port configuration represents egress connections.

Table 5.29. tcpconns parameters

Parameter	Type
localports	Port (Array)
remoteports	Port (Array)

Parameter	Type
listening	Boolean
allportssummary	Boolean

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - tcpconns

ExtraConfig:
  collectd::plugin::tcpconns::listening: false
  collectd::plugin::tcpconns::localports:
    - 22
  collectd::plugin::tcpconns::remoteports:
    - 22
```

### collectd::plugin::thermal

Use the **thermal** plugin to retrieve ACPI thermal zone information.

Table 5.30. thermal parameters

Parameter	Type
devices	Array
ignoreselected	Boolean
interval	Integer

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - thermal
```

### collectd::plugin::uptime

Use the **uptime** plugin to collect information about system uptime.

This plugin is enabled by default.

Table 5.31. uptime parameters

Parameter	Type
interval	Integer

**collectd::plugin::virt**

Use the **virt** plugin to collect CPU, disk, network load, and other metrics through the **libvirt** API for virtual machines on the host.

This plugin is enabled by default on compute hosts.

**Table 5.32. virt parameters**

Parameter	Type
connection	String
refresh_interval	Hash
domain	String
block_device	String
interface_device	String
ignore_selected	Boolean
plugin_instance_format	String
hostname_format	String
interface_format	String
extra_stats	String

**Example configuration:**

```
ExtraConfig:
  collectd::plugin::virt::hostname_format: "name uuid hostname"
  collectd::plugin::virt::plugin_instance_format: metadata
```

**Additional resources**

For more information about configuring the **virt** plugin, see [virt](#).

**collectd::plugin::vmem**

Use the **vmem** plugin to collect information about virtual memory from the kernel subsystem.

**Table 5.33. vmem parameters**

Parameter	Type
verbose	Boolean
interval	Integer

**Example configuration:**

```
parameter_defaults:
  CollectdExtraPlugins:
    - vmem

  ExtraConfig:
    collectd::plugin::vmem::verbose: true
```

**collectd::plugin::write\_http**

Use the **write\_http** output plugin to submit values to an HTTP server by using POST requests and encoding metrics with JSON, or by using the **PUTVAL** command.

**Table 5.34. write\_http parameters**

Parameter	Type
ensure	Enum[ <i>present</i> , <i>absent</i> ]
nodes	Hash[String, Hash[String, Scalar]]
urls	Hash[String, Hash[String, Scalar]]
manage_package	Boolean

**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** plugin, see [write\\_http](#).

**collectd::plugin::write\_kafka**

Use the **write\_kafka** plugin to send values to a Kafka topic. Configure the **write\_kafka** plugin with one or more topic blocks. For each topic block, you must specify a unique name and one Kafka producer. You can use the following per-topic parameters inside the topic block:

**Table 5.35. write\_kafka parameters**

Parameter	Type
kafka_hosts	Array[String]

Parameter	Type
topics	Hash
properties	Hash
meta	Hash

### Example configuration:

```
parameter_defaults:
  CollectdExtraPlugins:
    - write_kafka
  ExtraConfig:
    collectd::plugin::write_kafka::kafka_hosts:
      - remote.tld:9092
    collectd::plugin::write_kafka::topics:
      mytopic:
        format: JSON
```

### Additional resources:

For more information about how to configure the **write\_kafka** plugin, see [write\\_kafka](#).