Abstract

This document provides an overview of the logging and monitoring capabilities that are available in a Red Hat OpenStack Platform environment, and how to troubleshoot possible issues.
3.10.10. Manage Resource Types

CHAPTER 4. TROUBLESHOOTING
4.1. SUPPORT
4.2. TROUBLESHOOT IDENTITY CLIENT (KEYSTONE) CONNECTIVITY PROBLEMS
4.3. TROUBLESHOOT OPENSTACK NETWORKING ISSUES
4.4. TROUBLESHOOT NETWORKS AND ROUTES TAB DISPLAY ISSUES IN THE DASHBOARD
4.5. TROUBLESHOOT INSTANCE LAUNCHING ERRORS IN THE DASHBOARD
4.6. TROUBLESHOOT KEYSTONE V3 DASHBOARD AUTHENTICATION
   4.6.1. OpenStack Dashboard - Red Hat Access Tab
      4.6.1.1. Search
      4.6.1.2. Logs
      4.6.1.3. Support
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. ABOUT THIS GUIDE

WARNING

Red Hat is currently reviewing the information and procedures provided in this guide for this release.


If you require assistance for the current Red Hat OpenStack Platform release, please contact Red Hat support.
CHAPTER 2. 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.

2.1. 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.

2.2. 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 deployment` command with any other environment files that are relevant to your environment and deploy, as shown in the following example:

  ```bash
  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.

### 2.3. CONFIGURING LOGGING FEATURES

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

**Procedure**


2. Create entries in the `RsyslogElasticsearchSetting` parameter to suit your environment. The following snippet is an example configuration of the `RsyslogElasticsearchSetting` parameter:

   ```yaml
   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 2.3.1, “Configurable logging parameters”.

### 2.3.1. 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 2.1. Configurable logging parameters |}
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RsyslogElasticsearchSetting</td>
<td>Configuration for <code>rsyslog-elasticsearch</code> 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>.</td>
</tr>
<tr>
<td>RsyslogElasticsearchTlsCACert</td>
<td>Contains the content of the CA cert for the CA that issued the Elasticsearch server cert.</td>
</tr>
<tr>
<td>RsyslogElasticsearchTlsClientCert</td>
<td>Contains the content of the client cert for doing client cert authorization against Elasticsearch.</td>
</tr>
<tr>
<td>RsyslogElasticsearchTlsClientKey</td>
<td>Contains the content of the private key corresponding to the cert <code>RsyslogElasticsearchTlsClientCert</code>.</td>
</tr>
</tbody>
</table>

#### 2.4. 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.

1. 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:
  ```
2.5. 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]+ \+[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]+)"
```

2.6. 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.

2.7. 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
2.8. 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.

2.9. LOCATION OF LOG FILES FOR OPENSTACK SERVICES

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

2.9.1. Bare Metal Provisioning (ironic) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Ironic API</td>
<td>openstack-ironic-api.service</td>
<td>/var/log/containers/ironic/ironic-api.log</td>
</tr>
<tr>
<td>OpenStack Ironic Conductor</td>
<td>openstack-ironic-conductor.service</td>
<td>/var/log/containers/ironic/ironic-conductor.log</td>
</tr>
</tbody>
</table>

2.9.2. Block Storage (cinder) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Storage API</td>
<td>openstack-cinder-api.service</td>
<td>/var/log/containers/cinder-api.log</td>
</tr>
<tr>
<td>Block Storage Backup</td>
<td>openstack-cinder-backup.service</td>
<td>/var/log/containers/cinder/backup.log</td>
</tr>
<tr>
<td>Informational messages</td>
<td>The cinder-manage command</td>
<td>/var/log/containers/cinder/cinder-manage.log</td>
</tr>
<tr>
<td>Block Storage Scheduler</td>
<td>openstack-cinder-scheduler.service</td>
<td>/var/log/containers/cinder/scheduler.log</td>
</tr>
</tbody>
</table>
### 2.9.3. Compute (nova) log files

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

### 2.9.4. Dashboard (horizon) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log of certain user interactions</td>
<td>Dashboard interface</td>
<td>/var/log/containers/horizon/horizon.log</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>All processed HTTP requests</td>
<td>/var/log/containers/httpd/horizon_access.log</td>
</tr>
<tr>
<td>HTTP errors</td>
<td>/var/log/containers/httpd/horizon_error.log</td>
</tr>
<tr>
<td>Admin-role API requests</td>
<td>/var/log/containers/httpd/keystone_wsgi_admin_access.log</td>
</tr>
<tr>
<td>Admin-role API errors</td>
<td>/var/log/containers/httpd/keystone_wsgi_admin_error.log</td>
</tr>
<tr>
<td>Member-role API requests</td>
<td>/var/log/containers/httpd/keystone_wsgi_main_access.log</td>
</tr>
<tr>
<td>Member-role API errors</td>
<td>/var/log/containers/httpd/keystone_wsgi_main_error.log</td>
</tr>
</tbody>
</table>

#### 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.

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Identity Service</td>
<td>openstack-keystone.service</td>
<td>/var/log/containers/keystone/keystone.log</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Image Service API server</td>
<td>openstack-glance-api.service</td>
<td>/var/log/containers/glance/api.log</td>
</tr>
<tr>
<td>OpenStack Image Service Registry server</td>
<td>openstack-glance-registry.service</td>
<td>/var/log/containers/glance/registry.log</td>
</tr>
</tbody>
</table>

### 2.9.7. Networking (neutron) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Neutron DHCP Agent</td>
<td>neutron-dhcp-agent.service</td>
<td>/var/log/containers/neutron/dhcp-agent.log</td>
</tr>
</tbody>
</table>
### 2.9.8. 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 rhev-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 rhev-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 rhev-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 rhev-a24c-02 account-replicator: Beginning replication run
```
### 2.9.9. Orchestration (heat) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Heat API Service</td>
<td>openstack-heat-api.service</td>
<td>/var/log/containers/heat/heat-api.log</td>
</tr>
<tr>
<td>OpenStack Heat Engine Service</td>
<td>openstack-heat-engine.service</td>
<td>/var/log/containers/heat/heat-engine.log</td>
</tr>
<tr>
<td>Orchestration service events</td>
<td>n/a</td>
<td>/var/log/containers/heat/heat-manage.log</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack Manila API Server</td>
<td>openstack-manila-api.service</td>
<td>/var/log/containers/manila/api.log</td>
</tr>
<tr>
<td>OpenStack Manila Scheduler</td>
<td>openstack-manila-scheduler.service</td>
<td>/var/log/containers/manila/scheduler.log</td>
</tr>
<tr>
<td>OpenStack Manila Share Service</td>
<td>openstack-manila-share.service</td>
<td>/var/log/containers/manila/share.log</td>
</tr>
</tbody>
</table>

**NOTE**

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

### 2.9.11. Telemetry (ceilometer) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Service name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack ceilometer notification agent</td>
<td>ceilometer_agent_notification</td>
<td>/var/log/containers/ceilometer/agent-notification.log</td>
</tr>
<tr>
<td>OpenStack ceilometer central agent</td>
<td>ceilometer_agent_central</td>
<td>/var/log/containers/ceilometer/central.log</td>
</tr>
</tbody>
</table>
2.9.12. Log files for supporting services

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

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

2.9.13. aodh (alarming service) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Container name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarming API</td>
<td>aodh_api</td>
<td>/var/log/containers/httpd/aodh-api/aodh_wsgi_access.log</td>
</tr>
<tr>
<td>Alarm evaluator log</td>
<td>aodh_evaluator</td>
<td>/var/log/containers/aodh/aodh-evaluator.log</td>
</tr>
<tr>
<td>Alarm listener</td>
<td>aodh_listener</td>
<td>/var/log/containers/aodh/aodh-listener.log</td>
</tr>
<tr>
<td>Alarm notification</td>
<td>aodh_notifier</td>
<td>/var/log/containers/aodh/aodh-notifier.log</td>
</tr>
</tbody>
</table>

2.9.14. gnocchi (metric storage) log files

<table>
<thead>
<tr>
<th>Service</th>
<th>Container name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnocchi API</td>
<td>gnocchi_api</td>
<td>/var/log/containers/httpd/gnocchi-api/gnocchi_wsgi_access.log</td>
</tr>
</tbody>
</table>
### CHAPTER 2. INSTALLING AND CONFIGURING THE LOGS SERVICE

<table>
<thead>
<tr>
<th>Service</th>
<th>Container name</th>
<th>Log path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnocchi metricd</td>
<td>gnocchi_metricd</td>
<td>/var/log/containers/gnocchi/gnocchi-metricd.log</td>
</tr>
<tr>
<td>Gnocchi statsd</td>
<td>gnocchi_statsd</td>
<td>/var/log/containers/gnocchi/gnocchi-statsd.log</td>
</tr>
</tbody>
</table>
CHAPTER 3. CONFIGURING THE TIME SERIES DATABASE (GNOCCHI) FOR TELEMETRY

Time series database (Gnocchi) is a multi-project, metrics, and resource database. It is designed to store metrics at a very large scale while providing access to metrics and resources information to operators and users.

3.1. UNDERSTANDING THE TIME SERIES DATABASE

This section defines the commonly used terms for the Time series database (Gnocchi) features.

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 time stamp 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 time stamp 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 that the metric is associated to.

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.

3.2. METRICS

The Time series database (Gnocchi) stores metrics from Telemetry that designate anything that can be measured, 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:

- UUID to identify the metric
The Time series database stores the following metrics by default, as defined in the `etc/ceilometer/polling.yaml` file:

```
[root@controller-0 ~]# podman exec -ti ceilometer_agent_central cat /etc/ceilometer/polling.yaml
---
sources:
- name: some_pollsters
  interval: 300
  meters:
  - cpu
  - memory.usage
  - network.incoming.bytes
  - network.incoming.packets
  - network.outgoing.bytes
  - network.outgoing.packets
  - disk.read.bytes
  - disk.read.requests
  - disk.write.bytes
  - disk.write.requests
  - hardware.cpu.util
  - hardware.memory.used
  - hardware.memory.total
  - hardware.memory.buffer
  - hardware.memory.cached
  - hardware.memory.swap.avail
  - hardware.memory.swap.total
  - hardware.system_stats.io.outgoing.blocks
  - hardware.system_stats.io.incoming.blocks
  - hardware.network.ip.incoming.datagrams
  - hardware.network.ip.outgoing.datagrams
```

The `polling.yaml` file also specifies the default polling interval of 300 seconds (5 minutes).

### 3.3. TIME SERIES DATABASE COMPONENTS

Currently, Gnocchi uses the Identity service for authentication and Redis for incoming measure storage. To store the aggregated measures, Gnocchi relies on either Swift or Ceph (Object Storage). Gnocchi also leverages MySQL to store the index of resources and metrics.

The time series database provides the `statsd` deamon (`gnocchi-statsd`) that is compatible with the `statsd` protocol and can listen to the metrics sent over the network. To enable `statsd` support in Gnocchi, configure the `[statsd]` option in the configuration file. The resource ID parameter is used as the main generic resource where all the metrics are attached, a user and project ID that are associated with the resource and metrics, and an archive policy name that is used to create the metrics.

All the metrics are created dynamically as the metrics are sent to `gnocchi-statsd`, and attached with the provided name to the resource ID you configured.

### 3.4. RUNNING THE TIME SERIES DATABASE

Run the time series database by running the HTTP server and metric daemon:
3.5. RUNNING AS A WSGI APPLICATION

You can run Gnocchi through a WSGI service such as `mod_wsgi` or any other WSGI application. You can use the `gnocchi/rest/app.wsgi` file, which is provided with Gnocchi, to enable Gnocchi as a WSGI application.

The Gnocchi API tier runs using WSGI. This means it can be run using Apache `httpd` and `mod_wsgi`, or another HTTP daemon such as `uwsgi`. Configure the number of processes and threads according to the number of CPUs you have, usually around $1.5 \times \text{number of CPUs}$. If one server is not enough, you can spawn any number of new API servers to scale Gnocchi out, even on different machines.

3.6. METRICD WORKERS

By default, the `gnocchi-metricd` daemon spans all your CPU power to maximize CPU utilization when computing metric aggregation. You can use the `gnocchi status` command to query the HTTP API and get the cluster status for metric processing. This command displays the number of metrics to process, known as the processing backlog for the `gnocchi-metricd`. As long as this backlog is not continuously increasing, that means that `gnocchi-metricd` can cope with the amount of metric that are being sent. If the number of measure to process is continuously increasing, you might need to temporarily increase the number of the `gnocchi-metricd` daemons. You can run any number of metricd daemons on any number of servers.

For director-based deployments, you can adjust certain metric processing parameters in your environment file:

- **MetricProcessingDelay** - Adjusts the delay period between iterations of metric processing.
- **GnocchiMetricdWorkers** - Configure the number of `metricd` workers.

3.7. MONITORING THE TIME SERIES DATABASE

The `/v1/status` endpoint of the HTTP API returns various information, such as the number of measures to process (measures backlog), which you can easily monitor. To verify good health of the overall system, ensure that the HTTP server and the `gnocchi-metricd` daemon are running and are not writing errors in their log files.

3.8. BACKING UP AND RESTORING THE TIME SERIES DATABASE

To recover from an unfortunate event, backup both the index and the storage. You must create a database dump (PostgreSQL or MySQL), and create snapshots or copies of your data storage (Ceph, Swift or your file system). The procedure to restore is: restore your index and storage backups, re-install Gnocchi if necessary, and restart it.

3.9. BATCH DELETING OLD RESOURCES FROM GNOCCHI

To remove outdated measures, create the archive policy to suit your requirements. To batch delete resources, metrics and measures, use the CLI or REST API. For example, to delete resources and all their associated metrics that were terminated 30 days ago, run the following command:

```
openstack metric resource batch delete "ended_at < '-30days'"
```
3.10. CAPACITY METERING USING THE TELEMETRY SERVICE

The OpenStack Telemetry service provides usage metrics that you can use for billing, charge-back, and show-back purposes. Such metrics data can also be used by third-party applications to plan for capacity on the cluster and can also be leveraged for auto-scaling virtual instances using OpenStack Heat. For more information, see Auto Scaling for Instances.

You can use the combination of Ceilometer and Gnocchi for monitoring and alarms. This is supported on small-size clusters and with known limitations. For real-time monitoring, Red Hat OpenStack Platform ships with agents that provide metrics data, and can be consumed by separate monitoring infrastructure and applications. For more information, see Monitoring Tools Configuration.

3.10.1. Viewing measures

List all the measures for a particular resource:

# openstack metric measures show --resource-id UUID METER_NAME

List only measures for a particular resource, within a range of timestamps:

# openstack metric measures show --aggregation mean --start <START_TIME> --stop <STOP_TIME> --resource-id UUID METER_NAME

The timestamp variables <START_TIME> and <STOP_TIME> use the format iso-dateThh:mm:ss.

3.10.2. Creating new measures

You can use measures to send data to the Telemetry service, and they do not need to correspond to a previously-defined meter. For example:

# openstack metrics measures add -m 2015-01-12T17:56:23@42 --resource-id UUID METER_NAME

3.10.3. Example: Viewing cloud usage measures

This example shows the average memory usage of all instances for each project.

# openstack metric measures aggregation --resource-type instance --groupby project_id -m memory

3.10.4. Example: Viewing L3 cache use

If your Intel hardware and libvirt version supports Cache Monitoring Technology (CMT), you can use the cpu_l3_cache meter to monitor the amount of L3 cache used by an instance.

Monitoring the L3 cache requires the following:

- `cmt` in the LibvirtEnabledPerfEvents parameter.
- `cpu_l3_cache` in the gnocchi_resources.yaml file.
- `cpu_l3_cache` in the Ceilometer polling.yaml file.
Enable L3 cache monitoring

To enable L3 cache monitoring:

1. Create a YAML file for telemetry (for example, ceilometer-environment.yaml) and add cmt to the LibvirtEnabledPerfEvents parameter.

   ```yaml
   parameter_defaults:
     LibvirtEnabledPerfEvents: cmt
   ```

2. Launch the overcloud with this YAML file.

   ```bash
   #!/bin/bash

   openstack overcloud deploy \
   -t templates \
   <additional templates> \
   -e /home/stack/ceilometer-environment.yaml
   ```

3. Verify that `cpu_l3_cache` is enabled in gnocchi on the Compute node.

   ```bash
   $ sudo -i
   # podman exec -ti ceilometer_agent_compute cat /etc/ceilometer/gnocchi_resources.yaml | grep cpu_l3_cache
   ```

4. Verify that `cpu_l3_cache` is enabled for Telemetry polling.

   ```bash
   # podman exec -ti ceilometer_agent_compute cat /etc/ceilometer/polling.yaml | grep cpu_l3_cache
   ```

5. If `cpu_l3_cache` is not enabled for Telemetry, enable it and restart the service.

   ```bash
   # podman exec -ti ceilometer_agent_compute echo "     - cpu_l3_cache" >> /etc/ceilometer/polling.yaml
   
   # podman exec -ti ceilometer_agent_compute pkill -HUP -f "ceilometer.*master process"
   ```

   **NOTE**

   This podman change does not persist over a reboot.

After you have launched a guest instance on this compute node, you can use the `gnocchi measures show` command to monitor the CMT metrics.

```bash
(overcloud) [stack@undercloud-0 ~]$ gnocchi measures show --resource-id a6491d92-b2c8-4f6d-94ba-edc9dfe23ac cpu_l3_cache
```

+---------------------------+-------------+-----------+
| timestamp                 | granularity |     value |
|----------------------------+-------------+-----------|
| 2017-10-25T09:40:00+00:00 |       300.0 | 1966080.0 |
| 2017-10-25T09:45:00+00:00 |       300.0 | 1933312.0 |
| 2017-10-25T09:50:00+00:00 |       300.0 | 2129920.0 |
| 2017-10-25T09:55:00+00:00 |       300.0 | 1966080.0 |
```
To list the existing Telemetry alarms, use the `aodh` command. For example:

```
# aodh alarm list
```

---
<table>
<thead>
<tr>
<th>alarm_id</th>
<th>type</th>
<th>name</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>922f899c-27c8-4c7d-a2cf-107be51ca90a</td>
<td>gnocchi_aggregation_by_resources_threshold</td>
<td>iops-monitor-read-requests</td>
<td>insufficient data</td>
</tr>
</tbody>
</table>
---

To list the meters assigned to a resource, specify the `UUID` of the resource (an instance, image, or volume, among others). For example:

```
# gnocchi resource show 5e3fcbe2-7aab-475d-b42c-a440aa42e5ad
```

### 3.10.5. View Existing Alarms

### 3.10.6. Create an Alarm
To edit an existing threshold alarm, use the `aodh alarm update` command. For example, to increase the alarm threshold to 75%:

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

### 3.10.7. Disable or Delete an Alarm

To disable an alarm:

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

To delete an alarm:

```
# aodh alarm delete --name cpu_usage_high
```

#### 3.10.8. Example: Monitor the disk activity of instances

The following example demonstrates how to use an Aodh alarm to monitor the cumulative disk activity for all the instances contained within a particular project.

1. Review the existing projects, and select the appropriate UUID of the project you need to monitor. This example uses the `admin` project:

```
$ openstack project list
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>745d33000ac74d30a77539f8920555e7</td>
<td>admin</td>
</tr>
<tr>
<td>983739bb834a42dd48124a38def8538</td>
<td>services</td>
</tr>
<tr>
<td>be9e767af4d4c4b7ead1417c6dfedde2b</td>
<td>demo</td>
</tr>
</tbody>
</table>

Red Hat OpenStack Platform 16.2 Logging, Monitoring, and Troubleshooting Guide
2. Use the project’s UUID to create an alarm that analyses the `sum()` of all read requests generated by the instances in the `admin` project (the query can be further restrained with the `--query` parameter).

```bash
# aodh 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"}}'
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregation_method</td>
<td>sum</td>
</tr>
<tr>
<td>alarm_actions</td>
<td>[]</td>
</tr>
<tr>
<td>alarm_id</td>
<td>192aba27-d823-4ede-a404-7f6b3cc12469</td>
</tr>
<tr>
<td>comparison_operator</td>
<td>eq</td>
</tr>
<tr>
<td>description</td>
<td>gnocchi_aggregation_by_resources_threshold alarm rule</td>
</tr>
<tr>
<td>enabled</td>
<td>True</td>
</tr>
<tr>
<td>evaluation_periods</td>
<td>1</td>
</tr>
<tr>
<td>granularity</td>
<td>60</td>
</tr>
<tr>
<td>insufficient_data_actions</td>
<td>[]</td>
</tr>
<tr>
<td>metric</td>
<td>disk.read.requests.rate</td>
</tr>
<tr>
<td>name</td>
<td>iops-monitor-read-requests</td>
</tr>
<tr>
<td>ok_actions</td>
<td>[]</td>
</tr>
<tr>
<td>project_id</td>
<td>745d33000ac74d30a77539f8920555e7</td>
</tr>
<tr>
<td>query</td>
<td>'{&quot;=&quot;: {&quot;project_id&quot;: &quot;745d33000ac74d30a77539f8920555e7&quot;}}'</td>
</tr>
<tr>
<td>repeat_actions</td>
<td>False</td>
</tr>
<tr>
<td>resource_type</td>
<td>instance</td>
</tr>
<tr>
<td>severity</td>
<td>low</td>
</tr>
<tr>
<td>state</td>
<td>insufficient data</td>
</tr>
<tr>
<td>state_timestamp</td>
<td>2016-11-08T23:41:22.919000</td>
</tr>
<tr>
<td>threshold</td>
<td>42000.0</td>
</tr>
<tr>
<td>time_constraints</td>
<td>[]</td>
</tr>
<tr>
<td>timestamp</td>
<td>2016-11-08T23:41:22.919000</td>
</tr>
<tr>
<td>type</td>
<td>gnocchi_aggregation_by_resources_threshold</td>
</tr>
<tr>
<td>user_id</td>
<td>8c4aea738d774967b4ef388eb41ef5e</td>
</tr>
</tbody>
</table>

3.10.9. Example: Monitor CPU usage

If you want to monitor an instance’s performance, you would start by examining the gnocchi database to identify which metrics you can monitor, such as memory or CPU usage. For example, run `gnocchi resource show` against an instance to identify which metrics can be monitored:

1. Query the available metrics for a particular instance UUID:

   ```bash
   $ gnocchi resource show --type instance d71cdf9a-51dc-4bb8-8170-9cd95edd3f66
   ```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>created_by_project_id</td>
<td>44adccdc32614688ae765ed4e484f389</td>
</tr>
<tr>
<td>created_by_user_id</td>
<td>c24fa60e46d14f8d847fca90531b43db</td>
</tr>
<tr>
<td>creator</td>
<td>c24fa60e46d14f8d847fca90531b43db</td>
</tr>
<tr>
<td>display_name</td>
<td>test-instance</td>
</tr>
<tr>
<td>ended_at</td>
<td>None</td>
</tr>
<tr>
<td>flavor_id</td>
<td>14c7c918-df24-481c-b498-0d3ec57d2e51</td>
</tr>
<tr>
<td>flavor_name</td>
<td>m1.tiny</td>
</tr>
</tbody>
</table>
In this result, the metrics value lists the components you can monitor using Aodh alarms, for example cpu_util.

2. To monitor CPU usage, you will need the cpu_util metric. To see more information on this metric:

```bash
$ gnocchi metric show --resource d71cdf9a-51dc-4bba-8170-9cd95edd3f66 cpu_util
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Red Hat OpenStack Platform 16.2 Logging, Monitoring, and Troubleshooting Guide
archive_policy/aggregation_methods | std, count, min, max, sum, mean
| archive_policy/back_window | 0
| archive_policy/definition | - points: 8640, granularity: 0:05:00, timespan: 30 days, 0:00:00 |
| archive_policy/name | low
| created_by_project_id | 44adccdc32614688ae765ed4e484f389
| created_by_user_id | c24fa60e46d14f8d847fca90531b43db
| creator | c24fa60e46d14f8d847fca90531b43db:44adccdc32614688ae765ed4e484f389
| id | b148c392-37d6-4c8f-8609-e15fc15a4728
| name | cpu_util
| resource/created_by_project_id | 44adccdc32614688ae765ed4e484f389
| resource/created_by_user_id | c24fa60e46d14f8d847fca90531b43db
| resource/creator | c24fa60e46d14f8d847fca90531b43db:44adccdc32614688ae765ed4e484f389
| resource/ended_at | None
| resource/id | d71cdf9a-51dc-4bba-8170-9cd95edd3f66
| resource/original_resource_id | d71cdf9a-51dc-4bba-8170-9cd95edd3f66
| resource/project_id | 3cee262b907b4040b26b678d7180566b
| resource/revision_end | None
| resource/revision_start | 2017-11-17T00:05:27.516421+00:00
| resource/started_at | 2017-11-16T01:09:20.668344+00:00
| resource/type | instance
| resource/user_id | 1dbf5787b2ee46cf9fa61dfeaa9c9996
| unit | None
+---------------------------+--------------------------------------+

- **archive_policy** - Defines the aggregation interval for calculating the **std, count, min, max, sum, mean** values.

3. Use Aodh to create a monitoring task that queries `cpu_util`. This task will trigger events based on the settings you specify. For example, to raise a log entry when an instance’s CPU spikes over 80% for an extended duration:

```bash
aodh alarm create \
--project-id 3cee262b907b4040b26b678d7180566b \
--name high-cpu \
--type gnocchi_resources_threshold \
--description 'High CPU usage' \
--metric cpu_util \
--threshold 80.0 \
--comparison-operator ge \
--aggregation-method mean \
--granularity 300 \
--evaluation-periods 1 \
--alarm-action 'log://' \
--ok-action 'log://' \
--resource-type instance \
--resource-id d71cdf9a-51dc-4bba-8170-9cd95edd3f66
```

+---------------------------+--------------------------------------+
| Field | Value |
+---------------------------+--------------------------------------+
• **comparison-operator** - The `ge` operator defines that the alarm will trigger 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 will trigger. For example, setting this value to 2 will mean that the CPU usage will need to be over 80% for two polling periods before the alarm will trigger.

• **[u'log://']** - This value will log events to your 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.

4. To check if your alarm has been triggered, query the alarm's history:

```bash
daodh alarm-history show 1625015c-49b8-4e3f-9427-3c312a8615dd --fit-width
```

+---------------------------------+-------------------+-------------------+
| timestamp                       | type              | detail            |
+---------------------------------+-------------------+-------------------+
| 2017-11-16T05:20:48.891365      | gnocchi_resources_threshold |                    |

28
3.10.10. Manage Resource Types

Telemetry resource types that were previously hardcoded can now be managed by the gnocchi client. You can use the gnocchi client to create, view, and delete resource types, and you can use the gnocchi API to update or delete attributes.

1. Create a new resource-type:

```bash
$ gnocchi resource-type create testResource01 -a bla:string:True:min_length=123
```

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes/bla</td>
<td>max_length=255, min_length=123, required=True, type=string</td>
</tr>
<tr>
<td>name</td>
<td>testResource01</td>
</tr>
<tr>
<td>state</td>
<td>active</td>
</tr>
</tbody>
</table>
```

2. Review the configuration of the resource-type:

```bash
$ gnocchi resource-type show testResource01
```

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes/bla</td>
<td>max_length=255, min_length=123, required=True, type=string</td>
</tr>
<tr>
<td>name</td>
<td>testResource01</td>
</tr>
<tr>
<td>state</td>
<td>active</td>
</tr>
</tbody>
</table>
```

3. Delete the resource-type:

```bash
$ gnocchi resource-type delete testResource01
```

**NOTE**

You cannot delete a resource type if a resource is using it.
CHAPTER 4. TROUBLESHOOTING

This chapter contains logging and support information to assist with troubleshooting your Red Hat OpenStack Platform deployment.

4.1. SUPPORT

If client commands fail or you run into other issues, contact Red Hat Technical Support with a description of what happened, the full console output, all log files referenced in the console output, and an sosreport from the node that is (or might be) in trouble. For example, if you encounter a problem on the compute level, run sosreport on the Nova node, or if it is a networking issue, run the utility on the Neutron node. For general deployment issues, it is best to run sosreport on the cloud controller.

For information about the sosreport command (sos package), refer to What is a sosreport and how to create one in Red Hat Enterprise Linux 4.6 and later.

Check also the /var/log/messages file for any hints.

4.2. TROUBLESHOOT IDENTIFY CLIENT (KEYSTONE) CONNECTIVITY PROBLEMS

When the Identity client (keystone) is unable to contact the Identity service it returns an error:

Unable to communicate with identity service: [Errno 113] No route to host. (HTTP 400)

To debug the issue check for these common causes:

Identity service is down

Identity Service now runs within httpd.service. On the system hosting the Identity service, check the service status:

```
# systemctl status httpd.service
```

If the service is not active then log in as the root user and start it.

```
# systemctl start httpd.service
```

Firewall is not configured properly

The firewall might not be configured to allow TCP traffic on ports 5000 and 35357. If so, see Managing the Overcloud Firewall in the Advanced Overcloud Customization guide for instructions on checking your firewall settings and defining custom rules.

Service Endpoints not defined correctly

On the system hosting the Identity service check that the endpoints are defined correctly.

1. Obtain the administration token:

```
# grep admin_token /etc/keystone/keystone.conf
admin_token = 91f0866234a64fc299db8f26f8729488
```

2. Determine the correct administration endpoint for the Identity service:
Replace IP with the IP address or host name of the system hosting the Identity service. Replace VERSION with the API version (v2.0, or v3) that is in use.

3. Unset any pre-defined Identity service related environment variables:

```bash
# unset OS_USERNAME OS_TENANT_NAME OS_PASSWORD OS_AUTH_URL
```

4. Use the administration token and endpoint to authenticate with the Identity service. Confirm that the Identity service endpoint is correct. For example:

```bash
# openstack endpoint list --os-token=91f0556234a64fc299db8f26f8729488 --os-url=https://osp.lab.local:35357/v3/ --os-identity-api-version 3
```

Verify that the listed publicurl, internalurl, and adminurl for the Identity service are correct. In particular ensure that the IP addresses and port numbers listed within each endpoint are correct and reachable over the network.

If these values are incorrect, add the correct endpoint and remove any incorrect endpoints using the endpoint delete action of the openstack command. For example:

```bash
# openstack endpoint delete 2d32fa6feecc49aab5de538bdf7aa018 --os-token=91f0866234a64fc299db8f26f8729488 --os-url=https://osp.lab.local:35357/v3/ --os-identity-api-version 3
```

Replace TOKEN and ENDPOINT with the values identified previously. Replace ID with the identity of the endpoint to remove as listed by the endpoint-list action.

### 4.3. TROUBLESHOOT OPENSTACK NETWORKING ISSUES

This section discusses the different commands you can use and procedures you can follow to troubleshoot the OpenStack Networking service issues.

#### Debugging Networking Device

- Use the `ip a` command to display all the physical and virtual devices.
- Use the `ovs-vsctl show` command to display the interfaces and bridges in a virtual switch.
- Use the `ovs-dpctl show` command to showdatapaths on the switch.

#### Tracking Networking Packets

- Use the `tcpdump` command to see where packets are not getting through.

```bash
# tcpdump -n -i INTERFACE -e -w FILENAME
```

Replace INTERFACE with the name of the network interface to see where the packets are not getting through. The interface name can be the name of the bridge or host Ethernet device.

The `-e` flag ensures that the link-level header is dumped (in which the `vlan` tag will appear).
The \texttt{-w} flag is optional. You can use it only if you want to write the output to a file. If not, the output is written to the standard output (stdout).

For more information about \texttt{tcpdump}, refer to its manual page by running \texttt{man tcpdump}.

### Debugging Network Namespaces

- Use the \texttt{ip netns list} command to list all known network namespaces.
- Use the \texttt{ip netns exec} command to show routing tables inside specific namespaces.

```
# ip netns exec NAMESPACE_ID bash
# route -n
```

Start the \texttt{ip netns exec} command in a bash shell so that subsequent commands can be invoked without the \texttt{ip netns exec} command.

#### 4.4. TROUBLESHOOT NETWORKS AND ROUTES TAB DISPLAY ISSUES IN THE DASHBOARD

The \textit{Networks} and \textit{Routers} tabs only appear in the dashboard when the environment is configured to use OpenStack Networking. In particular note that by default the PackStack utility currently deploys Nova Networking and as such in environments deployed in this manner the tab will not be visible.

If OpenStack Networking is deployed in the environment but the tabs still do not appear ensure that the service endpoints are defined correctly in the Identity service, that the firewall is allowing access to the endpoints, and that the services are running.

#### 4.5. TROUBLESHOOT INSTANCE LAUNCHING ERRORS IN THE DASHBOARD

When using the dashboard to launch instances if the operation fails, a generic \texttt{ERROR} message is displayed. Determining the actual cause of the failure requires the use of the command line tools.

Use the \texttt{nova list} command to locate the unique identifier of the instance. Then use this identifier as an argument to the \texttt{nova show} command. One of the items returned will be the error condition. The most common value is \texttt{NoValidHost}.

This error indicates that no valid host was found with enough available resources to host the instance. To work around this issue, consider choosing a smaller instance size or increasing the overcommit allowances for your environment.

\textbf{NOTE}

To host a given instance, the compute node must have not only available CPU and RAM resources but also enough disk space for the ephemeral storage associated with the instance.

#### 4.6. TROUBLESHOOT KEYSTONE V3 DASHBOARD AUTHENTICATION

\texttt{django_openstack_auth} is a pluggable Django authentication back end, that works with Django’s \texttt{contrib.auth} framework, to authenticate a user against the OpenStack Identity service API.
Django_openstack_auth uses the token object to encapsulate user and Keystone related information. The dashboard uses the token object to rebuild the Django user object.

The token object currently stores:

- Keystone token
- User information
- Scope
- Roles
- Service catalog

The dashboard uses Django’s sessions framework for handling user session data. The following is a list of numerous session back ends available, which are controlled through the SESSION_ENGINE setting in your local_settings.py file:

- Local Memory Cache
- Memcached
- Database
- Cached Database
- Cookies

In some cases, particularly when a signed cookie session back end is used and, when having many or all services enabled all at once, the size of cookies can reach its limit and the dashboard can fail to log in. One of the reasons for the growth of cookie size is the service catalog. As more services are registered, the bigger the size of the service catalog would be.

In such scenarios, to improve the session token management, include the following configuration settings for logging in to the dashboard, especially when using Keystone v3 authentication.

1. In /usr/share/openstack-dashboard/openstack_dashboard/settings.py add the following configuration:

   ```python
   DATABASES = 
   { 
   default:
   { 
   ENGINE: django.db.backends.mysql, 
   NAME: horizondb, 
   USER: User Name, 
   PASSWORD: Password, 
   HOST: localhost, 
   }
   }
   ```

2. In the same file, change SESSION_ENGINE to:

   ```python
   SESSION_ENGINE = 'django.contrib.sessions.backends.cached_db'
   ```
3. Connect to the database service using the `mysql` command, replacing USER with the user name by which to connect. The USER must be a root user (or at least as a user with the correct permission: create db).

```bash
# mysql -u USER -p
```

4. Create the Horizon database.

```bash
mysql > create database horizondb;
```

5. Exit the mysql client.

```bash
mysql > exit
```

6. Change to the openstack_dashboard directory and sync the database using:

```bash
# cd /usr/share/openstack-dashboard/openstack_dashboard
$ ./manage.py syncdb
```

You do not need to create a superuser, so answer `n` to the question.

7. Restart Apache http server. For Red Hat Enterprise Linux:

```bash
# systemctl restart httpd
```

### 4.6.1. OpenStack Dashboard – Red Hat Access Tab

The Red Hat Access tab, which is part of the OpenStack dashboard, allows you to search for and read articles or solutions from the Red Hat Customer Portal, view logs from your instances and diagnose them, and work with your customer support cases.
Figure 4.1. Red Hat Access Tab.

![Red Hat Access Tab](image)

Red Hat Access: Search

![Search Form](image)

**IMPORTANT**

You must be logged in to the Red Hat Customer Portal in the browser in order to be able to use the functions provided by the Red Hat Access tab.

If you are not logged in, you can do so now:

1. Click *Log In*.
2. Enter your Red Hat login.
3. Enter your Red Hat password.
4. Click *Sign in*.

This is how the form looks:
If you do not log in now, you will be prompted for your Red Hat login and password when you use one of the functions that require authentication.

4.6.1.1. Search

You can search for articles and solutions from Red Hat Customer Portal by entering one or more search keywords. The titles of the relevant articles and solutions will then be displayed. Click on a title to view the given article or solution:
4.6.1.2. Logs

Here you can read logs from your OpenStack instances:

Find the instance of your choice in the table. If you have many instances, you can filter them by name, status, image ID, or flavor ID. Click View Log in the Actions column for the instance to check.
When an instance log is displayed, you can click **Red Hat Diagnose** to get recommendations regarding its contents:

**Figure 4.5. Instance Logs on the Red Hat Access Tab.**

If none of the recommendations are useful or a genuine problem has been logged, click **Open a New Support Case** to report the problem to Red Hat Support.

### 4.6.1.3. Support

The last option in the Red Hat Access Tab allows you to search for your support cases at the Red Hat Customer Portal:

**Figure 4.6. Search for Support Cases.**

You can also open a new support case by clicking the appropriate button and filling out the form on the following page:
Figure 4.7. Open a New Support Case.