The LMA Collector Plugin for Fuel Documentation

Release 0.9.0

Mirantis Inc.

April 29, 2016

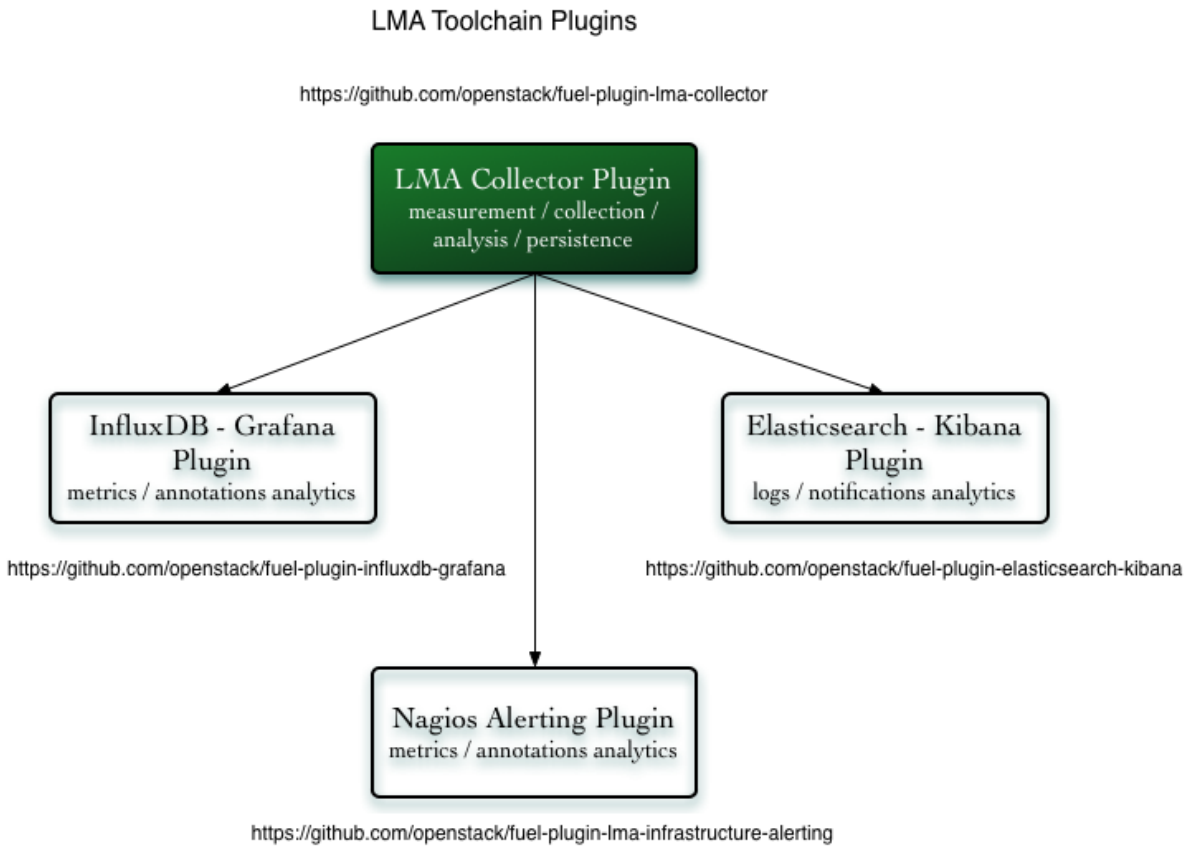
Contents

CHAPTER 1

Overview

The Logging, Monitoring & Alerting (LMA) Collector is an advanced monitoring agent solution that should be installed on each of the OpenStack nodes you want to monitor.

The LMA Collector (or Collector for short) is a key component of the LMA Toolchain project as shown in the figure below:



Each Collector is individually responsible for supporting the sensing, measurement, collection, analysis and alarm functions for the node it is running on.

A wealth of operational data is collected from a variety of sources including log files, collectd and RabbitMQ for the OpenStack notifications.

Note: The Collector which runs on the active controller of the control plane cluster, is called the *Aggregator* because it performs additional aggregation and multivariate correlation functions to compute service healthiness metrics at the cluster level.

A primary function of the Collector is to sanitise and transform the ingested raw operational data into internal messages using the Heka message structure. This message structure is used within the Collector's framework to match, filter and route messages to plugins written in Lua which perform various data analysis and computation functions.

As such, the Collector may also be described as a pluggable framework for operational data stream processing and routing.

Its main building blocks are:

- collectd which is bundled with a collection of monitoring plugins. Many of them are purpose-built for OpenStack.
- Heka (a golang data processing *swiss army knife* by Mozilla) which is the cornerstone component of the Collector. Heka supports out-of-the-box a number of input and output plugins that allows the Collector to integrate with a number of external systems' native protocol like Elasticsearch, InfluxDB, Nagios, SMTP, Whisper, Kafka, AMQP and Carbon to name a few.
- A collection of Heka plugins written in Lua to decode, process and encode the operational data.

There are three types of Lua plugins running in the Collector:

- The input plugins which collect, sanitize and transform the raw data into an internal message representation which is injected into the Heka pipeline for further processing.
- The filter plugins which execute the analysis and correlation functions.
- The output plugins which encode and transmit the messages to external systems like Elasticsearch, InfluxDB or Nagios where the data can be further processed and persisted.

The output of the Collector / Aggregator is of four kinds:

- The logs and notifications which are sent to Elasticsearch for indexing. Elasticsearch combined with Kibana provides insightful log analytics.
- The metrics which are sent to InfluxDB. InfluxDB combined with Grafana provides insightful time-series analytics.
- The health status metrics for the OpenStack services which are sent to Nagios (or via SMTP) for alerting and escalation purposes.
- The annotation messages which are sent to InfluxDB. The annotation messages contain information about what caused a service cluster or node cluster to change state. The annotation messages provide root cause analysis hints whenever possible. The annotation messages are also used to construct the alert notifications that are sent via SMTP or to Nagios.

1.1 Requirements

Requirement	Version/Comment
Mirantis OpenStack	8.0
A running Elasticsearch server (for log analytics)	1.7.4 or higher, the RESTful API must be enabled over port 9200
A running InfluxDB server (for metric analytics)	0.10.0 or higher, the RESTful API must be enabled over port 8086
A running Nagios server (for infrastructure alerting)	3.5 or higher, the command CGI must be enabled

1.2 Limitations

- The plugin is not compatible with an OpenStack environment deployed with Nova-Network.
- The Elasticsearch output plugin of the Collector is configured to use the **drop** policy which implies that the Collector will start dropping the logs and the OpenStack notifications when the output plugin has reached a buffering limit that is currently set to 1GB by default. This situation can typically happen when the Elasticsearch server has been inaccessible for a long period of time. This limitation will be addressed in a future release of the LMA Collector Plugin.
- When you re-execute tasks on deployed nodes using the Fuel CLI, hekad and collectd services will be restarted on these nodes during the post-deployment phase. See bug #1570850 for details.

CHAPTER 2

Release Notes

2.1 Version 0.9.0

- Changes
  - Upgrade to Heka *0.10.0*.
  - Collect libvirt metrics on compute nodes.
  - Detect spikes of errors in the OpenStack services logs.
  - Report OpenStack workers status per node.
  - Support multi-environment deployments.
  - Add support for Sahara logs and notifications.
- Bug fixes
  - Reconnect to the local RabbitMQ instance if the connection has been lost (#1503251).
  - Enable buffering for Elasticsearch, InfluxDB, Nagios and TCP outputs to reduce congestion in the Heka pipeline (#1488717, #1557388).
  - Return the correct status for Nova when Midonet is used (#1531541).
  - Return the correct status for Neutron when Contrail is used (#1546017).
  - Increase the maximum number of file descriptors (#1543289).
  - Avoid spawning several hekad processes (#1561109).
  - Remove the monitoring of individual queues of RabbitMQ (#1549721).
  - Rotate hekad logs every 30 minutes if necessary (#1561603).

2.2 Version 0.8.0

- Support for alerting in two different modes:
  - Email notifications.
  - Integration with Nagios.
- Upgrade to InfluxDB 0.9.5.
- Upgrade to Grafana 2.5.

- Management of the LMA collector service by Pacemaker on the controller nodes for improved reliability.
- Monitoring of the LMA toolchain components (self-monitoring).
- Support for configurable alarm rules in the Collector.

2.3 Version 0.7.0

- Initial release of the plugin. This is a beta version.

CHAPTER 3

Installation

Prior to installing the LMA Collector Plugin, you may want to install the backend services the Collector depends on. These backend services include:

- Elasticsearch
- InfluxDB
- Nagios

There are two options:

1. Install these backend services automatically using the Fuel Plugins listed below.

- Elasticsearch-Kibana Fuel Plugin Installation Guide.
- InfluxDB-Grafana Fuel Plugin Installation Guide.
- Infrastructure Alerting Fuel Plugin Installation Guide.

2. Install these backend services manually outside of your OpenStack environment. This installation must comply with the LMA Collector Plugin's *requirements*.

3.1 LMA Collector Fuel Plugin installation using the RPM file of the Fuel Plugins Catalog

To install the LMA Collector Fuel Plugin using the RPM file of the Fuel Plugins Catalog, follow these steps:

1. Download the RPM file from the Fuel Plugins Catalog.
2. Copy the RPM file to the Fuel Master node:

```
[root@home ~]# scp lma_collector-0.9-0.9.0-1.noarch.rpm \
root@<Fuel Master node IP address>:
```

3. Install the plugin using the Fuel CLI:

```
[root@fuel ~]# fuel plugins --install lma_collector-0.9-0.9.0-1.noarch.rpm
```

4. Verify that the plugin is installed correctly:

```
[root@fuel ~]# fuel plugins --list
id | name                 | version | package_version
---|----------------------|---------|----------------
1  | lma_collector        | 0.9.0   | 4.0.0
```

3.2 LMA Collector Fuel Plugin installation from source

Alternatively, you may want to build the RPM file of the plugin from source if, for example, you want to test the latest features, modify some built-in configuration or implement your own customization. But note that running a Fuel plugin that you have built yourself is at your own risk.

To install LMA Collector Plugin from source, you first need to prepare an environement to build the RPM file. The recommended approach is to build the RPM file directly onto the Fuel Master node so that you won't have to copy that file later on.

Prepare an environment to build the plugin on the Fuel Master Node

1. Install the standard Linux development tools:

```
[root@home ~] yum install createrepo rpm rpm-build dpkg-devel
```

2. Install the Fuel Plugin Builder. To do that, you should first get pip:

```
[root@home ~] easy_install pip
```

3. Then install the Fuel Plugin Builder (the *fpb* command line) with *pip*:

```
[root@home ~] pip install fuel-plugin-builder
```

Note: You may also need to build the Fuel Plugin Builder if the package version of the plugin is higher than the package version supported by the Fuel Plugin Builder you get from *pypi*. In this case, please refer to the section "Preparing an environment for plugin development" of the Fuel Plugins wiki if you need further instructions about how to build the Fuel Plugin Builder.

4. Clone the plugin git repository:

```
[root@home ~] git clone https://github.com/openstack/fuel-plugin-lma-collector.git
```

5. Check that the plugin is valid:

```
[root@home ~] fpb --check ./fuel-plugin-lma-collector
```

6. And finally, build the plugin:

```
[root@home ~] fpb --build ./fuel-plugin-lma-collector
```

7. Now that you have created the RPM file, you can install the plugin using the *fuel plugins –install* command:

```
[root@fuel ~] fuel plugins --install ./fuel-plugin-lma-collector/*.noarch.rpm
```

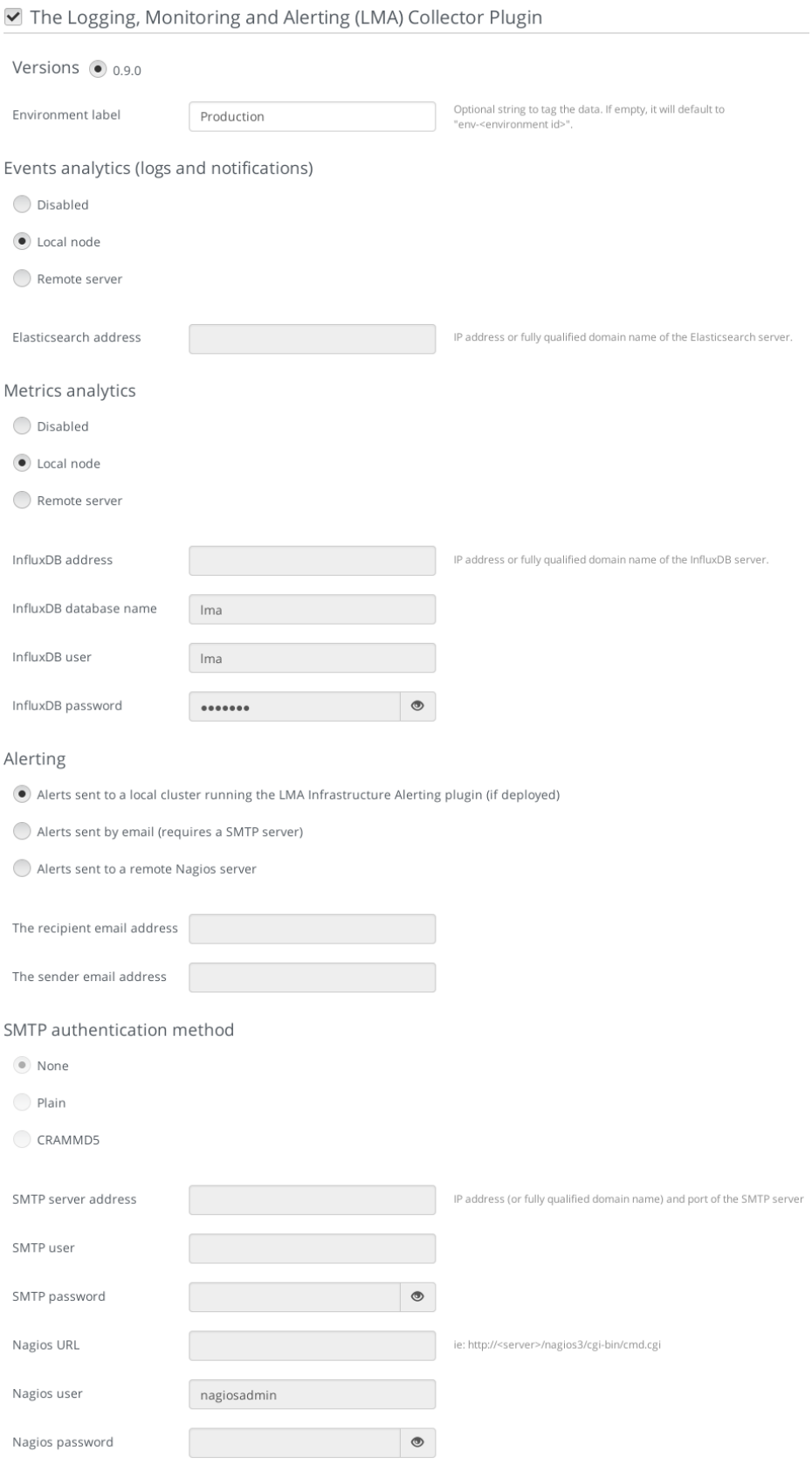
CHAPTER 4

Configuration Guide

4.1 Plugin configuration

To configure your plugin, you need to follow these steps:

1. Create a new environment with the Fuel web user interface.
2. Click on the 'Settings' tab of the Fuel web UI and select the 'Other' category.
3. Scroll down through the settings until you find the 'The Logging, Monitoring and Alerting (LMA) Collector Plugin' section. You should see a page like this.


The Logging, Monitoring and Alerting (LMA) Collector Plugin
Versions 0.9.0
Environment label
Production
Optional string to tag the data. If empty, it will default to "env-<environment id>".
Events analytics (logs and notifications)
Disabled
Local node
Remote server
Elasticsearch address
IP address or fully qualified domain name of the Elasticsearch server.
Metrics analytics
Disabled
Local node
Remote server
InfluxDB address
IP address or fully qualified domain name of the InfluxDB server.
InfluxDB database name
lma
InfluxDB user
lma
InfluxDB password
Alerting
Alerts sent to a local cluster running the LMA Infrastructure Alerting plugin (if deployed)
Alerts sent by email (requires a SMTP server)
Alerts sent to a remote Nagios server
The recipient email address
The sender email address
SMTP authentication method
None
Plain
CRAMMD5
SMTP server address
IP address (or fully qualified domain name) and port of the SMTP server
SMTP user
SMTP password
Nagios URL
ie: http://<server>/nagios3/cgi-bin/cmd.cgi
Nagios user
nagiosadmin
Nagios password

4. Tick the 'The Logging, Monitoring and Alerting (LMA) Collector Plugin' box and fill-in the required fields as indicated below.

1. Provide an 'Environment Label' of your choice to tag your data (optional).
2. For the 'Events Analytics' destination, select 'Local node' if you plan to use the Elasticsearch-Kibana Plugin in this environment. Otherwise, select 'Remote server' and specify the fully qualified name or IP address of an external Elasticsearch server.
3. For the 'Metrics Analytics' destination, select 'Local node' if you plan to use the InfluxDB-Grafana Plugin in this environment. Otherwise, select 'Remote server' and specify the fully qualified name or IP address of an external InfluxDB server. Then, specify the InfluxDB database name you want to use, a username and password that has read and write access permissions.
4. For 'Alerting', select 'Alerts sent by email' if you want to receive alerts sent by email from the Collector. Otherwise, select 'Alerts sent to a local cluster' if you plan to use the Infrastructure Alerting Plugin in this environment. Alternatively, you can select 'Alerts sent to a remote Nagios server'.
5. For 'Alerts sent by email', you can specify the SMTP authentication method you want to use. Then, specify the SMTP server fully qualified name or IP address, the SMTP username and password who have the permissions to send emails.
6. Finally, specify the Nagios server URL, username and password if you have chosen to send alerts to an external Nagios server.
5. Configure your environment as needed.
6. Assign roles to the nodes for the environment.
7. Verify networks on the Networks tab of the Fuel web UI.
8. Deploy your changes.

Note: The *LMA Collector Plugin* is a *hot-pluggable* plugin which means that it is possible to deploy the *LMA Collector* in an environment that is already deployed. To deploy the *LMA Collector* in an environment that is already deployed, you need to run the command below from the *Fuel master node*, for every OpenStack node of the current deployment:

```
[root@nailgun ~]# fuel nodes --env <env_id> --node <node_id> --deploy
```

4.2 Plugin verification

Once the OpenStack environment is ready, you may want to check that both the 'collectd' and 'hekad' processes of the LMA Collector are running on the OpenStack nodes:

```
[root@node-1 ~]# pidof hekad
5568
[root@node-1 ~]# pidof collectd
5684
```

4.3 Troubleshooting

If you see no data in the Kibana and/or Grafana dashboards, use the instructions below to troubleshoot the problem:

1. Check if the LMA Collector service is up and running:

```
# On the controller node(s)
[root@node-1 ~]# crm resource status lma_collector

# On non controller nodes
[root@node-1 ~]# status lma_collector
```

2. If the LMA Collector is down, restart it:

```
# On the controller node(s)
[root@node-1 ~]# crm resource start lma_collector

# On non controller nodes
[root@node-1 ~]# start lma_collector
```

3. Look for errors in the LMA Collector log file (located at /var/log/lma_collector.log) on the different nodes.
4. Look for errors in the collectd log file (located at /var/log/collectd.log) on the different nodes.
5. Check if the nodes are able to connect to the Elasticsearch server on port 9200.
6. Check if the nodes are able to connect to the InfluxDB server on port 8086.

CHAPTER 5

Alarms Configuration Guide

5.1 Overview

The process of running alarms in LMA is not centralized (as it is often the case in conventional monitoring systems) but distributed across all the Collectors.

Each Collector is individually responsible for monitoring the resources and the services that are deployed on the node and for reporting any anomaly or fault it may have detected to the *Aggregator*.

The anomaly and fault detection logic in LMA is designed more like an "Expert System" in that the Collector and the Aggregator use *facts* and *rules* that are executed within the Heka's stream processing pipeline.

The *facts* are the messages ingested by the Collector into the Heka pipeline. The rules are either threshold monitoring alarms or aggregation and correlation rules. Both are declaratively defined in YAML(tm) files that can be modified. Those rules are executed by a collection of Heka filter plugins written in Lua organised according to a configurable processing workflow.

These plugins are called *AFD plugins* for Anomaly and Fault Detection plugins and *GSE plugins* for Global Status Evaluation plugins.

Both the AFD and GSE plugins create metrics called respectively the *AFD metrics* and the *GSE metrics*.

Fig. 5.1: Message flow for the AFD and GSE metrics

The *AFD metrics* contain information about the health status of a resource such as a device, a system component like a filesystem, or service like an API endpoint, at the node level. Then, those *AFD metrics* are sent on a regular basis by each Collector to the Aggregator where they can be aggregated and correlated hence the name 'aggregator'.

The *GSE metrics* contain information about the health status of a service cluster, such as the Nova API endpoints cluster, or the RabbitMQ cluster as well as the clusters of nodes, like the Compute cluster or Controller cluster. The health status of a cluster is inferred by the GSE plugins using aggregation and correlation rules and facts contained in the *AFD metrics* it received from the Collectors.

In the current version of the LMA Toolchain, there are three *GSE plugins*:

- The Service Cluster GSE which receives metrics from the AFD plugins monitoring the services and emits health status for the clusters of services (nova-api, nova-scheduler and so forth).
- The Node Cluster GSE which receives metrics from the AFD plugins monitoring the system and emits health status for the clusters of nodes (controllers, computes and so forth).
- The Global Cluster GSE which receives metrics from the two other GSE plugins and emits health status for the top-level clusters (Nova, MySQL and so forth).

The meaning for each health status is as follow:

- **Down**: One or several primary functions of a cluster has failed or is failing. For example, the API service for Nova or Cinder isn't accessible.
- **Critical**: One or several primary functions of a cluster are severely degraded. The quality of service delivered to the end-user should be severely impacted.
- **Warning**: One or several primary functions of the cluster are slightly degraded. The quality of service delivered to the end-user should be slightly impacted.
- **Unknown**: There is not enough data to infer the actual health state of the cluster.
- **Okay**: None of the above was found to be true.

The *AFD and GSE metrics* are also consumed by other groups of Heka plugins called the *Persisters*.

- A *Persister* for InfluxDB turns the *GSE metric* messages into InfluxDB data-points and Grafana annotations. They are displayed in Grafana dashboards to represent the health status of the OpenStack services and clusters.
- A *Persister* for Elasticsearch turns the *AFD metrics* messages into AFD events which are indexed in Elasticsearch to be able to search and display the faults and anomalies that occured in the OpenStack environment.
- A *Persister* for Nagios turns the *GSE metrics* messages into passive checks that are sent to Nagios which in turn will send alert notifications when there is a change of state for the services and clusters being monitored.

The *AFD metrics* and *GSE metrics* are new types of metrics introduced in LMA v 0.8. They contain detailed information about the entities being monitored. Please refer to the Metrics section of the Developer Guide for further information about the structure of those messages.

Any backend system that has a *Persister* plugged into the Heka pipeline of the Aggregator can consume those metrics. The idea is to feed those systems with rich operational insights about how OpenStack is operating at scale.

5.2 Alarm Configuration

The LMA Toolchain comes out-of-the-box with predefined alarm and correlation rules. We have tried to make the alarm rules comprehensive and relevant enough to cover the most common use cases, but it is possible that your mileage varies depending on the specifics of your environment and monitoring requirements. It is obviously possible to modify the alarm rules or even create new ones. In this case, you will be required to modify the alarm rules configuration file and reapply the Puppet module that will turn the alarm rules into Lua code on each of the nodes you want the change to take effect. This procedure is explained below but first you need to know how the alarm rule structure is defined.

5.2.1 Alarm Structure

An alarm rule is defined declaratively using the YAML syntax as shown in the example below:

```
name: 'fs-warning'
description: 'Filesystem free space is low'
severity: 'warning'
enabled: 'true'
trigger:
  rules:
    - metric: fs_space_percent_free
      fields:
        fs: '*'
      relational_operator: '<'
      threshold: 5
```

```
      window: 60
      periods: 0
      function: min
```

Where

name:
Type: unicode
The name of the alarm definition

description:
Type: unicode
A description of the alarm definition for humans

severity:
Type: Enum(0 (down), 1 (critical) , 2 (warning))
The severity of the alarm

enabled:
Type: Enum('true' | 'false')
The alarm is enabled or disabled

relational_operator:
Type: Enum('lt' | '<' | 'gt' | '>' | 'lte' | '<=' | 'gte' | '>=')
The comparison against the alarm threshold

rules
Type: list
List of rules to execute

logical_operator
Type: Enum('and' | '&&' | 'or' | '||')
The conjonction relation for the alarm rules.

metric
Type: unicode
The name of the metric

value
Type: unicode
The value of the metric

fields

Type: list

List of field name / value pairs (a.k.a dimensions) used to select a particular device for the metric such as a network interface name or file system mount point. If value is specified as an empty string (“”), then the rule is applied to all the aggregated values for the specified field name. For example the file system mount point. If value is specified as the ‘*’ wildcard character, then the rule is applied to each of the metrics matching the metric name and field name. For example, the alarm definition sample given above would run the rule for each of the file system mount points associated with the *fs_space_percent_free* metric.

window

Type: integer

The in memory time-series analysis window in seconds

periods

Type: integer

The number of prior time-series analysis window to compare the window with (this is not implemented yet)

function

Type: enum(‘last’ | ‘min’ | ‘max’ | ‘sum’ | ‘count’ | ‘avg’ | ‘median’ | ‘mode’ | ‘roc’ | ‘mww’ | ‘mww_nonparametric’)

Where:

last:
: returns the last value of all the values

min:
: returns the minimum of all the values

max:
: returns the maximum of all the values

sum:
: returns the sum of all the values

count:
: returns the number of metric observations

avg:
: returns the arithmetic mean of all the values

median:
: returns the middle value of all the values (not implemented yet)

mode:
: returns the value that occurs most often in all the values (not implemented yet)

roc:
: The ‘roc’ function detects a significant rate of change when comparing current metrics values with historical data. To achieve this, it computes the average of the values in the current window, and the average of the values in the window before the current window and compare the difference against the standard deviation of the historical window. The function returns true if the difference exceeds the standard deviation multiplied by the ‘threshold’ value. This function uses the rate of change algorithm already available in the anomaly detection module of Heka. It can only be applied on normal distributions. With an alarm rule using the ‘roc’ function, the ‘window’

parameter specifies the duration in seconds of the current window and the 'periods' parameter specifies the number of windows used for the historical data. You need at least one period and so, the 'periods' parameter must not be zero. If you choose a period of 'p', the function will compute the rate of change using an historical data window of ('p' * window) seconds. For example, if you specify in the alarm rule:

window = 60
periods = 3
threshold = 1.5

The function will store in a circular buffer the value of the metrics received during the last 300 seconds (5 minutes) where:

Current window (CW) = 60 sec
Previous window (PW) = 60 sec
Historical window (HW) = 180 sec

And apply the following formula:

abs(avg(CW) - avg(PW)) > std(HW) * 1.5 ? true : false

mww:
: returns the result (true, false) of the Mann-Whitney-Wilcoxon test function of Heka that can be used only with normal distributions (not implemented yet)

mww-nonparametric:
: returns the result (true, false) of the Mann-Whitney-Wilcoxon test function of Heka that can be used with non-normal distributions (not implemented yet)

diff:
: returns the difference between the last value and the first value of all the values

threshold
: Type: float
The threshold of the alarm rule

5.2.2 How to modify an alarm?

To modify an alarm, you need to edit the */etc/hiera/override/alarming.yaml* file. This file has three different sections:

- The first section contains a list of alarms.
- The second section defines the mapping between the internal definition of a cluster and one or several Fuel roles. The definition of a cluster is abstrat. It can be mapped to any Fuel role(s). In the example below, we define three clusters for:
  - controller,
  - compute,
  - and storage

- The third section defines how the alarms are assigned to clusters. In the example below, the *controller* cluster is assigned to four alarms:
  - Two alarms ['cpu-critical-controller', 'cpu-warning-controller'] grouped as *system* alarms.
  - Two alarms ['fs-critical', 'fs-warning'] grouped as *fs* (file system) alarms.

Note: The alarm groups is a mere implementaton artifact (although it has some practicall usefulness) that is used to divide the workload across several Lua plugins. Since the Lua plugins runtime is sandboxed within Heka, it is preferable to run smaller sets of alarms in different plugins rather than a large set of alarms in a single plugin. This is to avoid having plugins shut down by Heka because they use too much CPU or memory. Furthermore, the alarm groups are used to identify what we call a *source*. A *source* is defined by a tuple which includes the name of the cluster and the name of the alarm group. For example the tuple ['controller', 'system'] identifies a *source*. The tuple ['controller', 'fs'] identifies another *source*. The interesting thing about the *source* is that it is used by the *GSE Plugins* to find out whether it has received enough data (from its 'known' *sources*) to issue a health status or not. If it doesn't, then the *GSE Plugin* will issue a *GSE Metric* with an *Unknown* health status when it has reached the end of the *ticker interval* period. By default, the *ticker interval* for the GSE Plugins is set to 10 seconds. This practically means that every 10 seconds, a GSE Plugin is compelled to send a *GSE Metric* regardless of the metrics it has received from the upstream *GSE Plugins* and/or *AFD Plugins*.

Here is an example of the definition of an alarm and how that alarm is assigned to a cluster:

```
lma_collector:
    #
    # The alarms list
    #
  alarms:
    - name: 'cpu-critical-controller'
      description: 'CPU critical on controller'
      severity: 'critical'
      enabled: 'true'
      trigger:
        logical_operator: 'or'
        rules:
          - metric: cpu_idle
            relational_operator: '<='
            threshold: 5
            window: 120
            periods: 0
            function: avg
          - metric: cpu_wait
            relational_operator: '>='
            threshold: 35
            window: 120
            periods: 0
            function: avg

    [Skip....]

    #
    # Cluster name to roles mapping section
    #
  node_cluster_roles:
    controller: ['primary-controller', 'controller']
    compute: ['compute']
    storage: ['cinder', 'ceph-osd']

    #
    # Cluster name to alarms assignement section
```

```
    #
  node_cluster_alarms:
    controller:
      system: ['cpu-critical-controller', 'cpu-warning-controller']
      fs: ['fs-critical', 'fs-warning']
```

In this example, you can see that the alarm *cpu-critical-controller* is assigned to the *controller* cluster (or in other words) to the nodes assigned to the *primary-controller* or *controller* roles.

This alarm tells the system that any node associated with the *controller* cluster is claimed to be critical (severity: 'critical') if any of the rules in the alarm evaluates to true.

The first rule says that the alarm evaluates to true if the metric *cpu_idle* has been in average (function: avg) below or equal (relational_operator: <=) to 5 (this metric is expressed in percentage) for the last 5 minutes (window: 120)

Or (logical_operator: 'or')

if the metric *cpu_wait* has been in average (function: avg) superior or equal (relational_operator: >=) to 35 (this metric is expressed in percentage) for the last 5 minutes (window: 120)

Once you have edited and saved the */etc/hiera/override/alarming.yaml* file, you need to re-apply the Puppet module:

```
# puppet apply --modulepath=/etc/fuel/plugins/lma_collector-0.9/puppet/modules/ \
/etc/fuel/plugins/lma_collector-0.9/puppet/manifests/configure_afd_filters.pp
```

This will restart the LMA Collector with your change.

5.3 GSE configuration

The LMA toolchain comes with a predefined configuration for the GSE plugins. As for the alarms, it is possible to modify this configuration.

The GSE plugins are defined declaratively in the */etc/hiera/override/gse_filters.yaml* file. By default, that file specifies three kinds of GSE plugins:

- *gse_cluster_service* for the Service Cluster GSE which evaluates the status of the service clusters.
- *gse_cluster_node* for the Node Cluster GSE which evaluates the status of the node clusters.
- *gse_cluster_global* for the Global Cluster GSE which evaluates the status of the global clusters.

The structure of a GSE plugin declarative definition is described below:

```
gse_cluster_service:
  input_message_types:
    - afd_service_metric
  aggregator_flag: true
  cluster_field: service
  member_field: source
  output_message_type: gse_service_cluster_metric
  output_metric_name: cluster_service_status
  interval: 10
  warm_up_period: 20
  clusters:
    nova-api:
      policy: highest_severity
      group_by: member
      members:
        - backends
        - endpoint
```

```
        - http_errors
    [...]
```

Where

input_message_types
: Type: list
The type(s) of AFD metric messages consumed by the GSE plugin.

aggregator_flag
: Type: boolean
Whether or not the input messages are received from the upstream collectors. This is true for the Service and Node Cluster plugins and false for the Global Cluster plugin.

cluster_field
: Type: unicode
The field in the input message used by the GSE plugin to associate the AFD/GSE metrics to the clusters.

member_field
: Type: unicode
The field in the input message used by the GSE plugin to identify the cluster members.

output_message_type
: Type: unicode
The type of metric messages emitted by the GSE plugin.

output_metric_name
: Type: unicode
The Fields[name] value of the metric messages that the GSE plugin emits.

interval
: Type: integer
The interval (in seconds) at which the GSE plugin emits its metric messages.

warm_up_period
: Type: integer
The number of seconds after a (re)start that the GSE plugin will wait before emitting its metric messages.

clusters
: Type: list
The list of clusters that the plugin manages. See *Cluster definition* for details.

5.3.1 Cluster definition

The GSE clusters are defined as shown in the example below:

```
gse_cluster_service:
  [...]

  clusters:
    nova-api:
      members:
        - backends
        - endpoint
        - http_errors
      group_by: member
      policy: highest_severity

  [...]
```

Where

members
: Type: list
The list of cluster members. The AFD/GSE messages are associated to the cluster when the *cluster_field* value is equal to the cluster name and the *member_field* value is in this list.

group_by
: Type: Enum(member, hostname)
This parameter defines how the incoming AFD metrics are aggregated.

  member:
  : aggregation by member, irrespective of the host that emitted the AFD metric.
  This setting is typically used for AFD metrics that are not host-centric.

  hostname:
  : aggregation by hostname then by member.
  This setting is typically used for AFD metrics that are host-centric such as
  those working on filesystem or CPU usage metrics.

policy:
: Type: unicode
The policy to use for computing the cluster status. See *Cluster policies* for details.

If we look more closely at the example above, it defines that the Service Cluster GSE plugin will emit a *gse_service_cluster_metric* message every 10 seconds that will report the current status of the *nova-api* cluster. This status is computed using the *afd_service_metric* metrics for which Fields[service] is 'nova-api' and Fields[source] is one of 'backends', 'endpoint' or 'http_errors'. The 'nova-api' cluster's status is computed using the 'highest_severity' policy which means that it will be equal to the 'worst' status across all members.

5.3.2 Cluster policies

The correlation logic implemented by the GSE plugins is policy-based. The cluster policies define how the GSE plugins infer the health status of a cluster.

By default, two policies are defined:

- *highest_severity*, it defines that the cluster's status depends on the member with the highest severity, typically used for a cluster of services.
- *majority_of_members*, it defines that the cluster is healthy as long as (N+1)/2 members of the cluster are healthy. This is typically used for clusters managed by Pacemaker.

The GSE policies are defined declaratively in the */etc/hiera/override/gse_filters.yaml* file at the *gse_policies* entry.

A policy consists of a list of rules which are evaluated against the current status of the cluster's members. When one of the rules matches, the cluster's status gets the value associated with the rule and the evaluation stops here. The last rule of the list is usually a catch-all rule that defines the default status in case none of the previous rules could be matched.

A policy rule is defined as shown in the example below:

```
# The following rule definition reads as: "the cluster's status is critical
# if more than 50% of its members are either down or criticial"
- status: critical
  trigger:
    logical_operator: or
    rules:
      - function: percent
        arguments: [ down, critical ]
        relational_operator: '>'
        threshold: 50
```

Where

status:
: Type: Enum(down, critical, warning, okay, unknown)
  The cluster's status if the condition is met

logical_operator
: Type: Enum('and' | '&&' | 'or' | '||')
  The conjonction relation for the condition rules

rules
: Type: list
  List of condition rules to execute

function
: Type: enum('count' | 'percent')
  Where:
  count:
  : returns the *number of members* that match the passed value(s).

  percent:
  : returns the *percentage of members* that match the passed value(s).

arguments:
: Type: list of status values
List of status values passed to the function

relational_operator:
: Type: Enum('lt' | '<' | 'gt' | '>' | 'lte' | '<=' | 'gte' | '>=')
The comparison against the threshold

threshold
: Type: float
The threshold value

Lets now take a more detailed look at the policy called *highest_severity*:

```
gse_policies:

  highest_severity:
    - status: down
      trigger:
        logical_operator: or
        rules:
          - function: count
            arguments: [ down ]
            relational_operator: '>'
            threshold: 0
    - status: critical
      trigger:
        logical_operator: or
        rules:
          - function: count
            arguments: [ critical ]
            relational_operator: '>'
            threshold: 0
    - status: warning
      trigger:
        logical_operator: or
        rules:
          - function: count
            arguments: [ warning ]
            relational_operator: '>'
            threshold: 0
    - status: okay
      trigger:
        logical_operator: or
        rules:
          - function: count
            arguments: [ okay ]
            relational_operator: '>'
            threshold: 0
    - status: unknown
```

The policy definition reads as:

- The status of the cluster is *Down* if the status of at least one cluster's member is *Down*.

- Otherwise the status of the cluster is *Critical* if the status of at least one cluster's member is *Critical*.
- Otherwise the status of the cluster is *Warning* if the status of at least one cluster's member is *Warning*.
- Otherwise the status of the cluster is *Okay* if the status of at least one cluster's entity is *Okay*.
- Otherwise the status of the cluster is *Unknown*.

CHAPTER 6

Licenses

6.1 Third Party Components

Name	Project Web Site	License
Heka	https://github.com/mozilla-services/heka	Mozilla Public License
collectd	http://collectd.org/	GPLv2
Collectd::CPU	http://collectd.org/	GPLv2
Collectd::Disk	http://collectd.org/	GPLv2
Collectd::Df	http://collectd.org/	GPLv2
Collectd::Interface	http://collectd.org/	GPLv2
Collectd::Load	http://collectd.org/	GPLv2
Collectd::Memory	http://collectd.org/	GPLv2
Collectd::Processes	http://collectd.org/	GPLv2 or later
Collectd::Swap	http://collectd.org/	GPLv2
Collectd::User	http://collectd.org/	GPLv2
Collectd::LogFile	http://collectd.org/	GPLv2
Collectd::User	http://collectd.org/	GPLv2
Collectd::WriteHttp	http://collectd.org/	GPLv2
Collectd::MySQL	http://collectd.org/	GPLv2
Collectd::DBI	http://collectd.org/	GPLv2
Collectd::Apache	http://collectd.org/	GPLv2
Collectd::Python	http://collectd.org/	MIT
Collectd::Python::RabbitMQ	http://collectd.org/	Apache v2
Collectd::Python::HAProxy	http://collectd.org/	Permissive

6.2 Puppet modules

Name	Project Web Site	License
puppet-collectd	https://github.com/puppet-community/puppet-collectd	Apache v2
puppetlabs-apache	https://github.com/puppetlabs/puppetlabs-apache	Apache v2
puppetlabs-stdlib	https://github.com/puppetlabs/puppetlabs-stdlib	Apache v2
puppetlabs-inifile	https://github.com/puppetlabs/puppetlabs-inifile	Apache v2
puppetlabs-concat	https://github.com/puppetlabs/puppetlabs-concat	Apache v2
puppetlabs-firewall	https://github.com/puppetlabs/puppetlabs-firewall	Apache v2
openstack-cinder	https://github.com/openstack/puppet-cinder	Apache v2
openstack-glance	https://github.com/openstack/puppet-glance	Apache v2
openstack-heat	https://github.com/openstack/puppet-heat	Apache v2
openstack-keystone	https://github.com/openstack/puppet-keystone	Apache v2
openstack-neutron	https://github.com/openstack/puppet-neutron	Apache v2
openstack-nova	https://github.com/openstack/puppet-nova	Apache v2
openstack-openstacklib	https://github.com/openstack/puppet-openstacklib	Apache v2

CHAPTER 7

Appendix A: References

- The LMA Collector plugin project at GitHub.
- The Elasticsearch-Kibana plugin project at GitHub.
- The InfluxDB-Grafana plugin project at GitHub.
- The LMA Infrastructure Alerting plugin project at GitHub.
- The official Kibana documentation.
- The official Elasticsearch documentation.
- The official InfluxDB documentation.
- The official Grafana documentation.
- The official Nagios documentation.

CHAPTER 8

Appendix B: List of metrics

This is the list of metrics that are emitted by the LMA collector service. They are listed by category then by metric name.

8.1 System

8.1.1 CPU

Metrics have a `cpu_number` field that contains the CPU number to which the metric applies.

- `cpu_idle`, percentage of CPU time spent in the idle task.
- `cpu_interrupt`, percentage of CPU time spent servicing interrupts.
- `cpu_nice`, percentage of CPU time spent in user mode with low priority (nice).
- `cpu_softirq`, percentage of CPU time spent servicing soft interrupts.
- `cpu_steal`, percentage of CPU time spent in other operating systems.
- `cpu_system`, percentage of CPU time spent in system mode.
- `cpu_user`, percentage of CPU time spent in user mode.
- `cpu_wait`, percentage of CPU time spent waiting for I/O operations to complete.

8.1.2 Disk

Metrics have a `device` field that contains the disk device number the metric applies to (eg 'sda', 'sdb' and so on).

- `disk_merged_read`, the number of read operations per second that could be merged with already queued operations.
- `disk_merged_write`, the number of write operations per second that could be merged with already queued operations.
- `disk_octets_read`, the number of octets (bytes) read per second.
- `disk_octets_write`, the number of octets (bytes) written per second.
- `disk_ops_read`, the number of read operations per second.
- `disk_ops_write`, the number of write operations per second.
- `disk_time_read`, the average time for a read operation to complete in the last interval.

- disk_time_write, the average time for a write operation to complete in the last interval.

8.1.3 File system

Metrics have a fs field that contains the partition's mount point to which the metric applies (eg '/', '/var/lib' and so on).

- fs_inodes_free, the number of free inodes on the file system.
- fs_inodes_reserved, the number of reserved inodes.
- fs_inodes_used, the number of used inodes.
- fs_space_free, the number of free bytes.
- fs_space_reserved, the number of reserved bytes.
- fs_space_used, the number of used bytes.
- fs_inodes_percent_free, the percentage of free inodes on the file system.
- fs_inodes_percent_reserved, the percentage of reserved inodes.
- fs_inodes_percent_used, the percentage of used inodes.
- fs_space_percent_free, the percentage of free bytes.
- fs_space_percent_reserved, the percentage of reserved bytes.
- fs_space_percent_used, the percentage of used bytes.

8.1.4 System load

- load_longterm, the system load average over the last 15 minutes.
- load_midterm, the system load average over the last 5 minutes.
- load_shortterm, the system load averge over the last minute.

8.1.5 Memory

- memory_buffered, the amount of memory (in bytes) which is buffered.
- memory_cached, the amount of memory (in bytes) which is cached.
- memory_free, the amount of memory (in bytes) which is free.
- memory_used, the amount of memory (in bytes) which is used.

8.1.6 Network

Metrics have a interface field that contains the interface name the metric applies to (eg 'eth0', 'eth1' and so on).

- if_errors_rx, the number of errors per second detected when receiving from the interface.
- if_errors_tx, the number of errors per second detected when transmitting from the interface.
- if_octets_rx, the number of octets (bytes) received per second by the interface.
- if_octets_tx, the number of octets (bytes) transmitted per second by the interface.

- `if_packets_rx`, the number of packets received per second by the interface.
- `if_packets_tx`, the number of packets transmitted per second by the interface.

8.1.7 Processes

- `processes_fork_rate`, the number of processes forked per second.
- `processes_count`, the number of processes in a given state. The metric has a `state` field (one of 'blocked', 'paging', 'running', 'sleeping', 'stopped' or 'zombies').

8.1.8 Swap

- `swap_cached`, the amount of cached memory (in bytes) which is in the swap.
- `swap_free`, the amount of free memory (in bytes) which is in the swap.
- `swap_used`, the amount of used memory (in bytes) which is in the swap.
- `swap_io_in`, the number of swap pages written per second.
- `swap_io_out`, the number of swap pages read per second.

8.1.9 Users

- `logged_users`, the number of users currently logged-in.

8.2 Apache

- `apache_bytes`, the number of bytes per second transmitted by the server.
- `apache_requests`, the number of requests processed per second.
- `apache_connections`, the current number of active connections.
- `apache_idle_workers`, the current number of idle workers.
- `apache_workers_closing`, the number of workers in closing state.
- `apache_workers_dnslookup`, the number of workers in DNS lookup state.
- `apache_workers_finishing`, the number of workers in finishing state.
- `apache_workers_idle_cleanup`, the number of workers in idle cleanup state.
- `apache_workers_keepalive`, the number of workers in keepalive state.
- `apache_workers_logging`, the number of workers in logging state.
- `apache_workers_open`, the number of workers in open state.
- `apache_workers_reading`, the number of workers in reading state.
- `apache_workers_sending`, the number of workers in sending state.
- `apache_workers_starting`, the number of workers in starting state.
- `apache_workers_waiting`, the number of workers in waiting state.

8.3 MySQL

8.3.1 Commands

`mysql_commands`, the number of times per second a given statement has been executed. The metric has a `command` field that contains the statement to which it applies. The values can be:

- `change_db` for the USE statement.
- `commit` for the COMMIT statement.
- `flush` for the FLUSH statement.
- `insert` for the INSERT statement.
- `rollback` for the ROLLBACK statement.
- `select` for the SELECT statement.
- `set_option` for the SET statement.
- `show_collations` for the SHOW COLLATION statement.
- `show_databases` for the SHOW DATABASES statement.
- `show_fields` for the SHOW FIELDS statement.
- `show_master_status` for the SHOW MASTER STATUS statement.
- `show_status` for the SHOW STATUS statement.
- `show_tables` for the SHOW TABLES statement.
- `show_variables` for the SHOW VARIABLES statement.
- `show_warnings` for the SHOW WARNINGS statement.
- `update` for the UPDATE statement.

8.3.2 Handlers

`mysql_handler`, the number of times per second a given handler has been executed. The metric has a `handler` field that contains the handler it applies to. The values can be:

- `commit` for the internal COMMIT statements.
- `delete` for the internal DELETE statements.
- `external_lock` for the external locks.
- `read_first` for the requests that read the first entry in an index.
- `read_key` for the requests that read a row based on a key.
- `read_next` for the requests that read the next row in key order.
- `read_prev` for the requests that read the previous row in key order.
- `read_rnd` for the requests that read a row based on a fixed position.
- `read_rnd_next` for the requests that read the next row in the data file.
- `rollback` the requests that perform rollback operation.
- `update` the requests that update a row in a table.

- `write` the requests that insert a row in a table.

8.3.3 Locks

- `mysql_locks_immediate`, the number of times per second the requests for table locks could be granted immediately.
- `mysql_locks_waited`, the number of times per second the requests for table locks had to wait.

8.3.4 Network

- `mysql_octets_rx`, the number of bytes received per second by the server.
- `mysql_octets_tx`, the number of bytes sent per second by the server.

8.3.5 Threads

- `mysql_threads_cached`, the number of threads in the thread cache.
- `mysql_threads_connected`, the number of currently open connections.
- `mysql_threads_running`, the number of threads that are not sleeping.
- `mysql_threads_created`, the number of threads created per second to handle connections.

8.3.6 Cluster

These metrics are collected with statement 'SHOW STATUS'. see Percona documentation for further details.

- `mysql_cluster_size`, current number of nodes in the cluster.
- `mysql_cluster_status`, `1` when the node is 'Primary', `2` if 'Non-Primary' and `3` if 'Disconnected'.
- `mysql_cluster_connected`, `1` when the node is connected to the cluster, if not `0`.
- `mysql_cluster_ready`, `1` when the node is ready to accept queries, if not `0`.
- `mysql_cluster_local_commits`, number of writesets commited on the node.
- `mysql_cluster_received_bytes`, total size in bytes of writesets received from other nodes.
- `mysql_cluster_received`, total number of writesets received from other nodes.
- `mysql_cluster_replicated_bytes` total size in bytes of writesets sent to other nodes.
- `mysql_cluster_replicated`, total number of writesets sent to other nodes.
- `mysql_cluster_local_cert_failures`, number of writesets that failed the certification test.
- `mysql_cluster_local_send_queue`, the number of writesets waiting to be sent.
- `mysql_cluster_local_recv_queue`, the number of writesets waiting to be applied.

8.3.7 Slow Queries

This metric is collected with statement 'SHOW STATUS where Variable_name = 'Slow_queries'.

- `mysql_slow_queries`, number of queries that have taken more than X seconds, depending of the MySQL configuration parameter 'long_query_time' (10s per default)

8.4 RabbitMQ

8.4.1 Cluster

- `rabbitmq_connections`, total number of connections.
- `rabbitmq_consumers`, total number of consumers.
- `rabbitmq_exchanges`, total number of exchanges.
- `rabbitmq_memory`, bytes of memory consumed by the Erlang process associated with all queues, including stack, heap and internal structures.
- `rabbitmq_used_memory`, bytes of memory used by the whole RabbitMQ process.
- `rabbitmq_remaining_memory`, the difference between `rabbitmq_vm_memory_limit` and `rabbitmq_used_memory`.
- `rabbitmq_messages`, total number of messages which are ready to be consumed or not yet acknowledged.
- `rabbitmq_total_nodes`, total number of nodes in the cluster.
- `rabbitmq_running_nodes`, total number of running nodes in the cluster.
- `rabbitmq_queues`, total number of queues.
- `rabbitmq_unmirrored_queues`, total number of queues that are not mirrored.
- `rabbitmq_vm_memory_limit`, the maximum amount of memory allocated for RabbitMQ. When `rabbitmq_used_memory` uses more than this value, all producers are blocked.
- `rabbitmq_disk_free_limit`, the minimum amount of free disk for RabbitMQ. When `rabbitmq_disk_free` drops below this value, all producers are blocked.
- `rabbitmq_disk_free`, the disk free space.
- `rabbitmq_remaining_disk`, the difference between `rabbitmq_disk_free` and `rabbitmq_disk_free_limit`.

8.4.2 Queues

All metrics have a `queue` field which contains the name of the RabbitMQ queue.

- `rabbitmq_queue_consumers`, number of consumers for a given queue.
- `rabbitmq_queue_memory`, bytes of memory consumed by the Erlang process associated with the queue, including stack, heap and internal structures.
- `rabbitmq_queue_messages`, number of messages which are ready to be consumed or not yet acknowledged for the given queue.

8.5 HAProxy

`frontend` and `backend` field values can be:

- cinder-api
- glance-api
- glance-registry-api

- heat-api
- heat-cfn-api
- heat-cloudwatch-api
- horizon-web (when Horizon is deployed without TLS)
- horizon-https (when Horizon is deployed with TLS)
- keystone-public-api
- keystone-admin-api
- mysqld-tcp
- murano-api
- neutron-api
- nova-api
- nova-ec2-api
- nova-metadata-api
- nova-novncproxy-websocket
- sahara-api
- swift-api

8.5.1 Server

- `haproxy_connections`, the number of current connections.
- `haproxy_ssl_connections`, the number of current SSL connections.
- `haproxy_pipes_free`, the number of free pipes.
- `haproxy_pipes_used`, the number of used pipes.
- `haproxy_run_queue`, the number of connections waiting in the queue.
- `haproxy_tasks`, the number of tasks.
- `haproxy_uptime`, the HAProxy server uptime in seconds.

8.5.2 Frontends

- `haproxy_frontend_bytes_in`, the total number of bytes received by all frontends.
- `haproxy_frontend_bytes_out`, the total number of bytes transmitted by all frontends.
- `haproxy_frontend_session_current`, the total number of current sessions for all frontends.

The following metrics have a `frontend` field that contains the name of the frontend server.

- `haproxy_frontend_bytes_in`, the number of bytes received by the frontend.
- `haproxy_frontend_bytes_out`, the number of bytes transmitted by the frontend.
- `haproxy_frontend_denied_requests`, the number of denied requests.
- `haproxy_frontend_denied_responses`, the number of denied responses.
- `haproxy_frontend_error_requests`, the number of error requests.

- haproxy_frontend_response_1xx, the number of HTTP responses with 1xx code.
- haproxy_frontend_response_2xx, the number of HTTP responses with 2xx code.
- haproxy_frontend_response_3xx, the number of HTTP responses with 3xx code.
- haproxy_frontend_response_4xx, the number of HTTP responses with 4xx code.
- haproxy_frontend_response_5xx, the number of HTTP responses with 5xx code.
- haproxy_frontend_response_other, the number of HTTP responses with other code.
- haproxy_frontend_session_current, the number of current sessions.
- haproxy_frontend_session_total, the cumulative number of sessions.

8.5.3 Backends

- haproxy_backend_bytes_in, the total number of bytes received by all backends.
- haproxy_backend_bytes_out, the total number of bytes transmitted by all backends.
- haproxy_backend_queue_current, the total number of requests in queue for all backends.
- haproxy_backend_session_current, the total number of current sessions for all backends.
- haproxy_backend_error_responses, the total number of error responses for all backends.

The following metrics have a backend field that contains the name of the backend server.

- haproxy_backend_bytes_in, the number of bytes received by the backend.
- haproxy_backend_bytes_out, the number of bytes transmitted by the backend.
- haproxy_backend_denied_requests, the number of denied requests.
- haproxy_backend_denied_responses, the number of denied responses.
- haproxy_backend_downtime, the total downtime in second.
- haproxy_backend_status, the global backend status where values 0 and 1 represent respectively DOWN (all backends are down) and UP (at least one backend is up).
- haproxy_backend_error_connection, the number of error connections.
- haproxy_backend_error_responses, the number of error responses.
- haproxy_backend_queue_current, the number of requests in queue.
- haproxy_backend_redistributed, the number of times a request was redispatched to another server.
- haproxy_backend_response_1xx, the number of HTTP responses with 1xx code.
- haproxy_backend_response_2xx, the number of HTTP responses with 2xx code.
- haproxy_backend_response_3xx, the number of HTTP responses with 3xx code.
- haproxy_backend_response_4xx, the number of HTTP responses with 4xx code.
- haproxy_backend_response_5xx, the number of HTTP responses with 5xx code.
- haproxy_backend_response_other, the number of HTTP responses with other code.
- haproxy_backend_retries, the number of times a connection to a server was retried.
- haproxy_backend_servers, the count of servers grouped by state. This metric has an additional state field that contains the state of the backends (either 'down' or 'up').
- haproxy_backend_session_current, the number of current sessions.

- haproxy_backend_session_total, the cumulative number of sessions.

8.6 Memcached

- memcached_command_flush, cumulative number of flush reqs.
- memcached_command_get, cumulative number of retrieval reqs.
- memcached_command_set, cumulative number of storage reqs.
- memcached_command_touch, cumulative number of touch reqs.
- memcached_connections_current, number of open connections.
- memcached_items_current, current number of items stored.
- memcached_octets_rx, total number of bytes read by this server from network.
- memcached_octets_tx, total number of bytes sent by this server to network.
- memcached_ops_decr_hits, number of successful decr reqs.
- memcached_ops_decr_misses, number of decr reqs against missing keys.
- memcached_ops_evictions, number of valid items removed from cache to free memory for new items.
- memcached_ops_hits, number of keys that have been requested.
- memcached_ops_incr_hits, number of successful incr reqs.
- memcached_ops_incr_misses, number of successful incr reqs.
- memcached_ops_misses, number of items that have been requested and not found.
- memcached_df_cache_used, current number of bytes used to store items.
- memcached_df_cache_free, current number of free bytes to store items.
- memcached_percent_hitratio, percentage of get command hits (in cache).

See memcached documentation for further details.

8.7 Libvirt

Every metric contains an instance_id field which is the UUID of the instance for the Nova service.

8.7.1 CPU

- virt_cpu_time, the average amount of CPU time (in nanoseconds) allocated to the virtual instance in a second.
- virt_vcpu_time, the average amount of CPU time (in nanoseconds) allocated to the virtual CPU in a second. The metric contains a vcpu_number field which is the virtual CPU number.

8.7.2 Disk

Metrics have a `device` field that contains the virtual disk device to which the metric applies (eg 'vda', 'vdb' and so on).

- `virt_disk_octets_read`, the number of octets (bytes) read per second.
- `virt_disk_octets_write`, the number of octets (bytes) written per second.
- `virt_disk_ops_read`, the number of read operations per second.
- `virt_disk_ops_write`, the number of write operations per second.

8.7.3 Memory

- `virt_memory_total`, the total amount of memory (in bytes) allocated to the virtual instance.

8.7.4 Network

Metrics have an `interface` field that contains the interface name to which the metric applies (eg 'tap0dc043a6-dd', 'tap769b123a-2e' and so on).

- `virt_if_dropped_rx`, the number of dropped packets per second when receiving from the interface.
- `virt_if_dropped_tx`, the number of dropped packets per second when transmitting from the interface.
- `virt_if_errors_rx`, the number of errors per second detected when receiving from the interface.
- `virt_if_errors_tx`, the number of errors per second detected when transmitting from the interface.
- `virt_if_octets_rx`, the number of octets (bytes) received per second by the interface.
- `virt_if_octets_tx`, the number of octets (bytes) transmitted per second by the interface.
- `virt_if_packets_rx`, the number of packets received per second by the interface.
- `virt_if_packets_tx`, the number of packets transmitted per second by the interface.

8.8 OpenStack

8.8.1 Service checks

- **`openstack_check_api`, the service's API status, 1 if it is responsive, if not 0.** The metric contains a `service` field that identifies the OpenStack service being checked.

`<service>` is one of the following values with their respective resource checks:

- 'nova-api': '/'
- 'cinder-api': '/'
- 'cinder-v2-api': '/'
- 'glance-api': '/'
- 'heat-api': '/'
- 'heat-cfn-api': '/'
- 'keystone-public-api': '/'

- 'neutron-api': '/'
- 'ceilometer-api': '/v2/capabilities'
- 'swift-api': '/healthcheck'
- 'swift-s3-api': '/healthcheck'

Note: All checks are performed without authentication except for Ceilometer.

8.8.2 Compute

These metrics are emitted per compute node.

- `openstack_nova_instance_creation_time`, the time (in seconds) it took to launch a new instance.
- `openstack_nova_instance_state`, the number of instances which entered a given state (the value is always 1). The metric contains a `state` field.
- `openstack_nova_free_disk`, the disk space (in GB) available for new instances.
- `openstack_nova_used_disk`, the disk space (in GB) used by the instances.
- `openstack_nova_free_ram`, the memory (in MB) available for new instances.
- `openstack_nova_used_ram`, the memory (in MB) used by the instances.
- `openstack_nova_free_vcpus`, the number of virtual CPU available for new instances.
- `openstack_nova_used_vcpus`, the number of virtual CPU used by the instances.
- `openstack_nova_running_instances`, the number of running instances.
- `openstack_nova_running_tasks`, the number of tasks currently executed.

These metrics are retrieved from the Nova API and represent the aggregated values across all compute nodes.

- `openstack_nova_total_free_disk`, the total amount of disk space (in GB) available for new instances.
- `openstack_nova_total_used_disk`, the total amount of disk space (in GB) used by the instances.
- `openstack_nova_total_free_ram`, the total amount of memory (in MB) available for new instances.
- `openstack_nova_total_used_ram`, the total amount of memory (in MB) used by the instances.
- `openstack_nova_total_free_vcpus`, the total number of virtual CPU available for new instances.
- `openstack_nova_total_used_vcpus`, the total number of virtual CPU used by the instances.
- `openstack_nova_total_running_instances`, the total number of running instances.
- `openstack_nova_total_running_tasks`, the total number of tasks currently executed.

These metrics are retrieved from the Nova API.

- `openstack_nova_instances`, the total count of instances in a given state. The metric contains a `state` field which is one of 'active', 'deleted', 'error', 'paused', 'resumed', 'rescued', 'resized', 'shelved_offloaded' or 'suspended'.

These metrics are retrieved from the Nova database.

- `openstack_nova_services`, the total count of Nova services by state. The metric contains a `service` field (one of 'compute', 'conductor', 'scheduler', 'cert' or 'consoleauth') and a `state` field (one of 'up', 'down' or 'disabled').

- `openstack_nova_service`, the Nova service state (either 0 for 'up', 1 for 'down' or 2 for 'disabled'). The metric contains a `service` field (one of 'compute', 'conductor', 'scheduler', 'cert' or 'consoleauth') and a `state` field (one of 'up', 'down' or 'disabled').

8.8.3 Identity

These metrics are retrieved from the Keystone API.

- `openstack_keystone_roles`, the total number of roles.
- `openstack_keystone_tenants`, the number of tenants by state. The metric contains a `state` field (either 'enabled' or 'disabled').
- `openstack_keystone_users`, the number of users by state. The metric contains a `state` field (either 'enabled' or 'disabled').

8.8.4 Volume

These metrics are emitted per volume node.

- `openstack_cinder_volume_creation_time`, the time (in seconds) it took to create a new volume.

Note: When using Ceph as the backend storage for volumes, the `hostname` value is always set to `rbd`.

These metrics are retrieved from the Cinder API.

- `openstack_cinder_volumes`, the number of volumes by state. The metric contains a `state` field.
- `openstack_cinder_snapshots`, the number of snapshots by state. The metric contains a `state` field.
- `openstack_cinder_volumes_size`, the total size (in bytes) of volumes by state. The metric contains a `state` field.
- `openstack_cinder_snapshots_size`, the total size (in bytes) of snapshots by state. The metric contains a `state` field.

`state` is one of 'available', 'creating', 'attaching', 'in-use', 'deleting', 'backing-up', 'restoring-backup', 'error', 'error_deleting', 'error_restoring', 'error_extending'.

These metrics are retrieved from the Cinder database.

- `openstack_cinder_services`, the total count of Cinder services by state. The metric contains a `service` field (one of 'volume', 'backup', 'scheduler') and a `state` field (one of 'up', 'down' or 'disabled').
- `openstack_cinder_service`, the Cinder service state (either 0 for 'up', 1 for 'down' or 2 for 'disabled'). The metric contains a `service` field (one of 'volume', 'backup', 'scheduler'), and a `state` field (one of 'up', 'down' or 'disabled').

8.8.5 Image

These metrics are retrieved from the Glance API.

- `openstack_glance_images`, the number of images by state and visibility. The metric contains `state` and `visibility` field.
- `openstack_glance_snapshots`, the number of snapshot images by state and visibility. The metric contains `state` and `visibility` field.

- `openstack_glance_images_size`, the total size (in bytes) of images by state and visibility. The metric contains `state` and `visibility` field.
- `openstack_glance_snapshots_size`, the total size (in bytes) of snapshots by state and visibility. The metric contains `state` and `visibility` field.

`state` is one of 'queued', 'saving', 'active', 'killed', 'deleted', 'pending_delete'. `visibility` is either 'public' or 'private'.

8.8.6 Network

These metrics are retrieved from the Neutron API.

- `openstack_neutron_networks`, the number of virtual networks by state. The metric contains a `state` field.
- `openstack_neutron_subnets`, the number of virtual subnets.
- `openstack_neutron_ports`, the number of virtual ports by owner and state. The metric contains `owner` and `state` fields.
- `openstack_neutron_routers`, the number of virtual routers by state. The metric contains a `state` field.
- `openstack_neutron_floatingips`, the total number of floating IP addresses.

`<state>` is one of 'active', 'build', 'down' or 'error'.

`<owner>` is one of 'compute', 'dhcp', 'floatingip', 'floatingip_agent_gateway', 'router_interface', 'router_gateway', 'router_ha_interface', 'router_interface_distributed' or 'router_centralized_snat'.

These metrics are retrieved from the Neutron database.

Note: These metrics are not collected when the Contrail plugin is deployed.

- `openstack_neutron_agents`, the total number of Neutron agents by service and state. The metric contains `service` (one of 'dhcp', 'l3', 'metadata' or 'openvswitch') and `state` (one of 'up', 'down' or 'disabled') fields.
- `openstack_neutron_agent`, the Neutron agent state (either 0 for 'up', 1 for 'down' or 2 for 'disabled'). The metric contains a `service` field (one of 'dhcp', 'l3', 'metadata' or 'openvswitch'), and a `state` field (one of 'up', 'down' or 'disabled').

8.8.7 API response times

- `openstack_<service>_http_responses`, the time (in second) it took to serve the HTTP request. The metric contains `http_method` (eg 'GET', 'POST', and so forth) and `http_status` (eg '200', '404', and so forth) fields.

`<service>` is one of 'cinder', 'glance', 'heat' 'keystone', 'neutron' or 'nova'.

8.8.8 Logs

- `log_messages`, the number of log messages per second for the given service and severity level. The metric contains `service` and `severity` (one of 'debug', 'info', ...) fields.

8.9 Ceph

All Ceph metrics have a `cluster` field containing the name of the Ceph cluster (*ceph* by default).

See cluster monitoring and RADOS monitoring for further details.

8.9.1 Cluster

- `ceph_health`, the health status of the entire cluster where values `1`, `2` , `3` represent respectively `OK`, `WARNING` and `ERROR`.
- `ceph_monitor_count`, number of ceph-mon processes.
- `ceph_quorum_count`, number of ceph-mon processes participating in the quorum.

8.9.2 Pools

- `ceph_pool_total_bytes`, total number of bytes for all pools.
- `ceph_pool_total_used_bytes`, total used size in bytes by all pools.
- `ceph_pool_total_avail_bytes`, total available size in bytes for all pools.
- `ceph_pool_total_number`, total number of pools.

The folllowing metrics have a `pool` field that contains the name of the Ceph pool.

- `ceph_pool_bytes_used`, amount of data in bytes used by the pool.
- `ceph_pool_max_avail`, available size in bytes for the pool.
- `ceph_pool_objects`, number of objects in the pool.
- `ceph_pool_read_bytes_sec`, number of bytes read by second for the pool.
- `ceph_pool_write_bytes_sec`, number of bytes written by second for the pool.
- `ceph_pool_op_per_sec`, number of operations per second for the pool.
- `ceph_pool_size`, number of data replications for the pool.
- `ceph_pool_pg_num`, number of placement groups for the pool.

8.9.3 Placement Groups

- `ceph_pg_total`, total number of placement groups.
- `ceph_pg_bytes_avail`, available size in bytes.
- `ceph_pg_bytes_total`, cluster total size in bytes.
- `ceph_pg_bytes_used`, data stored size in bytes.
- `ceph_pg_data_bytes`, stored data size in bytes before it is replicated, cloned or snapshotted.
- `ceph_pg_state`, number of placement groups in a given state. The metric contains a `state` field whose value is `<state>` is a combination separated by `+` of 2 or more states of this list: `creating`, `active`, `clean`, `down`, `replay`, `splitting`, `scrubbing`, `degraded`, `inconsistent`, `peering`, `repair`, `recovering`, `recovery_wait`, `backfill`, `backfill-wait`, `backfill_toofull`, `incomplete`, `stale`, `remapped`.

8.9.4 OSD Daemons

- `ceph_osd_up`, number of OSD daemons UP.
- `ceph_osd_down`, number of OSD daemons DOWN.
- `ceph_osd_in`, number of OSD daemons IN.
- `ceph_osd_out`, number of OSD daemons OUT.

The following metrics have an `osd` field that contains the OSD identifier.

- `ceph_osd_used`, data stored size in bytes for the given OSD.
- `ceph_osd_total`, total size in bytes for the given OSD.
- `ceph_osd_apply_latency`, apply latency in ms for the given OSD.
- `ceph_osd_commit_latency`, commit latency in ms for the given OSD.

8.9.5 OSD Performance

All the following metrics are retrieved per OSD daemon from the corresponding socket `/var/run/ceph/ceph-osd.<ID>.asok` by issuing the command `perf dump`.

All metrics have an `osd` field that contains the OSD identifier.

Note: These metrics are not collected when a node has both the ceph-osd and controller roles.

See OSD performance counters for further details.

- `ceph_perf_osd_recovery_ops`, number of recovery operations in progress.
- `ceph_perf_osd_op_wip`, number of replication operations currently being processed (primary).
- `ceph_perf_osd_op`, number of client operations.
- `ceph_perf_osd_op_in_bytes`, number of bytes received from clients for write operations.
- `ceph_perf_osd_op_out_bytes`, number of bytes sent to clients for read operations.
- `ceph_perf_osd_op_latency`, average latency in ms for client operations (including queue time).
- `ceph_perf_osd_op_process_latency`, average latency in ms for client operations (excluding queue time).
- `ceph_perf_osd_op_r`, number of client read operations.
- `ceph_perf_osd_op_r_out_bytes`, number of bytes sent to clients for read operations.
- `ceph_perf_osd_op_r_latency`, average latency in ms for read operation (including queue time).
- `ceph_perf_osd_op_r_process_latency`, average latency in ms for read operation (excluding queue time).
- `ceph_perf_osd_op_w`, number of client write operations.
- `ceph_perf_osd_op_w_in_bytes`, number of bytes received from clients for write operations.
- `ceph_perf_osd_op_w_rlat`, average latency in ms for write operations with readable/applied.
- `ceph_perf_osd_op_w_latency`, average latency in ms for write operations (including queue time).

- `ceph_perf_osd_op_w_process_latency`, average latency in ms for write operation (excluding queue time).
- `ceph_perf_osd_op_rw`, number of client read-modify-write operations.
- `ceph_perf_osd_op_rw_in_bytes`, number of bytes per second received from clients for read-modify-write operations.
- `ceph_perf_osd_op_rw_out_bytes`, number of bytes per second sent to clients for read-modify-write operations.
- `ceph_perf_osd_op_rw_rlat`, average latency in ms for read-modify-write operations with readable/applied.
- `ceph_perf_osd_op_rw_latency`, average latency in ms for read-modify-write operations (including queue time).
- `ceph_perf_osd_op_rw_process_latency`, average latency in ms for read-modify-write operations (excluding queue time).

8.10 Pacemaker

8.10.1 Resource location

- `pacemaker_resource_local_active`, `1` when the resource is located on the host reporting the metric, if not `0`. The metric contains a `resource` field which is one of 'vip__public', 'vip__management', 'vip__vrouter_pub' or 'vip__vrouter'.

8.11 Clusters

The cluster metrics are emitted by the GSE plugins (See the *Alarms Configuration Guide* for details).

- `cluster_service_status`, the status of the service cluster. The metric contains a `cluster_name` field that identifies the service cluster.
- `cluster_node_status`, the status of the node cluster. The metric contains a `cluster_name` field that identifies the node cluster.
- `cluster_status`, the status of the global cluster. The metric contains a `cluster_name` field that identifies the global cluster.

The supported values for these metrics are:

- *0* for the *Okay* status.
- *1* for the *Warning* status.
- *2* for the *Unknown* status.
- *3* for the *Critical* status.
- *4* for the *Down* status.

8.12 LMA self-monitoring

8.12.1 System

Metrics have a `service` field with the name of the service it applies to. Values can be: hekad, collectd, influxd, grafana-server or elasticsearch.

- `lma_components_count_processes`, number of processes currently running.
- `lma_components_count_threads`, number of threads currently running.
- `lma_components_cputime_user`, percentage of CPU time spent in user mode by the service. It can be greater than 100% when the node has more than one CPU.
- `lma_components_cputime_syst`, percentage of CPU time spent in system mode by the service. It can be greater than 100% when the node has more than one CPU.
- `lma_components_disk_bytes_read`, number of bytes read from disk(s) per second.
- `lma_components_disk_bytes_write`, number of bytes written to disk(s) per second.
- `lma_components_disk_ops_read`, number of read operations from disk(s) per second.
- `lma_components_disk_ops_write`, number of write operations to disk(s) per second.
- `lma_components_memory_code`, physical memory devoted to executable code (bytes).
- `lma_components_memory_data`, physical memory devoted to other than executable code (bytes).
- `lma_components_memory_rss`, non-swapped physical memory used (bytes).
- `lma_components_memory_vm`, virtual memory size (bytes).
- `lma_components_pagefaults_minflt`, minor page faults per second.
- `lma_components_pagefaults_majflt`, major page faults per second.
- `lma_components_stacksize`, absolute value of the start address (the bottom) of the stack minus the address of the current stack pointer.

8.12.2 Heka pipeline

Metrics have two fields: `name` that contains the name of the decoder or filter as defined by *Heka* and `type` that is either *decoder* or *filter*.

Metrics for both types:

- `hekad_msg_avg_duration`, the average time for processing the message (in nanoseconds).
- `hekad_msg_count`, the total number of messages processed by the decoder. This will reset to 0 when the process is restarted.
- `hekad_memory`, the total memory used by the Sandbox (in bytes).

Additional metrics for *filter* type:

- `heakd_timer_event_avg_duration`, the average time for executing the *timer_event* function (in nanoseconds).
- `hekad_timer_event_count`, the total number of executions of the *timer_event* function. This will reset to 0 when the process is restarted.

8.12.3 Backend checks

- `http_check`, the backend's API status, 1 if it is responsive, if not 0. The metric contains a `service` field that identifies the LMA backend service being checked.

`<service>` is one of the following values (depending of which Fuel plugins are deployed in the environment):

- 'influxdb'

8.13 Elasticsearch

The following metrics represent the simple status on the health of the cluster. See cluster health for further details.

- `elasticsearch_cluster_health`, the health status of the entire cluster where values `1`, `2` , `3` represent respectively `green`, `yellow` and `red`. The `red` status may also be reported when the Elasticsearch API returns an unexpected result (network failure for instance).
- `elasticsearch_cluster_active_primary_shards`, the number of active primary shards.
- `elasticsearch_cluster_active_shards`, the number of active shards.
- `elasticsearch_cluster_initializing_shards`, the number of initializing shards.
- `elasticsearch_cluster_number_of_nodes`, the number of nodes in the cluster.
- `elasticsearch_cluster_number_of_pending_tasks`, the number of pending tasks.
- `elasticsearch_cluster_relocating_shards`, the number of relocating shards.
- `elasticsearch_cluster_unassigned_shards`, the number of unassigned shards.

8.14 InfluxDB

The following metrics are extracted from the output of `show stats` command. The values are reset to zero when InfluxDB is restarted.

8.14.1 cluster

These metrics are only available if there are more than one node in the cluster.

- `influxdb_cluster_write_shard_points_requests`, the number of requests for writing a time series points to a shard.
- `influxdb_cluster_write_shard_requests`, the number of requests for writing to a shard.

8.14.2 httpd

- `influxdb_httpd_failed_auths`, the number of times failed authentications.
- `influxdb_httpd_ping_requests`, the number of ping requests.
- `influxdb_httpd_write_points_ok`, the number of points successfully written.
- `influxdb_httpd_query_requests`, the number of query requests received.
- `influxdb_httpd_query_response_bytes`, the number of bytes returned to the client.

- `influxdb_httpd_requests`, the number of requests received.
- `influxdb_httpd_write_requests`, the number of write requests received.
- `influxdb_httpd_write_request_bytes`, the number of bytes received for write requests.

8.14.3 write

- `influxdb_write_point_requests`, the number of write points requests across all data nodes.
- `influxdb_write_local_point_requests`, the number of write points requests from the local data node.
- `influxdb_write_remote_point_requests`, the number of write points requests to remote data nodes.
- `influxdb_write_requests`, the number of write requests across all data nodes.
- `influxdb_write_sub_ok`, the number of successful points send to subscriptions.
- `influxdb_write_ok`, the number of successful writes of consistency level.

8.14.4 runtime

- `influxdb_memory_alloc`, the number of bytes allocated and not yet freed.
- `influxdb_memory_total_alloc`, the number of bytes allocated (even if freed).
- `influxdb_memory_system`, the number of bytes obtained from the system.
- `influxdb_memory_lookups`, the number of pointer lookups.
- `influxdb_memory_mallocs`, the number of malloc operations.
- `influxdb_memory_frees`, the number of free operations.
- `influxdb_heap_idle`, the number of bytes in idle spans.
- `influxdb_heap_in_use`, the number of bytes in non-idle spans.
- `influxdb_heap_objects`, the total number of allocated objects.
- `influxdb_heap_released`, the number of bytes released to the operating system.
- `influxdb_heap_system`, the number of bytes obtained from the system.
- `influxdb_garbage_collections`, the number of garbage collections.
- `influxdb_go_routines`, the number of Golang routines.

CHAPTER 9

Indices and Tables

- search