The LMA Collector Plugin for Fuel Documentation

Release 0.8.0

Mirantis Inc.

December 14, 2015

Contents

1 User documentation **1**
- 1.1 Overview . . . 1
- 1.2 Release Notes . . . 3
- 1.3 Installation . . . 3
- 1.4 Configuration Guide . . . 5
- 1.5 Alarms Configuration Guide . . . 7
- 1.6 Licenses . . . 16
- 1.7 Appendix . . . 16

2 Developer documentation **18**
- 2.1 Overview . . . 18
- 2.2 Common Message Format . . . 19
- 2.3 Log Messages . . . 20
- 2.4 Notification Messages . . . 21
- 2.5 Metric Messages . . . 22
- 2.6 Supported Outputs . . . 38
- 2.7 Running tests . . . 39

3 Indices and Tables **40**

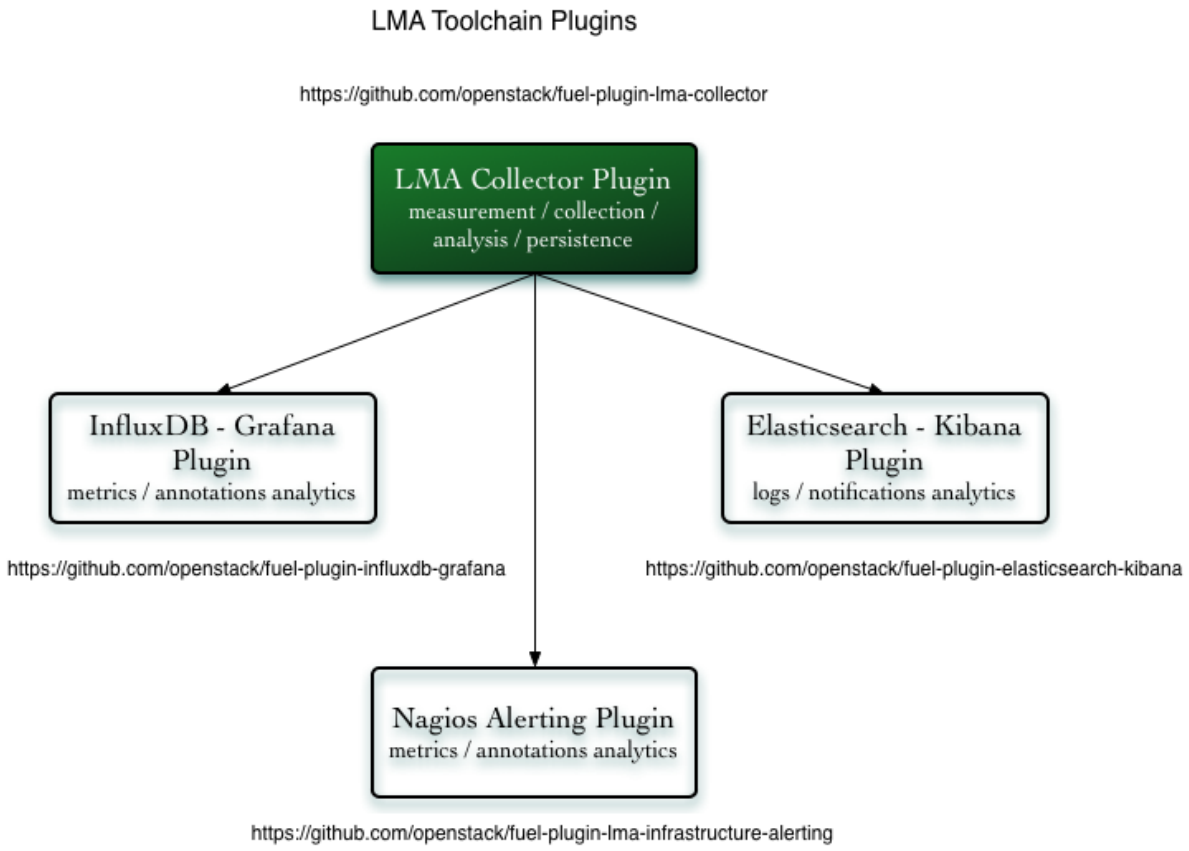
CHAPTER 1

User documentation

1.1 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 are 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 a 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.

1.1.1 Requirements

Requirement	Version/Comment
Mirantis OpenStack	7.0
A running Elasticsearch server (for log analytics)	1.4 or higher, the RESTful API must be enabled over port 9200
A running InfluxDB server (for metric analytics)	0.9.2 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

1.1.2 Limitations

The plugin is only compatible with OpenStack environments deployed with Neutron as the networking configuration.

The log and notification messages aren't buffered anymore by the LMA collector service before being sent to Elasticsearch (see #1488717 for details). This means that the data may be lost when the Elasticsearch server is unreachable. It will be fixed in a future maintenance release of the plugin.

1.2 Release Notes

1.2.1 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.

1.2.2 Version 0.7.0

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

1.3 Installation

Prior to installing the LMA Collector Plugin, you may want to install its dependencies:

- Elasticsearch and Kibana for log analytics
- InfluxDB and Grafana for metrics analytics
- Nagios for alerting

To install them automatically you can refer to the:

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

You can install Elasticsearch/Kibana, InfluxDB/Grafana and Nagios outside of the Fuel Plugin framework as long as your installation meets the LMA Collector plugin's *requirements*.

1.3.1 LMA Collector Fuel Plugin install 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, you need to 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-0.8-0.8.0-1.noarch.rpm \
root@<Fuel Master node IP address>:
```

3. Install the plugin using the Fuel CLI:

```
[root@fuel ~]# fuel plugins --install lma_collector-0.8-0.8.0-1.noarch.rpm
```

4. Verify that the plugin is installed correctly:

```
[root@fuel ~]# fuel plugins --list
id | name                 | version | package_version
---|----------------------|---------|----------------
1  | lma_collector        | 0.8.0   | 2.0.0
```

1.3.2 LMA Collector Fuel Plugin install 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 for building 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 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 git@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:

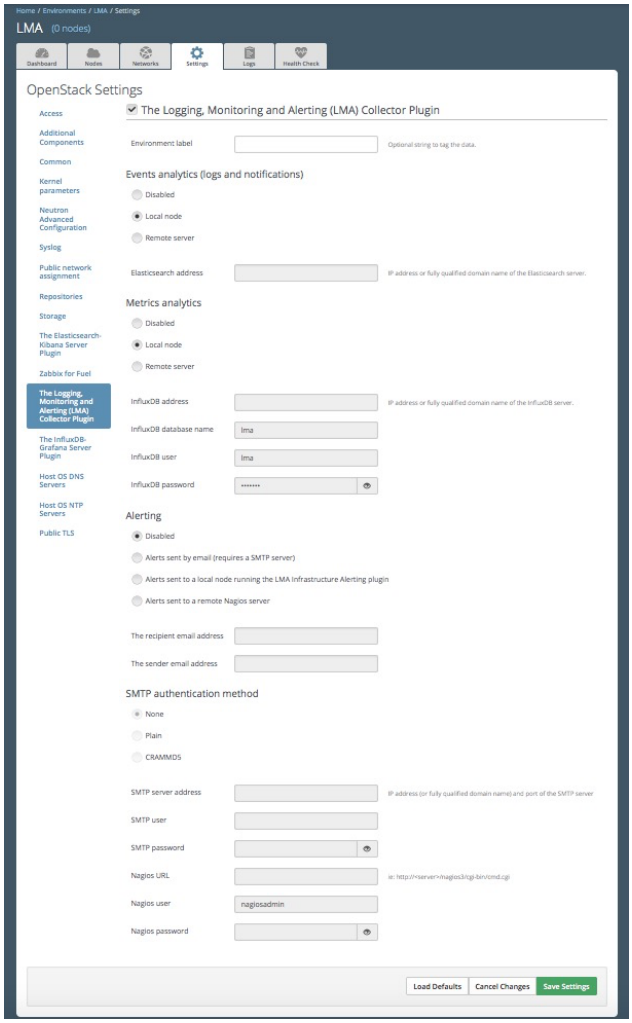
```
[root@fuel ~] fuel plugins --install ./fuel-plugin-lma_collector/*.noarch.rpm
```

1.4 Configuration Guide

1.4.1 Plugin configuration

To configure your plugin, you need to follow the folling steps:

1. Create a new environment with the Fuel web user interface.
2. Click on the Settings tab of the Fuel web UI.
3. Select the LMA collector plugin in the left column. The LMA Collector settings screen appears.



4. Select the LMA collector plugin checkbox and fill-in the required fields.

1. Select “Local node” for Events analytics if you deploy the Elasticsearch-Kibana plugin on a dedicated node in the same environment.
2. Select “Remote server” for Events analytics if you have an Elasticsearch-Kibana server already deployed and running. In that case, you have to enter the IP address or the fully qualified name of the server.
3. Select “Local node” for Metrics analytics if you deploy the InfluxDB-Grafana plugin on a dedicated node in the same environment.
4. Select “Remote server” for Metrics analytics if you have an InfluxDB-Grafana server already deployed and running. In that case, you have to enter the IP address or the fully qualified name of the server as well as the credentials and database to store the metrics.
5. Select “Alerts sent by email” for Alerting if you wish to receive alerts by email.
6. Select “Alerts sent to a local node” for Alerting if you deploy the LMA Infrastructure Alerting plugin on a dedicated node in the same environment.
7. Select “Alerts sent to a remote Nagios server” for Alerting if you have a Nagios server already deployed and running.

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.

1.4.2 Plugin verification

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

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

Please refer to the *Troubleshooting* section otherwise.

1.4.3 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 nodes
[root@node-1 ~]# crm resource status lma_collector

# On nodes which are not controllers
[root@node-1 ~]# status lma_collector
```

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

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

# On nodes which are not controllers
[root@node-1 ~]# status 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.

1.5 Alarms Configuration Guide

1.5.1 Overview

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

Each Collector is individuallly 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 you can modify. Those rules are executed by a collection of Heka filter plugins written in Lua that are organised according to a configurable processing workflow.

We call these plugins the *AFD plugins* for Anomaly and Fault Detection plugins and the *GSE plugins* for Global Status Evaluation plugins.

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

Fig. 1.1: Message flow for the AFD and GSE metrics

The *AFD metrics* contain information about the health status of a resource like 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 of aggregator.

The *GSE metrics* contain information about the health status of a service cluster, like 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 receives from the Collectors.

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

- 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 on).
- 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 on).
- 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 on).

The meaning associated with a health status is the following:

- **Down**: One or several primary functions of a cluster are failed. 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 we call the *Persisters*.

- There is a *Persister* for InfluxDB which 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.
- There is a *Persister* for Elasticsearch which 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.
- There is a *Persister* for Nagios which 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.

1.5.2 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.

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 like 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:
  returns the result (true, false) of the rate of change test function of
  Heka (not implemented yet)
- 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

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 assingned 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 that is 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) superiror 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-0.8/puppet/modules/ \
/etc/fuel/plugins/lma_collector-0.8/puppet/manifests/configure_afd_filters.pp
```

This will restart the LMA Collector with your change.

1.5.3 Cluster policies

GSE plugins are driven by policies that describe how plugins determine the cluster's health status.

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*.

1.6 Licenses

1.6.1 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

1.6.2 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-concat	https://github.com/puppetlabs/puppetlabs-concat	Apache v2
puppetlabs-firewall	https://github.com/puppetlabs/puppetlabs-firewall	Apache v2

1.7 Appendix

- 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 2

Developer documentation

2.1 Overview

The Mirantis OpenStack LMA (Logging, Monitoring and Alerting) Toolchain is comprised of a collection of open-source tools to help you monitor and diagnose problems in your OpenStack environment. These tools are packaged and delivered as Fuel plugins you can install from within the graphic user interface of Fuel starting with Mirantis OpenStack version 6.1.

From a high level view, the LMA Toolchain includes:

- The LMA Collector (or just the Collector) to gather all operational data that we think are relevant to increase the **operational visibility** over your OpenStack environment. Those data are collected from a variety of sources including the log messages, collectd, and the OpenStack notifications bus
- Pluggable external systems we call **satellite clusters** which can take action on the data received from the Collectors running on the OpenStack nodes.

The Collector is best described as a **pluggable message processing and routing pipeline**. Its core components are :

- Collectd that is bundled with a collection of monitoring plugins. Many of them are purpose-built for OpenStack.
- Heka which is the cornerstone component of the Collector.
- A collection of Heka plugins written in Lua to decode, process and encode the data to be sent to external systems.

The primary function of the Collector is to transform the acquired raw operational data into an internal message representation that is based on the Heka message structure. that can be further exploited to, for example, detect anomalies or create new metric messages.

The satellite clusters delivered as part of the LMA Toolchain starting with Mirantis OpenStack 6.1 include:

- Elasticsearch, a powerful open source search server based on Lucene and analytics engine that makes data like log messages and notifications easy to explore and analyse.
- InfluxDB, an open-source and distributed time-series database to store and search metrics.

By combining Elasticsearch with Kibana, the LMA Toolchain provides an effective way to search and correlate all service-affecting events that occurred in the system for root cause analysis.

Likewise, by combining InfluxDB with Grafana, the LMA Toolchain brings you insightful metrics analytics to visualise how OpenStack behaves over time. This includes metrics for the OpenStack services status and a variety of resource usage and performance indicators. The ability to visualise time-series over a period of time that can vary from 5 minutes to the last 30 days helps anticipating failure conditions and plan capacity ahead of time to cope with a changing demand.

Furthermore, the LMA Toolchain has been designed with the dual objective to be both insightful and adaptive.

It is, for example, quite possible (without any code change) to integrate the Collector with an external monitoring application like Nagios. This could simply be done through enabling the Nagios output plugin of Heka for a subset of messages matching the message matcher syntax of the output plugin. You should probably not modify the configuration of the LMA Collector manually but apply any configuration change to the Puppet manifests that are shipped with the LMA Collector plugin for Fuel. Many other integration combinations are possible thanks to the extreme flexibility of Heka.

We recommend you to read the Heka documentation to become more familiar with that technology.

The rest of this document is organised in several chapters that will take you through a description of the internal message structure for the categories of operational data that are handled by the LMA Toolchain.

2.2 Common Message Format

Heka turns the incoming data into Heka messages [1] with a well-defined format which is described below.

- **Timestamp** (number), the timestamp of the message (in nanoseconds since the Epoch).
- **Logger** (string), the datasource from the Heka's standpoint.
- **Type** (string), the type of message.
- **Hostname** (string), the name of the host that emitted the message.
- **Severity** (number), severity level as defined by the Syslog RFC 5424.
- **Payload** (string), the input data in most cases.
- **Pid** (number), the Process ID that generated the message.
- **Fields**, array of Field structures (see below).

2.2.1 Field Format

Every message (either originating from logs, metrics or notifications) is populated with a set of predefined fields:

Attributes in **bold** are always present in the messages while attributes in *italic* are optional.

- **deployment_mode** (string), the deployment mode of the Fuel environment (either 'multinode' or 'ha_compact').
- **deployment_id** (number), the deployment identifier of the Fuel environment.
- **openstack_region** (string), the name of the OpenStack region.
- **openstack_release** (string), the name of the OpenStack release.
- **openstack_roles** (string), a comma-separated list of the node's roles (eg 'controller', 'compute,cinder').
- *environment_label* (string), the label assigned to the OpenStack environment.

Note: All date/time fields represented as string are formatted according to the RFC3339 document.

[1] Heka message structure

2.3 Log Messages

The Heka collector service is configured to tail the following log files:

- System logs.
  - `/var/log/syslog`
  - `/var/log/messages`
  - `/var/log/debug`
  - `/var/log/auth.log`
  - `/var/log/cron.log`
  - `/var/log/daemon.log`
  - `/var/log/kern.log`
  - `/var/log/pacemaker.log`
- MySQL server logs (for controller nodes).
- RabbitMQ server logs (for controller nodes).
- Pacemaker logs (for controller nodes).
- OpenStack logs.
- Open vSwitch logs (all nodes).
  - `/var/log/openvswitch/ovsdb-server.log`
  - `/var/log/openvswitch/ovs-vswitchd.log`

2.3.1 Log Messages Format

In addition to the common *Common Message Format*, log-based messages have additional properties.

Attributes in **bold** are always present in the messages while attributes in *italic* are optional.

- **Logger** (string), `system.<service>`, `mysql` or `openstack.<service>`.
- **Type** (string), always `log`.
- **Fields**
  - **severity_label** (string), the textual representation of the severity level.
  - *programname* (string), the application name for Syslog-based messages.
  - *syslogfacility* (number), the Syslog facility for Syslog-based messages.
  - *http_method* (string), the HTTP method (for instance 'GET').
  - *http_client_ip_address* (string), the IP address of the client that originated the HTTP request.
  - *http_response_size* (number), the size of the HTTP response (in bytes).
  - *http_response_time* (number), the HTTP response time (in seconds).
  - *http_status* (string), the HTTP response status.
  - *http_url* (string), the requested HTTP URL.
  - *http_version* (string), the HTTP version (eg '1.1).

  - *request_id* (string), the UUID of the OpenStack request to which the message applies.
  - *tenant_id* (string), the UUID of the OpenStack tenant to which the message applies.
  - *user_id* (string), the UUID of the OpenStack user to which the message applies.

2.4 Notification Messages

OpenStack services can be configured to send notifications on the message bus about the executing task or the state of the cloud resources [2]. These notifications are received by the LMA collector service and turned into Heka messages.

2.4.1 Notification Messages Format

In addition to the *Common Message Format*, notification-based messages have additional properties.

Attributes in **bold** are always present in the messages while attributes in *italic* are optional.

- **Logger** (string), the OpenStack service that emitted the notification, (eg, `nova`).
- **Payload** (string), the payload of the OpenStack notification.
- **Hostname** (string), the name of the host that originated the notification.
- **Type** (string), always `notification`.
- **Fields**
  - **hostname** (string), the name of the host that originated the notification.
  - **publisher** (string), the name of the underlying service that emitted the notification (eg, `scheduler`).
  - **severity_label** (string), the textual representation of the severity level.
  - **event_type** (string), the notification's type (eg `compute.instance.create.end`).
  - *tenant_id* (string), the UUID of the OpenStack tenant to which the message applies.
  - *user_id* (string), the UUID of the OpenStack user to which the message applies.
  - *instance_id* (string), the UUID of the virtual instance to which the message applies.
  - *image_name* (string), the image used by the image.
  - *display_name* (string), the visible name of the resource.
  - *instance_type* (string), the type of instance (eg `m1.small`).
  - *availability_zone* (string), the availability zone of the instance.
  - *vcpus* (number), the number of VCPU provisioned for the instance.
  - *memory_mb* (number), the amount of RAM provisioned for the instance.
  - *disk_gb* (number), the disk space provisioned for the instance.
  - *old_state* (string), the previous state of the instance (eg `building`).
  - *state* (string), the state of the instance (eg `active`).
  - *old_task_state* (string), the previous task state for the instance (eg `block_device_mapping`).
  - *new_task_state* (string), the new task state for the instance (eg `spawning`).

[2] OpenStack notifications

  - *created_at* (string): the date of creation of the instance.
  - *launched_at* (string): the date when the instance was effectively launched.
  - *deleted_at* (string): the date of deletion of the instance.
  - *terminated_at* (string): the date when the instance was effectively terminated.

2.5 Metric Messages

Metrics are extracted from several sources:

- Data received from collectd.
- Log messages processed by the collector service.
- OpenStack notifications processed by the collector service.

2.5.1 Metric Messages Format

In addition to the common *Common Message Format*, metric messages have additional properties.

Attributes in **bold** are always present in the messages while attributes in *italic* are optional.

- **Logger** (string), the datasource from the Heka's standpoint, it can be `collectd`, `notification_processor` or `http_log_parser`.
- **Type** (string)
- `metric` or `heka.sandbox.metric` for the single-value metrics.
- `heka.sandbox.multivalue_metric` for the multi-valued metrics (eg annotations).
- `heka.sandbox.bulk_metric` for the metrics sent by bulk.
- `heka.sandbox.afd_service_metric` for the AFD service metrics.
- `heka.sandbox.afd_node_metric` for the AFD node metrics.
- `heka.sandbox.gse_service_cluster_metric` for the GSE service cluster metrics.
- `heka.sandbox.gse_node_cluster_metric` for the GSE node cluster metrics.
- `heka.sandbox.gse_cluster_metric` for the GSE global cluster metrics.
- **Severity** (number), it is always equal to 6 (INFO).
- **Fields**
- **name** (string), the name of the metric. See *List of metrics* for the current metric names that are emitted.
- **value** (number), the value associated to the metric.
- **type** (string), the metric's type, either `gauge` (a value that can go up or down), `counter` (an always increasing value) or `derive` (a per-second rate).
- **source** (string), the source from where the metric comes from, it can be the name of the collectd plugin, `<service>-api` for HTTP response metrics.
- **hostname** (string), the name of the host to which the metric applies. It may be different from the `Hostname` value. For instance when the metric is extracted from an OpenStack notification, `Hostname` is the host that captured the notification and `Fields[hostname]` is the host that emitted the notification.
- *interval* (number), the interval at which the metric is emitted (for the `collectd` metrics).

- *tenant_id* (string), the UUID of the OpenStack tenant to which the metric applies.
- *user_id* (string), the UUID of the OpenStack user to which the metric applies.

Metric messages may include additional fields to specify the scope of the measurement. When this is the case, these fields are detailed in the list of metrics presented hereafter.

2.5.2 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.

System

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.

Disk

Metrics have a `device` field that contains the disk device number to which the metric applies (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.

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.

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.

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.

Network

Metrics have a `interface` field that contains the interface name to which the metric applies (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 interace.
- `if_octets_rx`, the number of octets (bytes) received per second by the interace.
- `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 interace.

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

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').

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.

Users

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

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.

MySQL

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.

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 to which it applies. 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.

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.

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.

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.

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, `0` otherwise.
- `mysql_cluster_ready`, `1` when the node is ready to accept queries, `0` otherwise.
- `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.

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)

RabbitMQ

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`.

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.

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

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.

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 of total number of session.

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.

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.

OpenStack

Service checks

- **`openstack_check_api`, the service's API status, 1 if it is responsive, 0 otherwise.** 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.

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 count of instances which entered a given state (the value is always 1). The metric contains a `state` field.

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').

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').

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').

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'.

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.

- `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.

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 on) and `http_status` (eg '200', '404', and so on) fields.

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

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.

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.

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.

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.

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.

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).

Pacemaker

Resource location

- `pacemaker_resource_local_active`, `1` when the resource is located on the host reporting the metric, `0` otherwise. The metric contains a `resource` field which is one of 'vip__public', 'vip__management', 'vip__vrouter_pub' or 'vip__vrouter'.

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.

LMA self-monitoring

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 address of the start (i.e., bottom) of the stack minus the current value of the stack pointer.

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.

2.6 Supported Outputs

The LMA collector can forward part or all of the processed Heka messages to any kind of external system, provided that the system supports a protocol-based interface such as HTTP, SMTP or AMQP.

The supported backends are described hereunder.

2.6.1 Elasticsearch

The LMA collector is able to send *Log Messages* and *Notification Messages* to Elasticsearch.

There is one index per day and per type of message:

- Index for log messages is `log-<YYYY-MM-DD>`.
- Index for notification messages is `notification-<YYYY-MM-DD>`.

2.6.2 InfluxDB

The LMA collector is able to send *Metric Messages* to InfluxDB.

A metric message is stored into a measurement whose name is taken from *Fields[name]*. The datapoint's timestamp is taken from the *Timestamp* field and *Fields[value]* is stored as the *value* field. Note that numerical values are always encoded as float numbers.

Some tags are associated to all measurements:

- *deployment_id*
- *hostname*

If the metric message contains a non-empty *Fields[tag_fields]* list, the items listed in this field are encoded as additional key-value tags.

For instance, lets take the following Heka message:

```
2015/09/15 16:16:05
:Timestamp: 2015-09-15 16:15:37.645999872 +0000 UTC
:Type: metric
:Hostname: node-1
:Pid: 15595
:Uuid: e67f91c5-259b-489f-adfa-8eea0b389eb2
:Logger: collectd
:Payload: {"type":"cpu","values":[0],"type_instance":"idle","dsnames":["value"],
          "plugin":"cpu","time":1442333737.646,"interval":10,"host":"node-1",
```

```
          "dstypes":["derive"],"plugin_instance":"0"}
:EnvVersion:
:Severity: 6
:Fields:
    | name:"type" type:string value:"derive"
    | name:"source" type:string value:"cpu"
    | name:"deployment_mode" type:string value:"ha_compact"
    | name:"deployment_id" type:string value:"1"
    | name:"openstack_roles" type:string value:"primary-controller"
    | name:"openstack_release" type:string value:"2015.1.0-7.0"
    | name:"tag_fields" type:string value:"cpu_number"
    | name:"openstack_region" type:string value:"RegionOne"
    | name:"name" type:string value:"cpu_idle"
    | name:"hostname" type:string value:"node-1"
    | name:"value" type:double value:0
    | name:"environment_label" type:string value:"deploy_lma_infra_alerting_ha"
    | name:"interval" type:double value:10
    | name:"cpu_number" type:string value:"95"
```

Using the InfluxDB line protocol, it would be encoded like this:

```
cpu_idle,cpu_number=0,deployment_id=1,hostname=node-1 value=95.000000 1442333737645
```

2.7 Running tests

You need to have *tox* and *bundler* installed for running the tests.

Quickstart for Ubuntu Trusty:

```
apt-get install tox ruby ruby1.9.1-dev
gem install bundler
tox
```

CHAPTER 3

Indices and Tables

- search