The LMA Collector Developer Documentation

Release 0.9.0

Mirantis Inc.

Jan 16, 2018

Contents

CHAPTER 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.

CHAPTER 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).

2.1 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_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

CHAPTER 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`

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, or the OpenStack service daemon name for OpenStack log messages (eg "nova-compute").
  - *syslogfacility* (number), the Syslog facility for Syslog-based messages.
  - *python_module* (string), the Python module that generated the log message for OpenStack service logs.
  - *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.

CHAPTER 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[1]. These notifications are received by the LMA collector service and turned into Heka messages.

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.

[1] OpenStack notifications

- *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`).
- *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.

CHAPTER 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.

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 the User Documentation 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. Refer to the User Documentation for more details.

CHAPTER 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.

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>`.

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_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
```

CHAPTER 7

Installation without Fuel

This section provides instructions and hints on how the LMA Collector service can be deployed without using the Fuel plugin package. For instance, the Fuel version that you are running isn't compatible with the current release of the LMA Collector or you want to have more control on the configuration of the LMA Collector.

In such situations, it is possible to leverage directly the Puppet modules and write your own Puppet manifests to configure and run the LMA Collector service on the OpenStack nodes.

7.1 Pre-requisites

- The nodes are already deployed with the OpenStack services.
- The nodes can download and install packages from a repository server that you manage.
- Configuration management is done with Puppet >= 3.x. Both master and masterless methods are supported.
- You have already written the main Puppet manifests. You can have a look at the reference documentation and at the examples of the *lma_collector* Puppet module.
- The satellite clusters (Elasticsearch/Kibana, InfluxDB/Grafana and Nagios) are already deployed and the nodes where LMA Collectors run have access to them.

7.2 Download the packages

Before running the Puppet manifests, you have to make sure that the nodes will be able to download and install the necessary packages.

This small script will get you started:

```
WORK_DIR=/tmp/lma_collector
PACKAGES_DIR=${WORK_DIR}/packages
mkdir -p ${PACKAGES_DIR}
rm -rf ${PACKAGES_DIR:?}/*
```

```
pushd $WORK_DIR
git clone https://github.com/openstack/fuel-plugin-lma-collector.git
cd fuel-plugin-lma-collector
./pre_build_hook
cp ./repositories/ubuntu/*.deb ${PACKAGES_DIR}
(cd ${PACKAGES_DIR} && dpkg-scanpackages . > Packages)
echo "The packages directory is available at ${PACKAGES_DIR}"
popd
```

Then you should copy the *packages* directory to your local repository server and update the APT configuration on the deployed nodes accordingly to enable the new source repository.

7.3 Building the Puppet modules

You have to build locally the *lma_collector* and *heka* Puppet modules because they aren't yet available from PuppetForge.

```
WORK_DIR=/tmp/lma_collector
mkdir -p ${WORK_DIR}
rm -rf ${WORK_DIR:?}/*
pushd $WORK_DIR
git clone https://github.com/openstack/fuel-plugin-lma-collector.git
cd fuel-plugin-lma-collector/deployment_scripts/puppet/modules/
for module in heka lma_collector
do
   pushd $module
   puppet module build
   cp pkg/*.tar.gz ${WORK_DIR}
   popd
done
echo "The Puppet modules are available at ${WORK_DIR}"
popd
```

7.4 Installing the Puppet modules

After building the *lma_collector* and *heka* Puppet modules, you need to install them on your Puppet master or on all the nodes (in case of masterless installation).

```
puppet module install mirantis-heka-1.0.0.tar.gz
puppet module install mirantis-lma_collector-1.0.0.tar.gz
```

7.5 Running the main Puppet manifest(s)

Finally you can run your main Puppet manifest(s). For the masterless case, it would mean executing the *puppet apply* command similar to this snippet:

```
puppet apply /etc/puppet/manifests/
```

CHAPTER 8

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
```

For *tox* to run the Lua unit tests included in the `lma_collector` Puppet module additional system packages are required:

```
apt-get install cmake lua5.1 liblua5.1 liblua5.1-dev lua-cjson lua-unit
```

CHAPTER 9

Indices and Tables

- search