logstash-metlog extensions Documentation

Release 0.1

Victor Ng

October 06, 2016

Contents

CHAPTER 1

logstash-metlog

logstash-metlog is a set of extensions for logstash to provide statsd, CEF over syslog, Sentry and JSON to HDFS capabilities.

Full documentation can be found here.

Contents:

1.1 Plugin Configuration

Metlog provides some plugins to ease integration with logstash.

Input plugins provided:

- logstash.inputs.udp
- logstash.inputs.zeromq_hs (deprecated)

Filter plugins provided:

- logstash.filters.tagger
- logstash.filters.catchall

Output plugins provided:

- logstash.outputs.metlog_cef
- logstash.outputs.metlog_file
- logstash.outputs.metlog_sentry_dsn
- logstash.outputs.metlog_statsd
- logstash.outputs.metlog_sentry (deprecated)

1.1.1 Input plugins

udp

The udp input plugin provides a basic UDP listener service for logstash.

Messages may be lost using this input listener, and messages greater than 64kb may be truncated.

For typical configuration, you need to only care about the host and port that the listener will operate on. A typical configuration block will look like this

```
udp {
    type => "metlog"
    mode => "server"
    format => "json"

    host => "0.0.0.0"
    port => 5565
}
```

The above configuration will let logstash listen on all network interfaces on port 5565.

The type, mode and format should always be set to "metlog", "server" and "json" as per the example.

In the above example, the *type* is required by logstash.inputs.base. It is not used by the zeromq_hs plugin.

mode must always be set as 'server' for the socket to bind properly. *address* is any valid URL recognized by 0mq.

1.1.2 Filter plugins

tagger configuration

The tagger filter lets you define a pattern keypath into an event. Each keypath is applied in order. On the first match - all tags will be applied to the event.

Keypaths are defined using a '/' notation.

One common case is to match the *type* of an event so that events are routed to a final destination. In the following example, we want to route all *timer* type events to the statsd output plugin by adding the tag 'output_statsd' to the event.

```
filter {
    tagger {
        # all timer messages are tagged with 'output_statsd'
        type => "metlog"
        pattern => [ "type", "timer"]
        add_tag => [ "output_statsd" ]
    }
}
```

If a keypath does not exist within an event, it is ignored.

Multiple keypaths can be defined using a flattened key/value mapping as shown in the following example.

If the file type is either a 'timer' or 'counter', the 'output_statsd' tag will be applied.

```
filter {
    tagger {
        # all timer and counter messages are tagged with 'output_statsd'
        type => "metlog"
        pattern => [ "type", "timer", "type", "counter" ]
        add_tag => [ "output_statsd" ]
    }
}
```

catchall configuration

The catchall filter is used to select messages which have not been previously tagged by another filter. This only works properly because the Logstash FilterWorker pool processes messages serially through each of the filters defined in

logstash.conf

Unfortunately, filters cannot see configuration from other filters so you must specify the set of tags which indicate that the message has been successfully filtered.

The catchall should specify the superset of all tags which logstash should care about. A logstash event must match *none* of the tags in this superset for the catchall filter to add the 'filter-catchall' tag to the event.

A typical configuration block is shown below

```
catchall {
    # anything that isn't tagged already gets tagged here
    tags => [ "output_text", "output_statsd", "output_sentry", "output_cef" ]
    add_tag => ['filter-catchall']
}
```

1.1.3 Output plugins

metlog_statsd configuration

The standard statsd output plugin provided by logstash is designed to repeatedly create the same kind of statsd message.

This plugin provides a basic interface to talk to a statsd server.

The plugin will map event attributes into statsd using

```
namespace = event.fields['fields']['logger']
key = event.fields['fields']['name']
value = event.fields['payload'].to_f
rate = event.fields['fields']['rate'].to_f
```

The default sampling rate is 1.

The value of event.fields['type'] must be one of 'counter' or 'timer'.

For counter messages, the final statsd message is constructed using

```
`namespace`.`key`:`value`|c|`rate`
```

For timer messages, the final statsd message is constructed using

```
`namespace`.`key`:`value`|ms|`rate`
```

Configuration of the plugin requires setting a host, port and a list of tags which the output plugin should watch for. At least one tag must match for the output plugin to be triggered.

The following configuration monitors only the 'output_statsd' tag and sends statsd messages to localhost at port 8125.

```
output {
    metlog_statsd {
        # Route any message tagged with 'output_statsd'
        # to the statsd server
        tags => ["output_statsd"]
        host => '127.0.0.1'
        port => 8125
    }
}
```

metlog_cef configuration

CEF messages are routed to the syslog daemon running on the local machine. The only configuration you need is the tag that a logstash event must have to route to this output.

A typical configuration block is below

```
metlog_cef {
    # CEF gets routed over syslog
    tags => ["output_cef"]
}
```

metlog_file configuration

This output plugin is able to output either JSON blobs or plain text.

In general, JSON file outputs are used for

For plain text, the plugin will extract a single field in the JSON blob and will write that out. Typically, this is the *payload* key so your configuration will look like this

```
metlog_file {
    # The plaintext logfile
    tags => ["output_text"]
    format => "preformatted_field"
    prefix_timestamps => true
    formatted_field => "payload"
    path => "/var/log/metlog/metlog_classic.log"
}
```

If you need to address a different part of the logstash event, simply use '/' notation.

```
metlog_file {
    tags => ["output_some_random_text"]
    format => "preformatted_field"
    formatted_field => "fields/logtext"
    path => "/var/log/metlog/metlog_some_random_text.log"
}
```

metlog_sentry_dsn

The metlog_sentry_dsn output plugin relies on metlog using metlog-raven >= 0.3. The metlog client will embed the sentry DSN which we want to use for final routing. The only configuration you need is the tag that a logstash event must have to route to this output.

A typical configuration block is below

```
metlog_sentry_dsn {
    # This is a new Sentry output plugin which requires the
    # metlog-raven client to embed the DSN in the metlog message
    tags => [ "output_sentry" ]
}
```

A complete configuration

Tying all these parts together is sometimes not entirely obvious, so we've assembled a working vagrant image for you. You can go use our vagrant backend to get a working enviroment.

The logstash configuration for that instance can always be used as a reference point for a working configuration.

1.2 HDFS Configuration

Deployment notes for setting up metlog-json logs so that they get pushed into HDFS

You'll need a couple pieces in play:

1. logstash
2. logrotate
3. Metlog enabled application

Instructions:

1. Ensure that JSON logs are rotated properly and being written out to:
   - /var/log/<your_app>/metrics_hdfs.log=%Y-%m-%d

   Example:
   - /var/log/sync_web/metrics_hdfs.log=2012-03-20

1. Make sure you've got the filename correct - specifically that the logrotation is *not* compressing with gzip.
2. Put a copy of metrics_hdfs.ini file into /etc/mozilla-services/metlog/metrics_hdfs.ini

   A sample INI file is below

```
# This configuration file is used by the scheduled job to push
# JSON logs to HDFS
[metlog]
logger = metlog_hadoop_transport
sender_class = metlog.senders.StdOutSender
[metlog_metrics_hdfs]
HADOOP_USER = sync_dev
HADOOP_HOST = 10.1.1.10 # Put your Hadoop SSH host here
SRC_LOGFILE = /var/log/syncweb/metrics_hdfs.log=%%Y-%%m-%%d.gz
DST_FNAME = hadoop_logs/metrics_hdfs.log
TMP_DIR = /opt/logstash/hdfs_logs
```

3. Ensure that the HADOOP_USER has been provisioned within the Hadoop cluster and that the SSH public keys have been installed into LDAP.
4. Ensure that upload_log.py is installed into /opt/logstash/bin/upload_log.py This should have been installed when you installed the logstash-metlog RPM.
5. Install private SSH keys for HADOOP_USER into /opt/logstash/ssh-keys

- Make sure that the identify file (the private key) is named "id_private_<HADOOP_USER>" For the previous metrics_hdfs.ini file, that means your identify file is

```
/opt/logstash/ssh-keys/id_private_sync_dev
```

#. Setup the logrotate daily job. A sample configuration is shown below.

```
## Managed by puppet
/var/log/syncweb/application.log /var/log/syncweb/metrics_hdfs.log {
    daily
    compress
    copytruncate
```

```
    dateext
    dateformat=%Y-%m-%d
    rotate 7
    postrotate
        /opt/logstash/bin/upload_log.py \
          --ssh-keys=/opt/logstash/ssh-keys \
          --config /etc/mozilla-services/metlog/metrics_hdfs.ini \
          && /usr/bin/pkill -HUP logstash
    endscript
}
```

You'll also need to have 2 directories setup for HDFS pushes to work correctly :

DST_FNAME:

> The DST_FNAME in metrics_hdfs.ini refers to a relative path from the home directory of the HADOOP_USER. In the metrics_hdfs.ini file in this example, the 'hadoop_logs/metrics_hdfs.log' value will be mapped to: /home/sync_dev/hadoop_logs/metrics_hdfs.log.<TIMESTAMP>
>
> The <TIMESTAMP> will be replaced with the timestamp that the logfile was moved.

TMP_DIR:

> TMP_DIR is a path on the local filesystem from the machine pushing logs to HDFS. This directory will get a copy of the log file that will be pushed to HDFS. On successful push to HDFS, the log file will be removed from TMP_DIR, but unsuccessful pushes will leave the log file in the TMP_DIR.

1.3 Deprecated Input Plugins

1.3.1 zeromq_hs configuration

The zeromq_hs input plugin is provides a simple handshake service which exposes a ZMQ::REP socket that is used to synchronize the PUB/SUB 0mq input plugin. This plugin is only required if Metlog has declared a sender type of ZmqHandshakePubSender.

Using the zeromq_hs plugin requires setting a 0mq address so to bind a socket. All other required keys are inherited from the base input plugin from logstash.

```
input {
    zeromq_hs {
       # Setup a ZMQ::REP socket that listens on port 5180
       type => "metlog"
       mode => "server"
       address => "tcp://*:5180"
    }
}
```

1.3.2 metlog_sentry

This output plugin is deprecated and no longer supported

CHAPTER 2

Indices and tables

- genindex
- modindex
- search