rfc5424 syslog handler Documentation

Release 1.4.3

Joris Beckers

Jan 28, 2020

Contents

An up-to-date, RFC 5424 compliant syslog handler for the Python logging framework.

- Free software: BSD License
- Homepage: https://github.com/jobec/rfc5424-logging-handler
- Documentation: http://rfc5424-logging-handler.readthedocs.org/

CHAPTER 1

Features

- RFC 5424 Compliant.
- Python Logging adapter for easier sending of rfc5424 specific fields.
- No need for complicated formatting strings.
- TLS/SSL syslog support.
- Alternate transports like streams (ex. stderr, stdout, file, . . .).

CHAPTER 2

Contents

2.1 Installation

Install the package with pip:

```
pip install rfc5424-logging-handler
```

2.2 Usage

2.2.1 Basic example

After installing you can use this package like this:

```
import logging
from rfc5424logging import Rfc5424SysLogHandler

logger = logging.getLogger('syslogtest')
logger.setLevel(logging.INFO)

sh = Rfc5424SysLogHandler(address=('10.0.0.1', 514))
logger.addHandler(sh)

msg_type = 'interesting'
logger.info('This is an %s message', msg_type)
```

This will send the following message to the syslog server:

```
<14>1 2020-01-01T05:10:20.841485+01:00 myserver syslogtest 5252 - - \xef\xbb\xbfThis␣
↪is an interesting message
```

Note the UTF8 Byte order mark (BOM) preceding the message. While required by RFC 5424 section 6.4 if the message is known to be UTF-8 encoded, there are still syslog receivers that cannot handle it. To bypass this limitation, when initializing the handler Class, set the `msg_as_utf8` parameter to `False` like this:

```
sh = Rfc5424SysLogHandler(address=('10.0.0.1', 514), msg_as_utf8=False)
```

2.2.2 Sending RFC5424 specific fields

The example below sets as many RFC5424 specific fields as possible. If one of the fields isn't specified, a default value or the NIL value is used.

RFC5424 Field	Default value
hostname	Value of `socket.gethostname()`
appname	Name of the logger
procid	`process` attribute of the log record. Normally the process ID of the python application
structured_data	NIL
enterprise_id	None (and raises an error when sending structured data without enterprise ID)
msgid	NIL

Notice that the structured data field can be specified twice. Once when initiating the log handler (for structured data that's sent with every message) and once when sending the message (for structured data specific to this message).

```
import logging
from rfc5424logging import Rfc5424SysLogHandler

logger = logging.getLogger('syslogtest')
logger.setLevel(logging.INFO)

sh = Rfc5424SysLogHandler(
    address=('10.0.0.1', 514),
    hostname="otherserver",
    appname="my_wonderfull_app",
    procid=555,
    structured_data={'sd_id_1': {'key1': 'value1'}},
    enterprise_id=32473
)
logger.addHandler(sh)

msg_type = 'interesting'
extra = {
    'msgid': 'some_unique_msgid',
    'structured_data': {
        'sd_id2': {'key2': 'value2', 'key3': 'value3'}
    }
}
logger.info('This is an %s message', msg_type, extra=extra)
```

That will send the following message to the syslog server:

```
<14>1 2020-01-01T05:10:20.841485+01:00 otherserver my_wonderfull_app 555 some_unique_
↪msgid [sd_id_1@32473 key1="value1"][sd_id2@32473 key3="value3" key2="value2"]␣
↪\xef\xbb\xbfThis is an interesting message
```

If you want the appname, hostname or procid field to be empty, instead of it being determined automatically, set it to `NILVALUE` explicitly. Setting it to `None` or an empty string will cause it to be filled automatically.

```
import logging
from rfc5424logging import Rfc5424SysLogHandler, NILVALUE

logger = logging.getLogger('syslogtest')
logger.setLevel(logging.INFO)

sh = Rfc5424SysLogHandler(
    address=('10.0.0.1', 514),
    hostname=NILVALUE,
    appname=NILVALUE,
    procid=NILVALUE,
)
logger.addHandler(sh)

logger.info('My syslog message')

msg_type = 'interesting'
extra = {
    'msgid': 'some_unique_msgid',
    'structured_data': {
        'sd_id2': {'key2': 'value2', 'key3': 'value3'}
    }
}
logger.info('This is an %s message', msg_type, extra=extra)
```

That will send the following message to the syslog server:

```
<14>1 2020-01-01T05:10:20.841485+01:00 - - - - - \xef\xbb\xbfMy syslog message
```

2.2.3 Log in UTC time

Sometimes you have log sources all over the world in different timezones. In such a case it's sometimes easier to have all you timestamps in the UTC timezone.

You can enable this by setting the `utc_timestamp` argument to `True` like this.

```
from rfc5424logging import Rfc5424SysLogHandler

sh = Rfc5424SysLogHandler(
    address=('10.0.0.1', 514),
    utc_timestamp=True
)
```

2.2.4 TLS/SSL syslog connection

Sometimes logs contain sensitive date and shouldn't go over the network in plain text. For this, you can setup a TLS/SSL connection to the syslog server with the following example.

Check out the code the the `Rfc5424SysLogHandler` class for more options.

```
import logging
from rfc5424logging import Rfc5424SysLogHandler

logger = logging.getLogger('syslogtest')
```

(continues on next page)

(continued from previous page)

```
logger.setLevel(logging.INFO)

sh = Rfc5424SysLogHandler(
    address=('10.0.0.1', 514),
    tls_enable=True,
    tls_verify=True,
    tls_ca_bundle="/path/to/ca-bundle.pem"
)
logger.addHandler(sh)

msg_type = 'interesting'
logger.info('This is an %s message', msg_type)
```

2.2.5 Using a logging config dictionary

Python supports configuring the logging system from a dictionary. Below is an example using the rfc5424 log handler to log to syslog and the stream handler to log to console.

```
import logging
import logging.config

log_settings = {
    'version': 1,
        'formatters': {
        'console': {
            'format': '[%(asctime)s] [%(levelname)s] [%(name)s] %(message)s',
        },
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'console'
        },
        'syslog': {
            'level': 'INFO',
            'class': 'rfc5424logging.handler.Rfc5424SysLogHandler',
            'address': ('127.0.0.1', 514),
            'enterprise_id': 32473,
            'structured_data': {'sd_id_1': {'key1': 'value1'}},
        },
    },
    'loggers': {
        'syslogtest': {
            'handlers': ['console', 'syslog'],
            'level': 'DEBUG',
         },
    }
}
logging.config.dictConfig(log_settings)

logger = logging.getLogger('syslogtest')
logger.info('This message appears on console and is sent to syslog')
logger.debug('This debug message appears on console only')
```

2.2.6 Logger adapter

There's also an `LoggerAdapter` subclass available that makes it easier to send structured data and a message ID or to override fields with every message.

```
import logging
from rfc5424logging import Rfc5424SysLogHandler, Rfc5424SysLogAdapter

logger = logging.getLogger('syslogtest')
logger.setLevel(logging.INFO)

sh = Rfc5424SysLogHandler(address=('10.0.0.1', 514))
logger.addHandler(sh)
adapter = Rfc5424SysLogAdapter(logger)

adapter.info('This is an interesting message',
             structured_data={'sd_id2': {'key2': 'value2', 'key3': 'value3'}})

adapter.info('This is an interesting message',
             msgid='some_unique_msgid')

adapter.info('This is an interesting message',
             structured_data={'sd_id2': {'key2': 'value2', 'key3': 'value3'}},
             msgid='some_unique_msgid')

# Since version 1.0 it's also possible to override the appname, hostname and procid␣
↪per message
adapter.info('Some other message',
             msgid='some_unique_msgid',
             appname="custom_appname",
             hostname="my_hostname",
             procid="5678")
```

2.3 API

2.3.1 Handler

2.3.2 Adapter

2.4 Logstash and RFC5424

Due to the structured format of an RFC5424 it's easy to parse at the receiving side. Below is an example configuration for Logstash (part of the Elastic stack).

I'm interested in more example configurations for parsing RFC5424 with other syslog receivers. If you happen to have such configuration, feel free to open a pull request to have it added.

```
input {
    udp {
        port => 514
        type => "rfc5424"
    }
}
```

(continues on next page)

(continued from previous page)

```
filter {
    if [type] == "rfc5424" {
        grok {
            match => {
                "message" => "<%{NONNEGINT:syslog_pri}>%{NONNEGINT:version}%{SPACE}(?
↪:-|%{TIMESTAMP_ISO8601:syslog_timestamp})%{SPACE}(?:-|%{IPORHOST:hostname})%{SPACE}
↪(?:%{SYSLOG5424PRINTASCII:program}|-)%{SPACE}(?:-|%{SYSLOG5424PRINTASCII:process_id}
↪)%{SPACE}(?:-|%{SYSLOG5424PRINTASCII:message_id})%{SPACE}(?:-|(?<structured_data>
↪(\[.*?[^\\]\])+))(?:%{SPACE}%{GREEDYDATA:syslog_message}|)"
            }
            add_tag => [ "match" ]
        }
        if "match" in [tags] {
            syslog_pri {
                remove_field => "syslog_pri"
            }
            date {
                match => [ "syslog_timestamp", "ISO8601", "MMM dd HH:mm:ss", "MMM dd␣
↪HH:mm:ss.SSS" ]
                remove_field => "syslog_timestamp"
            }
            if [structured_data] {
                ruby {
                    code => '
                        # https://github.com/logstash-plugins/logstash-input-syslog/
↪issues/15#issuecomment-270367033
                        def extract_syslog5424_sd(syslog5424_sd)
                            sd = {}
                            syslog5424_sd.scan(/\[(?<element>.*?[^\\])\]/) do␣
↪|element|
                                data = element[0].match(/(?<sd_id>[^\ ]+)(?<sd_params>
↪ .*)?/)
                                sd_id = data[:sd_id].split("@", 2)[0]
                                sd[sd_id] = {}
                                next if data.nil? || data[:sd_params].nil?
                                data[:sd_params].scan(/ (.*?[=](?:""|".*?[^\\]"))/)␣
↪do |set|
                                    set = set[0].match(/(?<param_name>.*?)[=]\"(?
↪<param_value>.*)\"/)
                                    sd[sd_id][set[:param_name]] = set[:param_value]
                                end
                            end
                            sd
                        end
                        event.set("[sd]", extract_syslog5424_sd(event.get(
↪"[structured_data]")))
                    '
                    remove_field => "structured_data"
                }
            }
        }
    }
}
output {
    elasticsearch {
        hosts => ["localhost:9200"]
        index => "rfc5424-%{+YYYY.MM.dd}"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

2.5 Changelog

2.5.1 1.4.3 - 2019/05/19

Changed

- #34 Allow enterprise ID to contain sub-identifiers.

Fixed

- #31 Correct handling of failed connection attempt in TCP transport handler.

2.5.2 1.4.2 - 2019/04/08

Changed

- #32 `address` can now also be a list, making loading settings form a config file possible.

Added

- Python 3.7 tests and support.

2.5.3 1.4.1 - 2019/02/16

Fixed

- #29 Fix AttributeError when using TLS connection.

2.5.4 1.4.0 - 2019/01/30

Added

- #27 Make it possible to log to streams as an alternate transport.
- Added API documentation.

Changed

- Syslog facilities and framing options have moved from the `RfcSysLogHandler` class to module level variables. **You may have to adjust your references to them.**

2.5.5 1.3.0 - 2018/10/19

Added

- #23 Add support for TLS/SSL

2.5.6 1.2.1 - 2018/09/21

Fixed

- #21 Registered structured data IDs where also suffixed with an enterprise ID.

Added

- #22 Add `utc_timestamp` parameter to allow logging in UTC time.

2.5.7 1.1.2 - 2018/02/03

Fixed

- #15 When logging to `/dev/log` with python 2.7, the connection was permanently lost when the local syslog server was restarted.
- #16 The `extra` info of a message did not overwrite that of the logging adapter instance.

2.5.8 1.1.1 - 2017/12/08

Fixed

- #14 Fixed handling of `extra` parameter in logging adapter.

2.5.9 1.1.0 - 2017/11/24

Added

- The `msg` parameter for the logger handler can now be absent allowing "structured data only" messages.

Fixed

- Correct the automatic value of the `hostname` when the value is anything other then `NILVALUE`
- The syslog message is now empty in conformance with RFC5424 when it's value is `None` or an empty string.

2.5.10 1.0.3 - 2017/10/08

No functional changes. Only documentation was changed.

Added

- Logstash configuration example for RFC5424.

Changed

- Moved most of the documentation out of the readme file.

2.5.11 1.0.2 - 2017/08/31

Fixed

- Package description rendering on PyPi due to bug pypa/wheel#189

2.5.12 1.0.1 - 2017/08/30

Added

- #12: It's now possible to send syslog messages as MSG-ANY which suppresses the UTF-8 byte order mark (BOM) when sending messages.

2.5.13 1.0.0 - 2017/05/30

Changed

- #10: Procid, appname and hostname can now be set per message, both with the handler as well as with the adapter

---

Note: This release has a slight change in behaviour. Setting one of the appnama, hostname of procid message to None or an empty string will cause it to be filled in automatically. Previously, setting it to an empty string caused it to be set to NILVALUE (a -). You now need to set it explicilty to NILVALUE if you want to omit it from the message.

2.5.14 0.2.0 - 2017/01/27

Fixed

- Better input handling
- Better sanitizing of invalid input

2.5.15 0.1.0 - 2017/01/22

Added

- #4: Adapter class to make it easier to log message IDs or structured data
- Logging of EMERGENCY, ALERT and NOTICE syslog levels by using the adapter class
- Extensive test suite

2.5.16 0.0.2 - 2017/01/18

Added

- #5 Introduced Python 2.7 compatibility

2.5.17 0.0.1 - 2017/01/11

- Initial release