logging-gelf Documentation

Release 0.0.12

C.Dumay

May 20, 2019

Contents

A python 3 logging bundle to send logs in Graylog Extended Lenght Format (GELF) . This is a rewrote of Djehouty.

The following example shows how to send log in Graylog TCP input

```
import logging
from logging_gelf.formatters import GELFFormatter
from logging_gelf.handlers import GELFTCPSocketHandler

logger = logging.getLogger("gelf")
logger.setLevel(logging.DEBUG)

handler = GELFTCPSocketHandler(host="127.0.0.1", port=12201)
handler.setFormatter(GELFFormatter(null_character=True))
logger.addHandler(handler)
logger.debug("hello !")
```

CHAPTER 1

Get It Now

First, install logging-gelf using pip:

```
pip install -U logging-gelf
```

CHAPTER 2

Documentation content

2.1 API focus

The basic classes defined by the module, together with their functions, are listed below:

2.1.1 `logging_gelf.handlers` — Handlers

class logging_gelf.handlers.**GELFUDPSocketHandler**

New in version 0.0.7.

This handler send log entries over UDP.

makePickle(*record*)

Pickles the record's attribute dictionary in binary format.

Parameters **record** (*logging.LogRecord*) – record to format

Return type bytes

Basic UDP example

```
>>> import logging
>>> from logging_gelf.handlers import GELFUDPSocketHandler

# we create the logger
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)
>>> handler = GELFUDPSocketHandler(host="127.0.0.1", port=12202)
>>> logger.addHandler(handler)
```

class logging_gelf.handlers.**GELFTCPSocketHandler**

The *GELFTCPSocketHandler*, which inherit from `logging.handlers.SocketHandler`, sends logging output to a TCP network socket.

__init__ (*host*, *port*, *use_tls=False*, *cert_reqs=<ssl.CERT_NONE>*, *ca_certs=None*)
Returns a new instance of the *GELFTCPSocketHandler* class intended to communicate with a remote machine whose address is given by *host* and *port* over TCP.

Parameters

- **use_tls** (*bool*) – Enable TLS communication.
- **cert_reqs** (*enum.IntEnum*) – SSL context virify mode. This attribute must be one of `ssl.CERT_NONE`, `ssl.CERT_OPTIONAL` or `ssl.CERT_REQUIRED` (see ssl doc).
- **ca_certs** (*str*) – File which contains a set of concatenated "certification authority" certificates, which are used to validate certificates passed from the other end of the connection.

makeSocket (*timeout=1*, *after_idle_sec=1*, *interval_sec=3*, *max_fails=5*)
Returns the socket used to send log records.

Parameters

- **timeout** (*float*) – Set a timeout on blocking socket operations, can be a nonnegative floating point number expressing seconds.
- **after_idle_sec** (*int*) – Activates TCP keepalive after *after_idle_sec* second of idleness.
- **interval_sec** (*int*) – Sends a keepalive ping once every *interval_sec* seconds.
- **max_fails** (*int*) – Closes the connection after *max_fails* failed ping (= *max_fails* * *interval_sec*).

Returns a TCP socket.

Return type socket.socket

makePickle (*record*)
Pickles the record's attribute dictionary in binary format.

Parameters **record** (*logging.LogRecord*) – record to format

Return type bytes

Basic TCP example

```
>>> import logging
>>> from logging_gelf.handlers import GELFTCPSocketHandler

# we create the logger
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)
>>> handler = GELFTCPSocketHandler(host="127.0.0.1", port=12201, level=logging.DEBUG)
>>> logger.addHandler(handler)
```

See also:

Logging handlers Logging documentation

Socket Objects Python socket documentation

2.1.2 `logging_gelf.formatters` — Formatters

class logging_gelf.formatters.**GELFFormatter**

A subclass of `logging.Formatter` to format LogRecord into GELF.

__init__(*schema=<logging_gelf.schemas.GelfSchema>*, *null_character=False*, *JSONEncoder=json.JSONEncoder*, *exclude_patterns=None*)

A GELF formatter to format a `logging.LogRecord` into GELF.

Parameters

- **schema** (*logging_gelf.schemas.GelfSchema*) – The marshmallow schema to use to format data.
- **null_character** (*bool*) – Append a '0' at the end of the string. It depends on the input used.
- **JSONEncoder** (*json.JSONEncoder*) – A custom json encoder to use.
- **exclude_patterns** (*list|None*) – List of regexp used to exclude keys

New in version 0.0.12: The *exclude_patterns* parameter.

format(*record*)

Format the specified record into json using the schema which MUST inherit from *logging_gelf.schemas.GelfSchema*.

Parameters **record** (*logging.LogRecord*) – Contains all the information pertinent to the event being logged. :return: A JSON dump of the record. :rtype: str

filter_keys(data):

Filter GELF record keys using exclude_patterns

Parameters **data** (*dict*) – Log record has dict :return: the filtered log record :rtype: dict

New in version 0.0.12.

Testing the output

You can use the `logging.StreamHandler` to test your formatter:

```
>>> import sys
>>> import logging
>>> from logging_gelf.formatters import GELFFormatter

# we create the logger
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)

# we use StreamHandler to display the result
>>> handler = logging.StreamHandler(sys.stdout)
>>> handler.setFormatter(GELFFormatter())
>>> logger.addHandler(handler)

# we send a log entry
>>> logger.debug("hello !")
{"version": "1.1", "host": "host.example.com", "file": "<stdin>", "short_message":
↪"hello !", "timestamp": 1484820522.4268215, "level": 7, "line": 1}
```

The next example uses marshmallow and a custom JSONEncoder which transform all list, tuple or dict to strings:

```
>>> import logging
>>> import sys
>>> from logging_gelf.formatters import GELFFormatter, StringJSONEncoder
>>> from marshmallow import fields, Schema
>>> from logging_gelf.schemas import GelfSchema
>>>
>>> class Person(GelfSchema):
...     lastname = fields.String()
...     father = fields.Nested(Person)
...     firstname = fields.List(fields.String)
...
>>>
>>> me = dict(lastname="Dumay", firstname=["Cedric", "Julien"])
>>>
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)
>>>
>>> handler = logging.StreamHandler(sys.stdout)
>>> handler.setFormatter(
...     GELFFormatter(schema=Person, JSONEncoder=StringJSONEncoder))
>>> logger.addHandler(handler)
>>>
>>> logger.debug("A marshmallow example with Nested", extra=me)
{"host": "host.example.com", "_firstname": "['Cedric', 'Julien']", "file": "<stdin>",
↪"version": "1.1", "short_message": "A marshmallow example with Nested", "timestamp
↪": 1486643773.3877068, "level": 7, "line": 1, "_lastname": "Dumay"}
```

As we can see, firstname is not an array.

See also:

Formatter Objects Official python documentation

2.1.3 `logging_gelf.schemas` — Schemas

class logging_gelf.schemas.**GelfSchema**

Schema which allow to specify a mapping for `logging.LogRecord`. It based on `marshmallow.Schema`. All schema MUST inherit from this.

version
The const *version* specify the GELF version.

host
Hostname which emitted the log record. If not set, `socket.gethostname()` will be used.

short_message
Plain message.

full_message
Extended message

timestamp
`logging.LogRecord` creation time. If `record.created` is not set, current timestamp will be set.

level
Syslog level representation

lineno
Origine line number. This value will be dump into *line* to match GELF spec.

pathname
Origine file pathe. This value will be dump into *file* to match GELF spec.

classmethod to_syslog_level(*value*)
Map `value.levelno` into syslog level.

Parameters **value** (*logging.LogRecord*) – log record to serialize.

Returns syslog level

Return type int

classmethod to_timestamp(*value*)
Returns *value.created* or `time.time()`

Parameters **value** (*logging.LogRecord*) – log record to serialize.

Returns timestamp

Return type float

classmethod to_message(*value*)
Returns the `logging.LogRecord` formatted message.

Parameters **value** (*logging.LogRecord*) – log record to serialize.

Returns entry message

Return type str

fix_additional_fields(*data*)
A "post dump" method which finalize data by prefixing with a "_" the additionals fields.

Note: Only fields set in the model will be serilialized.

Example

```
>>> import logging
>>> from logging_gelf.schemas import GelfSchema
>>> rec = logging.LogRecord(
...  name="test-gelf", level=logging.DEBUG, pathname=None,
...  lineno=None, msg="test", args=list(), exc_info=None
)
>>> GelfSchema().dump(rec).data
{'level': 7, 'line': None, 'host': 'host.example.com', 'short_message': 'test',
↪'version': '1.1', 'file': None, 'timestamp': 1484831977.3012216}
```

Nested fields

As Graylog doesn't support objects, Nested marshmallow fields are "flat unpacked" using a pseudo path in keys:

```
>>> import logging
>>> import sys
>>> from logging_gelf.formatters import GELFFormatter
>>> from marshmallow import fields, Schema
>>> from logging_gelf.schemas import GelfSchema
```

(continues on next page)

(continued from previous page)

```
>>> class Person(Schema):
...     firstname = fields.String()
...
>>> class Family(GelfSchema):
...     lastname = fields.String()
...     father = fields.Nested(Person)
...
>>> familly = dict(lastname="Dumay", father=dict(firstname="Cedric"))
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)
>>> handler = logging.StreamHandler(sys.stdout)
>>> handler.setFormatter(GELFFormatter(schema=Family))
>>> logger.addHandler(handler)
>>> logger.debug("A marshmallow example with Nested", extra=familly)
{"level": 7, "_father_firstname": "Cedric", "short_message": "A marshmallow example␣
↪with Nested", "_lastname": "Dumay", "file": "<stdin>", "host": "host.example.com",
↪"timestamp": 1484919251.3890517, "version": "1.1", "line": 1}
```

Note: As we can see `familly['father']['firstname']` produce a GELF attribute *_father_firstname*

2.2 Guides

2.2.1 Forward logging extra to Graylog

To forward extra send in `logging.LogRecord`, we need customize the marshmallow serializer used in the *logging_gelf.formatters.GELFFormatter*:

```
>>> import sys
>>> import logging
>>> from logging_gelf.formatters import GELFFormatter
>>> from logging_gelf.schemas import GelfSchema
>>> from marshmallow import fields

# we create a custom schema
>>> class MyGelfSchema(GelfSchema):
...     username = fields.String()
...

# we create the logger
>>> logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)

# we use StreamHandler to display the result
>>> handler = logging.StreamHandler(sys.stdout)
>>> handler.setFormatter(GELFFormatter(schema=MyGelfSchema))
>>> logger.addHandler(handler)

# we send a log entry
>>> logger.debug("hello !", extra=dict(username="C.Dumay"))
{"level": 7, "_username": "C.Dumay", "timestamp": 1484842992.1332045, "host": "host.
↪example.com", "version": "1.1", "short_message": "hello !", "file": "<stdin>", "line
↪": 1}
```

Note: As we ca see, the extra var *username* is append as an additional value (prefixed by '_')

2.2.2 Use `logging.LoggerAdapter`

To use logger adapter, you need like the extra on logging event, a custom schema to serialize extra data (see: *Forward logging extra to Graylog*).

```
>>> import sys
>>> import logging
>>> from logging_gelf.formatters import GELFFormatter
>>> from logging_gelf.schemas import GelfSchema
>>> from marshmallow import fields
>>>
>>> # we create a custom schema
... class MyGelfSchema(GelfSchema):
...     username = fields.String()
...
>>> # we create the logger
... logger = logging.getLogger("gelf")
>>> logger.setLevel(logging.DEBUG)
>>>
>>> # we use StreamHandler to display the result
... handler = logging.StreamHandler(sys.stdout)
>>> handler.setFormatter(GELFFormatter(schema=MyGelfSchema))
>>> logger.addHandler(handler)
>>>
>>> # we create an adapter
... adapter = logging.LoggerAdapter(logger=logger, extra=dict(username="C.Dumay"))
>>> adapter.debug("hello !")
{"version": "1.1", "_username": "C.Dumay", "line": 1, "level": 7, "file": "<stdin>",
↪"timestamp": 1484904968.390859, "short_message": "hello !", "host": "host.example.
↪com"}
```

Note: LoggerAdapter extra set at initialization can be overwritten

```
>>> logger.debug("hello !", extra=dict(username="Dude"))
{"version": "1.1", "_username": "Dude", "line": 1, "level": 7, "file": "<stdin>",
↪"timestamp": 1484905204.7358975, "short_message": "hello !", "host": "host.example.
↪com"}
```

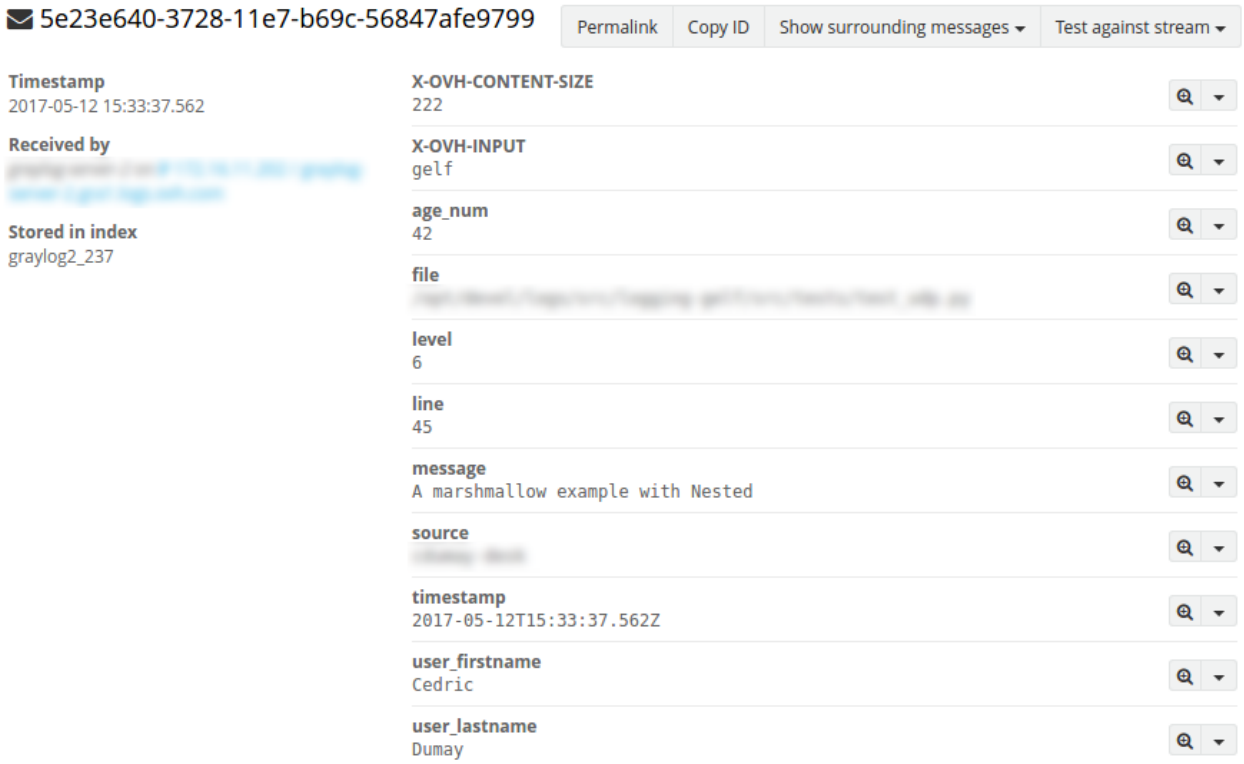
See also:

LoggerAdapter Objects Full python documentation

2.2.3 Send logs to OVH LDP

You can easly send logs to the OVH Logs Data Platform service using an implementation of this library: logging-ldp

Log entry in Graylog



Python Module Index

l

Index

Symbols

F

G

H

L

M

P

S

T

V