graypy Documentation

Release stable

August 31, 2015

Contents

Using easy_install:

```
easy_install graypy
```

Install with requirements for `GELFRabbitHandler`:

```
easy_install graypy[amqp]
```

CHAPTER 1

Usage

Messages are sent to Graylog2 using a custom handler for the builtin logging library in GELF format:

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFHandler('localhost', 12201)
my_logger.addHandler(handler)

my_logger.debug('Hello Graylog2.')
```

Alternately, use `GELFRabbitHandler` to send messages to RabbitMQ and configure your Graylog2 server to consume messages via AMQP. This prevents log messages from being lost due to dropped UDP packets (`GELFHandler` sends messages to Graylog2 using UDP). You will need to configure RabbitMQ with a 'gelf_log' queue and bind it to the 'logging.gelf' exchange so messages are properly routed to a queue that can be consumed by Graylog2 (the queue and exchange names may be customized to your liking):

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFRabbitHandler('amqp://guest:guest@localhost/%2F', 'logging.gelf')
my_logger.addHandler(handler)

my_logger.debug('Hello Graylog2.')
```

Tracebacks are added as full messages:

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFHandler('localhost', 12201)
my_logger.addHandler(handler)

try:
    puff_the_magic_dragon()
```

```
except NameError:
    my_logger.debug('No dragons here.', exc_info=1)
```

CHAPTER 2

Configuration parameters

GELFHandler:

- **host** - the host of the graylog server.
- **port** - the port of the graylog server (default 12201).
- **chunk_size** - message chunk size. messages larger than this size will be sent to graylog in multiple chunks (default *1420*).
- **debugging_fields** - send debug fields if true (the default).
- **extra_fields** - send extra fields on the log record to graylog if true (the default).
- **fqdn** - use fully qualified domain name of localhost as source host (socket.getfqdn()).
- **localname** - use specified hostname as source host.
- **facility** - replace facility with specified value. if specified, record.name will be passed as *logger* parameter.

GELFRabbitHandler:

- **url** - RabbitMQ URL (ex: amqp://guest:guest@localhost:5672/%2F).
- **exchange** - RabbitMQ exchange. Default 'logging.gelf'. A queue binding must be defined on the server to prevent log messages from being dropped.
- **debugging_fields** - send debug fields if true (the default).
- **extra_fields** - send extra fields on the log record to graylog if true (the default).
- **fqdn** - use fully qualified domain name of localhost as source host - socket.getfqdn().
- **exchange_type** - RabbitMQ exchange type (default *fanout*).
- **localname** - use specified hostname as source host.
- **facility** - replace facility with specified value. if specified, record.name will be passed as *logger* parameter.

CHAPTER 3

Using with Django

It's easy to integrate `graypy` with Django's logging settings. Just add a new handler in your `settings.py` like this:

```
LOGGING = {
    ...

    'handlers': {
        'graypy': {
            'level': 'WARNING',
            'class': 'graypy.GELFHandler',
            'host': 'localhost',
            'port': 12201,
        },
    },

    'loggers': {
        'django.request': {
            'handlers': ['graypy'],
            'level': 'ERROR',
            'propagate': True,
        },
    },
}
```

CHAPTER 4

Custom fields

A number of custom fields are automatically added if available:

- function
- pid
- process_name
- thread_name

You can disable these additional fields if you don't want them by adding an argument to the handler:

```
handler = graypy.GELFHandler('localhost', 12201, debugging_fields=False)
```

graypy also supports additional fields to be included in the messages sent to Graylog2. This can be done by using Python's LoggerAdapter and Filter. In general, LoggerAdapter makes it easy to add static information to your log messages and Filters give you more flexibility, for example to add additional information based on the message that is being logged.

Example using LoggerAdapter:

```
import logging
import graypy

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFHandler('localhost', 12201)
my_logger.addHandler(handler)

my_adapter = logging.LoggerAdapter(logging.getLogger('test_logger'),
                                   { 'username': 'John' })

my_adapter.debug('Hello Graylog2 from John.')
```

Example using Filter:

```
import logging
import graypy

class UsernameFilter(logging.Filter):
    def __init__(self):
        # In an actual use case would dynamically get this (e.g. from memcache)
        self.username = "John"
```

```
    def filter(self, record):
        record.username = self.username
        return True

my_logger = logging.getLogger('test_logger')
my_logger.setLevel(logging.DEBUG)

handler = graypy.GELFHandler('localhost', 12201)
my_logger.addHandler(handler)

my_logger.addFilter(UsernameFilter())

my_logger.debug('Hello Graylog2 from John.')
```

Contributors:

- Sever Banesiu
- Daniel Miller