ClusterLogger Documentation

Release 0.1.1

David Zuber

July 28, 2015

Contents

Logging package for contextual information of clusters.

CHAPTER 1

Features

- Hazel Hen logging filter.

1.1 Welcome to ClusterLogger‘s documentation!

CHAPTER 2

Contents:

2.1 Installation

At the command line either via easy_install or pip:

```
$ easy_install clusterlogger
$ pip install clusterlogger
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv clusterlogger
$ pip install clusterlogger
```

2.2 Usage

2.2.1 Filter

Logfilters in this case provide contextual information about the cluster platform.

To use a filter on a logger:

```
import logging
import clusterlogger

logger = logging.getLogger(__name__)
logger.addFilter(clusterlogger.HazelHenFilter)
```

Now logs are sent with contextual information of the specific cluster platform.

2.2.2 Handler

Messages are sent to Graylog using a custom handler for the builtin logging library in GELF format via TCP. For UDP use the graypy package. Some clusters might not allow outgoing UDP connections.:

```
import logging
import clusterlogger

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

```
handler = clusterlogger.GELFTCPHandler('localhost', 12201)
my_logger.addHandler(handler)

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

2.3 Reference

Automatic generated Documenation by apidoc and autodoc.

2.3.1 clusterlogger

Submodules

clusterlogger.handler

GELFTCPHandler(host[, port, ...])	Graylog Extended Log Format handler

Classes

clusterlogger.handler.GELFTCPHandler

class clusterlogger.handler.**GELFTCPHandler**(*host*, *port=12201*, *debugging_fields=True*, *extra_fields=True*, *fqdn=False*, *localname=None*, *facility=None*)

Bases: logging.handlers.SocketHandler

Graylog Extended Log Format handler

This handler uses TCP Sockets.

__init__(*host*, *port=12201*, *debugging_fields=True*, *extra_fields=True*, *fqdn=False*, *localname=None*, *facility=None*)

Initialize a new GELF TCP Handler

Parameters

- **host** (str) – The host of the graylog server.
- **port** (int) – The port of the graylog server (default 12201).
- **debugging_fields** (bool) – Send debug fields if true (the default).
- **extra_fields** (bool) – Send extra fields on the log record to graylog if true (the default).
- **fqdn** (str) – Use fully qualified domain name of localhost as source host (socket.getfqdn()).
- **localname** (str) – Use specified hostname as source host.
- **facility** (str) – Replace facility with specified value. If specified, record.name will be passed as *logger* parameter.

Methods

__init__(host[, port, debugging_fields, ...])	Initialize a new GELF TCP Handler
makePickle(record)	

Attributes

name

makePickle(*record*)

clusterlogger.logfilter

HazelHenFilter()	Filter for adding contextual information on Hazel Hen

Classes

clusterlogger.logfilter.HazelHenFilter

class clusterlogger.logfilter.**HazelHenFilter**

Bases: logging.Filter

Filter for adding contextual information on Hazel Hen

Hazel Hen is the Cray XC40 system at HLRS in Stuttgart.

This filter adds information about the currently running job.

__init__()

Initialize a new HazelHenFilter

Raises None

Methods

__init__()	Initialize a new HazelHenFilter
filter(record)	Add contextual information to the log record

jobid = None

The job identifier assigned to the job by the batch system. This is the same number you see when you do qstat. -1 if logging when not running in a job.

logname = None

Value of the LOGNAME variable in the environment in which qsub was executed

jobname = None

The job name supplied by the user

queue = None

The name of the queue from which the job is executed

fqdn = None

The fully qualified domain name.

sitename = None
The site name of the cluster

platform = None
The cluster platform on the site. E.g. `hazelhen`.

filter (*record*)
Add contextual information to the log record

Parameters **record** (`logging.LogRecord`) – the log record

Returns True, if log should get sent

Return type `bool`

Raises None

Module contents

Data

`clusterlogger.`**absolute_import = _Feature((2, 5, 0, 'alpha', 1), (3, 0, 0, 'alpha', 0), 16384)**

2.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/RayCrafter/clusterlogger/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

ClusterLogger could always use more documentation, whether as part of the official ClusterLogger docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/RayCrafter/clusterlogger/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.4.2 Get Started!

Ready to contribute? Here's how to set up *clusterlogger* for local development.

1. Fork the *clusterlogger* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/clusterlogger.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

2.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/RayCrafter/clusterlogger under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

2.4.4 Tips

To run a subset of tests:

```
$ py.test test/test_clusterlogger.py
```

2.5 Credits

2.5.1 Development Lead

- David Zuber <zuber.david@gmx.de>

2.5.2 Contributors

None yet. Why not be the first?

2.6 History

2.6.1 0.1.0 (2015-07-17)

- First release on PyPI.

2.6.2 0.1.1 (2015-07-18)

- Add `sitename` and `platform` attributes to the log record.

CHAPTER 3

Feedback

If you have any suggestions or questions about **ClusterLogger** feel free to email me at zuber.david@gmx.de.

If you encounter any errors or problems with **ClusterLogger**, please let me know! Open an Issue at the GitHub https://github.com/RayCrafter/clusterlogger main repository.

3.1 Indices and tables

- genindex
- modindex
- search

C

Symbols

A

C

F

G

H

J

L

M

P

Q

S