Metlog-raven Documentation

Release 0.1

Rob Miller, Victor Ng

May 17, 2012

CONTENTS

metlog-raven is a plugin extension for *Metlog <http://github.com/mozilla-services/metlog-py>*. metlog-raven provides logging extensions to capture stacktraces and some frame information such as local variables to faciliate debugging.

The plugin acts as a thin wrapper around the Raven <https://github.com/dcramer/raven> library for Sentry.

More information about how Mozilla Services is using Metlog (including what is being used for a router and what endpoints are in use / planning to be used) can be found on the relevant spec page.

CHAPTER

ONE

WELCOME TO METLOG-RAVEN'S DOCUMENTATION!

1.1 Configuration

Configuration is normally handled through Metlog's configuration system using INI configuration files. A raven plugin must use the *metlog_raven.raven_plugin:config_plugin* as the provider of the plugin. The suffix of the configuration section name is used to set the method name on the Metlog client. Any part after *metlog_plugin_* will be used as the method name.

In the following example, we will bind a method *raven* into the Metlog client so that we can send stacktrace information to the Metlog server.

```
[metlog_plugin_raven]
provider=metlog_raven.raven_plugin:config_plugin
```

You do not need to specify any other options to enable the plugin. The default settings are probably fine for you. You may however set 4 options if you choose.

- **str_length: This is the maximum length of each traceback string.** Default is 200 characters
- **list_length: The number of stack frames which will be captured** by the logger. Default is 50 frames.
- **logger: The name that metlog will use when logging messages. By** default this string is empty
- **payload: The default message that will be sent with each stacktrace.** By default this string is empty
- severity: The default severity of the error. Default is 3 as defined by *metlog.client:SEVERITY.ERROR* <https://github.com/mozilla-services/metlog-py/blob/master/metlog/client.py>

1.2 Usage

Logging exceptions is passive. The raven plugin will never catch an exception without rethrowing it. The plugin simply captures the exception information passes it on to the metlog server.

The raven plugin provides 2 ways to send messages. You can use decorator syntax, or access the plugin through the standard client.

Using the example configuration listed above, the following snippet will log network details.

```
from metlog.decorators.base import CLIENT_WRAPPER
client = CLIENT_WRAPPER.client
```

```
try:
    do_some_exception_throwing_thing()
except SomeExceptionType, se:
    client.raven()
    raise
```

or you can use the decorator syntax

```
from metlog_raven.raven_plugin import capture_stack

@capture_stack
def some_function(foo, bar):
    # Some code here that throws exceptions

...
some_function(x, y)
```

When the decorator syntax is used, exceptions with stack information are based to the metlog backend.

1.3 Data structure

The raven plugin will send quite a lot of information. Important keys to note:

- The type of the message will be set to 'stacktrace'
- The severity is set to 3 (SEVERITY.ERROR)

In the context of determining the source of the error, the 'fields' section of the metlog blob has 2 keys which are of particular interest.

- culprit: This is the function name that threw the exception
- frames: This is a list of stackframes captured. The frames are ordered from inner most to outer most frame. The frame that caused the exception will be at the end of the list.

Each frame is represented as a dictionary with the keys:

> abs_path: Absolute path to the current module context_line: source code of the function where the exception passed through filename: relative path filename of the current module function: function name that the exception passed through lineno: line number in the source module where the exception passed through module: module name pre_context: 2 source lines prior to the exception in the current module post_context: 2 source lines after the exception in the current module: vars: local variables at the time immediately after the exception has been caught

A sample of the JSON emitted is provided below to illustrate all the details that are captured. This blob is generated by the included test suite.

```
{u'env_version': u'0.8',
 u'fields': {u'culprit': u'test_metlog.exception_call2',
             u'frames': [{u'abs_path': u'/Users/victorng/dev/metlog-raven/metlog_raven/raven_plugin.p
                          u'context_line': u'                get_stack_info(iter_traceback_frames(exc
                          u'filename': u'metlog_raven/raven_plugin.py',
                          u'function': u'metlog_call',
                          u'lineno': 37,
                          u'module': u'metlog_raven.raven_plugin',
                          u'post_context': [u'',
                                            u'            culprit = get_culprit(frames)'],
                          u'pre_context': [u'            frames = varmap(lambda k, v: shorten(v,',
                                           u'                string_length=str_length,',
```

```
                                u'                list_length=list_length),'],
                 u'vars': {u'args': [5, 5],
                           u'e': u"ZeroDivisionError('integer division or modulo by zero',)"
                           u'exc_info': [u"<type 'exceptions.ZeroDivisionError'>",
                                         u"ZeroDivisionError('integer division or modulo by
                                         u'<traceback object at 0x10dcf0128>'],
                           u'exc_traceback': u'<traceback object at 0x10dcf0128>',
                           u'exc_type': u"<type 'exceptions.ZeroDivisionError'>",
                           u'exc_value': u"ZeroDivisionError('integer division or modulo by
                           u'kwargs': {},
                           u'list_length': 50,
                           u'self': u'<metlog_raven.raven_plugin.capture_stack object at 0x1
                           u'str_length': 200}},
                          {u'abs_path': u'/Users/victorng/dev/metlog-raven/metlog_raven/tests/test_met
                           u'context_line': u'    return exception_call2(y, x, 42)',
                           u'filename': u'tests/test_metlog.py',
                           u'function': u'exception_call1',
                           u'lineno': 20,
                           u'module': u'test_metlog',
                           u'post_context': [u'', u'@capture_stack'],
                           u'pre_context': [u'',
                                            u'@capture_stack',
                                            u'def exception_call1(x, y):'],
                           u'vars': {u'x': 5, u'y': 5}},
                          {u'abs_path': u'/Users/victorng/dev/metlog-raven/metlog_raven/tests/test_met
                           u'context_line': u'    return a + b + c / (a-b)',
                           u'filename': u'tests/test_metlog.py',
                           u'function': u'exception_call2',
                           u'lineno': 16,
                           u'module': u'test_metlog',
                           u'post_context': [u'', u'@capture_stack'],
                           u'pre_context': [u'import json',
                                            u'',
                                            u'def exception_call2(a, b, c):'],
                           u'vars': {u'a': 5, u'b': 5, u'c': 42}}]},
 u'logger': u'test_metlog:exception_call1',
 u'payload': u'',
 u'severity': 6,
 u'timestamp': u'2012-04-02T17:21:55.135474',
 u'type': u'stacktrace'}
```

1.4 raven_plugin

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*