riemann-client

Release 5.1.0

September 05, 2014

Contents

A Riemann client library and command line tool for Python. It supports UDP and TCP transports, queries, and all metric types. The client library aims to provide a simple, minimal API does not require direct interaction with protocol buffers. There is also a queued client that can queue or batch events and then send them in a single message.

- Source on GitHub
- Documentation on Read the Docs
- Packages on PyPI

CHAPTER 1

Usage

As a command line tool:

```
riemann-client [--host HOST] [--port PORT] send [-s SERVICE] [-S STATE] [-m METRIC] [...]
riemann-client [--host HOST] [--port PORT] query QUERY
```

The host and port used by the command line tool can also be set with the `RIEMANN_HOST` and `RIEMANN_PORT` environment variables. By default, `localhost:5555` will be used.

As a library:

```
import riemann_client.client

with riemann_client.client.Client() as client:
        client.event(service="riemann-client", state="awesome")
        client.query("service = 'riemann-client'")
```

A more detailed example, using both a non-default transport and a queued client:

```
from riemann_client.transport import TCPTransport
from riemann_client.client import QueuedClient

with QueuedClient(TCPTransport("localhost", 5555)) as client:
        client.event(service="one", metric_f=0.1)
        client.event(service="two", metric_f=0.2)
        client.flush()
```

The `QueuedClient` class modifies the `event()` method to add events to a queue instead of immediately sending them, and adds the `flush()` method to send the current event queue as a single message.

1.1 Client API

Clients manage the main user facing API, and provide functions for sending events and querying the Riemann server. UDP, TCP and TLS transports are provided by the `riemann_client.transport` module, and the protocol buffer objects are provided by the `riemann_client.riemann_pb2` module.

class `riemann_client.client.`**`Client`**(*transport=None*)

Bases: `object`

A client for sending events and querying a Riemann server.

Two sets of methods are provided - an API dealing directly with protocol buffer objects and an extended API that takes and returns dictionaries representing events.

Protocol buffer API:

- send_event()
- send_events()
- send_query()

Extended API:

- event()
- events()
- query()

Clients do not directly manage connections to a Riemann server - these are managed by riemann_client.transport.Transport instances, which provide methods to read and write messages to the server. Client instances can be used as a context manager, and will connect and disconnect the transport when entering and exiting the context.

```
>>> with Client(transport) as client:
...     # Calls transport.connect()
...     client.query('true')
...     # Calls transport.disconnect()
```

static create_dict(*event*)

Translates an Event object to a dictionary of event attributes

All attributes are included, so create_dict(create_event(input)) may return more attributes than were present in the input.

Parameters **event** – A protocol buffer Event object

Returns A dictionary of event attributes

static create_event(*data*)

Translates a dictionary of event attributes to an Event object

Parameters **data** (*dict*) – The attributes to be set on the event

Returns A protocol buffer Event object

event(**data*)

Sends an event, using keyword arguments to create an Event

```
>>> client.event(service='riemann-client', state='awesome')
```

Parameters ****data** – keyword arguments used for create_event()

Returns The response message from Riemann

events(**events*)

Sends multiple events in a single message

```
>>> client.events({'service': 'riemann-client', 'state': 'awesome'})
```

param *events event dictionaries for create_event()

returns The response message from Riemann

query(*query*)

Sends a query to the Riemann server

```
>>> client.query('true')
```

Returns A list of event dictionaries taken from the response

Raises Exception if used with a `UDPTransport`

send_event(*event*)
Sends a single event to Riemann

Parameters **event** – An `Event` protocol buffer object

Returns The response message from Riemann

send_events(*events*)
Sends multiple events to Riemann in a single message

Parameters **events** – A list or iterable of `Event` objects

Returns The response message from Riemann

send_query(*query*)
Sends a query to the Riemann server

Returns The response message from Riemann

class riemann_client.client.**QueuedClient**(*transport=None*)
Bases: `riemann_client.client.Client`

A Riemann client using a queue that can be used to batch send events.

A message object is used as a queue, with the `send_event()` and `send_events()` methods adding new events to the message and the `flush()` sending the message.

clear_queue()
Resets the message/queue to a blank `Msg` object

flush()
Sends the waiting message to Riemann

Returns The response message from Riemann

send_event(*event*)
Adds a single event to the queued message

Returns None - nothing has been sent to the Riemann server yet

send_events(*events*)
Adds multiple events to the queued message

Returns None - nothing has been sent to the Riemann server yet

1.2 Transport API

Transports are used for direct communication with the Riemann server. They are usually used inside a `Client`, and are used to send and receive protocol buffer objects.

class riemann_client.transport.**BlankTransport**
Bases: `riemann_client.transport.Transport`

A transport that collects messages in a list, and has no connection

Used by `--transport none`, which is useful for testing commands without contacting a Riemann server. This is also used by the automated tests in `riemann_client/tests/test_riemann_command.py`.

connect()
Creates a list to hold messages

disconnect()
Clears the list of messages

send(*message*)
Adds a message to the list, returning a fake 'ok' response

Returns A response message with `ok = True`

exception riemann_client.transport.**RiemannError**
Bases: exceptions.Exception

Raised when the Riemann server returns an error message

class riemann_client.transport.**SocketTransport**(*host='localhost'*, *port=5555*)
Bases: riemann_client.transport.Transport

Provides common methods for Transports that use a sockets

address

Returns A tuple describing the address to connect to

Return type (host, port)

socket
Returns the socket after checking it has been created

class riemann_client.transport.**TCPTransport**(*host='localhost'*, *port=5555*, *timeout=None*)
Bases: riemann_client.transport.SocketTransport

Communicates with Riemann over TCP

Parameters

- **host** (*str*) – The hostname to connect to
- **port** (*int*) – The port to connect to
- **timeout** (*int*) – The time in seconds to wait before raising an error

connect()
Connects to the given host

disconnect()
Closes the socket

send(*message*)
Sends a message to a Riemann server and returns it's response

Parameters **message** – The message to send to the Riemann server

Returns The response message from Riemann

Raises RiemannError if the server returns an error

class riemann_client.transport.**TLSTransport**(*host='localhost'*, *port=5555*, *timeout=None*, *ca_certs=None*)
Bases: riemann_client.transport.TCPTransport

Communicates with Riemann over TCP + TLS

Options are the same as TCPTransport unless noted

Parameters **ca_certs** (*str*) – Path to a CA Cert bundle used to create the socket

connect()
Connects using `TLSTransport.connect()` and wraps with TLS

class riemann_client.transport.**Transport**
Bases: `object`

Abstract transport definition

Subclasses must implement the `connect()`, `disconnect()` and `send()` methods.

Can be used as a context manager, which will call `connect()` on entry and `disconnect()` on exit.

connect()

disconnect()

send()

class riemann_client.transport.**UDPTransport**(*host='localhost'*, *port=5555*)
Bases: `riemann_client.transport.SocketTransport`

connect()
Creates a UDP socket

disconnect()
Closes the socket

send(*message*)
Sends a message, but does not return a response

Returns None - can't receive a response over UDP

riemann_client.transport.**socket_recvall**(*socket*, *length*, *bufsize=4096*)
A helper method to read of bytes from a socket to a maximum length

CHAPTER 2

Installation

`riemann-client` requires Python 2.6 or above, and can be installed with `pip install riemann-client`. It will use Google's protobuf library when running under Python 2, and GreatFruitOmsk's protobuf-py3 when running under Python 3. Python 3 support is experimental and is likley to use Google's *protobuf* once it supports Python 3 fully.

2.1 Requirements

- click
- protobuf (when using Python 2)
- protobuf (when using Python 3)

CHAPTER 3

Changelog

3.1 Version 5.1.0

- Added Python 3 support
- Changed `riemann_client.riemann_pb2` to wrap `_py2` and `_py3` modules
- Changed `setup.py` to dynamically select a `protobuf` dependency

3.2 Version 5.0.x

- Added API documentation (riemann-client.readthedocs.org)
- Replaced `argparse` with `click` for an improved CLI
- Various command line parameters changed
- `--event-host` became `--host`
- `--print` was removed, `send` always prints the sent event
- Minor fixes to `QueuedClient` API
- `UDPTransport.send` returns `None` instead of `NotImplemented`

3.3 Version 4.2.x

- Added `events()` and `send_events()` methods to the client
- Added `clear_queue()` method to the queued client
- Add `--timeout` option for TCP based transports

3.4 Version 4.1.x

- Full Riemann protocol support (TLS transport, event attributes)
- Fixes for multiple broken features (`--tags`, `--print`)
- Raise errors when clients are used incorrectly

- Client displays errors from Riemann nicely
- Relaxed version requirements to fit CentOS 6 packages

3.5 Version 3.0.x

- Renamed module from `riemann` to `riemann_client`
- Command line interface was rewritten, and is now the only part of the library that respects the `RIEMANN_HOST` and `RIEMANN_PORT` environment variables
- Support for querying the Riemann index was added
- Internally, transports now define `send` instead of `write`, and `TCPTransport.send` returns Riemann's response message

CHAPTER 4

Licence

`riemann-client` is licensed under the MIT Licence. The protocol buffer definition is sourced from the Riemann Java client, which is licensed under the Apache Licence.

CHAPTER 5

Authors

`riemann-client` was written by Sam Clements, while working at DataSift.

r