feedr Documentation

Release 0.5.1

nir0s

January 20, 2015

Contents

feedr (yes, I know, it's a bad name. I'll change it when I come up of something better) can generate custom data and send it away..

It can randomly generate data or get its format and data from a configuration file.

You could use feedr to test your logstash configuration, your elasticsearch cluster performance, load test your RabbitMQ cluster, your InfluxDB handlers.. you get the gist.

feedr even comes with built in Apache Access log (regular and extended) and Apache error log formatters for you to check how you're handling apache logs (and you don't even need apache to do that. how cool is that?)

feedr uses a configuration file in which you can configure your formatters and transports to your liking.

Contents:

CHAPTER 1

Quick Start

well... all you really need is python and pip.. and..

```
pip install feedr
```

then you can run

```
mouth feed -m 100
```

and VOILA! (look in your current dir for a "generated.log" file.)

to go beyond *mouth feed*, run:

```
mouth -h
mouth feed # to send send infinite amount of messages
mouth list transports # to list the default transports available to you
mouth list formatters # to list the default formatters available to you
mouth list fake # for a list of fake data fields you can use
```

and to configure feedr, see the configuration section.

Note: you can use the supplied vagrantfile which will load a vm with feedr for you..(soon, it will also contain rabbitmq, logstash, elasticsearch and kibana so that you can do some more play.)

CHAPTER 2

Configuration

feedr's config is a python file with a single dictionary called GENERATOR. (It's a python file (rathern than JSON) to allow you to use code when generating the data or configuring the transports.)

In the configuration file, you can configure your formatters and transports.

feedr will, by default, look for a config.py file in your current working directory unless the "-c" flag is used which will let you specify a specific path. If no config file is found, default configuration will be used.

```
GENERATOR = {
    'formatters': {
        'MyJsonFormatter': {
            'type': 'Json',
            'data': {
                'date_time': ['15-04-2014 10:00:00'],
                'level': ['ERROR', 'DEBUG'],
                'name': ['myname', 'notmyname'],
            }
        },
        'MyCustomFormatter': {
            'type': 'Custom',
            'format': ['date_time', ' ', 'level', ': ', 'module', ' - ', 'free_email'],
            'data': {
                'date_time': ['15-04-2014 10:00:00'],
                'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
                'module': ['module1', 'module2'],
                'free_email': ['mymail@gmail.com'],
            }
        },
    },
    'transports': {
        'MyAmqpTransport': {
            'type': 'Amqp',
            'host': 'localhost',
            'queue': 'myqueue',
            'exchange': '',
            'routing_key': 'myroutingkey',
        },
        'MyFileTransport': {
            'type': 'File',
            'file': 'generated.log',
            'max_bytes': 100000000,
            'backup_count': 20,
        },
```

```
        'MyUDPTransport': {
            'type': 'UDP',
            'host': 'localhost',
            'port': 999,
        }
    },
}
```

2.1 Advanced Config

Since the config is really just a python file, you can pretty much do anything with it. For instance, you could randomize a uuid field as shown below.

Additionaly, you can use the special $RAND string to randomize data.

Note: The $RAND string can only be used for fields that are supported by feedr. You can run "feedr list fake" to see which fields can be randomized.

Note: Remember that the entire data dict is kept in memory, so don't randomize millions of data objects or they will hog your resources.

```
import uuid

'formatters': {
    'MyCustomFormatter': {
        'format': ['date_time', ' ', 'uuid', ' ', 'level', ': ', 'module', ' - ', 'free_email'],
        'data': {
            'date_time': '$RAND',
            'uuid': [str(uuid.uuid1()) for i in xrange(100)],
            'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
            'module': ['module1', 'module2'],
            'free_email': '$RAND',
        }
    },
},
```

Tip: to generate real unix time fields, use the current_date_time field. this can help you immitate real time event generation.

Tip: check this out for a configuration file example.

CHAPTER 3

Formatters

Formatters create a data structure and applies data into it.

Formatters are classes that contain the logic of how to format your data. They receive data and a format based on your configuration or fake data for you.

Developing new formatters is as easy as as baking a pop-tart. All you have to do is implement a new class and provide your format and data structures. You can check the Apache Access Log implementation as a reference.

To list all formatters run:

```
mouth list formatters
```

Contents:

3.1 The Custom Formatter

Allows configuring a custom formatter... which will pretty much do whatever you want. YES, YOU CAN BE A GOD!

3.1.1 Configuration Example

```
'MyCustomFormatter': {
    'type': 'Custom',  # formatter type (MANDATORY)
    'format': ['date_time', ' ', 'uuid', ' ', 'level', ': ', 'module', ' - ', 'free_email'],
    # format is an ordered list of fields that will be used to create your message string.
    # there is a default format.. so this is OPTIONAL.
    'data': {
        'date_time': '$RAND',
        'uuid': [str(uuid.uuid1()) for i in xrange(3)],
        'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
        'module': ['module1', 'module2'],
        'free_email': '$RAND',
    }
    # the data corresponds to the format by the name of the key.
    # again, there is default data for testing, so this is OPTIONAL.
},
```

3.1.2 Example Output

```
2014-07-10 15:44:31 ef00aa3e-082f-11e4-8e1a-843a4bd58c5c INFO: module2 - fay.wilkie@hotmail.com
2014-07-10 15:44:31 ef01261c-082f-11e4-8e1a-843a4bd58c5c INFO: module1 - feil.alfreda@hotmail.com
2014-07-10 15:44:31 ef008e5a-082f-11e4-8e1a-843a4bd58c5c ERROR: module1 - clark02@gmail.com
```

3.2 The Json Formatter

Allows configuring a json formatter.

3.2.1 Configuration Example

```
'MyJsonFormatter': {
    'type': 'Custom',  # formatter type (MANDATORY)
    'data': {
        'date_time': '$RAND',
        'uuid': [str(uuid.uuid1()) for i in xrange(3)],
        'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
        'module': ['module1', 'module2'],
        'free_email': '$RAND',
    },
    'jsonify': False,  # should the dict be converted to JSON? (OPTIONAL - defaults to True)
    # there is some default data for testing so this is OPTIONAL.
},
```

3.2.2 Example Output

```
{"free_email": "irvin.corwin@gmail.com", "current_date_time": "2014-07-10 15:45:36", "uuid": "160a785
{"free_email": "torp.wiley@hotmail.com", "current_date_time": "2014-07-10 15:45:36", "uuid": "160a57c
{"free_email": "aflatley@yahoo.com", "current_date_time": "2014-07-10 15:45:36", "uuid": "160a1386-08
```

3.3 The Apache Access Formatter

Allows immitating Apache Access logs.

3.3.1 Configuration Example

```
'MyApacheAccessFormatter': {
    'type': 'ApacheAccess',  # formatter type (MANDATORY)
    'data': {
        'ipv4': '$RAND',
        'current_day_of_month': '$RAND',
        'current_month_name_short': '$RAND',
        'current_year': '$RAND',
        'current_time': '$RAND',
        'current_time_zone_number': '$RAND',
        'http_versions': '$RAND',
        'http_verbs': '$RAND',
        'uri_path': '$RAND',
```

```
        'http_error_codes': '$RAND',
        'random_int': '$RAND',
    },
    # By default, all fields above will be randomized.
    # You CAN override them by supplying specific lists of data for a field.
},
```

3.3.2 Example Output

```
87.48.231.47 - - [10/Jul/14:15:49:07 +0000] "HEAD /app HTTP/1.1" 501 8959
216.205.174.8 - - [10/Jul/14:15:49:07 +0000] "POST /main HTTP/1.0" 503 2483
48.70.200.122 - - [10/Jul/14:15:49:07 +0000] "POST /posts/wp-content/tags HTTP/1.1" 302 322
```

3.4 The Apache Access Extended Formatter

Allows immitating Apache Access Extended logs. This is an exact replica of the Apache Access formatter with 2 added fields.

3.4.1 Configuration Example

```
'MyApacheAccessExFormatter': {
    'type': 'ApacheAccessEx',  # formatter type (MANDATORY)
    'data': {
        'ipv4': '$RAND',
        'current_day_of_month': '$RAND',
        'current_month_name_short': '$RAND',
        'current_year': '$RAND',
        'current_time': '$RAND',
        'current_time_zone_number': '$RAND',
        'http_versions': '$RAND',
        'http_verbs': '$RAND',
        'uri_path': '$RAND',
        'http_error_codes': '$RAND',
        'random_int': '$RAND',
        'uri': '$RAND',
        'user_agent': '$RAND',
    },
    # By default, all fields above will be randomized.
    # You CAN override them by supplying specific lists of data for a field.
},
```

3.4.2 Example Output

```
98.156.136.253 - - [29/Jul/14:08:16:35 +0000] "PUT /category/explore HTTP/1.1" 200 6212 "http://www.s
21.192.37.113 - - [29/Jul/14:08:16:35 +0000] "OPTIONS /app/search HTTP/1.0" 400 6961 "http://walker.c
143.223.90.90 - - [29/Jul/14:08:16:35 +0000] "POST /search/app/app HTTP/1.0" 404 2031 "http://www.job
```

3.5 The Apache Error Formatter

Allows immitating Apache Error logs.

3.5.1 Configuration Example

```
'MyApacheAccessFormatter': {
    'type': 'ApacheAccess',  # formatter type (MANDATORY)
    'data': {
        'current_day_of_week_short': '$RAND',
        'ipv4': '$RAND',
        'current_day_of_month': '$RAND',
        'current_month_name_short': '$RAND',
        'current_year': '$RAND',
        'current_time': '$RAND',
        'catch_phrase': '$RAND',
        'syslog_error_levels_lower': '$RAND',
   },
    # By default, all fields above will be randomized.
    # You CAN override them by supplying specific lists of data for a field.
},
```

3.5.2 Example Output

```
[Thu Jul 10 15:52:39 14] [error] [client 32.23.226.214] Implemented user-facing support
[Thu Jul 10 15:52:39 14] [error] [client 73.130.255.174] Integrated bi-directional structure
[Thu Jul 10 15:52:39 14] [critical] [client 120.20.243.249] Secured intangible time-frame
```

feedr.formatters.**fake_data**(*data_type*)
returns fake data for the data type requested.

will try and get fake data from the local format mappings. if fake data for the specific data type wasn't found, it will try and use fake-factory to fake the data.

Parameters **data_type** (*string*) – the type of data to fake.

Return type string

class feedr.formatters.**BaseFormatter**(*config*)
Bases: object

base class for all formatters

generate_data()

f(*config*, *name*)
retrieves configuration keys

if config wasn't supplied or a key doesn't exist, returns $RAND which will initiate data faking

class feedr.formatters.**CustomFormatter**(*config*)
Bases: feedr.formatters.BaseFormatter

returns a generated log string in a custom format

this is also a formatter other formatters can rely on to generate application specific logs. see the ApacheAccessFormatter class for reference.

for every item in the format list, if an item in the data dict corresponds with it and the field's data equals "$RAND", use faker to fake an item for it. else, choose one item from the list randomly. if there no item in the data to correspond with the format, it will just append to format's field name to the log.

example:

```
'CustomFormatter': {
    'format': ['name', ' - ', 'level'],
    'data': {
        'name': $RAND,
        'level': ['ERROR', 'DEBUG', 'INFO', 'CRITICAL'],
    }
}
```

the output of the above example might be:

```
Sally Fields - ERROR
or
Jason Banks - DEBUG
or
Danny Milwee - ERROR
or
...
```

generate_data()

this will generate a message according to *self.format* with data from *self.data*.

all fields in the data dict will be iterated over and matched to the items in the format list. if a match is found and $RAND is set in one of the fields, random data will be generated for that field. If not, data will be chosen from the list. If no match is found, the explicit item in the format list will be appended.

example:

```
format = ['Mr. ' 'first_name', 'last_name']
data = {
    'first_name': ['Jason, Josh]',
    'last_name': '$RAND'
}
```

the output of the above example might be:

```
'Mr. Jason Williams'
or
'Mr. Josh Brolin'
or
'Mr. Jason Bananas'
...
```

class feedr.formatters.**JsonFormatter**(*config*)

Bases: feedr.formatters.BaseFormatter

generates log strings in json format (or leave as dict)

all fields in the data dict will be iterated over. if $RAND is set in one of the fields, random data will be generated for that field. If not, data will be chosen from the list.

example:

```
'JsonFormatter': {
    'data': {
        'date_time': '$RAND',
        'level': ['ERROR', 'DEBUG'],
```

```
            'address': '$RAND',
        }
    },
```

the output of the above example might be:

```
{"date_time": "2006-11-05 13:31:09", "name": "Miss Nona Breitenberg DVM", "level": "ERROR"}  # N
or
{"date_time": "1985-01-20 11:41:16", "name": "Almeda Lindgren", "level": "DEBUG"}  # NOQA
or
{"date_time": "1973-05-21 01:06:04", "name": "Jase Heaney", "level": "DEBUG"}  # NOQA
or
...
```

generate_data()

class feedr.formatters.**ApacheAccessFormatter**(*config*)
Bases: feedr.formatters.CustomFormatter

returns an apache-access-log like string

you can easily construct new formatters by inheriting the custom formatter.

all you have to do is specify the format and the data. a helper method *f* is supplied in the *BaseFormatter* Class that will allow you to retrieve basic formatter configuration for your fields.

class feedr.formatters.**ApacheAccessExFormatter**(*config*)
Bases: feedr.formatters.CustomFormatter

returns an apache-extended-access-log like string

class feedr.formatters.**ApacheErrorFormatter**(*config*)
Bases: feedr.formatters.CustomFormatter

returns an apache-error-log like string

CHAPTER 4

Transports

Transports are the methods in which logs are sent.

transports are good for several things: configuring a client for transportation, writing/sending the log/event, and returning statistical data about your run.

a transport class must return a client for feedr to use and be able to use that client to send a log generated by the formatter. returning statistical data isn't mandatory.. but recommended.

implementing new transports is as simple as implementing a transport - easy.

to see a list of all transports run:

```
mouth list transports
```

Note: while some transports have a default configuration (like the File transport), some don't (like the UDP transport). see the transports section for configuration options for the desired transport.

Contents:

4.1 The File Transport

Allows writing to a file.

Note: Currently, only a rotating file transport is supported.

4.1.1 Configuration Example

```
'MyFileTransport': {
    'type': 'File',  # transport type (MANDATORY)
    'file': 'mylogfile.log',  # the path to the file to write the data to. (OPTIONAL - default is ge
    'max_bytes': 102301202,  # the size of the file afterwhich it is rotated. (OPTIONAL - default is
    'backup_count': 5,  # number of files to keep in rotation. (OPTIONAL - default is 20)
},
```

4.2 The UDP Transport

Allows sending messages via udp.

4.2.1 Configuration Example

```
'MyUDPTransport': {
    'type': 'UDP',  # transport type (MANDATORY)
    'host': 'www.google.com',  # the host to send to (MANDATORY)
    'port': '9921',  # the port to send to (MANDATORY)
},
```

4.3 The Stream Transport

Allows streaming messages to your shell's standard output.

Currently, requires no configuration. Later, you'll be able to specify which stream to write to (stdout, stderr).

4.4 The AMQP Transport

Allows sending messages via the amqp protocol using pika.

4.4.1 Configuration Example

```
'MyAMQPTransport': {
    'type': 'AMQP',  # transport type (MANDATORY)
    'host': 'localhost',  # host to send to (MANDATORY)
    'queue': 'nice_queue',  # queue to write to (OPTIONAL - default is an empty string)
    'exchange': 'nice_exchange',  # exchange to use (OPTIONAL - default is not to use one)
    'routing_key': 'nice_key',  # routing key to use (OPTIONAL - default is the queue name)
    'exchange_type': 'topic',  # exchange type to use (OPTIONAL - defaults to fanout)
    'delivery_mode': 2,  # pika delivery mode (OPTIONAL - defaults to 2)
    'durable': True,  # whether the queue should be durable and survive server restarts (OPTIONAL - d
    'auto_delete': True,  # should the queue auto delete itself once there are no messages in it (OPT
    'exclusive': False,  # should messages be exclusive to the client who published them (OPTIONAL -
},
```

4.5 The Elasticsearch Transport

Allows indexing documents in Elasticsearch.

4.5.1 Configuration Example

```
'MyElasticsearchTransport': {
    'type': 'Elasticsearch',  # transport type (MANDATORY)
    'host': 'myelasticsearchcluster.com',  # the host to send to (MANDATORY)
    'port': '9921',  # the port to send to (OPTIONAL - defaults to 9200)
    'url_prefix': 'yo',  # url prefix to append (OPTIONAL - defaults to an empty string)
    'timeout': 1,  # connection timeout (OPTIONAL - defaults to 10)
    'index': 'myindex',  # index to write to (OPTIONAL - defaults to 'logstash-YYYY.MM.dd' to)
    'doc_type': 'mydoctype',  # doc type to write (OPTIONAL - defaults to 'doc')
    'sleep': 3  # sleep time before printing index docs count (OPTIONAL - defaults to 3)
},
```

Note: The *sleep* parameter is a workaround for the time it takes to index all documents before printing the documents count. It will be removed in the future and a smart mechanism will be implemented.

4.6 The Logentries Transport

Allows sending messages to logentries.

Note: Follow this (configure section) to get your token.

4.6.1 Configuration Example

```
'MyLogentriesTransport': {
    'type': 'Logentries',  # transport type (MANDATORY)
    'token': '94f0e5d3-ad52-4d49-b4f1-feb956cc1111' (MANDATORY)
},
```

4.7 The Loggly Transport

Allows sending messages to Loggly.

Note: You can get your token by logging in to Loggly and going to http://#USERNAME#.loggly.com/tokens. Alternatively, from the home screen, go to "Source Setup" and then to "Customer Tokens".

4.7.1 Configuration Example

```
'MyLogglyTransport': {
    'type': 'Loggly',  # transport type (MANDATORY)
    'domain': 'logs.loggly.com',  # (OPTIONAL - defaults to logs-01.loggly.com)
    'token': '68660f49-ad27-4521-9dad-0d9d54924111',  # (MANDATORY)
},
```

4.8 The MongoDB Transport

Allows indexing documents in MongoDB.

4.8.1 Configuration Example

```
'MyMongoDBTransport': {
    'type': 'MongoDB',  # transport type (MANDATORY)
    'host': 'mymongocluster.com',  # the host to send to (MANDATORY)
    'port': '27100',  # the port to send to (OPTIONAL - defaults to 9200)
    'db': 'mytestdb',  # db to write to (OPTIONAL - defaults to 'test')
    'collection': 'mytestcollection',  # collection to write to (OPTIONAL - defaults to 'my_collecti
```

```
    'sleep': 3  # sleep time before printing index docs count (OPTIONAL - defaults to 3)
},

.. note:: The `sleep` parameter is a workaround for the time it takes to index all documents before p
```

4.9 The InfluxDB Transport

Allows indexing metrics in InfluxDB.

4.9.1 Configuration Example

```
'MyInfluxDBTransport': {
    'type': 'InfluxDB',  # transport type (MANDATORY)
    'host': '10.2.2.7',  # the host to send to (MANDATORY)
    'user': 'root',  # the influx database user
    'password': 'root',  # the influx database password
    'database': 'metrics'  # the database to send to
}
```

You can define a *Json* formatter to send metrics to influx

```
'MyInfluxDBFormatter': {
    'type': 'Json',
    'data': {
        'points': [[[1.1,4.3,2.1],[1.2,2.0,2.0]]],
        'name': ["web_devweb01_load"],
        'columns': [["min1", "min5", "min15"]]
    },
    'jsonify': False,
},
```

Note: Make sure jsonify is False. Additionally, note that you cannot currently send a list of of timeseries simultaneously. What you CAN do, use randomize different timeseries by providing multiple values in the *name* field.

---

class feedr.transports.**BaseTransport**(*config*)
Bases: object

configure()

send(*client*, *log*)

class feedr.transports.**FileTransport**(*config*)
Bases: feedr.transports.BaseTransport

a RotatingFileHandler transport implementation

configure()

send(*client*, *data*)

close()

get_data()

class feedr.transports.**AMQPTransport**(*config*)
Bases: feedr.transports.BaseTransport

an amqp transport implementation

configure()

send(*client*, *data*)

close()

class feedr.transports.**UDPTransport**(*config*)
Bases: feedr.transports.BaseTransport

a udp transport implementation

configure()

send(*client*, *data*)

close()

class feedr.transports.**StreamTransport**(*config*)
Bases: feedr.transports.BaseTransport

a shell stream transport implementation

configure()

send(*client*, *data*)

class feedr.transports.**ElasticsearchTransport**(*config*)
Bases: feedr.transports.BaseTransport

an Elasticsearch transport implementation

configure()

send(*client*, *data*)

close()

get_data()

class feedr.transports.**LogentriesTransport**(*config*)
Bases: feedr.transports.BaseTransport

a logentries transport implementation

configure()

send(*client*, *data*)

close()

class feedr.transports.**LogglyTransport**(*config*)
Bases: feedr.transports.BaseTransport

a Loggly transport implementation

configure()

send(*client*, *data*)

close()

class feedr.transports.**MongoDBTransport**(*config*)
Bases: feedr.transports.BaseTransport

a MongoDB transport implementation

configure()

send(*client*, *data*)

close()

get_data()

class feedr.transports.**InfluxDBTransport**(*config*)
Bases: feedr.transports.BaseTransport

an InfluxDB transport implementation

configure()

send(*client*, *data*)

close()

get_data()

CHAPTER 5

API

Contents:

feedr.feedr.**get_current_time**()
returns the current time (no microseconds tho)

feedr.feedr.**calculate_throughput**(*elapsed_time*, *messages*)
calculates throughput and extracts the number of seconds for the run from the elapsed time

Parameters

- **elapsed_time** – run time
- **messages** (*int*) – number of messages to write

Returns throughput and seconds

Return type tuple

feedr.feedr.**send**(*instance*, *client*, *formatter*, *format_config*, *messages*, *gap*, *batch*)
sends data and prints the time it took to send it

Parameters

- **instance** – transport class instance
- **client** – client to use to send send
- **format** (*string*) – formatter to use
- **format_config** (*dict*) – formatter configuration to use
- **messages** (*int*) – number of messages to send
- **gap** (*float*) – gap in seconds between 2 messages
- **batch** (*int*) – number of messages per batch

feedr.feedr.**config_transport**(*transports*, *transport*, *transport_config*)
returns a configured instance and client for the transport

Parameters

- **transports** – transport classes to choose from.
- **transport** (*string*) – transport to use
- **transport_config** (*dict*) – transport configuration

feedr.feedr.**generator**(*config=None*, *transport=None*, *formatter=None*, *gap=None*, *messages=0*, *batch=1*, *verbose=False*)

generates data

this will generate data in the requested format and protocol.

Parameters

- **config** (*string*) – path to config file path
- **transport** (*string*) – transport type to use
- **formatter** (*string*) – formatter to use
- **gap** (*float*) – gap in seconds between 2 messages
- **messages** (*int*) – number of messages to send
- **batch** (*int*) – number of messages to stack before sending
- **verbose** (*bool*) – sets verbose state for internal logging.

feedr.feedr.**list_fake_types**()

prints a list of random data types with an example

feedr.feedr.**list_transports**()

feedr.feedr.**list_formatters**()

exception feedr.feedr.**FeedrError**

Bases: exceptions.Exception

CHAPTER 6

InHouseFaker

feedr uses fake factory to generate most of its random data. While it provides a large number of useful random fields, some data types are missing from it.

I do want to send some pull requests with updates to fake-factory.. but for now.. an InHouseFaker was implemented. The InHouseFaker class in the format_mappings module returns random data for some data types if they cannot be generated by fake-factory.

You can easily contribute to it though I'd suggest contributing to fake-factory instead as feedr isn't meant to provide this functionality for a long term.

CHAPTER 7

Use Case

To show off the strength feedr possesses, let's take an example that has nothing to do with logs.

You're now leading a monitoring project where you need to build your first Graphite cluster.

You decided you wanna transport your metrics via AMQP so you have your RabbitMQ cluster installed and Graphite is ready to pull metrics. It's important that you feed multiple metrics with multiple values and multiple namespaces to test the integrity of your cluster's configuration and performance.

But oh no! How will I test my cluster? Well.. you could install StatsD or Diamond on all of your instances and configure them to send metrics.. and then find out that your configuration is incorrect and iterate over it 11 times.. or...

scratches head... oh wait. I know! I'll use feedr's AMQP transport!

Ok... so the config would look something like this:

```
'transports': {
    'TestRabbitCluster': {
        'type': 'AMQP',
        'host': '54.120.11.12',  # your RabbitMQ's host IP Address (or load balancer.. assuming you
        'queue': 'test_queue',
        'exchange': 'test_exchange',
        'routing_key': 'test_routing_key',
    },
}
```

Great. So you have your transport configured and now you wanna start sending some metrics. So you need to configure a formatter.

You want to send metrics from 2 availability zones, 3 types of middleware servers, each having 3 base metric types, each with average and max measurements.

So you'd do something like that...

```
'MyMetricsFormatter': {
    'type': 'Custom',
    'format': ['zone', '.', 'application', '.', 'metric', '.', 'measurement', ' ', 'value'],
    'data': {
        'zone': ['zone1', 'zone2'],
        'application': ['webserver', 'database', 'registration_server'],
        'metric': ['cpu_percentage', 'memory_percentage', 'hdd_free_percentage'],
        'measurement': ['avg', 'max'],
        'value': [i for i in xrange(100)],
    }
},
```

Note: This configuration is only logical for a hypothetical case (paradox, aye?). You might want to define multiple formatters with multiple types of metrics. For instance, the values here (1-100) are only relevant to specific types of metrics (in our case, %).

Then, all you have to do is run:

```
mouth feed -t TestRabbitCluster -f MyMetricsFormatter -c my_config_file.py -g 0.001 -m 100000000
```

The (partial) output, might look like this (if you printed it to a file):

```
zone1.database.hdd_free_percentage.max 55
zone1.registration_server.hdd_free_percentage.avg 91
zone2.database.hdd_free_percentage.avg 40
zone1.database.cpu_percentage.avg 93
zone1.webserver.memory_percentage.max 14
zone1.database.hdd_free_percentage.max 93
zone2.database.cpu_percentage.avg 10
zone1.database.memory_percentage.avg 6
zone2.database.cpu_percentage.max 4
zone2.database.memory_percentage.max 79
zone2.registration_server.memory_percentage.avg 49
zone2.webserver.cpu_percentage.avg 53
zone1.database.memory_percentage.max 68
zone2.registration_server.hdd_free_percentage.max 68
zone2.webserver.cpu_percentage.max 66
zone2.database.cpu_percentage.max 98
zone1.database.memory_percentage.max 61
zone2.database.cpu_percentage.max 24
zone1.webserver.memory_percentage.max 24
zone1.registration_server.cpu_percentage.avg 43
zone2.database.hdd_free_percentage.avg 23
zone2.registration_server.cpu_percentage.max 48
zone2.database.memory_percentage.max 78
zone1.registration_server.memory_percentage.avg 76
zone2.database.memory_percentage.avg 83
zone1.database.cpu_percentage.max 39
zone2.webserver.cpu_percentage.avg 23
zone2.database.memory_percentage.avg 41
zone2.webserver.memory_percentage.max 29
zone2.registration_server.memory_percentage.max 33
zone1.webserver.cpu_percentage.max 25
zone1.database.cpu_percentage.avg 11
zone2.webserver.cpu_percentage.avg 3
zone2.registration_server.cpu_percentage.max 24
zone2.database.cpu_percentage.max 22
zone2.database.hdd_free_percentage.avg 50
zone2.webserver.memory_percentage.max 42
zone2.webserver.hdd_free_percentage.avg 2
zone2.webserver.memory_percentage.max 83
zone2.registration_server.memory_percentage.max 59
zone2.webserver.hdd_free_percentage.avg 35
zone2.registration_server.hdd_free_percentage.avg 43
zone2.registration_server.cpu_percentage.avg 90
zone2.registration_server.cpu_percentage.max 45
zone1.database.cpu_percentage.max 34
zone1.database.hdd_free_percentage.max 90
```

Now, you would be able to, for instance, use Vagrant to load a cluster of feedr instances in AWS that would bombard

your cluster with metrics.. and then, POOF! Just “vagrant destroy” the machines when you’re done.

Of course... I would say that you should periodically run these tests (even randomly) to check that your cluster can withstand surges of metrics.. but.. i’m not your production manager. You can daemonize the process and omit the -m flag so that messages are sent constantly.

CHAPTER 8

Indices and tables

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

f

A

B

C

E

F

G

I

J

L

M

S

U