logcabin Documentation

Release 1.0.10.final.0

Barnaby Gray

December 13, 2014

Contents

logcabin is a program for aggregating and processing events from a diverse range of sources and formats, and outputting to the file system, database or a search engine.

CHAPTER 1

logcabin

logcabin is a program for aggregating and processing events from a diverse range of sources and formats, and outputting to the file system, database or a search engine.

1.1 Quickstart

Install:

```
$ pip install logcabin
```

Configure:

```
$ wget https://raw.github.com/artirix/logcabin/master/config/simple.py -O config.py
```

Run:

```
$ logcabin
```

Send some messages:

```
$ echo '{"message": "test log event"}' | nc -u localhost 6000
$ cat output.log
```

1.2 Dependencies

pip will install gevent, which needs libevent-dev (or equivalent package) to build:

```
$ apt-get install libevent-dev
```

All other dependencies are optional, and only required if you use that module.

1.2.1 zeromq

Install:

```
$ apt-get install libzmq1-dev (or equivalent)
$ pip install -U pyzmq
(>= 2.2.0.1)
```

1.2.2 mongodb

Install:

```
$ pip install pymongo
```

1.3 Docs

See: http://logcabin.readthedocs.org/en/latest/

1.4 Contributing

Contributions welcome! Please:

- Fork the project on Github
- Make your feature addition or bug fix, write tests, commit.
- Send me a pull request. Bonus points for topic branches.

1.5 Changelog

1.0.10

- Fix for breaking change in elasticsearch 1.0. fixes #4.

1.0.9

- elasticsearch output: handle bad requests better. fixes #3.

1.0.8

- Add examples to docs.

1.0.7

- Add some debugging logging

1.0.6

- Switch to milliseconds - more ISO8601 standard

1.0.5

- Save precious bytes on json output

1.0.4

- Improve error messages on unparseable Json
- Add some debug logging to Graphite

1.0.3

- add copy action in Mutate object

1.0.2

- Fix bug in when setting File statedir

1.0.1

- Add resume to input File tailing

1.0

- Add timestamp rolling to Output File
- Add url unescape filter
- Add event setter and tidy docs

0.1b12

- graphite output should reconnect on socket errors

0.1b11

- Stats filter: zero=True/False option to generate zero data points

0.1b10

- Add Python stage for custom python code

0.1b9

- Yield in input stages for better behaviour
- Improve exception catching

0.1b8

- Robustness and general bug fixes

0.1b7

- Add support to stats for wildcarding and nested fields

0.1b6

- Add index/type formatting for elasticsearch

0.1b5

- Bug fix for flow stopping at If/Switch
- Add rename and unset to Mutate

0.1b4

- Documentation improvements

0.1b3

- Add file input and zeromq output.

0.1b2

- Initial release

1.6 TODO

- New branch with gevent 1.0.1 and zmq 3
- Add Dockerfile
- Update documentation

- Add samples
- Add fail2ban filter file
- MongoDB configuration
- Add influxDB output
- Add sample configuration for rsyslog and others syslog family

1.7 Features

- Simple, but flexible configuration DSL in python.
- High concurrency: using gevent coroutines all I/O processing is parallelized.
- Wide variety of input formats: syslog, udp, zeromq.
- Wide variety of output formats: file, mongodb, elasticsearch, graphite.
- Decent performance: on a t1.micro, the smallest Amazon EC2 instance size, logcabin can burst 10,000 req/s and sustain 1,000 req/s.
- Fast startup (< 1s).
- Lightweight on memory (typically ~20MB resident).
- Easy to add / extend with python code.

1.8 Alternatives

Logging frameworks is a fairly crowded space, and indeed logcabin was inspired by, and owes a debt of gratitude to logstash, and its brilliant ideas.

So why create another logging framework?

- some frameworks only support simple topologies with a single data flow through the pipeline. This doesn't allow for sophistication when receiving data from a diverse range of sources.
- easy on the resources - logstash is built on jruby and jvm consumes >200MB resident. This is a decent chunk of memory to run locally as a lightweight log forwarder or for cheap t1.micro instances.
- fast startup is crucial for testing configurations iteratively.
- gevent + zmq = cool toys to play with :-)

CHAPTER 2

Configuration

2.1 Basics

Logcabin uses a python DSL to configure it.

To use a specific configuration pass the `-c`/`--config` option with the filename. It defaults to `config.py`.

The configuration is a set of stages, which can be any of the input, filter or output stages. The configuration is interpreted once and the constructed stages are built into the definition of a pipeline.

With the defined pipeline, logcabin will launch each instance in a greenlet, each with independent input and output queues, so no single stage blocks the processing of any other (provided it's not tying up CPU).

Example:

```
# import the stages we wish to use
from flow import Fanin, Fanout
from inputs.udp import Udp
from inputs.zeromq import Zeromq
from filters.json import Json
from filters.mutate import Mutate
from outputs.log import Log
from outputs.elasticsearch import Elasticsearch

# take input from vanilla udp or a zeromq connection
with Fanin():
    Udp(port=6000)
    Zeromq(address='tcp://*:2130')

# transform the plain text input into an structured event with the Json filter, only if field==1.
with If('field==1'):
    Json()

# set myfield=abc
Mutate(set={'myfield': 'abc'})

# broadcast this to the logcabin log and index to elasticsearch /test/event
with Fanout():
    Log()
    Elasticsearch(index='test', type='event')
```

This configures two inputs, which are both processed through the Json filter, and then output to two outputs in parallel: Log and Elasticsearch.

For full details of the inputs, filters and outputs see sections below.

2.2 Examples

Below are some example configurations.

2.2.1 Files

```
# import the inputs, filters and outputs
from inputs.file import File as IFile
from filters.regex import Regex
from outputs.file import File as OFile

# read line by line from input.log
IFile('input.log')
# extract from message format 'timestamp - message'
Regex('(?P<timestamp>.+) - (?P<message>.+)')
# and output the resulting structured event (json) to output.log
OFile('output.log')

# try me:
# DATE=$(date); echo "$DATE - message" >> input.log
```

2.2.2 Inputs

```
# import the inputs and an output
from flow import Fanin
from inputs.udp import Udp
from inputs.zeromq import Zeromq
from outputs.log import Log

# Multiple input sources can be simultaneously received. They are read in
# parallel and events 'fan in' to the rest of the pipeline.
with Fanin():
    Udp(port=6000)
    Zeromq()

# log the results to logcabin.log
Log()
```

2.2.3 Outputs

```
# import everything we're using
from flow import Fanout
from inputs.zeromq import Zeromq
from outputs.file import File
from outputs.elasticsearch import Elasticsearch
from outputs.mongodb import Mongodb

# single zeromq input
Zeromq()

# Broadcast the event in parallel to all of the following outputs. The event
# will simultaneously be written to mylogs.log, indexed to elasticsearch and
# saved to mongodb.
```

```
with Fanout():
    File(filename='mylogs.log', max_size=10, compress='gz')
    Elasticsearch(index='events', type='{program}')
    Mongodb()
```

2.2.4 Complex

```
# import everything we're using
from flow import Fanin, Switch
from inputs.udp import Udp
from inputs.zeromq import Zeromq
from filters.json import Json
from filters.stats import Stats
from outputs.graphite import Graphite
from outputs.elasticsearch import Elasticsearch
from outputs.file import File
from outputs.s3 import S3

# input from a couple of sources
with Fanin():
    Udp(port=6000)
    Zeromq()

# parse json
Json()
# generate statistic counts (suitable for graphite)
Stats(timings={'rails.{controller}.{action}.duration': 'duration'})
# write the data to a rotating log file
File(filename='mylogs.log', max_size=1000000, compress='gz')
# decide the destination because on some tags
with Switch() as case:
    # on log roll, archive the file to S3
    with case("'fileroll' in tags"):
        S3(access_key='xyz',
           secret_key='123',
           bucket='mybucket',
           path='logs/{timestamp:%Y%m%dT%H%M}')
    # write the aggregate statistics to graphite
    with case("'stat' in tags"):
        Graphite()
    # otherwise just index into elasticsearch
    with case.default:
        Elasticsearch(index='logcabin', type='event')
```

CHAPTER 3

Flows

The pipeline can fanin, and fanout and branch at any point. The following stages control flow in the pipeline.

3.1 Fanin

Fanins create many parallel inputs that will feed onto the same next stage, so multiple sources can be used for input of events (eg. udp and zeromq).

3.2 Sequence

Sequences are a series of stages, in order. The top-level of the configuration is implicitly a Sequence.

3.3 Fanout

Fanouts create many parallel outputs that run independently.

3.4 Branching

`If` and `Switch` can be used to conditionally call stages.

class `logcabin.flow.`**`Fanin`**`(`***kwargs*`)`
This merges all of the outputs of the child stages to a single queue.

Syntax:

```
with Fanin():
    Udp()
    Zeromq()
```

class `logcabin.flow.`**`Sequence`**`(`***kwargs*`)`
This connects the output of the preceding stage to the input of the next, and so on, so the event is processed by each stage one after the other, in order.

Syntax:

```
with Sequence():
    Mutate()
    Mutate()
    ...
```

class logcabin.flow.**Fanout**(***kwargs*)
This enqueues the event onto multiple input queues in parallel.

Syntax:

```
with Fanout():
    Log()
    Elasticsearch()
    Mongodb()
    ...
```

class logcabin.flow.**If**(*condition*, *on_error='reject'*)
Conditionally execute stages.

The syntax is as follows. The condition may be a lambda expression or code string:

```
with If('field==1'):
    Json()
```

class logcabin.flow.**Switch**(*on_error='reject'*)
Branch flow based on a condition.

The cases are specified using this syntax. The condition may be a lambda expression or code string:

```
with Switch() as case:
    with case(lambda ev: ev.field == 'value'):
        Json()
    with case('field2 == "value2"'):
        Mutate()
    with case.default:
        Regex(regex='abc')
```

CHAPTER 4

Inputs

4.1 file

class logcabin.inputs.file.**File**(*path*, *statedir=None*)
Tails events from a log file on disk.

Creates events with the field 'data' set to the line received.

Parameters

- **path** (*string*) – path on the file system to the log file(s), wildcards may be used to match multiple files.
- **statedir** (*string*) – writable directory to store state for files

Example:

```
File(path='/var/log/syslog')
```

4.2 udp

class logcabin.inputs.udp.**Udp**(*port*, *allow_hosts=[]*)
Receives from a udp port.

Creates events with the field 'data' set to the packet received.

Parameters

- **port** (*integer*) – listening port
- **allow_hosts** (*list*) – udp authorized clients or empty for disable

Example:

```
Udp(port=6000)

Udp(port=6000, allow_hosts=['1.1.1.1', '192.168.1.1'])
```

4.3 zeromq

class logcabin.inputs.zeromq.**Zeromq**(*address='tcp://*:2120'*, *mode='bind'*, *socket='PULL'*)

Receives from a zeromq socket.

Creates events with the field 'data' set to the packet received.

Parameters

- **address** (*string*) – zeromq address to bind on (default: *tcp://*:2120*)
- **mode** (*string*) – connect or bind (default: bind)
- **socket** (*string*) – PULL or SUB (default: PULL)

Example:

```
Zeromq(address='tcp://*:2121', mode='bind', socker='PULL')
```

CHAPTER 5

Filters

5.1 json

class logcabin.filters.json.**Json**(*field='data'*, *consume=True*, *on_error='reject'*)
Parse a json encoded field.

Parameters

- **field** (*string*) – the field containing the json (default: data)
- **consume** (*boolean*) – whether to remove the field after decoding (default: true)

Example:

```
Json()
```

5.2 syslog

class logcabin.filters.syslog.**Syslog**(*field='data'*, *consume=True*, *on_error='reject'*)
Parse a syslog encoded field.

This sets the fields:

- timestamp
- facility
- severity
- host
- program
- pid
- message

Parameters

- **field** (*string*) – the field containing the syslog message (default: data)
- **consume** (*boolean*) – whether to remove the field after decoding (default: true)

Example:

```
Syslog()
```

5.3 mutate

class logcabin.filters.mutate.**Mutate**(*set={}*, *rename={}*, *copy={}*, *unset=[]*)
Filter that allows you to add, rename, copy and drop fields

Parameters

- **set** (*map*) – fields to set (optional). The values if strings may format other fields from the event.
- **rename** (*map*) – fields to rename (a: b renames b to a) (optional)
- **unset** (*list*) – fields to unset (optional)

Example:

```
Mutate(set={'fullname': '{first} {last}'})
```

Renaming:

```
Mutate(rename={'@timestamp': 'timestamp', '@message': 'message'})
```

Unsetting:

```
Mutate(unset=['junk', 'rubbish'])
```

5.4 python

class logcabin.filters.python.**Python**(*function*, *on_error='reject'*)
Call out to a python function for adding custom functionality.

Parameters **function** (*callable*) – callable taking the event as an argument

Example:

```
Python(function=lambda ev: ev.count = int(ev.count))
```

Alternatively, a function can be passed, for more complex functionality:

```
def clean(ev):
    ev.header = ev.header.strip()
    ev.message = ev.message.strip()

Python(function=clean)
```

5.5 regex

class logcabin.filters.regex.**Regex**(*regex*, *field='data'*, *on_error='reject'*)
Parse a field with a regular expression. The regex named groups `(?P<name>...)` will be create event fields (overwriting any existing).

If you extract a 'timestamp' field, this will automatically be parsed as a datetime and used as the event timestamp (instead of the default of the time received).

Parameters

- **regex** (*string*) – the regular expression
- **field** (*string*) – the field to run the regex on (default: data)

Example:

```
Regex(regex='(?P<timestamp>.+) - (?P<message>.+)')
```

5.6 stats

class `logcabin.filters.stats.`**`Stats`**(*period=5*, *metrics=None*, *zero=True*)

Filter that produces aggregate statistics.

It will produce:

•name.count: number of data points

•name.rate: the data points per second

•name.mean: mean of data points

•name.min: minimum data point

•name.median: median data point

•name.upper95: 95th% data point

•name.upper99: 99th% data point

•name.max: maximum data point

•name.stddev: standard deviation

This is emitted as a single event, every period.

Parameters

- **period** (*integer*) – period to report stats, in seconds
- **metrics** (*map*) – field names => values. Any fields from the events can be formatting into the field names. Values can be an event field, nested path to a field (separated by .) and can contain wildcard '*', to indicate generating statistics from any numerical fields.
- **zero** (*boolean*) – output zero for previously seen metrics (useful to disambiguate no activity and output broken)

Example:

```
Stats(metrics={'rails.{controller}.{action}.duration': 'duration'})
```

Wildcards can be used to pull out nested structures:

```
Stats(metrics={'app.{1}': 'timings.*'})
```

CHAPTER 6

Outputs

6.1 elasticsearch

class logcabin.outputs.elasticsearch.**Elasticsearch**(*index*, *type*, *host='localhost'*, *port=9200*)

Outputs to an elasticsearch index.

Parameters

- **host** (*string*) – elasticsearch host
- **port** (*integer*) – elasticsearch port
- **index** (*string*) – (required) elasticsearch index. This can be formatted by fields in the event.
- **type** (*string*) – (required) elasticsearch type. This can be formatted by fields in the event.

Example configuration for kibana:

```
Mutate(rename={'@timestamp': 'timestamp', '@message': 'message'})
Elasticsearch(index='logstash-{@timestamp:%Y.%m.%d}', type='event')
```

6.2 file

class logcabin.outputs.file.**File**(*filename*, *max_size=None*, *max_count=10*, *compress=None*)

Log to file.

The file format is a line per event as json.

When a log is rolled, a 'virtual' event will be generated with the tag 'fileroll', and the field 'filename' which can be used by further outputs to process a log file when it rolls (eg. batch upload to S3). This event contains a 'trigger' field containing the original event that caused the log roll.

Parameters

- **filename** (*string*) – the log filename (required). You can use event format values in this (eg. 'output-{program}.log')
- **max_size** (*integer*) – maximum size of file before rolling to .1, .2, etc.
- **max_count** (*integer*) – maximum number of rolled files (default: 10)
- **compress** (*string*) – set to 'gz' to compress the file after rolling.

Example:

```
File(filename='mylogs.log', max_size=10, compress='gz')
```

6.3 graphite

class logcabin.outputs.graphite.**Graphite**(*host='localhost'*, *port=2004*)
Upload stats data to a graphite server.

Parameters

- **host** (*string*) – graphite server hostname
- **port** (*string*) – graphite server port

Example:

```
Graphite(host='graphite')
```

6.4 log

class logcabin.outputs.log.**Log**(*message=''*)
Logging output.

Parameters **message** (*string*) – message to log (optional)

Example:

```
Log(message="event:")
```

6.5 mongodb

class logcabin.outputs.mongodb.**Mongodb**(*host='localhost'*, *port=27017*, *database='test'*, *collection='events'*)
Outputs to a mongodb collection.

Parameters

- **host** (*string*) – mongodb host
- **port** (*integer*) – mongodb port
- **database** (*string*) – mongodb database
- **collection** (*string*) – mongodb collection

Example:

```
Mongodb(host="mongodb", database="logs")
```

6.6 perf

class logcabin.outputs.perf.**Perf**(*period=60*)
Simple performance counter output.

Parameters **period** (*integer*) – interval between reports in seconds

Example:

```
Perf(period=5)
```

6.7 s3

class logcabin.outputs.s3.**S3**(*access_key*, *secret_key*, *bucket*, *path*)
Uploads to an S3 bucket.

This should follow the File output, in order to upload the rolled log files to S3:

```
File(filename='log/batch.log', )
If(lambda ev: 'fileroll' in ev.tags):
    S3(access_key='...', secret_key='...',
       bucket='x', path='logs/{timestamp:%Y-%m-%d/%H%M%S}.log')
```

Bucket or path may by formatted by event, eg. to upload to a timestamped path: path='{timestamp:%Y-%m-%d/%H%M%S}.log'

Parameters

- **access_key** (*string*) – Amazon S3 access key
- **secret_key** (*string*) – Amazon S3 secret key
- **bucket** (*string*) – the bucket name
- **path** (*string*) – the path

6.8 zeromq

class logcabin.outputs.zeromq.**Zeromq**(*address='tcp://127.0.0.1:2120'*, *mode='connect'*, *socket='PUSH'*)
Outputs on a zeromq socket.

Parameters

- **address** (*string*) – zeromq address (default: *tcp://*:2120*)
- **mode** (*string*) – connect or bind (default: connect)
- **socket** (*string*) – PUSH or PUB (default: PUSH)

Example:

```
Zeromq(address="tcp://relay:2120", mode="connect", socket="PUSH")
```

CHAPTER 7

Index

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

l

E

F

G

I

J

L

M

P

R

S

U

Z