zerolog Documentation

Release 0.1

theghouls

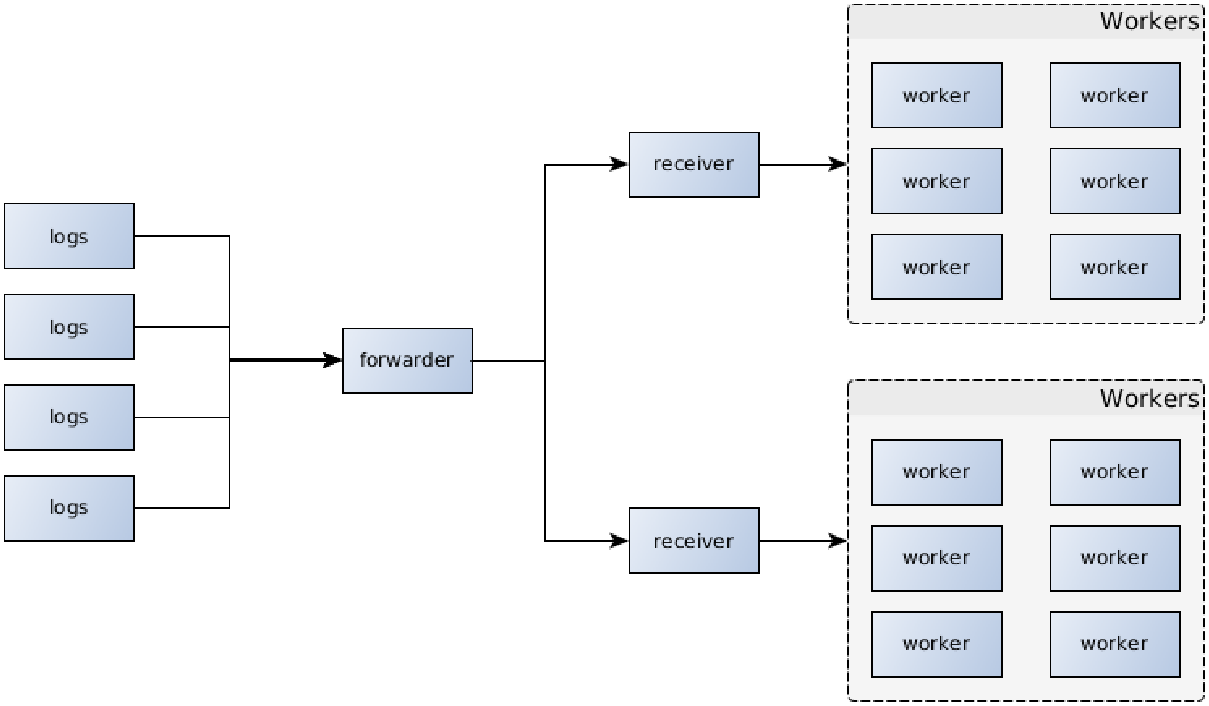
Jul 06, 2017

Contents

Zerolog is a very simple python library designed to help you capture logs in realtime from multiples sources sending log over TCP (rsyslog, syslog-ng, etc.) and process them as you want for your needs

It will remain pretty simple and only provide you basics tools to capture log and process them for your needs, plus CLI tool to start all elements. Those elements are :

- A forwarder, who listen to incoming log and publish them
- A receiver, receiving messages from forwarder, and pass them to workers
- A worker, that process messages for you need



CHAPTER 1

Installation

You can install the project from source

```
pip install -r requirements.txt
python setup.py install
```

Contents:

Running elements

Zerolog provide command line tool to start every element. This section will explain how to start each element of zerolog

Running forwarder

The forwarder is in charge of receiving logs from external services and to forward them to receivers. Thanks to zeromq PUB/SUB pattern, you can also use topic to filter messages. In this way you can associate receivers with specifics logs messages.

Like for all elements, zerolog provide a command to start a forwarder :

```
usage: zerolog forwarder [-h] [-f FRONTEND] [-b BACKEND] [-m MONITOR]

optional arguments:
    -h, --help            show this help message and exit
    -f FRONTEND, --frontend FRONTEND
                          frontend port for sending data to receivers
    -b BACKEND, --backend BACKEND
                          backend port for receiving data
    -m MONITOR, --monitor MONITOR
                          monitor port for publishing data
```

Forwarder's parameters are all optional and will use default values if not provided, so you can run a forwarder using only :

```
zerolog forwarder
```

Only monitor socket will not be set if the `-m` option is not provided. The monitor socket only role is to output messages received by the forwarder, so you can easily check if logs messages are able to hit the forwarder and are correctly sending back messages to receivers.

Running receiver

The receiver is in charge to filter messages incomming from forwarder and dispatch them to a pool of workers. By default, receiver will not filter on any topic and will receive all messages from forwarder.

Command usage :

```
usage: zerolog receiver [-h] [-t TOPIC] [-l LOG_CONFIG]
                     [--output-port OUTPUT_PORT | --output-socket OUTPUT_SOCKET]
                     forwarder_address forwarder_port

positional arguments:
forwarder_address     address of running forwarder
forwarder_port        port of running forwarder

optional arguments:
  -h, --help          show this help message and exit
  -t TOPIC, --topic TOPIC
                      optional topic to subscribe
  -l LOG_CONFIG, --log-config LOG_CONFIG
                      location of logging configuration file (python
                      standard logging configuration file)
  --output-port OUTPUT_PORT
                      output TCP port for workers (default to 6200)
  --output-socket OUTPUT_SOCKET
                      output unix socket for workers
```

`forwarder_address` and `forwarder_port` are both mandatory. The first one need to be a valid address for connecting to your forwarder (e.g: `"127.0.0.1"` and the second one is the `frontend` port of your forwarder.

`topic` parameter is optional and default is to accept any message from forwarder. But you can configure your can use specific topic for filtering messages, for example if you only want nginx logs for this receiver. This allow you to use only one forwarder for all messages and all processes.

`--log-config` parameter is pretty obvious here. This one can be usefull for sending an email, or logging to sentry (for example) if the receiver crash.

`output-port` and `output-socket` parameters will be used to bind the ventilator socket (the one in charge of sending messages to workers). You can't set them both, but you can choose between unix socket (if workers are on the same machine for example) or classic TCP port

Running workers

The worker is in charge of processing logs messages and apply your own logic on them.

Command :

```
usage: zerolog worker [-h] [-d DIRECTORY] backend worker_class

positional arguments:
    backend               backend to connect for receiving message (e.g:
                          tcp://127.0.0.1:6200)
    worker_class          class to use as worker

optional arguments:
    -h, --help            show this help message and exit
    -d DIRECTORY, --directory DIRECTORY
                          directory to append to sys.path for import (optional)
```

As you can see, this command aim to allow you to start your own workers. Only restriction is that your workers must take a `backend` parameter in `__init__` method, and have a `run` method.

For simplicity you can simply extends the `BaseWorker` class avaible in zerolog.

`backend` parameter here is a full zeromq connection string, allowing you to connect to any backend (TCP or unix socket)

`worker_class` represent a full import path to the wanted worker class (e.g: `zerolog.worker.BaseWorker`.

`--directory` parameter allow you to append a specific directory to python path. With this you can use a `worker_class` from anywhere and without installing anything, a simple python file could be used to declare your worker

Logging error in receiver

Receiver start command can take a logging file in parameter. If provided this file will be used to configure the logger of the module.

For example, here is a simple log file :

```
[loggers]
keys=root,simpleExample

[handlers]
keys=consoleHandler

[formatters]
keys=simpleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_simpleExample]
level=DEBUG
handlers=consoleHandler
qualname=simpleExample
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=simpleFormatter
args=(sys.stdout,)
```

```
[formatter_simpleFormatter]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
datefmt=
```

As you can see this is standard python logging configuration file.

Receiver is configured to log an error in case of exception and keyboard interruption. Using a custom logger can allow you to have quick informations in case of failure and will be very helpful in case of receiver crash.

Creating your own workers

One of the main goal of zerolog is to provide all needed elements to start receive and process logs easily. Principales parts of the chain are built-in and can be used "as-is" but one must be implemented: the worker.

The worker is the core of your process, it's in charge of processing logs to fit your needs.

To help you creating your worker, zerolog provide a `BaseWorker` class that can be inherited. This class provide anything needed to start (binding to receiver, run loop). All you need to do to create your own worker is to implement `process_data` method in your worker class. Let's take a simple example :

```python
import sqlite3
from zerolog.worker import BaseWorker


class MyWorker(BaseWorker):

    def __init__(self, backend, *args, **kwargs):
        super(MyWorker, self).__init__(backend, *args, **kwargs)
        self.connection = sqlite3.connect("/tmp/example.db")
        self.cursor = self.connection.cursor()

    def process_data(self, data):
        print(data)
        self.cursor.execute('''
        INSERT INTO logs (message) values (?)''', [data])
        self.connection.commit()
```

This example is very simple, it will only print data on stdout and save received message in sqlite database. But this example give you a good start for creating your own worker.

Now let's start this worker. Let's assume that your `worker.py` file is located in `/home/user/` directory.

```
zerolog worker "tcp://127.0.0.1:6001" worker.MyWorker -d /home/user/
```

And that's it ! Your worker is running, ready to receive messages and process them

Using supervisor

All your elements are ready to be used and can be start using CLI, but it's not realy suitable for production.

For starting and managing your processes you can use for example supervisor. Here is a simple configuration file :

```
[program:zerolog-forwarder]
command=zerolog forwarder
autostart=true
autorestart=true

[program:zerolog-receiver]
command=zerolog receiver 127.0.0.1 6001
autostart=true
autorestart=true

[program:zerolog-worker]
command=zerolog worker "tcp://127.0.0.1:6200" worker.MyWorker -d /home/user
numprocs=8
process_name=worker_%(process_num)s
```

With this file, all your work chain is ready to get messages and process them

Using zerolog with logstash

Since logstash got a zeromq output, you can simply integrate it with zerolog.

Replacing zerolog forwarder with logstash

Logstash can act like the forwarder, all you need to do is to configure your logstash to output logs to a zeromq pub socket, for example :

```
input {
  tcp {
    port => 6000
  }
 }
 output {
   zeromq {
     address => ["tcp://*:6001"]
     mode => "server"
     topology => "pubsub"
     topic => "apache-logs"
   }
 }
```

Note that in pubsub mode you can also specify a topic to route logs to correct receivers.

Replacing zerolog receiver with logstash

Maybe you don't need any forwarder at all and you only want a simple receiver to workers pattern. Logstash can also replaces receiver or sends logs directly to workers, for example :

```
input {
  tcp {
    port => 6000
  }
 }
 output {
    zeromq {
      address => ["tcp://*:6200"]
      mode => "server"
      topology => "pushpull"
    }
  }
```

With this configuration file, all workers will receive tasks directly from logstash, avoiding the use of receiver

Commands reference

forwarder

```
usage: zerolog forwarder [-h] [-f FRONTEND] [-b BACKEND] [-m MONITOR]

optional arguments:
    -h, --help            show this help message and exit
    -f FRONTEND, --frontend FRONTEND
                          frontend port for sending data to receivers
    -b BACKEND, --backend BACKEND
                          backend port for receiving data
    -m MONITOR, --monitor MONITOR
                          monitor port for publishing data
```

receiver

```
usage: zerolog receiver [-h] [-t TOPIC] [-l LOG_CONFIG]
                        [--output-port OUTPUT_PORT | --output-socket OUTPUT_SOCKET]
                        forwarder_address forwarder_port

positional arguments:
forwarder_address      address of running forwarder
forwarder_port         port of running forwarder

optional arguments:
  -h, --help           show this help message and exit
  -t TOPIC, --topic TOPIC
                       optional topic to subscribe
  -l LOG_CONFIG, --log-config LOG_CONFIG
                       location of logging configuration file (python
                       standard logging configuration file)
  --output-port OUTPUT_PORT
                       output TCP port for workers (default to 6200)
  --output-socket OUTPUT_SOCKET
                       output unix socket for workers
```

worker

```
usage: zerolog worker [-h] [-d DIRECTORY] backend worker_class

positional arguments:
    backend               backend to connect for receiving message (e.g:
                          tcp://127.0.0.1:6200)
    worker_class          class to use as worker

optional arguments:
    -h, --help            show this help message and exit
    -d DIRECTORY, --directory DIRECTORY
                          directory to append to sys.path for import (optional)
```

Zerolog API

Forwarder

zerolog.forwarder.**start_forwarder**(*pub_port*, *receive_port*, *mon_port=None*, *backend_socket=None*, *frontend_socket=None*)

Start a zeromq proxy for forwarding messages from TCP socket to zmq PUB socket

Parameters

- **pub_port** (*int*) – port number to use for publishing messages to workers
- **receive_port** (*int*) – port number to use for receiving messages
- **mon_port (optional)** (*int*) – port to use for monitor socket
- **backend_socket (optionnal)** (*str*) – socket type to use for backend socket
- **frontend_socket (optionnal)** (*str*) – socket type to use for frontend socket

Receiver

Reciver module

Contain default receiver. Receiver will be in charge to get data from forwarder

class zerolog.receiver.**Receiver**(*forwarder_address*, *forwarder_port*, *topic=None*, **args*, ***kwargs*)

Bases: object

Main receiver class.

You can use different patterns for receiver and workers :

- Workers connect to receiver over TCP
- Workers connect to receiver over ipc protocol (same machine)

For more flexibility this class allow you to pass an output_port parameter or a output_socket, the last one allowing you to use ipc protocol between workers and receiver.

Warning: If both output_port and output_socket are set, an error will be raised. If none are set, default is to use TCP bind on port 6200

Goal of this class

This class is very basic since there is no specific action here, only data data forwarding from forwarder to workers. Yet it's configurable enough to cover major use cases, and can be easily override.

But keep in mind that for large application this class will process a very large amount of data. Keep it simple and put logic in workers, there here for that prupose

---

Parameters

- **forwarder_address** (*str*) – IP or domain of forwarder
- **forwarder_port** (*int*) – port to listen for incomming forwarder messages
- **topic** (*str*) – publish topic to listen (default to all)
- **output_port** (*int*) – port to bind for sending data to workers
- **output_socket** (*str*) – location of ipc socket to use for sending data to workers
- **logging_config** (*str*) – file path to logging configuration

Raise TypeError

recv_data()

Receive data from forwarder and return them. Channel will not be return

Returns data from forwarder

Return type bytes

run()

Main receiver loop

setup_output_socket(*output_port=None*, *output_socket=None*)

Setup PUSH output socket

Worker

Base worker implmentation

class `zerolog.worker.`**BaseWorker**(*backend*, **args*, ***kwargs*)

Bases: `object`

Base worker class.

This class cannot be used "as is", it will raises an `ImplementationError` in `process_data` methode. `BaseWorker` provide only a skeleton to help you to develop your own workers

Note: For conveniance, default recv method for backend socket is `recv_string()` from pyzmq. But you can change it easily by overloading `recv_func` in your worker, for example:

```
def __init__(self, backend, *args, **kwargs):
    super(MyWorkerClase, self).__init__(backend, *args, **kwargs)
    self.recv_func = self.backend.recv_json()
```

Parameters **backend** (*str*) – backend zeromq string to connect to receiver. e.g: `ipc://unix.socket`

`process_data`(*data*)

Process data

Parameters **`data`** (`mixed`) – data received from backend

Raises NotImplementedError

`run`()

Main loop for receive messages and process them

`self.recv_func` is used to receive data from backend

Python Module Index

Index