Logging

Vasken Dermardiros

March 25, 2022

Problem

Debugging be like...

```
print("Here")
df = fetch_some_data(building = 5)
print("Got me some data!")
print(df.head())
```

Problem (2)

Works now!

```
# print("Here")
df = fetch_some_data(building = 5)
# print("Got me some data!")
# print(df.head())
```

Problem (3)

Jeez… it's not working anymore

```
print("Here")
df = fetch_some_data(building = 5)
print("Got me some data!")
print(df.head())
print("Data was for: ")
print(building)
```

Problem (4)

Well, not only is this quite poor practice, there are also features you probably didn't know existed. `f-strings` are your friends!

```python
pew = 5.123
print(f"{pew}")
print(f"The value is: {pew}")
print(f"{pew:.1f}")
print(f"{pew = }")
print(f"{pew + 1 = }")
```

```
5.123
The value is: 5.123
5.1
pew = 5.123
pew + 1 = 6.123
```

Problem (5)

Wrapping up on debugging, best to:

1. Put a bunch of prints and locate the bug
2. Commit your changes
3. Apply the fix to the bug
4. Commit your changes
5. `git rebase -i` and drop the commit with the prints
6. Result: clean fix and no need to go back and delete those prints!

Solution

Control the verbosity of your code dynamically using `logging`! Useful for debugging, troubleshooting, and able to reduce output when not needed to save on space, I/O and so on.

```
1 import logging
2 logging.debug(f"Debug level: ...")
3 logging.info(f"Info level: Eh.")
4 logging.warning(f"Warning level: Oh?")
5 logging.error(f"Error level: OH!")
6 logging.critical(f"Critical level: OH SHIT!")
```

By default, only warning and worst will print to console.

```
1 WARNING:root:Warning level: Oh?
2 ERROR:root:Error level: OH!
3 CRITICAL:root:Critical level: OH SHIT!
```

Agenda

- Demo why `print()` sucks and you should only use it in a jupyter notebook
- Go over the levels of logging in the Python `logging` standard library
- What's a useful print?
- Formating the output to include a timestamp and code location
- `logging_convenience` snippet
- Timer decorators
- Log aggregators and graylog

What I won't be covering: non-Python environments, things I know I don't know and things I don't know I don't know.

Why do we log?

Logging serves two purposes[1].->https://docs.python-guide.org/writing/logging/:

- **Diagnostic logging** records events related to the application's operation. If a user calls in to report an error, for example, the logs can be searched for context.
- **Audit logging** records events for business analysis. A user's transactions can be extracted and combined with other user details for reports or to optimize a business goal.

At this stage, we're much more concerned by the former. The latter can be enabled when we start aggregating our logs in a central location.

Levels of Logging

The `logging` package contains 5 main levels for logging[2].->https://www.loggly.com/use-cases/6-python-logging-best-practices-you-should-be-aware-of/:

- DEBUG: You should use this level for debugging purposes in development.
- INFO: You should use this level when something interesting — but expected — happens (e.g., a user starts a new project in a project management application).

Levels of Logging (2)

- WARNING: You should use this level when something unexpected or unusual happens. It's not an error, but you should pay attention to it.
- ERROR: This level is for things that go wrong but are usually recoverable (e.g., internal exceptions you can handle or APIs returning error results).
- CRITICAL: You should use this level in a doomsday scenario. The application is unusable. At this level, someone should be woken up at 2 a.m.

In deployment, we should only log `warning` and up, and even there, `warning` should not be retained for too long.

What's a useful print?

1. Something broke.
2. Data not yet available. Try 2 of 3. Sleeping for 20 seconds.
3. 9
4.

What's a useful print?

1. Something broke.
2. Data not yet available. Try 2 of 3. Sleeping for 20 seconds.
3. 9
4.

By reading the print, does it force you towards action? Or does it force you to open up VSCode and start messaging the developer of that code? Or taking screenshots and submitting a ticket?

Are you the developer or the operator? Are you helping or blocking?

Logging output: stdout

```
1  import logging
2
3  logging.basicConfig(level=logging.INFO)
4
5  logging.info('This message will be logged')
6  logging.debug('This message will not be logged
       ')
```

```
1  INFO:root:This message will be logged
```

Logging output: to file

```
1  import logging
2
3  logging.basicConfig(
4      filename='myfirstlog.log',
5      level=logging.DEBUG,
6      format='%(asctime)s | %(name)s | %(
           levelname)s | %(message)s'
7  )
8
9  logging.info('This message will be logged')
10 logging.debug('This message will now be logged
       ')
```

```
1  2022-03-22 17:13:06,999 | root | INFO | This
       message will be logged
2  2022-03-22 17:13:06,999 | root | DEBUG | This
       message will now be logged
```

Logging output: handlers

Sticking still with the `basicConfig` module, we can attach multiple handlers:

- **StreamHandler**: streaming, typically to `sys.stdout`
- **FileHandler**: writing to file
- **RotatingFileHandler**: writing to a rotating file; e.g. new file every day

Logging output: handlers (2)

```
handlers = [
    logging.StreamHandler(sys.stdout),
    logging.FileHandler(filename=log_filename,
        mode="w")
]
logging.basicConfig(
    level=logging.INFO,
    format="[%(asctime)s][%(name)s][%(
       levelname)s][%(filename)s:%(lineno)d] -
        %(message)s",
    handlers=handlers,
)
```

Logging output: information

```
format = "[%(asctime)s][%(name)s][%(levelname)s][%(
filename)s:%(lineno)d] - %(message)s"
```

- asctime: ISO-8601 timestamp
- name: username
- levelname: debug, info, warning, error, critical
- filename: where log statement is, not `main()`
- lineno: line number of log statement
- message: print

```
[2022-03-22 17:29:15,543][root][WARNING][delete_me.
py:29] - Random text
```

About timestamps

We are more and more a global team. Well, our buildings are very global! Is local time Montreal time? Local to the user? Local to the building?

We'll have to use a standard. What the internets suggests is to use ISO-8601 timestamps[3].->https://en.wikipedia.org/wiki/ISO_8601, e.g. `2022-03-25T09:00-06:00`. This way, the timezone offset is included as well.

```
logging.basicConfig(
    format="%(asctime)s %(message)s",
    datefmt="%Y-%m-%dT%H:%M:%S%z"
)
```

`logging_convenience` function

Function[4].->https://git.brainboxai.net/Toolkit/KitUtils/ goes in your `main.py` and makes it very convenient to output the logs to the terminal and to a local file.

```
1 import logging
2
3 from KitUtils.logging_convenience import
      logging_standard
4 logging_standard(level='debug', stream=True,
      log_filename=None)
5
6 logging.warning("Random text")
```

It's based on `basicConfig`. Could potentially be extended to rely on an external *.ini config file using `logging.config.fileConfig()` [5].->https://coralogix.com/blog/python-logging-best-practices-tips/ as well and to be pushing to an aggregator.

Timing things: decorators

```
1  def timer_logging(func):
2      """Print the runtime of the decorated
           function"""
3      import functools
4      import time
5      @functools.wraps(func)
6      def wrapper_timer(*args, **kwargs):
7          start_time = time.perf_counter()
8          value = func(*args, **kwargs)
9          end_time = time.perf_counter()
10         run_time = end_time - start_time
11         logging.info(f"Finished {func.__name__
               !r} in {run_time:.4f} secs")
12         return value
13     return wrapper_timer
```

Timing things: decorators (2)

```
1  @timer_logging
2  def getting_so_sleepy():
3      time.sleep(10)
4      return
```

```
[2022-03-22 17:43:36,941][root][INFO][sleep.py:39]
- Finished 'getting_so_sleepy'in 10.0051 secs
```

Log aggregators

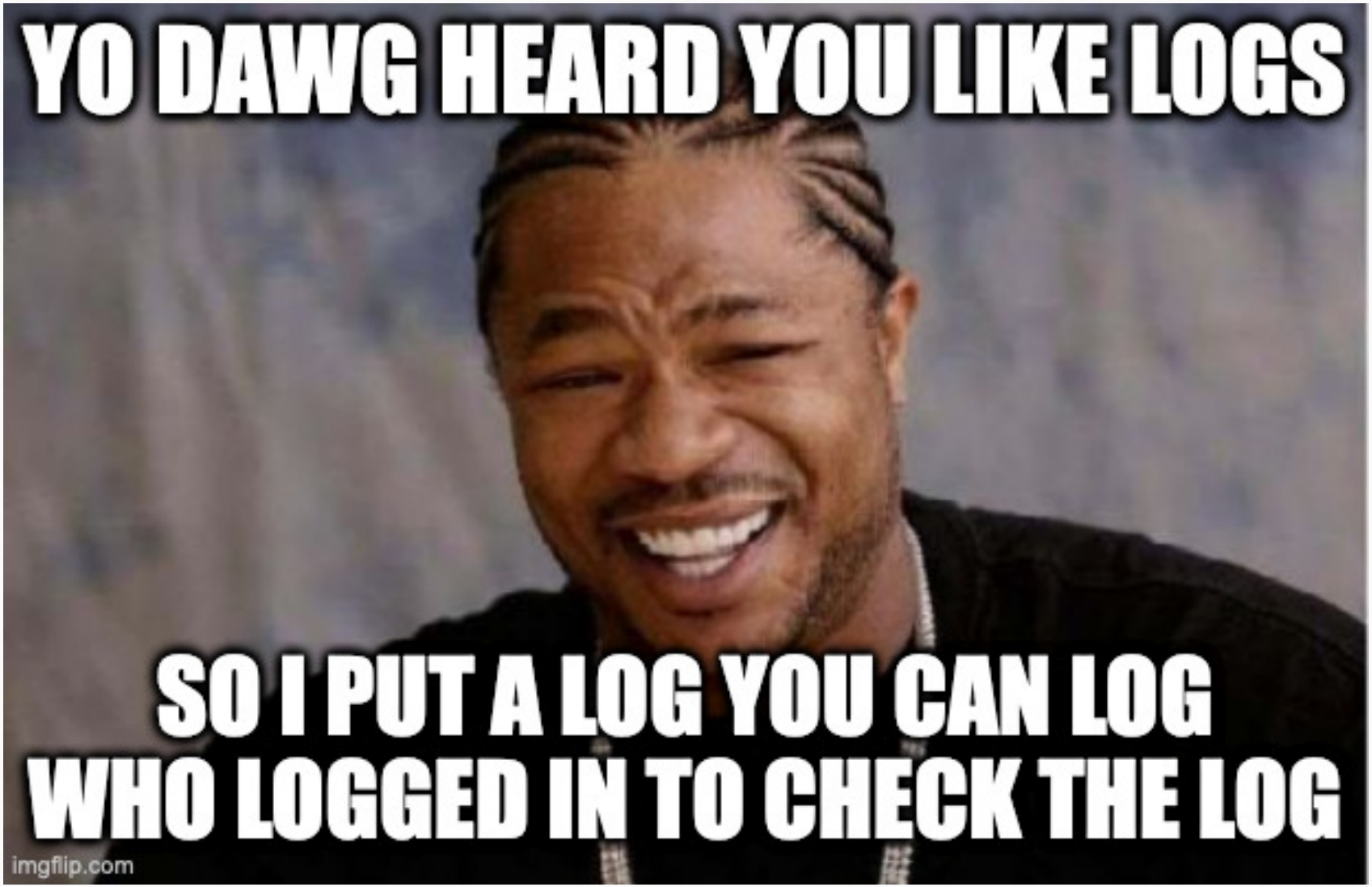
- loggly: https://www.loggly.com
- fluentd: https://www.fluentd.org/
- graylog: https://www.graylog.org/
- …and many many more!

Graylog: what's it for?

> Graylog provides answers to your team's security, application, and IT infrastructure questions by enabling you to combine, enrich, correlate, query, and visualize all your log data in one place.

- Log collection
- Sort / filter / parse
- Log analysis
- Alerts and triggers
- Correlations: e.g. viewing 4G modem, MQTT broker, DB logs together to uncover problems
- Archiving
- Scheduled reports
- User audit logs: logs who checked which log

Graylog: user audit logs



Graylog: dashboard

Install locally to test: https://hub.docker.com/r/graylog/graylog/

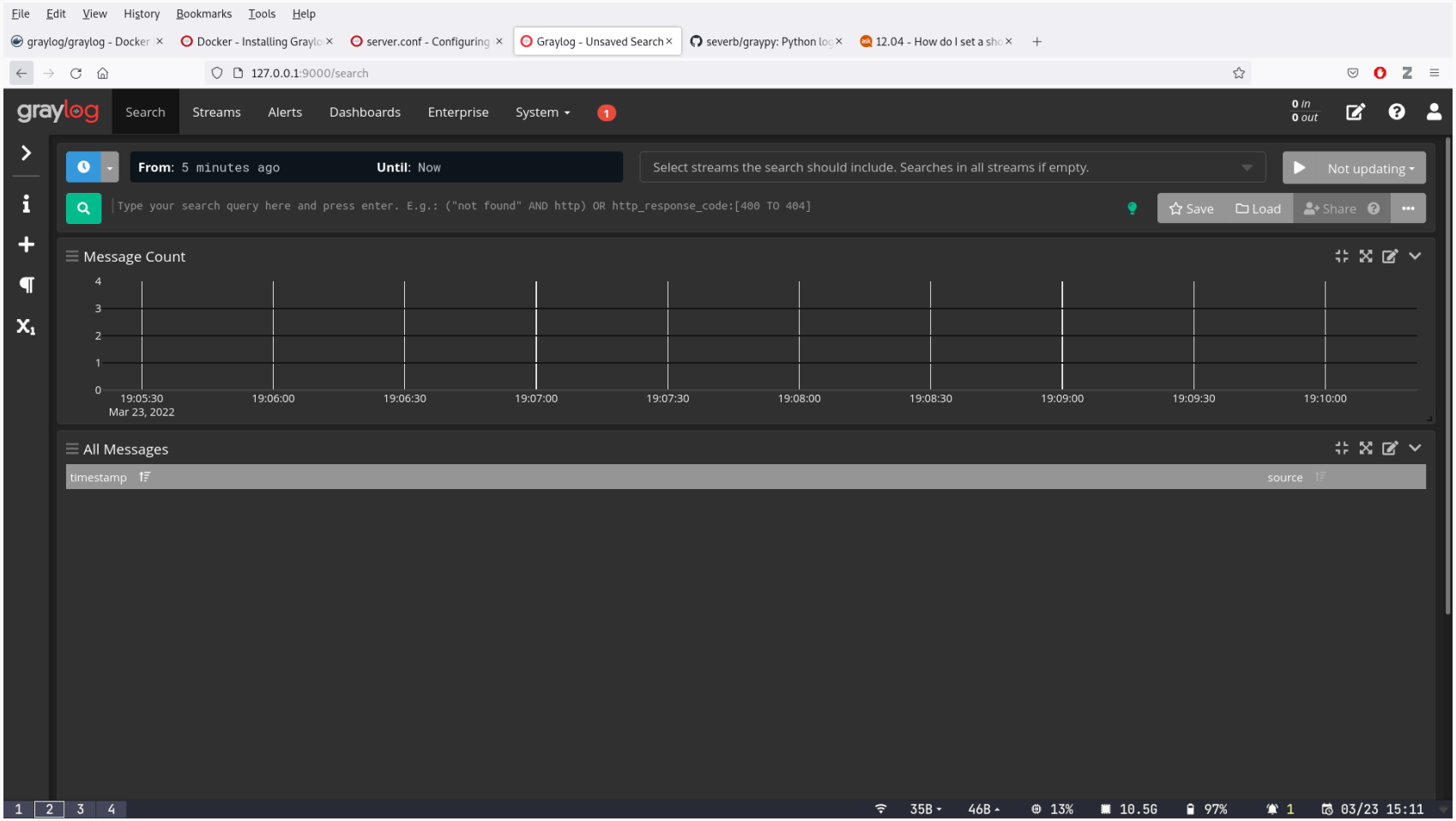


Figure 1: Graylog dashboard

Python `graypy` package

Python `graylog` package to add handlers to `logging` library:
https://github.com/severb/graypy

- GELFUDPHandler - UDP log forwarding
- GELFTCPHandler - TCP log forwarding
- GELFTLSHandler - TCP log forwarding with TLS support
- GELFHTTPHandler - HTTP log forwarding
- GELFRabbitHandler - RabbitMQ log forwarding

GELF: graylog extended logging format

Python graypy package (2)

```
1  import logging
2  import graypy
3
4  my_logger = logging.getLogger('test_logger')
5  my_logger.setLevel(logging.DEBUG)
6
7  handler = graypy.GELFUDPHandler(
8      '127.0.0.1', 12201
9  )
10 my_logger.addHandler(handler)
11
12 my_logger.debug("Coo coo! C'est moi!")
```

Python graypy package (3)

After adding a GELF UDP input and configuring the stream in graylog…

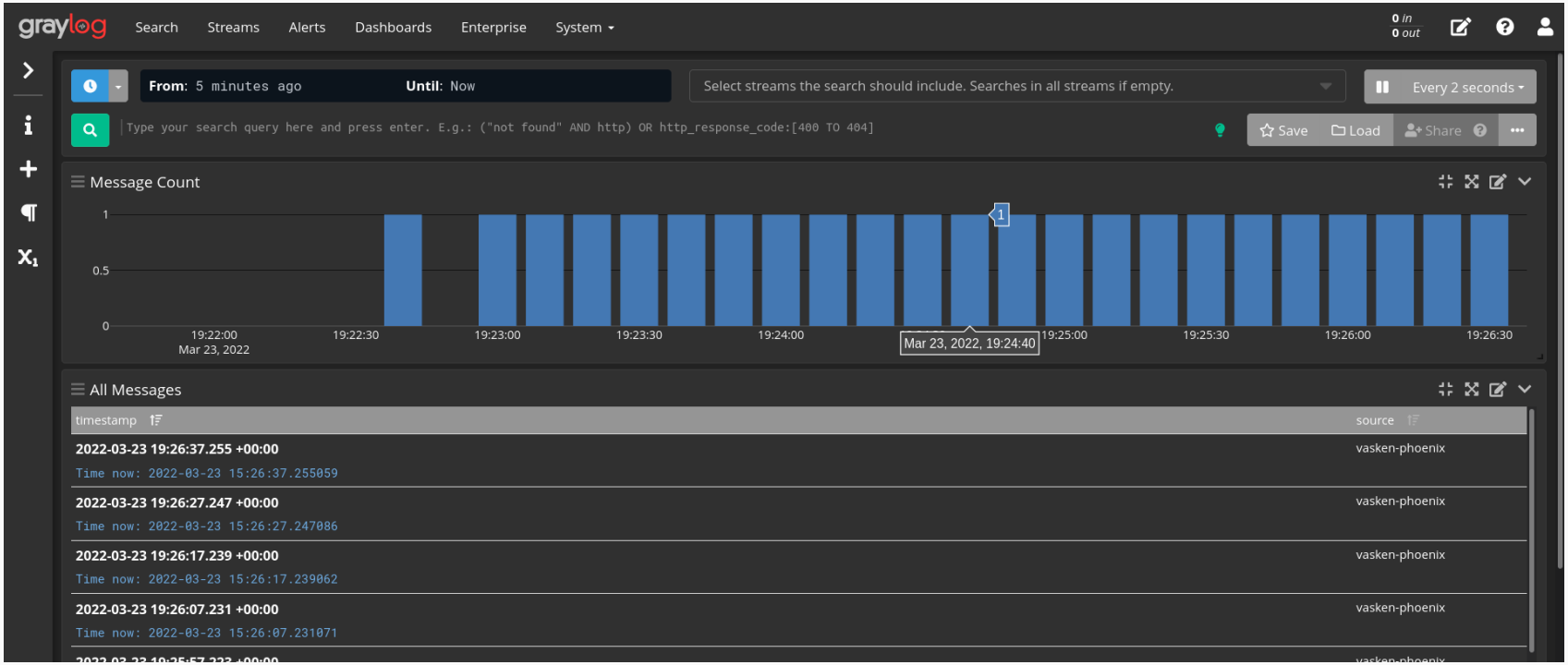


Figure 2: We are logging!

Graylog: message

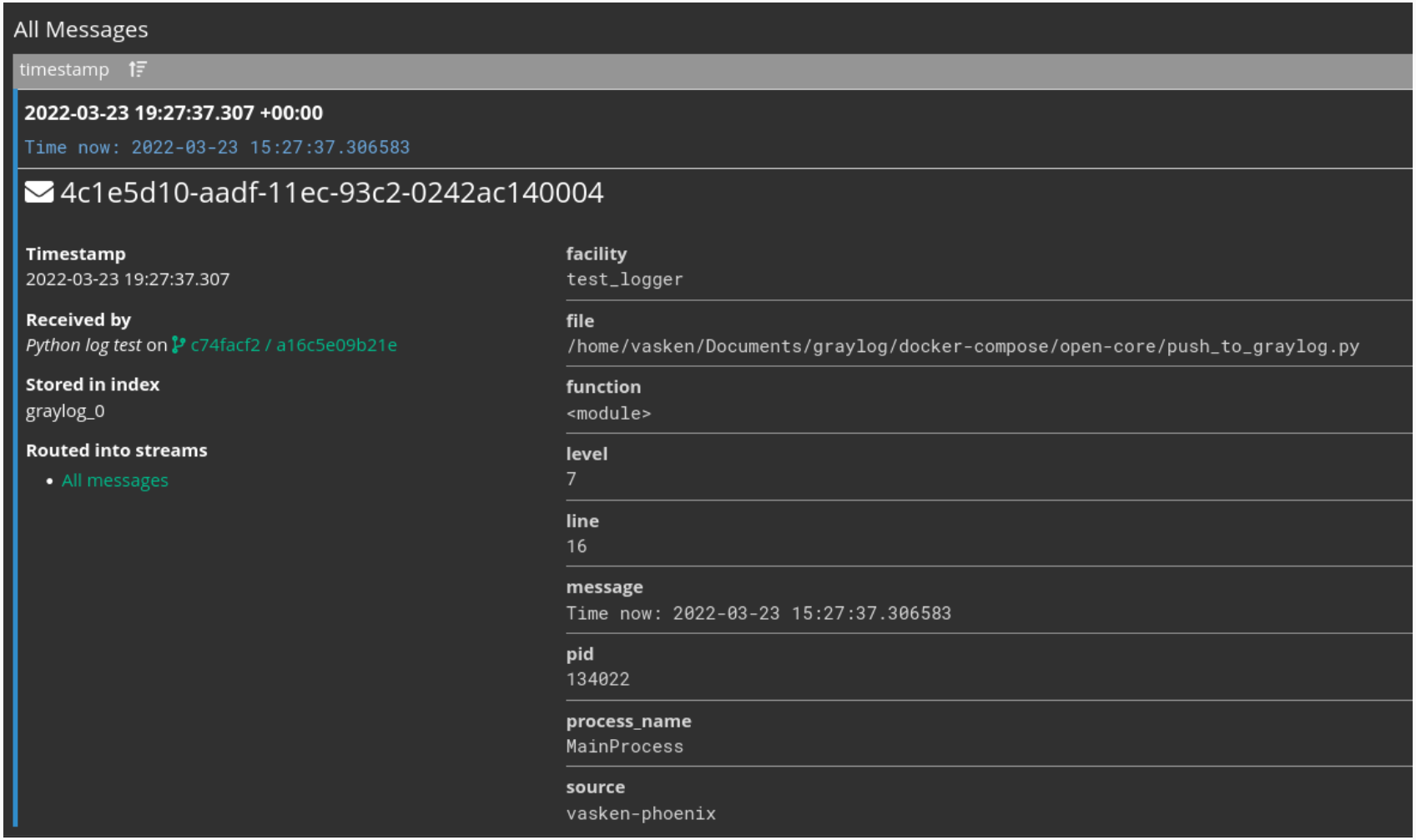
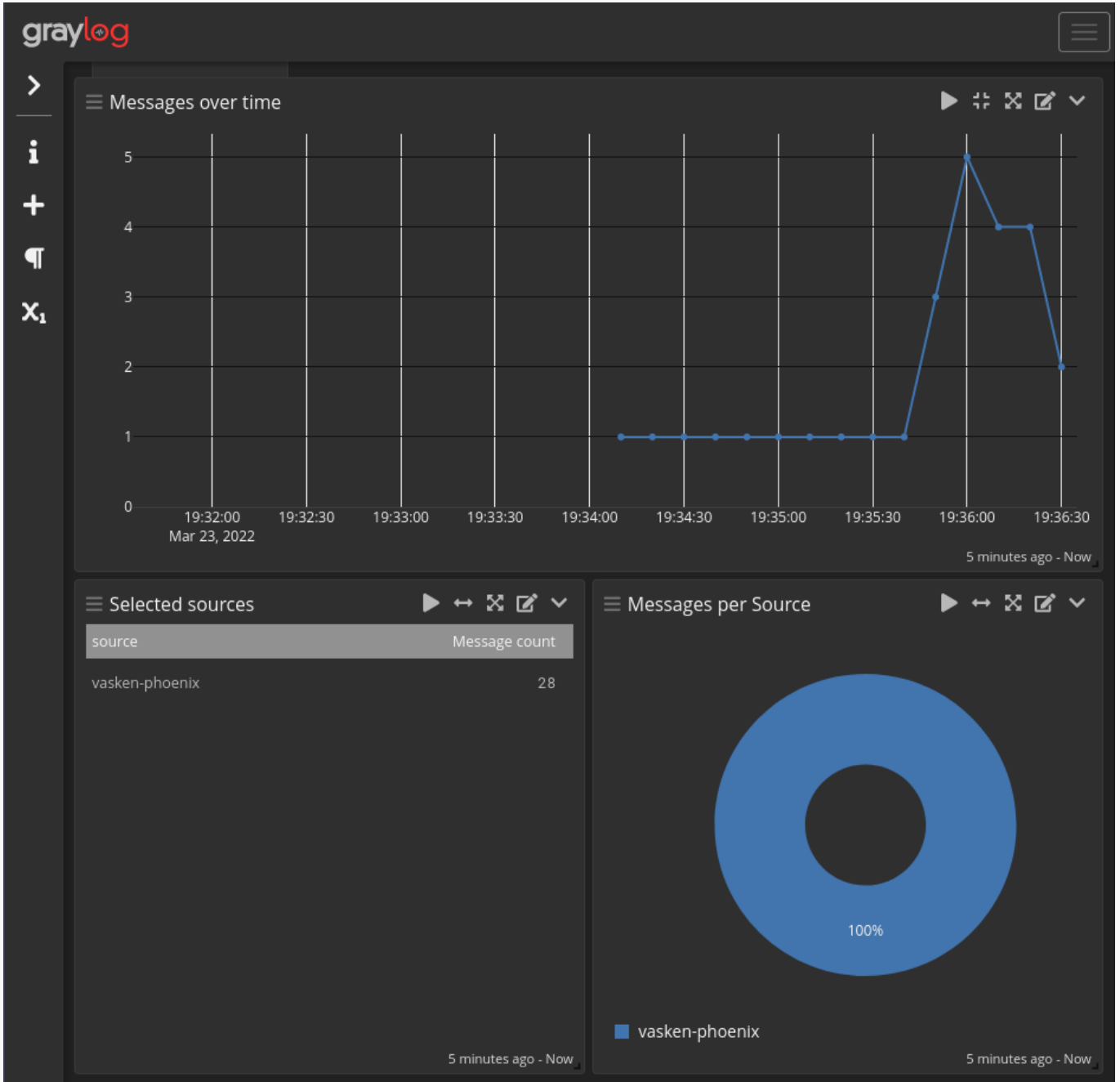


Figure 3: Open up a message

Graylog: custom dashboards



Docker

The difference between a local device, a VM and docker is that docker is ephemeral by design: isolated and stateless. When things are printed to terminal or the filesystem in docker, as soon as the container is restarted, the files are lost.

By default, docker prefers logging to the host device using JSON file or you can log to a data volume among many alternatives. Another is to use the Sidecar approach in Kubernetes where another container handles the logs.

And at this point, I have no idea what I'm talking about…

- https://www.datadoghq.com/blog/docker-logging/
- https://sematext.com/guides/docker-logs/

Grazie per l’attenzione!

Domande?