Logging the right way!

Devi A S L

PyCon India, 2020

About

- @asldevi
- 15 years of experience in industry
- Staff Engineer at Razorpay
- Ex-Architect @ PowerToFly

Razorpay

Why Logging?

We want to find out what had happened

Why Logging?

We want to find out what **exactly** had happened

Why Logging?

We want to find out what **exactly** had happened **quickly**

Why Logging?

We want to find out what **exactly** had happened **quickly** and it’s **cause**

Agenda

- Making logs easily searchable, filterable and aggregate-able
- Balancing debuggability and logging cost
- Tracing requests across microservices in logs

Sample Log Line

```
logger.info("Transaction succeeded with id %s, amount
%f, payment mode %s", txn["id"], txn["amount"],
txn["mode"])
```

Sample Log Line

```
logger.info("Transaction succeeded with id %s, amount
%f, payment mode %s", txn["id"], txn["amount"],
txn["mode"])
```

Razorpay

Searchability

```
[app][info] 2020-10-02T18:23:13.972808: Transaction succeeded
with id Fbgi4yFzwLc4NQ, amount 1025.00, payment mode NEFT
```

- Human readable, not machine readable
- Filter searches ❌*
- Aggregates ❌*

* Unless your log management system is doing the heavy lifting

Treat logs as event streams

Enter Structured Logging

```
~~logger.info("Transaction succeeded with id %s, amount
%f, payment mode %s", txn["id"], txn["amount"],
txn["mode"])~~
```

```
logger.info("new_transaction”, id=txn[‘id’],
amount=txn[‘amount’], payment_mode=txn[‘mode’],
is_successful=True)
```

Structured Logging

```
{
    message: ‘new_transaction’,
    id: ‘Fbgi4yFzwLc4NQ,
    amount: 1025.00,
    payment_mode: ’NEFT’,
    is_successful: True,
    timestamp: 2020-10-02T18:23:13.972808
}
```

- Filter searches ✓
- Aggregates ✓

Structured Logging with `Structlog`

- Drop in replacement to standard library
    - `s/logging.getLogger/structlog.get_logger`
- Plenty of powerful pipelines
    - add metadata, redact sensitive info etc

Razorpay

Logging is not Cheap

Number of logs $\propto$ number of requests

Cost $\propto$ number of logs * cost of (network + storage) * retention period

Logging is not Cheap

At Razorpay, we have 2 TB logs/day flowing in

💰 💰 💰

Optimize for cost or debuggability?

Dynamic logging

Change Log Level in Run time

```
logging.config.listen(<port>)
```

Works with fileConfig, dictConfig etc

Change log level in run time

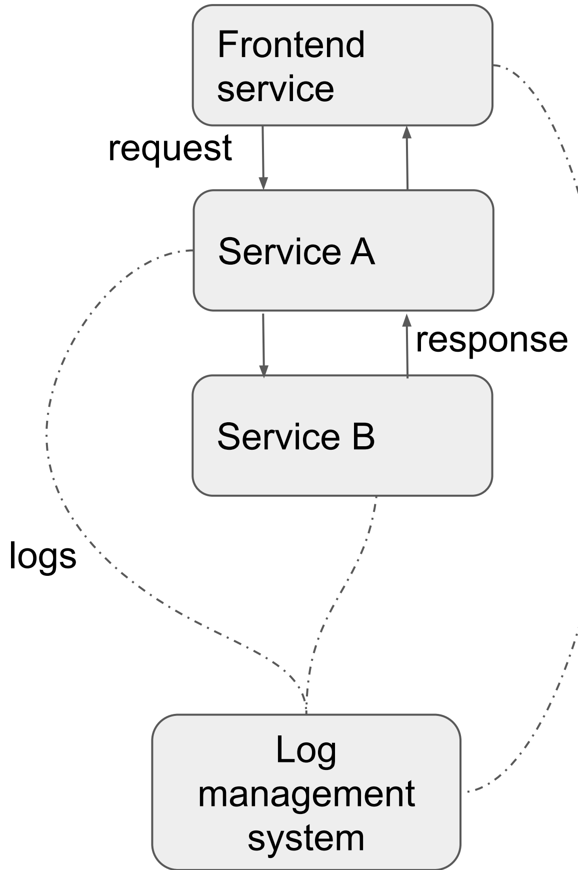
```
logger = structlog.getLogger(...)

logger.setLevel(level)
```

$\uparrow$

can be changed
run-time
based on metrics
errors/latencies

Logging microservices

Request flow and logs



```
{
      message: log_line_from_frontend,
      timestamp: 91
}
{
      message: log_line_from_frontend,
      timestamp: 92
}
{
      message: log_line_from_A,
      timestamp: 93
}
{
      message: log_line_from_A,
      Timestamp: 94
}
{
      message: log_line_from_B,
      timestamp: 95
}
```

Tracing requests end-to-end

```
{
     message: log_line_from_frontend,
     timestamp: 91
}
{
     message: log_line_from_frontend,
     timestamp: 92
}
{
     message: log_line_from_A,
     timestamp: 93
}
{
     message: log_line_from_A,
     Timestamp: 94
}
{
     message: log_line_from_B,
     timestamp: 95
}
```

$\longrightarrow$

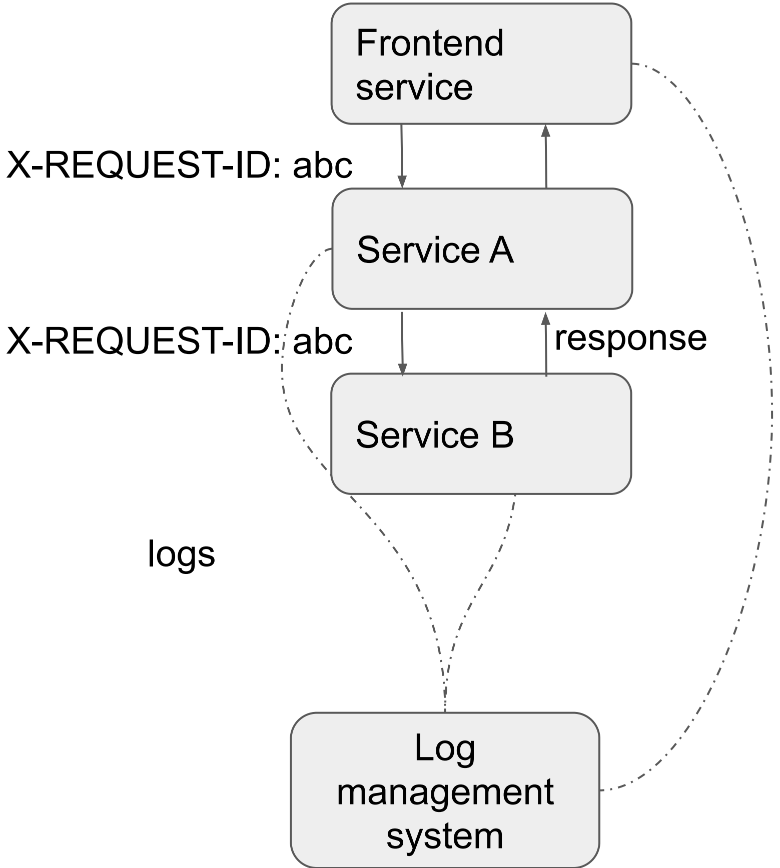
```
{
     message: log_line_from_frontend,
     timestamp: 91, request_id: abc
}
{
     message: log_line_from_frontend,
     timestamp: 92, request_id: xyz
}
{
     message: log_line_from_D,
     timestamp: 93, request_id: abc
}
{
     message: log_line_from_D,
     Timestamp: 94, request_id: def
}
{
     message: log_line_from_A,
     timestamp: 95, request_id: abc
}
```

Tracing requests end-to-end

```
{
      message: log_line_from_frontend, ...
      timestamp: 91
}
{
      message: log_line_from_frontend, ...
      timestamp: 92
}
{
      message: log_line_from_A,  ...
      timestamp: 93
}
{
      message: log_line_from_A,  ...
      Timestamp: 94
}
{
      message: log_line_from_B,  ...
      timestamp: 95
}
```

```
{
      message: log_line_from_frontend, ...
      timestamp: 91, request_id: abc
}
{
      message: log_line_from_frontend, ...
      timestamp: 92, request_id: xyz
}
{
      message: log_line_from_A, ...
      timestamp: 93, request_id: abc
}
{
      message: log_line_from_A, ...
      Timestamp: 94, request_id: def
}
{
      message: log_line_from_B, ...
      timestamp: 95, request_id: abc
}
```

Request ID through Header propagation



Recap

- Set log levels appropriately, put your own if required
- Machine readable logs aka structured logs - always
- Dynamic logging to have control of log volume
- Unique correlation id across all microservices connecting them all

References

- https://docs.python.org/2/howto/logging-cookbook.html#configuration-server-example (is similar in py3)
- https://www.structlog.org/en/stable/index.html
- https://medium.com/hiredscore-engineering/logging-lets-do-it-right-41d568d3bfcd

Questions?

Grow with us

@asldevi