The Secret Life of Software

Rediscover Your Production System

The Secret Life of Software

Rediscover Your Production System

$ whoami

What is your code doing?

Complexity

We are building more complex systems than ever before.

It is rare to find a system consisting of "just" a webserver and database.

You'll likely have

- Multiple Web Servers and proxies

You'll likely have

- Multiple Web Servers and proxies
- Multiple Databases and indexes

Hopefully also...

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability

Hopefully also...

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices

Then you'll need

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices
- Caching

Then you'll need

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices
- Caching
- Multiple geographic regions

Then you'll need

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices
- Caching
- Multiple geographic regions
- CDN

Then you'll need

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices
- Caching
- Multiple geographic regions
- CDN
- Business Analytics

Then you'll need

- Multiple Web Servers and proxies
- Multiple Databases and indexes
- High Availability
- Support many devices
- Caching
- Multiple geographic regions
- CDN
- Business Analytics
- CI/CD Pipeline

...and don't forget.

Each of these involves N microservices

So, monitoring?

Monitoring

vs

Observability

Observability Mindset

Three Pillars

Logs

Metrics

Tracing

Logs

Likely familiar to many of you...

```
INFO workflow_trace Starting workflow [name=wf, input={container: overcloud}]
INFO workflow_trace Workflow 'wf' [IDLE -> RUNNING, msg=None] (execution_id=ID)
...
INFO workflow_trace Task 'send_message' [RUNNING -> SUCCESS, msg=None] (execution_id=ID)
```

Or, generally...

```
TIMESTAMP PID LOG_LEVEL LOG_NAME MESSAGE
```

Logs - Errors

- Exception handling (with services like Sentry)
- Alerts/Notifications
- Open Source

```
from sentry_sdk import init, capture_message
init("mydsn@sentry.io/123")
def my_app():
    raise Exception("Everything is broken")
```

Logs - Add Structure

- Use structlog
- Pretty logs for development
- Structured data for production

```
>>> import structlog
>>> LOG = structlog.get_logger("myapp.auth")
>>> LOG.info("User login failed", login_attempt=10, other_data="datas")
2019-07-02 13:36.27 User login failed login_attempt=10 other_data=datas
>>> # Or with the JSON renderer
>>> LOG.info("User login failed", login_attempt=10, other_data="datas")
2019-07-02 13:36.27 {"event": "User login failed",
                     "login_attempt": 10, "other_data": "datas"}
```

Logs - Add Request UUIDs

```
import flask; import uuid; app = flask.Flask(__name__)
def before():
   request_id = flask.request.headers.get('X-Request-ID')
   if not request_id:
       request_id = str(uuid.uuid4())
   flask.g.request_id = request_id

def after(resp):
   resp.headers.add('X-Request-ID', flask.g.request_id)
   return resp

app.before_request(before)
app.after_request(after)
```

Logs - Limitations

- Too granular, hard to see trends
- Hard to monitor
- Expensive to store

Metrics

There are many options here like Prometheus, InfluxDB and Datadog

Basic metrics; error rate, response time, request volume

Metrics - Database

- Number of database queries
- Query duration

Metrics - statsd

```
import statsd
client = statsd.StatsClient("localhost", 8125)
timer = Timer('application_name')

@timer.decorate()
def my_fn():
    a = 1
    with timer.time("measure-span")
        b = a * 10
    return b
```

Tracing

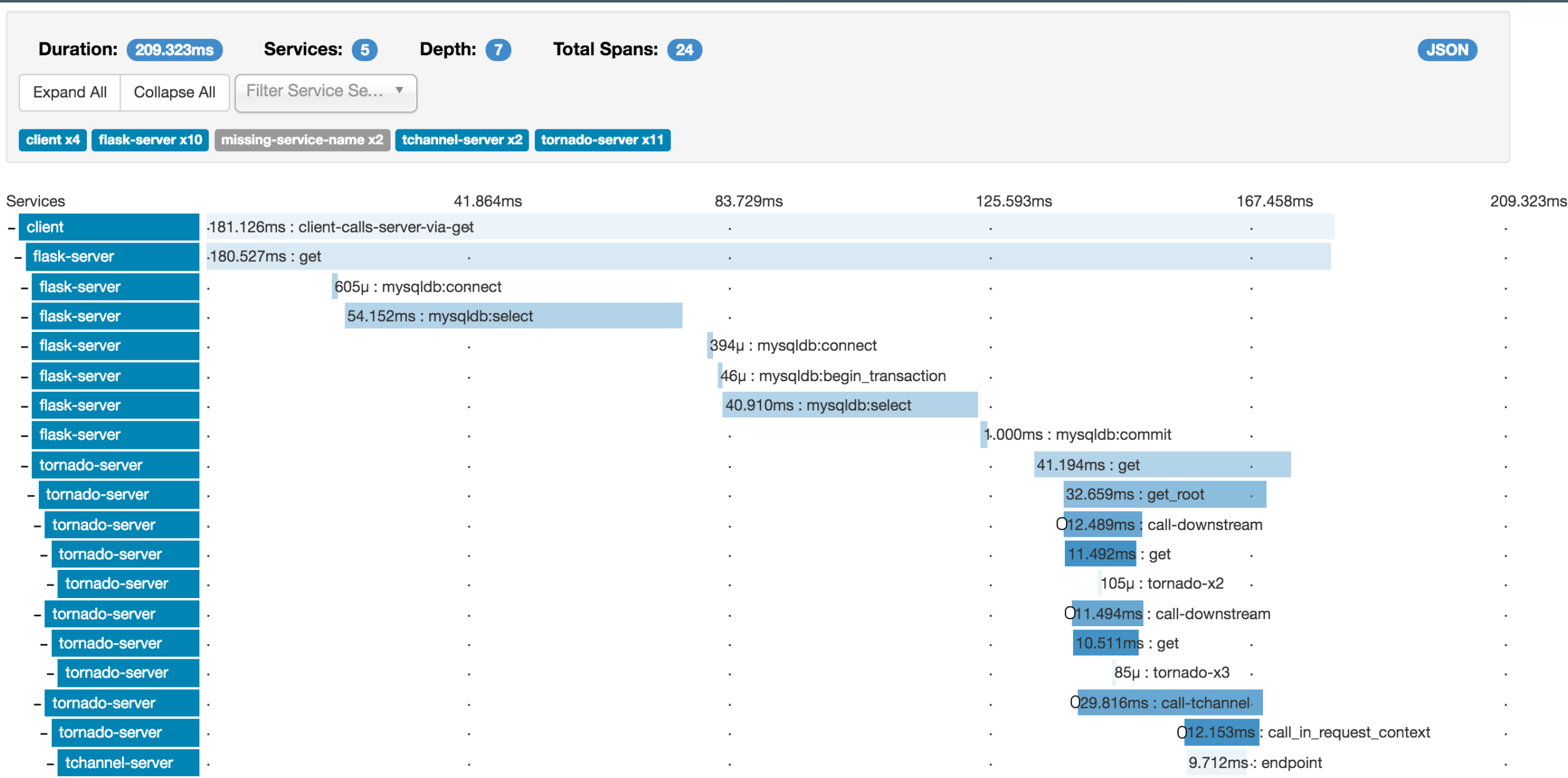
- Possibly the most useful
- Possibly also the hardest
- Solutions from Datadog APM, Elastic APM, Zipkin and others

Tracing Integration

```
import opentracing
from flask_opentracing import FlaskTracing

app = Flask(__name__)

opentracing_tracer = ## some OpenTracing tracer implementation
tracing = FlaskTracing(opentracing_tracer, True, app, [optional_args])
```


Duration: 209.323ms
Services: 5
Depth: 7
Total Spans: 24
JSON
Expand All
Collapse All
Filter Service Se...
client x4
flask-server x10
missing-service-name x2
tchannel-server x2
tornado-server x11
Services
41.864ms
83.729ms
125.593ms
167.458ms
209.323ms
client
181.126ms : client-calls-server-via-get
flask-server
180.527ms : get
flask-server
605µ : mysqldb:connect
flask-server
54.152ms : mysqldb:select
flask-server
394µ : mysqldb:connect
flask-server
46µ : mysqldb:begin_transaction
flask-server
40.910ms : mysqldb:select
flask-server
1.000ms : mysqldb:commit
tornado-server
41.194ms : get
tornado-server
32.659ms : get_root
tornado-server
12.489ms : call-downstream
tornado-server
11.492ms : get
tornado-server
105µ : tornado-x2
tornado-server
11.494ms : call-downstream
tornado-server
10.511ms : get
tornado-server
85µ : tornado-x3
tornado-server
29.816ms : call-tchannel
tornado-server
12.153ms : call_in_request_context
tchannel-server
9.712ms : endpoint

Recap

Logs

Metrics

Tracing

Are we done?

It is not enough

- Just doing these doesn't mean you are done
- How you use this data and how to share and present it matters
- Integrating all of these together is where the real power lies

A Practical Approach

- Start collecting data
- Learn from it
- Rinse repeat

Further Reading

> Three Pillars, Zero Answers:
> We Need to Rethink Observability
> Ben Sigelman

Thanks!