Aalto University
School of Electrical
Engineering

Logging, metrics and tracing

7.3.2019
Santeri Paavolainen

Why logging, metrics and tracing relevant?

- **In a simple (monolithic) system**
  - Single logging configuration, few metrics to monitor
  - Single (or only a few) logging locations
  - Persistent system → backups usually sufficient for retention
- **Distributed (e.g. microservice) system**
  - Microservices composed of different subsystems and components
  - Lifetime of a single instance, container etc. limited
  - Huge variability in logging methods, metrics to collect * number of individual collection points (instances/containers) large
- **Distributed systems notoriously difficult to debug**
  - "No information" is a disaster
  - Post-hoc often difficult to have information (instance gone!)
- **A priori design is necessary!**

Differences

Logging

- "What happened?"
- Development, problem-solving, auditing (audit logs)
- Logging levels (typical)
  - *TRACE, DEBUG, INFO, WARNING, ERROR, FATAL*
- Structured vs. unstructured logs
- Tags

Metrics

- "What is state?"
- Instaneous, time average, series
- Absolute vs. relative
- Huge variability
  - *LB 2xx/3xx/4xx/5xx*
  - *Method call time*
- Tagged metrics
  - *Region, instance type, code version, service mode, …*

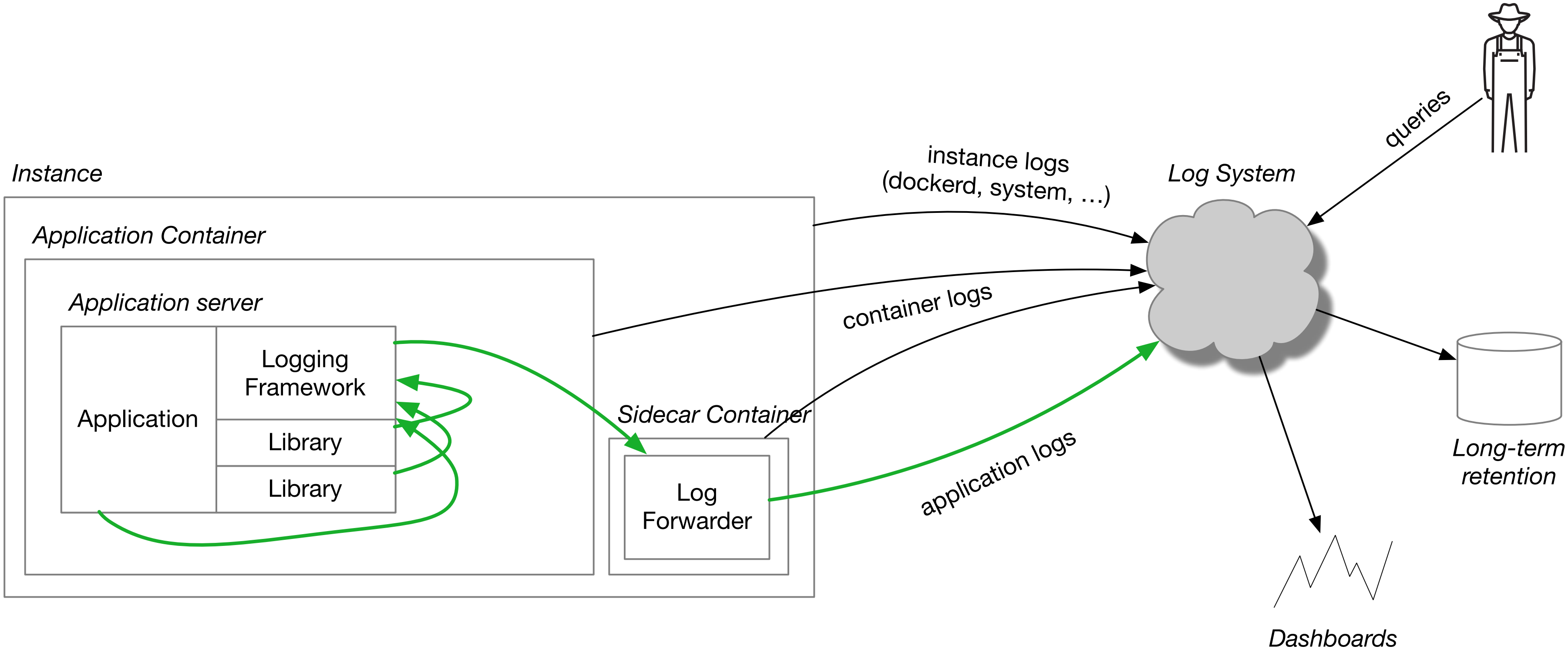
Tracing

- "How these are related?"
- Tag "initial request" with a trace id
  - *Pass trace id to any downstream requests*
  - *Branches of new trace ids linked to parent*
- Trees
- Need to include trace id in logging (tags etc.)

Logging

Considerations: Logging

- **What to log?**
  - Affects total volume
  - Preferably run-time configurable, minimally at deployment
- **How to log?**
  - Format, timestamps etc.
  - Link to program code
- **Security considerations**
  - Sensitive information
  - Security of logs (authenticity)
- **How to collect?**
  - To-disk and separate sender?
  - Direct network?
  - What protocol and format?
- **Where to collect?**
  - Server / system / software selection
  - Overall infrastructure design
- **What to collect?**
  - Retention time
  - Tiered storage


Instance
Application Container
Application server
Application
Logging Framework
Library
Library
Sidecar Container
Log Forwarder
instance logs (dockerd, system, ...)
container logs
application logs
Log System
queries
Long-term retention
Dashboards

Logging

- **Tons of logging frameworks, libraries and tools over all languages**
  - Some languages have pretty standardized plug-in logging mechanisms (some don't)
  - When using libraries etc. may end up with conflicts (Java …)
  - What to choose, how to configure, how to use often reflect programmer's preferences → no universal rule to follow
  - Some performance considerations too for high-performance applications
- **Overall these low-level concerns not really part of this course**

Some numbers

- **Assumptions**
  - 1 customer arrives / second
  - 0.5 request / s / customer (average)
  - 30 minutes / customer on site
  - 30 log entries / request
  - 100 B / log entry (a bit over a 80 character line)
- **Result:**
  - 27 000 log entries / s
  - 2.7 MiB / s
  - 233 GiB / day

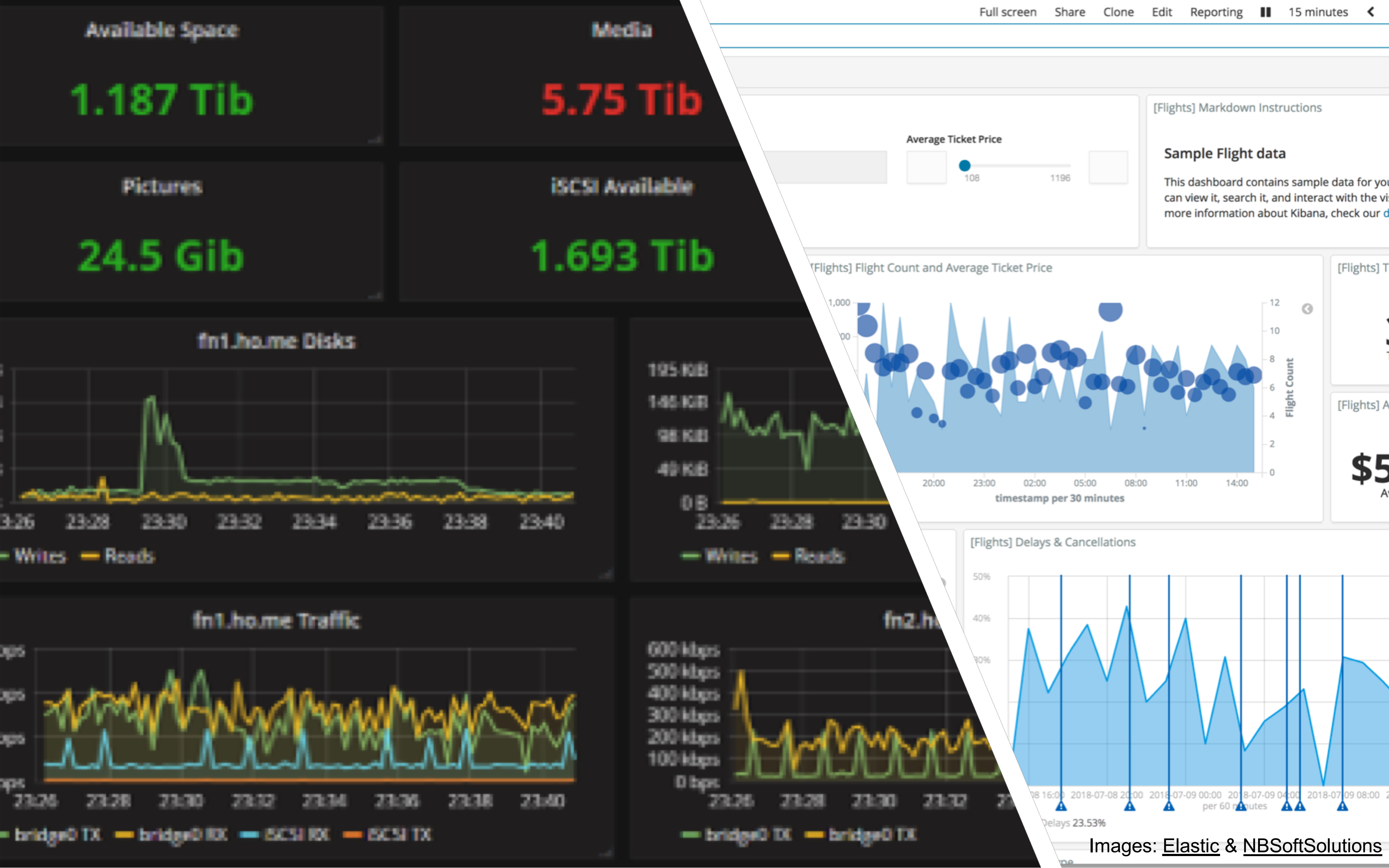
30 minutes low or high?

More or less log entries per request?

Maybe can compress structured logs?

How long retention? Does it compress well?

Metrics



Sample Flight data

This dashboard contains sample data for you
can view it, search it, and interact with the vi
more information about Kibana, check our d

Images: Elastic & NBSoftSolutions

Metrics

- **Values of some units**
  - 5xx over last minute
  - # of active users
  - CPU% usage now
  - # of containers on instance
  - $ sales over last minute
- **Collectors usually don't store historical data**
  - Limited storage for time averages
- **Usually LOTS and LOTS and LOTS of metrics**
  - Duplicity across systems, instances and containers
  - Some unique over whole system ($ sales)
- **Visualization important**
  - Humans bad at interpreting raw numbers, good at spotting visual trends
- **Hard numbers useful for alerting**
  - Alerting a whole another topic, not going to that on this course (part of operations)

So …

What to collect?

- OS, container, DB, other apps usually instrumented themselves
- Highly dependent on your problem
  - *Performance critical? Business value? For marketing?*
- Generally: You will be more sorry for not collecting enough metrics
  - *But they take time …*

How to collect?

- Problematic
- Java has JMX framework, other languages usually don't → need libraries to push, per-framework components

- **How long to collect?**
  - Preferably keep long-term at least in aggregated form

How to process?

- Defining meaningful (=USEFUL!) graphs & dashboards takes time
  - *Actually an UX problem!*
- Alerting … let's not go there

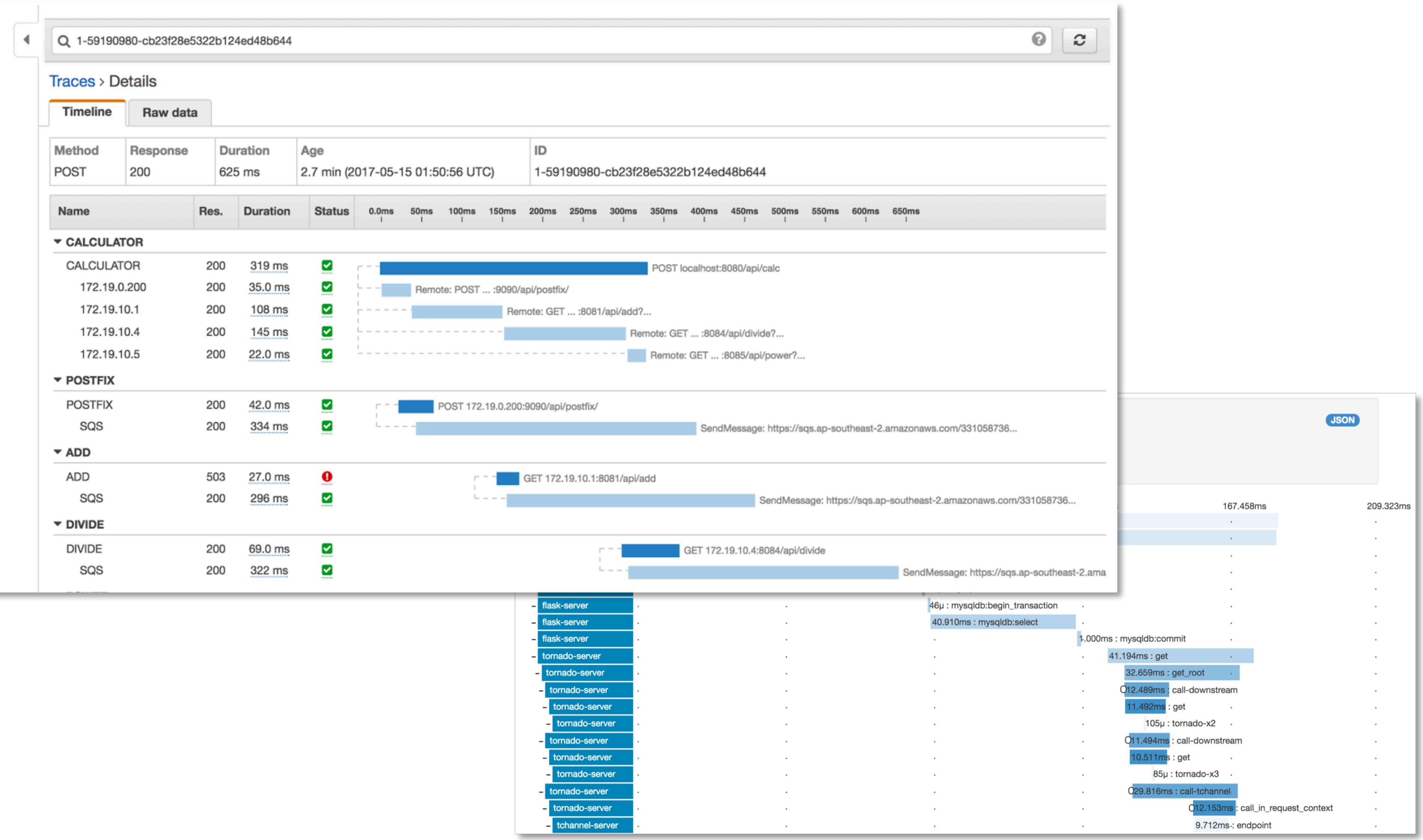
Tracing

Game of Guess Which Go Together

```
INFO  [2019-02-02T15:11:19Z] c.a.b.y: Incoming request /foo user=null

INFO  [2019-02-02T15:11:20Z] c.a.b.y: Incoming request /der user=fnord valid_until=2019-02-05T00:00:00 from=GB

ERROR [2019-02-02T15:11:20Z] c.a.b.r.a: Exception InvalidParameterResponse at ProcessFile.java:1223

WARN  [2019-02-02T15:11:19Z] c.a.b.z: Invalid password for user=gabagaba

DEBUG [2019-02-02T15:11:22Z] c.a.b.o.y: d=0x555422231a a=null b=[gerbil,snaptree] action=get status=partial-success
remote=sp-54521.c.a.b.local

TIMEOUT cass12.cluster.local: write queue full, client not draining

INFO  [2019-02-02T15:11:22Z] c.a.b.a9: received=ProcessEmail from=unknown to=anuser@example.com body=template-voucher-
offer retry=0

INFO  [2019-02-02T15:11:23Z] c.a.b.a9: received=ProcessEmail from=unknown to=anuser@example.com body=template-voucher-
offer retry=0

INFO  [2019-02-02T15:12:54Z] c.a.b.a9: received=ProcessEmail from=unknown to=anuser@example.com body=template-voucher-
offer retry=0

Error at @221125abf: Invalid allocation on request=0x66621a581
```

Distributed tracing

- **Approaches**
  - Annotate log entries with trace identifiers (post-hoc analysis)
  - Have separate tracing logic (focus on timings and dependencies)
  - Not exclusive, can be used together (performance vs. debugging)
- **Solutions**
  - AWS X-Ray, Datadog APM & Tracing, Google Stackdriver, …
  - OpenTracing, Zipkin, Jaeger, OpenCensus, …
- **Generally less understood and applied (wrt logging and metrics)**



1-59190980-cb23f28e5322b124ed48b644

Traces › Details

Method	Response	Duration	Age	ID
POST	200	625 ms	2.7 min (2017-05-15 01:50:56 UTC)	1-59190980-cb23f28e5322b124ed48b644

Name	Res.	Duration	Status	
CALCULATOR				
CALCULATOR	200	319 ms	✓	POST localhost:8080/api/calc
172.19.0.200	200	35.0 ms	✓	Remote: POST ... :9090/api/postfix/
172.19.10.1	200	108 ms	✓	Remote: GET ... :8081/api/add?...
172.19.10.4	200	145 ms	✓	Remote: GET ... :8084/api/divide?...
172.19.10.5	200	22.0 ms	✓	Remote: GET ... :8085/api/power?...
POSTFIX				
POSTFIX	200	42.0 ms	✓	POST 172.19.0.200:9090/api/postfix/
SQS	200	334 ms	✓	SendMessage: https://sqs.ap-southeast-2.amazonaws.com/331058736...
ADD				
ADD	503	27.0 ms	!	GET 172.19.10.1:8081/api/add
SQS	200	296 ms	✓	SendMessage: https://sqs.ap-southeast-2.amazonaws.com/331058736...
DIVIDE				
DIVIDE	200	69.0 ms	✓	GET 172.19.10.4:8084/api/divide
SQS	200	322 ms	✓	SendMessage: https://sqs.ap-southeast-2.ama

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

Images: zipkin.io & AWS

Summary

- **Which one you prefer:**
  - System you KNOW is hosed?
  - System which APPEARS to work?
- **Logging, metrics and tracing are tools for the FIRST one**
  - Identifying the problem **metrics**
  - Locating the problem **logging (tracing)**
  - Understanding the problem **logging**
  - After fix is rolled out, verifying that problem has gone away **all**