Monitoring and observability of distributed systems

Michał Kalisz
RTB House

http://mimuw.rtbhouse.com

Introduction

Why is it worth to monitor systems?

- To have visibility of all hardware and software components
- To prevent from system incidents, faults or outages
- To be capable to find cause of problem
- To predict system growth
- To reduce manual effort

Introduction

Real world outage example.
AWS 07.12.2021 - one of regions went down

- From 7:30 AM PST to 2:25 PM PST
- https://aws.amazon.com/message/12721/
- unexpected behaviour from a large number of clients inside the internal network
- persistent congestion and performance issues on the devices connecting internal and main AWS network
- This congestion immediately impacted the availability of real-time monitoring data for our internal operations teams, which impaired their ability to find the source of congestion and resolve it.
- Limited communication:
    - Our Support Contact Center also relies on the internal AWS network.
    - Service Health Dashboard (after 1 hours first information in status dashboard was visible)

Introduction

Other examples:

- Air-Traffic Control System in LA Airport
  - The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. To make matters worse, a backup system that was supposed to take over in such an event crashed within a minute after it was turned on. The outage disrupted about 800 flights across the country.
  - Cause: Inside the control system unit is a countdown timer that ticks off time in milliseconds. The VCSU uses the timer as a pulse to send out periodic queries to the VSCS. It starts out at the highest possible number that the system's server and its software can handle—232. It's a number just over 4 billion milliseconds. When the counter reaches zero, the system runs out of ticks and can no longer time itself. So it shuts down.
- Knight Capital's $440 million loss
  - Knight Capital's software went out and bought at the "market", meaning it paid ask price and then sold at the bid price--instantly. Over and over and over again. One of the stocks the program was trading, electric utility Exelon, had a bid/ask spread of 15 cents. Knight Capital was trading blocks of Exelon common stock at a rate as high as 40 trades per second--and taking a 15 cent per share loss on each round-trip transaction. As one observer put it: "Do that 40 times a second, 2,400 times a minute, and you now have a system that's very efficient at burning money".

  More: https://www.cse.psu.edu/~gxt29/bug/softwarebug.html

Introduction

Monitoring

- Monitoring is the systematic process of collecting, analyzing and using information to track a project's progress toward reaching its objectives and to guide management decisions.

Observability

- Observability is the ability to measure a system's current state based on the data it generates, such as logs, metrics, and traces.

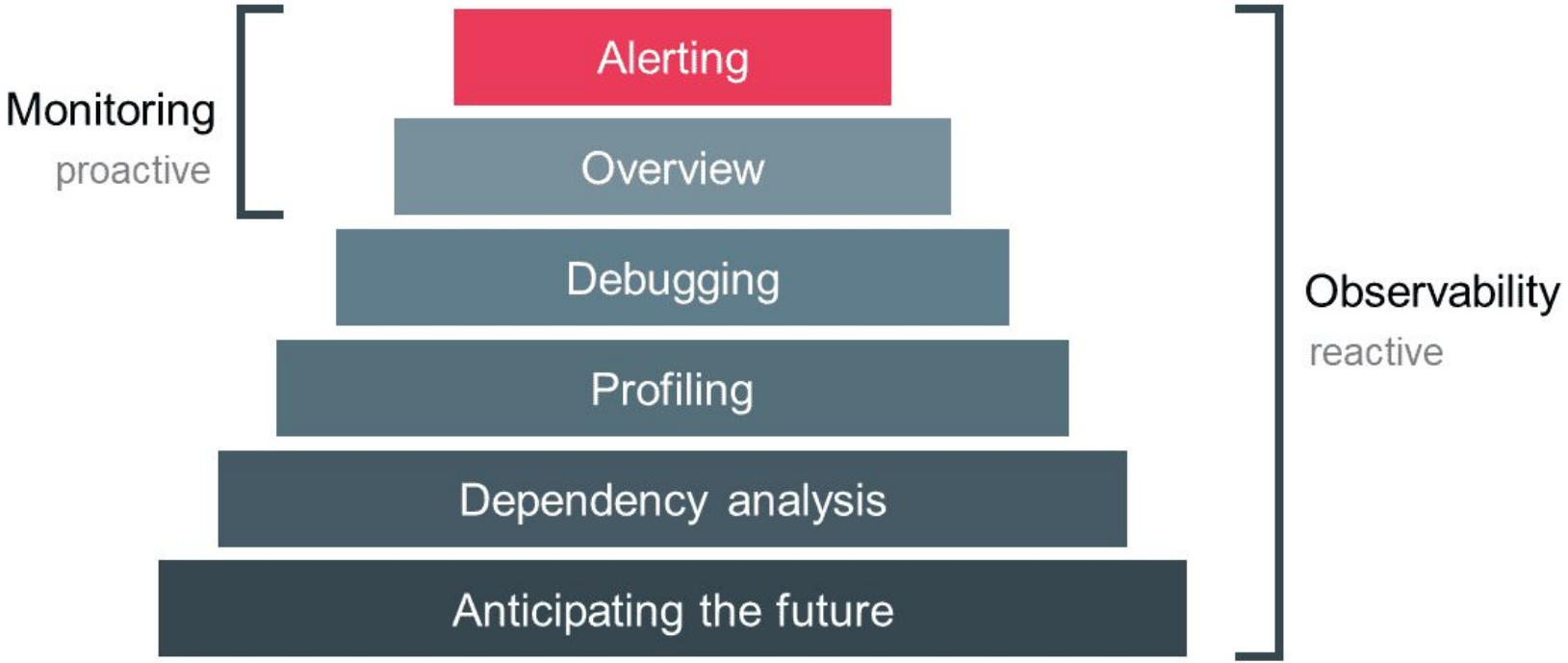
Introduction

Monitoring tells you whether the system works. Observability lets you ask why it's not working.

Baron Schwarz

https://orangematter.solarwinds.com/2017/09/14/monitoring-isnt-observability/

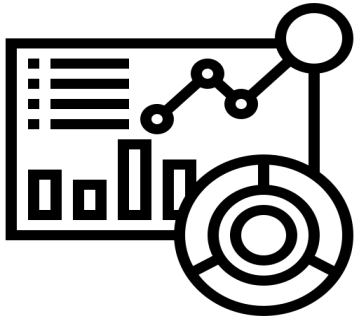
Introduction

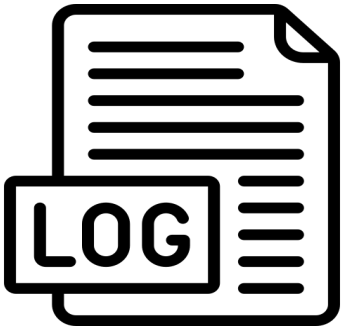


Practical Distributed Systems, 2022

Monitoring & Observability

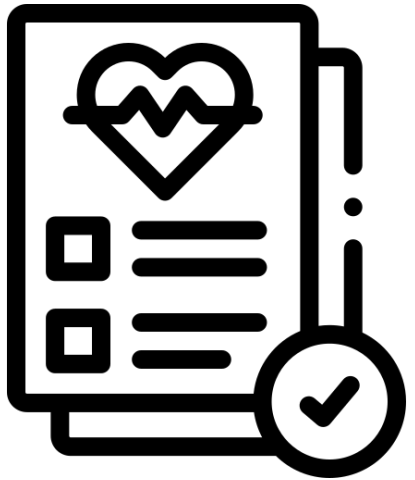
- Health checks
- Metrics
- Logs
- Tracing





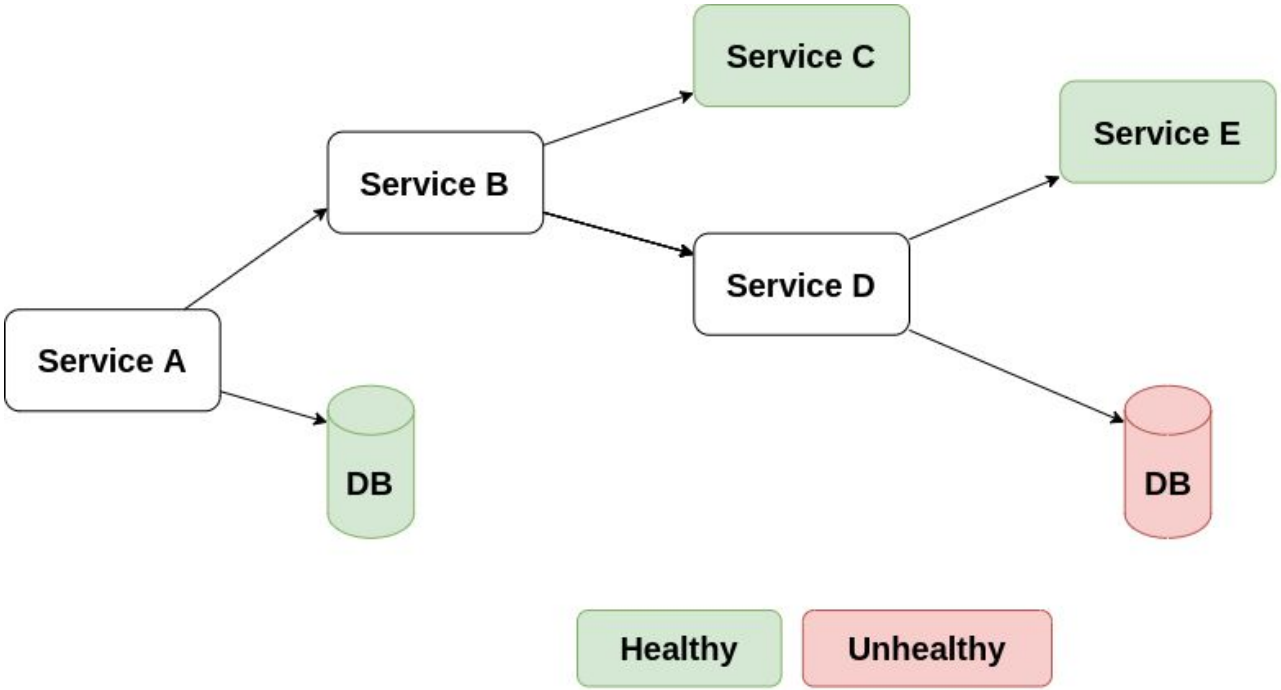
Health checks

- Verify if service is capable to operate, ie. to handle request?
- This can be achieved by answering the following questions:
  - Is service running
  - Has connection to database
  - Dependent services are healthy



Health checks

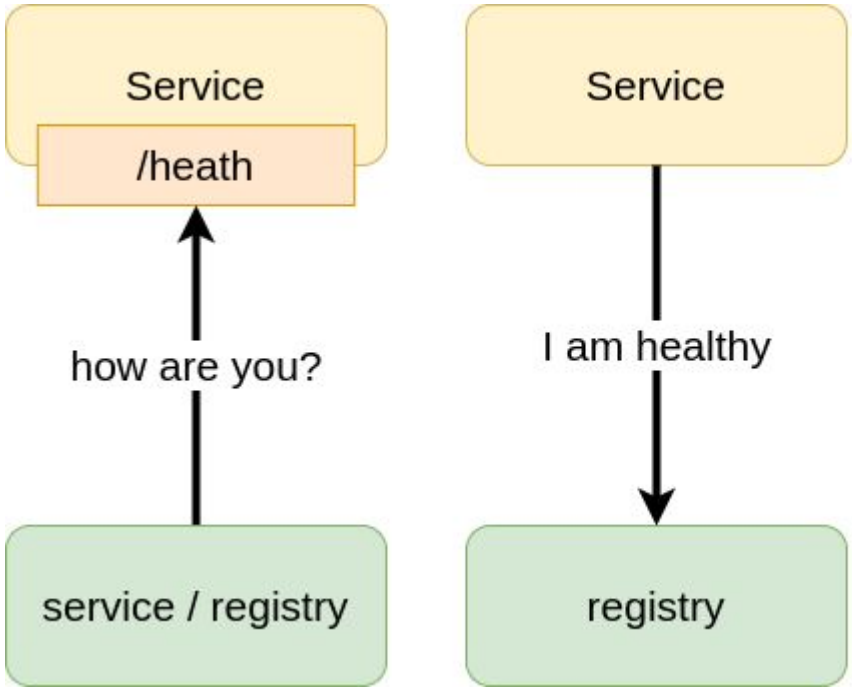
- Service dependencies



What health status should have services A, B, D ?

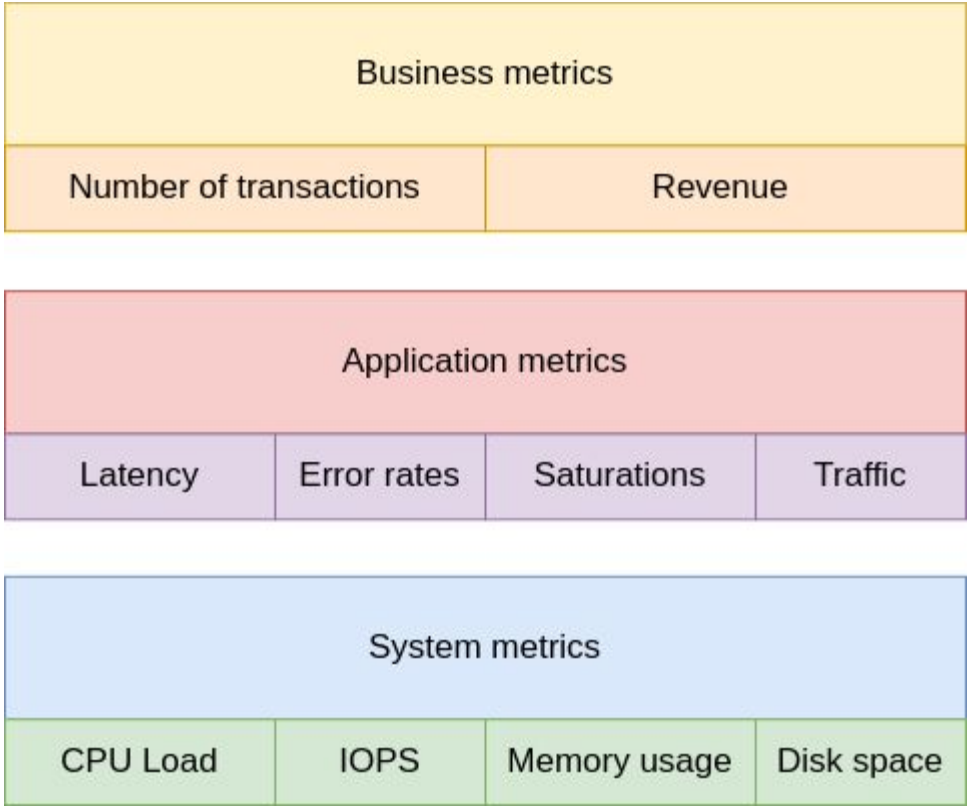
Health checks

- Service status - pull vs push



Metrics

- Metrics groups:
  - System metrics
  - Service metrics
  - Business metrics



Metrics

- “Four Golden Signals” for systems monitoring
  - Latency
    - The time it takes to service a request
  - Traffic
    - A measure of how much demand is being placed on your system
  - Errors
    - The rate of requests that fail
  - Saturation
    - How "full" your service is

Service metrics

- service level indicators (SLIs)
  - a measure of the service level provided by a service provider to a customer
- service level objectives (SLOs)
  - a target value or range of values for a service level that is measured by an SLI.
- service level agreements (SLAs)
  - an explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain

Monitoring metrics types (for dropwizard)

- Counter
- Gauge
- Histogram
- Timers
- Meters

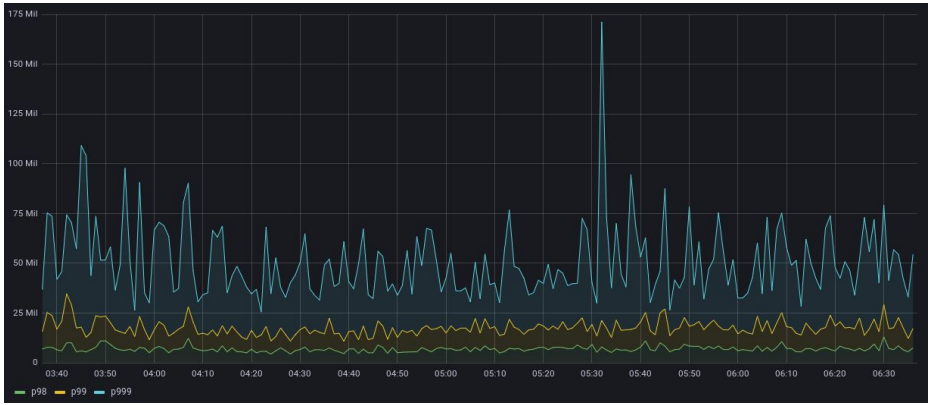
Monitoring metrics types - Gauge / Counter

- Gauge - represents a single numerical value
  Examples:
  - Memory usage
  - Processing queue size

- Counter - enable to increase / decrease / reset value
  Examples:
  - Errors counter
  - Completed tasks

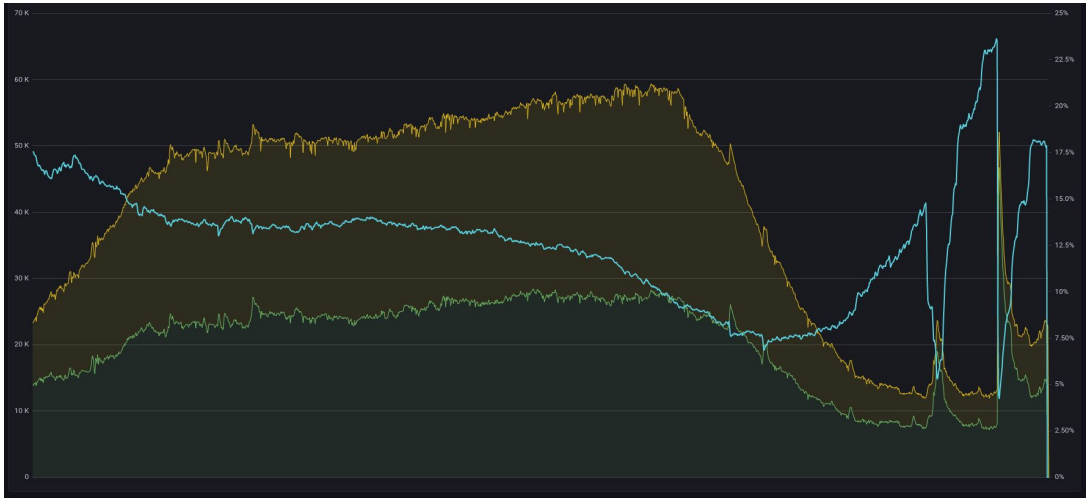
Monitoring metrics types - Histogram

- A Histogram measures the distribution of values in a stream of data
- Measures:
  - minimum, maximum, mean
  - median, 75th, 90th, 95th, 98th, 99th, and 99.9th percentiles.



Monitoring metrics types - Meter

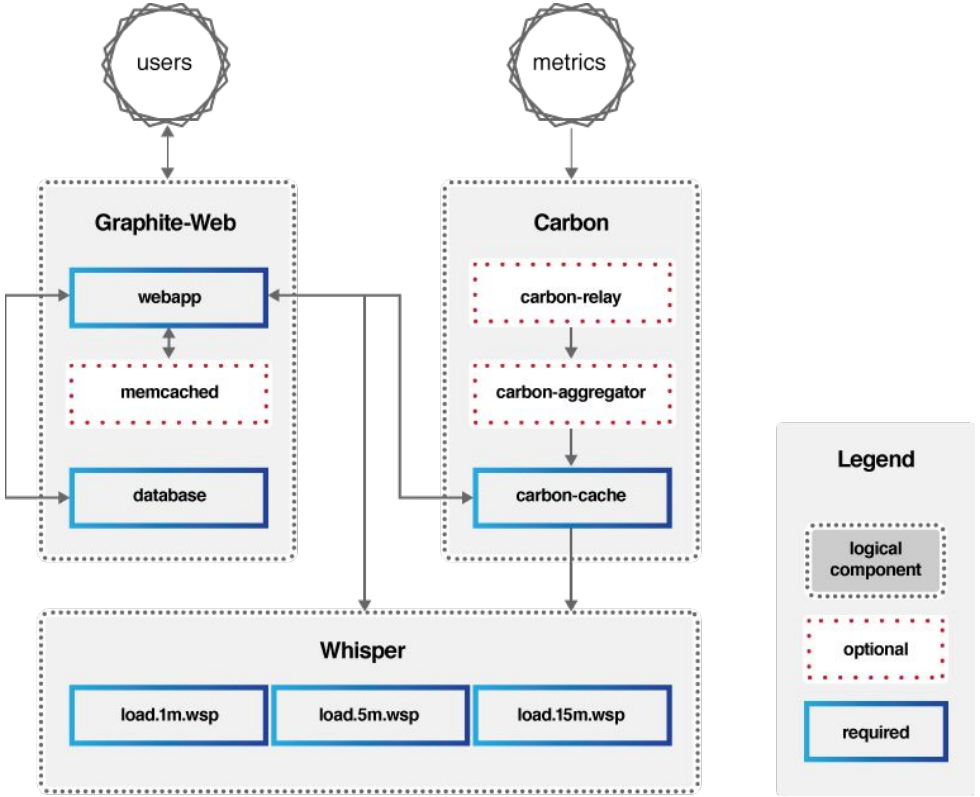
- Meters - measures the rate at which a set of events occur.
- Average rates:
  - Service entire lifetime
  - the 1-, 5-, and 15-minute moving averages.
- Examples:
  - When average is enough - Like unix *top* tool
  - Processed events



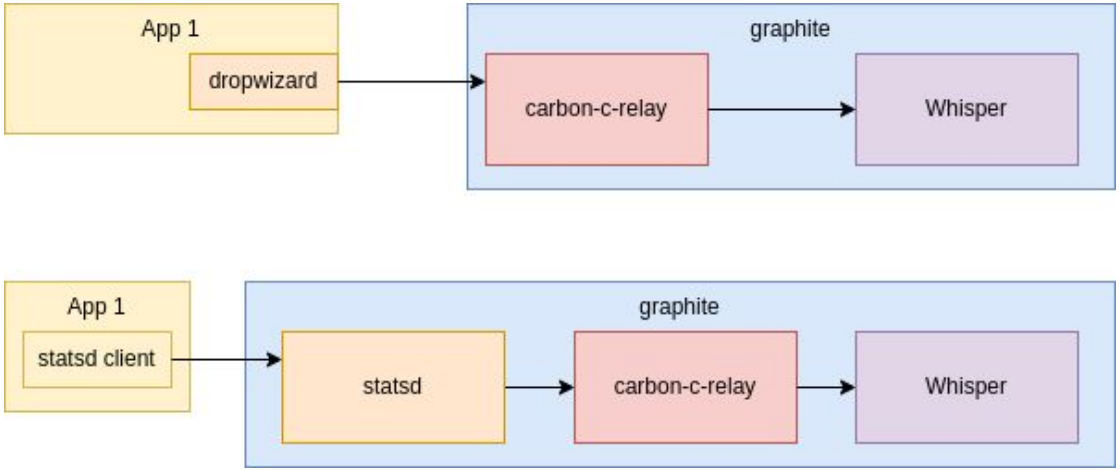
Monitoring metrics types - Timer

- A timer is basically a histogram of the duration of a type of event and a meter of the rate of its occurrence.

- Passive (push)
- Components:
  - Carbon - listen, process, aggregate data
  - Whisper - storage
  - Graphite-Web - GUI & API for rendering graphs and dashboards

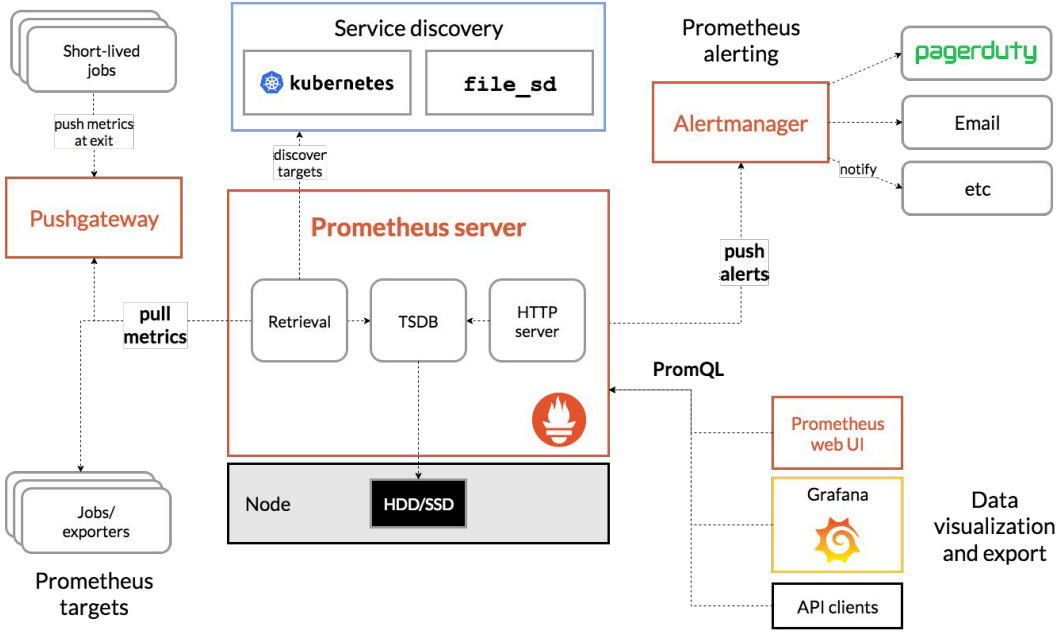


- Client-side aggregation:
  - dropwizard / micrometer library on client side
- Server-side aggregation:
  - Statsd - single metric aggregation



Prometheus

- Active (pull)
- the main Prometheus server which scrapes and stores time series data
- client libraries for instrumenting application code
- a push gateway
- an alertmanager to handle alerts



Logs structure

- Provides details about unexpected or inconsistent event
- Can provide processing result for debug purposes
- Should be meaningful

```
2022-03-23 19:09:01,048 [thread-24] ERROR cannot find resource

2022-03-23 19:09:01,048 [thread-24] ERROR timed out
```

Logs structure

- For single node application - what data should be added to logs?
  - Timestamp
  - Level of message
  - Name of service
  - Request ID
  - Message
  - User / principal ID

Logs structure

- For single node application - what we expect to do with logs?
  - Easy access - defined path / place to look for
  - Enable filter / search
  - Enable to group
- In many case simple unix command line tools would be enough
  - grep, tail/head, sed/awk, sort, uniq etc.

Logs structure

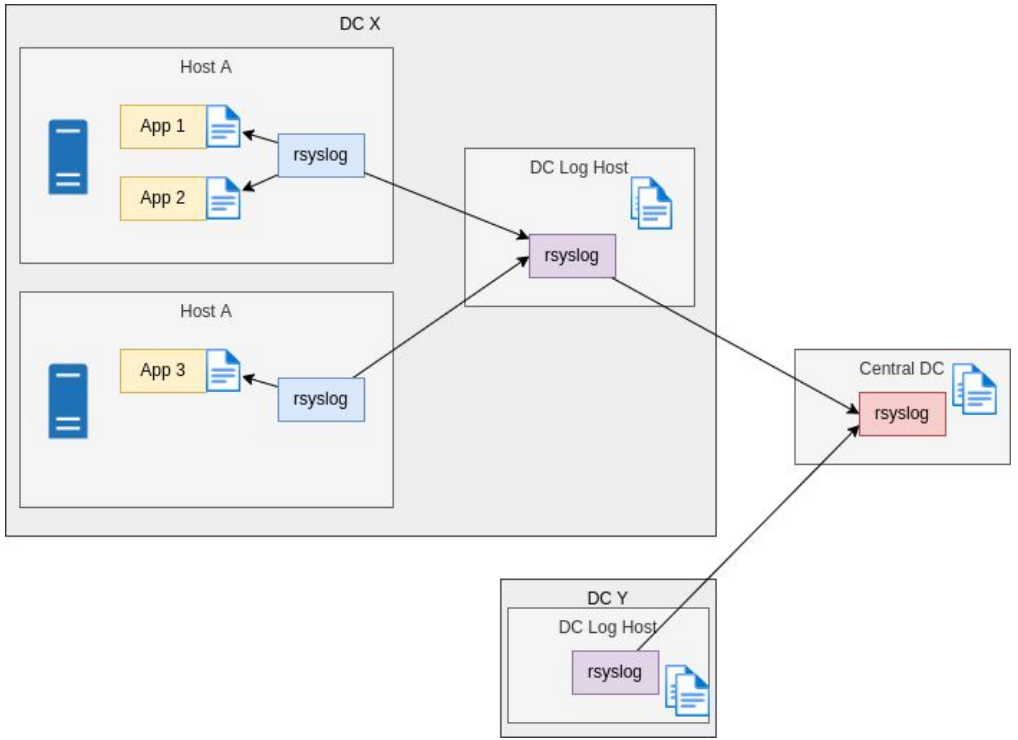
- For distributed systems, microservices architecture etc.
  - Centralized- one place to access logs from many applications
  - Enable to filter / search
  - Enable to group
  - Enable to correlate events between services

Logs structure

- Extended logs information:
  - Timestamp
  - Level of message
  - Name of service
  - Request ID
  - Message
  - User / principal ID
  - Application version (commit hash)/ runtime env. / docker img?
  - Node name, DC/region name
  - Correlation ID

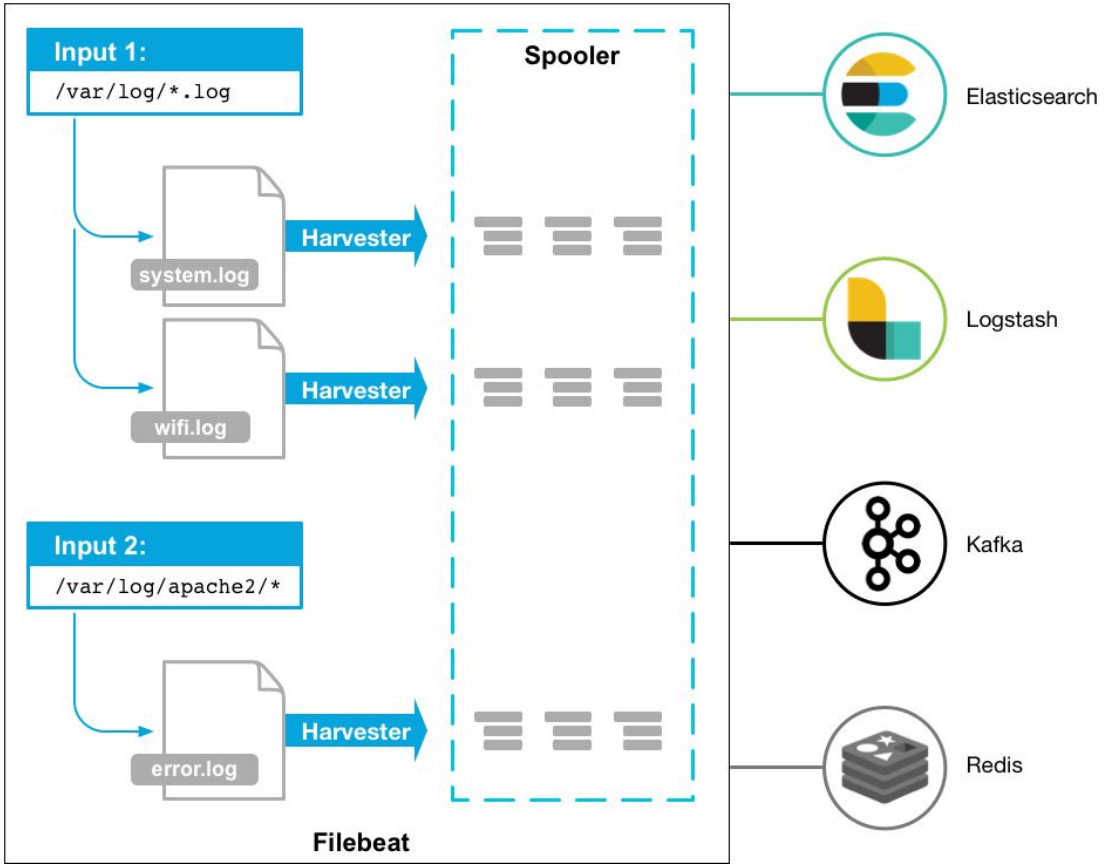
Logs centralization - rsyslog

- How gather data in one place?
  - Application creates a standard file in local file system with some log rotation (build-in or external (i.e. logrotate)),
  - Rsyslog sends logs from local to central node
  - Global logs in central



Logs centralization: ELK

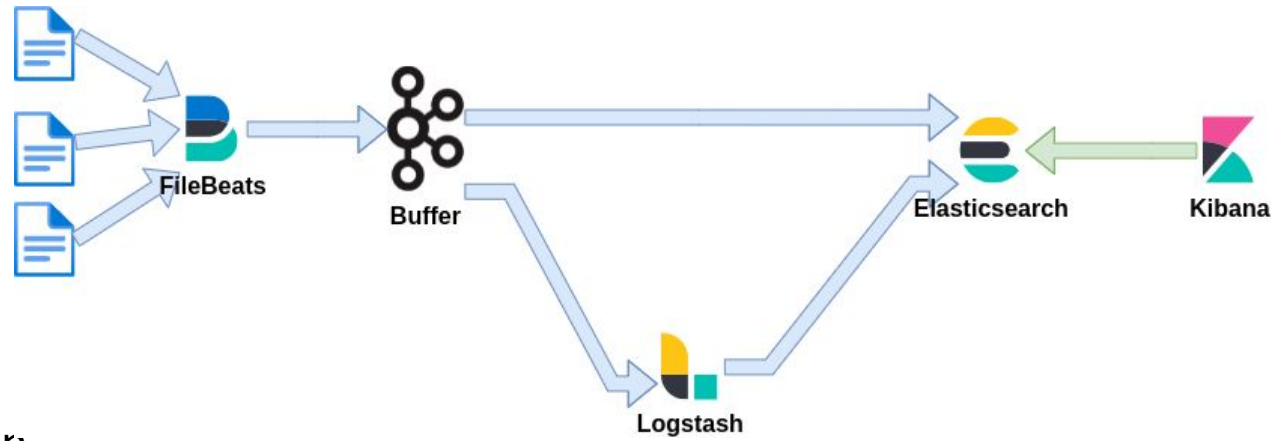
- ELK stack:
  - Beats
  - Logstash
  - Elasticsearch
  - Kibana



https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

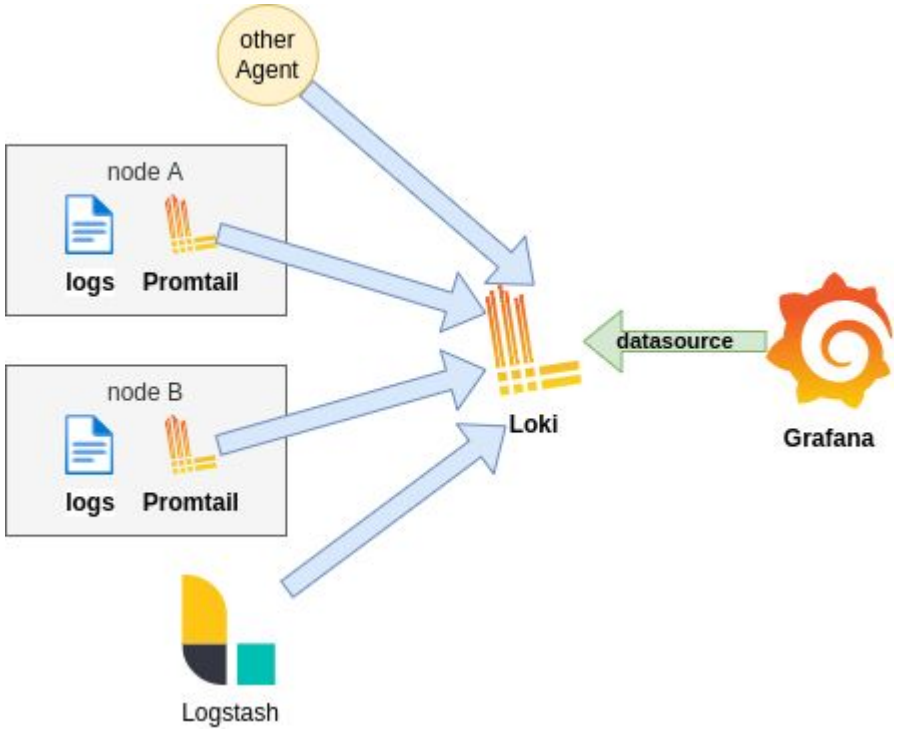
Logs centralization - ELK

- Read logs by FileBeats
- Optionally Buffered
- Optionally parse by Logstash
- Results saved in Elasticsearch
- Read and visualized in Kibana



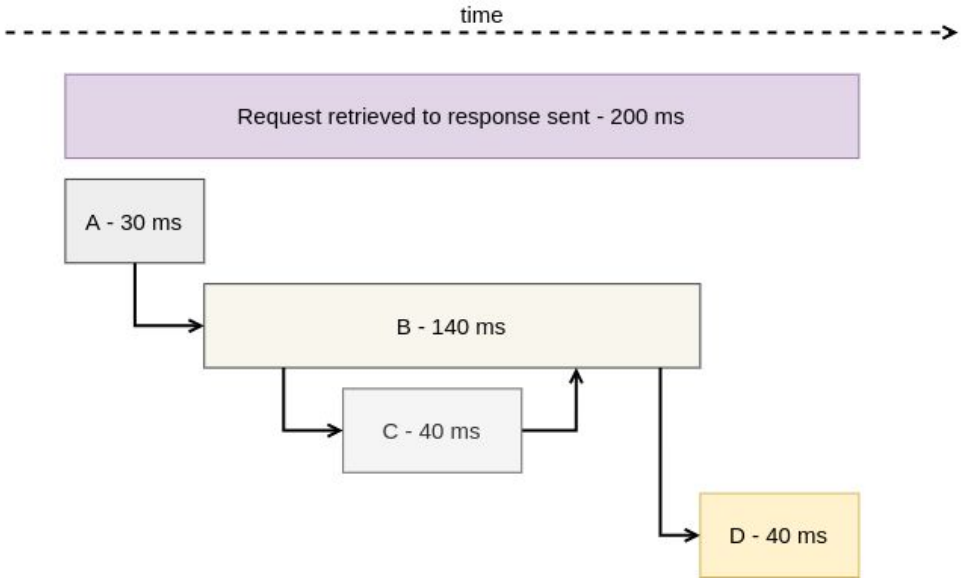
Logs centralization - Grafana Loki

- Promtail is an agent which ships the contents of local logs to a Grafana Loki instance
- Loki provides Push API
- Input:
  - Messages streams with labels
- Labels are indexed.
  Expected low cardinality



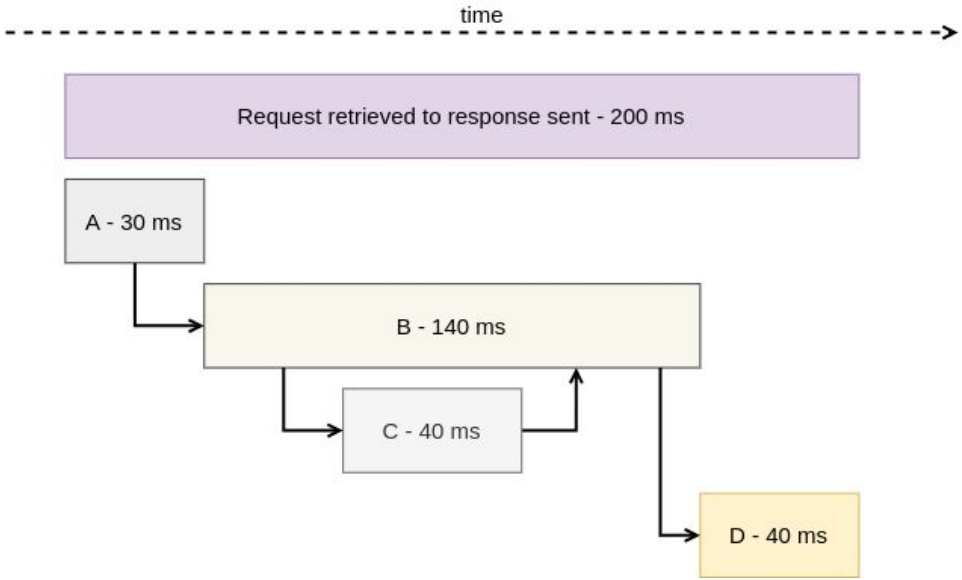
Tracing

- A trace is a representation of a series of causally related distributed events that encode the end-to-end request flow through a distributed system.



Tracing

- What we would like to trace?
  - End user request from start to end
  - Call function A (*span*)
  - Call remote service B, D (*distributed tracing*)
  - Asynchronous call function C
  - Connect all events in one trace.

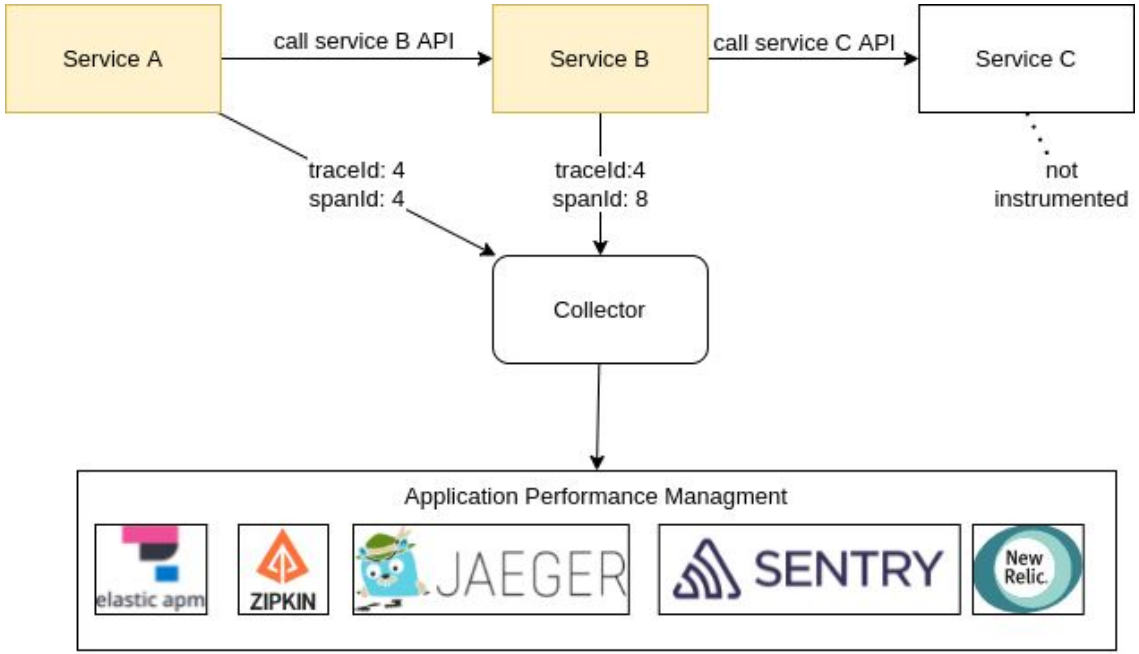


Tracing

- To fully track distributed services it is necessary to “instrument”:
  - Libraries
  - Frameworks
  - Dependent services
- Impact on performance should be measured - especially for high performance services
- It can be hard for legacy systems
- No all languages are supported

Tracing API

- OpenTelemetry as open, vendor-agnostic API and set of tools
- OpenTelemetry Collector:
  - Receives Tracing data
  - Process it
  - Export to External tools - most often APM

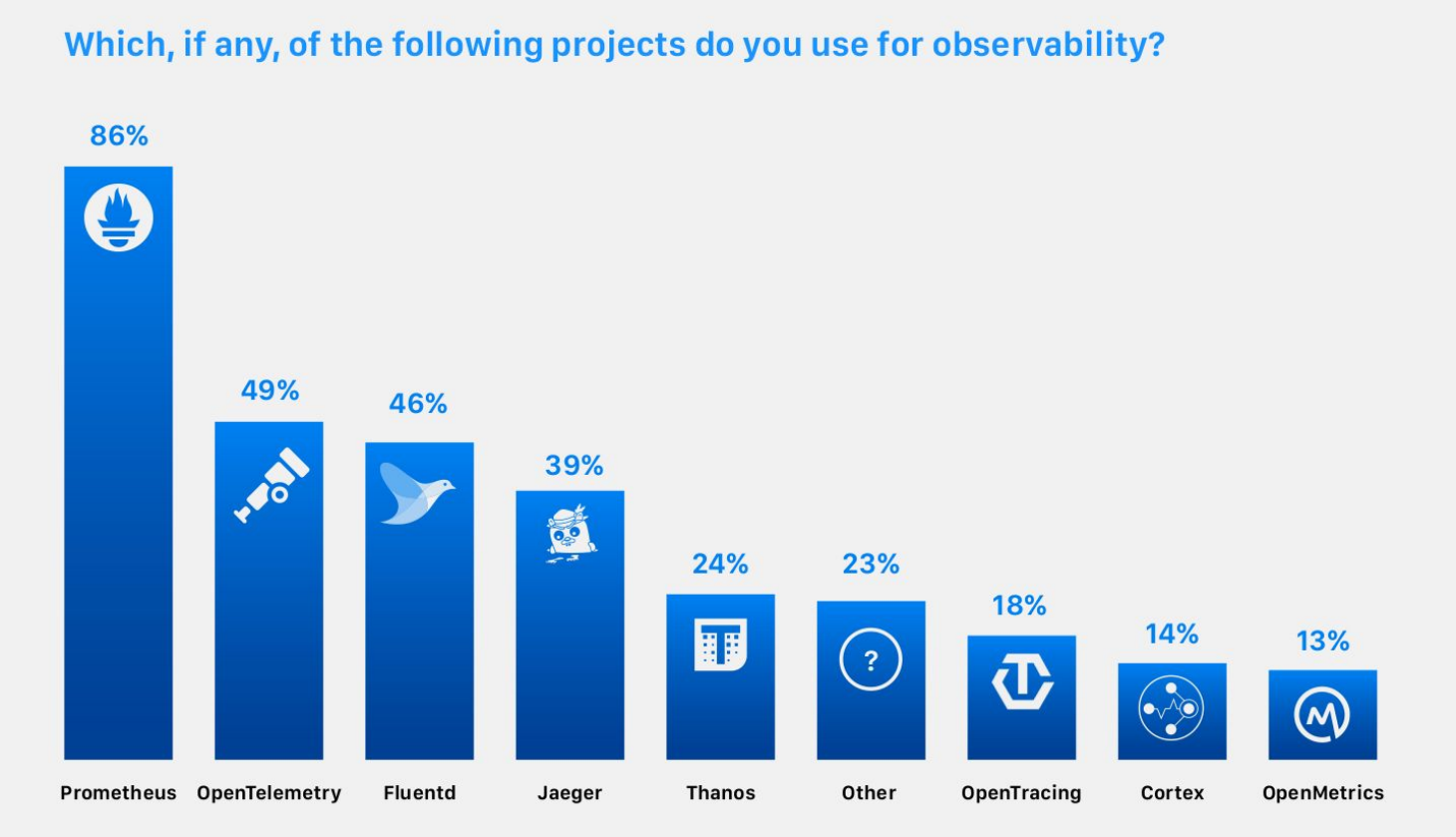


Postmortens

- After an outage occur, postmortens should be treated as learning opportunity
  - incident is documented
  - root cause(s) are well understood
  - reduce the likelihood and/or impact of recurrence
  - blameless: focus on identifying the contributing causes of the incident

https://sre.google/sre-book/postmortem-culture/

Summary



https://www.cncf.io/wp-content/uploads/2022/03/CNCF_Observability_MicroSurvey_030222.pdf

Links

- SRE Books: https://sre.google/books/
- Prometheus: https://prometheus.io/docs/introduction/overview/
- Graphite: https://graphite.readthedocs.io/en/latest/
- Distributed Systems Observability book: https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html
- Prometheus + Grafana demo https://grafana.demo.do.prometheus.io
- Elastic search demo:

  https://demo.elastic.co/