PRACTICES FOR MICROSERVICE MONITORING

Ioannis Sermetziadis
Back-end Software Engineer @ Numbrs

MONITORING DEFINITION

"Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes."

WHO?

back-end & operation engineers

WHEN?

deployment

WHY?

failure

reliability

SLAs

resource optimization

WHAT?

application metrics

platform metrics

system events

comparisons

top 10% and bottom 10%

Back-box vs white-box

system external behavior vs access to internal details

focus on symptoms vs follow symptom-cause chain to detect root cause

examine internal layers

access internal metrics from execution

The 4 Golden Signals

latency - distinguish between latency of successful and failed requests

traffic - broken down by the nature of requests (ready-only or mutating)

error - rate of failing

saturation - Memory, I/O constraints, CPU

Monitoring system simplicity rules

configurable alert thresholds

not everything is an alert

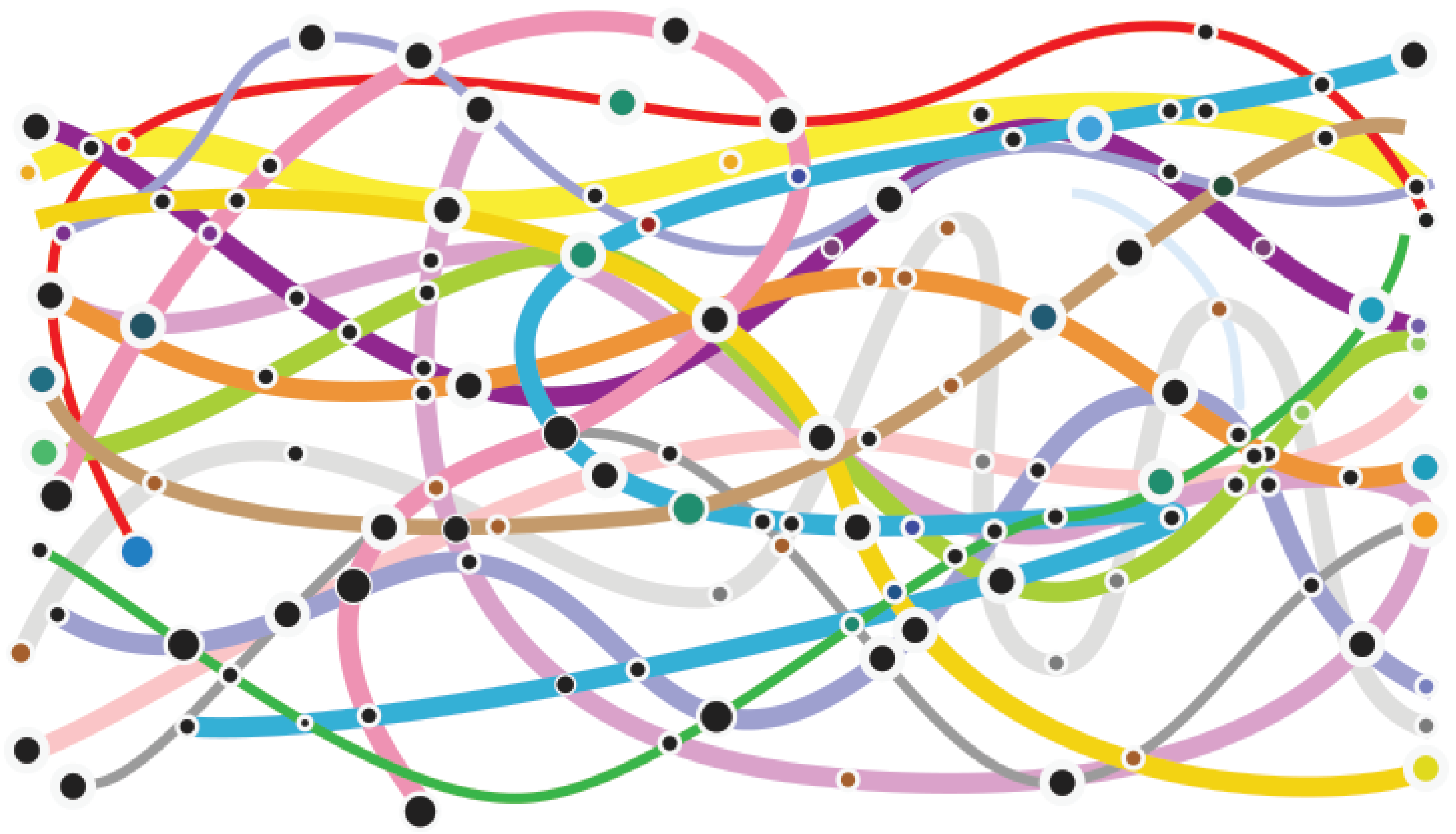
easily removable instrumentation code

identify and remove data not used in reports

MICROSERVICES DEFINITION

"Architectural style that structures an application as a collection of loosely coupled and fine-grained services."

MICROSERVICES HAZARDS


Integration Mess: Point-to-point connectivity

12factor principles

attachable backing services as loosely-coupled resources

state-less and state-full backing processes

disposable processes, fast start-up and shut-down, processing quick requests

logs as streams of events

Health APIs

identify issues with external resources, invalid configuration, unrecoverable state, application crash

liveliness & readiness

container orchestrators scaling & recovery automation

Logging

time-ordered raw data

log libraries, formatters, aggregators, processors/analyzers/alerts

developers are responsible to generate useful logs

instrumentation better applied as cross-cutting concern

AWS Cloudwatch, ELK, Apache Flume, Splunk

Distributed tracing

analyze latency problems

analogous to a performance profiler in monolithic apps

service dependency graphs

OpenTracing - a standardised API for request tracing that avoids vendor lock-in

OpenTracing / Jaeger

distributed context propagation

distributed transaction monitoring

root cause analysis

service dependency analysis

performance / latency monitoring

Application metrics

infrastructure-level metrics, such as CPU, memory, and disk utilization

application-level metrics, such as service request latency and number of requests executed

metrics services collect, aggregate, store, visualize

Prometheus, InfluxDB, Grafana

Exception tracking

catch error context

de-duplicate exceptions

generates alerts

manage the resolution of exceptions

Audit-logging

who performed what action

cross-cutting concern

out-of-the-box with event sourcing

Dark traffic - Teeing (pre-release)

rare incident debugging

test refactored services by comparison

services meshes, sidecars, traffic management, telemetry

Canary deploy (pre-release)

monitor new service versions

metrics comparison

internal traffic

percentage of external traffic

Production release

error rate inspection

decrease in overall traffic

increase in latency

proactive monitoring

Post-release

release of feature-flags

issues due to different workloads and traffic patterns

Chaos engineering

fault Injection

kill services

inject error conditions (latency, responses, events)

misbehaving network

identify hidden problems prior to any outage

Testing in Production, not Staging

staging cluster size not equals production

different configuration

lack of adequate monitoring

database execution plans depend on data volume

soak testing = performance tests verifying system's stability and performance characteristics over an extended period of time

no isolated artificial environment

no stubbing

Testing in Production - Caution

failing tests should not impact real users

eliminates the need for integration and staging environments

hit production backing resources

facilitate execution paths that depend on complex business conditions

3rd party dependencies mocking

avoid persisting test data or persist in name-spaced buckets

references

https://thenewstack.io/the-hows-whys-and-whats-of-monitoring-microservices/

https://landing.google.com/sre/sre-book/toc/index.html

https://12factor.net/istributed-tracing/

https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

https://medium.com/@copyconstruct/monitoring-in-the-time-of-cloud-native-c87c7a5bfa3e