Week 6

Logging, Monitoring and Resiliency

Group chat

https://www.messenger.com/t/2271159009676149

Agenda

- Observability techniques: logging & monitoring
- Resiliency techniques: retries & circuit-breaker
- Assignment: make a BE system more resilient

Logging

- Logs are lines of text containing info
- Logging is an action to keep record of events at software runtime

Logging usages

- Diagnostic: record events happening during software runtime
- Auditing: record abstract and business info

Logs attributes

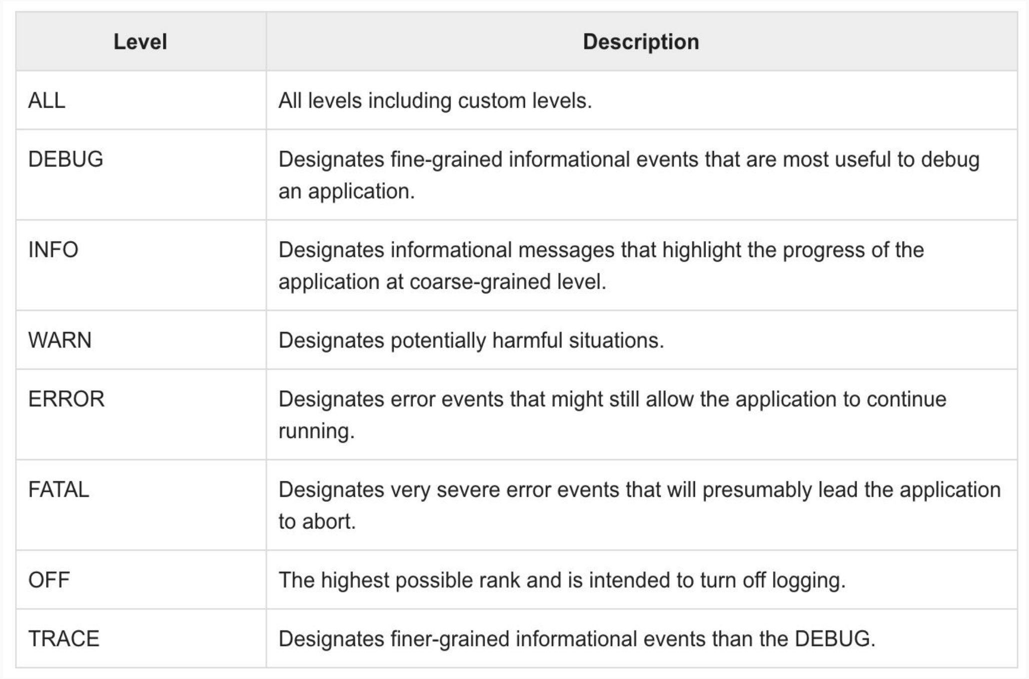
- Log format
- Levelled logging
- Log aggregation
- Causal ordering
- Log correlation

Logs attributes - Log Format

- Raw text, key-value based, structured and detailed
- More detailed structured logs are easy to read and query

Logs attributes - Levelled Logging

- Severities levels
- Visibility
- Usually statically initialized



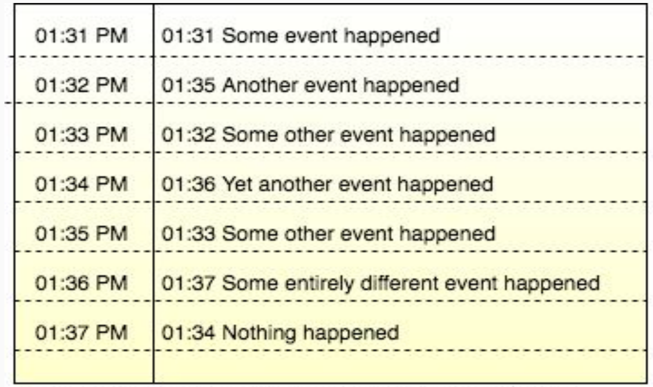
Level	Description
ALL	All levels including custom levels.
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.
WARN	Designates potentially harmful situations.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates very severe error events that will presumably lead the application to abort.
OFF	The highest possible rank and is intended to turn off logging.
TRACE	Designates finer-grained informational events than the DEBUG.

Logs attributes - Log Aggregation

- Collect, store, search and visualize
- Splunk, Logstash, Graylog2, ELK, etc

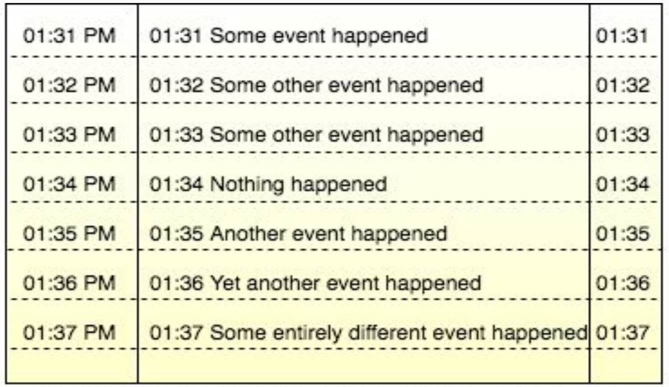
Logs attributes - Causal Ordering

- Generated vs written timestamps



01:31 PM	01:31 Some event happened
01:32 PM	01:35 Another event happened
01:33 PM	01:32 Some other event happened
01:34 PM	01:36 Yet another event happened
01:35 PM	01:33 Some other event happened
01:36 PM	01:37 Some entirely different event happened
01:37 PM	01:34 Nothing happened

Logs without causal ordering. Before y'all, the timestamps on logs were the timestamps when the logs were actually written to disk rather than when they were generated. This results in ambiguity in order, because the order in which they are processed might not always be the same as the order in which they are generated.



01:31 PM	01:31 Some event happened	01:31
01:32 PM	01:32 Some other event happened	01:32
01:33 PM	01:33 Some other event happened	01:33
01:34 PM	01:34 Nothing happened	01:34
01:35 PM	01:35 Another event happened	01:35
01:36 PM	01:36 Yet another event happened	01:36
01:37 PM	01:37 Some entirely different event happened	01:37

Logs with causal ordering. Now the logs have the timestamps which match the time in which they are generated, regardless of when they are written. This enables easier debugging, as now logs are in the exact order in which they are generated. An additionally field in the logs enables sorting the logs by the exact nanoseconds as well.

Logs attributes - Log Correlation

- Logs relevant to a particular request or event

Caveats

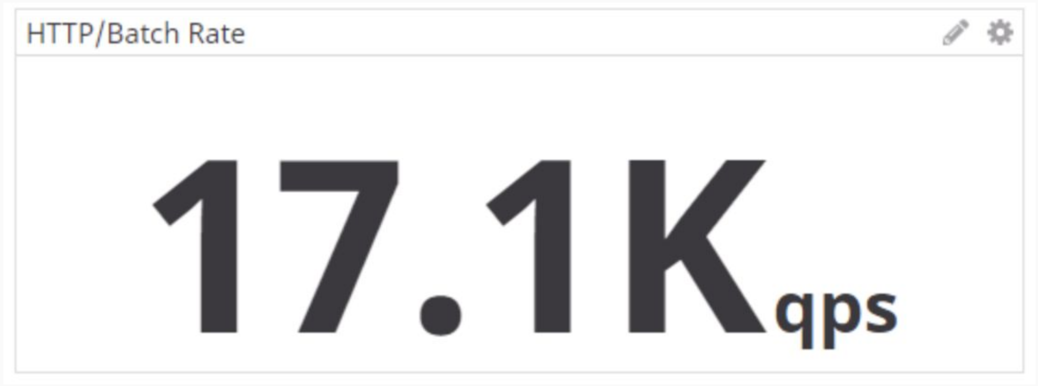
- Being readable and usable
- System performance
- Maintenance cost
- Actionable information

Instrumentation/ Monitoring

- System's diagnostic information
- Intentionally published by code or collected by agent
- 3 main primitives: counter, gauge and histogram

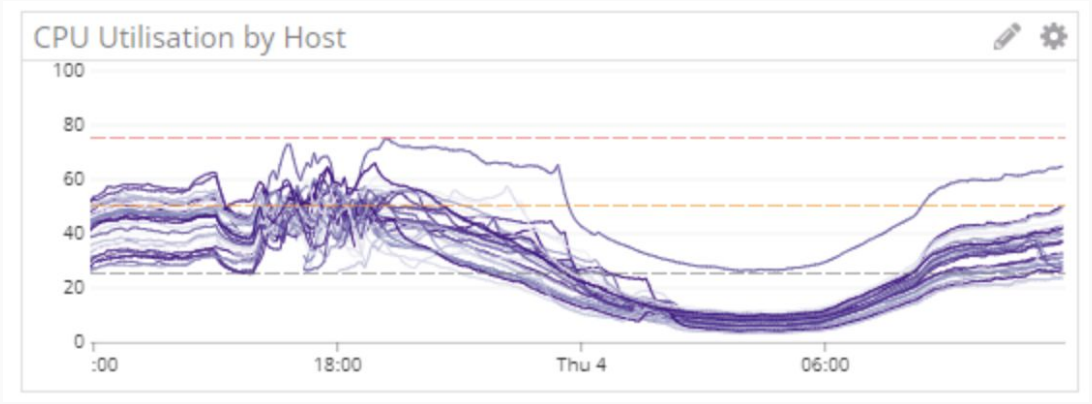
Instrumentation/ Monitoring - Counter

- Single monotonically increasing counter
- At a current time



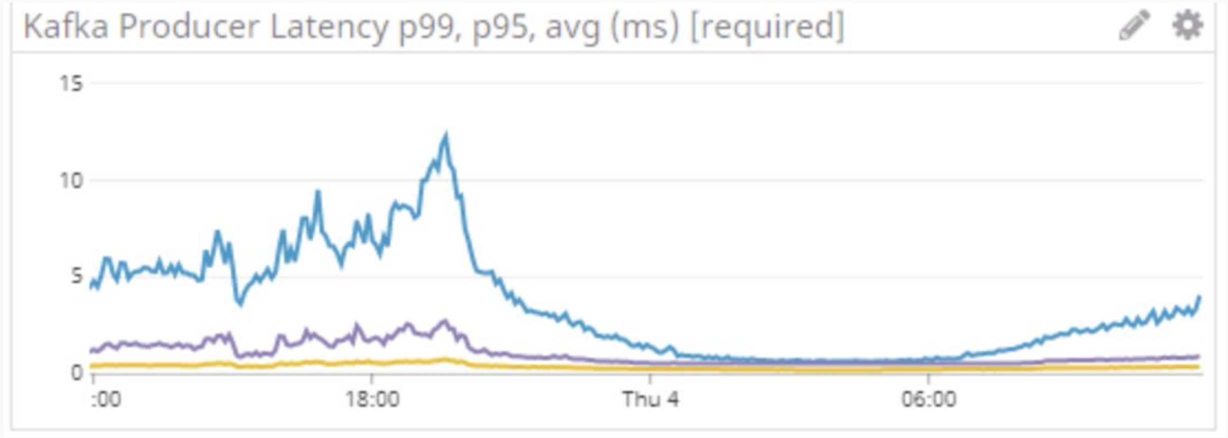
Instrumentation/ Monitoring - Gauge

- Numerical value can arbitrarily go up and down



Instrumentation/ Monitoring - Histogram

- Samples observations and counts them in configurable buckets
- Computing operations: quantile, average, min, max, count and sum

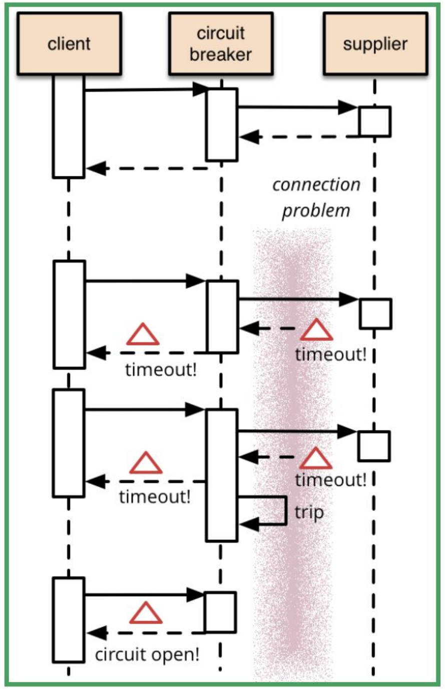


Resiliency

- Ability to still work when unexpected events happen
- Circuit breaker and retries

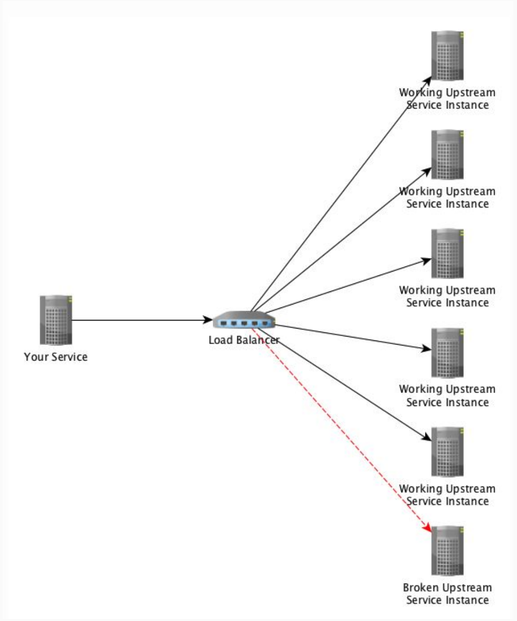
Resiliency - Circuit breaker

Wrap a function call with failure monitor, circuit breaker trips/opens when failure rate reach threshold



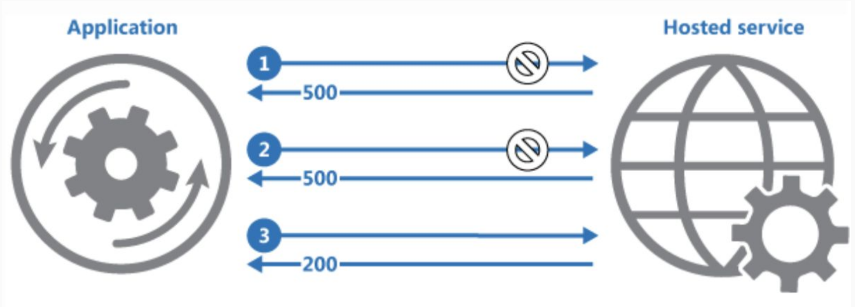
Resiliency - Circuit breaker

- Service level vs instance level


Working Upstream Service Instance
Working Upstream Service Instance
Working Upstream Service Instance
Your Service
Load Balancer
Working Upstream Service Instance
Working Upstream Service Instance
Broken Upstream Service Instance

Retries

- Self-correcting faults: network, timeout, etc
- Software mechanism that monitors, detects failure, repeats the request



1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

Retries

- Retries for all kinds of errors?
- Frequency?
- Retries vs circuit breakers?

Assignment

Logging, Instrumentation, and Resiliency - Assignment

Grab

Thanks