DevOps, Docker and Gitlab-CI

Part 4: Observability

Version 1.1.0 (2023-02-15)

-- Cyril zarak Duval, root CRI/ACU 2020

DevOps and observability

DevOps is also about visibility for everyone

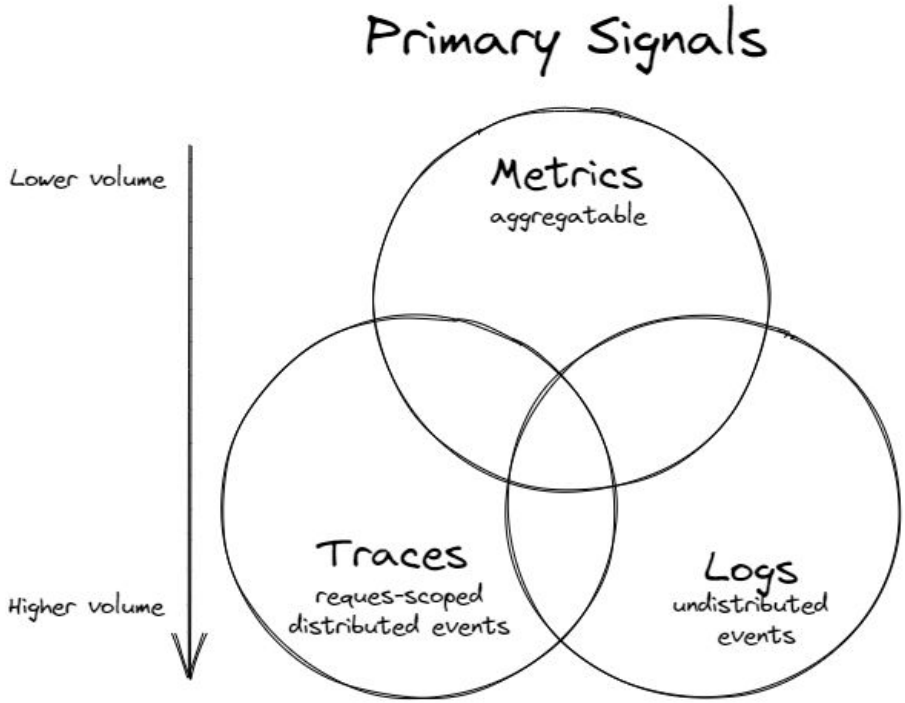
What is observability

- ➔ Once an application is in production, how does it behave ?
- ➔ Is it overloaded ?
- ➔ Is it working well ?
- ➔ Are there clients on it ?
- ➔ Are they facing errors ?
- ➔ Bugs ?
- ➔ If so, what kind ?
- ➔ How to investigate easily ?
- ➔ Shall we consider scaling up/down ?
- ➔ For an on-call ops, how to understand what's going on ?

What is observability

- A solution to all these questions are observability
- 3 pillars:
  - Monitoring
  - Logging
  - Tracing
- Ops shall provide platforms to receive these signals
- Dev shall provide such observability in their apps
  - And if applicable, documentation about the observability
    - And the actions to take if any

What is observability

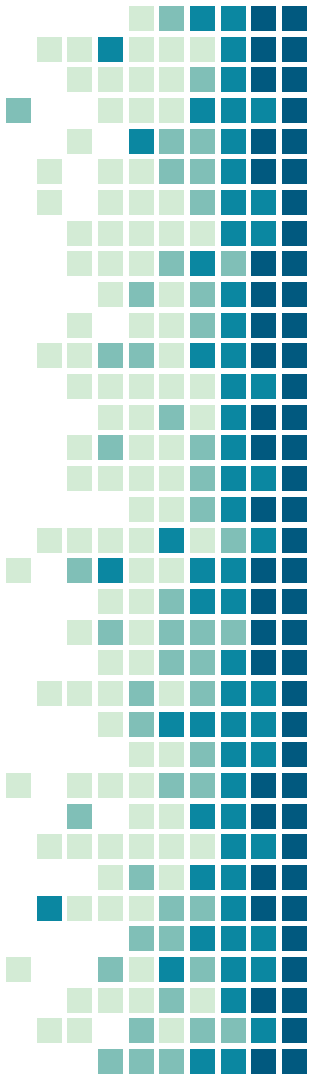


metrics

Who doesn’t love graphs, charts and histograms ?

Metrics

- Metrics is about exposing internal stats in a numerical form
- Metrics are meant to be aggregated
- Metrics are to be collected by an external tool
- Metrics are usually meant to be plotted
- 2 kinds of metrics:
  - Data already numeric
  - Data converted to be represented by numbers
- Metrics represent a state of a system at a given time
- Used to understand what is happening, not why



What is observability – metrics

- Most popular way of exposing metrics now:
  - Expose a HTTP route
    - Or HTTPS
    - /metrics
  - Prometheus format
- metric_name_unit{label1="value1", label2="value2"} value

What is observability – metrics

- What kind of metrics to expose ?
- total metrics
  - Number of requests handled in total
  - Number of file read in total
  - ….
- count/size/… metrics
  - Number of requests handled right now
  - Number of open files right now
  - Size of the event queue
  - …

What is observability - metrics

- What kind of metrics to expose ?
- seconds metrics
  - Amount of time taken to answer a request
  - Time taken writing data to cache
  - ....
- metadata metrics
  - Version of the running app
  - Running architecture
  - ...

What is observability - metrics

- Metrics are key to see what's going on
- We plot graph and we can visualize
- High level metrics (KPI) and low level
- Used for alerting
  - Ex: sudden drop of connected users
- Used for reporting
  - Ex: increasing amount of time taken to handle a request after an update
  - Ex: average user document size increasing over months

logs

You logged things without even knowing it

What is observability – logs

- ➔ Logs are the most useful indication to understand what is going on in the app in details, with description
  - ◆ They don't provide any global overview though
- ➔ Useful to get information about:
  - ◆ Understanding what's going wrong
  - ◆ Which client/route/component is:
    - Used
    - Not working
    - Hammered
  - ◆ What is the app doing

What is observability – logs

- Logs can hold a lot of value
  - Even legal one, mind the GDPR for example
- 2 schools of thoughts about providing logs:
  - stdout/stderr (pull model)
  - syslog/elastic/… client (push model)
- **Logs must be structured**
  - syslog format
  - JSON
  - Homemade but consistent

What is observability – logs

- Why should logs be structured ?
- Useful to search for specific things
  - Logs will often be put in Loki, Elastic, …
  - They provide query languages
    - Ex: {component="auth", severity="error"}
    - Ex: client_id: 10 AND route: "/login"
- Having a structure (and a consistent and documented one) is important
- GiB of logs to be generated: not read manually

What is observability – logs

- Logs shall have a severity level:
  - DEBUG
  - INFO
  - WARNING
  - ERROR
- Severity level must be configurable
- The amount of logs generated must be chosen carefully
- For DEBUG, don't care
- Starting from INFO, one must be wise
- Use a logging library

What is observability – logs

- ➔ What kind of logs can we have ?
- ➔ One think of application logging first
  - ◆ What is my application doing
  - ◆ Examples:
    - INFO User making a query on /api/v1/tickets
    - WARNING Cache is full. Dropping half its content
    - ERROR Exception uncaught while handling query
    - DEBUG read 17 bytes from file /var/run/myapp/fEcUs.tmp
    - FATAL Could not write data: disk is full

What is observability - logs

- What kind of logs can we have ?
- Security logs
  - Admin user changed its password
  - Deletion of xxx
  - New device logged-in
- Audit, system and infra logs are used for ops

tracing

Follow the trail of events in details

What is observability – tracing

- Logs are great to see some generic information about ongoing operation in the application
- They are not focused on a single user request
  - Single transaction
- Traces are meant to cover this case

What is observability - tracing

- What if you application throw an error while handling a client query ?
- You don't want the whole app to crash for most cases
- Just return an error to the client
- You also need the error to be reported to you to fix it
- Logs ?
  - Stacktrace are multi-lines
  - They have their own context
  - Put in the logs some concise information usually
    - ex: "Can't connect to DB"
    - ex: "Can't find <...> for <...> via <...>"

What is observability – tracing

- Send the whole stacktrace and its context to another service
- Error tracking service
- Example: Sentry
- Regroup similar errors and plot their occurence
- Integrated with Gitlab to report bugs and regression
- Provide context
  - Browser used, account id, … if useful
  - Crumbs
  - Runtime data
  - Alerts

What is observability – tracing

- ➔ Tracing also covers multi spans transactions for microservices architectures
- ➔ A trace is created for each transaction
  - ◆ Very high granularity
  - ◆ Often downsampled
- ➔ Used to understand interactions between services
  - ◆ Is the database slow ? microservice A, B or C ?
- ➔ A bit more difficult to put in place than logs & metrics
  - ◆ More specialized
- ➔ Check OpenTelemetry, Sentry, ZipKin, Tempo, …

A connected element of observability: monitoring

Collect, store, observe, alert, report

Monitoring

- Monitoring is a subcategory of observability
  - Or another concept but tightly linked to observability
- Monitoring is about 5 points:
  - Collect
  - Store
  - Visualize
  - Alert
  - Report

Monitoring - Collecting

- The collecting component of a monitoring solution is in charge of getting the data
- Pull/Push
- Metrics-oriented mostly
- Examples:
  - Netdata
  - Node exporter/Prometheus
  - Metricbeats
  - Telegraf

Monitoring - Storing

- Metrics are representation of things happening at a given time
- Need to store them to have trends, history and evolution
- The volumetry can be quite high
  - Depends on the resolution (1s, 15s, 30s, 1m, ...)
  - Depends on the amount of metrics
    - Cardinality
- Use dedicated databases for this (often Time Serie DataBases)
- Example: Prometheus, Netdata, InfluxDB

Monitoring - Visualizing

- Once collected over time, metrics are best represented with graphs (visualization in general)
- Need to have real-time dashboards
- Graphs, histograms, plots, ...
- Focus on the UI/UX
- Example:
  - Grafana
  - Netdata
  - Kibana
  - Chronograf

Monitoring - Reporting

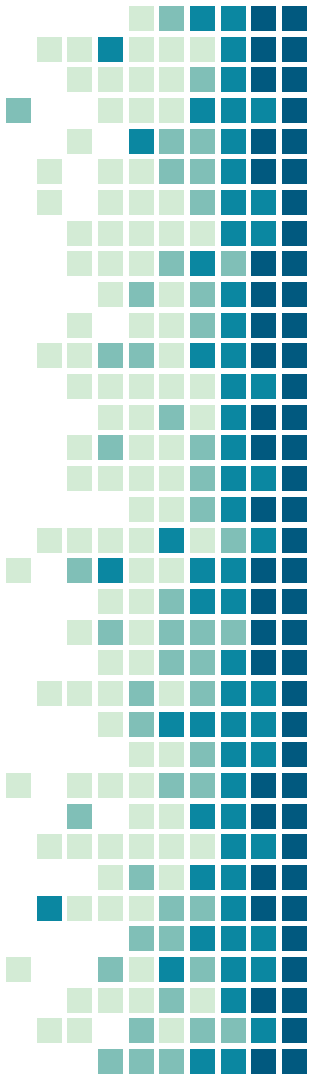
- ➔ Reporting is most of the time forgotten and less popular
- ➔ Report once in a while (weekly, monthly, …) what happened, and trends
- ➔ Create a report (i.e. PDF file) with visualizations
- ➔ Used to be aware of non-immediate situation
- ➔ Often mangled with the visualization & alerting component

Monitoring - Alerting

- Observability & Monitoring are useful to see and try to understand what's going on with your app/infra
- Used for post-mortems
  - Evidence of the issue
  - Provide tools for RCA to try to determine the RCE
- If one can understand through metrics an issue that happened, why not alert when it happens ?
  - Or even before if we can

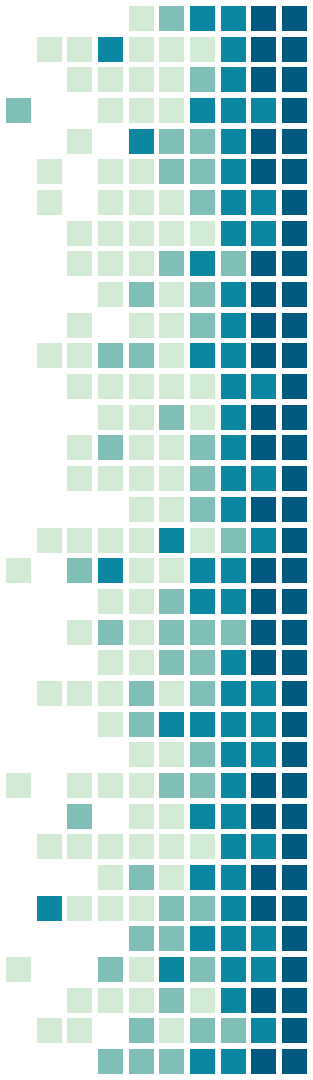
Monitoring - Alerting

- ➔ Alerting should be done first on high-level metrics
  - ◆ Number of clients
  - ◆ Number of videos being watched
  - ◆ Number of emails sent
  - ◆ Latency increasing
- ➔ Alerting can be done on low-level metrics with care
  - ◆ If high CPU but no impact on the client, is it an alert ?
  - ◆ 10% of disk left, is it the same if 1GiB left of 1TiB ?



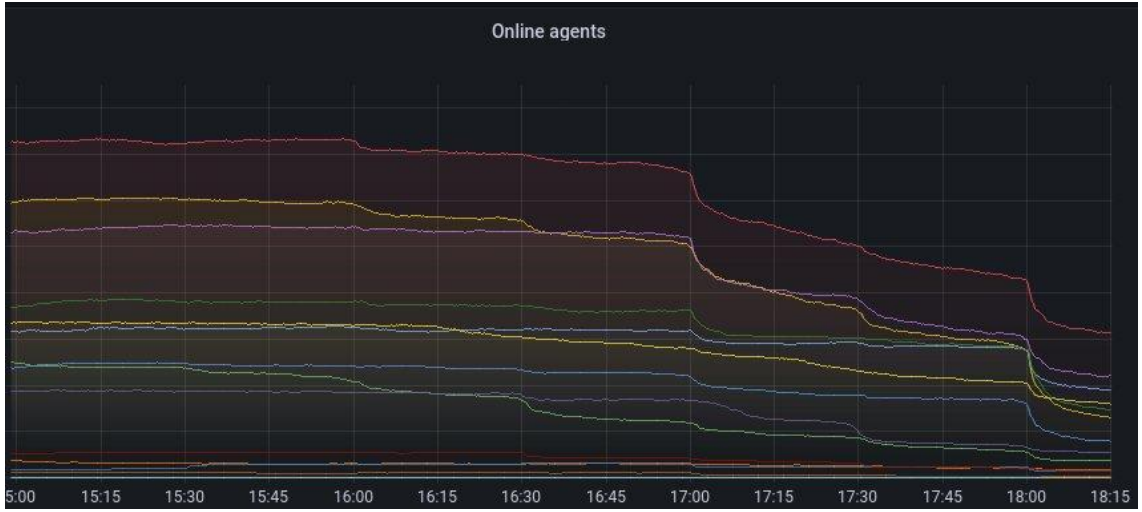
Monitoring - Alerting

- Alerts should have multiple level of criticality
  - Think about whether to wake up an ops or not for example
  - Lowest level(s) can even be moved to reporting
- Too many alerts = alert fatigue
- Too many false positives = more chances to miss an important one



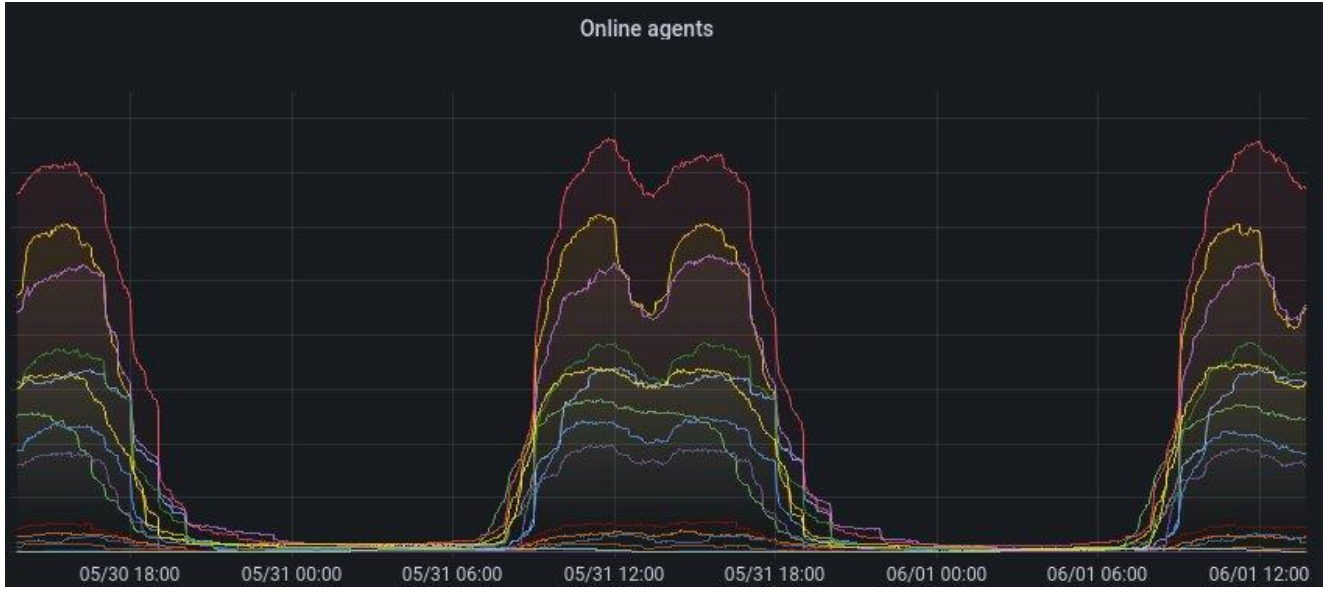
Monitoring – Alerting

➔ Be extra careful with alerting



Monitoring – Alerting

➔ Be extra careful with alerting



Monitoring – Alerting

- ➔ How to do proper alerting ?
- ➔ Many ask the question and few found the answer
- ➔ Appears to always be a balance between too many and too few alerts

Monitoring - Alerting

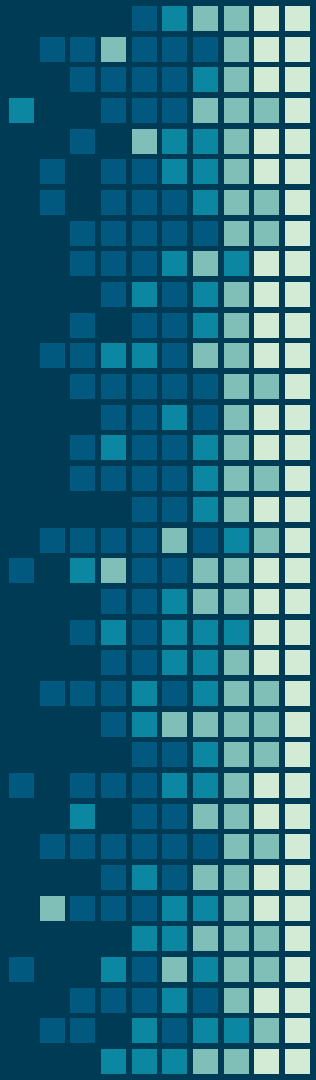
- Alerts should be derived from SLIs, taking into account the SLOs and sold SLA
  - Service Level Indicators - A quantifiable indicator of the level of service provided (often mistaken for KPIs)
  - Service Level Objectives - An objective set on a SLI about how much a SLI can fail
  - Service Level Agreement - What has been sold to the client in terms of disponibility

Monitoring - Alerting

- SLA is 100% - SLO, represents the % of availability
  - Also expressed in a number of 9
    - Three 9s means 99.9% of availability
  - Also expressed in allowed failure time per period of time
- 99.9% of SLA = 8.7h/y, 44min/m
- 99.99% = 52min/y, 4.3min/m
- 99.999% = 5.2min/y, 26.3s/m
- Alerts can use this budget of allowed failure to avoid false positives by working on the burn rate

Thanks !

Questions ?



Slides available on zarak.fr/

Contact: cyril@cri.epita.fr

zarak production#5492