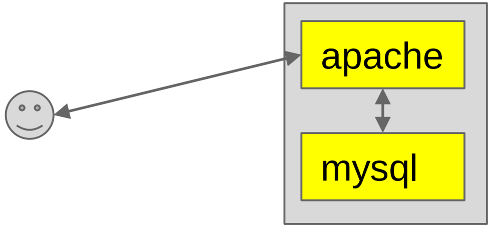
Troubleshooting complex systems

Today's best engineering practice and AI RCA challenges

bjeunhomme@gmail.com July 7, 2022

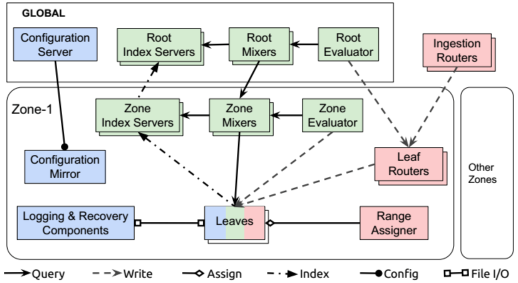
Complex systems?

Simple system



- Single host, single cluster
- 2 components
- 2-3 important log files
- 1 single query path
- Everything usually works

Modern cloud system



Google monarch overview
Source: https://research.google/pubs/pub50652/

- Multiple clusters, multiple hosts per cluster
- Several components per cluster
- Countless log files
- Numerous, everchanging query paths
- Brokenness is the norm, not the exception

The troubleshooting challenge

Traditional approach: reading logs

- Let’s take a not so complex example system
- 3 clusters, 4 components per cluster running on 5 hosts per component
- That’s 3 x 4 x 5 = 60 key log files already
- If each host writes only 100 log lines per second, it’s 6000 lines per second

The troubleshooting challenge

Traditional approach: reading logs

- Let’s take a not so complex example system
- 3 clusters, 4 components per cluster running on 5 hosts per component
- That’s 3 x 4 x 5 = 60 key log files already
- If each host writes only 100 log lines per second, it’s 6000 lines per second

Two options to handle this information flood:

1. Automate log processing (ad-hoc or AI based)
2. Summarize

What the industry leaders do differently

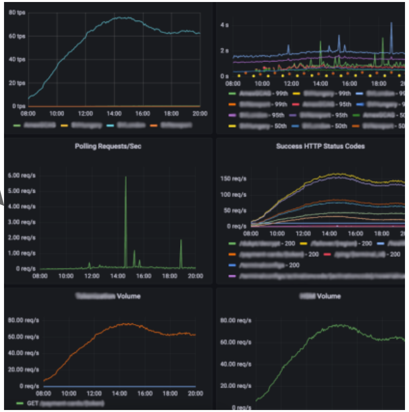
They understand this

- Automating logs processing doesn’t work:
    - When brokenness is the norm, reporting all anomalies is just noise
    - Ad-hoc log processing automation is laborious and brittle
    - Processing logs with AI effectively is still research today

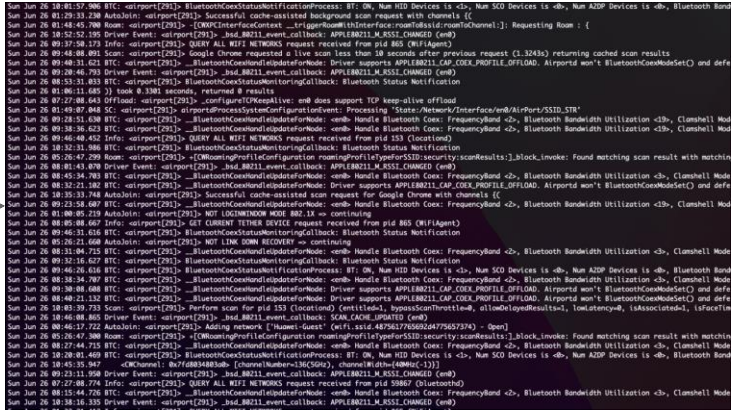
They understand this

- Automating logs processing doesn’t work:
  - When brokenness is the norm, reporting all anomalies is just noise
  - Ad-hoc log processing automation is laborious and brittle
  - Processing logs with AI effectively is still research today
- Summarizing is simple and effective for troubleshooting

They look at this

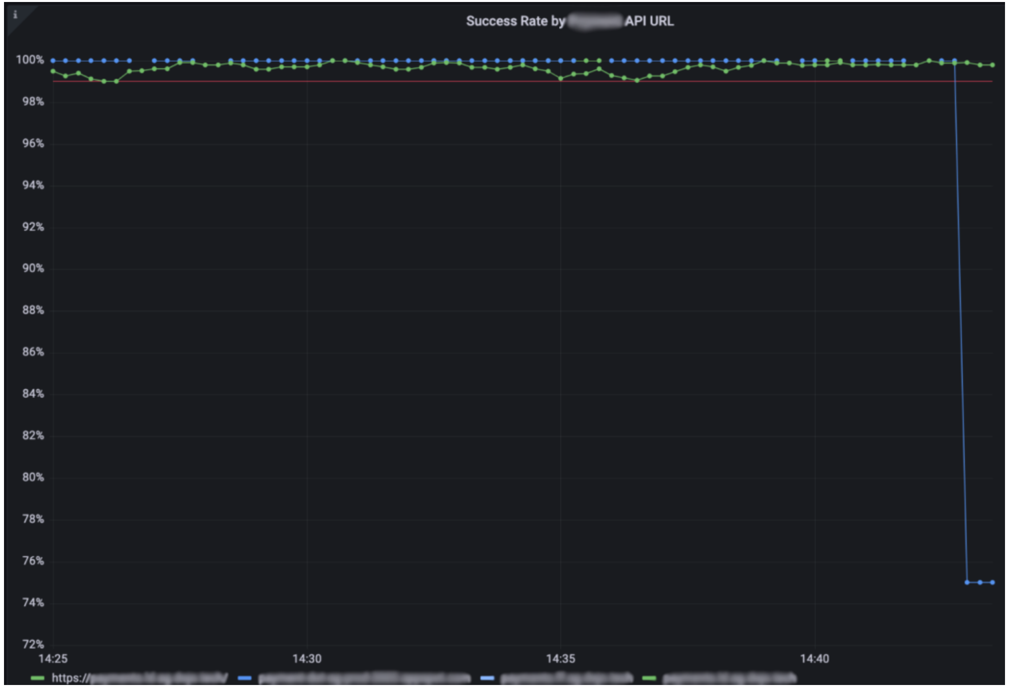


Not at that



```
Sun Jun 26 10:01:57.906 BTC: <airport[291]> BluetoothCoexStatusNotificationProcess: BT: ON, Num HID Devices is <1>, Num SCO Devices is <0>, Num A2DP Devices is <0>, Bluetooth Band
Sun Jun 26 01:29:33.230 AutoJoin: <airport[291]> Successful cache-assisted background scan request with channels {(
Sun Jun 26 01:48:45.700 Roam: <airport[291]> -[CWXPCInterfaceContext __triggerRoamWithInterface:roamToBssid:roamToChannel:]: Requesting Roam : {
Sun Jun 26 10:52:52.195 Driver Event: <airport[291]> _bsd_80211_event_callback: APPLE80211_M_RSSI_CHANGED (en0)
Sun Jun 26 09:37:50.173 Info: <airport[291]> QUERY ALL WIFI NETWORKS request received from pid 865 (WiFiAgent)
Sun Jun 26 09:48:08.091 Scan: <airport[291]> Google Chrome requested a live scan less than 10 seconds after previous request (1.3243s) returning cached scan results
Sun Jun 26 09:40:31.621 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: Driver supports APPLE80211_CAP_COEX_PROFILE_OFFLOAD. Airportd won't BluetoothCoexModeSet() and defe
Sun Jun 26 09:20:46.793 Driver Event: <airport[291]> _bsd_80211_event_callback: APPLE80211_M_RSSI_CHANGED (en0)
Sun Jun 26 08:53:31.033 BTC: <airport[291]> BluetoothCoexStatusMonitoringCallback: Bluetooth Status Notification
Sun Jun 26 01:06:11.685 )} took 0.3301 seconds, returned 0 results
Sun Jun 26 07:27:08.643 Offload: <airport[291]> _configureTCPKeepAlive: en0 does support TCP keep-alive offload
Sun Jun 26 01:49:07.048 SC: <airport[291]> airportdProcessSystemConfigurationEvent: Processing 'State:/Network/Interface/en0/AirPort/SSID_STR'
Sun Jun 26 09:28:51.630 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <19>, Clamshell Mod
Sun Jun 26 09:38:36.623 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <19>, Clamshell Mod
Sun Jun 26 09:46:40.452 Info: <airport[291]> QUERY ALL WIFI NETWORKS request received from pid 153 (locationd)
Sun Jun 26 10:32:31.986 BTC: <airport[291]> BluetoothCoexStatusMonitoringCallback: Bluetooth Status Notification
Sun Jun 26 05:26:47.299 Roam: <airport[291]> +[CWRoamingProfileConfiguration roamingProfileTypeForSSID:security:scanResults:]_block_invoke: Found matching scan result with matchin
Sun Jun 26 08:01:43.070 Driver Event: <airport[291]> _bsd_80211_event_callback: APPLE80211_M_RSSI_CHANGED (en0)
Sun Jun 26 08:45:34.703 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <3>, Clamshell Mode
Sun Jun 26 08:32:21.102 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: Driver supports APPLE80211_CAP_COEX_PROFILE_OFFLOAD. Airportd won't BluetoothCoexModeSet() and defe
Sun Jun 26 10:35:33.748 AutoJoin: <airport[291]> Successful cache-assisted scan request for Google Chrome with channels {(
Sun Jun 26 09:23:58.607 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <19>, Clamshell Mod
Sun Jun 26 01:00:05.219 AutoJoin: <airport[291]> NOT LOGINWINDOW MODE 802.1X => continuing
Sun Jun 26 08:05:08.667 Info: <airport[291]> GET CURRENT TETHER DEVICE request received from pid 865 (WiFiAgent)
Sun Jun 26 09:46:31.616 BTC: <airport[291]> BluetoothCoexStatusMonitoringCallback: Bluetooth Status Notification
Sun Jun 26 05:26:21.660 AutoJoin: <airport[291]> NOT LINK DOWN RECOVERY => continuing
Sun Jun 26 08:31:04.715 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <3>, Clamshell Mode
Sun Jun 26 09:32:16.627 BTC: <airport[291]> BluetoothCoexStatusMonitoringCallback: Bluetooth Status Notification
Sun Jun 26 09:46:26.616 BTC: <airport[291]> BluetoothCoexStatusNotificationProcess: BT: ON, Num HID Devices is <1>, Num SCO Devices is <0>, Num A2DP Devices is <0>, Bluetooth Band
Sun Jun 26 08:38:34.707 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <3>, Clamshell Mode
Sun Jun 26 09:30:08.608 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: Driver supports APPLE80211_CAP_COEX_PROFILE_OFFLOAD. Airportd won't BluetoothCoexModeSet() and defe
Sun Jun 26 08:40:21.132 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: Driver supports APPLE80211_CAP_COEX_PROFILE_OFFLOAD. Airportd won't BluetoothCoexModeSet() and defe
Sun Jun 26 10:03:39.733 Scan: <airport[291]> Perform scan for pid 153 (locationd) (entitled=1, bypassScanThrottle=0, allowDelayedResults=1, lowLatency=0, isAssociated=1, isFaceTim
Sun Jun 26 10:46:08.865 Driver Event: <airport[291]> _bsd_80211_event_callback: SCAN_CACHE_UPDATED (en0)
Sun Jun 26 00:46:17.722 AutoJoin: <airport[291]> Adding network ['Huawei-Guest' (wifi.ssid.4875617765692d4775657374) - Open]
Sun Jun 26 05:26:47.300 Roam: <airport[291]> +[CWRoamingProfileConfiguration roamingProfileTypeForSSID:security:scanResults:]_block_invoke: Found matching scan result with matchin
Sun Jun 26 08:27:44.715 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <3>, Clamshell Mode
Sun Jun 26 10:20:01.469 BTC: <airport[291]> BluetoothCoexStatusNotificationProcess: BT: ON, Num HID Devices is <1>, Num SCO Devices is <0>, Num A2DP Devices is <0>, Bluetooth Band
Sun Jun 26 10:45:35.947      <CWChannel: 0x7fd8034803a0> [channelNumber=136(5GHz), channelWidth={40MHz(-1)}]
Sun Jun 26 09:23:11.950 Driver Event: <airport[291]> _bsd_80211_event_callback: APPLE80211_M_RSSI_CHANGED (en0)
Sun Jun 26 07:27:08.774 Info: <airport[291]> QUERY ALL WIFI NETWORKS request received from pid 59867 (bluetoothd)
Sun Jun 26 08:15:44.726 BTC: <airport[291]> __BluetoothCoexHandleUpdateForNode: <en0> Handle Bluetooth Coex: FrequencyBand <2>, Bluetooth Bandwidth Utilization <3>, Clamshell Mode
Sun Jun 26 10:38:16.335 Driver Event: <airport[291]> _bsd_80211_event_callback: APPLE80211_M_RSSI_CHANGED (en0)
```

Practical example: a real outage



How to implement it

Infrastructure must have to succeed

A powerful TSDB and graphing engine is unavoidable

- They all did it: Google -> monarch, Facebook -> Gorilla/Beringei, Uber -> M3…
- Example: Gorilla requirements in 2013 (source: Facebook Gorilla paper)
  - 2 billion unique time series identified by a string key
  - 700 million data points (time stamp and value) added per minute
  - More than 40,000 queries per second at peak
  - Support time series with 15 second granularity (4 points per minute per time series)
  - [...]
  - Support at least 2x growth per year

But, do we need so much engineering effort?

It depends.

- For small needs: free solutions such as prometheus/grafana, influxdb etc. Caveats: scalability and O&M
- Medium scale: several off the shelf solutions in the industry, but be selective!
  - $ per timeseries varies a lot between vendors (>10x differences)
  - Powerful aggregations, in particular percentiles over different timeseries are a must
  - Query language simplicity and power are crucial, and few vendors get it right
- High scale:
  - In house will be expensive (dozens of engineers) but still much cheaper than buying
  - Some components can be reused: M3 and Beringei are opensource
  - Not a good place to cut corners: do it right, or buy it from someone who did

Criteria for a good solution

Must have:

- Low $ per timeseries
- Aggregations and joins: can it do this?
    - Plot the 95th percentile of query latency over all my HTTP frontends, per cluster
    - One curve per cluster, **without typing the list of clusters** (discover it automatically)
    - Plot Σ(queries by status code) per second / Σ(queries) per second without typing a list of codes
- How complex do the queries look, to do the above? It must be 2-3 lines
- Resolution of 1 point every 15 seconds, or even 1 per second for network gear
- Support at least 20 labels per timeseries
- Notify about, and ideally autoblock, timeseries with excessive cardinality
- If high scale, ingest billions or trillions of timeseries simultaneously

Instrumentation effort

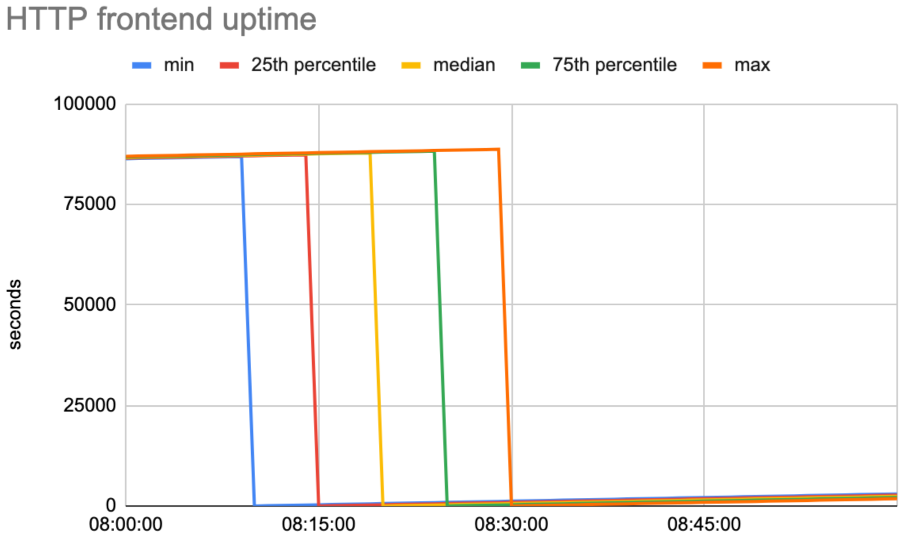
- Applications need to be instrumented
- Adding a metric isn't more effort than adding a log line. Java example:

```
static final Counter requests =
    Counter.build().name("requests").help("Requests count.").register();
[...]
    requests.inc();
```

- Shortcuts:
  - Create instrumented libs for communication (RPC, REST, kafka etc) and reuse them everywhere
  - Istio, dapr.io and friends: sidecars can help, but come at an efficiency cost
  - Deploy everywhere an agent for system metrics
- Best practices:
  - Think about relevant metrics at design time
  - It isn't about quantity of metrics and graphs, it's about quality - use a few, well thought out graphs
  - Standardize labels and build generic graphs once that everyone can use

Example generic graphs that can be built just once

Example graph 1: uptime

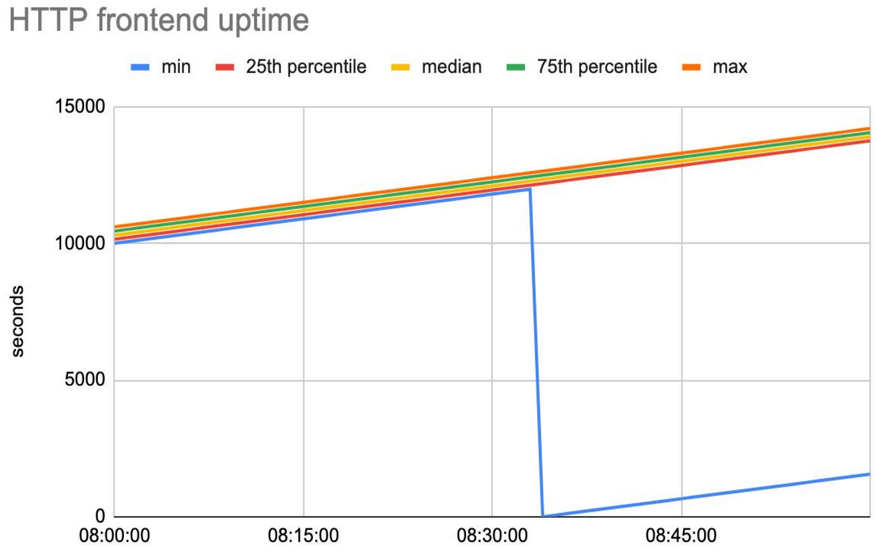


Slow restart from 08:10 to 08:30
Likely due to a gradual rollout

It answers several key questions at once:

- Was the latest release a long time ago?
- Are all the workers crashlooping?
- Did a worker crash or restart recently?

Example graph 1: uptime

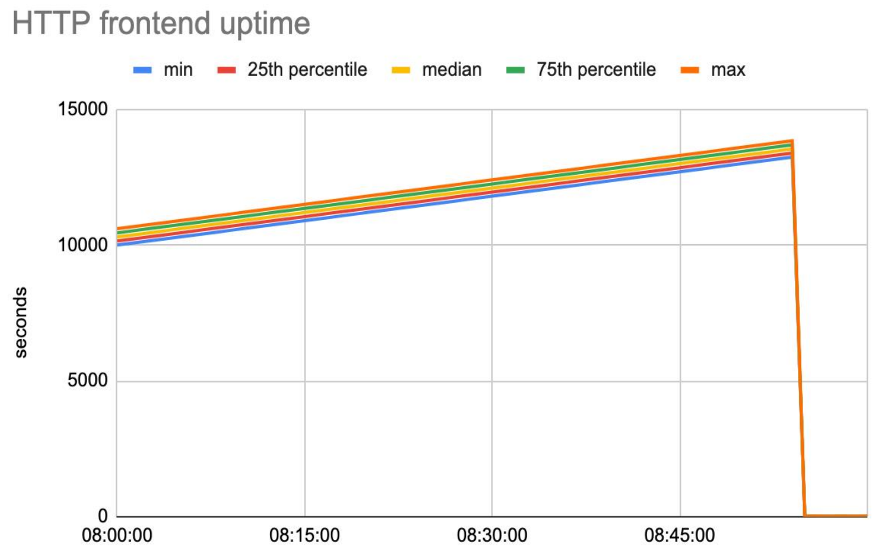


One (or a few) worker(s) restarted at 08:34

It answers several key questions at once:

- Was the latest release a long time ago?
- Are all the workers crashlooping?
- Did a worker crash or restart recently?

Example graph 1: uptime

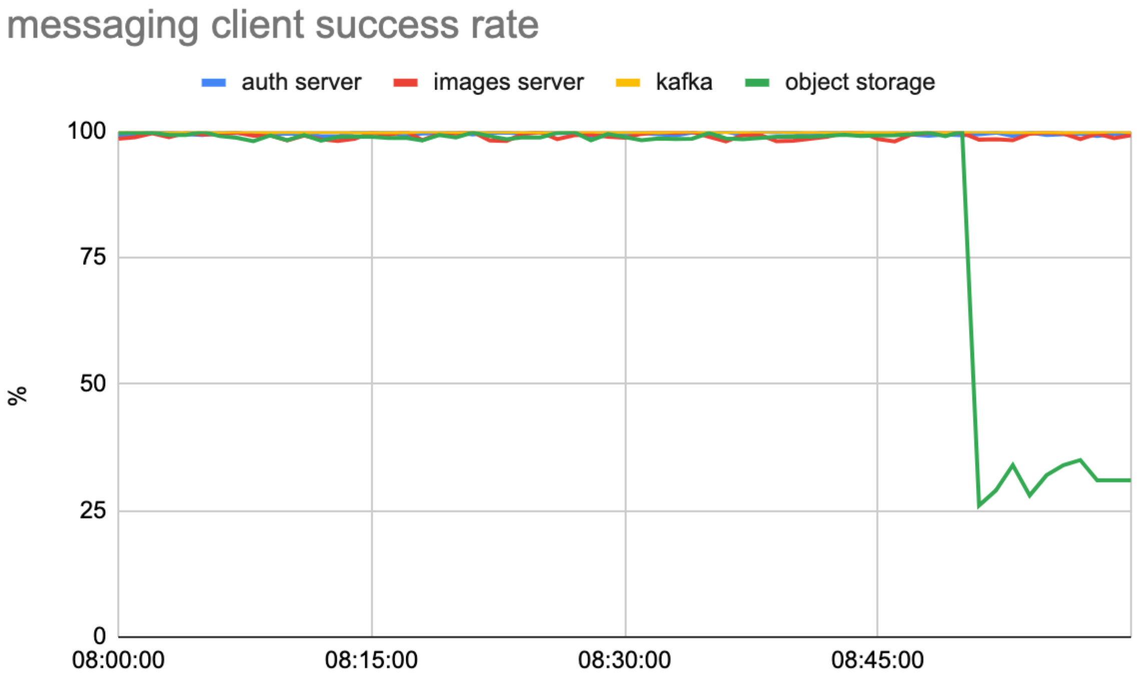


All workers entered a crash loop at 08:55

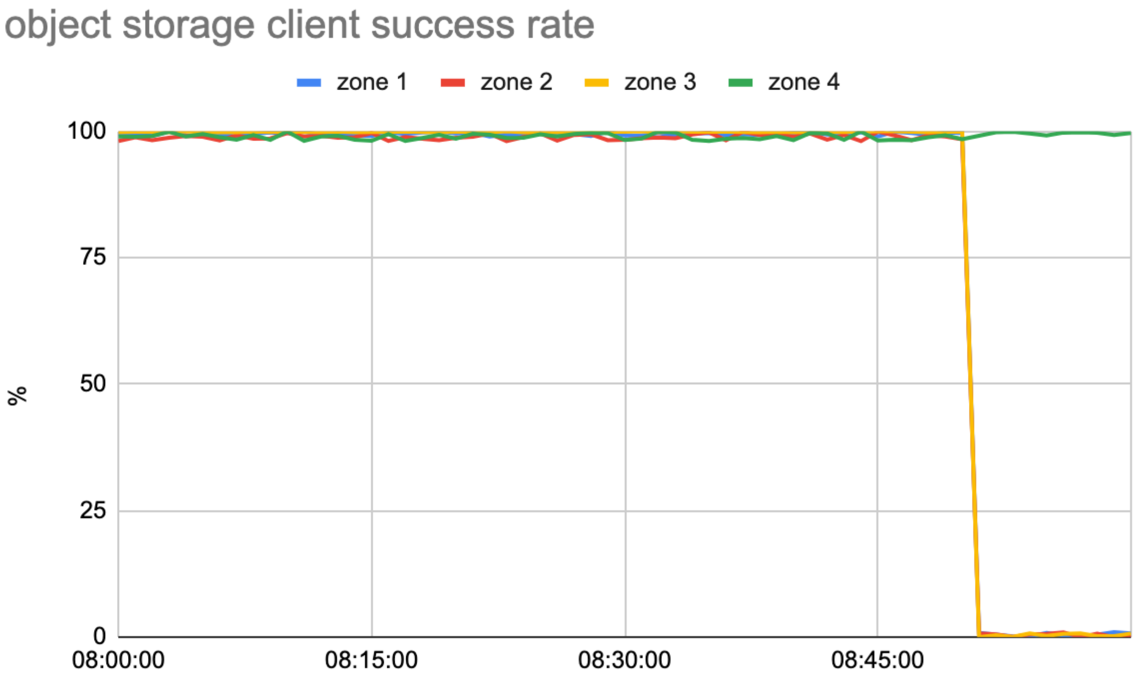
It answers several key questions at once:

- Was the latest release a long time ago?
- Are all the workers crashlooping?
- Did a worker crash or restart recently?

Example graph 2a: success rate by server



Example graph 2b: success rate by cluster



The challenge is as organizational as it is technical

Convincing yourself, others, or the boss 😊

- Challenges to convince an organization to adopt those practices
  - Expensive infra in $ and/or in engineering effort
  - Low but continuous effort needed from the developers to instrument their applications

Convincing yourself, others, or the boss 😊

- Challenges to convince an organization to adopt those practices
  - Expensive infra in $ and/or in engineering effort
  - Low but continuous effort needed from the developers to instrument their applications
- But from a cost perspective
  - There’s a reason why all industry leaders did it
  - Without the proper infra, the O&M cost becomes unsustainable at scale
  - What you don’t invest in infra, you’ll spend in inefficient disaster recovery
  - AIOps RCA research budgets speak for themselves: orgs are willing to pay a lot for effective RCA

Convincing yourself, others, or the boss 😊

- Challenges to convince an organization to adopt those practices
  - Expensive infra in $ and/or in engineering effort
  - Low but continuous effort needed from the developers to instrument their applications
- But from a cost perspective
  - There’s a reason why all industry leaders did it
  - Without the proper infra, the O&M cost becomes unsustainable at scale
  - What you don’t invest in infra, you’ll spend in inefficient disaster recovery
  - AIOps RCA research budgets speak for themselves: orgs are willing to pay a lot for effective RCA
- How to get started?
  - Start small and prove it: use a small system that can fit on free infra and show the difference
  - Make MTTR part of the **developers** KPIs, they’ll have incentives to instrument

Leads for successful RCA in AIOps research

RCA challenges today

- Lack of labeled data
  - Ops are reluctant to label it
  - Telemetry changes all the time with new releases and production noise
  - No data for rare problems
- Red herrings in the midst of complex production events
- Lack of instrumentation
- Anomaly detection: at scale, anomaly is the norm, reporting it doesn’t help

How to label data and identify red herrings?

- Knowing when it works and when it’s broken is a solved problem
  - Synthetic monitoring is low effort
  - Measuring success rates and latencies at the ingress point is even less effort
- RCA research could use this high quality signal without any effort from ops
  - Know with high confidence when it’s broken and when it’s working
  - Learn what anomalies are benign
  - Correlate potential cause timeline with time of breakage to eliminate red herrings
  - Bonus points for comparing clusters where it works to clusters where it’s broken
- Train during QA chaos testing, when a lot of brokenness should happen
- Hint: >50% of outages are due to config changes and releases

  -> make version and config hash first class citizens, not just another feature

Lack of instrumentation

- Could instrumentation quality be evaluated automatically during QA?
- What about instrumenting automatically?

Thank you!

Questions and comments: bjeunhomme@gmail.com