Spotify R&D

# How we Implemented High Throughput Logging at Spotify

Lauren Muhlhauser | EM | Core Infrastructure

# A Brief and Incomplete History of Logging at Spotify

- Back when backend services ran on their own VMs, they logged as much as they wanted to disk.
- As services were migrated to multi-tenant GKE clusters within a shared project we needed a solution for ephemerality.
- GKE provides a managed logging agent, previously FluentD - now FluentBit, that runs on each GKE node. This tails container log files and sends logs to the same project.
- This does not work for our multitenant clusters, as we want to forward logs from namespace ‘foo’ to project ‘foo' accordingly.

# A Brief and Incomplete History of Logging at Spotify

- We were running a custom FluentD image with a handful of plugins built in.
- This was run in conjunction with an init container that maps each namespace to its host project.
- Logs were collected across namespaces on a single GKE node and then forwarded to each corresponding project.

# Google Cloud Logging Output Plugin

- Our FluentD config created n+2 instances of Google's GCL output plugin.
    - One output per namespace.
    - One output to handle any logs tagged with "kubernetes.**" that were not matched by the namespace-specific output plugin and send logs to the multi-tenant project.
    - One output that caught any remaining logs and sent them to the multi-tenant project.
- Each instance of the output plugin had 2 threads. These threads were used to flush FluentD's internal buffer by sending log entries to GCL.
    - We configured FluentD to buffer to files on disk, have 1MB chunks, and flush data every five seconds.

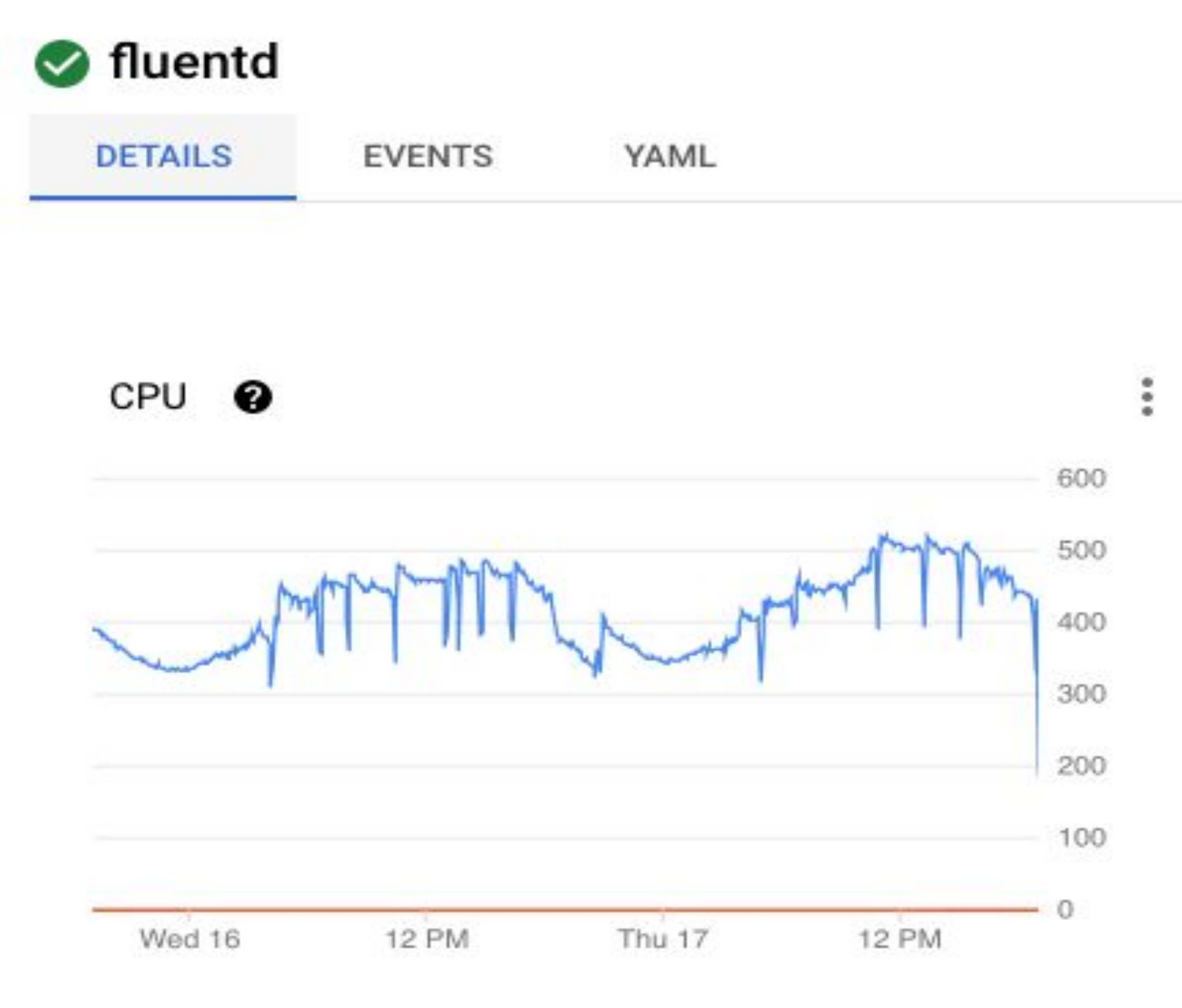
# A logging setup that works for dozens of namespaces won't necessarily work for over 1700 namespaces.

# Initial Investigation

- Users began to report delayed or completely absent logs.
- A small portion of logs were rejected by log ingestion quotas.
  - Some users may intentionally keep their tenant project's log quotas low to prevent runaway logging costs.
  - Some users may have exhausted their quotas and not know why they're not seeing logs.

# FluentD is Limited to a Single Core

- Many of our FluentD pods were using that single core fully and dropping logs as a result.
- The total CPU used by all FluentD pods in a cluster:
  500 CPU / 500 GKE nodes = 1 CPU fully used by each pod
- FluentD was so CPU bound in some cases that it didn't even detect some logs and didn't attempt to forward them to GCL.
- We actually don't know the number of logs lost due to CPU bottlenecks.



# Solutions We Considered

# FluentBit and Multi-Threading

## Pros

- FluentBit is the default logging agent in GKE 1.17+
- FluentBit added multithreading support in 1.7.x

## Cons

- Various versions OOMing
- Limited to 256 outputs
- With no timeline from Google on when/if we would be able to configure the out-of-the-box k8s FluentBit offering, we once again would roll out our own image + config.
  - This would allow us to configure the number of workers we need for our output plugins.

# Logback to Send Java Service Logs Directly to GCL

## Pros

- Services are directly responsible for their log volume, so if they overproduce it won't get silently dropped
- Can configure Java services to write to GCL in their parent project instead of the multi-tenant GKE project
- We can add tracing data to the GCL logs

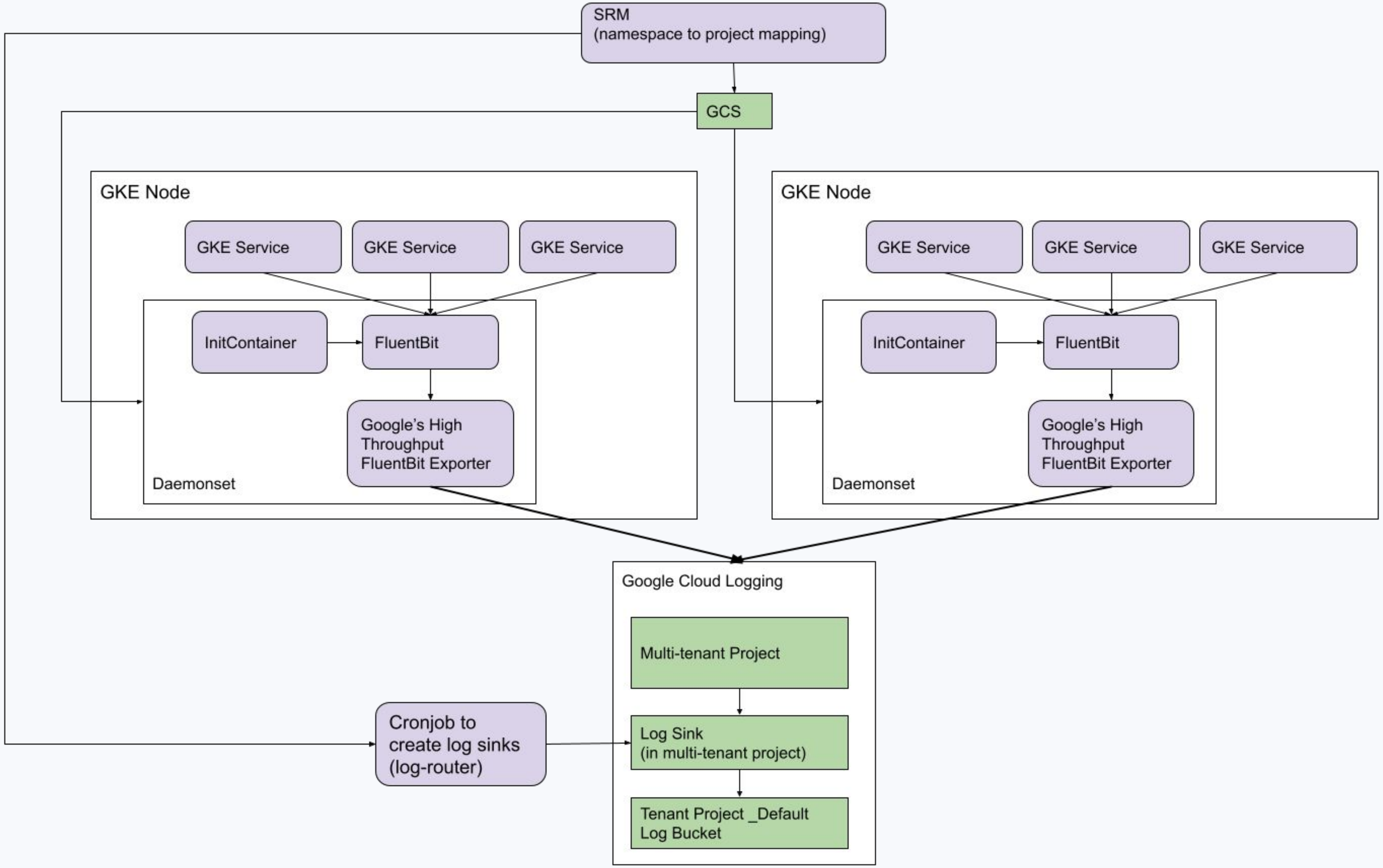
## Cons

- Updates/changes would require PRs and deploying every individual service
- Observability team would be expected to manage default configs and troubleshooting for the whole company
- This is only a solution for Java services

# Our Final Solution

# FluentBit + High Throughput Exporter Daemonset

- FluentBit + High Throughput Exporter daemonset deployed in each GKE cluster.
  - Specific version of FluentBit that did not OOM.
- Only 1 FluentBit output required, all logs pass through the exporter.
- Improved resource allocation for the deployment.
- Per project log sinks in the multi-tenant project.
- No more massive 1700+ output FluentD config file.


SRM
(namespace to project mapping)
GCS
GKE Node
GKE Service
GKE Service
GKE Service
InitContainer
FluentBit
Google's High Throughput FluentBit Exporter
Daemonset
GKE Node
GKE Service
GKE Service
GKE Service
InitContainer
FluentBit
Google's High Throughput FluentBit Exporter
Daemonset
Google Cloud Logging
Multi-tenant Project
Log Sink
(in multi-tenant project)
Tenant Project _Default Log Bucket
Cronjob to create log sinks (log-router)

Logging Architecture

# It worked a little too well...

- We were unsure of how many logs we were dropping and which projects were logging the largest amount.
  - Guesstimated at least 80% of logs were dropped.
  - For java services, we used data studio to show teams what percentage of logs were dropped.
- We worked with procurement during the rollout to observe cost.
- Got up to testing + ⅔ production clusters before we had to roll back the solution due to excessive cost.

# Back to the drawing board...

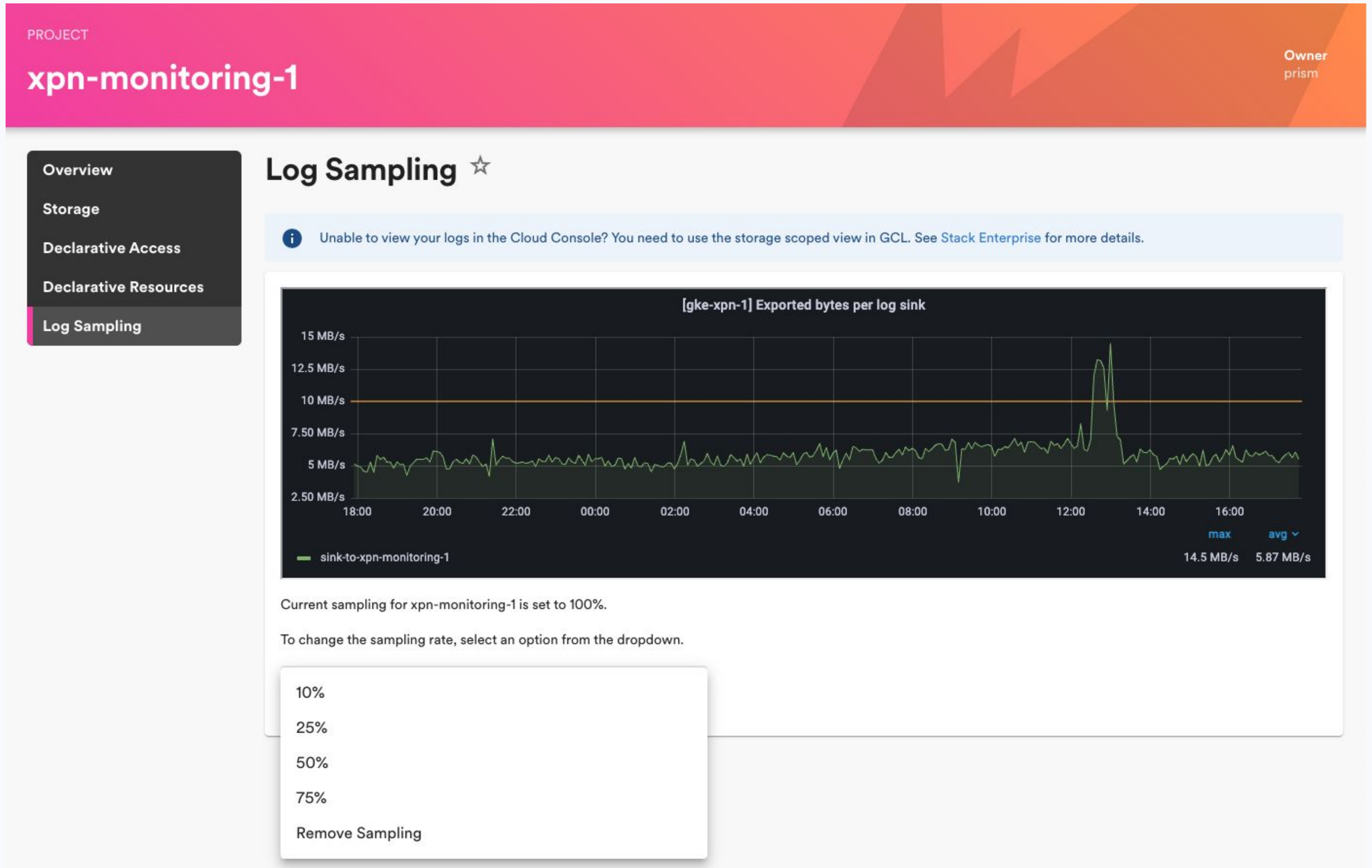
- Turns out log throughput was even higher than we thought for some projects, it makes sense that services in certain clusters weren't getting any logs at all!
- We used the cost data from the initial rollout to notify high spend project owners.
- We now had per project metrics for log throughput available to us for the next rollout.
- Google informed us that we could set sampling within project sinks with this config: sample(insertId, 0.10) AND ...

# Sampling

- To automate sampling on high spend projects, we created an alert in GCP that checks for high throughput per project.
- The alert hits pubsub which then gets picked up by our sampling service.
- The service sets sampling to 10% (meaning you will get 10% of your project’s logs).
- The project owners will be notified via email that their project has been sampled.

# Log Sampling in Backstage

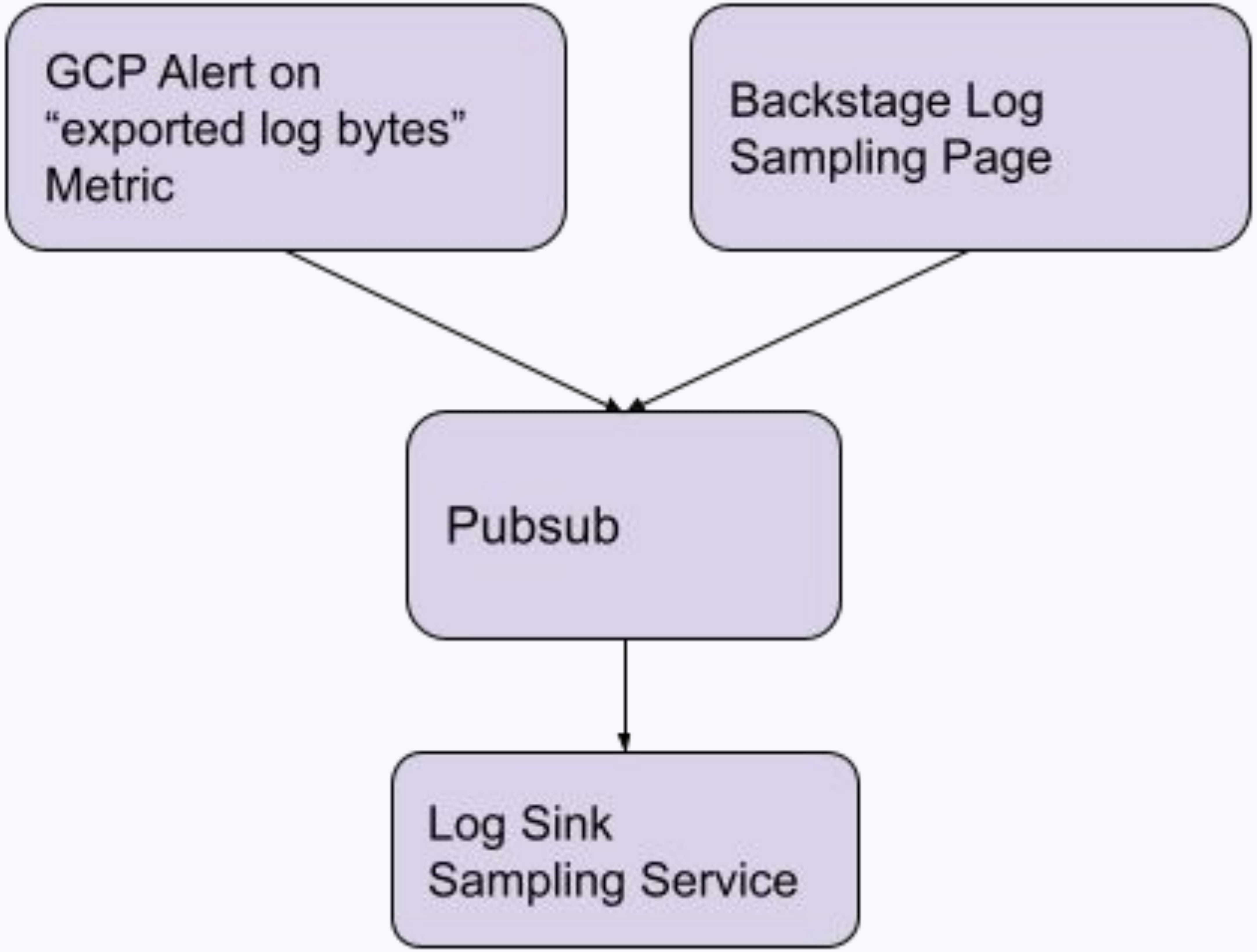
- We also added a page in Backstage that allows teams to view and adjust the sampling for their project.



Unable to view your logs in the Cloud Console? You need to use the storage scoped view in GCL. See Stack Enterprise for more details.

Current sampling for xpn-monitoring-1 is set to 100%.

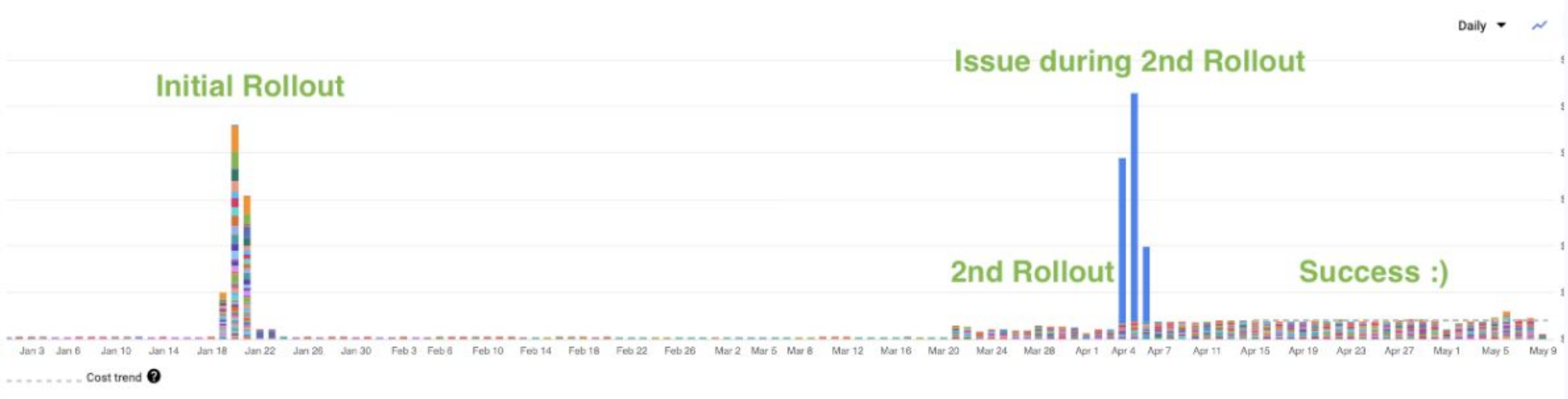
To change the sampling rate, select an option from the dropdown.


GCP Alert on "exported log bytes" Metric
Backstage Log Sampling Page
Pubsub
Log Sink Sampling Service

Log Sampling Architecture

# Crisis averted!

- With automated sampling, after a few adjustments, we were able to successfully roll out the FluentBit based solution the second time around.
- High throughput projects were sampled accordingly (to reduce cost) and no longer throttling other projects within the same cluster.



# Limitations

- There are a few limitations to the multi-tenant high throughput solution:
  - Currently, the default log view in GCP does not display logs, you need to navigate to the storage view.
  - Log based metrics and alerts in the GCP console do not work with logs in the _Default log bucket.
  - Google’s Error Reporting feature does not work with logs in the _Default log bucket.
  - You cannot create a log sink within a tenant project for logs in the _Default log bucket. Log sinks would need to be in the multi-tenant project.
- We have feature requests open to resolve these issues.
- As a workaround, teams can send logs to GCL directly from their service
  - This will route logs directly to their parent project and allow them to use the above features.
  - There are instructions for: C#, Java, Go, Node, PHP, Python, and Ruby

# Logging Best Practices

- By default, only WARN and ERROR logs should be sent to GCL.
- Use tracing instead.
- Use metrics instead.
- Disable logging to GCL completely if you do not need it.

Each of the above has trade-offs. The important thing to keep in mind is that full request logging can be very expensive for high-traffic services.

# Everything you log should have a purpose.

Whether it’s usage data, user events, or application errors and exceptions, it should be valuable to your team. A basic rule of thumb for whether to log something can be to ask yourself, **“Is this information immediately useful in some way, and will it provide the details I need to understand the underlying cause and make decisions?”**

# Thank you

Spotify R&D