Observability at Tigris Data

Peter Boros
Founding Engineer @ Tigris Data

About Me

- Founding Engineer at Tigris
- platform / databases
- Performance minded

- 1000s of production databases
- Mostly MySQL (Percona, Dropbox, Zuora)

About Tigris

- S3 Compatible object storage
- Data is always close to the user

- Thousands of buckets
- Petabytes of data
- Billions of requests every day

Agenda

- Check out the title slide
- Check out the agenda
- Journey of Tigris Observability
- 3 pillars of observability
  - Logs
  - Traces
  - Metrics
- Battle scars
- Architectural recommendations
- Future directions

Takeaways

- Logging should be able to take extra load
- Sample traces
- Manage metrics cardinality

Started not so long ago

Tigris is a young company growing fast

Why do we need observability?

- We need to support our customers who are operating at scale
- Tighten the feedback loop, shipping faster
- This helps us more precisely optimize the system

3 Pillars of observability

- Metrics
- Logs
- Traces

Extended 3 pillars

- Visualization
- Continuous profiling

Observability as a Service

We used a couple of them

DATADOG

Grafana Cloud

All of them are great
Providing all the pillars
There are many other options

Problematic:

- Number of custom metrics
- Number of hosts
- Amount of logs

Great if you can control these

We needed to do something

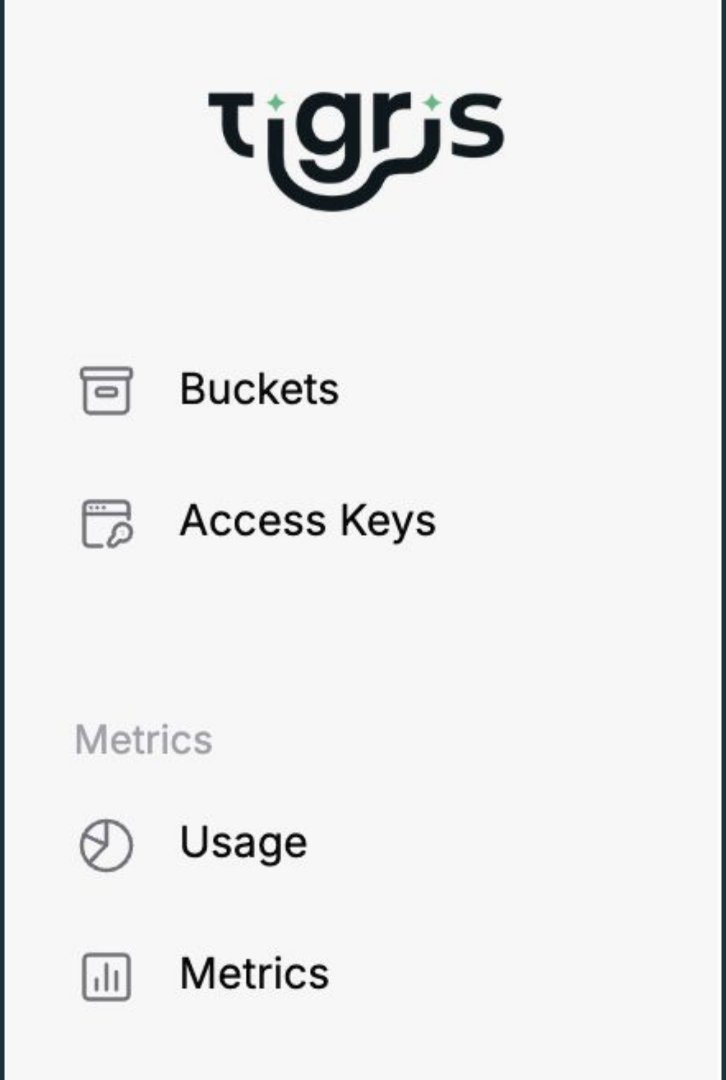
- We kept adding kubernetes cluster
- We kept adding regions
- Users kept coming
- We wanted to provide users granular metrics
- We wanted to build some features on top of these granular metrics

→ Our projected cloud observability bill would have been 6 figures in months

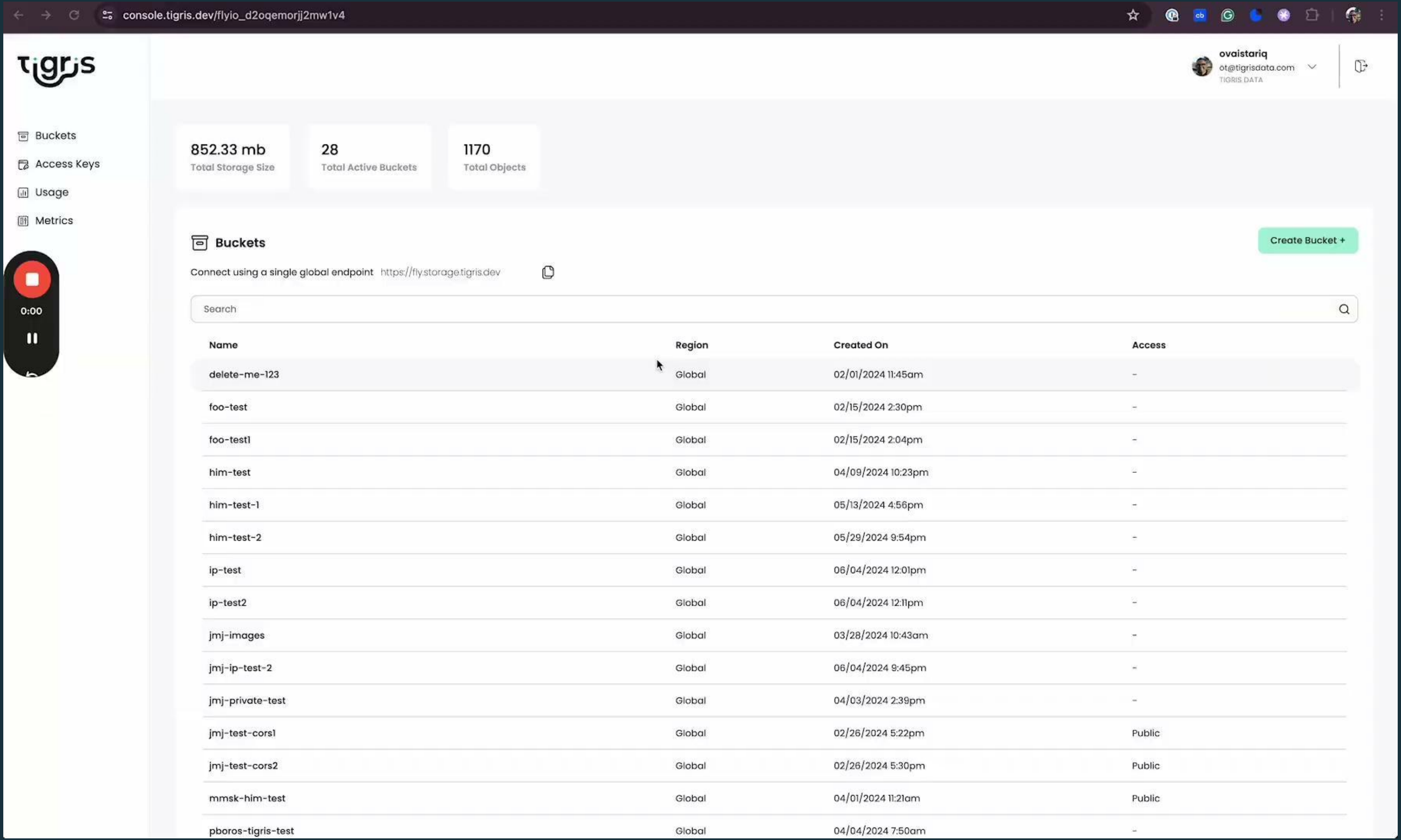
Running our own observability stack

- Total infra cost is 1-2% of the total cloud based observability solution cost
- In-house workload
- Control, it will behave exactly how you want it

Our own stack



Tigris console



console.tigris.dev/flyio_d2oqemorjj2mw1v4

ovaistariq
ot@tigrisdata.com
TIGRIS DATA

Buckets
Access Keys
Usage
Metrics

0:00

852.33 mb	28	1170
Total Storage Size	Total Active Buckets	Total Objects

Buckets

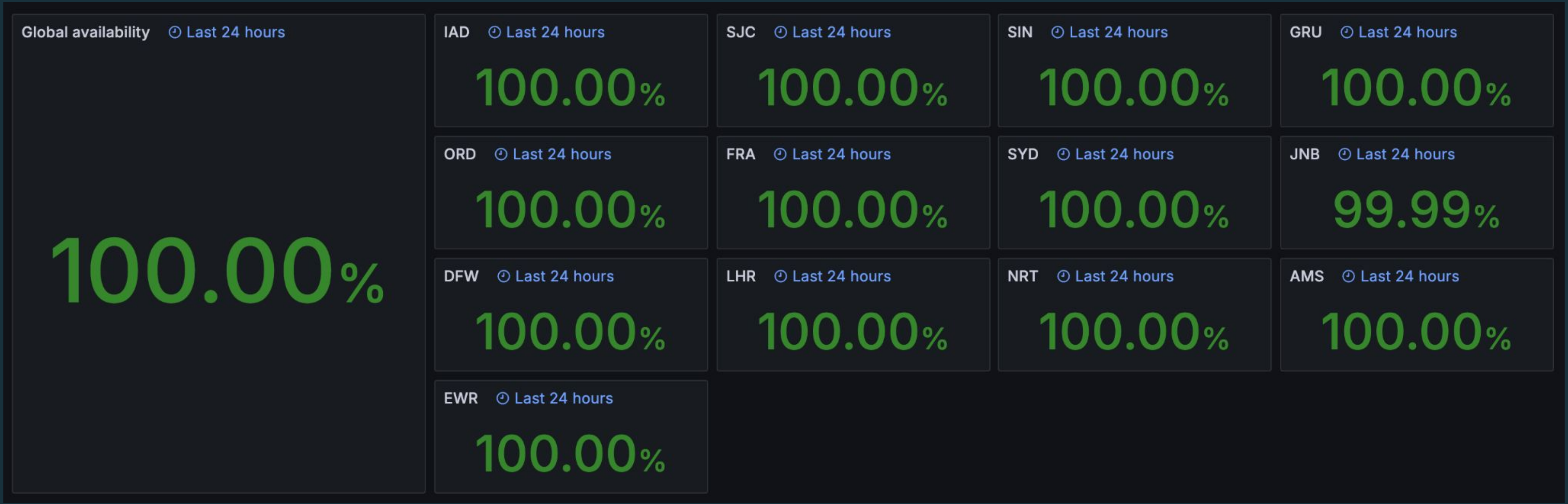
Create Bucket +

Connect using a single global endpoint https://fly.storage.tigris.dev

Search

Name	Region	Created On	Access
delete-me-123	Global	02/01/2024 11:45am	-
foo-test	Global	02/15/2024 2:30pm	-
foo-test1	Global	02/15/2024 2:04pm	-
him-test	Global	04/09/2024 10:23pm	-
him-test-1	Global	05/13/2024 4:56pm	-
him-test-2	Global	05/29/2024 9:54pm	-
ip-test	Global	06/04/2024 12:01pm	-
ip-test2	Global	06/04/2024 12:11pm	-
jmj-images	Global	03/28/2024 10:43am	-
jmj-ip-test-2	Global	06/04/2024 9:45pm	-
jmj-private-test	Global	04/03/2024 2:39pm	-
jmj-test-cors1	Global	02/26/2024 5:22pm	Public
jmj-test-cors2	Global	02/26/2024 5:30pm	Public
mmsk-him-test	Global	04/01/2024 11:21am	Public
pboros-tigris-test	Global	04/04/2024 7:50am	-

Logging: Loki



Global availability Last 24 hours
100.00%

IAD Last 24 hours
100.00%

SJC Last 24 hours
100.00%

SIN Last 24 hours
100.00%

GRU Last 24 hours
100.00%

ORD Last 24 hours
100.00%

FRA Last 24 hours
100.00%

SYD Last 24 hours
100.00%

JNB Last 24 hours
99.99%

DFW Last 24 hours
100.00%

LHR Last 24 hours
100.00%

NRT Last 24 hours
100.00%

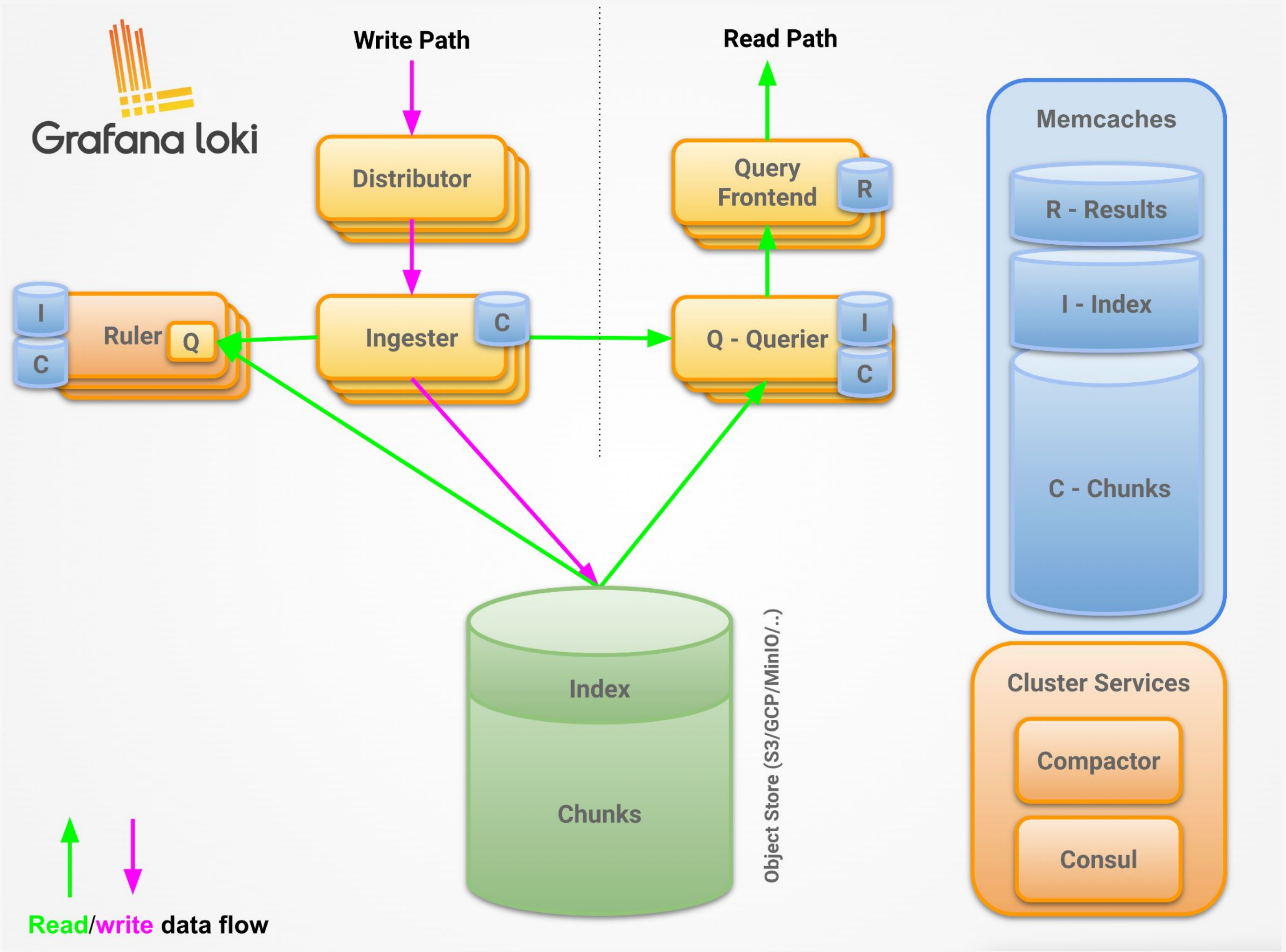
AMS Last 24 hours
100.00%

EWR Last 24 hours
100.00%

Logging: Loki

Evaluated few log solutions

- Parse logs and index or not
- Famous for parsing and indexing: ELK
- We chose loki
    - Optimized for the write path
    - Object storage backend


Grafana loki
Write Path
Read Path
Distributor
Query Frontend
R
Memcaches
R - Results
I - Index
C - Chunks
I
C
Ruler
Q
Ingester
C
Q - Querier
I
C
Index
Chunks
Object Store (S3/GCP/MinIO/..)
Cluster Services
Compactor
Consul
Read/write data flow

Advantages

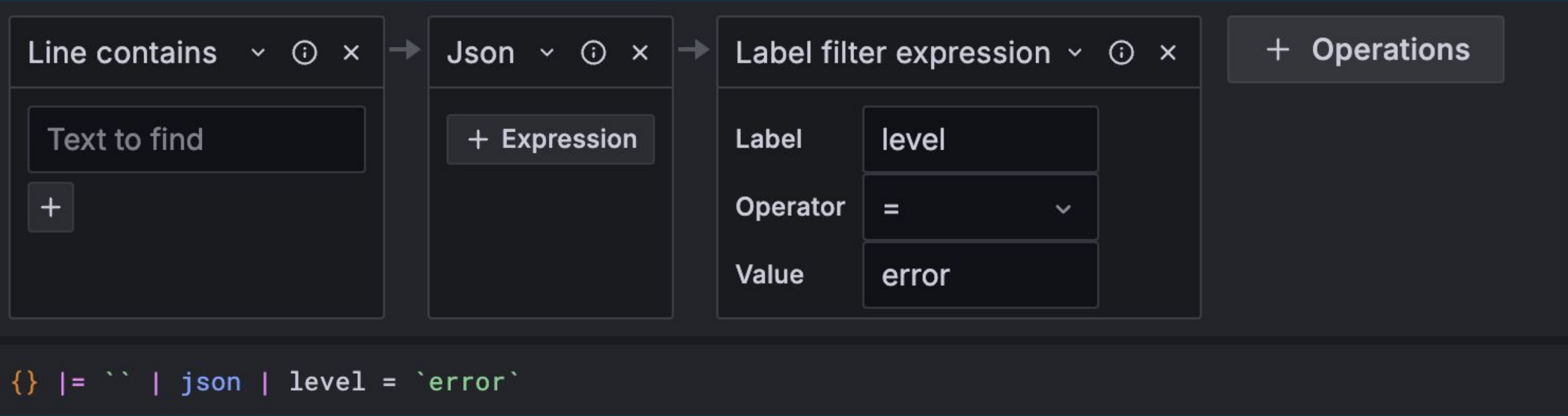
- Ingest anything
- Very versatile search options
- Very little indexing

Disadvantages

- Slow search that is done on the client side

Searching on the client side

- Response time can be minutes
- Querying recent data is fast



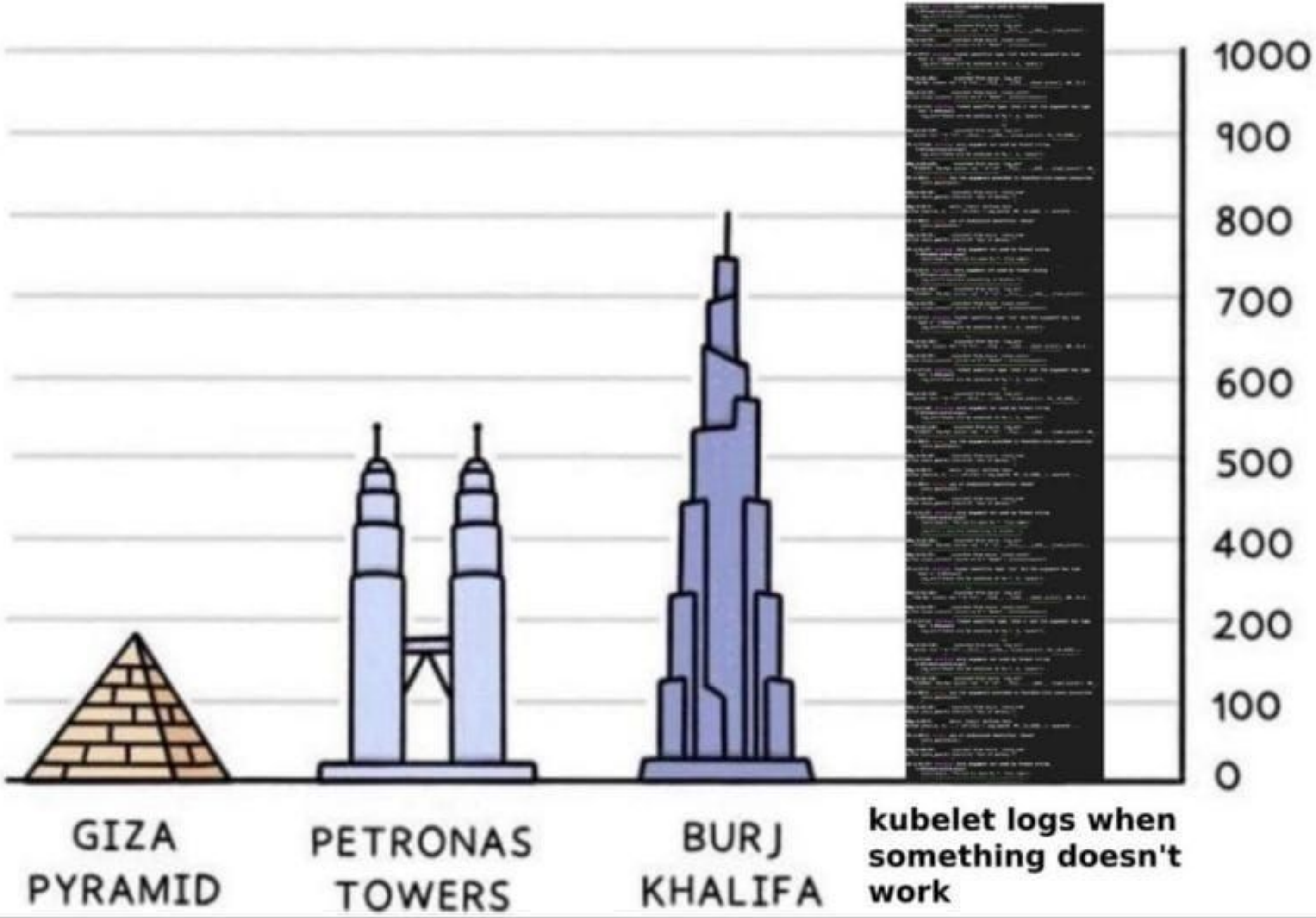
Label: level

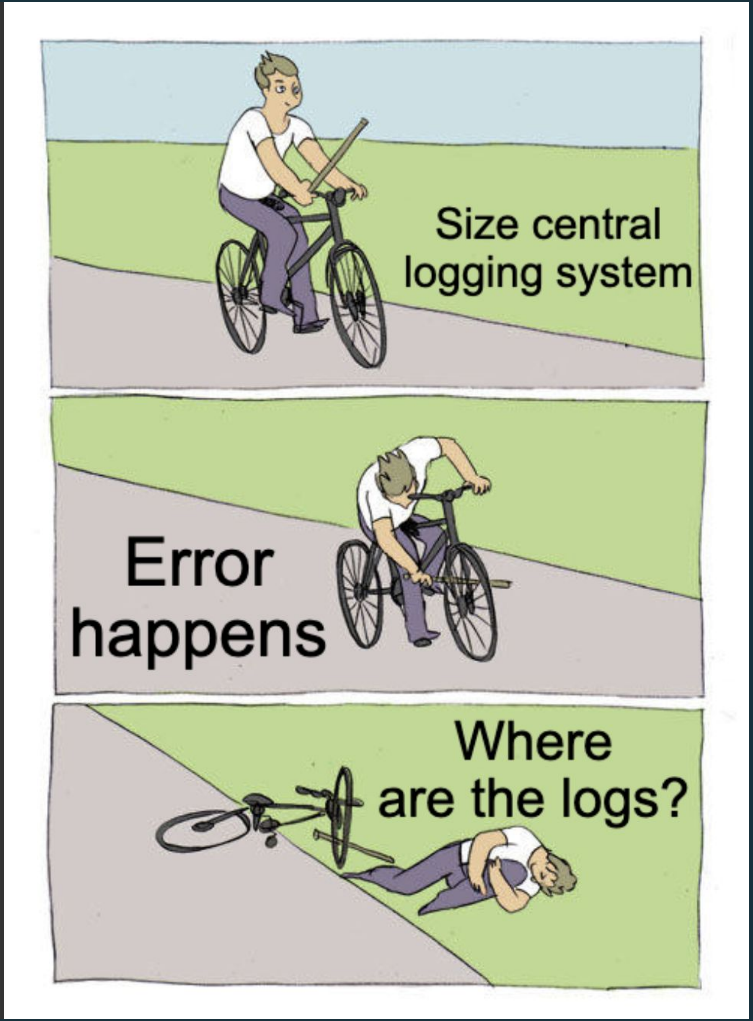
Operator: =

Value: error

```
{} |= `` | json | level = `error`
```

Logging: Pitfalls


1000
900
800
700
600
500
400
300
200
100
0
GIZA PYRAMID
PETRONAS TOWERS
BURJ KHALIFA
kubelet logs when something doesn't work



Don't size for the happy path.

Test that you can still ingest 10-100x logs volume.

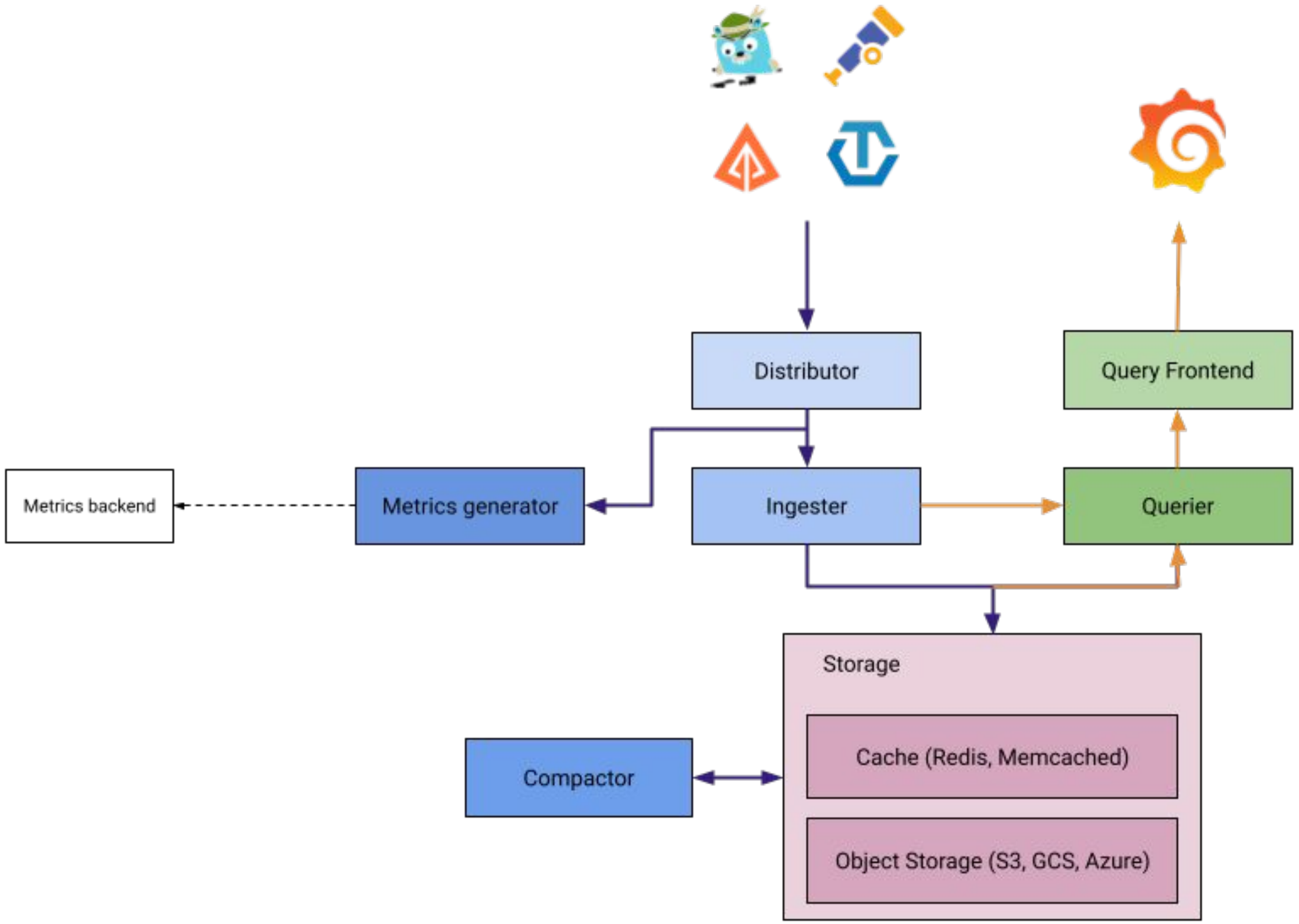
Learned this the hard way.

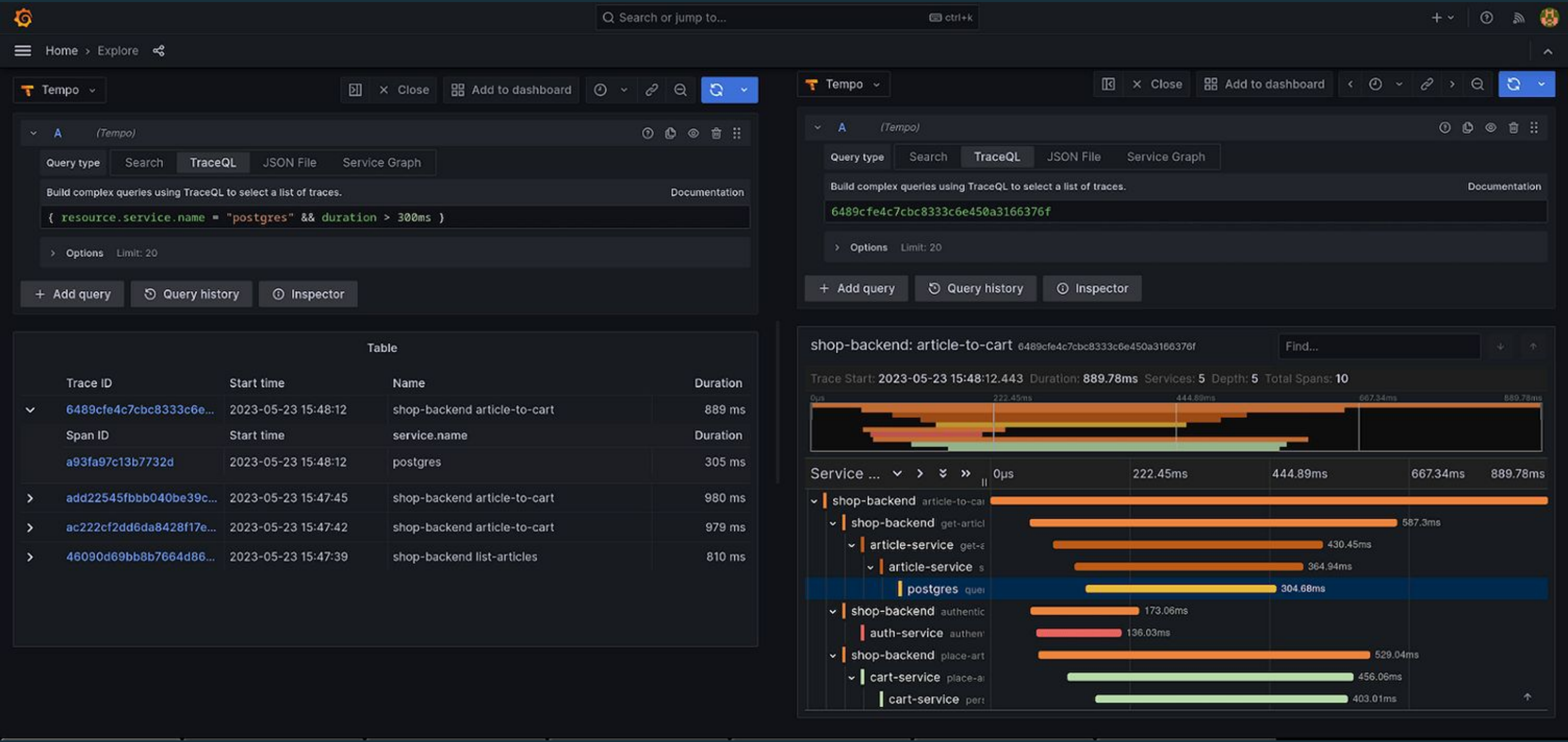
Story time: debug logs

Takeaway

Be prepared for an increased amount of logs

Traces: Tempo


Distributor
Query Frontend
Metrics backend
Metrics generator
Ingester
Querier
Storage
Compactor
Cache (Redis, Memcached)
Object Storage (S3, GCS, Azure)



Build complex queries using TraceQL to select a list of traces. Documentation

```
{ resource.service.name = "postgres" && duration > 300ms }
```

Options Limit: 20

Table

Trace ID	Start time	Name	Duration
6489cfe4c7cbc8333c6e...	2023-05-23 15:48:12	shop-backend article-to-cart	889 ms
Span ID	**Start time**	**service.name**	**Duration**
a93fa97c13b7732d	2023-05-23 15:48:12	postgres	305 ms
add22545fbbb040be39c...	2023-05-23 15:47:45	shop-backend article-to-cart	980 ms
ac222cf2dd6da8428f17e...	2023-05-23 15:47:42	shop-backend article-to-cart	979 ms
46090d69bb8b7664d86...	2023-05-23 15:47:39	shop-backend list-articles	810 ms

Build complex queries using TraceQL to select a list of traces. Documentation

```
6489cfe4c7cbc8333c6e450a3166376f
```

Options Limit: 20

shop-backend: article-to-cart 6489cfe4c7cbc8333c6e450a3166376f

Trace Start: 2023-05-23 15:48:12.443 Duration: 889.78ms Services: 5 Depth: 5 Total Spans: 10

Service	Operation	Duration
shop-backend	article-to-car	
shop-backend	get-articl	587.3ms
article-service	get-a	430.45ms
article-service	s	364.94ms
postgres	quer	304.68ms
shop-backend	authentic	173.06ms
auth-service	authen	136.03ms
shop-backend	place-art	529.04ms
cart-service	place-a	456.06ms
cart-service	pers	403.01ms

- Extremely good insights
- Very very useful at the start
- Problems come with traffic

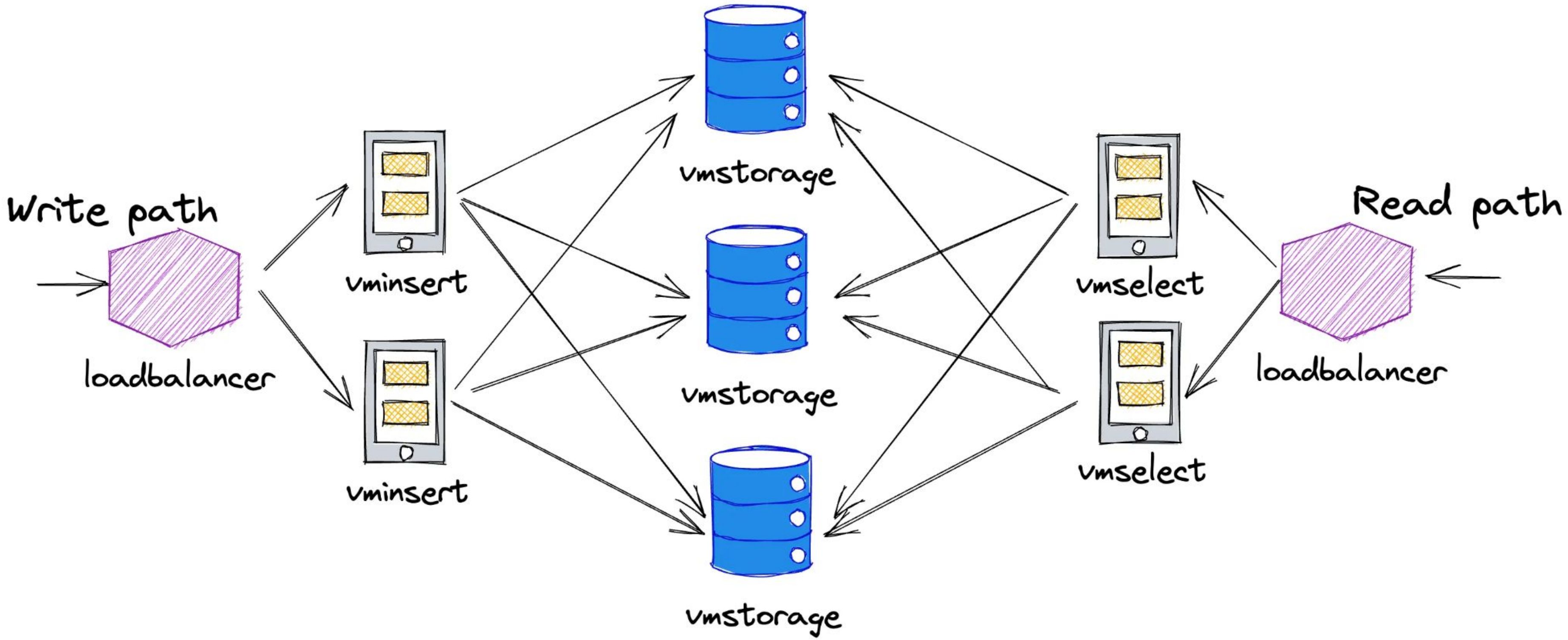
- Lots of requests → lots of traces
- Billions of OK requests are not interesting
- Sampling
  - Head: Trace a % of request
  - Tail: Save the trace if it was interesting
    - App level
    - OpenTelemetry collector level
- Metrics and Logs are actually a subset of traces

Takeaway

Sample traces

Metrics: VictoriaMetrics

- We started on Mimir, but switched to VictoriaMetrics for efficiency reasons
- We have one set of metrics
  - User facing data
  - Public metrics
  - Billing
- Extremely important, most learnings are here


Write path
loadbalancer
vminsert
vminsert
vmstorage
vmstorage
vmstorage
vmselect
vmselect
Read path
loadbalancer

Efficiency comes at a price

- No rebalancing of shards
- No healing of data for missing storage
- Good physical and logical backup options

Short term metrics:

- As granular as possible
- Used for troubleshooting, alerting
- Wiping them is fine as a resharding method

Long term metrics:

- Very low granularity is fine
- Source for billing, trends, etc
- Different backup characteristics

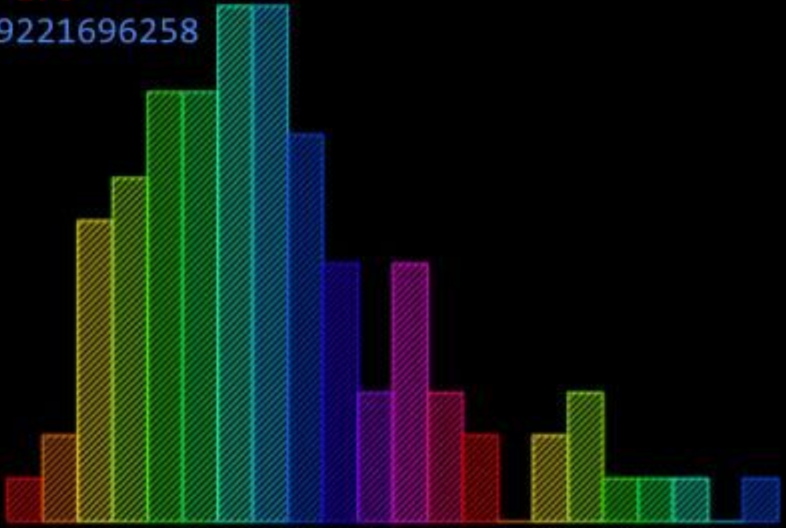
Metrics sources

- We use tally to create prometheus metrics
- Scraped by vmagent
- Originally one per cluster
- Sharded afterwards
- VictoriaMetrics operator helps a lot

Timers vs Histograms

- Tracking response time metrics is trickly
- Rolling aggregation
  - Tricky with high metric churn rate
- Histogram
  - Response time buckets
  - Can be converted to response time quantile

```
# HELP rpc_durations_histogram_seconds RPC latency distributions.
# TYPE rpc_durations_histogram_seconds histogram
rpc_durations_histogram_seconds_bucket{le="-8.999999999999979e-05"} 57
rpc_durations_histogram_seconds_bucket{le="1.0000000000000216e-05"} 90
rpc_durations_histogram_seconds_bucket{le="0.00011000000000000022"} 120
rpc_durations_histogram_seconds_bucket{le="0.0003100000000000002"} 160
rpc_durations_histogram_seconds_bucket{le="0.0005100000000000003"} 168
rpc_durations_histogram_seconds_bucket{le="0.0007100000000000003"} 170
rpc_durations_histogram_seconds_bucket{le="+Inf"} 170
rpc_durations_histogram_seconds_sum -0.00017415729221696258
rpc_durations_histogram_seconds_count 170
```

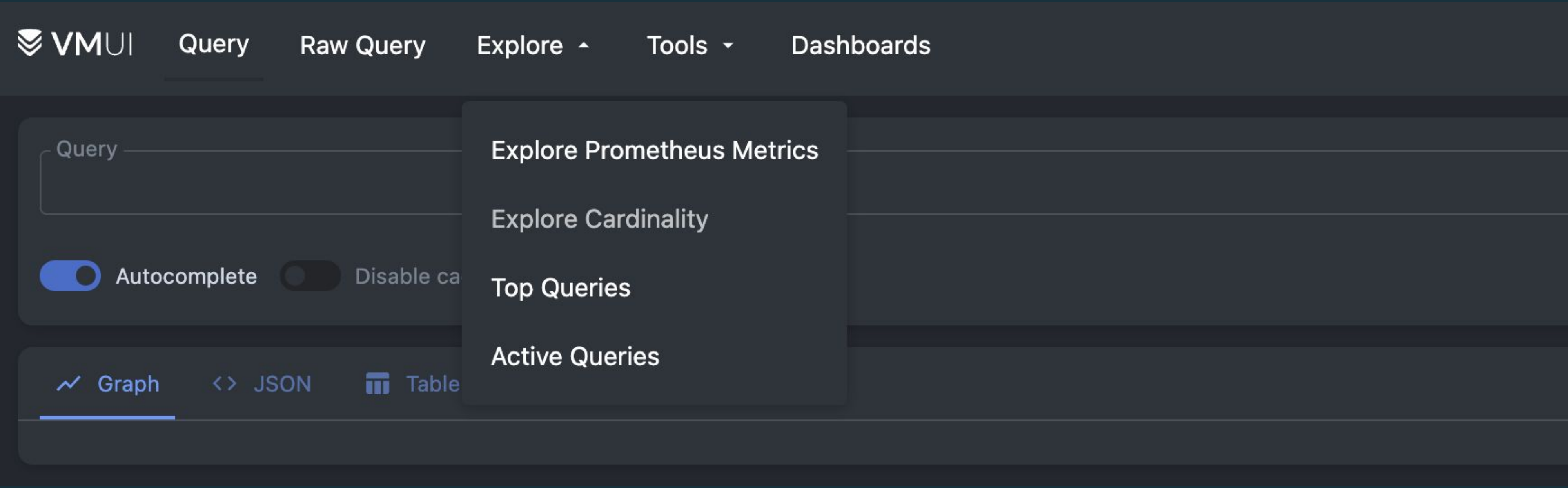


- High cardinality metrics
  - Buckets / Tenants
  - Infrastructure size (pods, machines)
  - Storage tiers
  - Object size

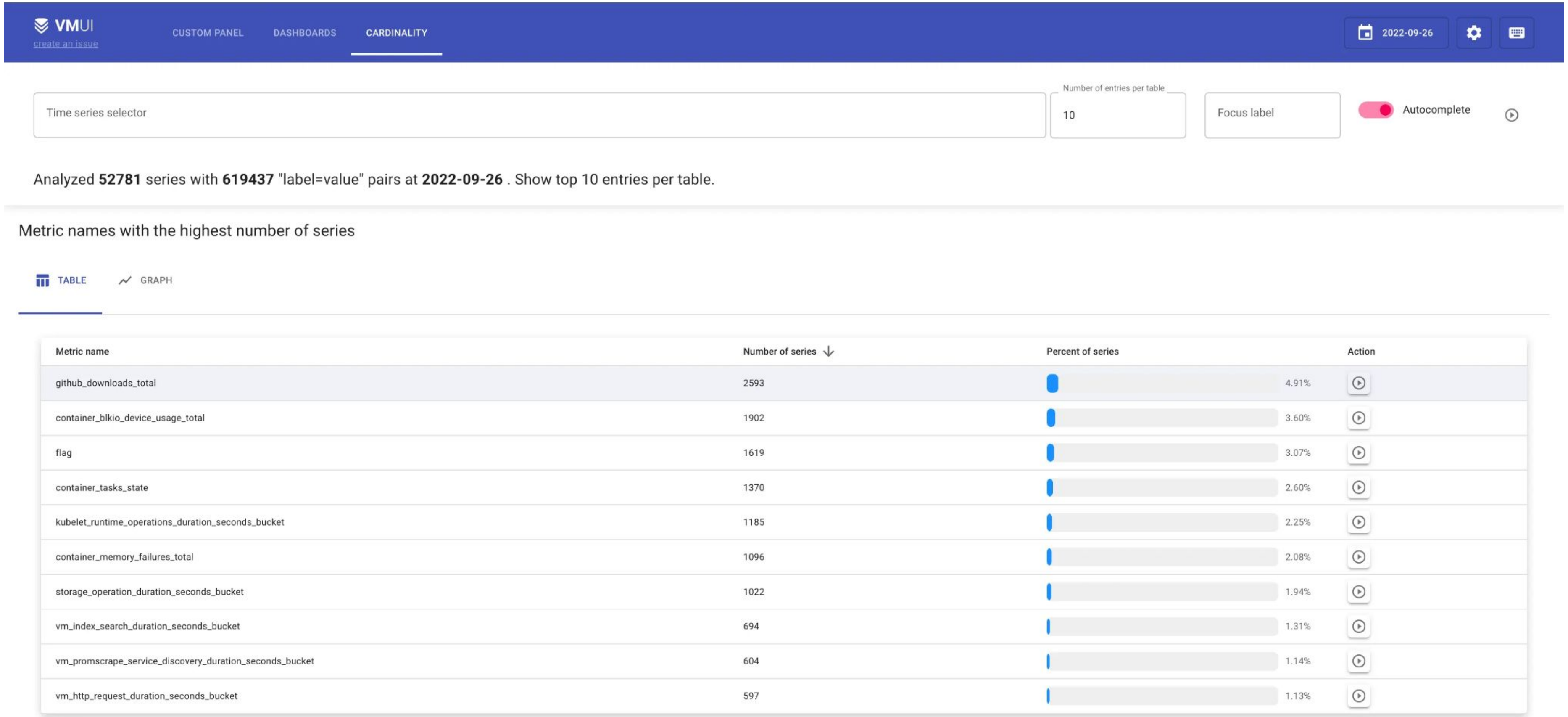
- Multiplying quickly
- Queries over many time series is very expensive
- Unique tag combinations

- Some expensive queries are valuable ones
  - Overall traffic
  - Overall response time in a region

Managing cardinality



Story time: exploding cardinality with errors



Analyzed **52781** series with **619437** "label=value" pairs at **2022-09-26** . Show top 10 entries per table.

Metric names with the highest number of series

Metric name	Number of series	Percent of series	Action
github_downloads_total	2593	4.91%	
container_blkio_device_usage_total	1902	3.60%	
flag	1619	3.07%	
container_tasks_state	1370	2.60%	
kubelet_runtime_operations_duration_seconds_bucket	1185	2.25%	
container_memory_failures_total	1096	2.08%	
storage_operation_duration_seconds_bucket	1022	1.94%	
vm_index_search_duration_seconds_bucket	694	1.31%	
vm_promscrape_service_discovery_duration_seconds_bucket	604	1.14%	
vm_http_request_duration_seconds_bucket	597	1.13%	

Application level aggregation

- Just double emit metrics and query the aggregate
- Works for us for buckets

For hosts and pods, this approach is not good.

Streaming aggregation

- Configured at the vmagent level
- Very efficient, done on the fly

- No backfills
- Can be inaccurate for histograms

```yaml
streamAggrConfig:
 keepInput: true
 rules:
   - match: requests_ok
     interval: 10s
     without:
       - instance
       - pod
     outputs:
       - total_prometheus
```

```
requests_ok:10s_without_instance_pod_total_prometheus
```

Streaming aggregation problems

- Asynchronous nature
- Summarizing histograms

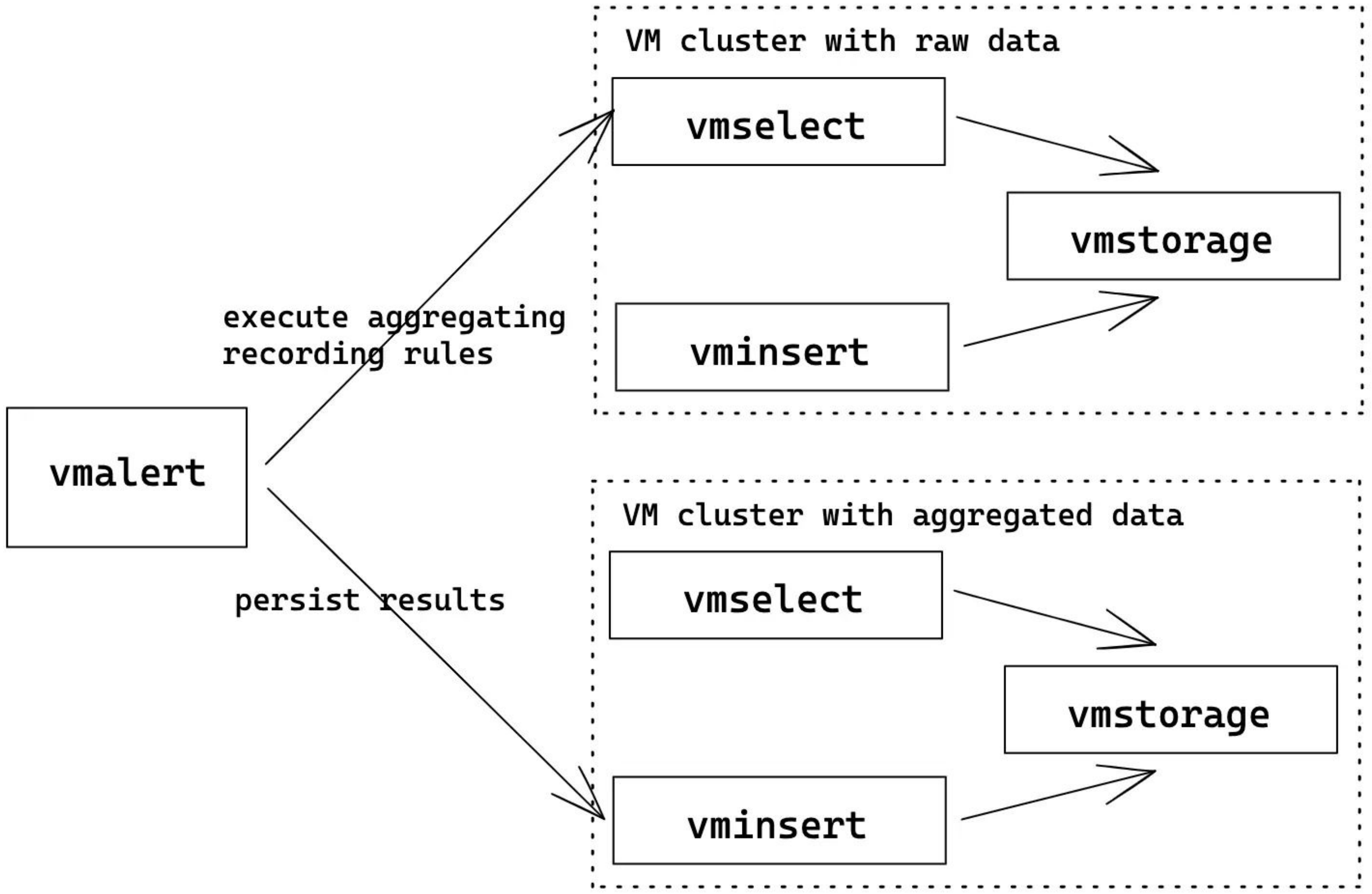
Recording rules with vmalert

- Backfills
- Can correct itself
    - Frequency and lookback window can be different

- Inefficient
- Wish there was a similar, in-engine solution

Recording rules

```yaml
spec:
 groups:
   - name: recording-rules-sjc-short1
     interval: 10s
     rules:
       - record: requests_aggr_ok:sum:some_name
         expr: |-
           sum by (http_method, region, size, env) (
             requests_ok{service="myservice"}
           )
```


VM cluster with raw data
vmselect
vmstorage
vminsert
execute aggregating recording rules
vmalert
persist results
VM cluster with aggregated data
vmselect
vmstorage
vminsert

Takeaway

Manage cardinality by aggregations

Takeaways

- Logging should be able to take extra load
- Sample traces
- Manage metrics cardinality

Future directions

- Continuous profiling
- Move fully to OpenTelemetry
- Make even more observability data accessible

Tigris

- Thanks to Tigris for sending me here
- If you want to try tigris out: storage.new
- Our public availability dashboard

Questions?

tigris

Thank you!

tigrisdata.com