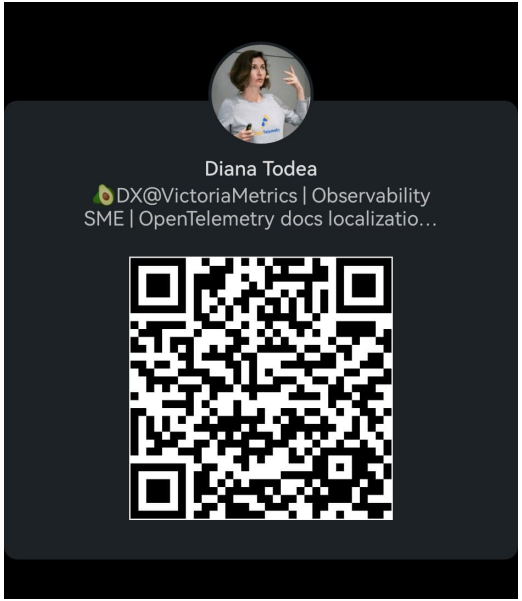
Beyond Prometheus: pushing the boundaries of scalable monitoring with VictoriaMetrics

Cloud Native Bergen | 28 October 2025

Diana Todea - DX Engineer

Bsky: @didiviking.bsky.social
X: @dianavtodea
Github: @didiViking/Conferences_Talks

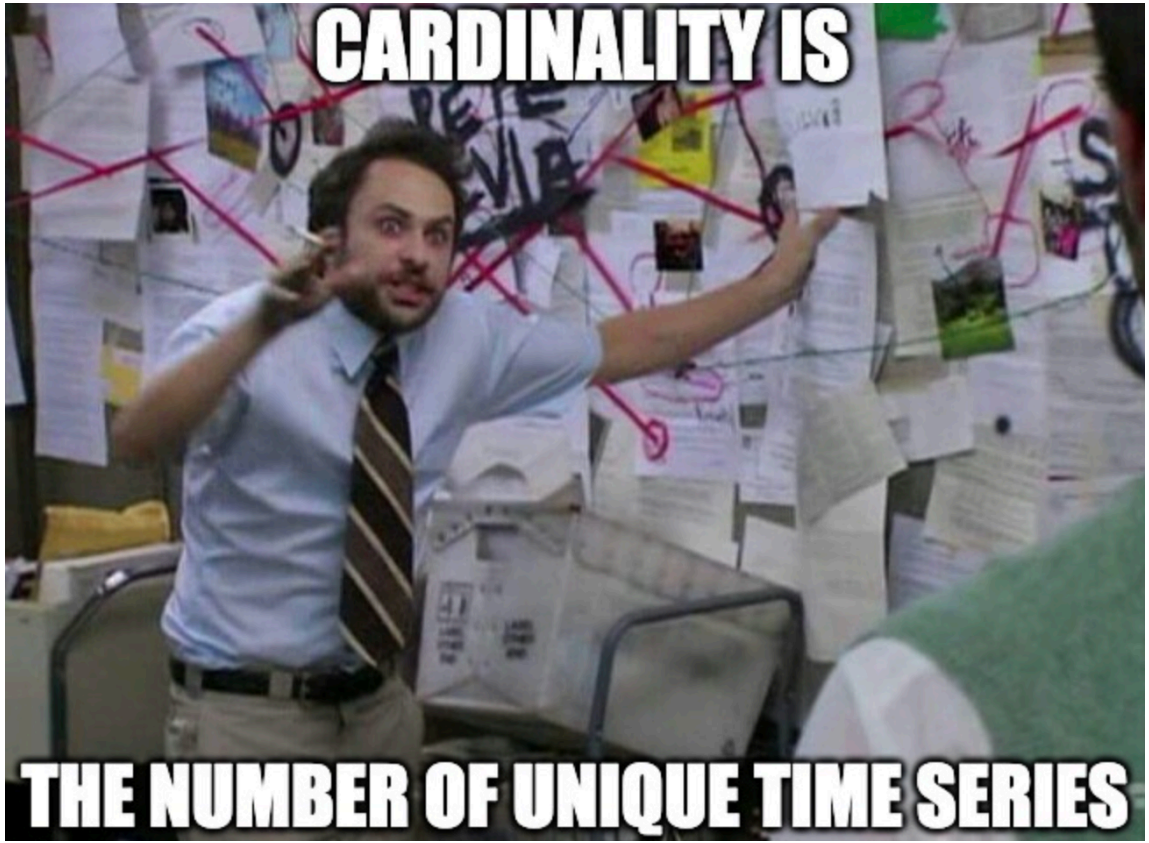


Agenda

- Our common frenemy: cardinality
- To be or not to be Prometheus
- My next superhero
- Happy demo!
- Fatherly advice


im lagging
how often
all the

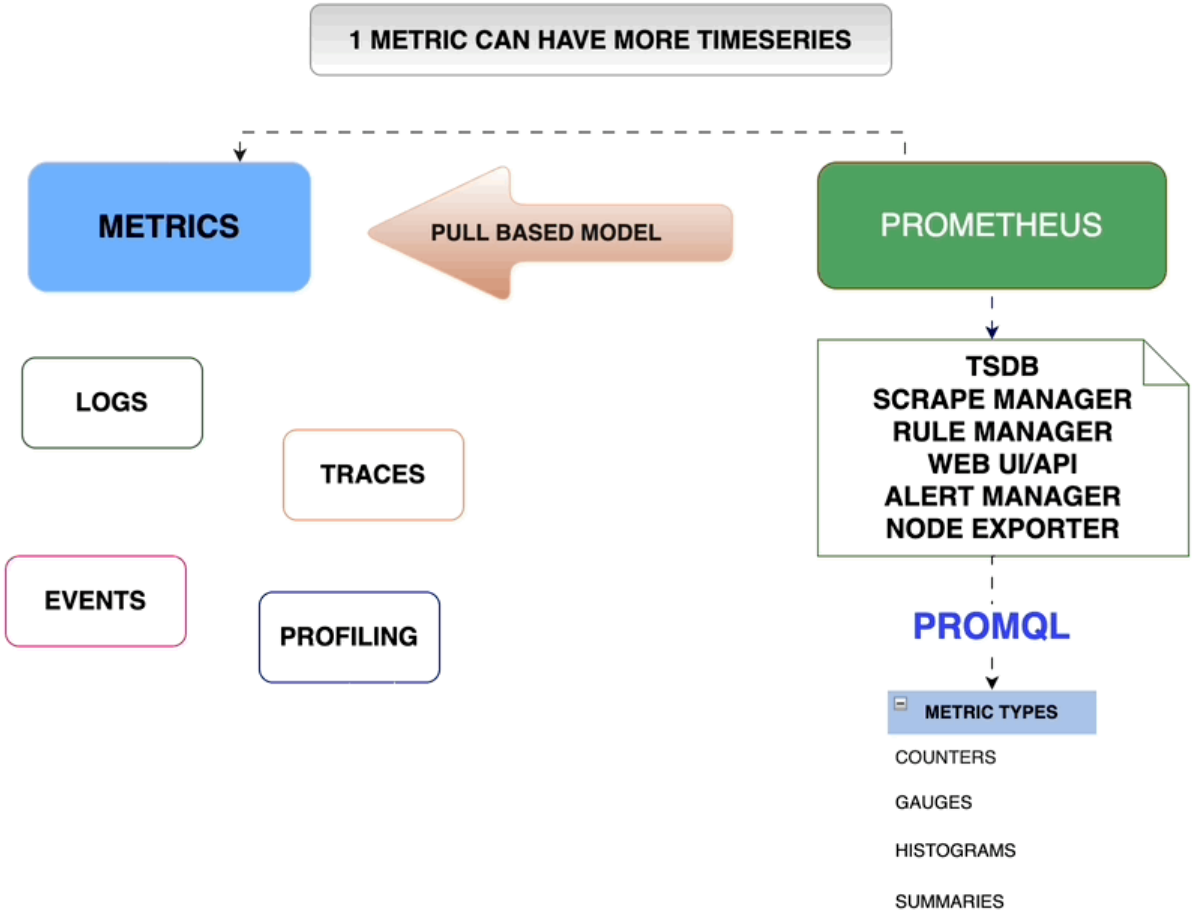

SLOW
MONITORING


CARDINALITY IS
THE NUMBER OF UNIQUE TIME SERIES

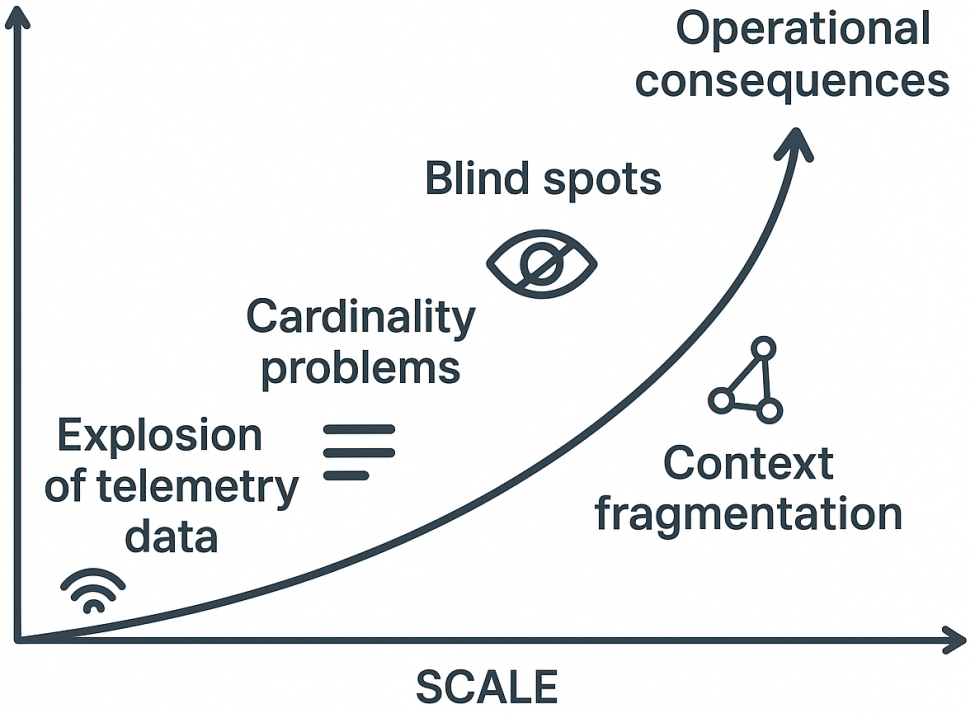

HIGH CARDINALITY MEANS
HIGH NUMBER OF
ACTIVE TIME SERIES

High cardinality impact on o11y

- Increased resource consumption (storage, memory, CPU)
- Slower query performance
- System stability risks
- Increased cost
- Alert noise and signal dilution
- Increased complexity affecting actionable insights


1 METRIC CAN HAVE MORE TIMESERIES
METRICS
PULL BASED MODEL
PROMETHEUS
LOGS
TRACES
EVENTS
PROFILING
TSDB
SCRAPE MANAGER
RULE MANAGER
WEB UI/API
ALERT MANAGER
NODE EXPORTER
PROMQL
METRIC TYPES
COUNTERS
GAUGES
HISTOGRAMS
SUMMARIES

What happens when we scale?


Operational consequences
Blind spots
Cardinality problems
Explosion of telemetry data
Context fragmentation
SCALE

High Cardinality = High Risk

- Pain points: out-of-memory errors, slow queries, alert storms

```
http_request_duration_seconds{user_id="12345", endpoint="/checkout", status="200", session_id="abcde123", pod="web-xyz"}
```

- user_id and session_id have huge cardinality
- every request generates a unique time series
- hundreds of thousands of active users
- hard to query, slow dashboards, OOM risk in Prometheus or alert manager

High cardinality root causes

- labels with a large number of unique values (user_id, request_id, url, ip)
- over-scraping/exporting
- exporters that tag too much (unique IPs, device IDs)

Over-scraping

```
# Prometheus scrape config
scrape_configs:
  - job_name: 'node'
    scrape_interval: 5s
    static_configs:
      - targets: ['node1:9100', 'node2:9100']
```

- You're over-collecting by 12x
- Increased CPU/network usage
- Your TSDB has to ingest/store redundant samples

Over-exporting

```
api_request_duration_seconds{endpoint="/search",status="200",user_id="1234",region="us-west",...}
```

SOOO many labels

- Stop! Unique labels are killing your metrics
- Unused metrics = wasted resources
- Cardinality up, performance down

Over-tagging

```
node_disk_reads_total{device="nvme0n1"}
node_disk_reads_total{device="nvme1n1"}
node_disk_reads_total{device="nvme2n1"}
```

- If all node exporter collectors are enabled each metric gets multiplied

Prometheus' limitations

- **Cardinality**: optimize with good labels, label drop, set limits on scrape jobs.
- **Long-term storage:** improve the architecture through **sharding, federation, and highly available replicas.**

Optimize Prometheus

- Manage your **scrape jitters**, they are **the most common** cause of oversize TSDB blocks.

```
$ ./oy-scrape-jitter \
--prometheus.url=http://localhost:9090 \
--log.results-only
level=info aligned_targets=2 unaligned_targets=22 max_ms=30

prometheus:
prometheusSpec:
additionalArgs:
- name: "scrape.timestamp-tolerance"
value: "30ms"

$ ./oy-scrape-jitter \
--prometheus.url=http://localhost:9090 \
--log.results-only
level=info aligned_targets=23 unaligned_targets=1 max_ms=34
```

Optimize Prometheus

- Use **pprof**, a GO native tool to check in detail Prometheus performance.

```
(pprof) top
Showing nodes accounting for 82.47MB, 67.76% of 121.71MB total
Dropped 137 nodes (cum <= 0.61MB)
Showing top 10 nodes out of 123
flat flat% sum% cum cum%
17.01MB 13.98% 13.98% 17.01MB 13.98% github.com/prometheus/
prometheus/model/labels.(*Builder).Labels
11.44MB 9.40% 23.37% 11.44MB 9.40% github.com/prometheus/
prometheus/scrape.(*scrapeCache).addRef
10.89MB 8.94% 32.32% 10.89MB 8.94% github.com/prometheus/
prometheus/scrape.newScrapePool.func1
9MB 7.40% 39.71% prometheus/tsdb/chunkenc.NewXORChunk
9MB 7.40% github.com/prometheus/
7.74MB 6.36% 46.08% Encode
7.74MB 6.36% github.com/golang/snappy.
7.71MB 6.33% 52.41% 7.71MB 6.33% github.com/prometheus/
prometheus/tsdb/index.(*MemPostings).addFor
6MB 4.93% 57.34% prometheus/tsdb.newMemSeries
9MB 7.40% github.com/prometheus/
5.19MB 4.26% 61.60% 5.19MB 4.26% github.com/prometheus/
prometheus/scrape.(*scrapeCache).trackStaleness
4MB 3.29% 64.89% (inline)
4MB 3.29% bufio.NewWriterSize
3.50MB 2.88% 67.76% 7.50MB 6.16% github.com/prometheus/
prometheus/tsdb.(*memSeries).mmapCurrentHeadChunk
```

Optimize Prometheus

- Use **promtool** for pprof data: run **promtool debug all** before restarting Prometheus.

```
promtool debug pprof http://localhost:9090
collecting: http://localhost:9090/debug/pprof/threadcreate
collecting: http://localhost:9090/debug/pprof/profile?seconds=30
collecting: http://localhost:9090/debug/pprof/block
collecting: http://localhost:9090/debug/pprof/goroutine
collecting: http://localhost:9090/debug/pprof/heap
collecting: http://localhost:9090/debug/pprof/mutex
collecting: http://localhost:9090/debug/pprof/trace?seconds=30
Compiling debug information complete, all files written in "debug.tar.
gz".
```

Optimize Prometheus

- **Use recording rules**: enabling the pre-computation of the expensive queries.

```
groups:
- name: k8s.rules
rules:
- expr: |-
sum by (cluster, namespace, pod, container) (
irate(container_cpu_usage_seconds_total{job="kubelet",
metrics_path="/metrics/cadvisor", image!=""}[5m])
) * on (cluster, namespace, pod) group_left(node) topk by
(cluster, namespace, pod) (
1, max by(cluster, namespace, pod, node) (kube_pod_
info{node!=""})
)
record: node_namespace_pod_container:container_cpu_usage_
seconds_total:sum_irate
```

Optimize Prometheus

- **Query logging** is configured via the Prometheus configuration file under the global key with the **query_log_file** setting.
  Each entry in this query log helps us find useful statistics about the query.


```
{
"params": {
"end": "2025-10-14T02:49:53.242Z",
"query": "max_over_time(reloader_last_reload_
successful{namespace=~\".+\"}[5m]) == 0",
"start": "2025-10-14T02:49:53.242Z",
"step": 0
},
"ruleGroup": {
"file": "/etc/prometheus/rules/rules_file.yaml",
"name": "config-reloaders"
},
"spanID": "0000000000000000",
"stats": {
"timings": {
"evalTotalTime": 0.000173986,
"resultSortTime": 0,
"queryPreparationTime": 0.000103703,
"innerEvalTime": 0.000063172,
"execQueueTime": 0.000017709,
"execTotalTime": 0.000197365
},
"samples": {
"totalQueryableSamples": 0,
"peakSamples": 2
}
},
"ts": "2025-10-14T02:49:53.243Z"
}
```

Optimize Prometheus

- **Query limits**: tune how many queries can be executed simultaneously and how broad those queries are allowed to be with command-line flags passed to the Prometheus at startup.
  --query.max-concurrency
  --query.max-samples

Optimize Prometheus

- **Tune garbage collection**: the Go garbage collector (GC) setting a percentage of how much the live heap size (memory) can increase from the previous garbage collection before the GC kicks in to mark and reclaim memory.

Optimize Prometheus

- Use the **GOMEMLIMIT** setting: it does not supersede GOGC and can be used in conjunction with it. It functions by providing a soft limit on the maximum amount of memory a Go program can safely use.


SHE'S LOOKING AT YOU, KID

VictoriaMetrics Solutions

Open Source

Metrics Logs Traces

Enterprise
On Premise | Cloud | Hybrid

VictoriaMetrics **Key Features**

Compatibility with Prometheus and Grafana

Compatibility with Prometheus ecosystem: Exporters, service discovery, alerting rules, query frameworks

Native Push/Pull support without limitations

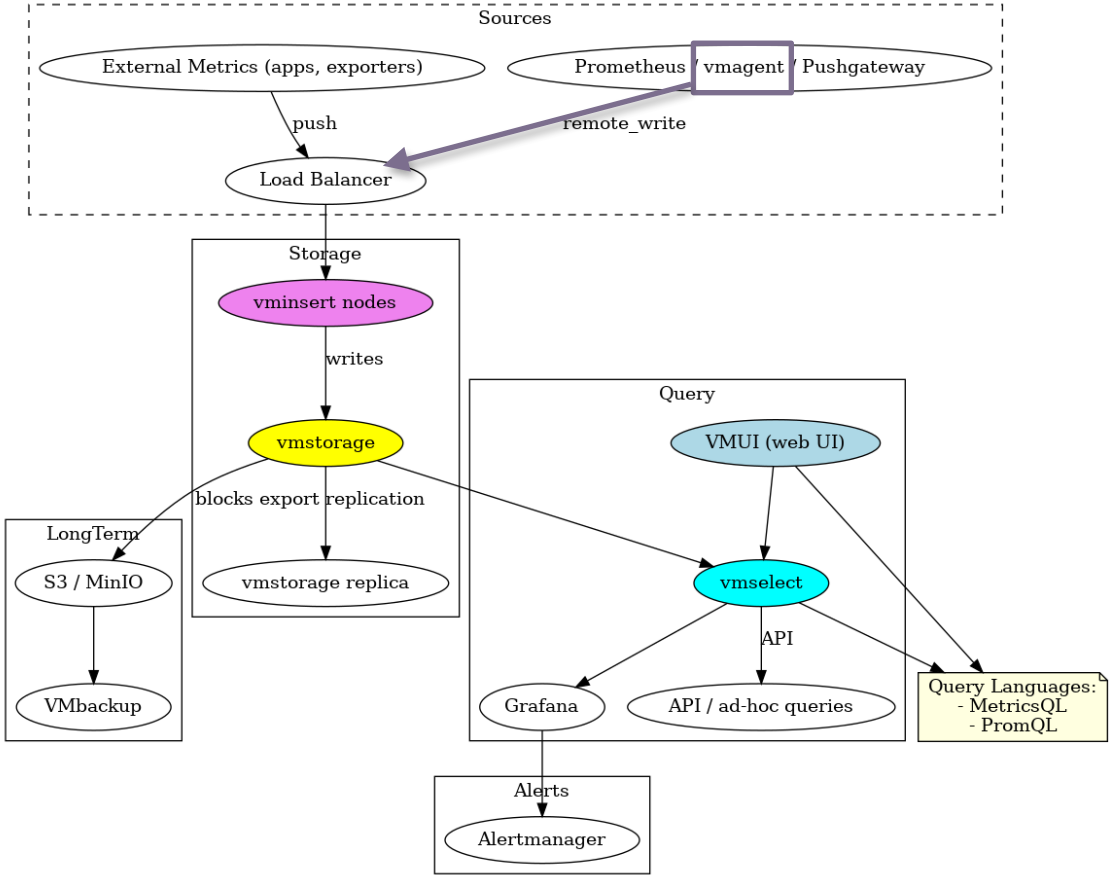
Rich features set

Clustering support with multi-tenancy

Lightweight metrics collector, best you can find

Backup/restore utilities

Auth proxy


Sources
External Metrics (apps, exporters)
Prometheus / vmagent / Pushgateway
push
remote_write
Load Balancer
Storage
vminsert nodes
writes
vmstorage
blocks export replication
vmstorage replica
LongTerm
S3 / MinIO
VMbackup
Query
VMUI (web UI)
vmselect
API
Grafana
API / ad-hoc queries
Query Languages:
- MetricsQL
- PromQL
Alerts
Alertmanager

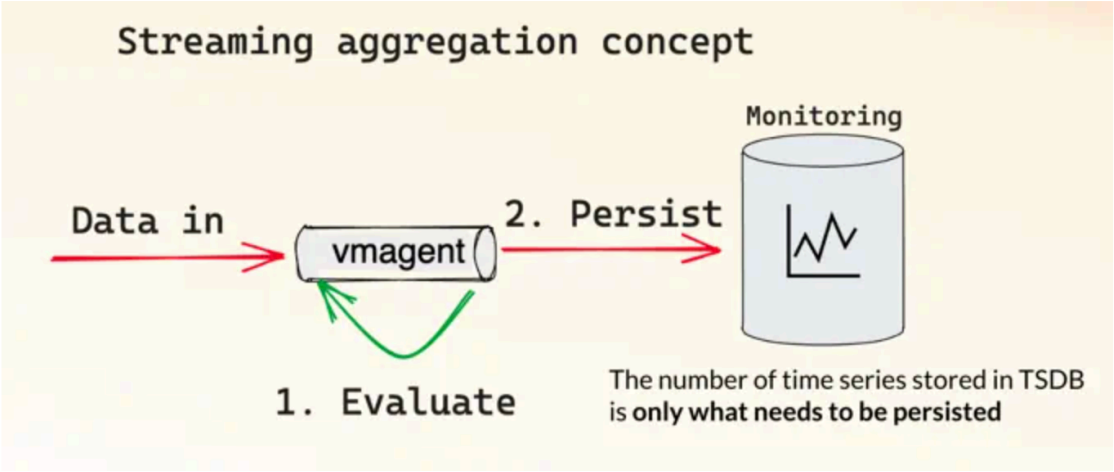
Main gains over Prometheus

Improved benchmarks

	Prometheus	VictoriaMetrics
CPU avg used	0.79/3 cores	0.76/3 cores
Disk usage	83.5 GiB	✅ 33 GiB
Memory max used	8.12/12 GiB	✅ 4.5/12 GiB
Read latency 50th	70.5ms	✅ 4.3ms
Read latency 99th %ile	7s	✅ 3.6s

Main gains over Prometheus

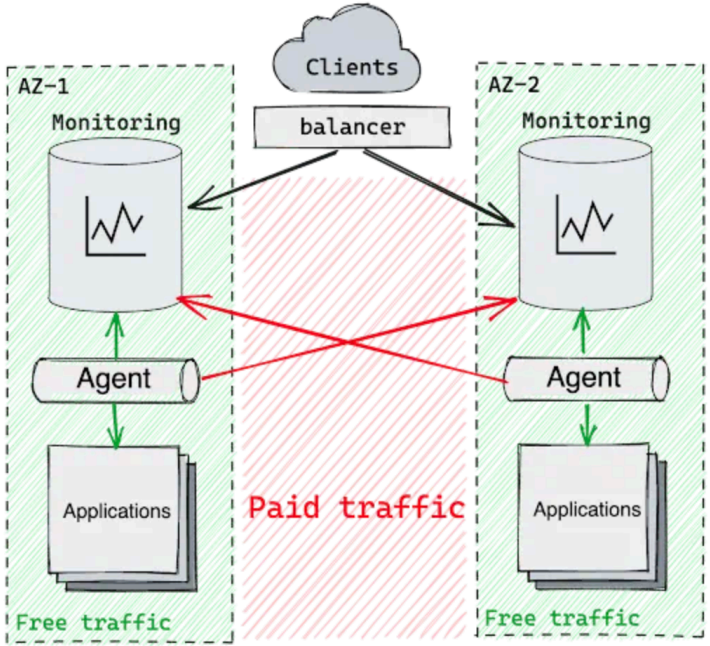
Streaming aggregation



Main gains over Prometheus

Save network cost with vmagent

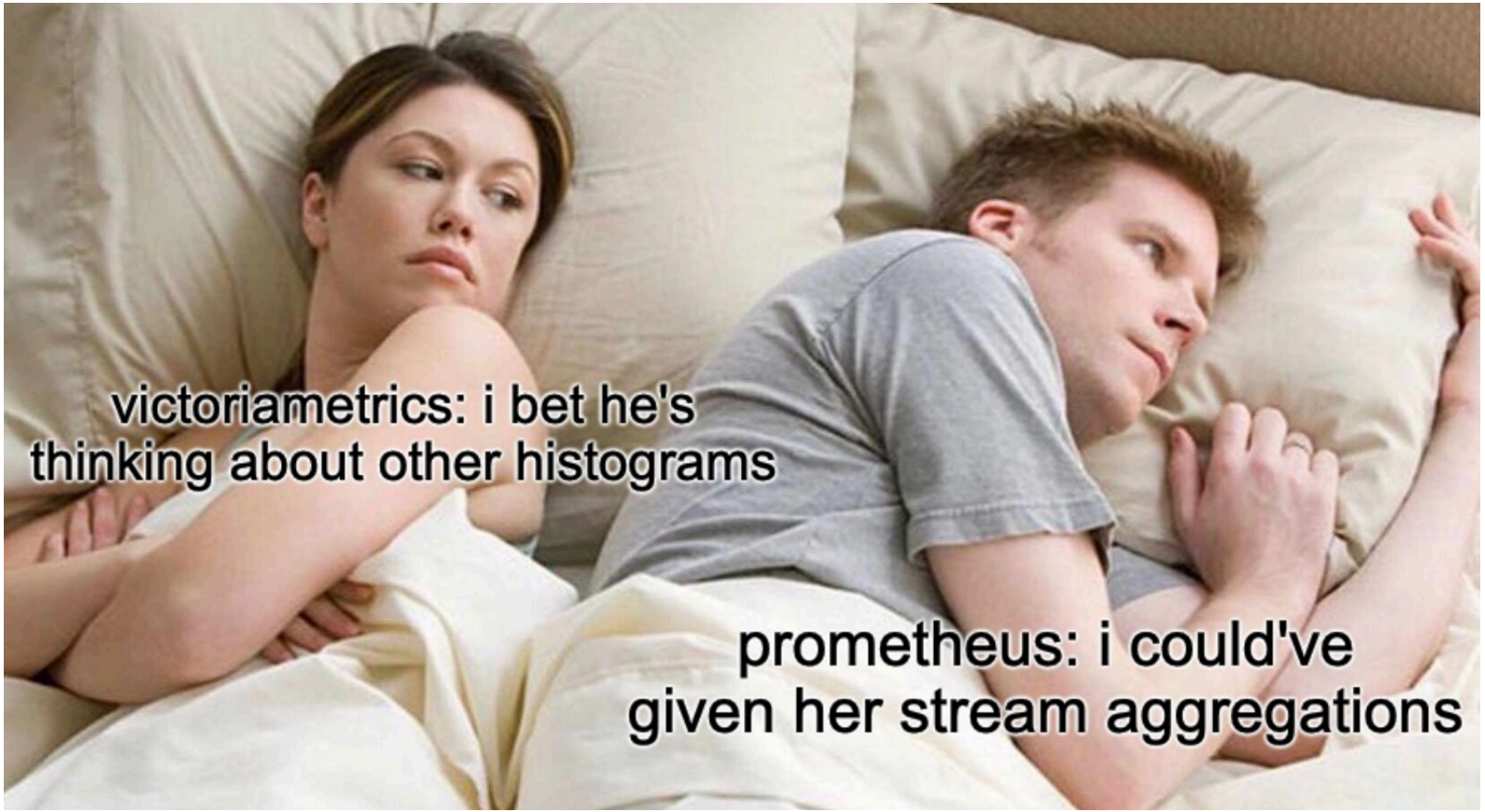
Settings	Trade-off
remoteWrite.vmProtoCompressLevel	Increased compression level, traded for higher CPU usage
remoteWrite.maxBlockSize, **remoteWrite.maxRowsPerBlock**, **remoteWrite.flushInterval**	Increased batch size leading to better compression ratio, traded for latency
remoteWrite.significantFigures, **remoteWrite.roundDigits**	Reduced precision/entropy, better compression ratio



Cardinality explorer helps identify

- Metric names with the highest number of series
- Labels with the highest number of series
- Values with the highest number of series for the selected label
- label=name pairs with the highest number of series
- Labels with the highest number of unique values

Nobody:


victoriametrics: i bet he's thinking about other histograms
prometheus: i could've given her stream aggregations

VictoriaMetrics **Query Language**

MetricsQL

- PromQL compatible, no need to change dashboards or alerting rules
- 100+ additional functions to improve querying experience
- Templating support in queries

Top 5 features

- Graphite compatible filters and Graphite support
- group_left(*) / group_right(*) with prefix support
- @ modifier for arbitrary timestamps and subexpressions
- default / if / ifnot binary operators
- `WITH` templates + string concatenation

Spot high cardinality on VictoriaMetrics clusters

- High memory usage on vmstorage – indexes for many unique series sit in memory

vmstorage

CPU usage spike on vmstorage, each unique series requires a separate TSID mapping and index entry

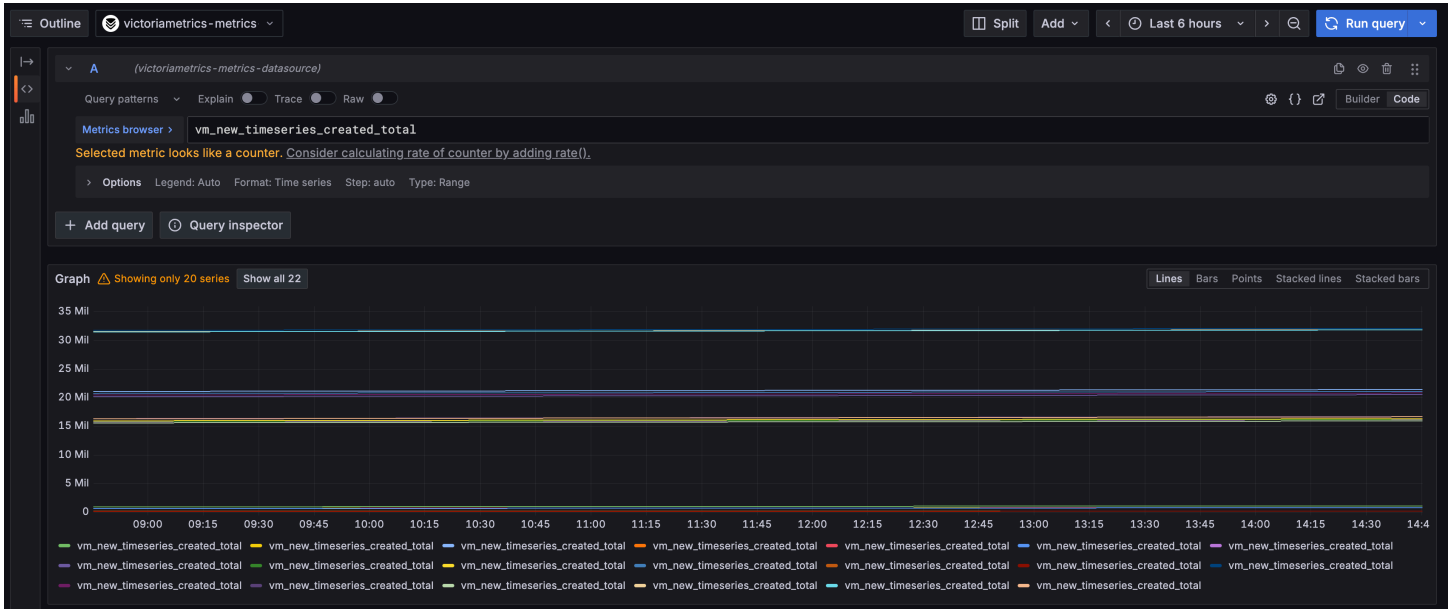
vmselect - query processing

Memory spikes on vmselect, when queries need to touch millions of series

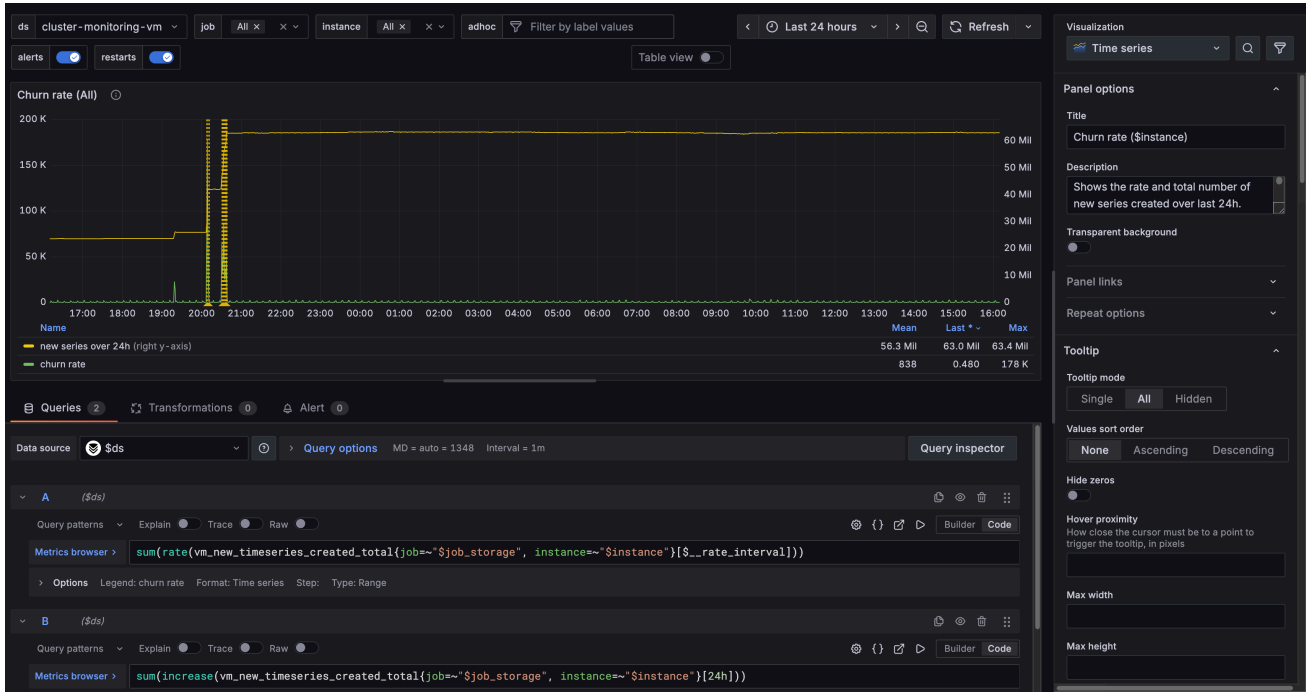
vminsert connects to multiple vmstorage nodes

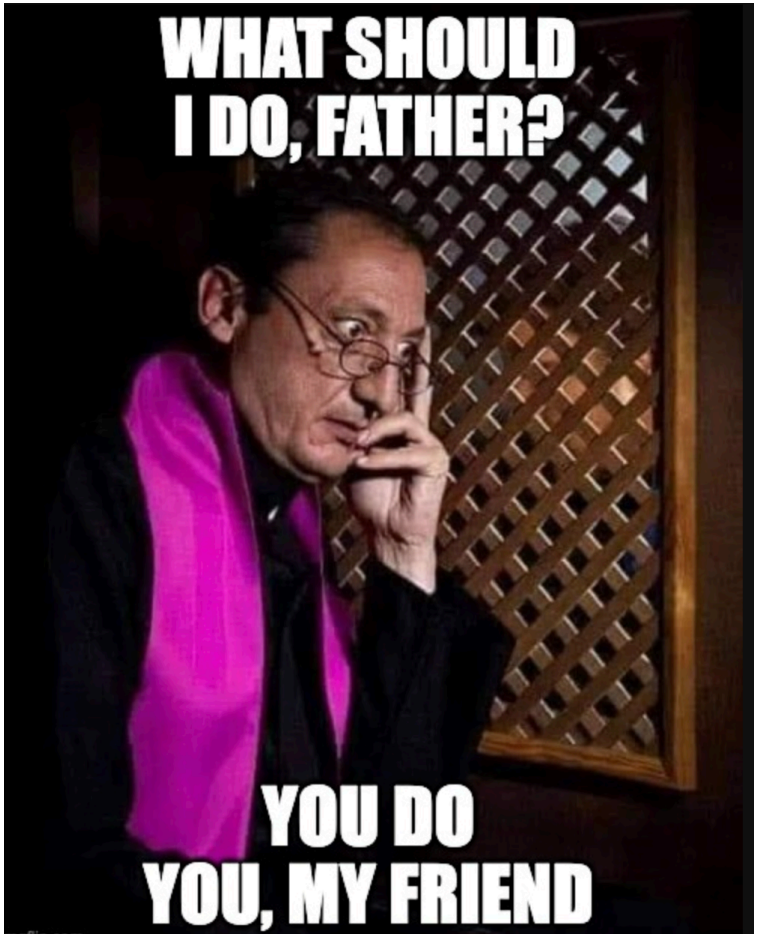
- Increased ingestion latency, vminsert takes longer to batch and deduplicate incoming series before writing

Useful queries



How to use it in a Grafana dashboard




WHAT SHOULD
I DO, FATHER?
YOU DO
YOU, MY FRIEND

Take-aways

- Reduce histogram buckets, aggregate data with functions like sum() or avg()
- Look for metrics insights in vmui's Cardinality Explorer
- If labels can’t be removed, pre-aggregate data before ingestion with stream aggregation
- Reduce query load by aggregating with MetricsQL instead of pulling raw series
- VictoriaMetrics uses optimized inverted indexes, efficient storage, aggressive compression

Resources

- https://play.victoriametrics.com/ & https://play-grafana.victoriametrics.com/
- https://github.com/VictoriaMetrics/prometheus-benchmark
- https://victoriametrics.com/blog/cardinality-explorer/
- https://medium.com/@romanhavronenko/victoriametrics-promql-compliance-d4318203f51e
- https://valyala.medium.com/prometheus-vs-victoriametrics-benchmark-on-node-exporter-metrics-4ca29c75590f
- https://github.com/VictoriaMetrics-Community/opentelemetry-demo

Explore VictoriaMetrics Solutions

VictoriaMetrics

Open Source Available on Github

VictoriaTraces

Open Source Available on Github

VictoriaLogs

Open Source Available on Github

Tussen takk

Spørsmål?

Bsky: @didiviking.bsky.social
X: @dianavtodea
Github: @didiViking/Conferences_Talks
LinkedIn: @diana-todea-b2a79968

VictoriaMetrics Community