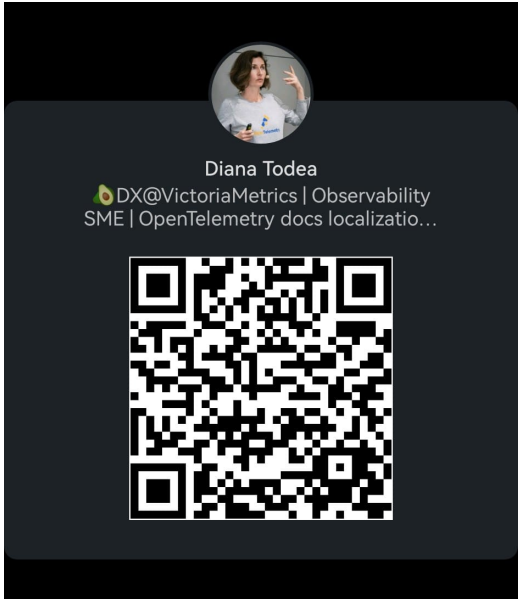
Beyond Prometheus: pushing the boundaries of scalable monitoring with VictoriaMetrics

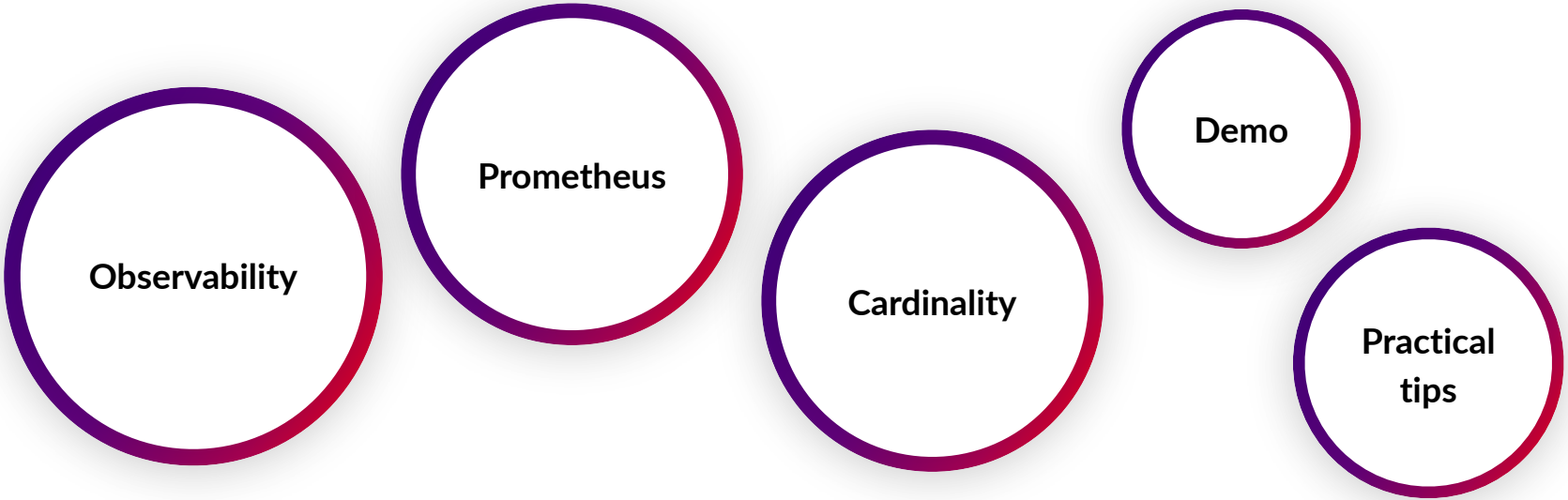
Criteo, Paris | 15 October 2025

Diana Todea - DX Engineer

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



Agenda

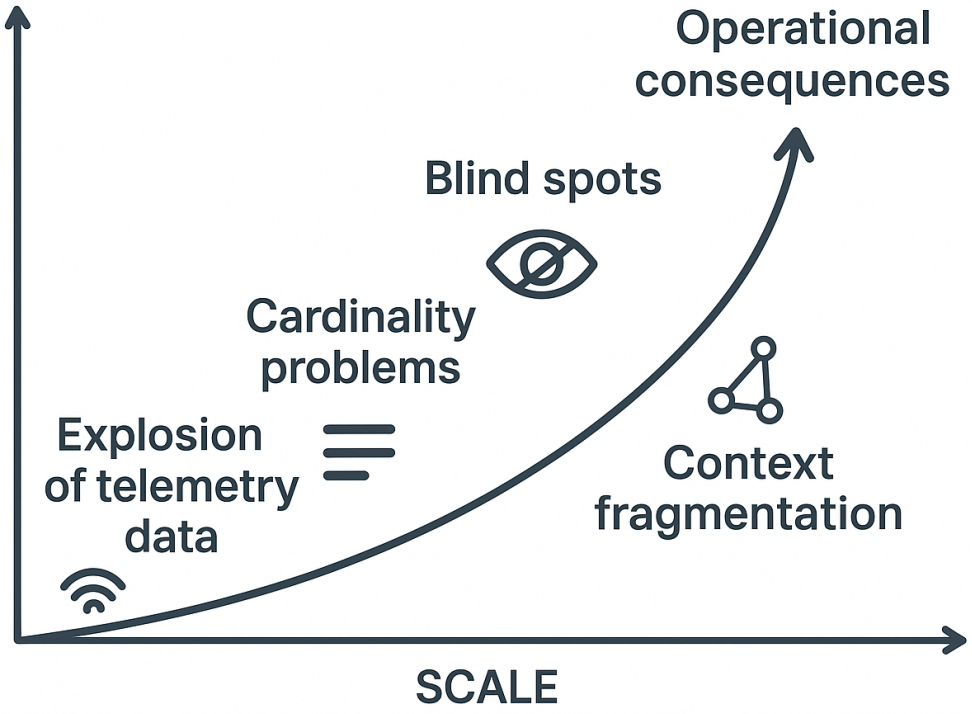

Observability
Prometheus
Cardinality
Demo
Practical tips

+

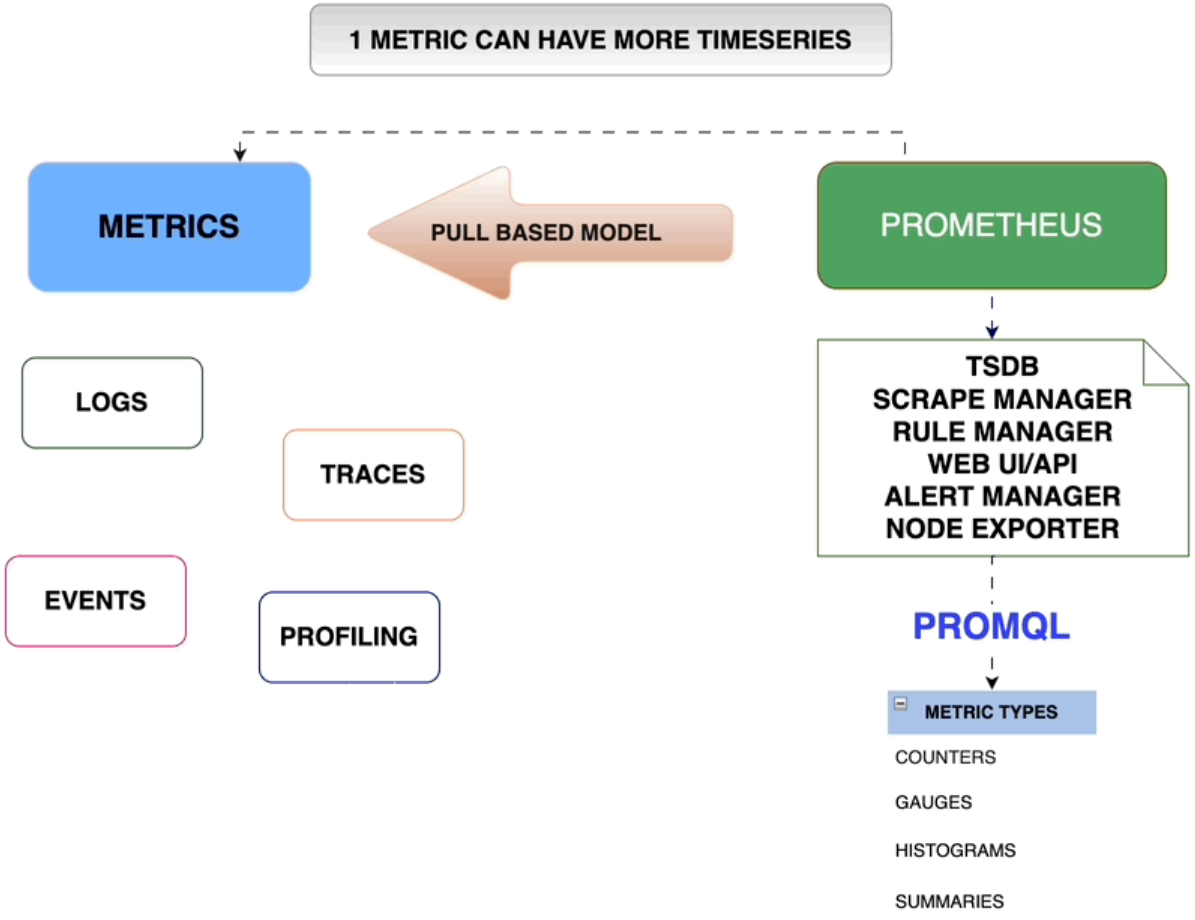
=

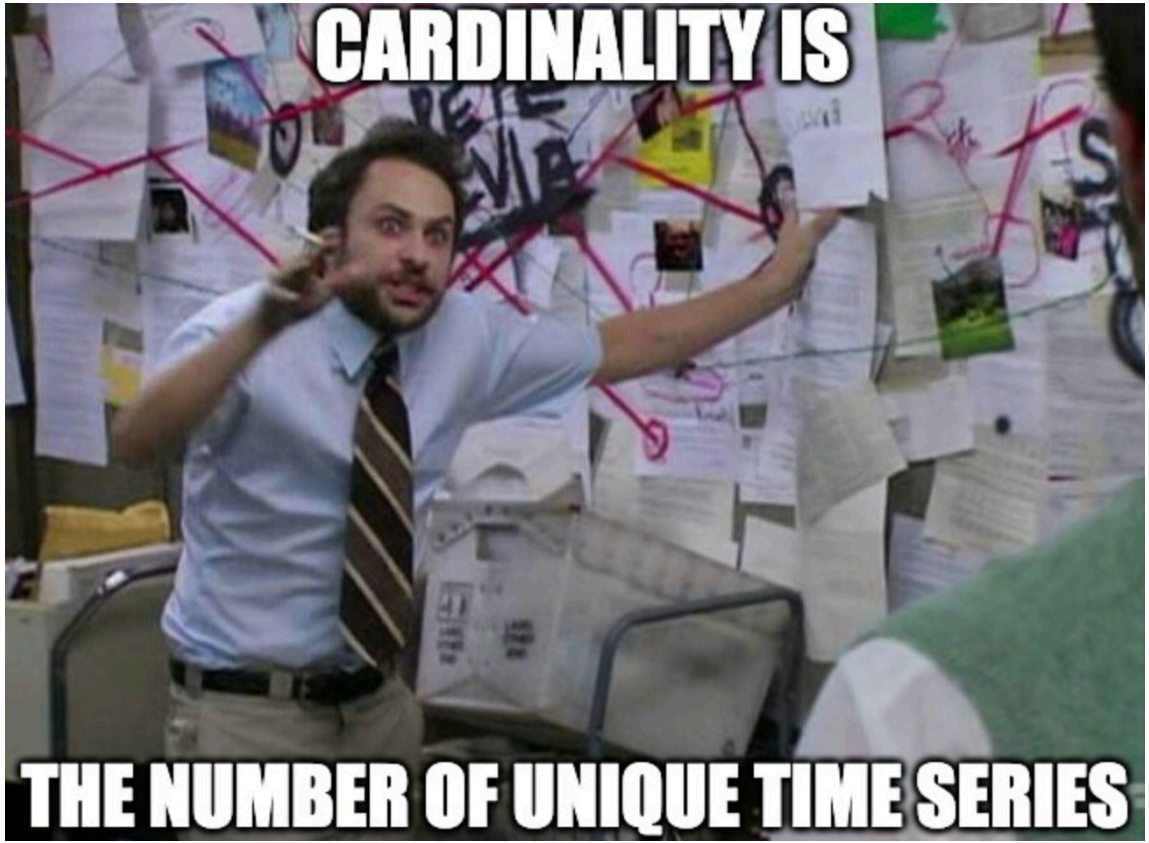
Actionable Insights

What happens when we scale?


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

Collect the right observability signals


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


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

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",...}
```

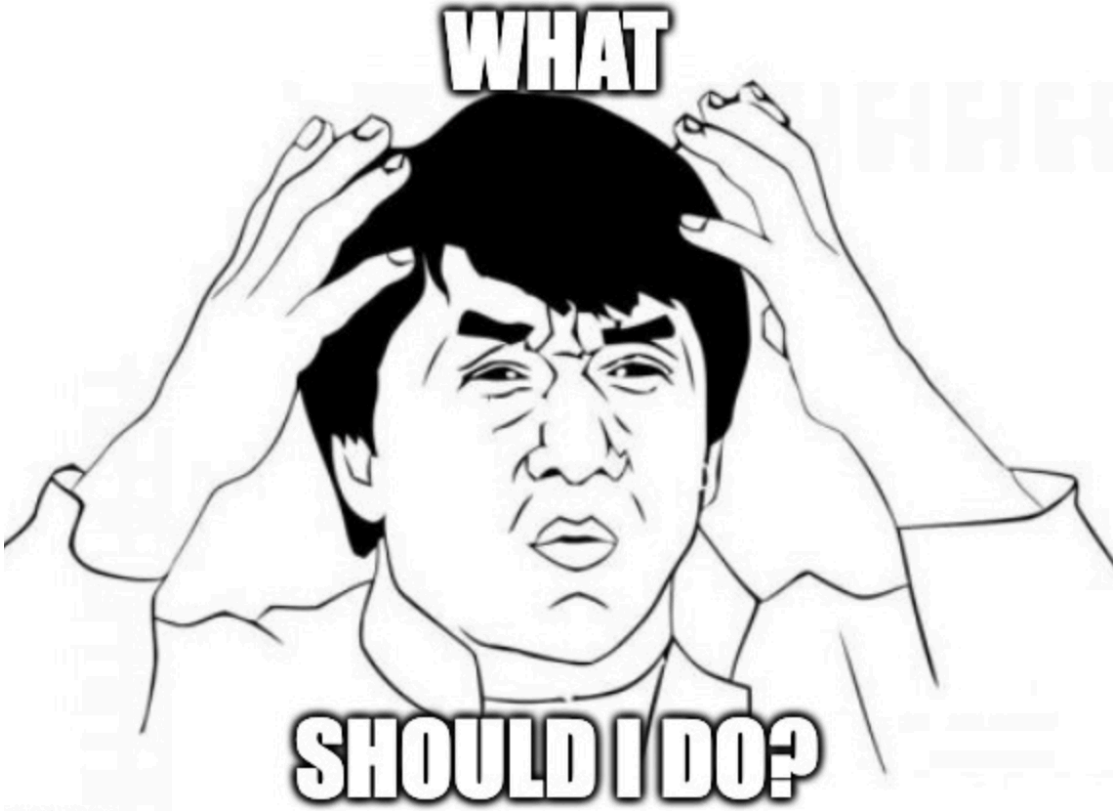
SOOO many labels

- 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


WHAT
SHOULD I DO?

Scrape strategies

- Don't scrape faster than metrics change

```
scrape_interval: 60s
```

- Drop unneeded labels

```
relabel_configs:
  - source_labels: [__name__, user_id]
    regex: api_request_duration_seconds;.*
    action: labeldrop
```

Use pre-aggregated views

- Aggregate early and export less (stream aggregation)
- Summarize with histograms, not IDs

```
api_request_duration_seconds_bucket{le="0.1", endpoint="/search"}
```

Instance grouping & job deduplication

```
- job_name: app-v1
- job_name: app-v2
```

```
- job_name: app
  relabel_configs:
    - source_labels: [__meta_kubernetes_label_app_version]
      target_label: version
```

Cut the noise at the source

```
metric_relabel_configs:
  - source_labels: [__name__]
    regex: "go_gc_.*"
    action: drop
```

(*if not used)

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

VictoriaMetrics **Query Language**

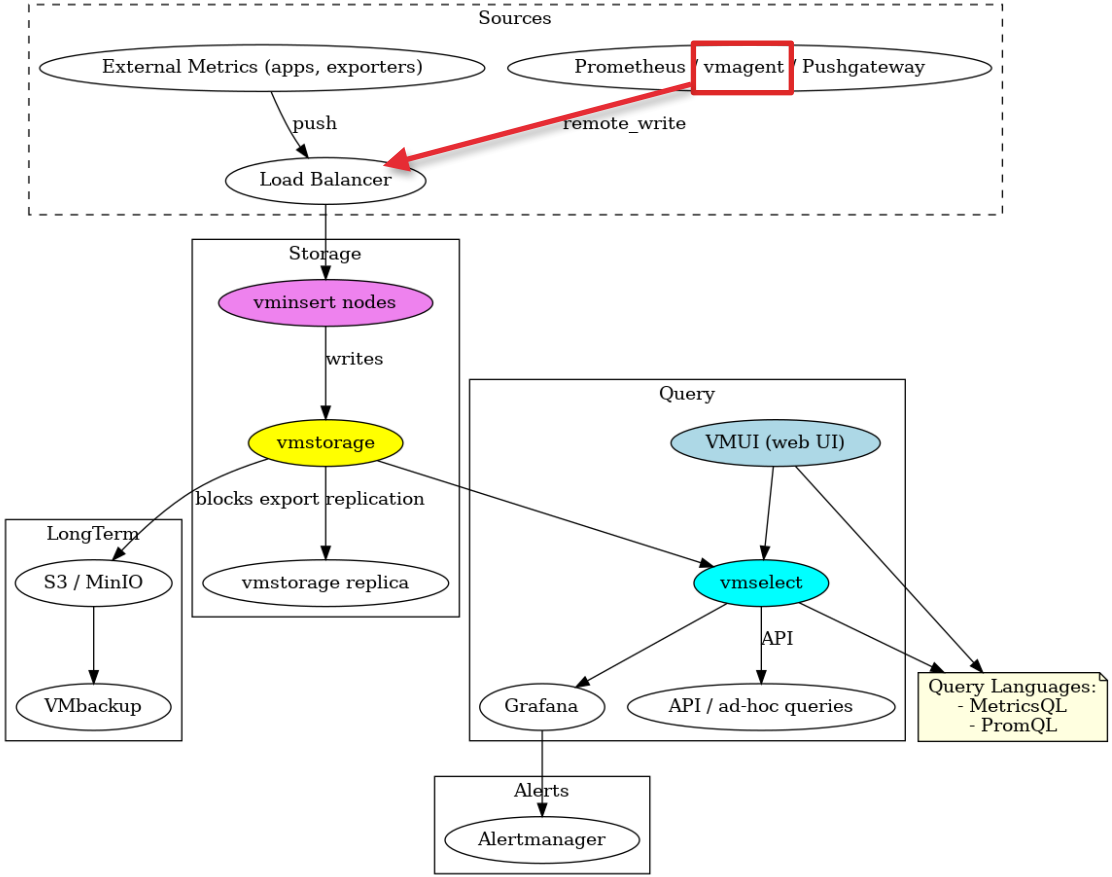
MetricsQL

- 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


APPRENEZ
VICTORIAMETRICS


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

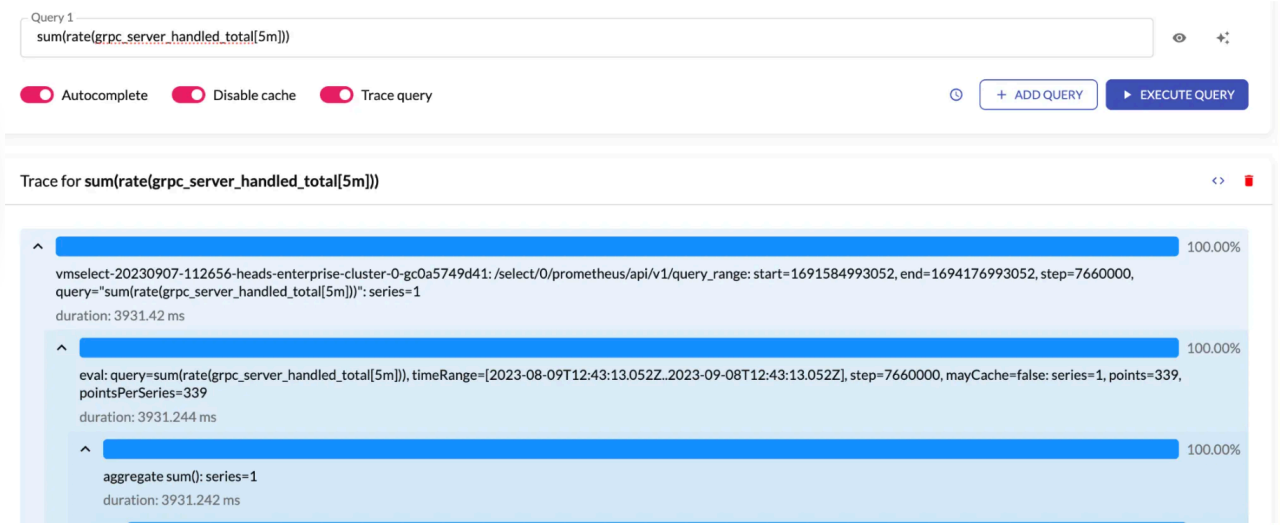
Main gains over Prometheus

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

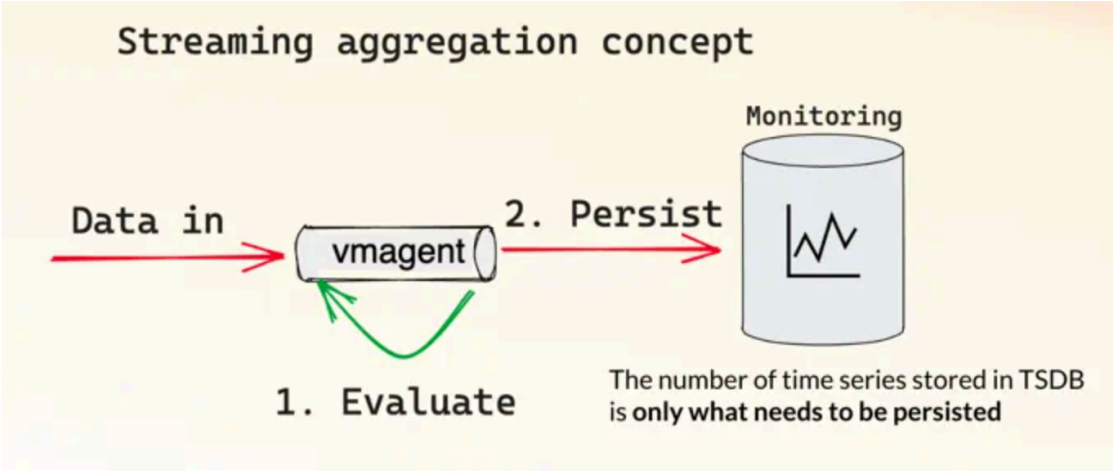
Main gains over Prometheus

Query tracing to find bottlenecks



Main gains over Prometheus

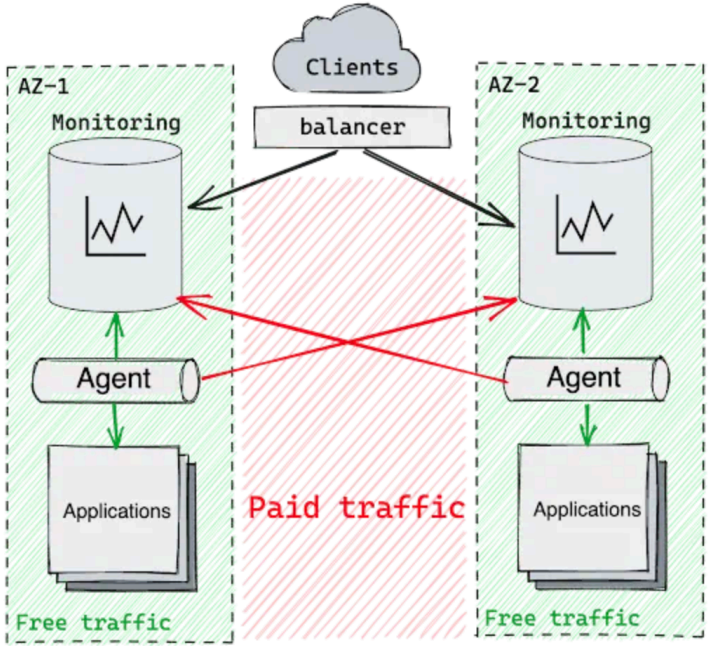
Streaming aggregation



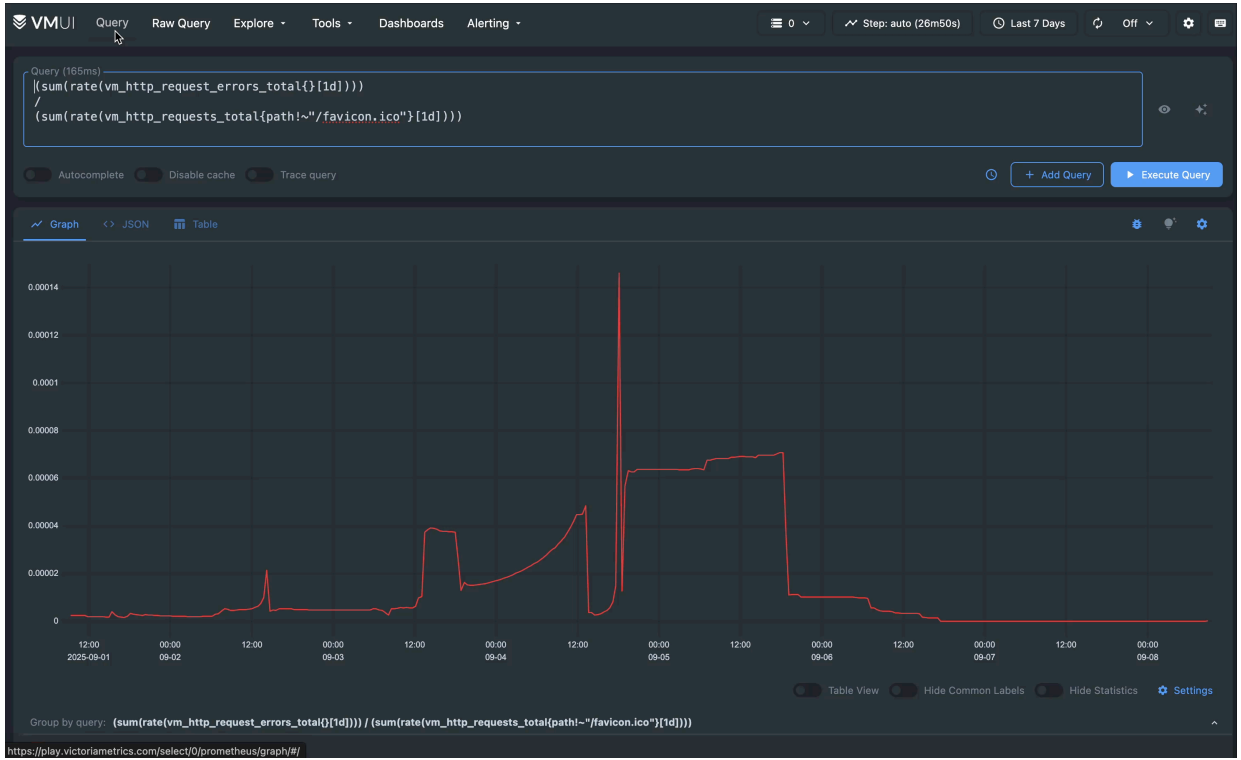
Main gains over Prometheus

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



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

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

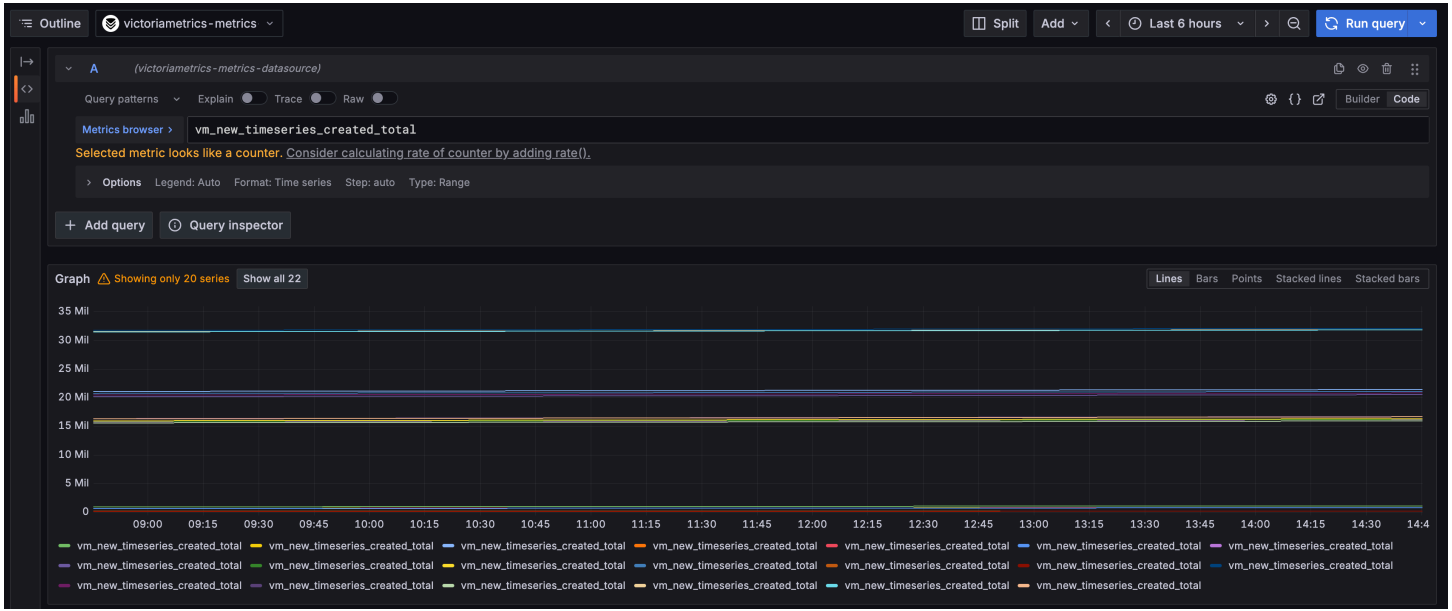
vmselect - query processing

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

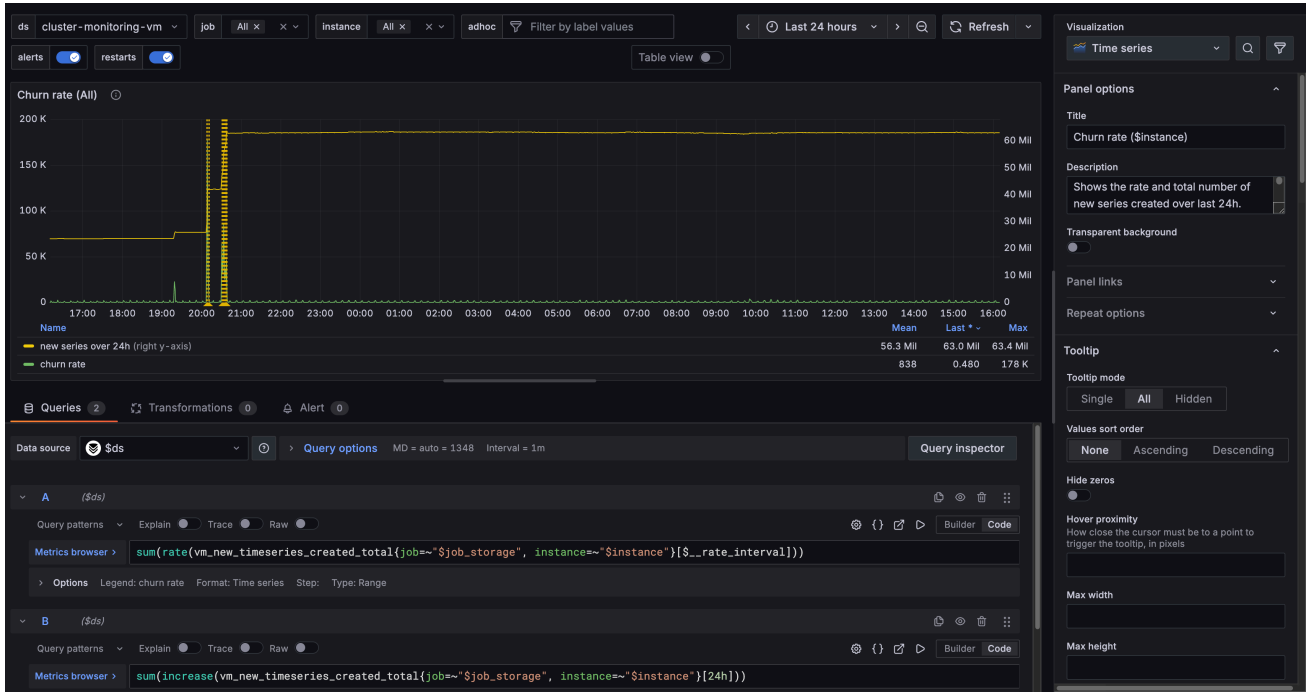
vminsert connects to multiple vmstorage nodes

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

Useful queries



How to use it in a Grafana dashboard



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
- Correlate your insights with your scrape jobs, dashboards and use cases

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

More on **VictoriaMetrics** Solutions

VictoriaMetrics

Open Source Available on Github

VictoriaLogs

Open Source Available on Github

VictoriaMetrics Enterprise

Commercial Licence

VictoriaMetrics Cloud

Pay-as-you-go

Merci d’avoir écouté!

Avez-vous des questions?

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

VictoriaMetrics Community