Build Fast, Scalable App Monitoring with Open Source

Robert Hodges - Altinity
Roman Khavronenko - VictoriaMetrics

Altinity

VICTORIA METRICS

Let's make some introductions

Robert Hodges

Database geek with 30+ years on DBMS systems. Day job: CEO at Altinity

Roman Khavronenko

Distributed systems and monitoring engineer. Day job: SE at VictoriaMetrics

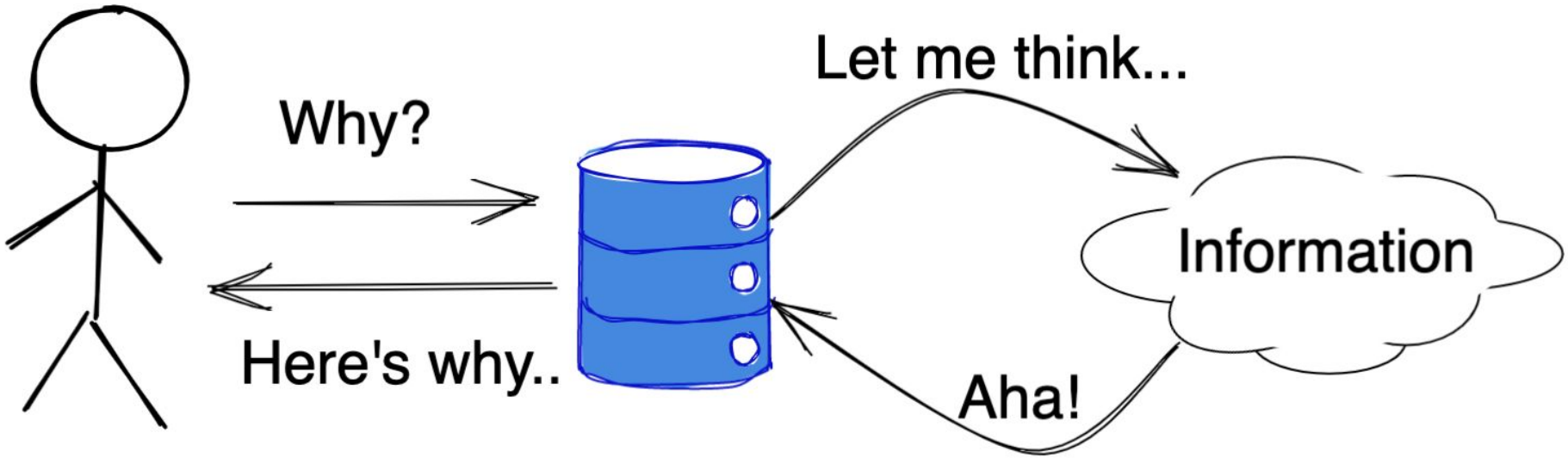
What is application monitoring?

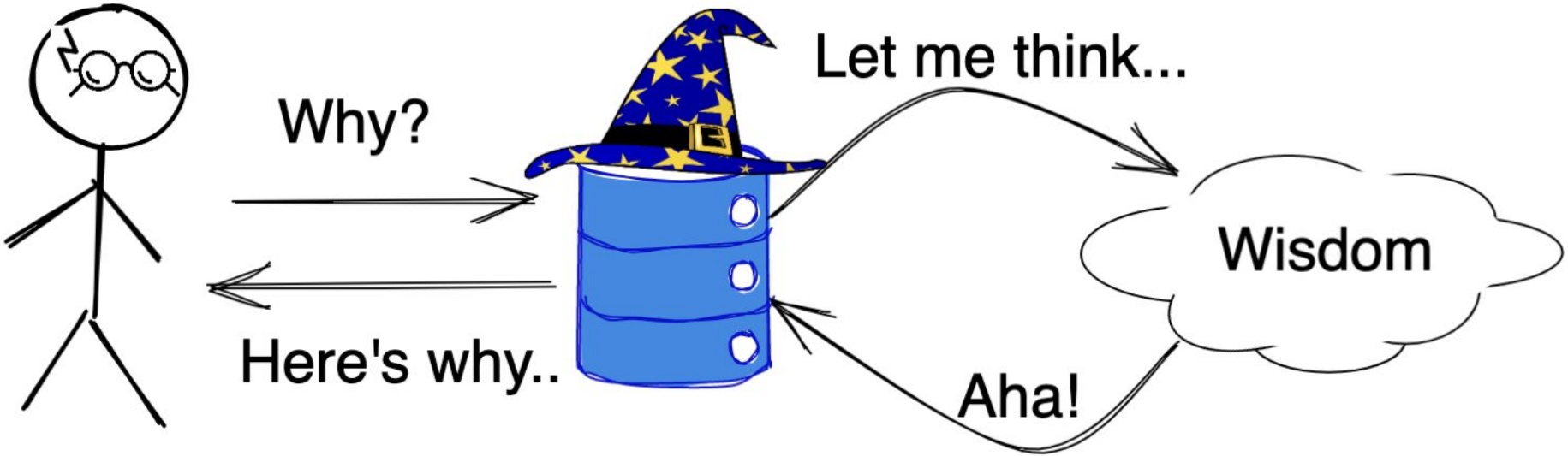
Monitoring is for answering questions

- Why users are getting errors?
- When it started?
- How many users are affected?
- Which service is failing?

To get an answer to the question you need 3 things

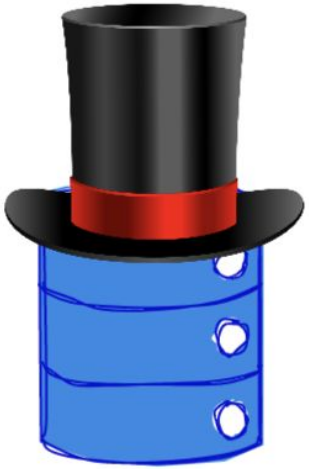
1. The question
2. The information to process
3. The respondent


Why?
Here's why..
Let me think...
Information
Aha!


Why?
Let me think...
Wisdom
Here's why..
Aha!

Too young...

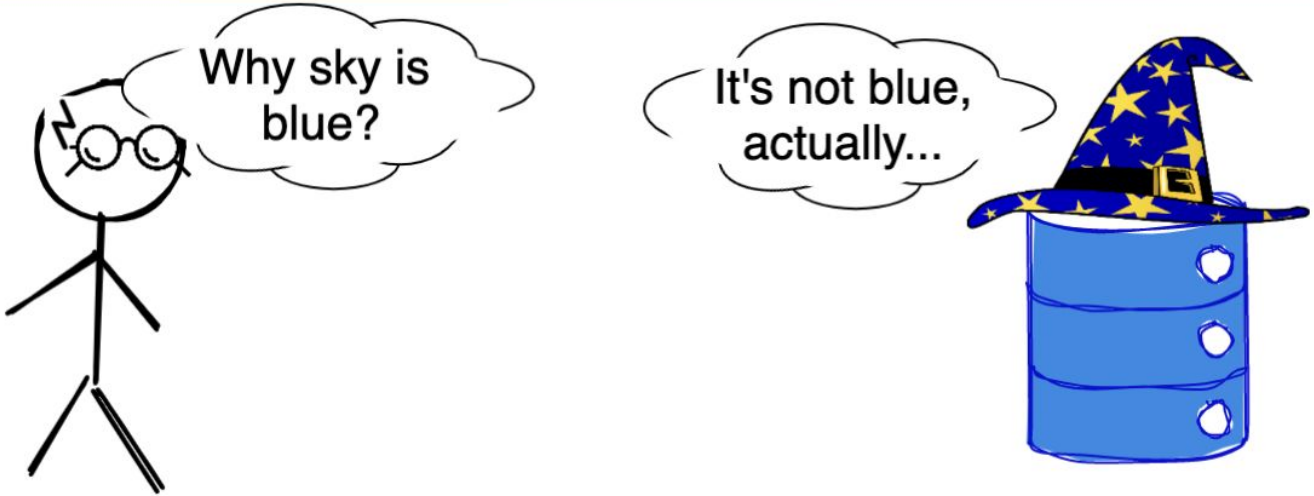
Too old...

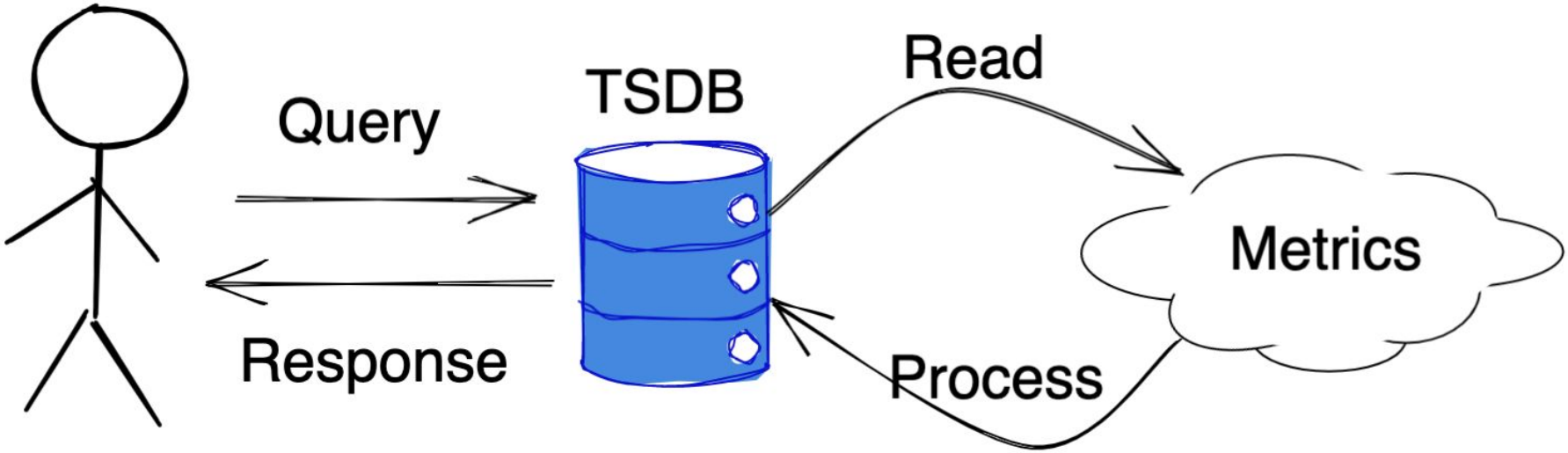


Too pricy...

Just right!

#0173BD #51A5DE #C0E8F7 #FDD0E1 #FFB4CE




Query
TSDB
Read
Metrics
Response
Process

Using VictoriaMetrics

VictoriaMetrics - Open Source Time Series Database & Monitoring Solution

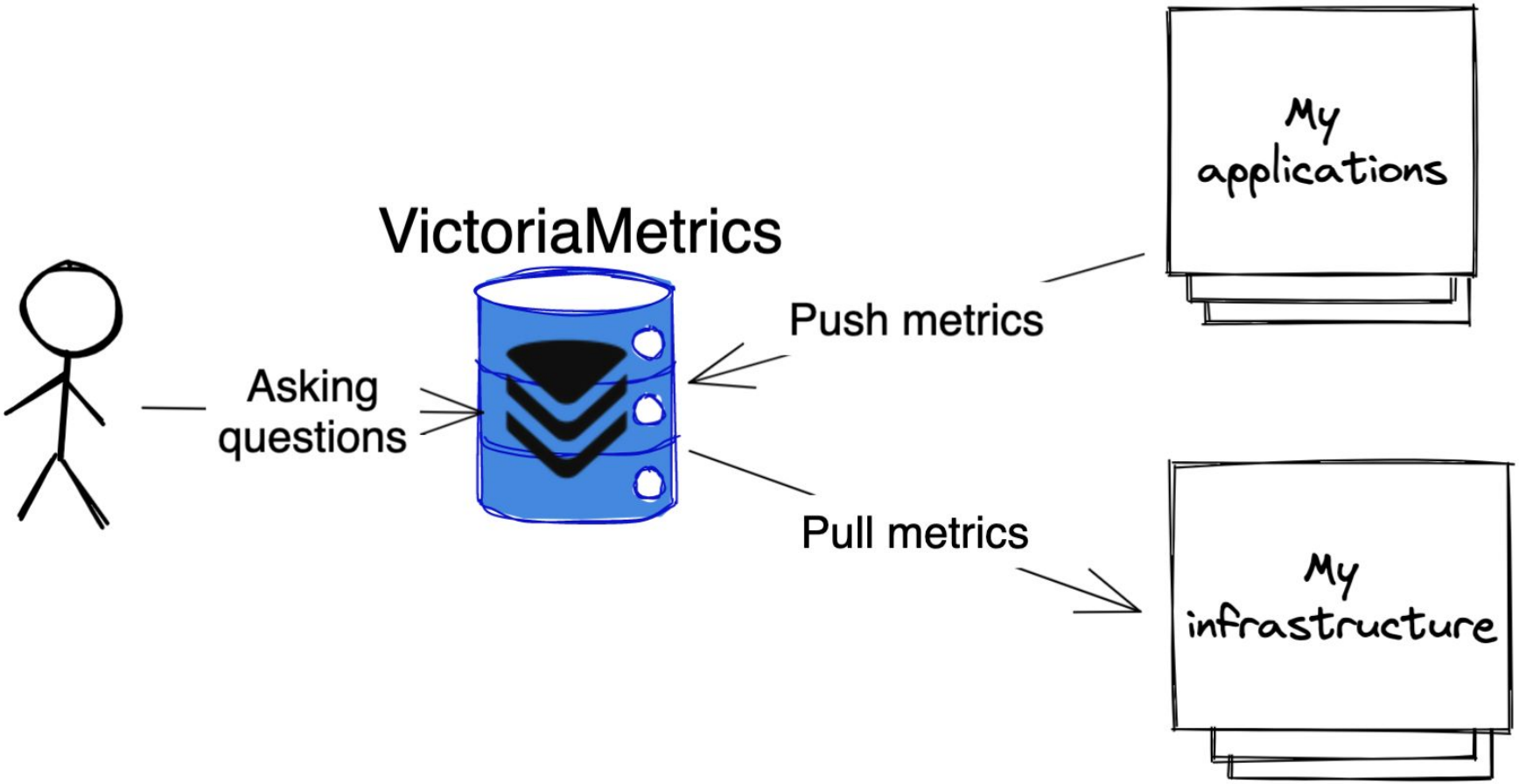
- Vertically and horizontally scalable
- Operational simplicity
- Cost-efficient
- Prometheus compatible
- Free forever

VictoriaMetrics

VictoriaMetrics - Open Source Time Series Database & Monitoring Solution

- Kubernetes monitoring
- Hardware and infrastructure monitoring
- Application Performance Monitoring (APM)
- IoT
- Edge computing
- Alerting

VictoriaMetrics

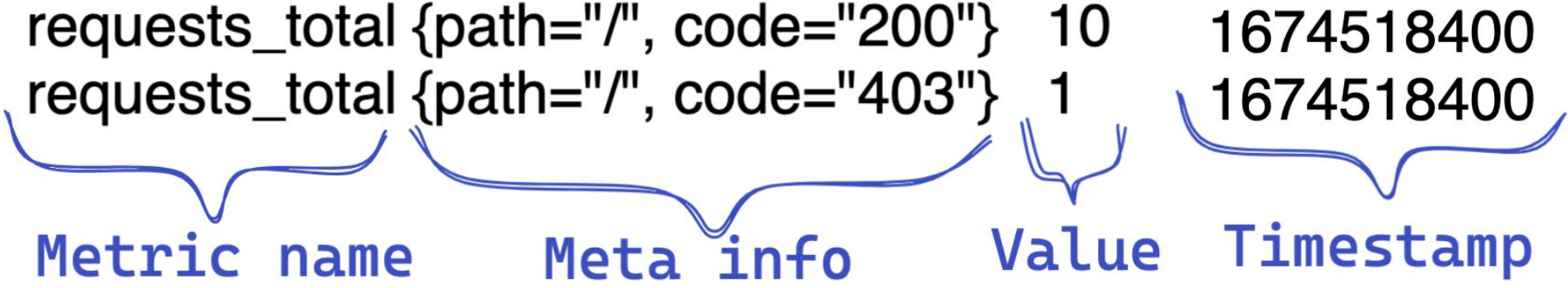

VictoriaMetrics
My applications
Push metrics
Asking questions
Pull metrics
My infrastructure

What is a metric?

Metric is a numeric measure or observation of something:

- Number of served requests
- Requests latency
- CPU or memory usage
- Occupied or free disk space

Metrics structure



Storage for metrics

- VictoriaMetrics data model is schemaless
- No need to define metric names or their labels in advance
- User is free to add or change ingested metrics anytime.

```
request_errors_total {reason="unsupported"}   0   1674518400
requests_total {path="/pprof/"}               1   1674518400
requests_total {path="/pprof/cmdline"}        0   1674518400
user_sessions                                 5   1674518400
app_version{version="v1.1.18"}                1   1674518400
```

Storage for metrics

samples (series_id Uint64, timestamp Int64, value Float64)
ORDER BY (series_id, timestamp)

series_id	value	timestamp
< time series ID >	0	1674518400
< time series ID >	1	1674518400
< time series ID >	0	1674518400
< time series ID >	5	1674518400
< time series ID >	1	1674518400

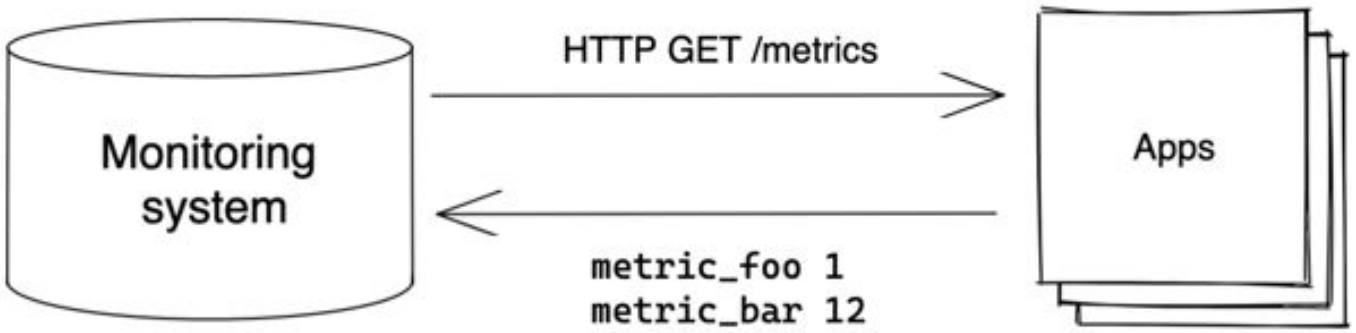
samples (value Float64 Codec(Gorilla), timestamp Int64 Codec(DoubleDelta))

OSA Con 2021: How ClickHouse Inspired Us to Build a High Performance TSDB

- VictoriaMetrics is specialized solution for time series data
- Compression reaches 0.4 Bytes per sample
- Ingestions speed 300k samples/s per CPU core
- Scanning speed 50Mil samples/s per CPU core

Pull model

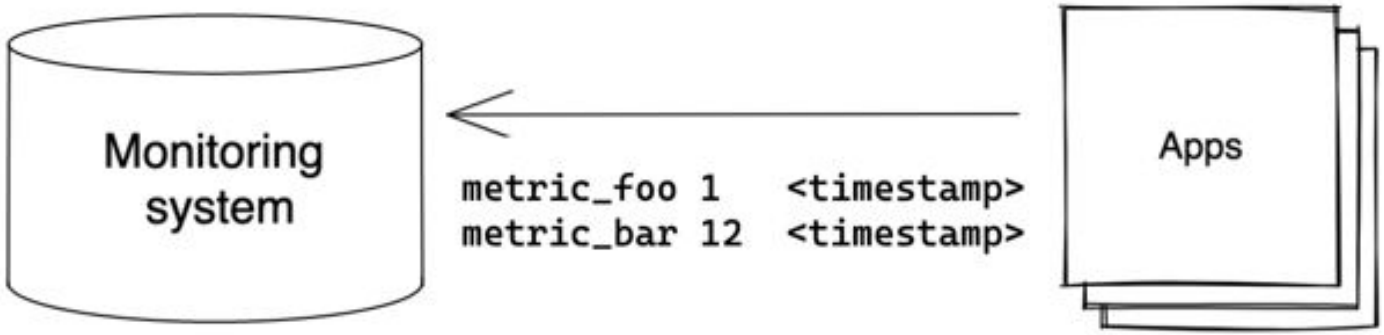
Monitoring system decides when
and where to get metrics




```
> curl https://my.application/metrics
    requests_total{path="/",code="200"}10
    requests_total{path="/",code="240300"}1
```

Push model

Application decides when
and where to send metrics




```
> curl -d "requests_total{path="/",code="200"} 10" -X POST
http://victoriametrics/api/v1/import/prometheus
```

More than one protocol for metrics

- Prometheus remote write API.
- Prometheus text exposition format.
- DataDog protocol.
- InfluxDB line protocol over HTTP, TCP and UDP.
- Graphite plaintext protocol with tags.
- OpenTSDB put message.
- HTTP OpenTSDB /api/put requests.
- JSON line format.
- Arbitrary CSV data.

Querying via MetricsQL

OSA Con 2022

Specifics of data analysis in Time Series Databases

Time series data is special. Not only its nature but also the ways that we store and interact with it. In this talk, we'll cover the differences between storing time series data in classic relational databases
and a new generation of time series databases like VictoriaMetrics and Prometheus.

Presentation Slides

Querying via MetricsQL

```
> requests_total
  requests_total {path="/", code="200"}   10
  requests_total {path="/", code="403"}   1
```

```
> requests_total{code="200"}
  requests_total {path="/", code="200"}   10
```

```
> sum(requests_total) by(path)
  {path="/"}   11
```

Demo time!

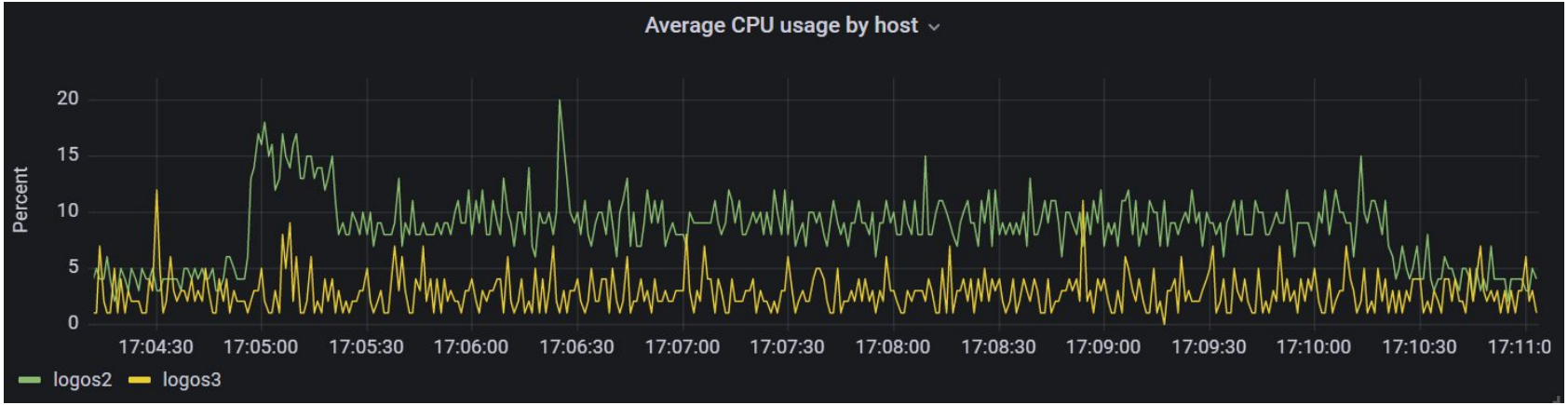
- Run VictoriaMetrics
- Write some metrics
- Execute read queries

Frequently asked questions

- Can I monitor MySQL Server, Postgres, MongoDB, ClickHouse?
  - Yes, there are plenty of exporters, dashboards and alerting rules there.
- Can I monitor my applications?
  - Yes, there are libraries for multiple programming languages to instrument the application with metrics.
- How expensive monitoring is?
  - With VictoriaMetrics, cost of storing metrics from 100 instances, each instance emits 1000 metrics every 30s for the total cost will be:
    - 100GB of disk space $0.045 per GB-month: 100*0.045*12 = $54
    - One t3.medium instance, $0.0418 per hour: 0.0418*730*12 = $366
    - Total: $420 per year for monitoring 100 instances.
- Can I run it in Kubernetes?
  - Sure! We have k8s operator and helm charts for VictoriaMetrics!

Using ClickHouse

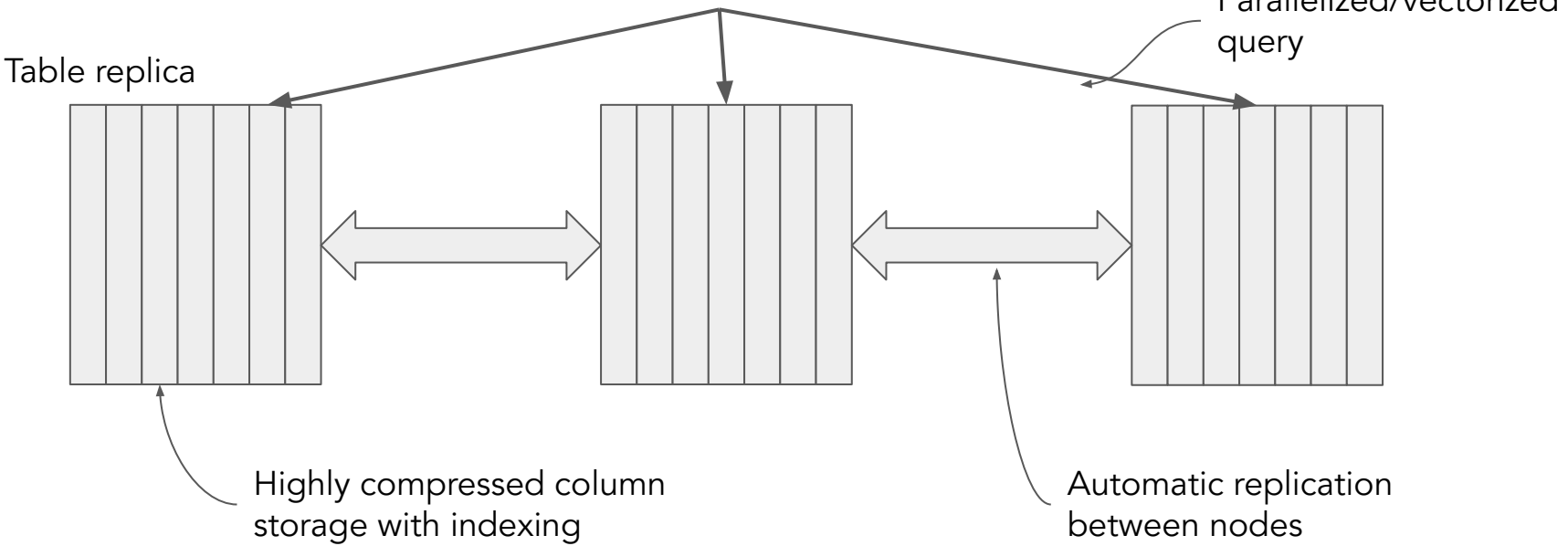
ClickHouse: a real-time analytic database



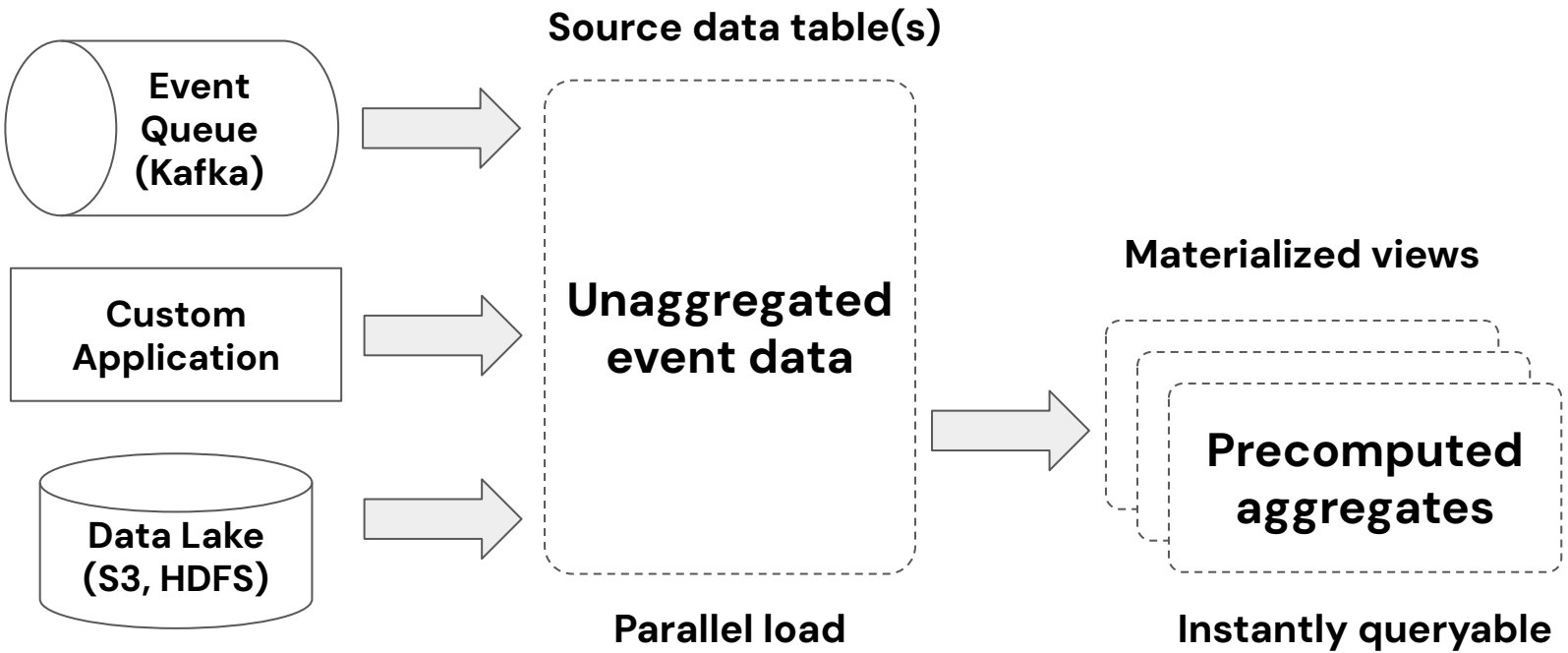
It understands SQL
It's Apache 2.0
It handles many use cases beyond monitoring
It also handles time series data very well

ClickHouse optimizes for fast response on large datasets

```
SELECT host, avg(idle)
FROM vmstat GROUP BY host
```



ClickHouse can load millions of events per second



…And supports [many] dozens of input formats

```
INSERT INTO some_table Format <format>

        TabSeparated
        TabSeparatedWithNames
        CSV
        CSVWithNames
        CustomSeparated
        Values
        JSON
        JSONEachRow
        Protobuf
        Parquet
        ...
```

There are many ways to store and manipulate time data

Date -- Precision to day

DateTime -- Precision to second

DateTime64 -- Precision to nanosecond

BI tools like Grafana like DateTime values

toYear(), toMonth(), toWeek(), toDayOfWeek, toDay(), toHour(), ...

toStartOfYear(), toStartOfQuarter(), toStartOfMonth(), toStartOfHour(), toStartOfMinute(), …, toStartOfInterval()

toYYYYMM()

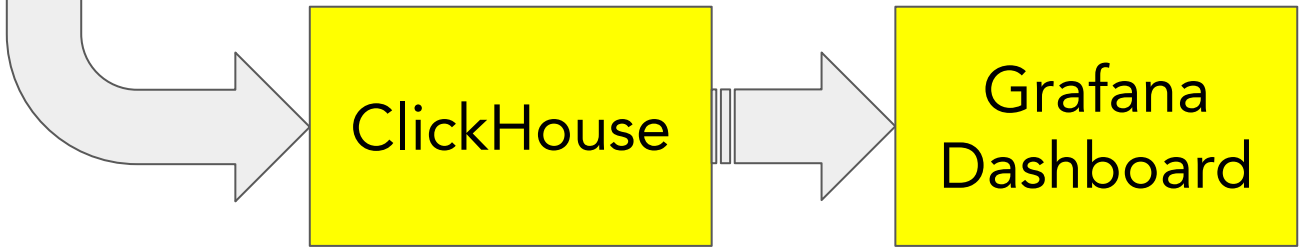
toYYYYMMDD()

toYYYYMMDDhhmmsss()

And many more!

Let's build a simple host monitoring system

```
$ vmstat 1 -n
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b    swpd    free    buff  cache    si    so    bi    bo    in    cs us sy id wa st
 0  0 166912 2645740   36792 3360652    0     0     3   101     1     1  2  1 98  0  0
 1  0 166912 2645360   36792 3360652    0     0     0     0 1182 3986  7  1 93  0  0
```



Step 1: Generate vmstat data

```
#!/usr/bin/env python3
import datetime, json, socket, subprocess
host = socket.gethostname()
with subprocess.Popen(['vmstat', '-n', '1'], stdout=subprocess.PIPE) as proc:
    proc.stdout.readline() # discard first line
    header_names = proc.stdout.readline().decode().split()
    values = proc.stdout.readline().decode()
    while values != '' and proc.poll() is None:
        dict = {}
        dict['timestamp'] = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        dict['host'] = host
        for (header, value) in zip(header_names, values.split()):
            dict[header] = int(value)
        print(json.dumps(dict), flush=True)
        values = proc.stdout.readline().decode()
```

Here's the output

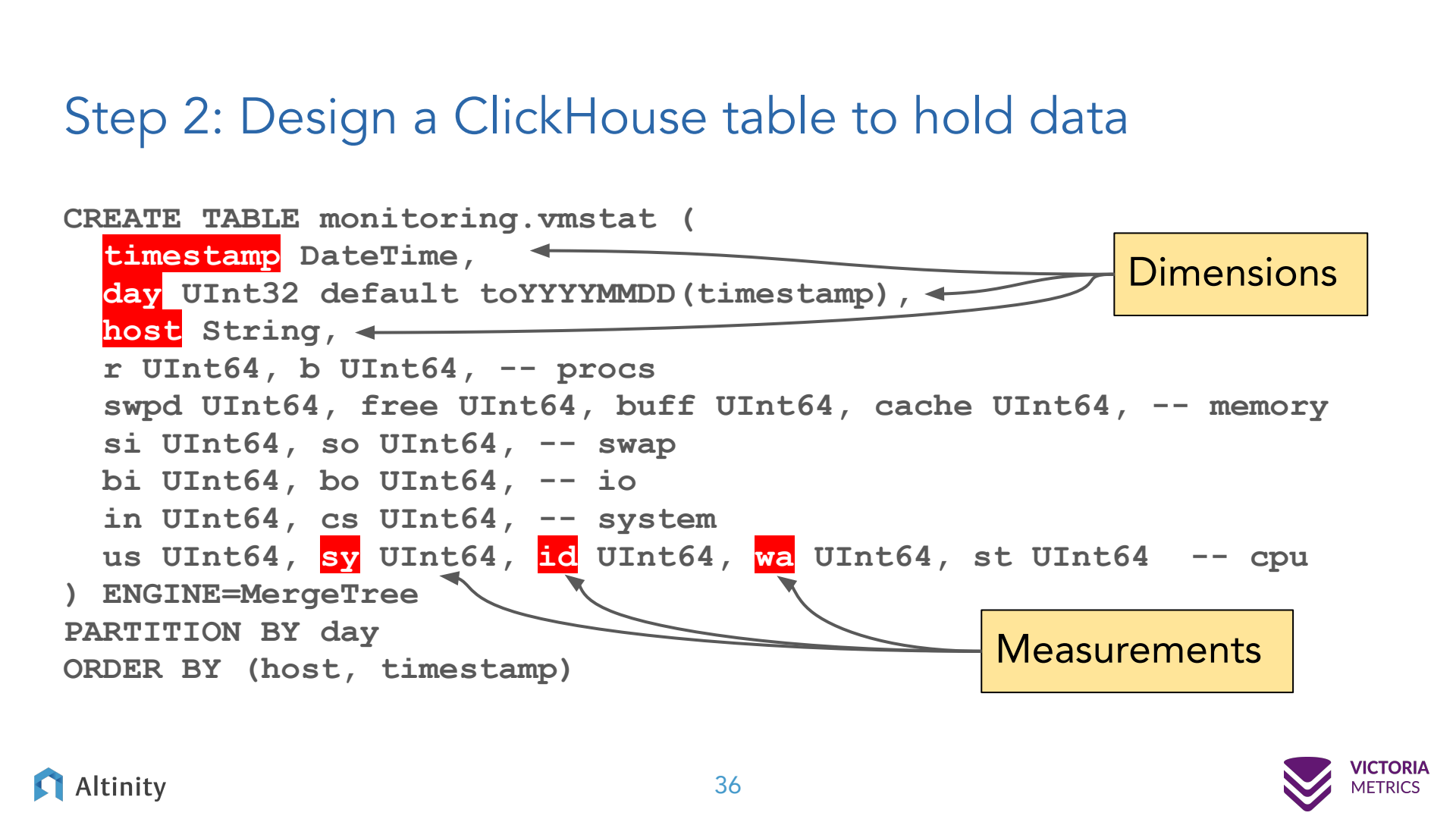

```
{"timestamp": "2023-01-22 18:13:16", "host": "logos3", "r": 0, "b":
0, "swpd": 166912, "free": 2523688, "buff": 41412, "cache": 3408292,
"si": 0, "so": 0, "bi": 3, "bo": 101, "in": 1, "cs": 0, "us": 2,
"sy": 1, "id": 98, "wa": 0, "st": 0}

{"timestamp": "2023-01-22 18:13:17", "host": "logos3", "r": 0, "b":
0, "swpd": 166912, "free": 2523696, "buff": 41412, "cache": 3408316,
"si": 0, "so": 0, "bi": 0, "bo": 216, "in": 1214, "cs": 4320, "us":
1, "sy": 1, "id": 98, "wa": 0, "st": 0}

{"timestamp": "2023-01-22 18:13:18", "host": "logos3", "r": 0, "b":
0, "swpd": 166912, "free": 2527120, "buff": 41412, "cache": 3408572,
"si": 0, "so": 0, "bi": 0, "bo": 0, "in": 1172, "cs": 4162, "us": 2,
"sy": 1, "id": 98, "wa": 0, "st": 0}
```

Step 2: Design a ClickHouse table to hold data

```
CREATE TABLE monitoring.vmstat (
   timestamp DateTime,
   day UInt32 default toYYYYMMDD(timestamp),
   host String,
   r UInt64, b UInt64, -- procs
   swpd UInt64, free UInt64, buff UInt64, cache UInt64, -- memory
   si UInt64, so UInt64, -- swap
   bi UInt64, bo UInt64, -- io
   in UInt64, cs UInt64, -- system
   us UInt64, sy UInt64, id UInt64, wa UInt64, st UInt64  -- cpu
) ENGINE=MergeTree
PARTITION BY day
ORDER BY (host, timestamp)
```



Step 3: Load data into ClickHouse

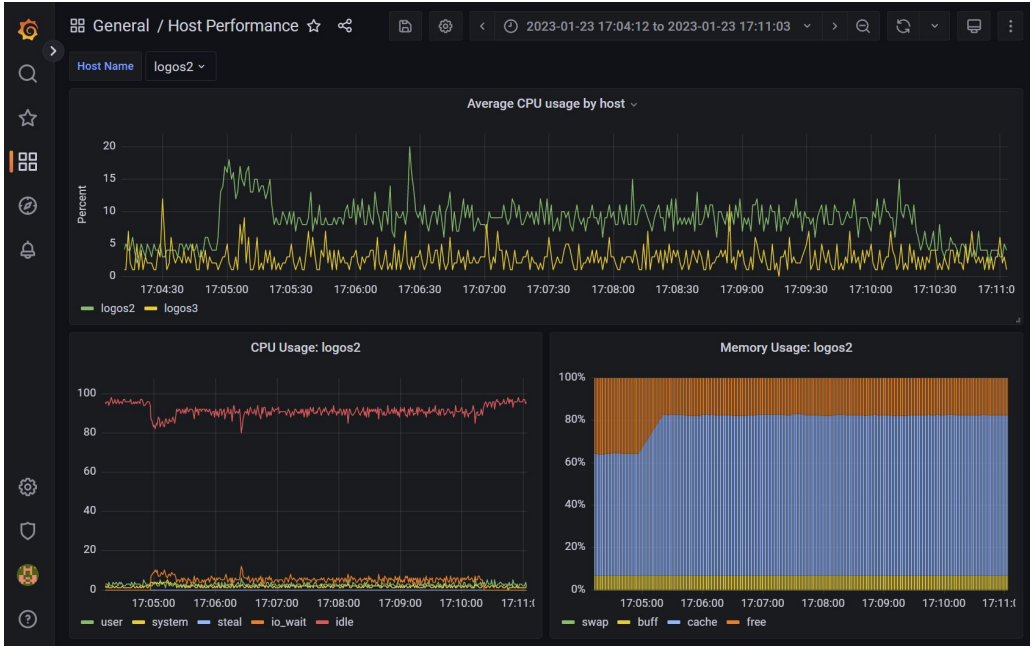
```
INSERT INTO vmstat Format JSONEachRow
```

E.g.

```
INSERT='INSERT%20INTO%20vmstat%20Format%20JSONEachRow'
cat vmstat.dat | curl -X POST --data-binary @- \
  "http://logos3:8123/?database=monitoring&query=${INSERT}"
```

(Or a Python script)

Step 4: Build a Grafana dashboard to show results



[ClickHouse data source for Grafana]

[Altinity plugin for ClickHouse]

After loading you can go crazy with analytical queries

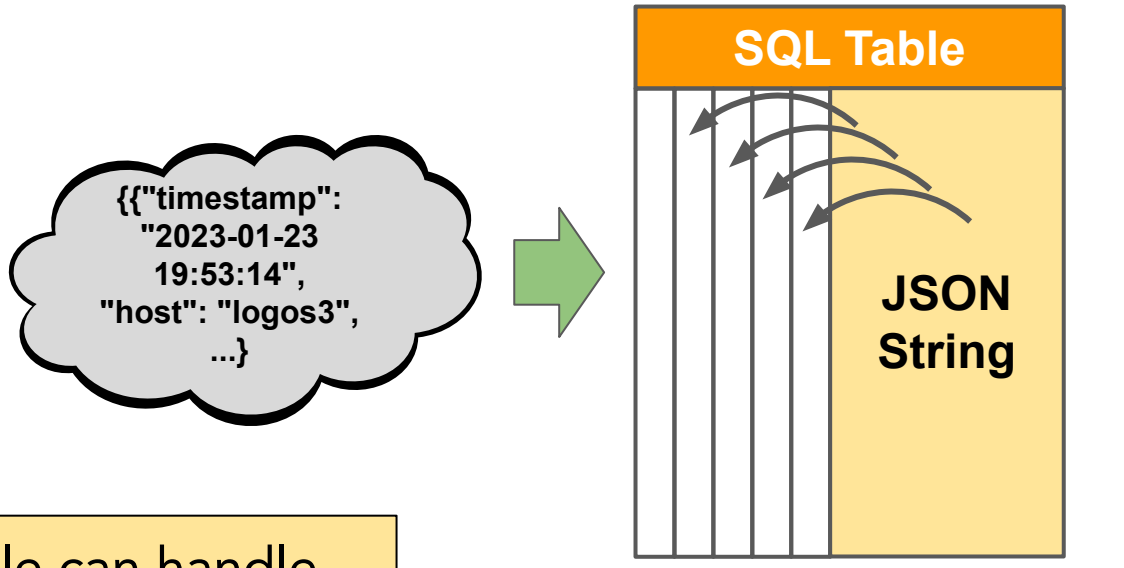
```
SELECT host, count() AS loaded_minutes
FROM (
    SELECT
        toStartOfMinute(timestamp) AS minute, host, avg(100 - id) AS load
    FROM monitoring.vmstat
    WHERE timestamp > (now() - toIntervalDay(1))
    GROUP BY minute, host HAVING load > 25
)
GROUP BY host ORDER BY loaded_minutes DESC
```

host	loaded_minutes
logos3	6
logos2	5

2 hosts had > 25% load for at least a minute in the last 24 hours

DEMO TIME!

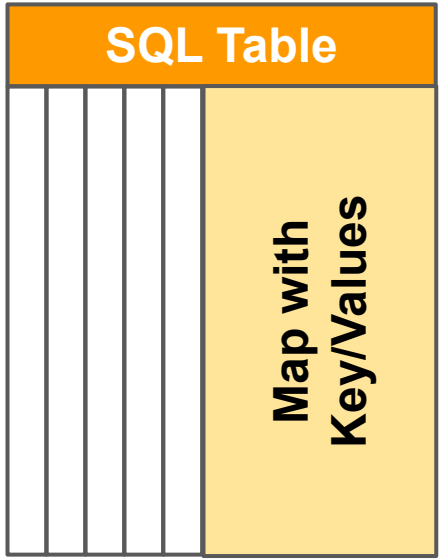
Can ClickHouse store data in a “schemaless” way?



One table can handle many entity types!

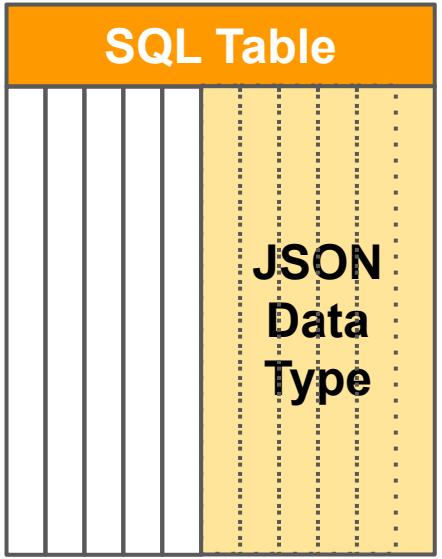
JSON String (“blob”) with derived header values

More schemaless ways to store data



Map: Header values & key value pairs

Arrays: Header values with key-value pairs



JSON data type mapped to column storage

Where is the software to build monitoring?

Event streaming

- Apache Kafka
- Apache Pulsar
- Vectorized Redpanda

ELT

- Apache Airflow
- Rudderstack

Rendering/Display

- Apache Superset
- Cube.js
- Grafana

Client Libraries

- C++ - ClickHouse CPP
- Golang - ClickHouse Go
- Java - ClickHouse JDBC
- Javascript/Node.js - Apla
- ODBC - ODBC Driver for ClickHouse
- Python - ClickHouse Driver, ClickHouse SQLAlchemy

More client library links HERE

Kubernetes

- Altinity Operator for ClickHouse

Where can I find out more about ClickHouse?

ClickHouse official docs – https://clickhouse.com/docs/

Altinity Blog – https://altinity.com/blog/

Altinity Youtube Channel – https://www.youtube.com/channel/UCE3Y2lDKl_ZfjaCrh62onYA

Altinity Knowledge Base – https://kb.altinity.com/

Meetups, other blogs, and external resources. Use your powers of Search!

Wrap-up

Comparing VictoriaMetrics and ClickHouse databases

VictoriaMetrics

Talks MetricsQL, PromQL, Graphite QL

Stores time series data

No explicit schema

Easy to load data using simple clients

Can pull data from Prometheus exporters and Kafka

Time-series specific functions and transformations

Integrates with any BI tool that speaks PromQL

Extremely fast and scalable

ClickHouse

Talks SQL

Stores any kind of data

Uses tables; many ways to represent data

Easy to load data using simple clients

Can pull data from Kafka and object storage

Versatile queries including JOIN and aggregation

Most BI tools have ClickHouse adapters

Extremely fast and scalable

Help for building monitoring systems that work

VictoriaMetrics Inc.

VictoriaMetrics Community

VictoriaMetrics Enterprise

VictoriaMetrics Managed platform

Altinity Inc.

Altinity.Cloud managed ClickHouse platform

Enterprise support for ClickHouse

Altinity Developer Academy classes

Altinity Stable Builds for ClickHouse

Altinity Kubernetes Operator for ClickHouse

Thank you!
Questions?

https://altinity.com
Contact Altinity

https://victoriametrics.com
Contact VictoriaMetrics

Altinity

VICTORIA METRICS