DevOpsCon

Fast, Cheap, DIY Observability with Open Source Analytics and Visualization

Josh Lee, Altintity

devmio NEW YORK WEEK

Josh Lee

Open Source Advocate

Altinity

ClickHouse® is a registered trademark of ClickHouse, Inc.
Altinity is not affiliated with or associated with ClickHouse, Inc.
We are but humble open source contributors

Observability is our ability to understand a system from its outputs alone

FUEL TRANSFER IN FLIGHT PROHIBITED
PROPELLER FEATHERING BUTTONS
HEATER CONTROL PANEL
GENERATOR LOAD PANEL
PRESSURIZATION CONTROL PANEL

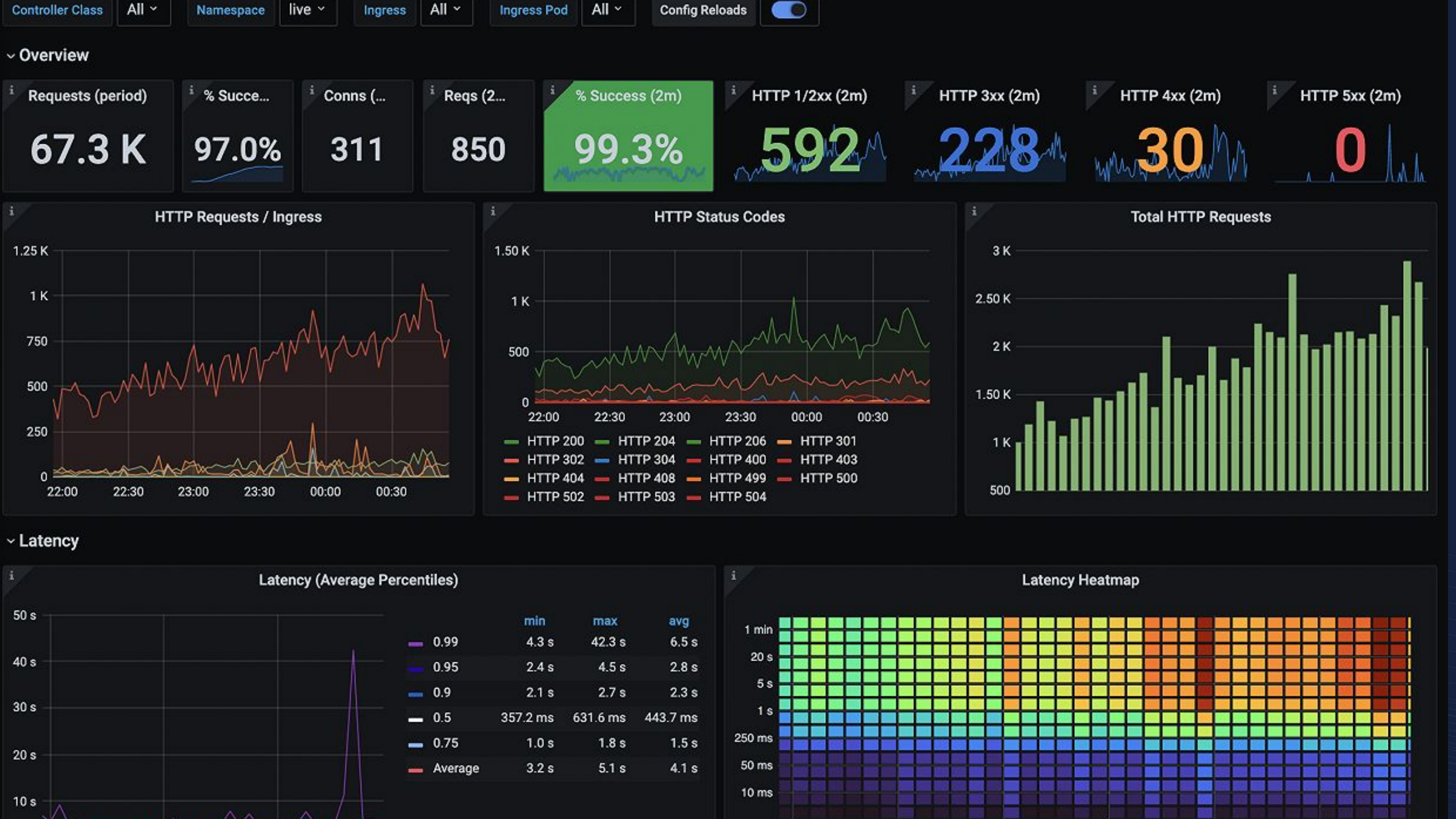
HYDRAULIC
VENTILATION
PRESSURIZATION
CENTER PUMPS
ANTI-ICE
PASS SIGNS
LANDING
FUEL CONTROL
RUN
CUTOFF

A Typical Request Log

```
2024-07-01 09:35:34 GET /home 200 ...
```

Adding Duration

```
2024-07-01 09:35:34 231ms GET /home 200
```



Controller Class: All
Namespace: live
Ingress: All
Ingress Pod: All
Config Reloads: ●

Overview

Requests (period)	% Succe...	Conns (...	Reqs (2...	% Success (2m)	HTTP 1/2xx (2m)	HTTP 3xx (2m)	HTTP 4xx (2m)	HTTP 5xx (2m)
67.3 K	97.0%	311	850	99.3%	592	228	30	0

HTTP Requests / Ingress

HTTP Status Codes

Total HTTP Requests

Latency

Latency (Average Percentiles)

	min	max	avg
0.99	4.3 s	42.3 s	6.5 s
0.95	2.4 s	4.5 s	2.8 s
0.9	2.1 s	2.7 s	2.3 s
0.5	357.2 ms	631.6 ms	443.7 ms
0.75	1.0 s	1.8 s	1.5 s
Average	3.2 s	5.1 s	4.1 s

Latency Heatmap

Back to our log...

```
2024-07-01 09:35:34 231ms GET /home 200
```

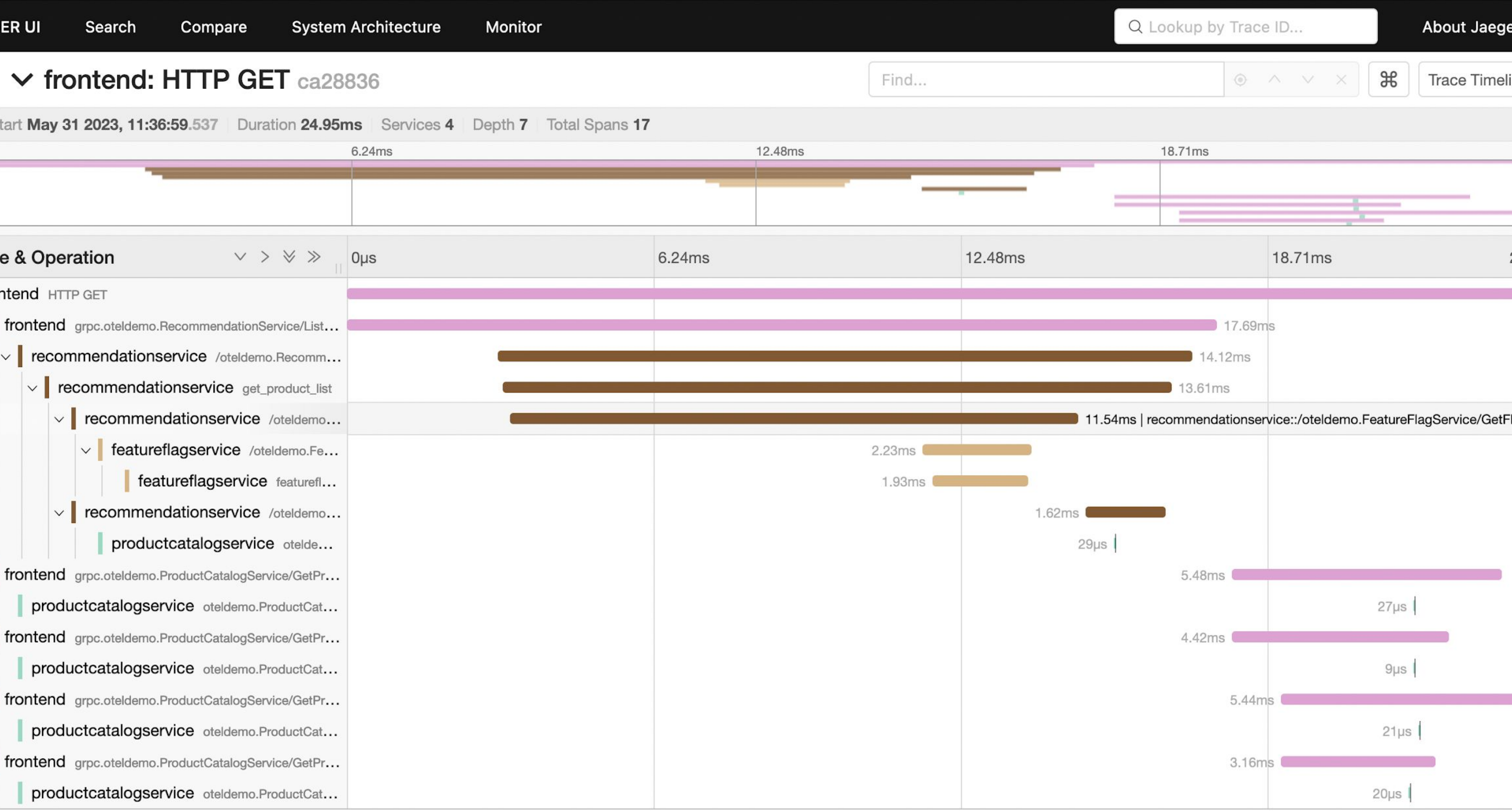
Back to our log…

```
Request:123 2024-07-01 09:35:34 231ms GET /home
200
```

Connecting the trace:

```
Trace:4ea3 Span:123 2024-07-01 09:35:34 231ms
GET /home 200

Trace:4ea3 Span:456 ParentSpan:123 2024-07-01
09:35:34 201ms GET /api/users 201
```



frontend: HTTP GET ca28836

tart May 31 2023, 11:36:59.537 Duration 24.95ms Services 4 Depth 7 Total Spans 17

e & Operation	Duration
ntend HTTP GET	
frontend grpc.oteldemo.RecommendationService/List…	17.69ms
recommendationservice /oteldemo.Recomm…	14.12ms
recommendationservice get_product_list	13.61ms
recommendationservice /oteldemo…	11.54ms | recommendationservice::/oteldemo.FeatureFlagService/GetFl
featureflagservice /oteldemo.Fe…	2.23ms
featureflagservice featurefl…	1.93ms
recommendationservice /oteldemo…	1.62ms
productcatalogservice otelde…	29μs
frontend grpc.oteldemo.ProductCatalogService/GetPr…	5.48ms
productcatalogservice oteldemo.ProductCat…	27μs
frontend grpc.oteldemo.ProductCatalogService/GetPr…	4.42ms
productcatalogservice oteldemo.ProductCat…	9μs
frontend grpc.oteldemo.ProductCatalogService/GetPr…	5.44ms
productcatalogservice oteldemo.ProductCat…	21μs
frontend grpc.oteldemo.ProductCatalogService/GetPr…	3.16ms
productcatalogservice oteldemo.ProductCat…	20μs

Observability is not any one signal…

Metrics

Aggregable

Is there a problem?

Traces

Request-Scoped

Where is the problem?

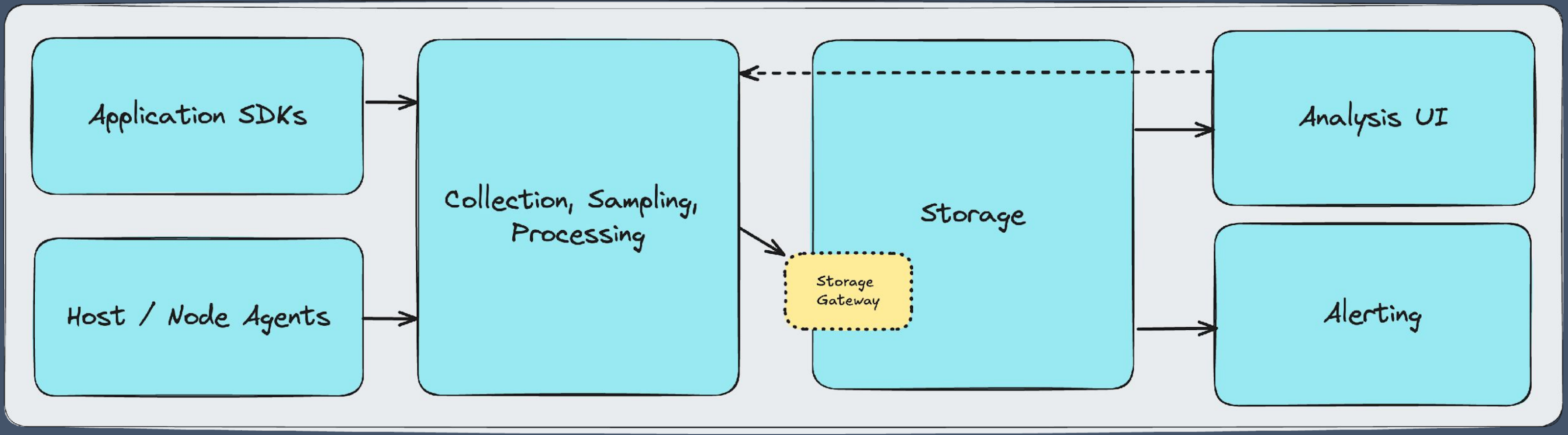
Logs

Verbose, time-stamped records

What is the problem?

```
$ vmstat -n 2 10
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 343296 21690808 2290104 6897160    0    0     3   187    0    2  4  2 94  0  0
 0  0 343296 21690800 2290104 6897160    0    0     0    60 2989 7688  2  1 97  0  0
 0  0 343296 21690140 2290104 6897164    0    0     0    72 4704 13677  3  2 95  0  0
 0  0 343296 21689888 2290104 6897164    0    0     0    14 3132 9364  2  1 97  0  0
 0  0 343296 21690220 2290104 6897168    0    0     0    86 3014 7995  1  1 97  0  0
 0  0 343296 21690448 2290104 6897176    0    0     0    20 2660 7297  1  1 98  0  0
 0  0 343296 21690268 2290104 6897176    0    0     0    12 2695 7222  1  1 98  0  0
 1  0 343296 21690196 2290104 6897180    0    0     0    80 3641 10419  2  1 97  0  0
 0  0 343296 21689696 2290104 6897180    0    0     0    14 4108 12605  3  2 95  0  0
 0  0 343296 21689900 2290104 6897184    0    0     0    60 2688 7270  2  1 97  0  0
```

A complete observability solution



Introducing ClickHouse

- SQL-compatible
- Massively scalable
- Really, really fast

Telemetry is WORM

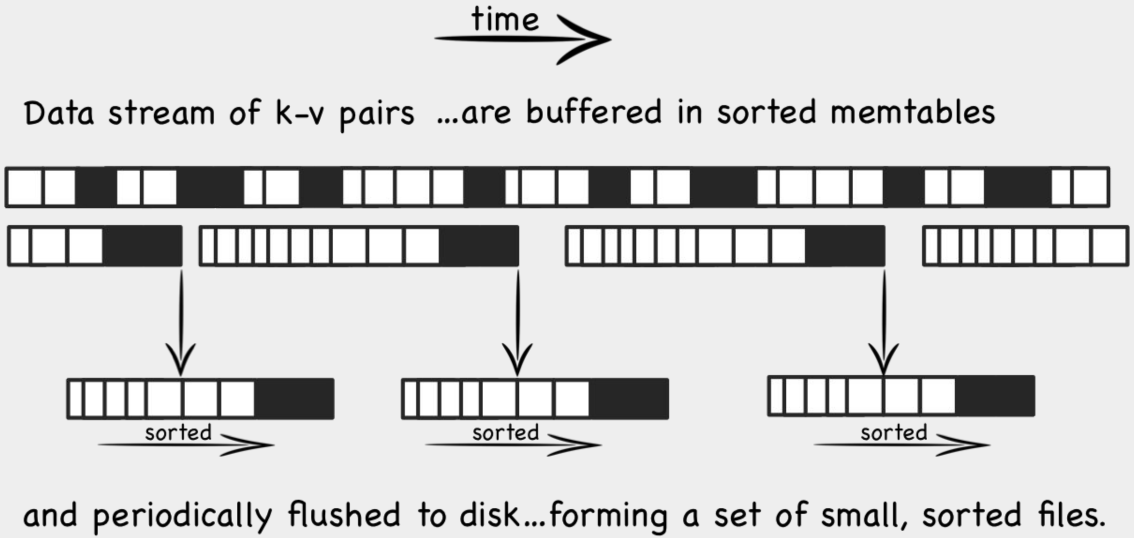
Write-Once, Read-Many

Telemetry is WORM

Write-Once, Read-Maybe

B-Trees: Optimized for Reads

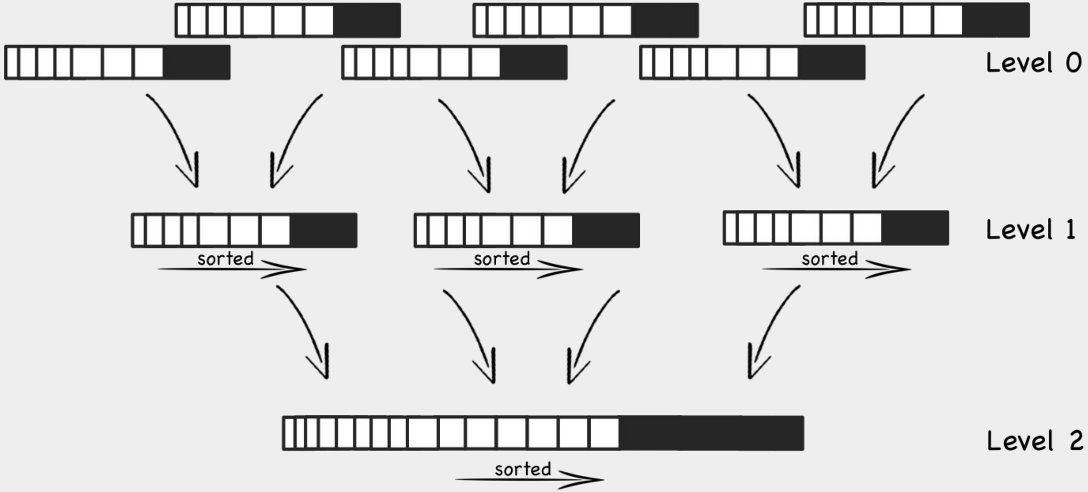
Log-Structured Merge Trees: Optimized for ingestion



time
Data stream of k-v pairs ...are buffered in sorted memtables
sorted
sorted
sorted
and periodically flushed to disk...forming a set of small, sorted files.

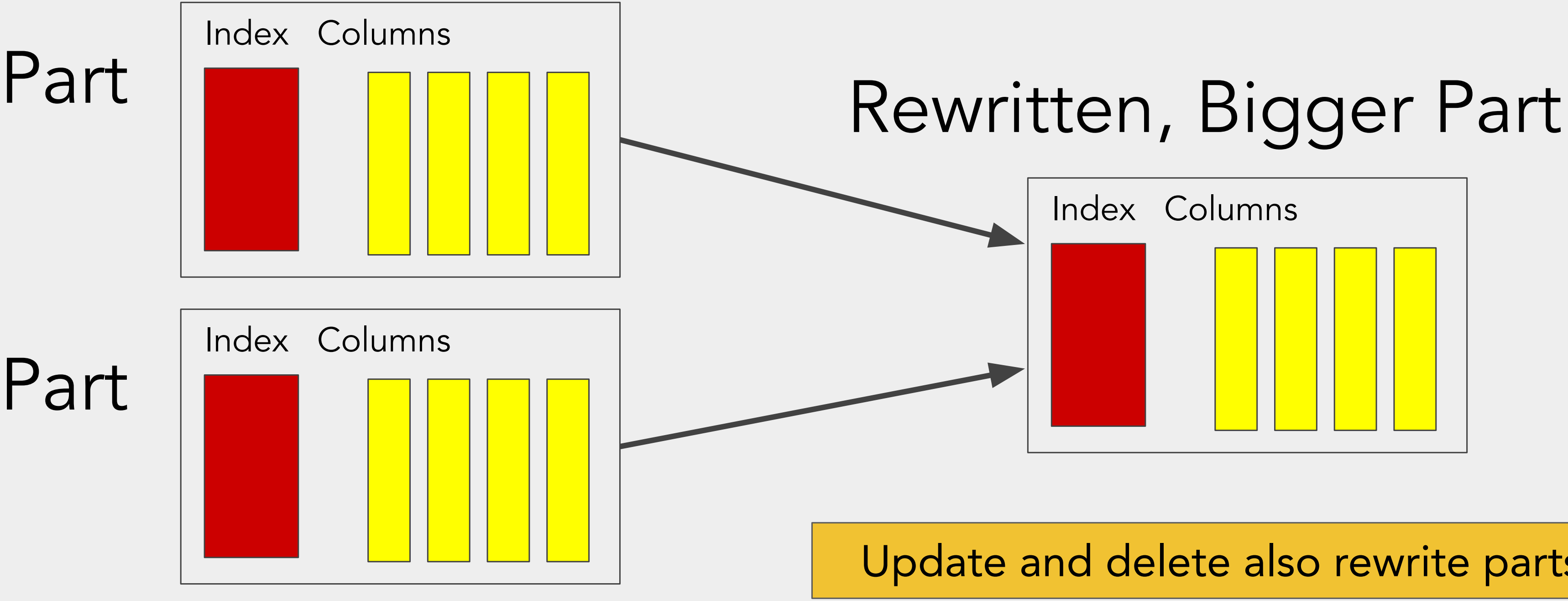
http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

Log-Structured Merge Trees: Background compaction

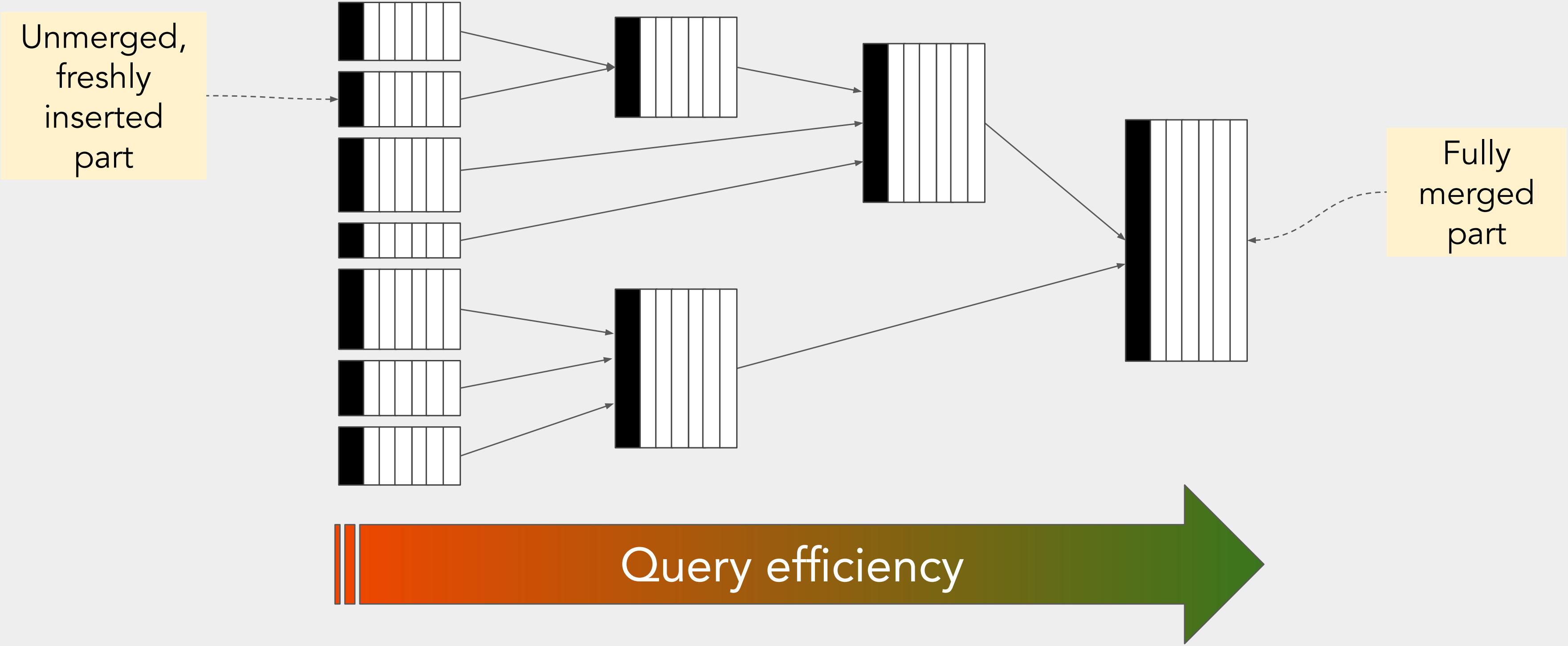


Compaction continues creating fewer, larger and larger files

http://www.benstopford.com/2015/02/14/log-structured-merge-trees/



Update and delete also rewrite parts


Unmerged, freshly inserted part
Fully merged part
Query efficiency

ClickHouse for Observability

How does this help?

- Fast writes
- Time-friendly
- Easy cleanup
- Cost-effective

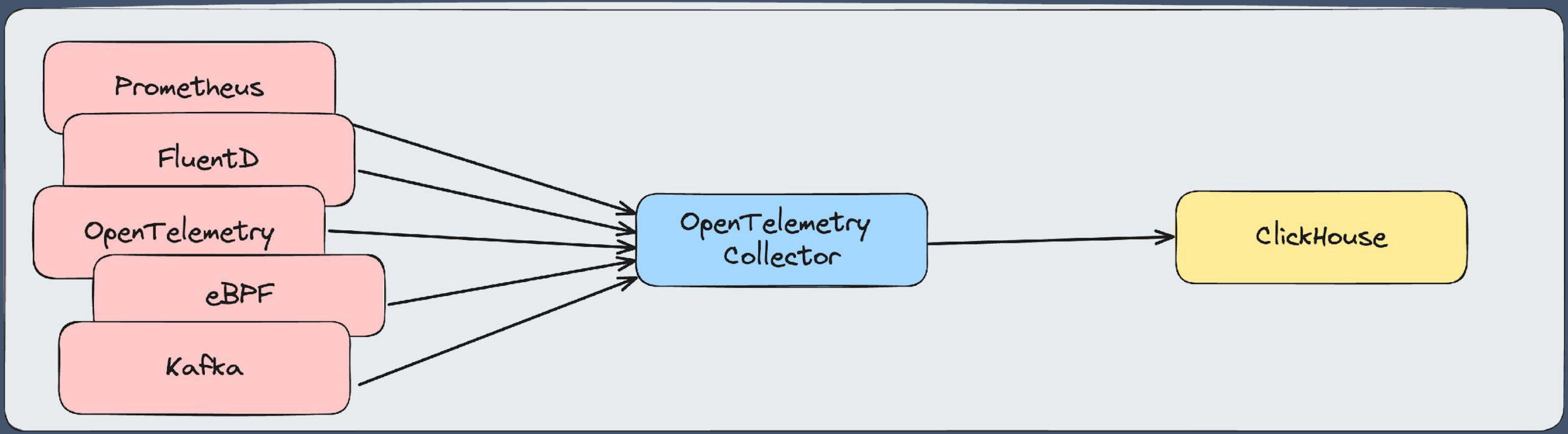
ClickHouse for Observability

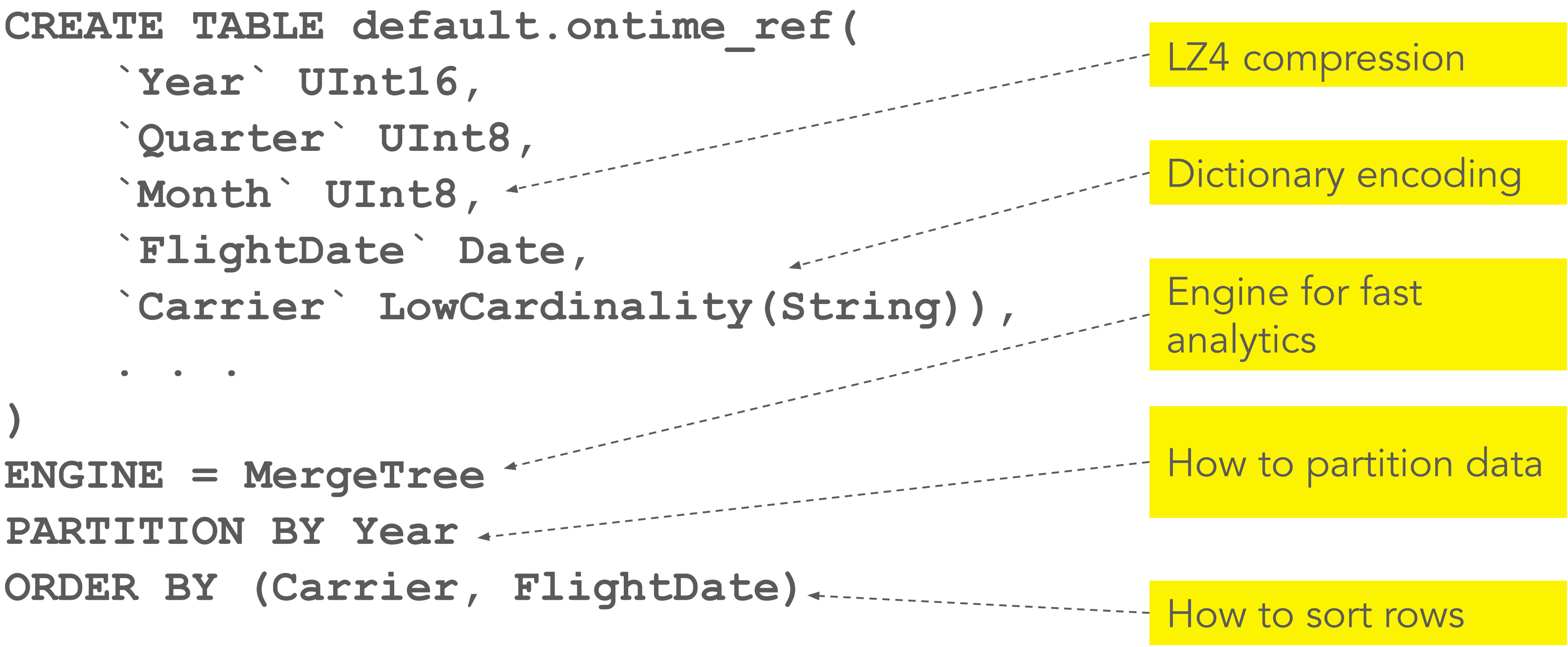
Integrations

- Grafana Datasource Plugin
- Jaeger w/ ClickHouse backend
- Kafka table engine

ClickHouse for Observability

Integrations via OpenTelemetry





```
CREATE TABLE default.ontime_ref(
    `Year` UInt16,
    `Quarter` UInt8,
    `Month` UInt8,
    `FlightDate` Date,
    `Carrier` LowCardinality(String)),
    . . .
)
ENGINE = MergeTree
PARTITION BY Year
ORDER BY (Carrier, FlightDate)
```

LZ4 compression

Dictionary encoding

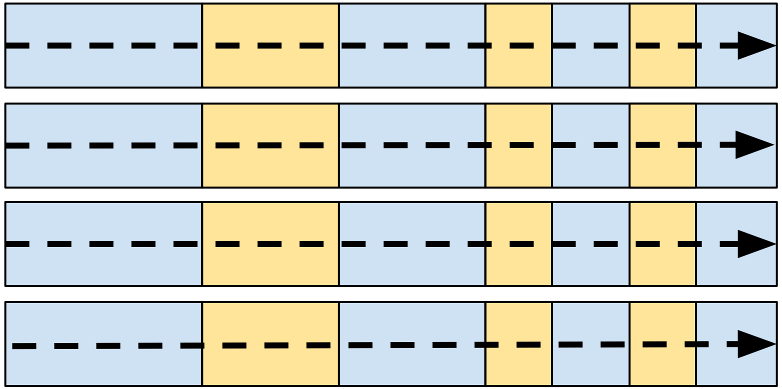
Engine for fast analytics

How to partition data

How to sort rows

PostgreSQL, MySQL

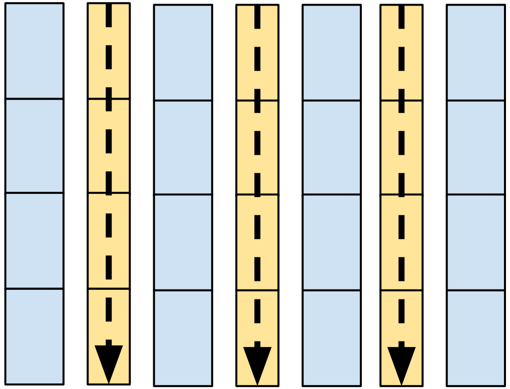
Read all columns in row



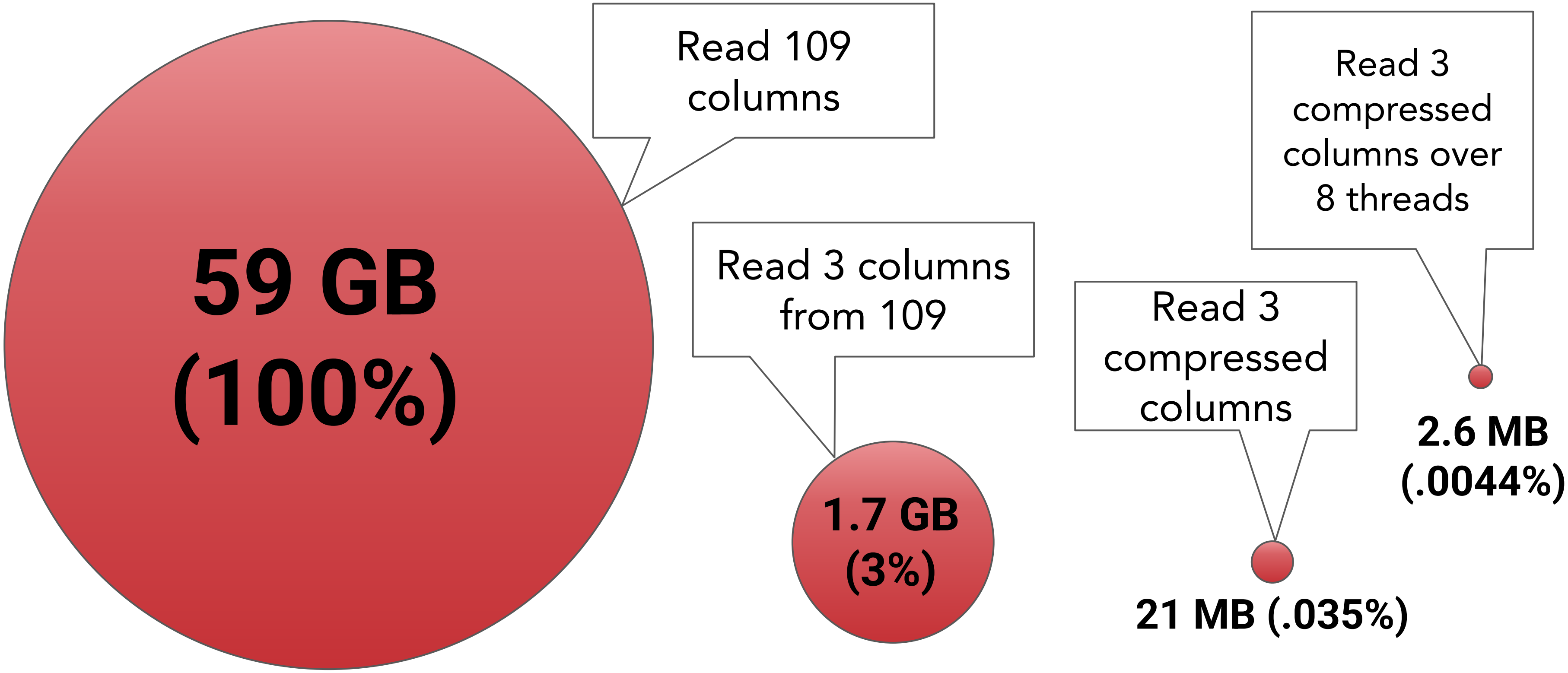
Rows compressed minimally or not at all

ClickHouse

Read only selected columns



Columns highly compressed



Read 109 columns

59 GB (100%)

Read 3 columns from 109

1.7 GB (3%)

Read 3 compressed columns

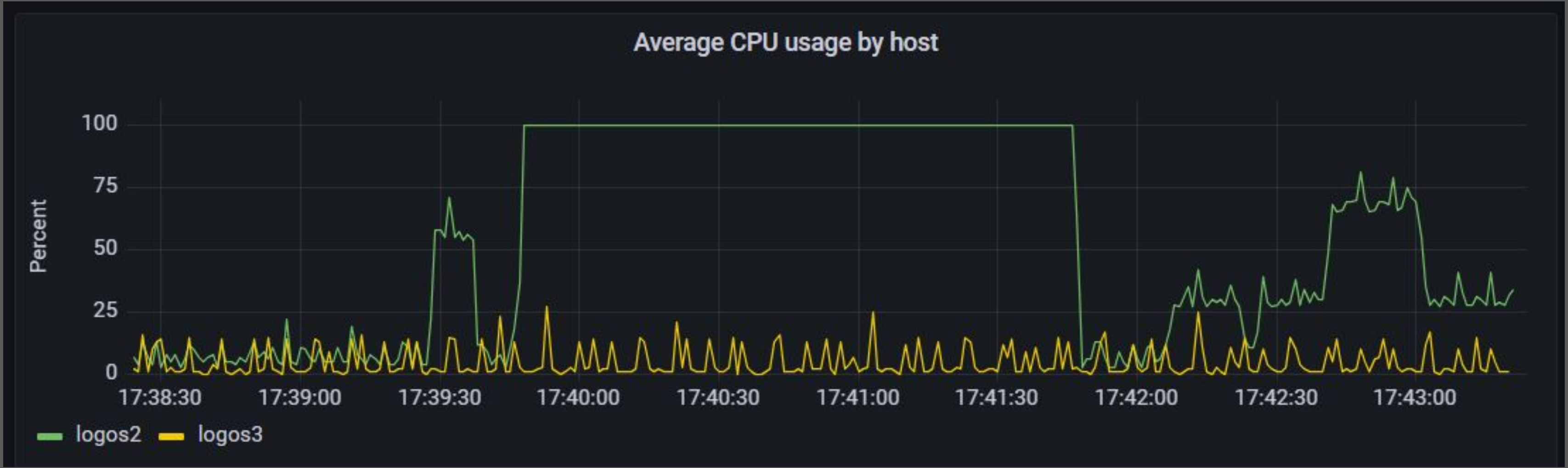
21 MB (.035%)

Read 3 compressed columns over 8 threads

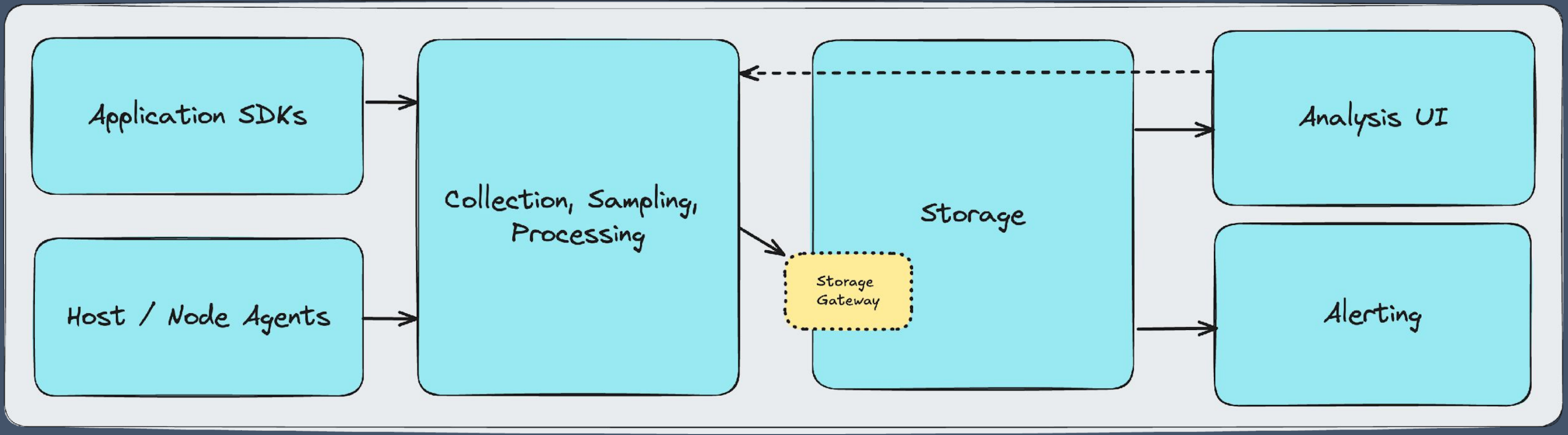
2.6 MB (.0044%)

Ok, back to our simple o11y demo…

```
$ vmstat -n 2 10
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 343296 21690808 2290104 6897160    0    0     3   187    0    2  4  2 94  0  0
 0  0 343296 21690800 2290104 6897160    0    0     0    60 2989 7688  2  1 97  0  0
 0  0 343296 21690140 2290104 6897164    0    0     0    72 4704 13677  3  2 95  0  0
 0  0 343296 21689888 2290104 6897164    0    0     0    14 3132 9364  2  1 97  0  0
 0  0 343296 21690220 2290104 6897168    0    0     0    86 3014 7995  1  1 97  0  0
 0  0 343296 21690448 2290104 6897176    0    0     0    20 2660 7297  1  1 98  0  0
 0  0 343296 21690268 2290104 6897176    0    0     0    12 2695 7222  1  1 98  0  0
 1  0 343296 21690196 2290104 6897180    0    0     0    80 3641 10419  2  1 97  0  0
 0  0 343296 21689696 2290104 6897180    0    0     0    14 4108 12605  3  2 95  0  0
 0  0 343296 21689900 2290104 6897184    0    0     0    60 2688 7270  2  1 97  0  0
```

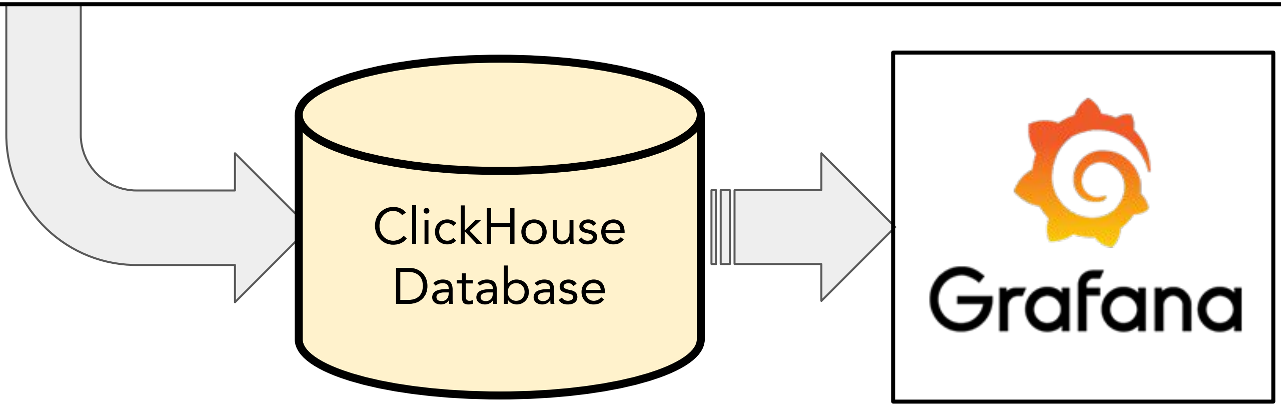

Average CPU usage by host
Percent
100
75
50
25
0
17:38:30
17:39:00
17:39:30
17:40:00
17:40:30
17:41:00
17:41:30
17:42:00
17:42:30
17:43:00
logos2
logos3

A complete observability solution



Sooo…How do we ingest vmstat data and display it?

```
$ 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()
```

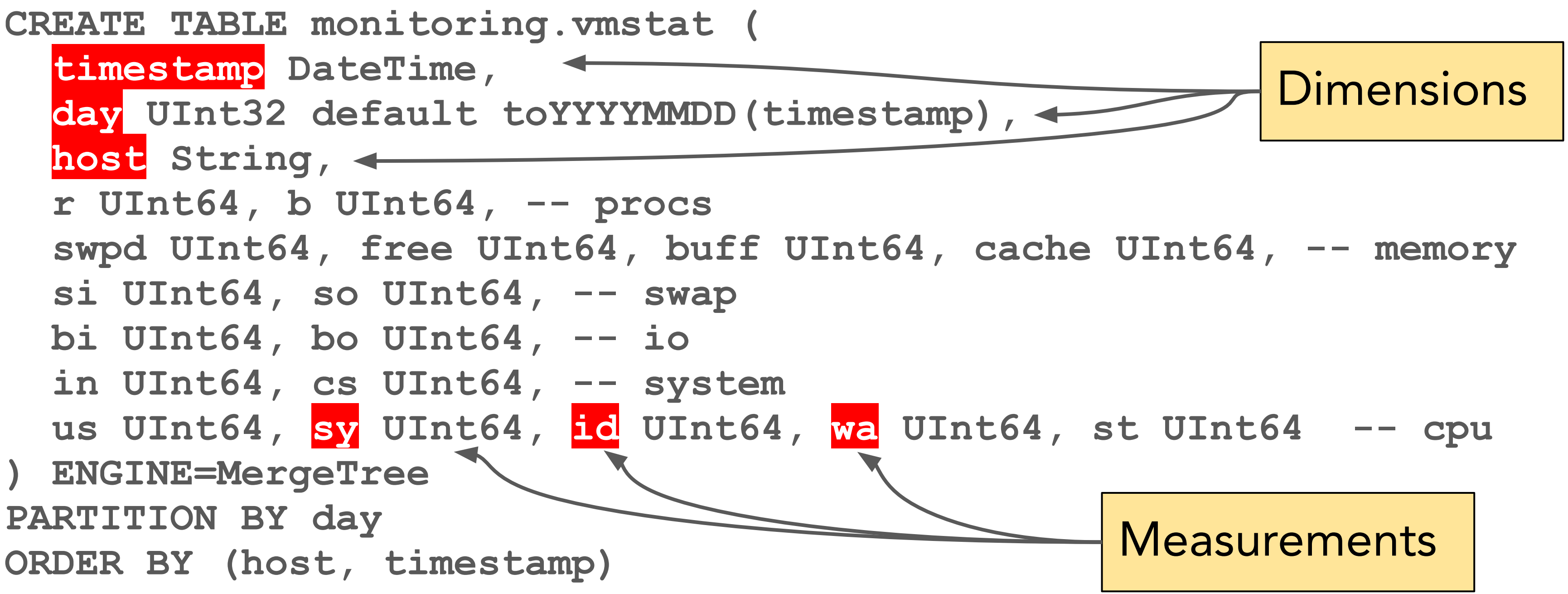
Here’s the output

```
{"timestamp": "2024-01-22 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": "2024-01-22 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": "2024-01-22 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)
```

Dimensions

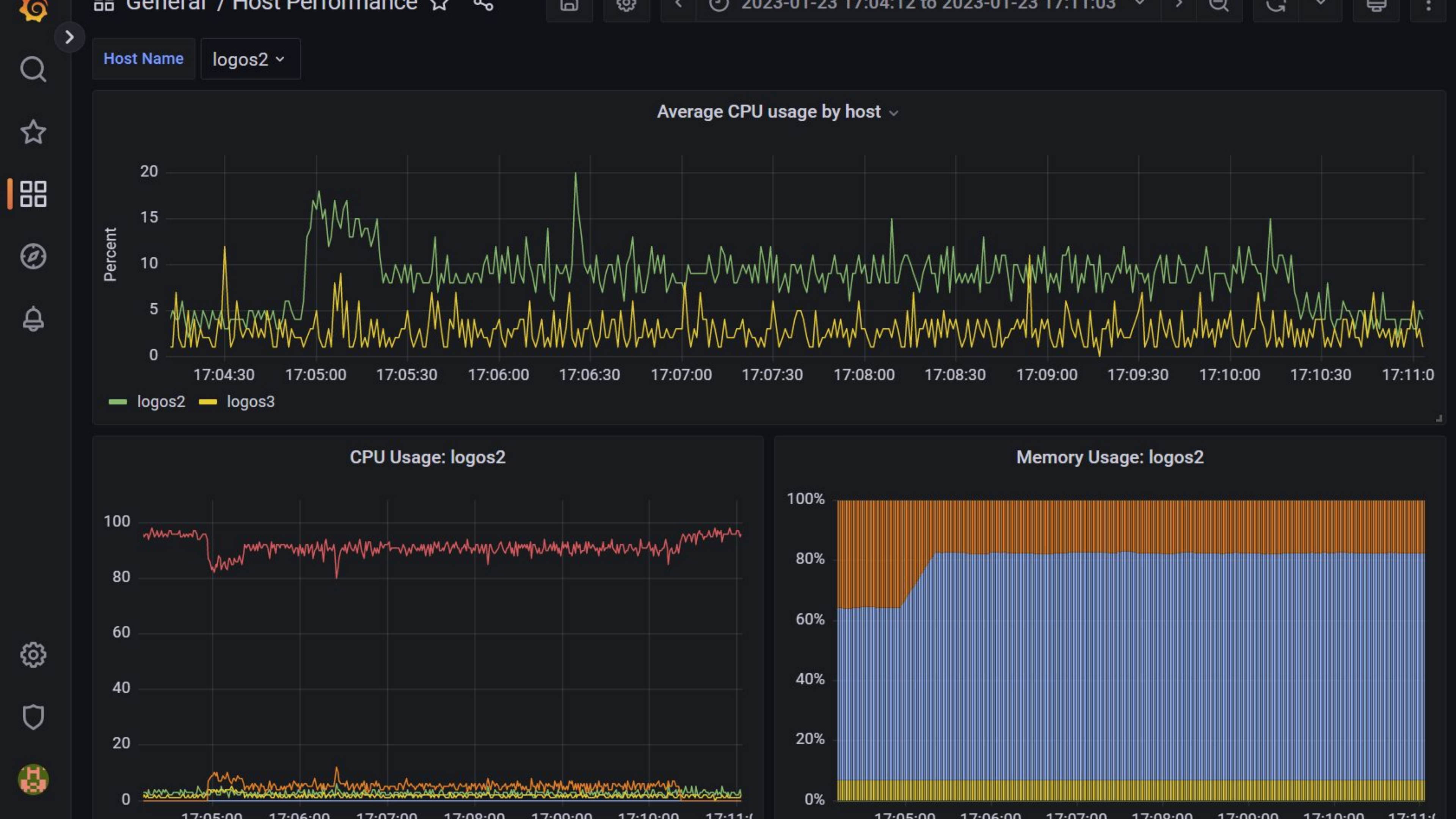
Measurements

Step 3: Load data into ClickHouse

```
INSERT INTO vmstat Format JSONEachRow
```

```
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)



General / Host Performance

2023-01-23 17:04:12 to 2023-01-23 17:11:03

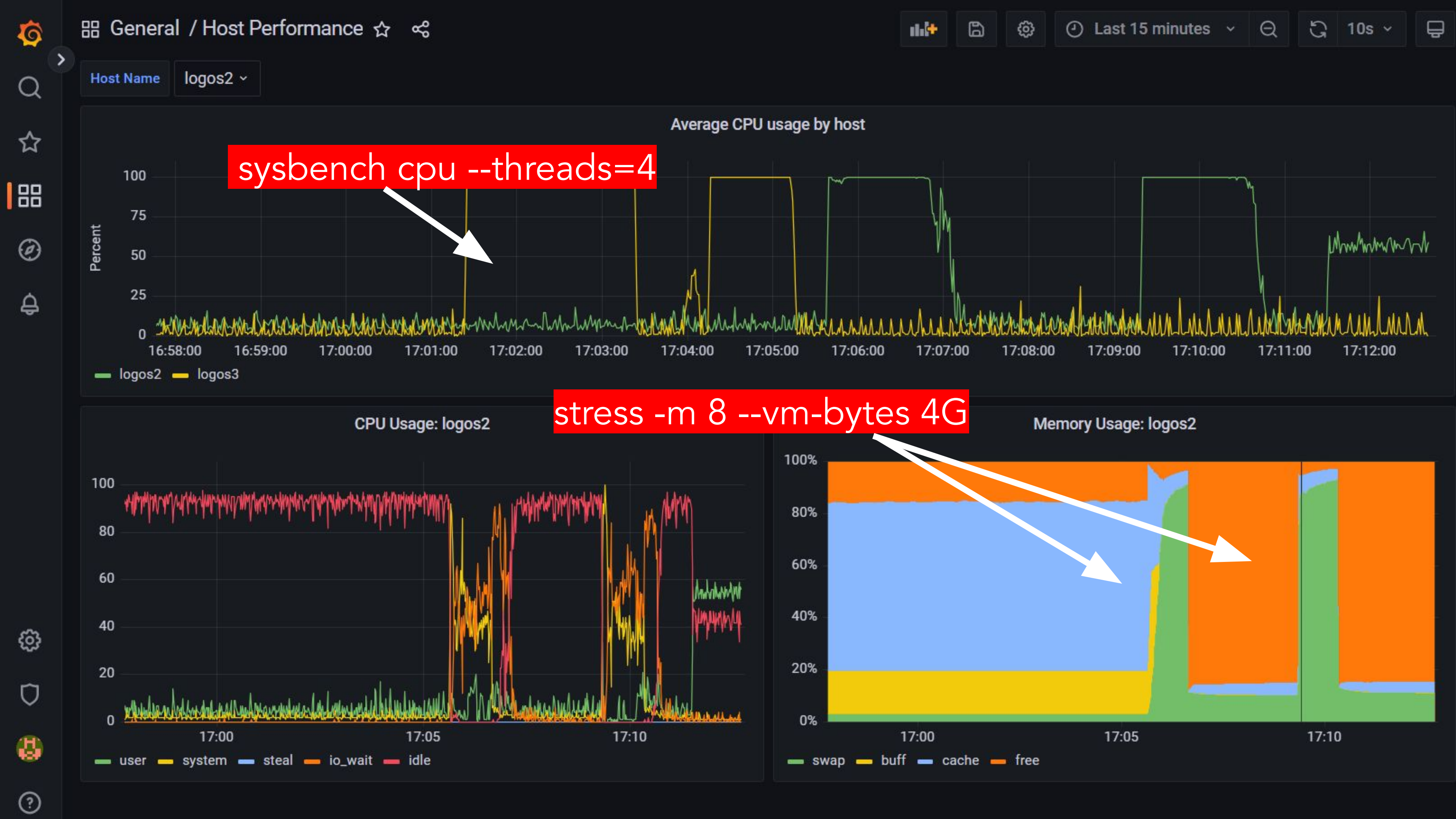
Host Name: logos2

Step 5: Go crazy!

```
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

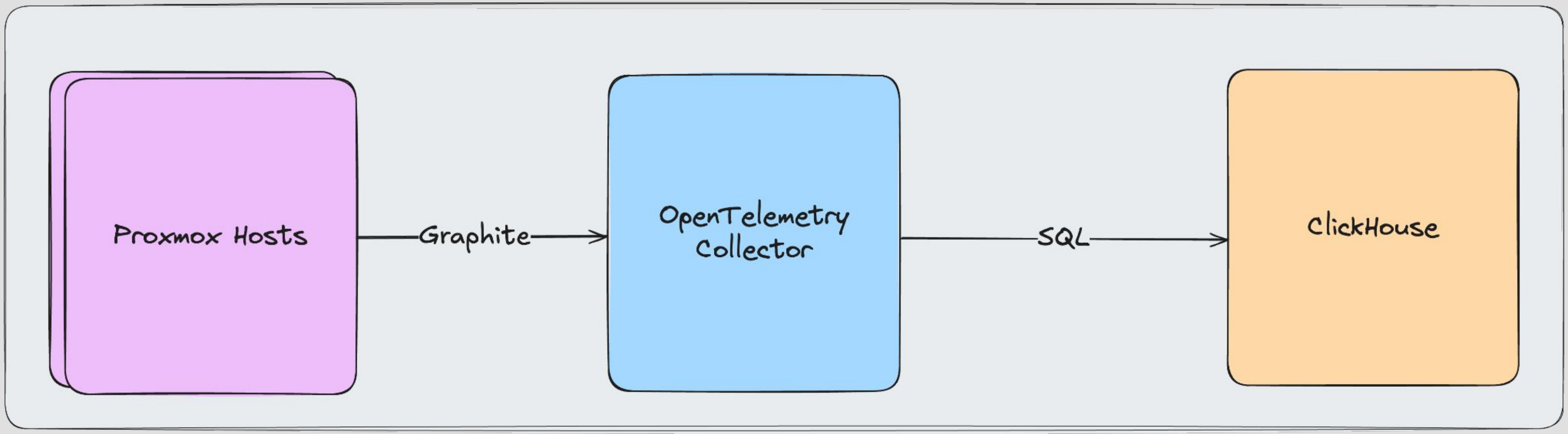


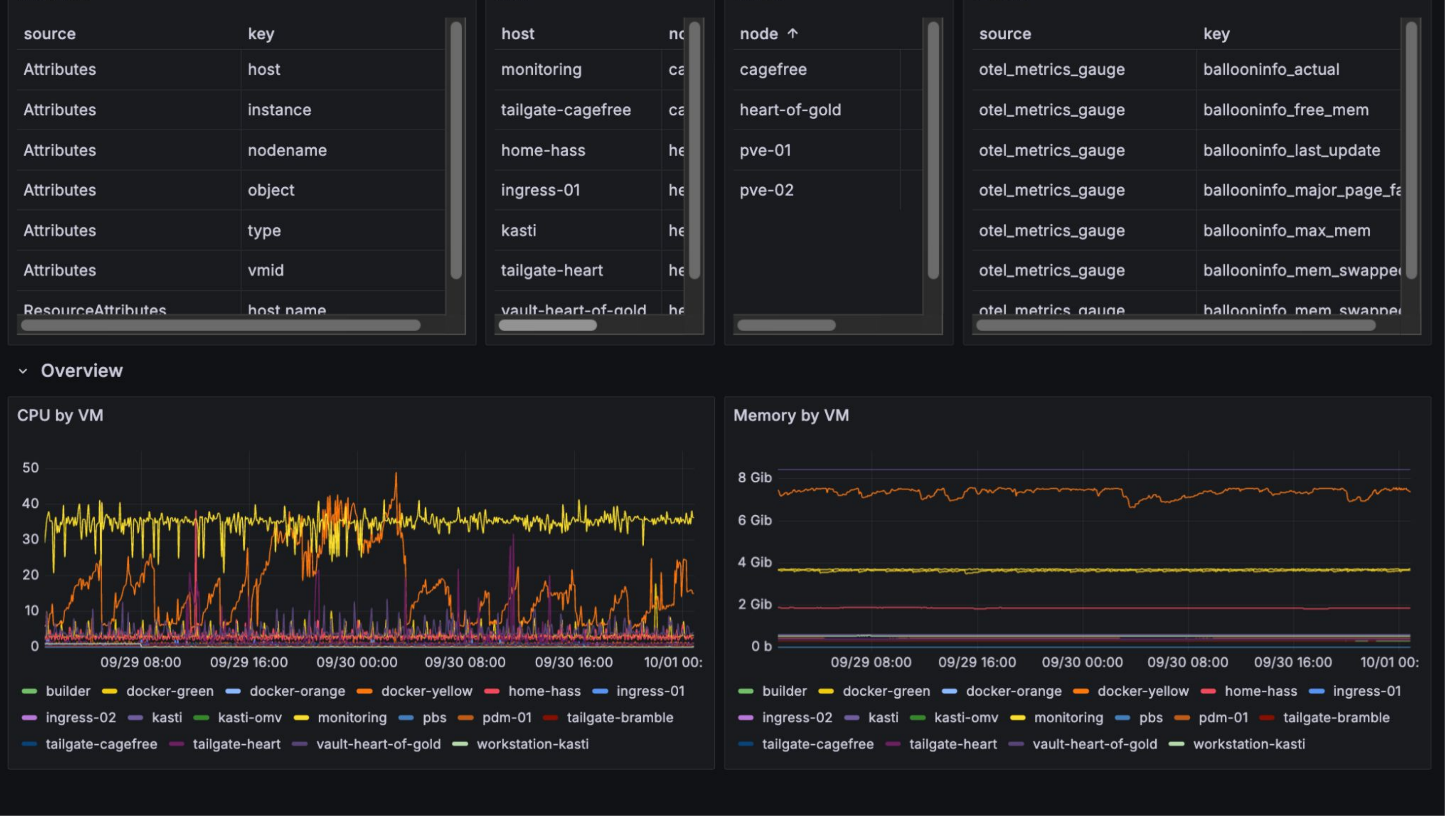
```
sysbench cpu --threads=4
```

```
stress -m 8 --vm-bytes 4G
```

Example #2: Monitoring Proxmox Hosts

Monitoring Proxmox Hosts





source	key
Attributes	host
Attributes	instance
Attributes	nodename
Attributes	object
Attributes	type
Attributes	vmid
ResourceAttributes	host.name

host	nc
monitoring	ca
tailgate-cagefree	ca
home-hass	he
ingress-01	he
kasti	he
tailgate-heart	he
vault-heart-of-gold	he

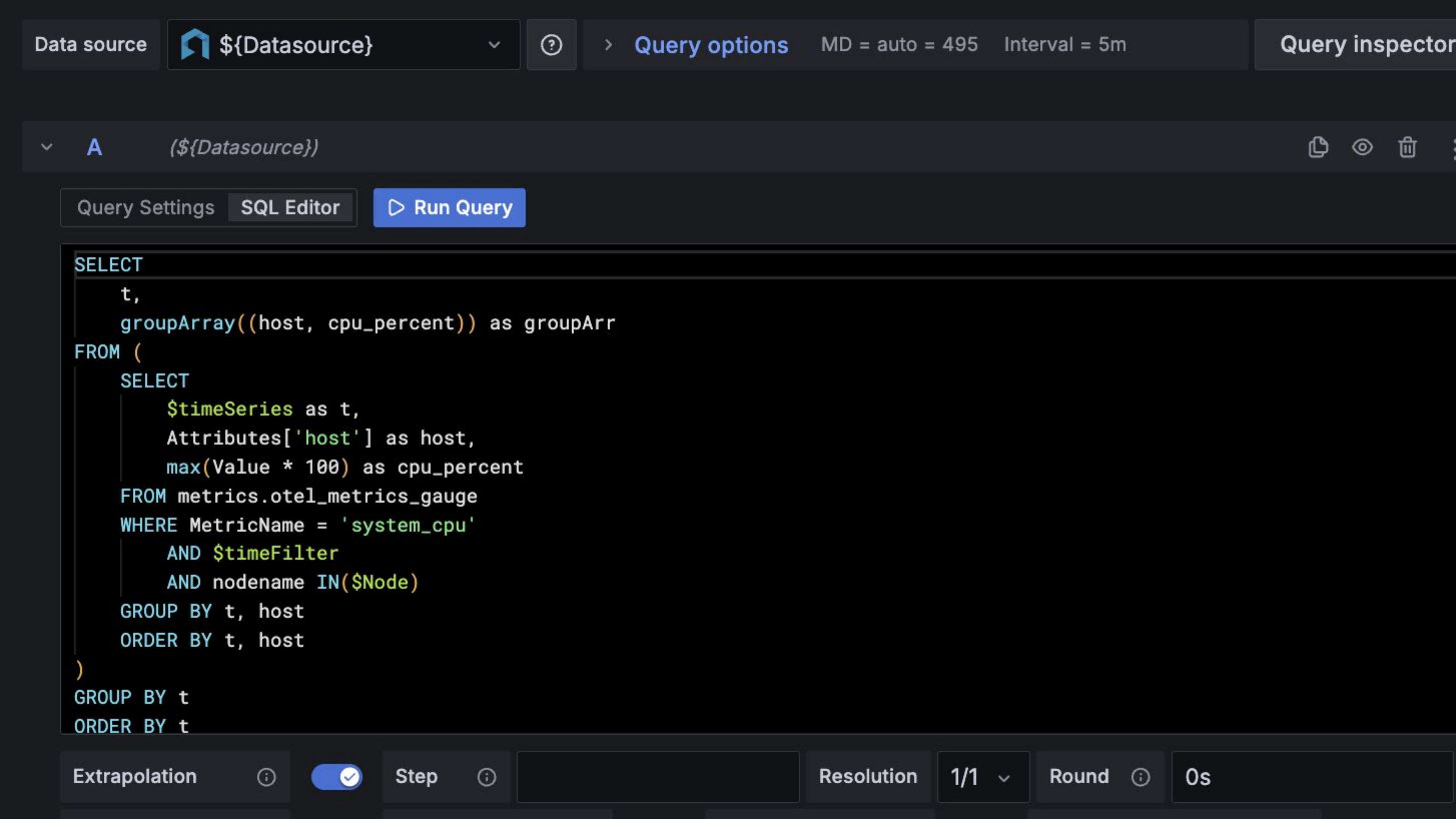
node ↑
cagefree
heart-of-gold
pve-01
pve-02

source	key
otel_metrics_gauge	ballooninfo_actual
otel_metrics_gauge	ballooninfo_free_mem
otel_metrics_gauge	ballooninfo_last_update
otel_metrics_gauge	ballooninfo_major_page_fa
otel_metrics_gauge	ballooninfo_max_mem
otel_metrics_gauge	ballooninfo_mem_swappe
otel_metrics_gauge	ballooninfo_mem_swappe

Overview

CPU by VM

Memory by VM



Data source ${Datasource}

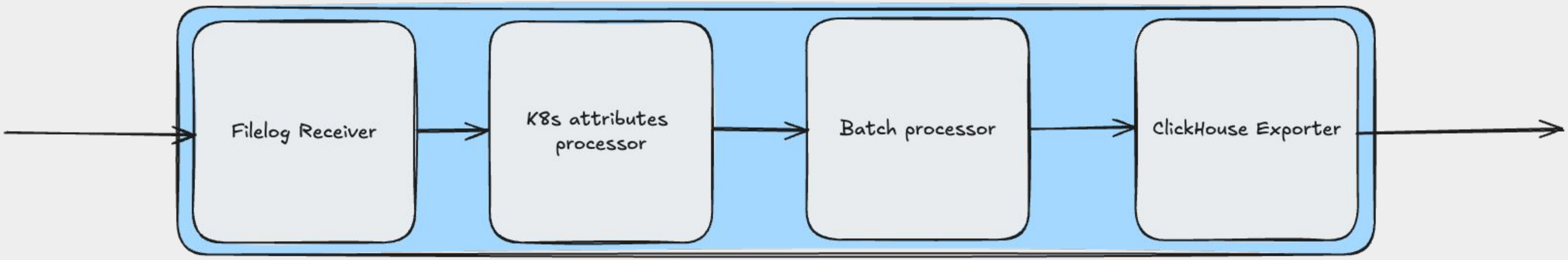
MD = auto = 495 Interval = 5m

(${Datasource})

```
SELECT
    t,
    groupArray((host, cpu_percent)) as groupArr
FROM (
    SELECT
        $timeSeries as t,
        Attributes['host'] as host,
        max(Value * 100) as cpu_percent
    FROM metrics.otel_metrics_gauge
    WHERE MetricName = 'system_cpu'
        AND $timeFilter
        AND nodename IN($Node)
    GROUP BY t, host
    ORDER BY t, host
)
GROUP BY t
ORDER BY t
```

Extrapolation ☑ Step Resolution 1/1 Round 0s

Example #3: Ingesting K8s Logs w/ OpenTelemetry + ClickHouse



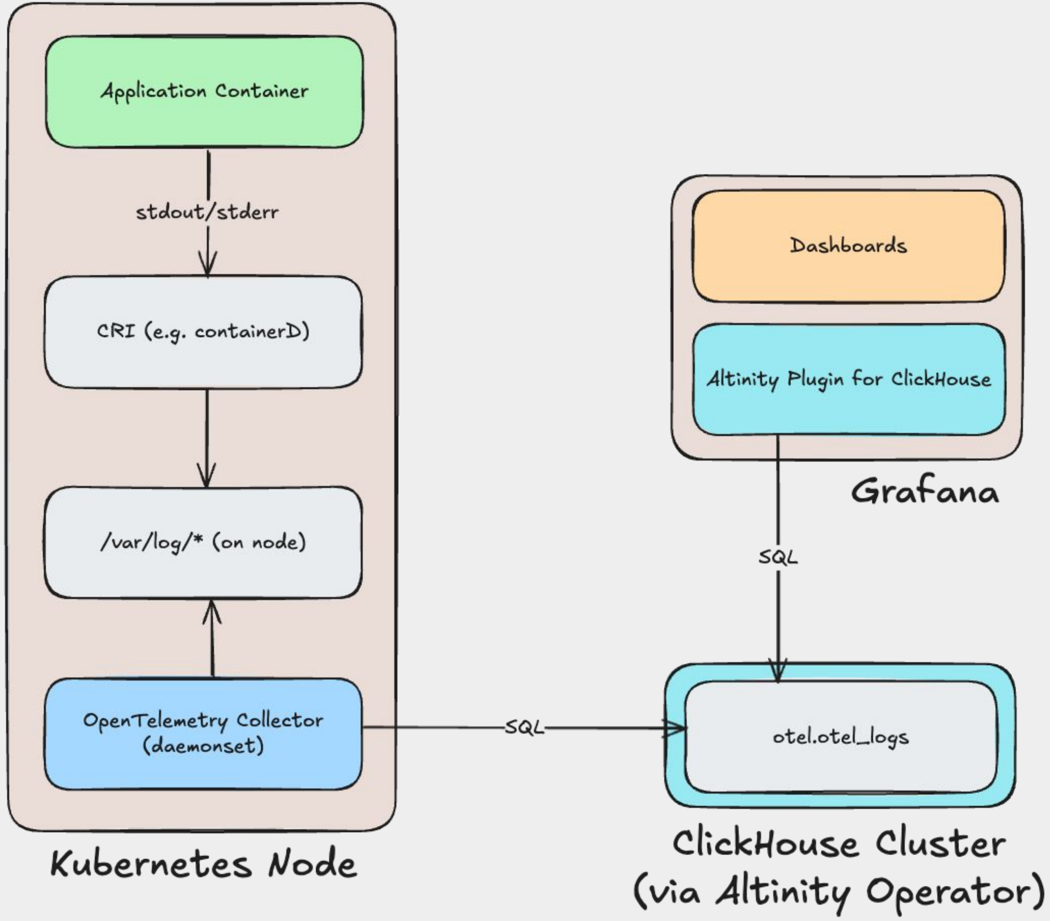
Filelog Receiver

K8s attributes processor

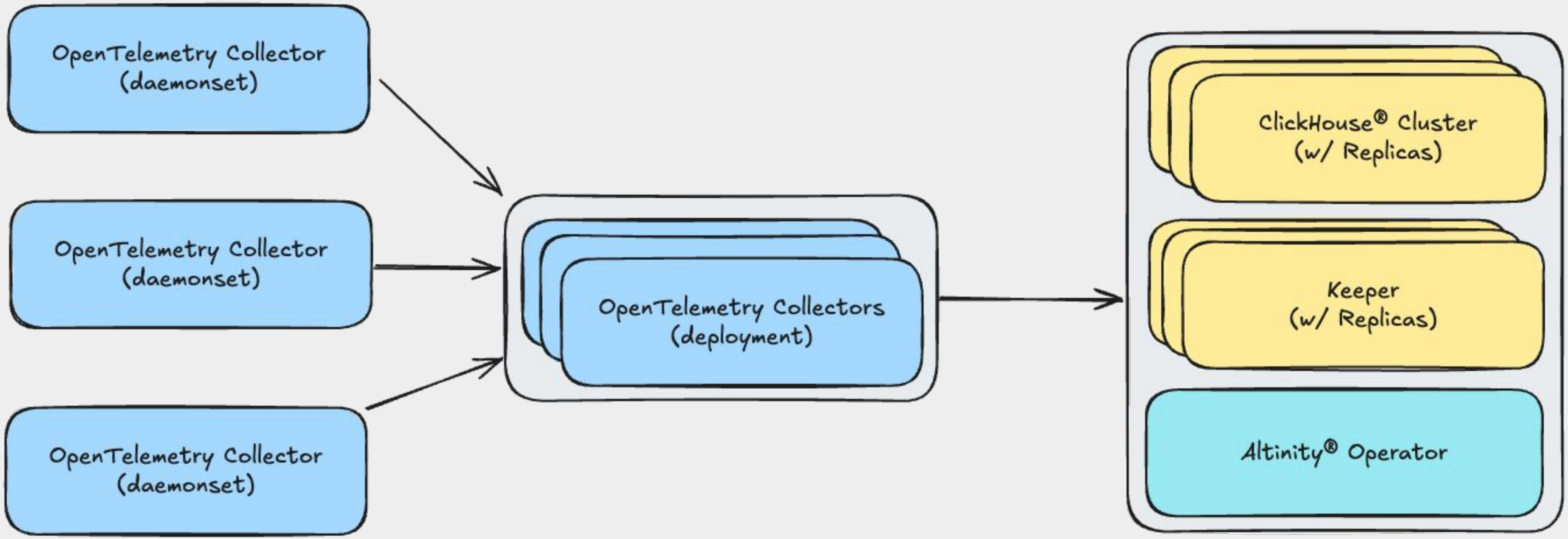
Batch processor

ClickHouse Exporter

OpenTelemetry Collector Pipeline


Application Container
stdout/stderr
CRI (e.g. containerD)
/var/log/* (on node)
OpenTelemetry Collector (daemonset)
Kubernetes Node
SQL
Dashboards
Altinity Plugin for ClickHouse
Grafana
SQL
otel.otel_logs
ClickHouse Cluster
(via Altinity Operator)

Finally… scaling for production


OpenTelemetry Collector
(daemonset)
OpenTelemetry Collector
(daemonset)
OpenTelemetry Collector
(daemonset)
OpenTelemetry Collectors
(deployment)
ClickHouse® Cluster
(w/ Replicas)
Keeper
(w/ Replicas)
Altinity® Operator

Cheap, fast, AND good?!

What have we learned?

- Keep it simple
- Start with what you’ve got
- OpenTelemetry is a swiss-army knife
- ClickHouse is pretty cool

Thank you and happy querying!

Josh Lee - Altinity

Connect with me

Resources & slides