Observability Platform for KDB+

SMBC日興証券株式会社
Equity System Development 部
Marcus Izumi

2025年08月27日

いっしょに、明日のこと。
Share the Future

SMBC日興証券

About

- High-Performance time-series database for real-time/historic market data

- q language: query, transform, analyze data on KDB+

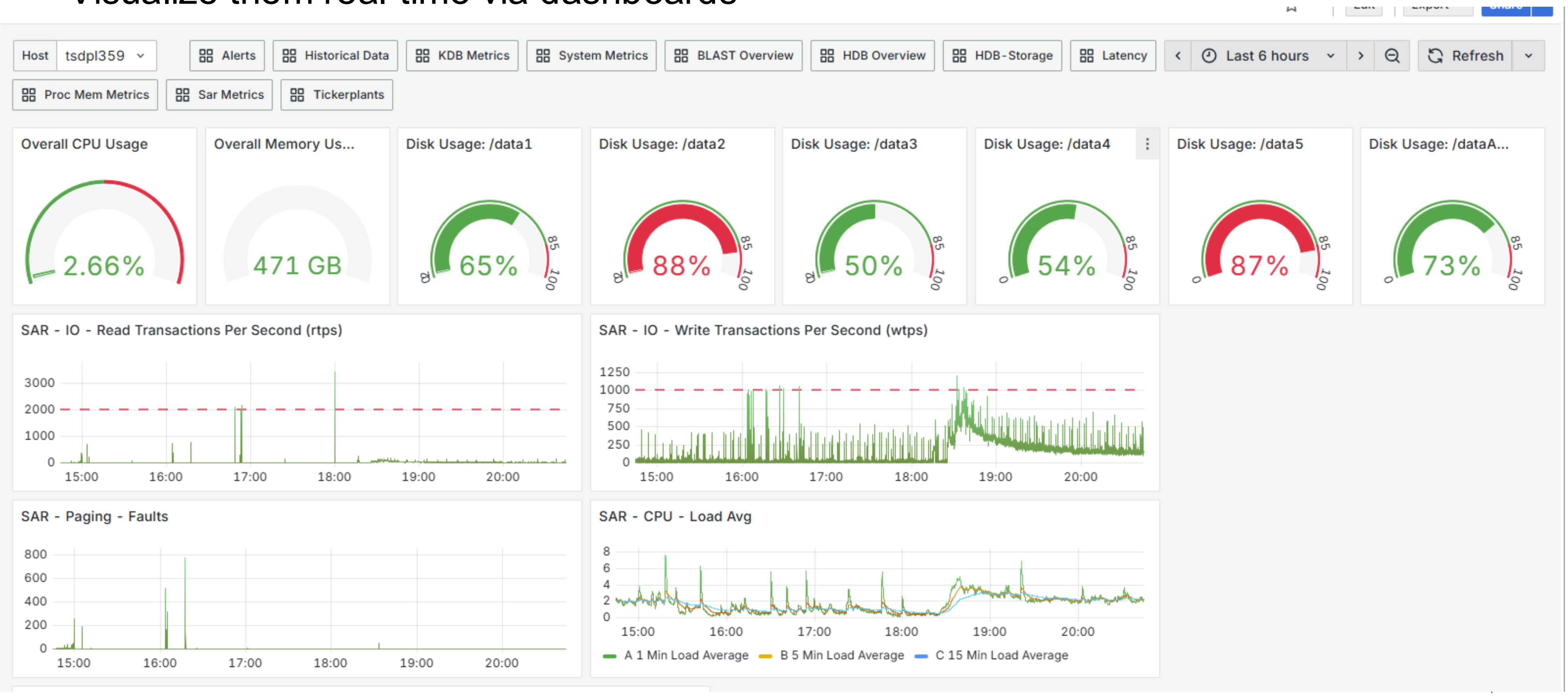
About

End Goal:

- Building an observability platform for KDB+

- Collect various metrics such as system metrics, market data metrics, historic metrics

- Visualize them real-time via dashboards



Host tsdpl359

Last 6 hours

Summary

1. Project Overview
2. Setting up KDB+ Database
3. Capturing Metrics
4. Visualizing Metrics
5. The Product
6. Reflections

Project Overview

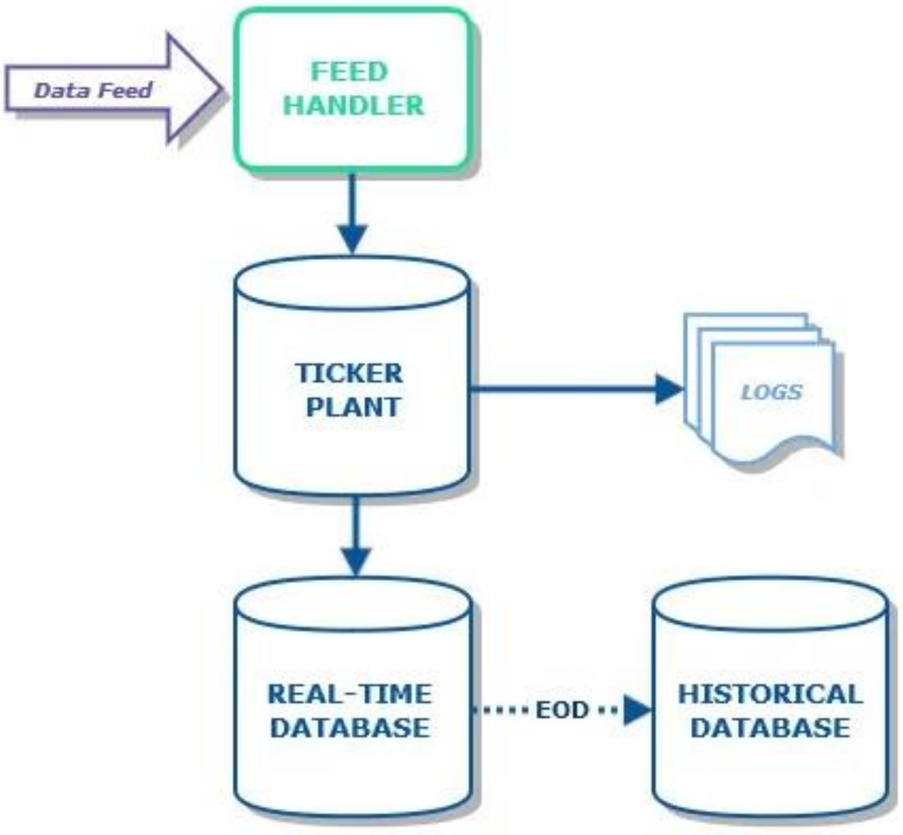
Project Milestones

1. Set up KDB+ databases to capture metrics

2. Collect various metrics and store them into database
   - System metrics, market data metrics, etc.

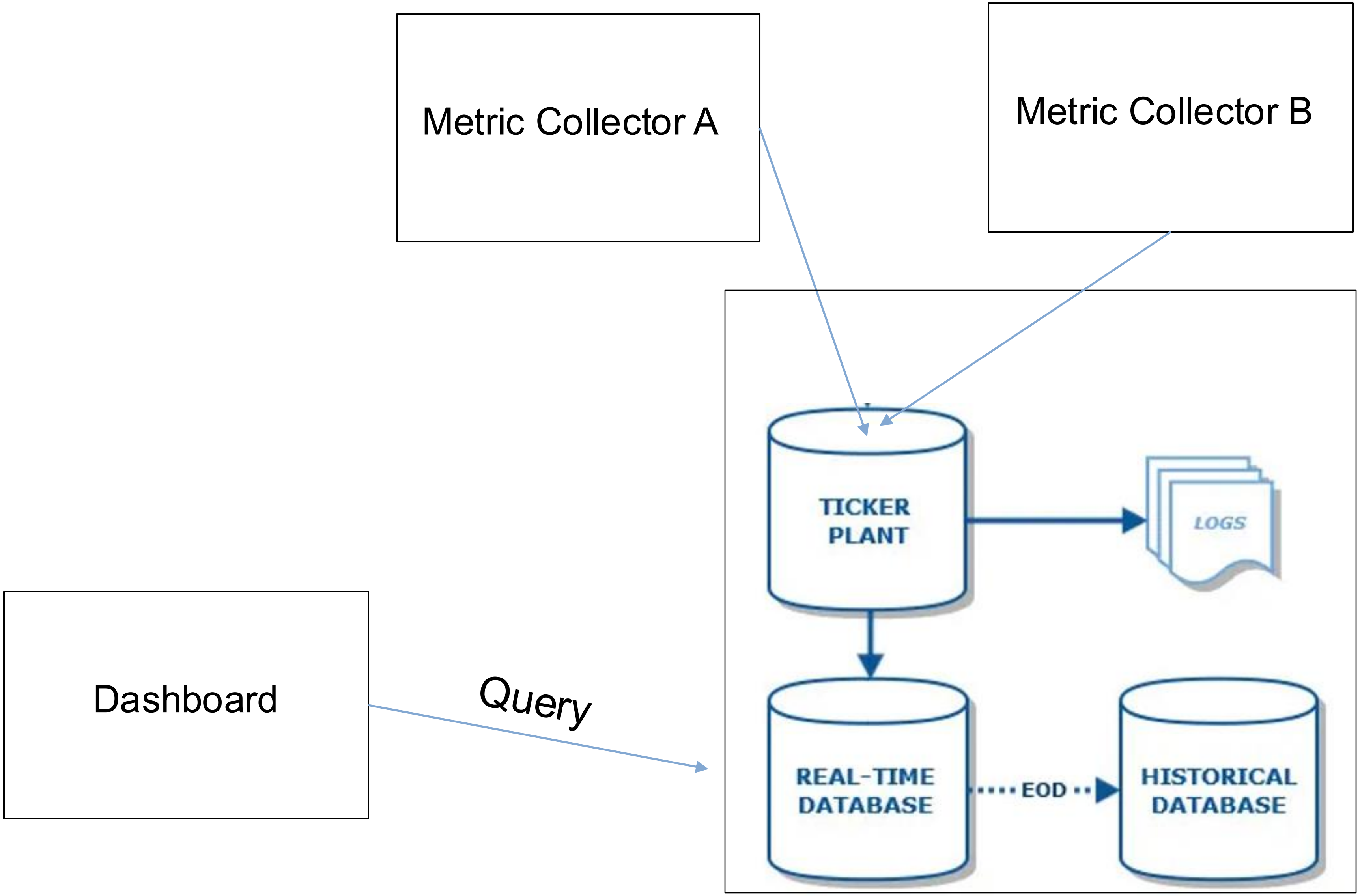
3. Connect metrics database to dashboard for visualization

Basic Architecture of KDB+ systems



- Individual processes written in the q language

Architecture of the Observability Platform

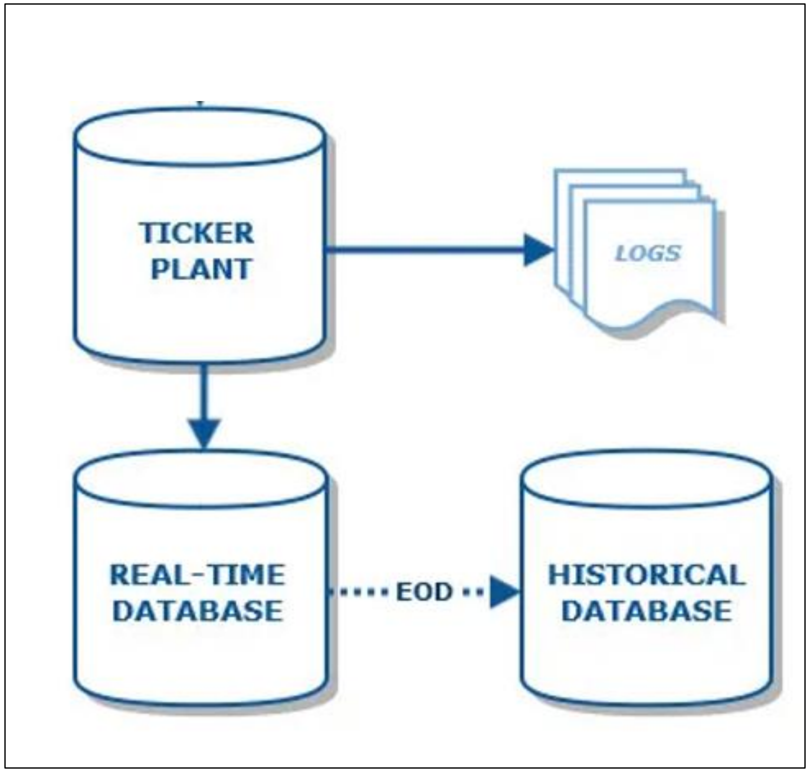


Setting up KDB+ databases

Setting up KDB+ Databases to Store Metrics

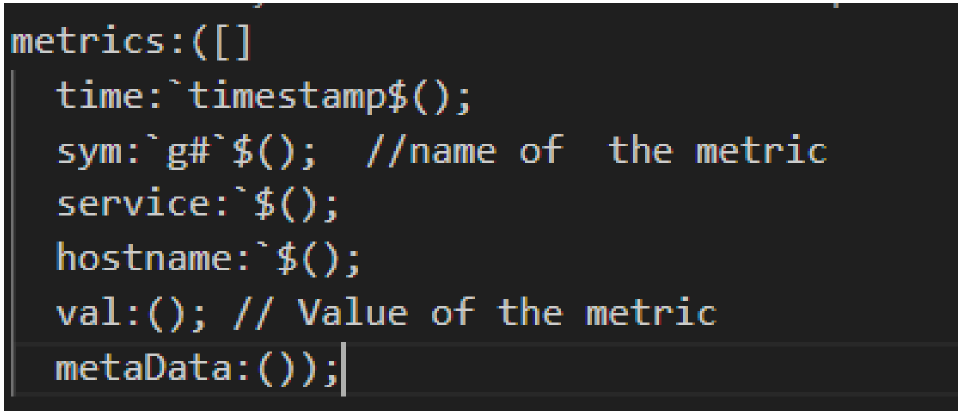
- To-Do: Define schema to store metrics

Existing Code



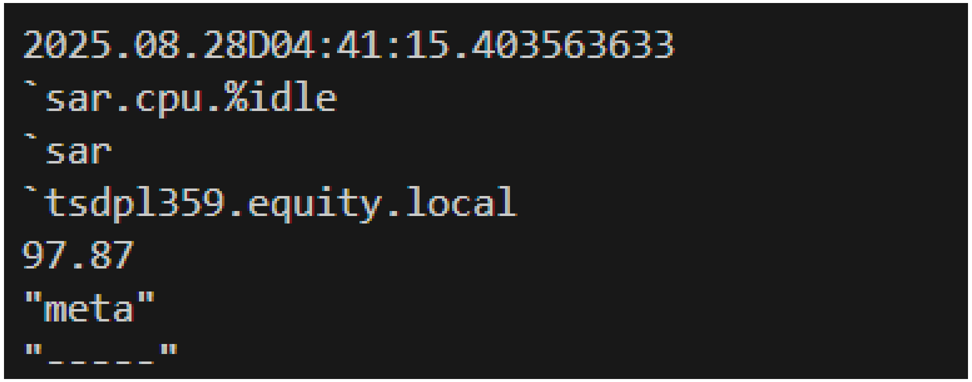
Setting up KDB+ database schema

Defining schema for database:



```
metrics:([]
  time:`timestamp$();
  sym:`g#`$();  //name of  the metric
  service:`$();
  hostname:`$();
  val:(); // Value of the metric
  metaData:());
```

Sample Record:



```
2025.08.28D04:41:15.403563633
`sar.cpu.%idle
`sar
`tsdpl359.equity.local
97.87
"meta"
"-----"
```

- Allow queries by metric name

Collecting Various Metrics

Different Levels of Metrics

- System metrics
  - CPU usage, mem usage, disk I/O

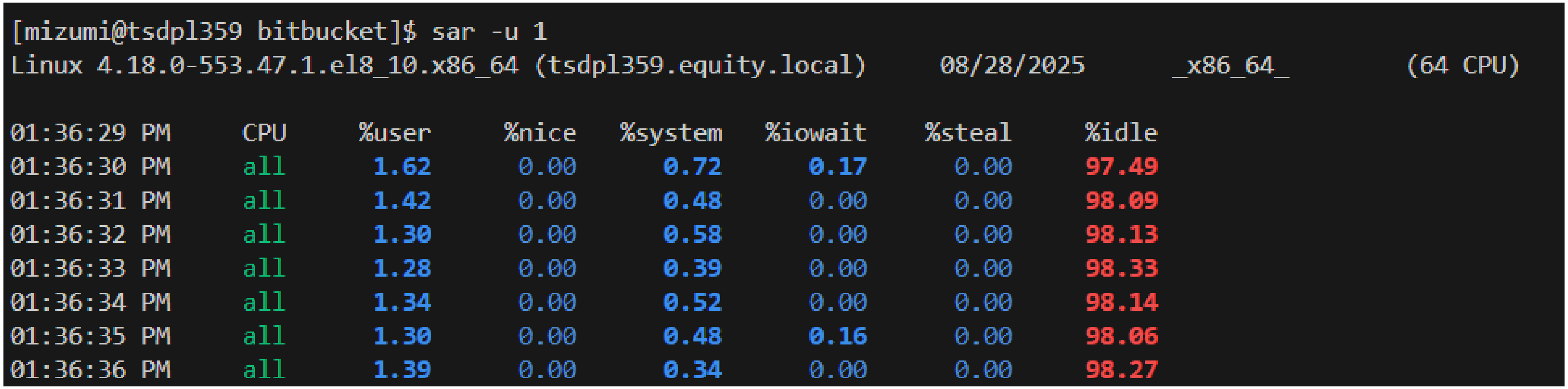
- Market data metrics
  - Table growth, pipeline latencies

- KDB+ in-process metrics
  - # of queries processed by KDB+ database

- Historic Metrics
  - HDB storage sizes

Collecting system metrics

SAR – System Activity Report



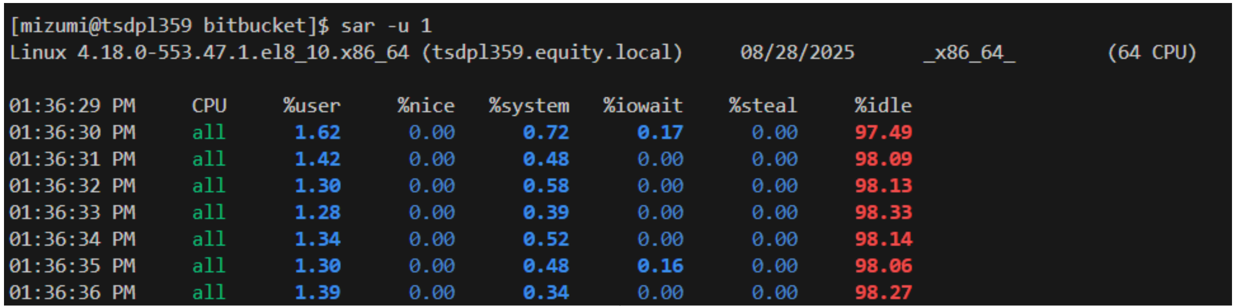
```
[mizumi@tsdpl359 bitbucket]$ sar -u 1
Linux 4.18.0-553.47.1.el8_10.x86_64 (tsdpl359.equity.local)    08/28/2025      _x86_64_        (64 CPU)

01:36:29 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
01:36:30 PM     all      1.62      0.00      0.72      0.17      0.00     97.49
01:36:31 PM     all      1.42      0.00      0.48      0.00      0.00     98.09
01:36:32 PM     all      1.30      0.00      0.58      0.00      0.00     98.13
01:36:33 PM     all      1.28      0.00      0.39      0.00      0.00     98.33
01:36:34 PM     all      1.34      0.00      0.52      0.00      0.00     98.14
01:36:35 PM     all      1.30      0.00      0.48      0.16      0.00     98.06
01:36:36 PM     all      1.39      0.00      0.34      0.00      0.00     98.27
```

- Command-Line Linux Monitoring Tool

- Reports CPU, mem, disk I/O

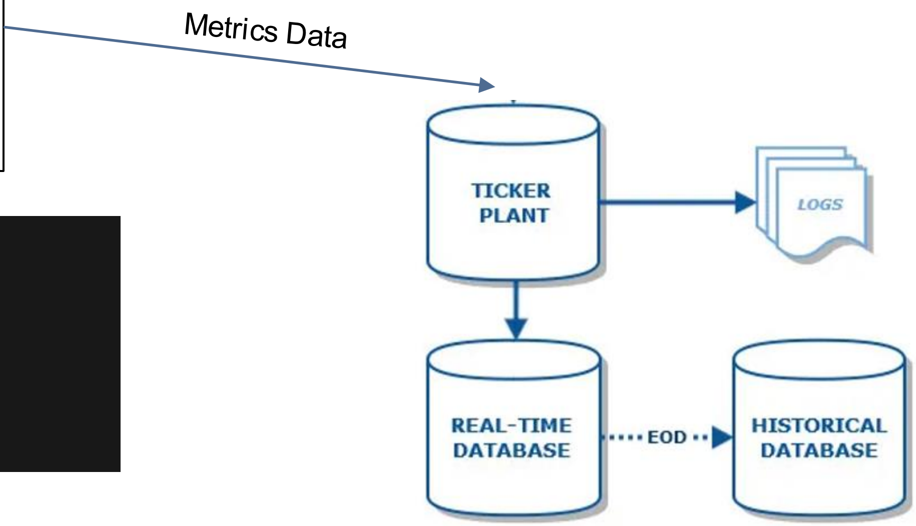
- Contains stats over time

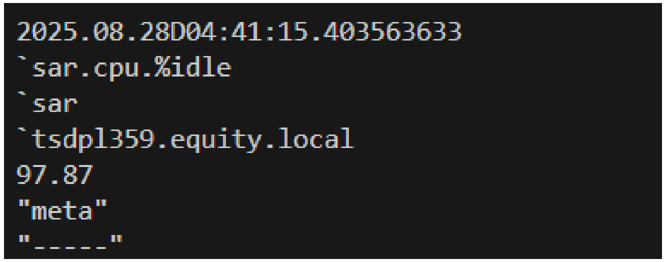
Creating Real-Time Sar Metrics Collector



```
[mizumi@tsdpl359 bitbucket]$ sar -u 1
Linux 4.18.0-553.47.1.el8_10.x86_64 (tsdpl359.equity.local)     08/28/2025      _x86_64_        (64 CPU)

01:36:29 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
01:36:30 PM     all      1.62      0.00      0.72      0.17      0.00     97.49
01:36:31 PM     all      1.42      0.00      0.48      0.00      0.00     98.09
01:36:32 PM     all      1.30      0.00      0.58      0.00      0.00     98.13
01:36:33 PM     all      1.28      0.00      0.39      0.00      0.00     98.33
01:36:34 PM     all      1.34      0.00      0.52      0.00      0.00     98.14
01:36:35 PM     all      1.30      0.00      0.48      0.16      0.00     98.06
01:36:36 PM     all      1.39      0.00      0.34      0.00      0.00     98.27
```

System Metrics Collector



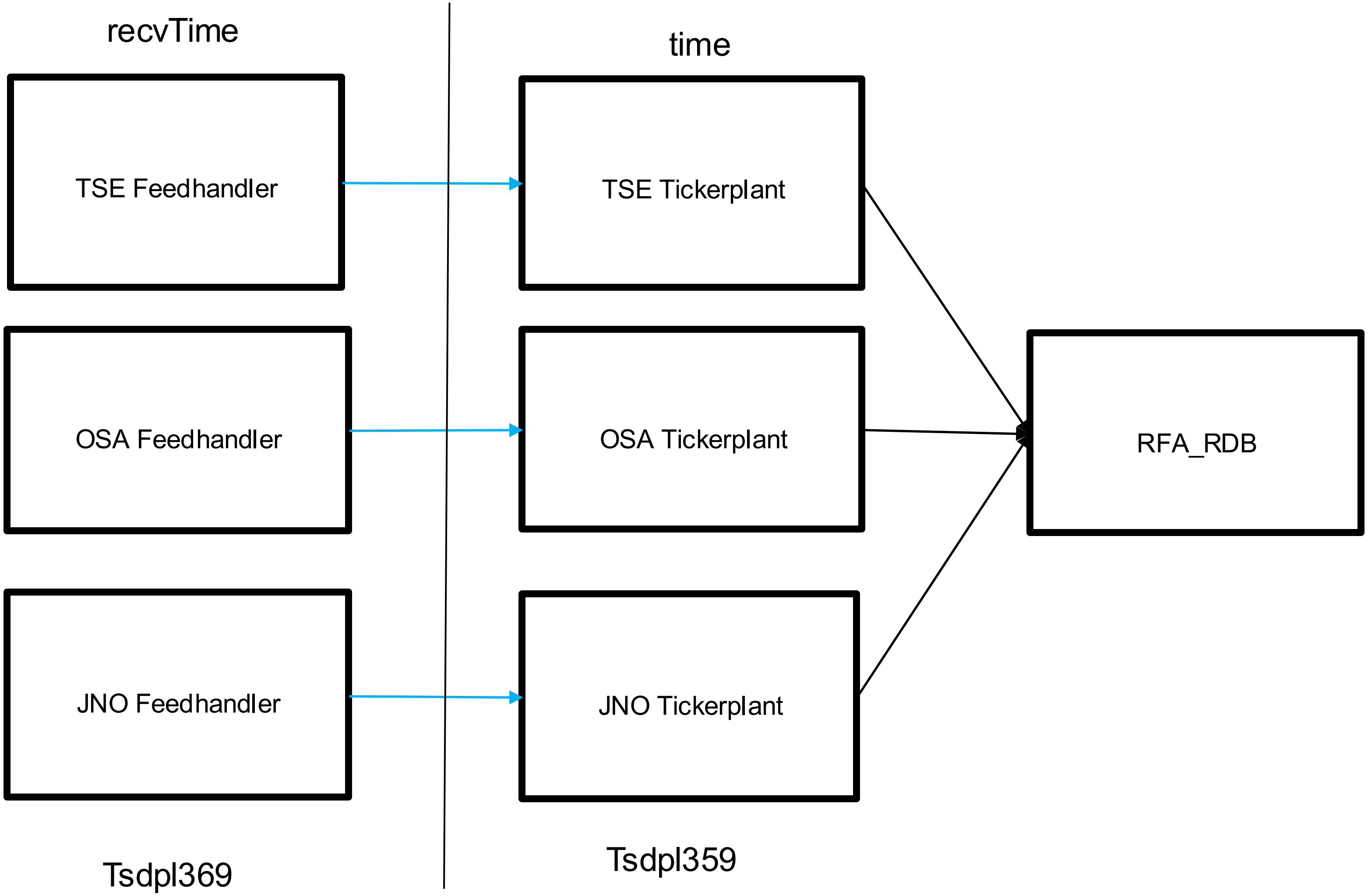


```
2025.08.28D04:41:15.403563633
`sar.cpu.%idle
`sar
`tsdpl359.equity.local
97.87
"meta"
"-----"
```

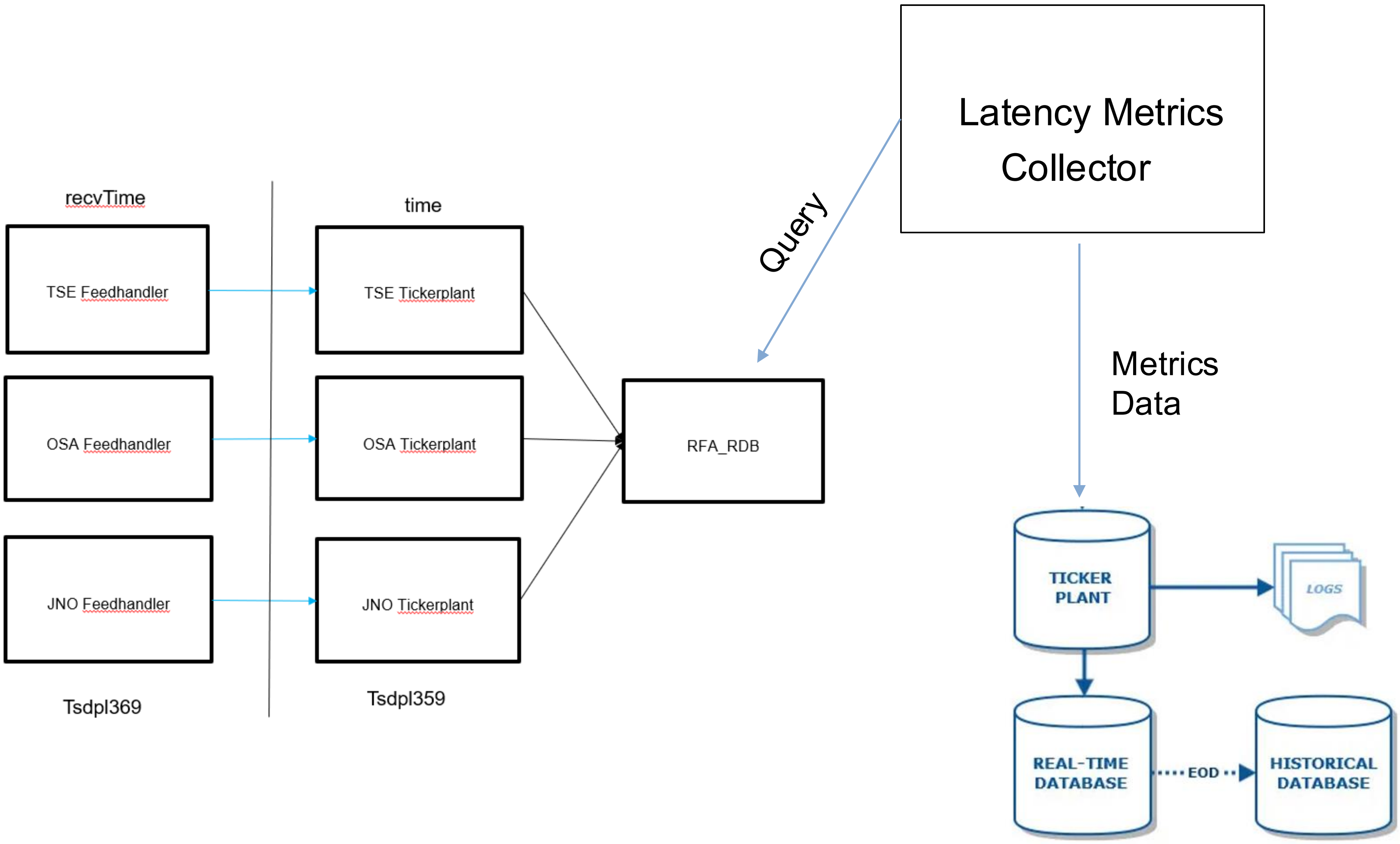
Collecting Market Data Metrics

Market Data Metrics: Pipeline Latency

Pipeline Latency: time between feed handler and ticker plant

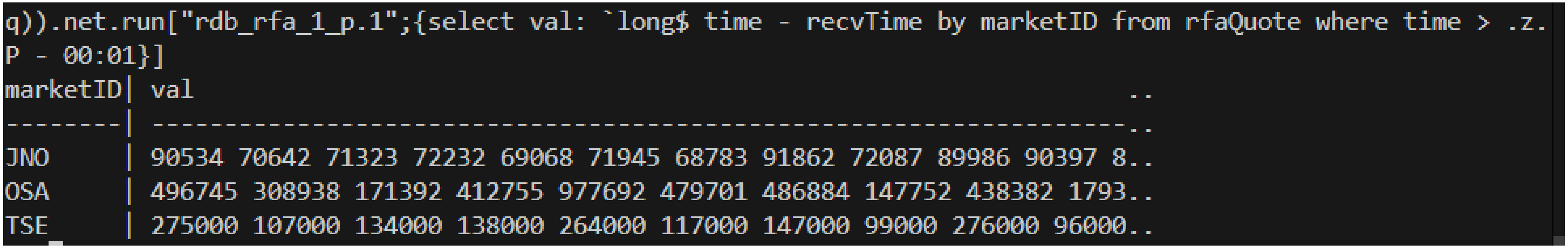


Market Data Metrics: Pipeline Latency



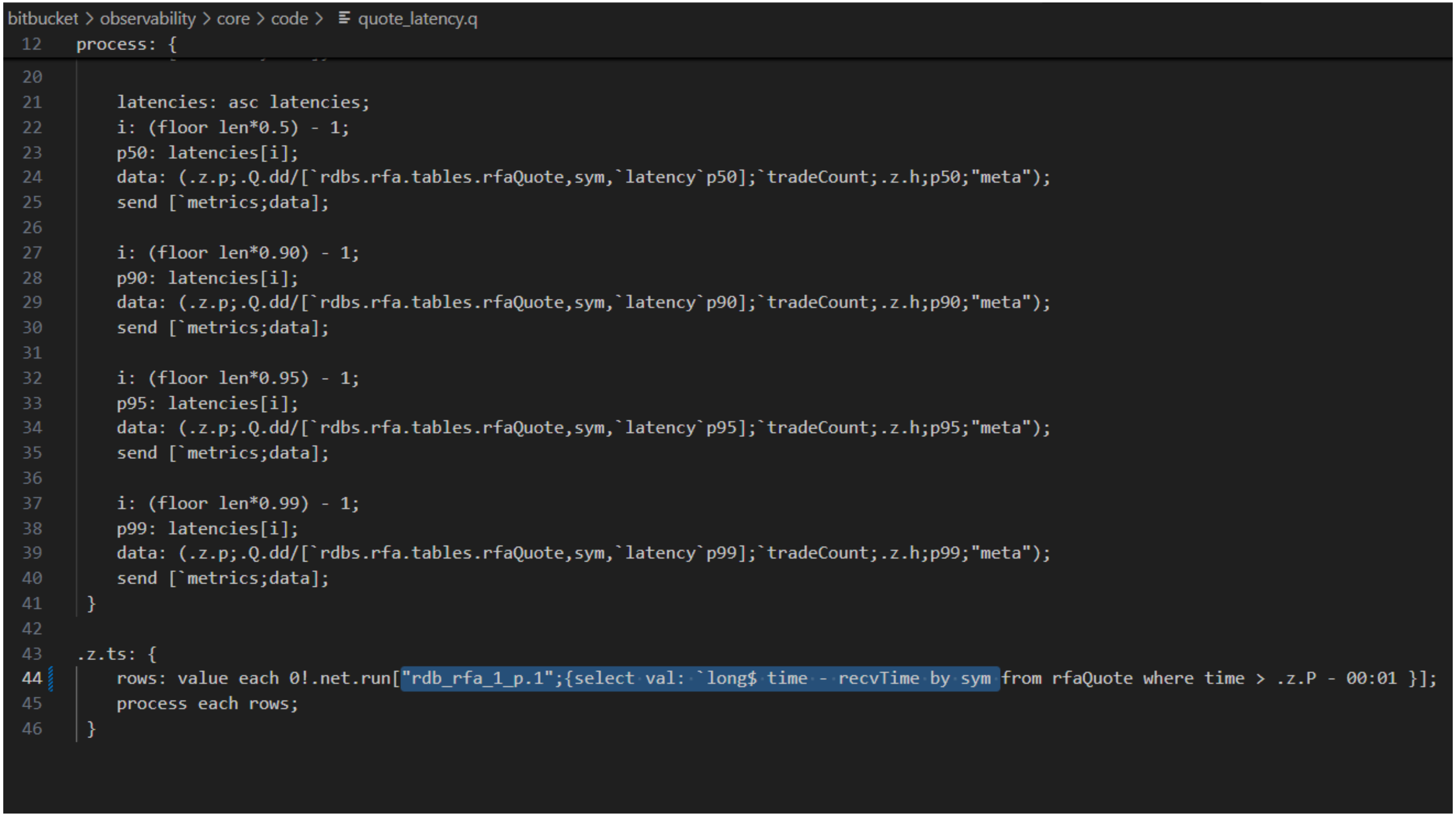
Resulting Market Data Metrics per Market

Summarizing per market is less expensive than per instrument – less memory allocations



```
q)).net.run["rdb_rfa_1_p.1";{select val: `long$ time - recvTime by marketID from rfaQuote where time > .z.
P - 00:01}]
marketID| val                                                                ..
--------| -------------------------------------------------------------------..
JNO     | 90534 70642 71323 72232 69068 71945 68783 91862 72087 89986 90397 8..
OSA     | 496745 308938 171392 412755 977692 479701 486884 147752 438382 1793..
TSE     | 275000 107000 134000 138000 264000 117000 147000 99000 276000 96000..
```

Computing Latencies: p50, p90, p95, p99



bitbucket > observability > core > code > quote_latency.q

```
process: {

    latencies: asc latencies;
    i: (floor len*0.5) - 1;
    p50: latencies[i];
    data: (.z.p;.Q.dd/[`rdbs.rfa.tables.rfaQuote,sym,`latency`p50];`tradeCount;.z.h;p50;"meta");
    send [`metrics;data];

    i: (floor len*0.90) - 1;
    p90: latencies[i];
    data: (.z.p;.Q.dd/[`rdbs.rfa.tables.rfaQuote,sym,`latency`p90];`tradeCount;.z.h;p90;"meta");
    send [`metrics;data];

    i: (floor len*0.95) - 1;
    p95: latencies[i];
    data: (.z.p;.Q.dd/[`rdbs.rfa.tables.rfaQuote,sym,`latency`p95];`tradeCount;.z.h;p95;"meta");
    send [`metrics;data];

    i: (floor len*0.99) - 1;
    p99: latencies[i];
    data: (.z.p;.Q.dd/[`rdbs.rfa.tables.rfaQuote,sym,`latency`p99];`tradeCount;.z.h;p99;"meta");
    send [`metrics;data];
 }

.z.ts: {
    rows: value each 0!.net.run["rdb_rfa_1_p.1";{select val: `long$ time - recvTime by sym from rfaQuote where time > .z.P - 00:01 }];
    process each rows;
 }
```

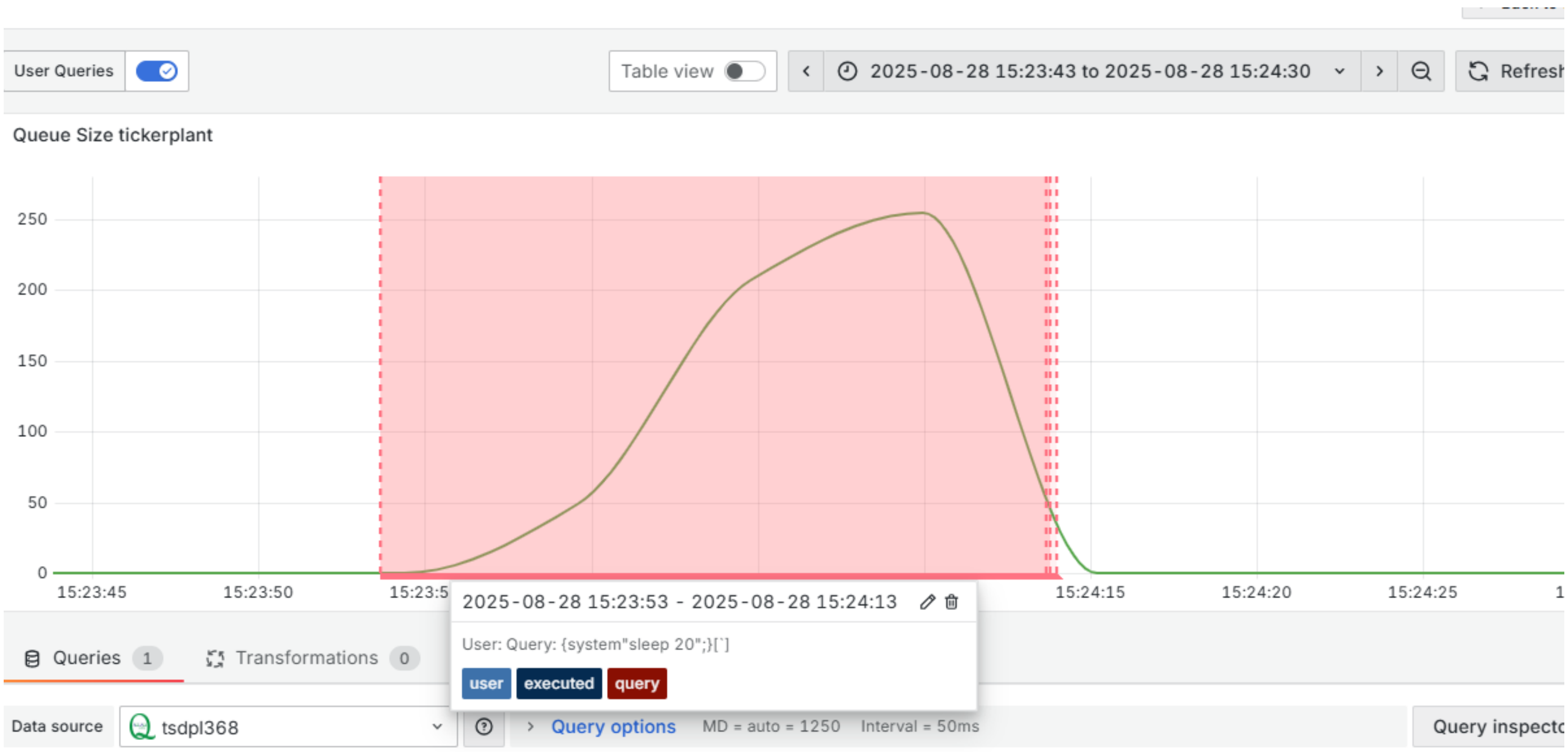
Visualization will come later!

Collecting KDB+’s In-Process Metrics

In-process metrics of KDB+ processes

Motivation:

- Collect query info from KDB+ processes
- Monitor load on KDB+ processes realtime
- Ability to cross-reference queries against system metrics in dashboard



Queue Size tickerplant

2025-08-28 15:23:43 to 2025-08-28 15:24:30

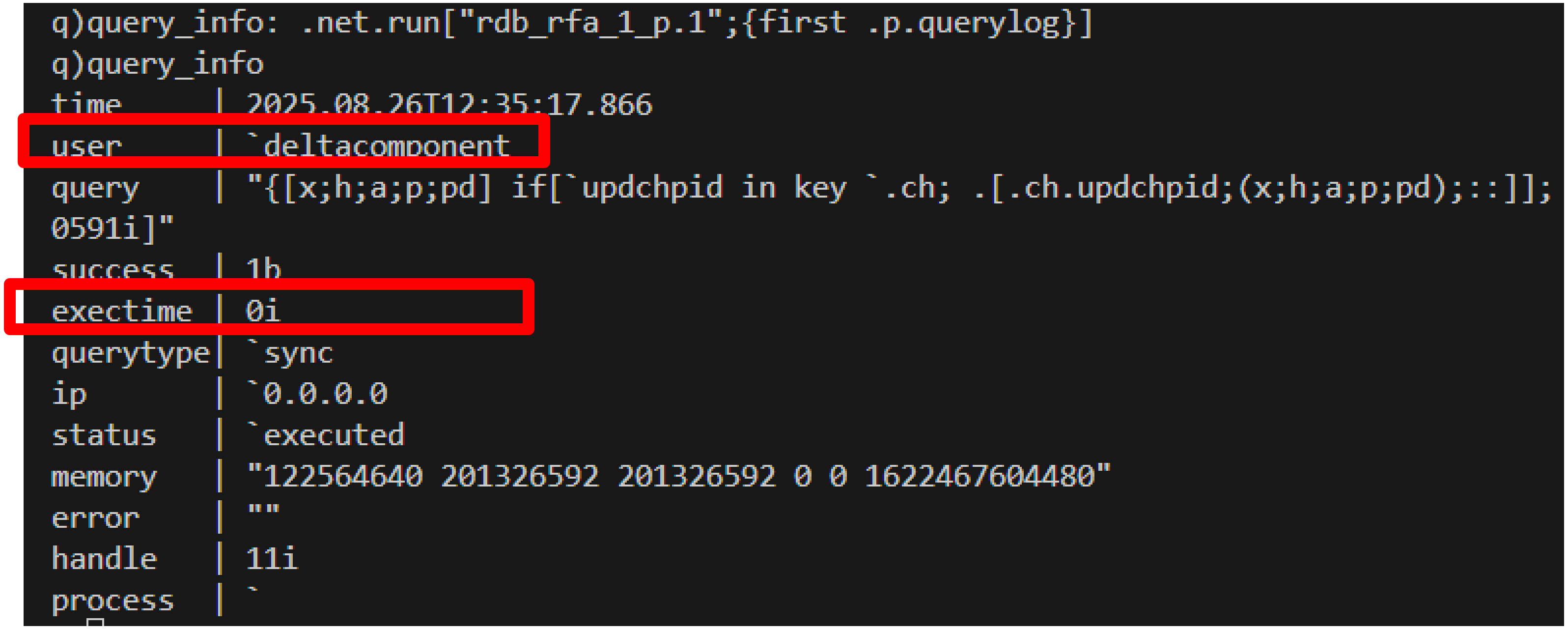
2025-08-28 15:23:53 - 2025-08-28 15:24:13

User: Query: {system"sleep 20";}[`]

Data source: tsdpl368

Query options MD = auto = 1250 Interval = 50ms

Accessing Query-logs of KDB+ Processes

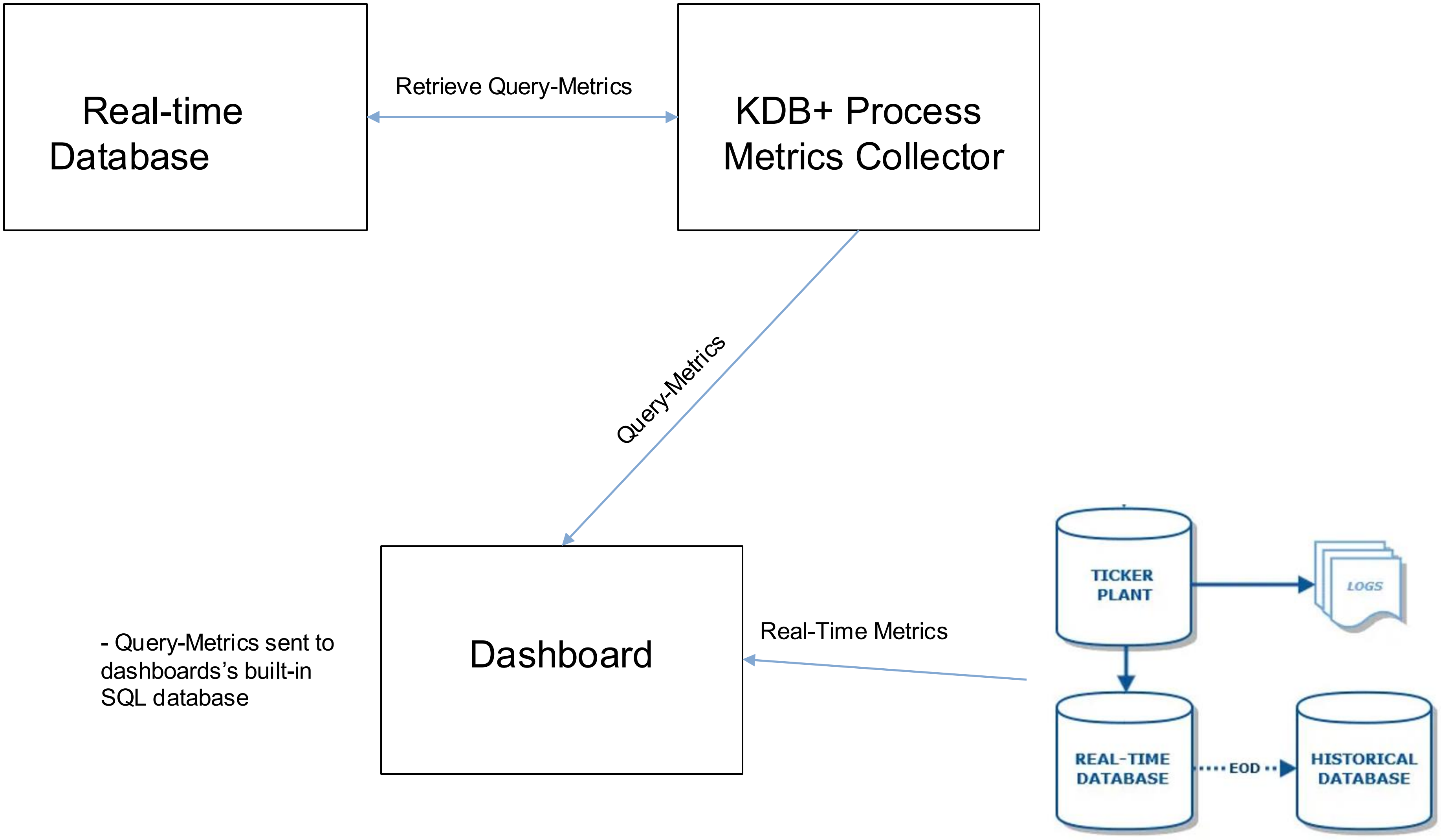


```
q)query_info: .net.run["rdb_rfa_1_p.1";{first .p.querylog}]
q)query_info
time     | 2025.08.26T12:35:17.866
user     | `deltacomponent
query    | "{[x;h;a;p;pd] if[`updchpid in key `.ch; .[.ch.updchpid;(x;h;a;p;pd);::]];
0591i]"
success  | 1b
exectime | 0i
querytype| `sync
ip       | `0.0.0.0
status   | `executed
memory   | "122564640 201326592 201326592 0 0 1622467604480"
error    | ""
handle   | 11i
process  | `
```

- KDB+ processes in our system maintain a querylog[]

- Can generate metrics such as query timespans, query size

Accessing Query-logs of KDB+ Processes



Real-time Database

Retrieve Query-Metrics

KDB+ Process Metrics Collector

Query-Metrics

Dashboard

Real-Time Metrics

- Query-Metrics sent to dashboards’s built-in SQL database

Connecting to Dashboard for Visualization

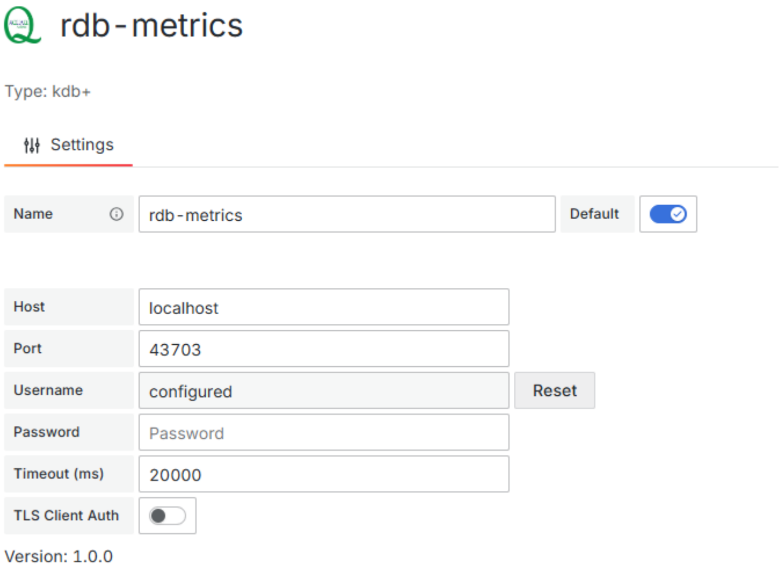
Why Grafana?

- Rich visualizations – heatmaps, gauges

- Alert rules through a Symphony integration

Connecting Metrics to Dashboard:

Using KDB+ plug-ins on Grafana…



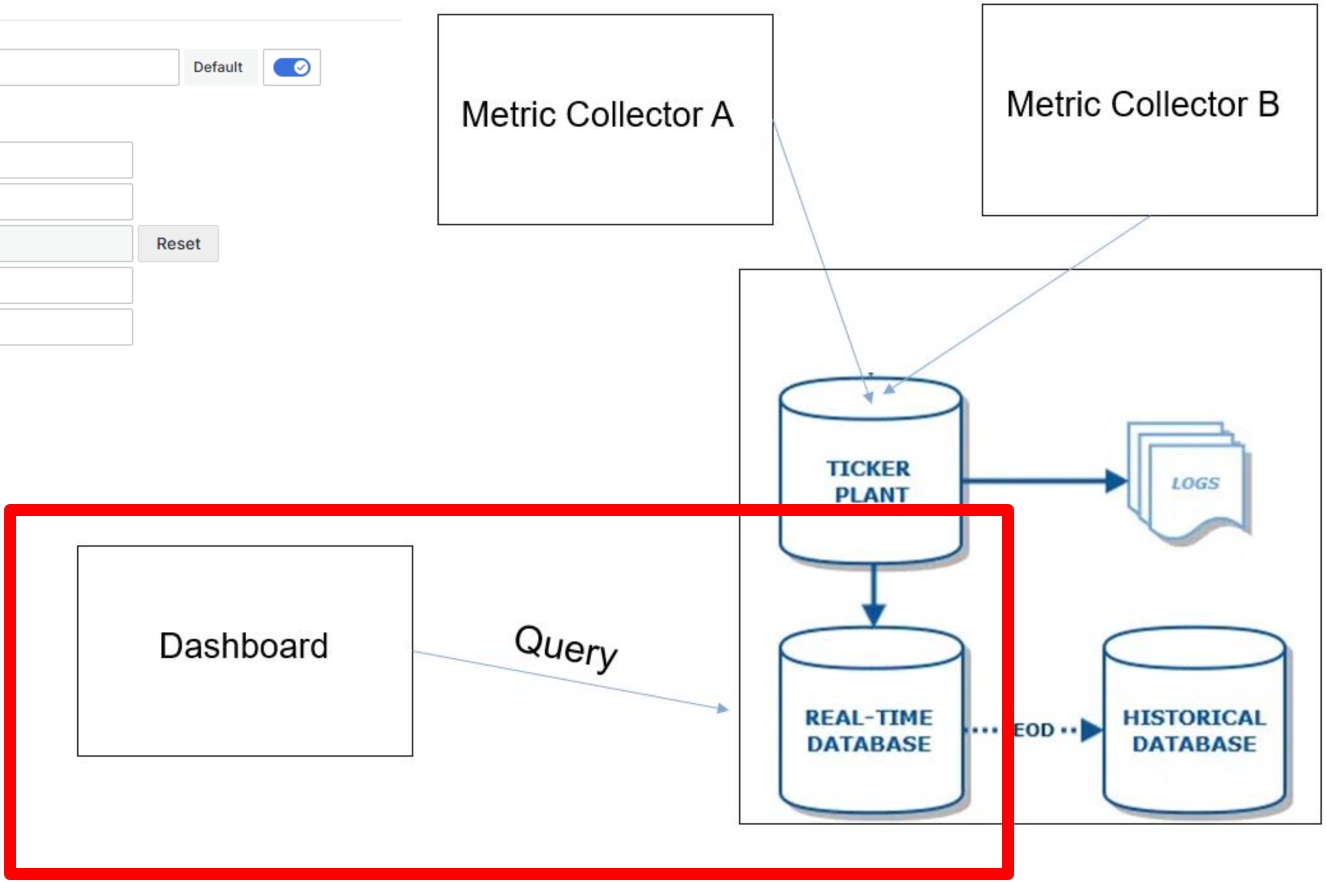
rdb-metrics

Type: kdb+

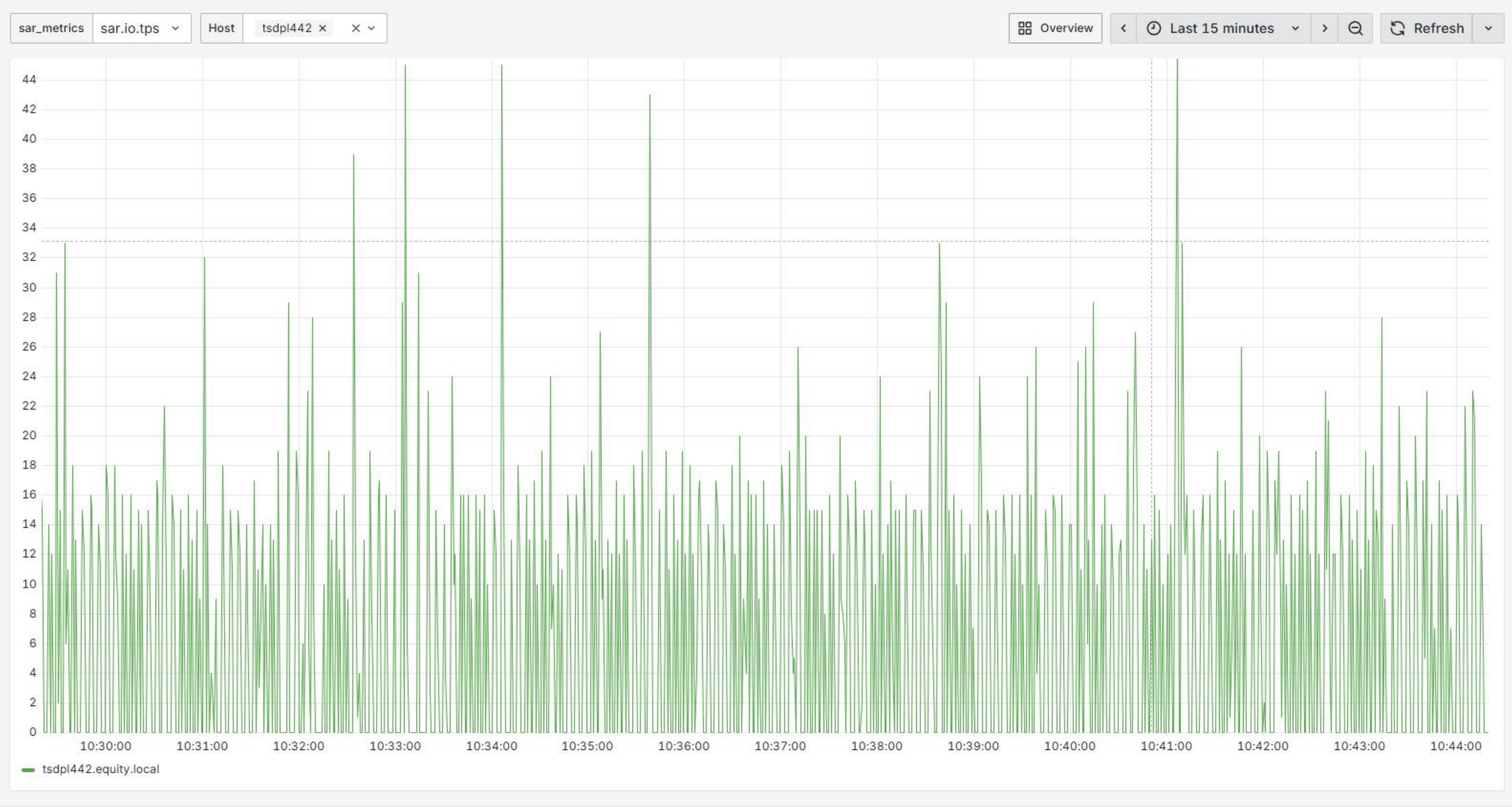
Settings

Name	rdb-metrics	Default	●
Host	localhost		
Port	43703		
Username	configured	Reset	
Password	Password		
Timeout (ms)	20000		
TLS Client Auth	○		

Version: 1.0.0

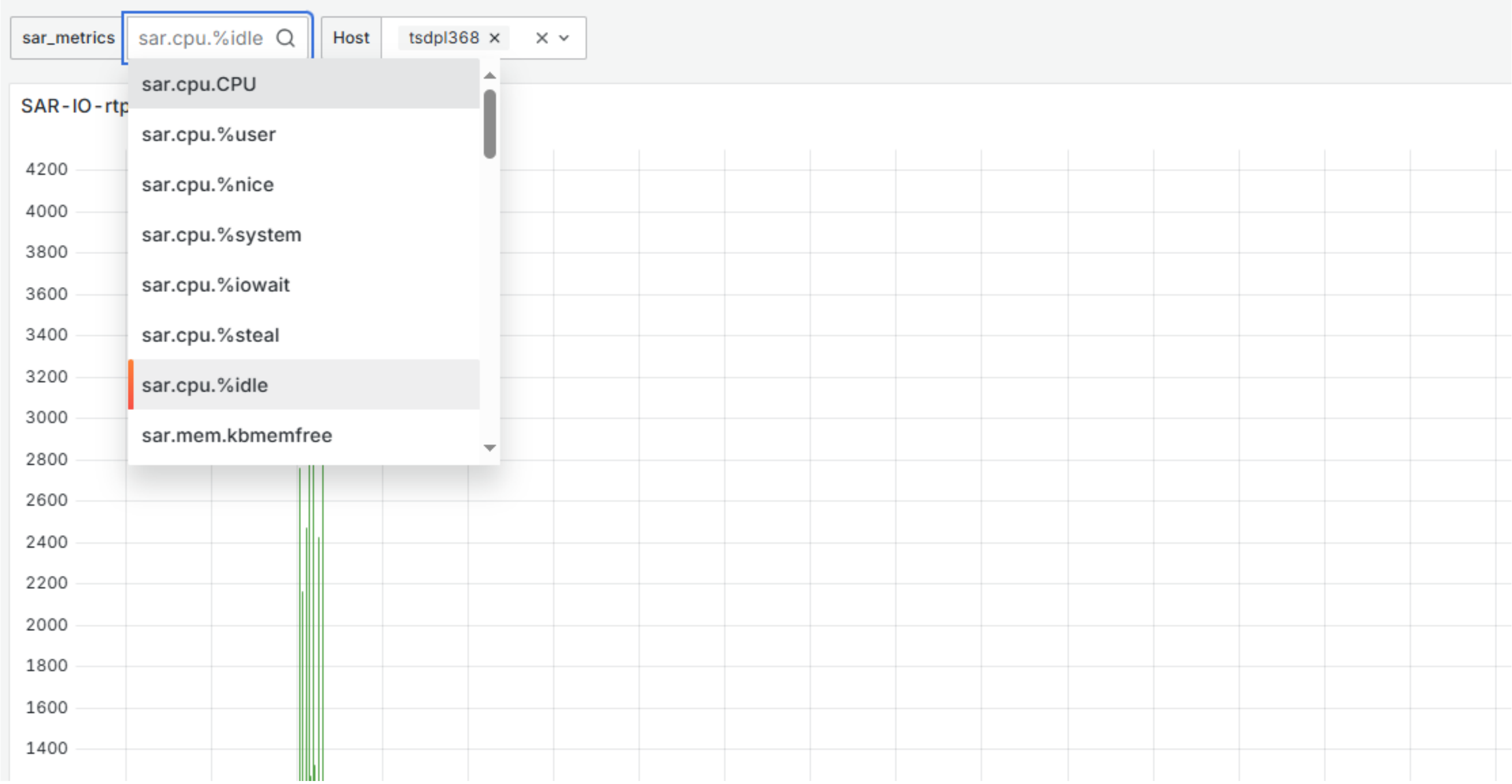


Time-series Visualization!



Using Variable Features on Grafana

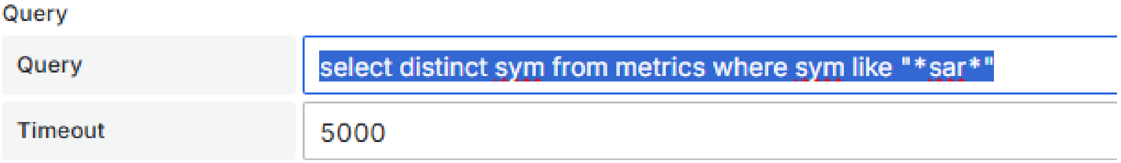
Drop-down list of interested metrics



sar_metrics: sar.cpu.%idle

Host: tsdpl368

Metrics shown are queried from metrics RDB



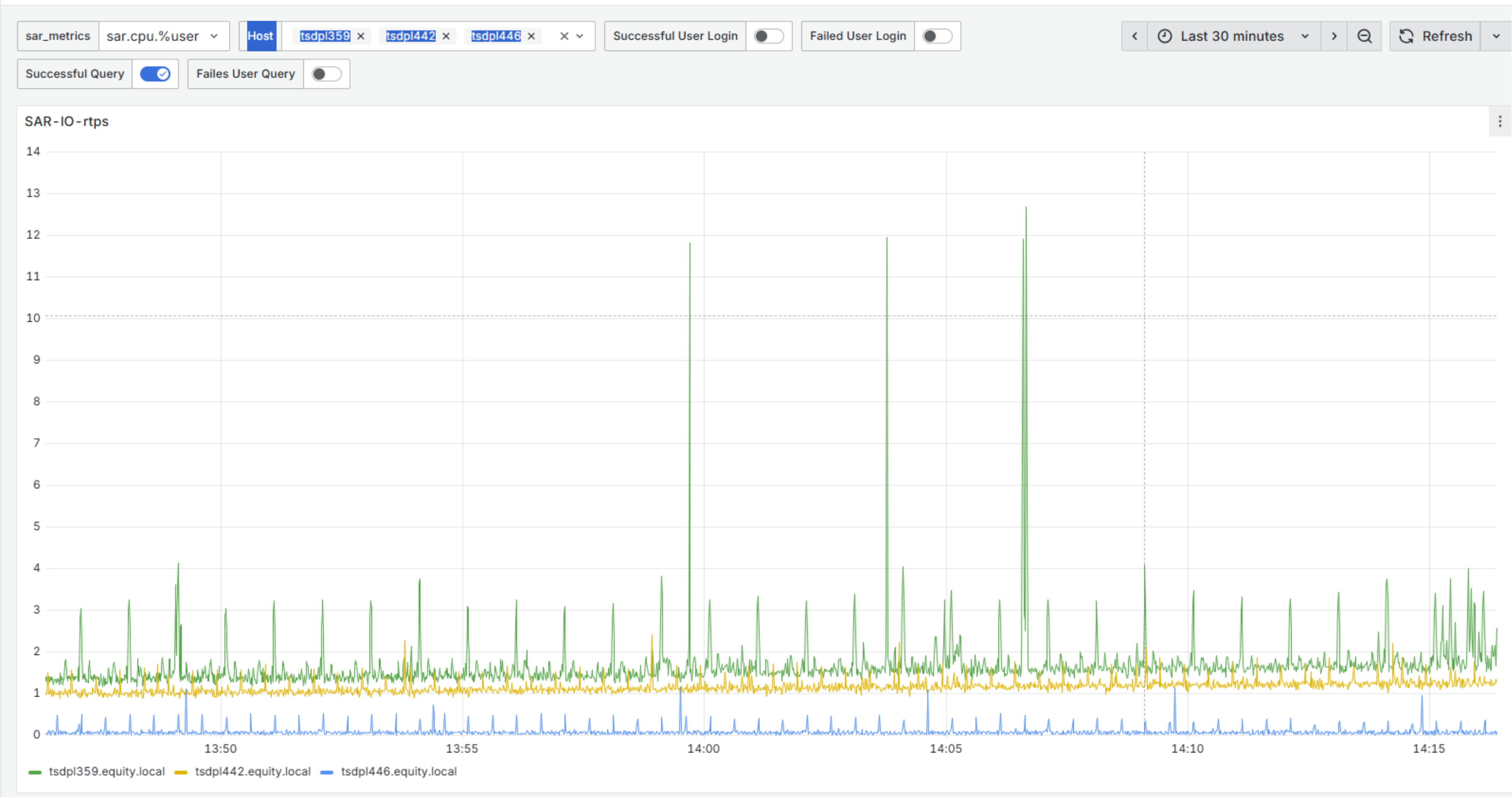
Query

Query	select distinct sym from metrics where sym like "*sar*"
Timeout	5000

Inspecting Different Hosts on Dashboards

Entire stack running on different hosts – each hosts store smaller subset of data; queries will run faster

Grafana can parallelize queries on different host – scalable as we add new hosts – longer term we can extend to prod environment



sar_metrics sar.cpu.%user

Host tsdpl359 × tsdpl442 × tsdpl446 ×

Successful User Login ○

Failed User Login ○

Successful Query ●

Failes User Query ○

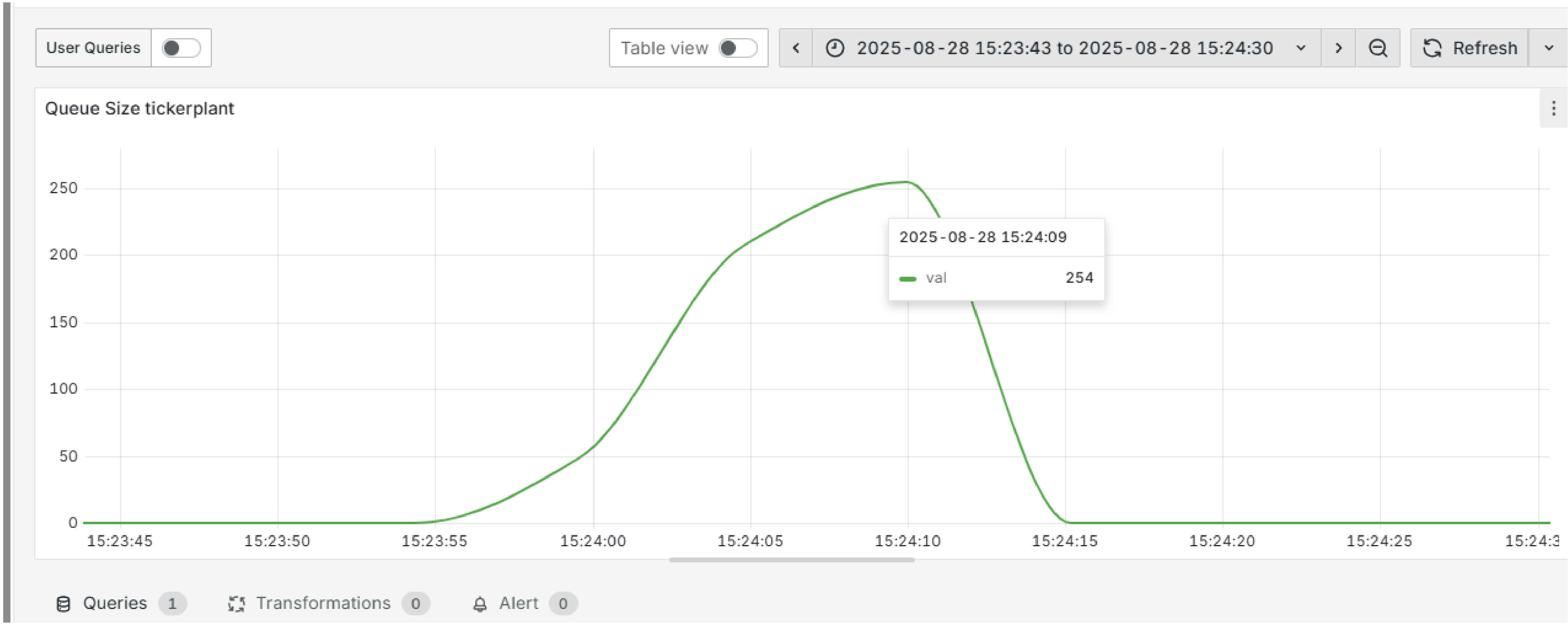
Last 30 minutes

SAR-IO-rtps

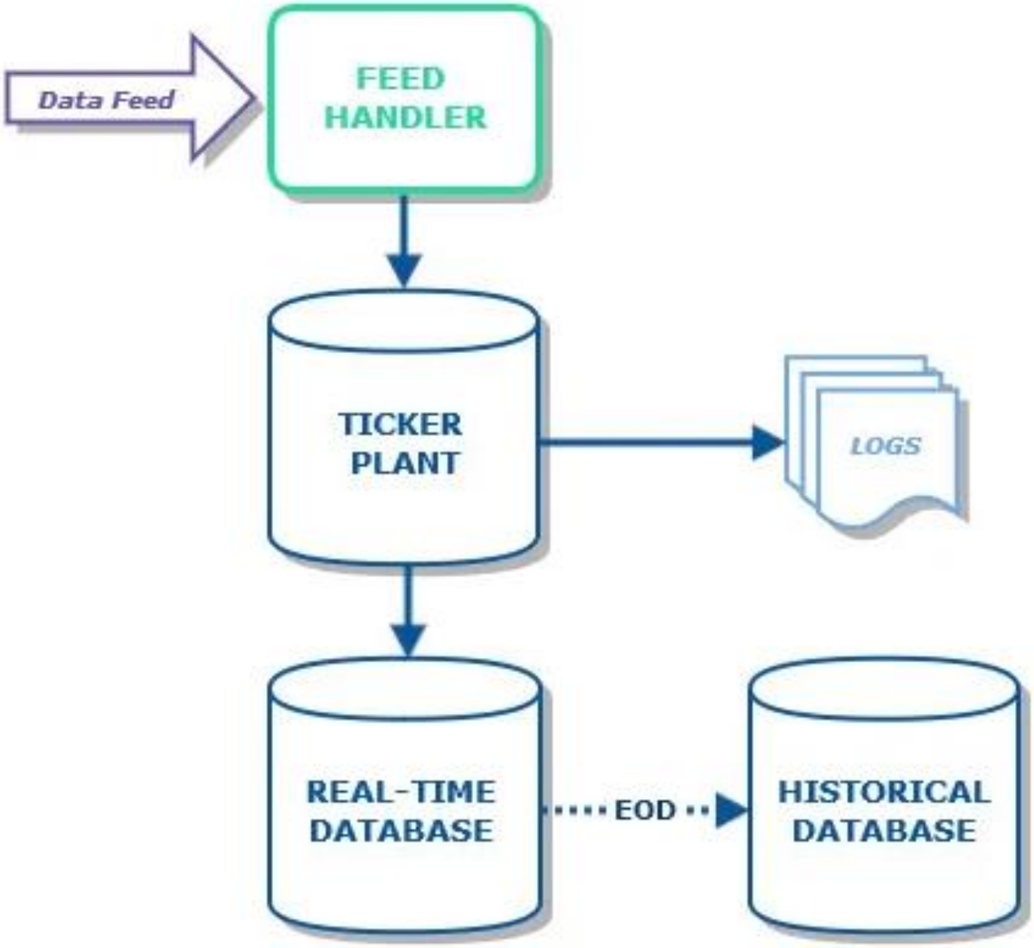
Example Dashboards

Use Case #1: Inspect Busy Processes by Tickerplant Queue Size

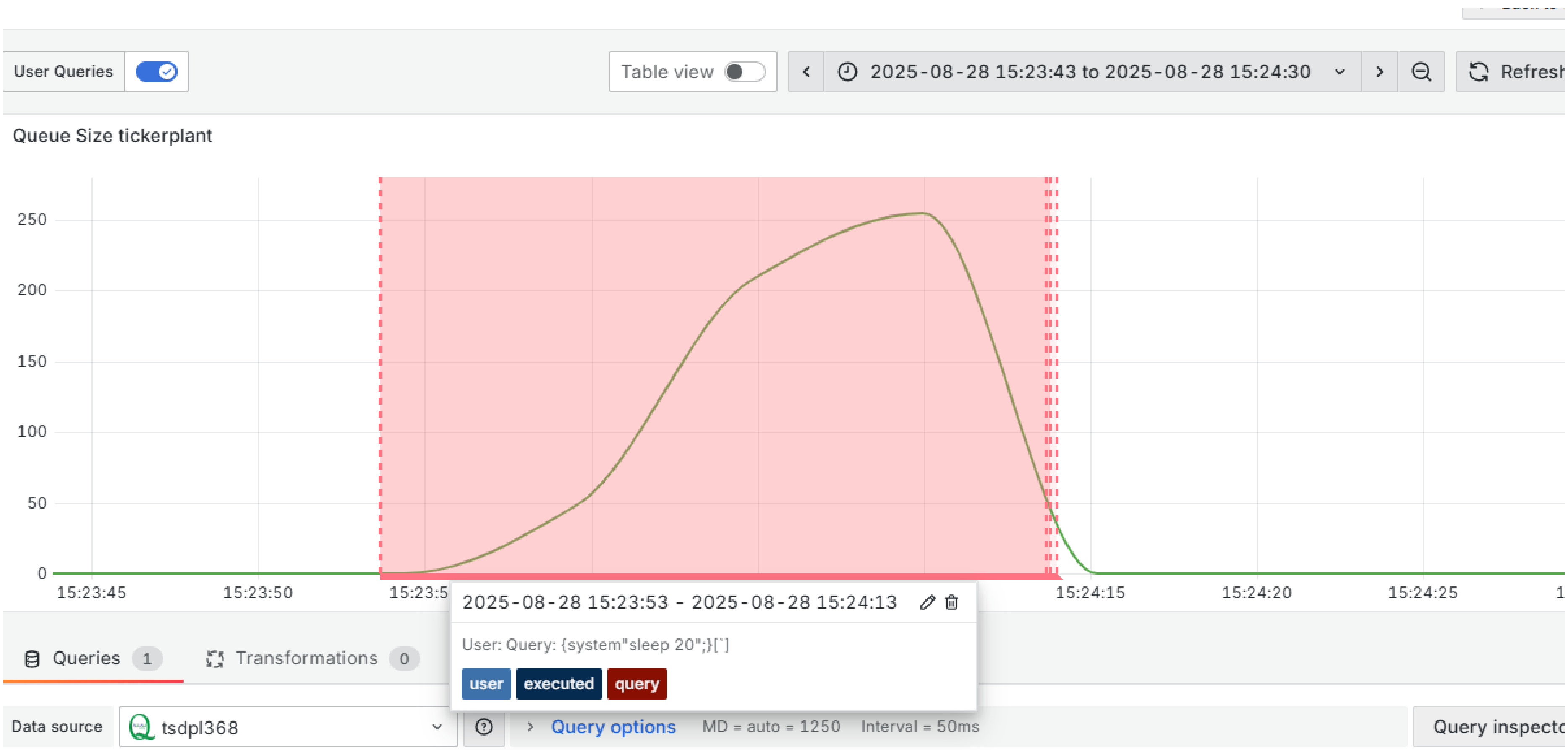
Tickerplant queue size for detecting long queries with real-time database



Use Case #1: Inspect Busy Processes by Tickerplant Queue Size



Use Case #1: Inspect Busy Processes by Tickerplant Queue Size



2025-08-28 15:23:53 - 2025-08-28 15:24:13

User: Query: {system"sleep 20";}[`]

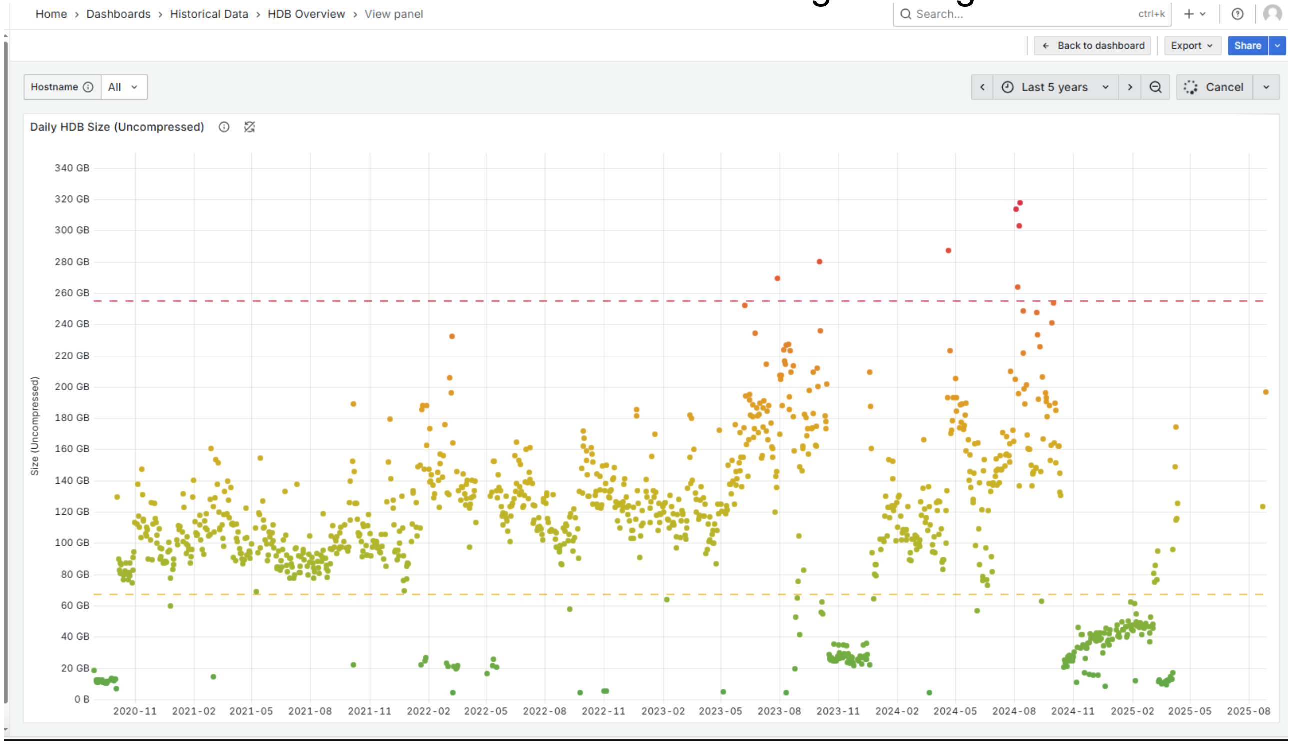
user executed query

Data source: tsdpl368

MD = auto = 1250 Interval = 50ms

Use Case #2: Visualizing growth of HDB

Breakdown of historical data to aid decisions in storage management



Home › Dashboards › Historical Data › HDB Overview › View panel

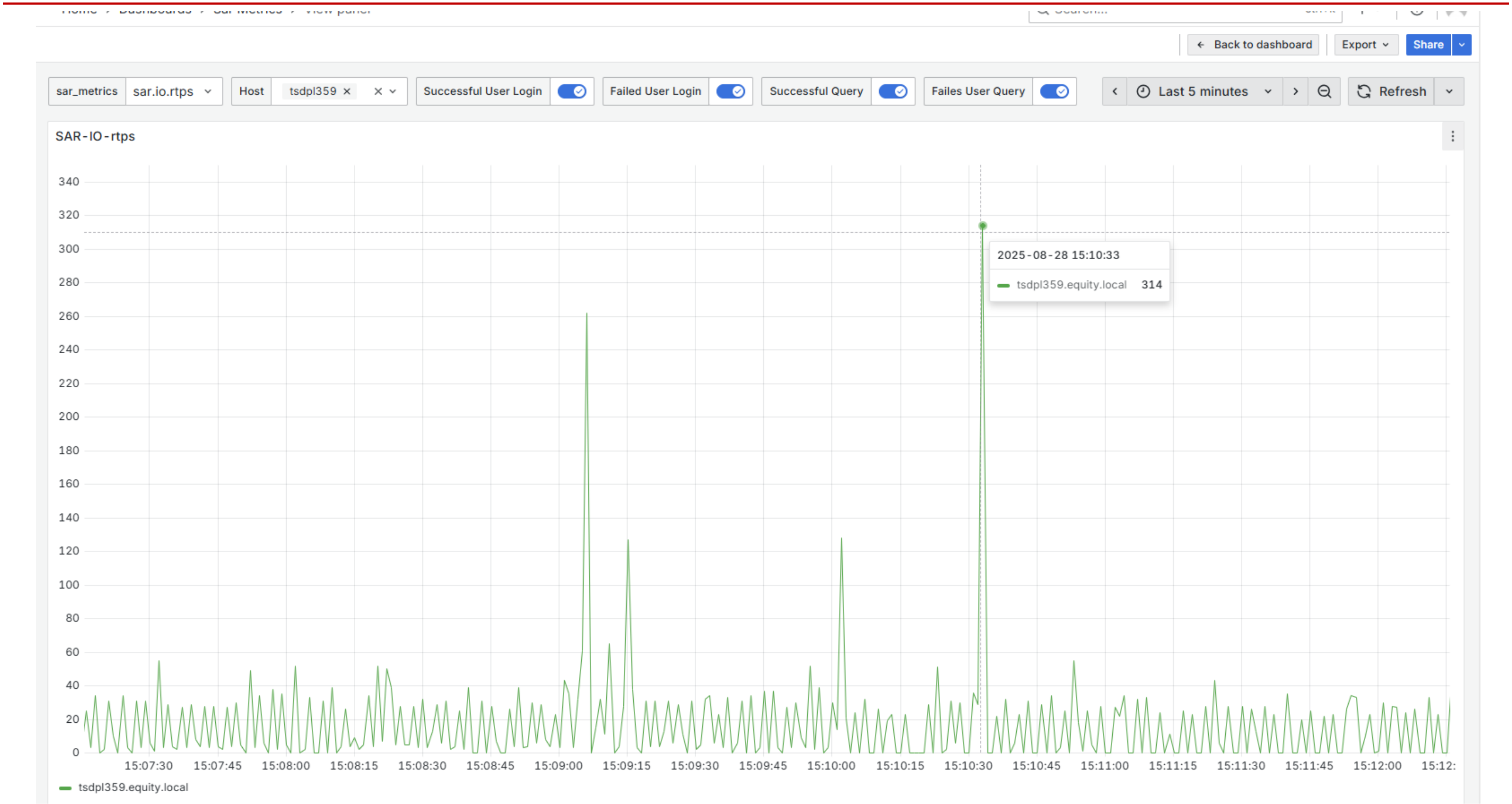
Hostname: All

Last 5 years

Daily HDB Size (Uncompressed)

- Keep track of market data and growth overtime
- Which market data is taking up the most space
- Predict trends of how much storage will be needed

System Performance Dashboard



sar_metrics: sar.io.rtps

Host: tsdpl359 ×

Successful User Login ☑

Failed User Login ☑

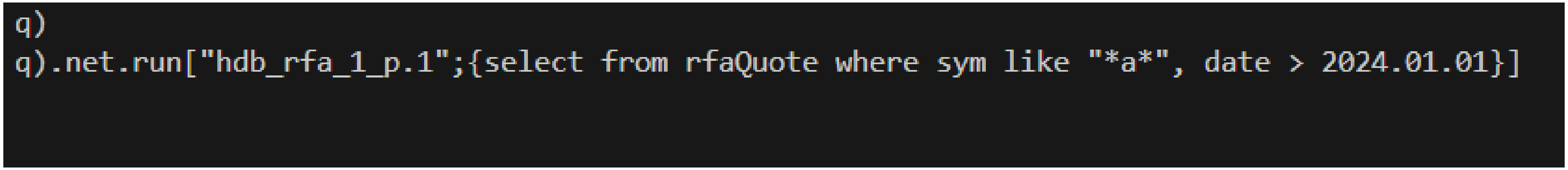
Successful Query ☑

Failes User Query ☑

Last 5 minutes

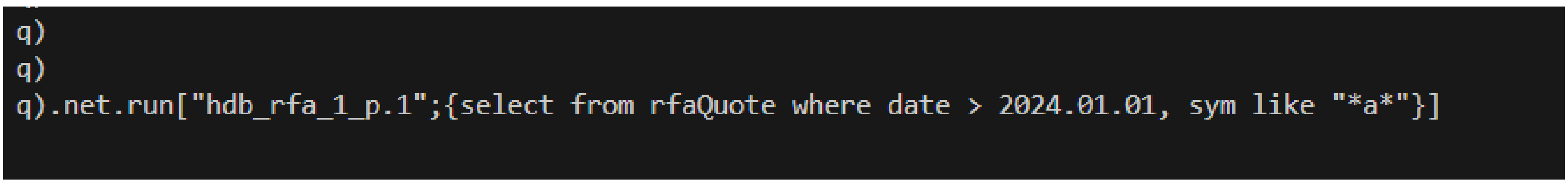
SAR-IO-rtps

This is a Slow Query



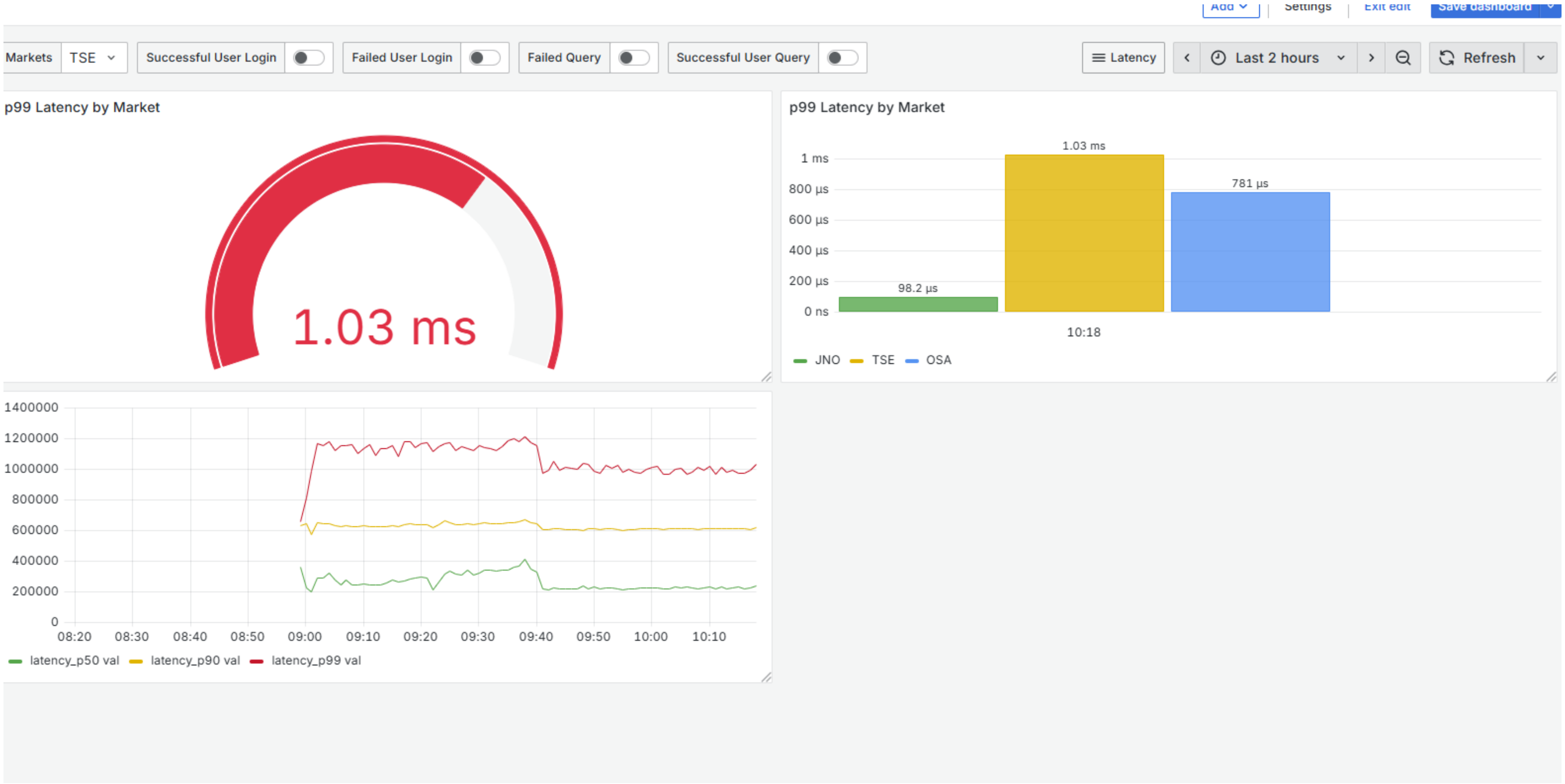
```
q)
q).net.run["hdb_rfa_1_p.1";{select from rfaQuote where sym like "*a*", date > 2024.01.01}]
```

This is much faster:



```
q)
q)
q).net.run["hdb_rfa_1_p.1";{select from rfaQuote where date > 2024.01.01, sym like "*a*"}]
```

Latency Dashboards



Reflections

What I achieved:

- Built an observability stack on KDB+ architecture
- Metrics collector in q
- Interactive dashboard for real-time visualization

What I learned:

- Debugging through linux processes
- kdb+, q
    - Learned q, database maintenances (backfilling databases)
- Observability concepts
- debugging skills!!!
    - Very different experience from TypeScript

Thank You for Listening!

いっしょに、明日のこと。
Share the Future

SMBC日興証券

Questions?