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

Robert Hodges - Altinity

FOSDEM 2024
Let’s make some introductions

**Robert Hodges**
Database geek for 40 years.
Open source since 2006.
ClickHouse since 2019.

**Altinity Engineering**
Other database geeks with centuries of experience in DBMS and applications

Authors of [Altinity Kubernetes Operator for ClickHouse](#), [Altinity Grafana plugin for ClickHouse](#), [clickhouse-backup project](#), etc.
Monitoring is for answering questions

- Why are users seeing performance problems?
- When did it start?
- How many users are affected?
- Which service is at fault?
How we answered those questions in days gone by…

```bash
$ vmstat -n 2 10
```

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
<th>swpd</th>
<th>free</th>
<th>buff</th>
<th>cache</th>
<th>si</th>
<th>so</th>
<th>bi</th>
<th>bo</th>
<th>in</th>
<th>cs</th>
<th>us</th>
<th>sy</th>
<th>id</th>
<th>wa</th>
<th>st</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690808</td>
<td>2290104</td>
<td>6897160</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>187</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>94</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690800</td>
<td>2290104</td>
<td>6897160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>2989</td>
<td>7688</td>
<td>2</td>
<td>1</td>
<td>97</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690140</td>
<td>2290104</td>
<td>6897164</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>72</td>
<td>4704</td>
<td>13677</td>
<td>3</td>
<td>2</td>
<td>95</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21689888</td>
<td>2290104</td>
<td>6897164</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>3132</td>
<td>9364</td>
<td>2</td>
<td>1</td>
<td>97</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690220</td>
<td>2290104</td>
<td>6897168</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>86</td>
<td>3014</td>
<td>7995</td>
<td>1</td>
<td>1</td>
<td>97</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690448</td>
<td>2290104</td>
<td>6897176</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>2660</td>
<td>7297</td>
<td>1</td>
<td>1</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21690268</td>
<td>2290104</td>
<td>6897176</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>2695</td>
<td>7222</td>
<td>1</td>
<td>1</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>343296</td>
<td>21690196</td>
<td>2290104</td>
<td>6897180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>3641</td>
<td>10419</td>
<td>2</td>
<td>1</td>
<td>97</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21689696</td>
<td>2290104</td>
<td>6897180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>4108</td>
<td>12605</td>
<td>3</td>
<td>2</td>
<td>95</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>343296</td>
<td>21689900</td>
<td>2290104</td>
<td>6897184</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>2688</td>
<td>7270</td>
<td>2</td>
<td>1</td>
<td>97</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
The modern way is a lot nicer
Let’s build a monitoring system with open source

- Analytic database
  - Data ingest
  - Data storage and query processing
  - Query results
  - Visualization
  - Alerting

Metrics
Logs
Traces
Introducing ClickHouse, a real-time analytic database

Understands SQL

Runs on bare metal to cloud

Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

Is Open source (Apache 2.0)
ClickHouse optimizes for fast response on large datasets

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

Highly compressed column storage with indexing

Parallelized/vectorized query

Automatic replication between nodes

Table replica
Grafana pairs well with ClickHouse for observability apps

Understands time series data
Simple installation
Many data sources
Lots of display plugins
Interactive zoom-in/zoom-out
Great for monitoring dashboards
Is open source (AGPL 3.0)
Soo... How do we ingest vmstat data and display it?

```bash
$ 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
```

![ClickHouse Database](image)

![Grafana](image)
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

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Host</th>
<th>r</th>
<th>b</th>
<th>swpd</th>
<th>free</th>
<th>buff</th>
<th>cache</th>
<th>si</th>
<th>so</th>
<th>bi</th>
<th>bo</th>
<th>in</th>
<th>cs</th>
<th>us</th>
<th>sy</th>
<th>id</th>
<th>wa</th>
<th>st</th>
</tr>
</thead>
<tbody>
<tr>
<td>2024-01-22 18:13:16</td>
<td>logos3</td>
<td>0</td>
<td>0</td>
<td>166912</td>
<td>2523688</td>
<td>41412</td>
<td>3408292</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>101</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2024-01-22 18:13:17</td>
<td>logos3</td>
<td>0</td>
<td>0</td>
<td>166912</td>
<td>2523696</td>
<td>41412</td>
<td>3408316</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>216</td>
<td>1214</td>
<td>4320</td>
<td>1</td>
<td>1</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2024-01-22 18:13:18</td>
<td>logos3</td>
<td>0</td>
<td>0</td>
<td>166912</td>
<td>2527120</td>
<td>41412</td>
<td>3408572</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1172</td>
<td>4162</td>
<td>2</td>
<td>1</td>
<td>98</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

Altinity plugin for ClickHouse
ClickHouse data source for Grafana
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
```

<table>
<thead>
<tr>
<th>host</th>
<th>loaded_minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>logos3</td>
<td>6</td>
</tr>
<tr>
<td>logos2</td>
<td>5</td>
</tr>
</tbody>
</table>

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
Question 1: Do I really have to write code for everything?
And there’s more…
Using Fluentbit and OTEL to collect monitoring data
Question 2: Why not use PostgreSQL?

**PostgreSQL, MySQL**

- Read all columns in row
- Rows minimally or not compressed

**ClickHouse**

- Read only selected columns
- Columns highly compressed
ClickHouse is often 1000x faster on analytic queries

- 59 GB (100%)
  - Read 109 columns
  - Read 3 columns from 109
  - Read 3 compressed columns over 8 threads

- 1.7 GB (3%)
  - Read 3 compressed columns

- 21 MB (.035%)
  - 2.6 MB (.0044%)
Question 3: How to handle data from many collectors?

Kafka Topic
Contains messages

Collector
Kafka Table
Consumes messages

Materialized View
Transfers rows

MergeTree Table
Stores rows

https://kb.altinity.com/altinity-kb-integrations/altinity-kb-kafka/
Question 4: How else can we map JSON to tables?

```json
{"@timestamp": "1998-05-22 21:37:39", "clientip": "54.72.5.0", ...}
```
Representing JSON as paired arrays and maps

Map: Header values with mapped key value pairs

SQL Table

Map with Key/Values

{"@timestamp":"1998-05-22 21:37:39","clientip":"54.72.5.0",...}

Array of Keys

Array of Values

Arrays: Header values with key-value pairs

SQL Table

Altinity
Where can I find out more?

Sample code: [https://github.com/Altinity/clickhouse-sql-examples](https://github.com/Altinity/clickhouse-sql-examples)

ClickHouse official docs – [https://clickhouse.com/docs/](https://clickhouse.com/docs/)


Fluentbit docs – [https://docs.fluentbit.io/manual/](https://docs.fluentbit.io/manual/)

Open Telemetry docs - [https://opentelemetry.io/docs/](https://opentelemetry.io/docs/)

Altinity Blog and YouTube Channel - [https://altinity.com](https://altinity.com)
Thanks

May the source be with you!

Robert Hodges - Altinity
https://altinity.com

Email: rhodges at altinity dot com
Slack: @Robert Hodges (CNCF, ClickHouse, AltinityDB)
LinkedIn: https://www.linkedin.com/in/berkeleybob2105/