Tracing a Cloud-Native Database

ClickHouse® & OpenTelemetry

Josh Lee • Altinity • Community Over Code • Sept 14 2025

Josh Lee

Open Source Advocate

@ Altinity

Agenda

Brief Intro to OpenTelemetry & Tracing

ClickHouse for Observability

ClickHouse Architecture

Observability for ClickHouse

Demo!

A humble log...

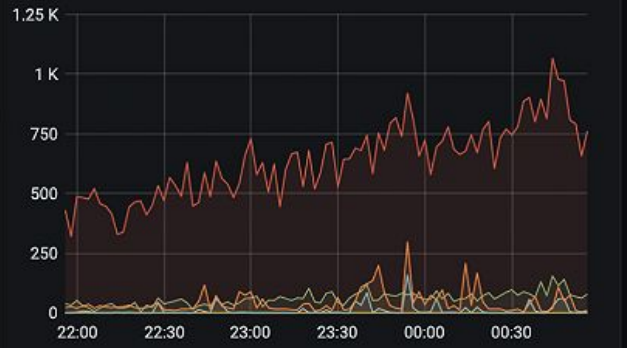
```
2024-07-01 09:35:34 231ms GET /home 200
```

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

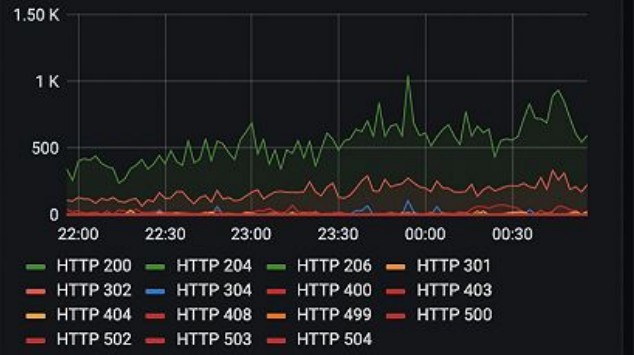
Overview

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

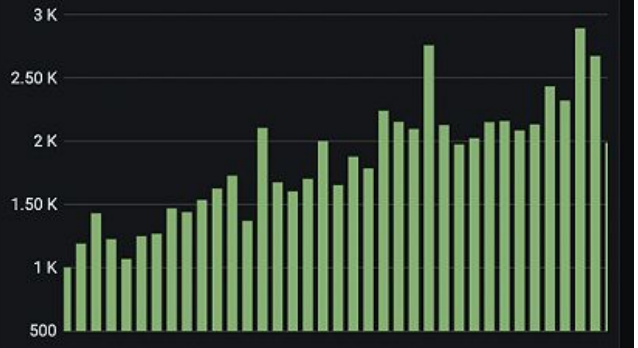
HTTP Requests / Ingress



HTTP Status Codes

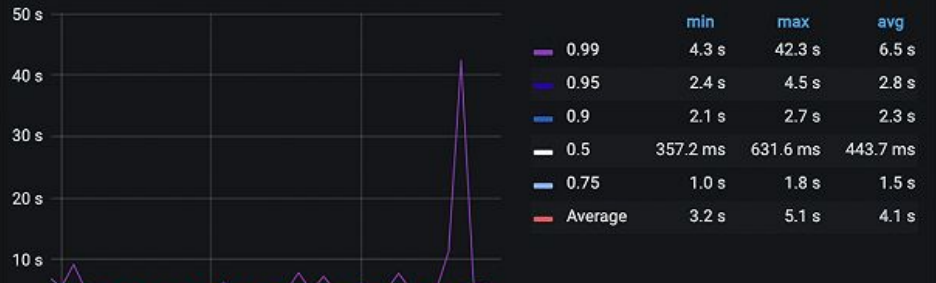


Total HTTP Requests



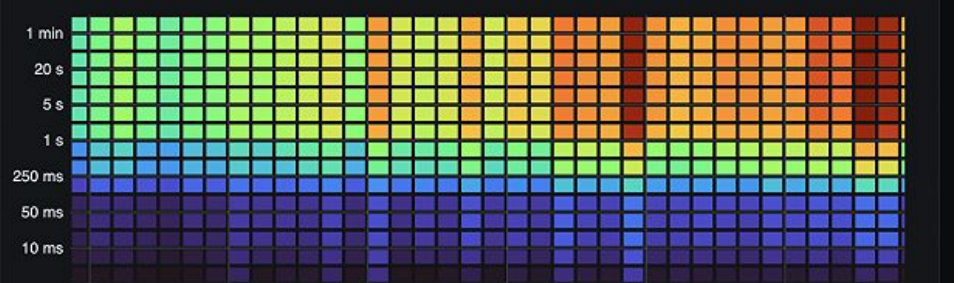
Latency

Latency (Average Percentiles)

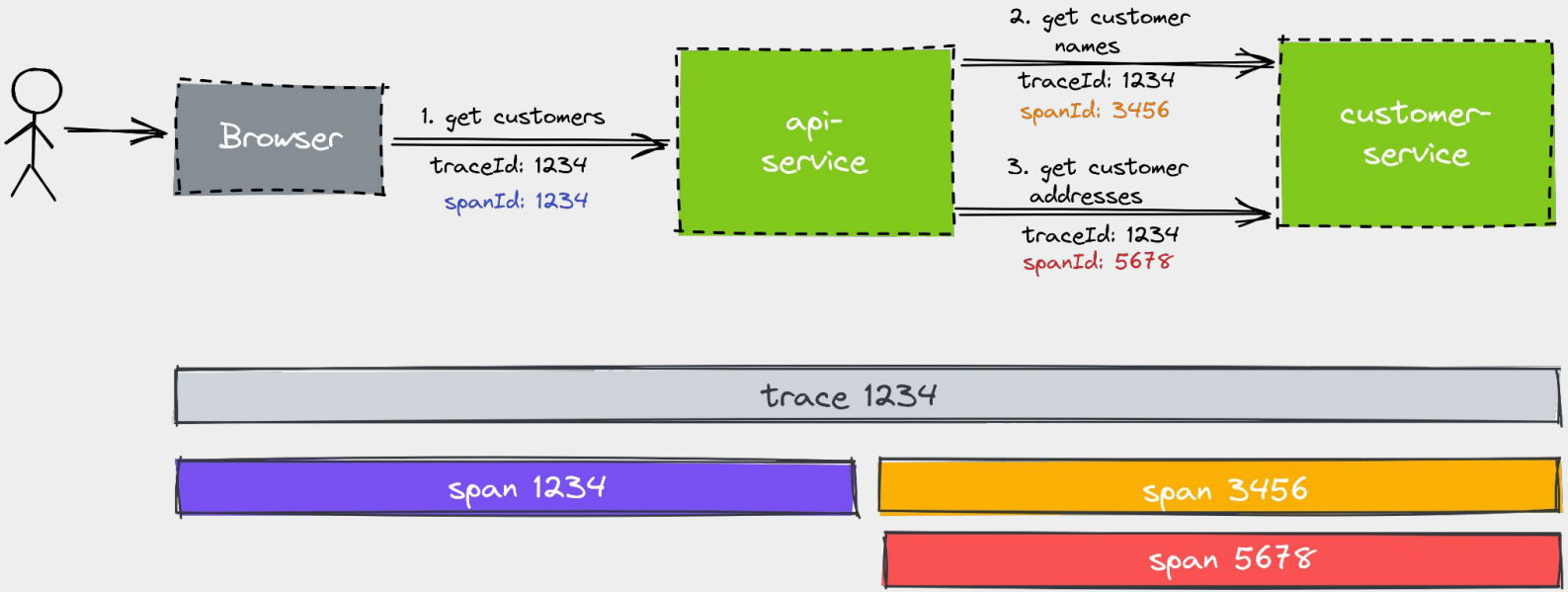


	min	max	avg
0.99	4.3 s	42.3 s	6.5 s
0.95	2.4 s	4.5 s	2.8 s
0.9	2.1 s	2.7 s	2.3 s
0.5	357.2 ms	631.6 ms	443.7 ms
0.75	1.0 s	1.8 s	1.5 s
Average	3.2 s	5.1 s	4.1 s

Latency Heatmap



What is Distributed Tracing?



Distributed Tracing is the *“Killer App”*

Understand complete request flows

Create a real-time map of system topology and dependencies

Derive metrics from the richness of trace metadata

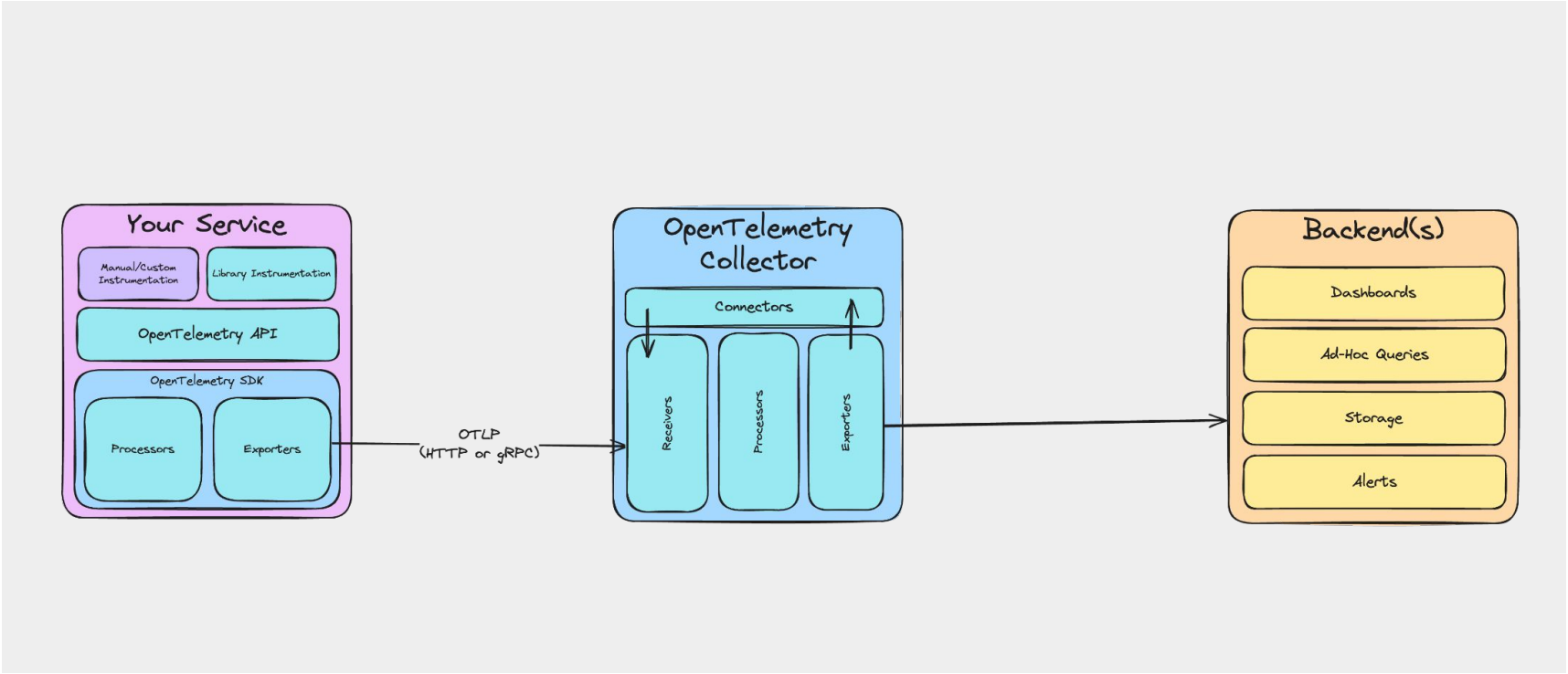
Enrich logs and metrics with context

Introducing OpenTelemetry

What is OTel?

- Specifications:
  - OTLP
  - Semantic Conventions
  - W3C Trace Context
- Libraries & Tools:
  - Language-specific SDKs & APIs
  - Instrumentation Libraries
  - The OpenTelemetry Collector
  - Extensions
- Community
  - Vendor-neutral, Open Source
  - 2nd most active CNCF project

What is OpenTelemetry?



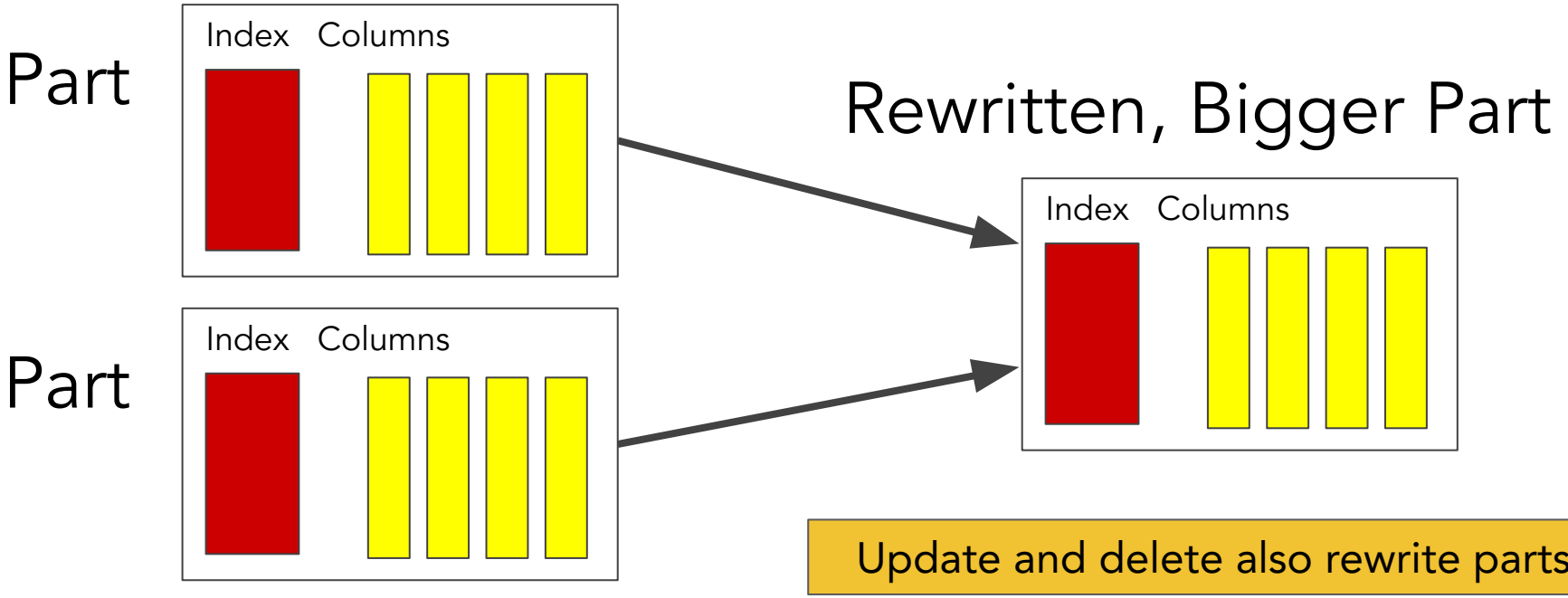
Introducing ClickHouse

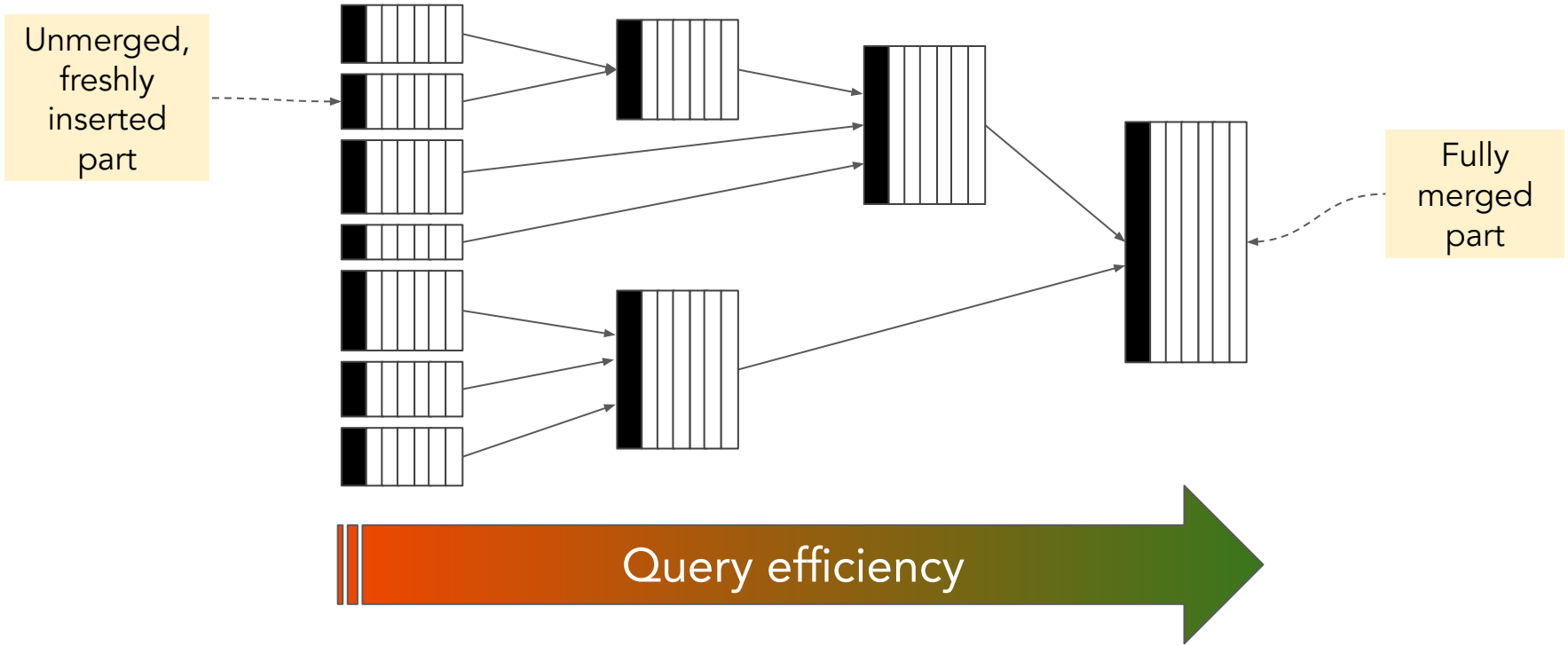
What is ClickHouse?

- Apache 2.0
- Single binary
- SQL Compatible
- Columnar
- *Really really* fast
- Laptop to exabyte scale
- Eats cardinality for breakfast

Why ClickHouse?

Telemetry is **WORM:** Write-Once Read-Many


Part
Index
Columns
Part
Index
Columns
Rewritten, Bigger Part
Index
Columns
Update and delete also rewrite parts


Unmerged, freshly inserted part
Fully merged part
Query efficiency

More Benefits

- Time-friendly
- Easy cleanup
- TTL & tiered storage
- Extreme cardinality
- Excellent compression
- Flexible schema

Observability with ClickHouse

Storing and querying distributed traces from applications

Observability Integrations

OpenTelemetry Collector Exporter

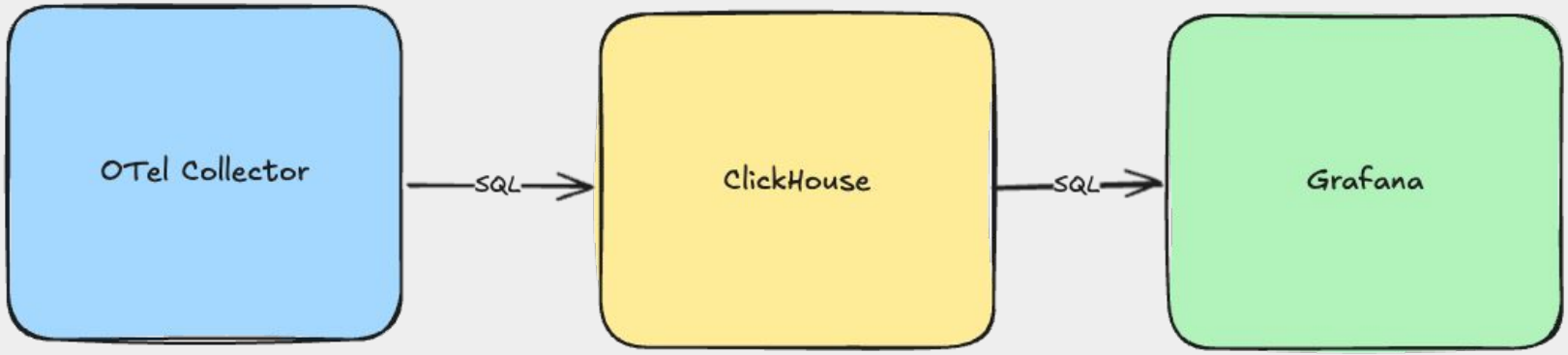
Grafana Datasource

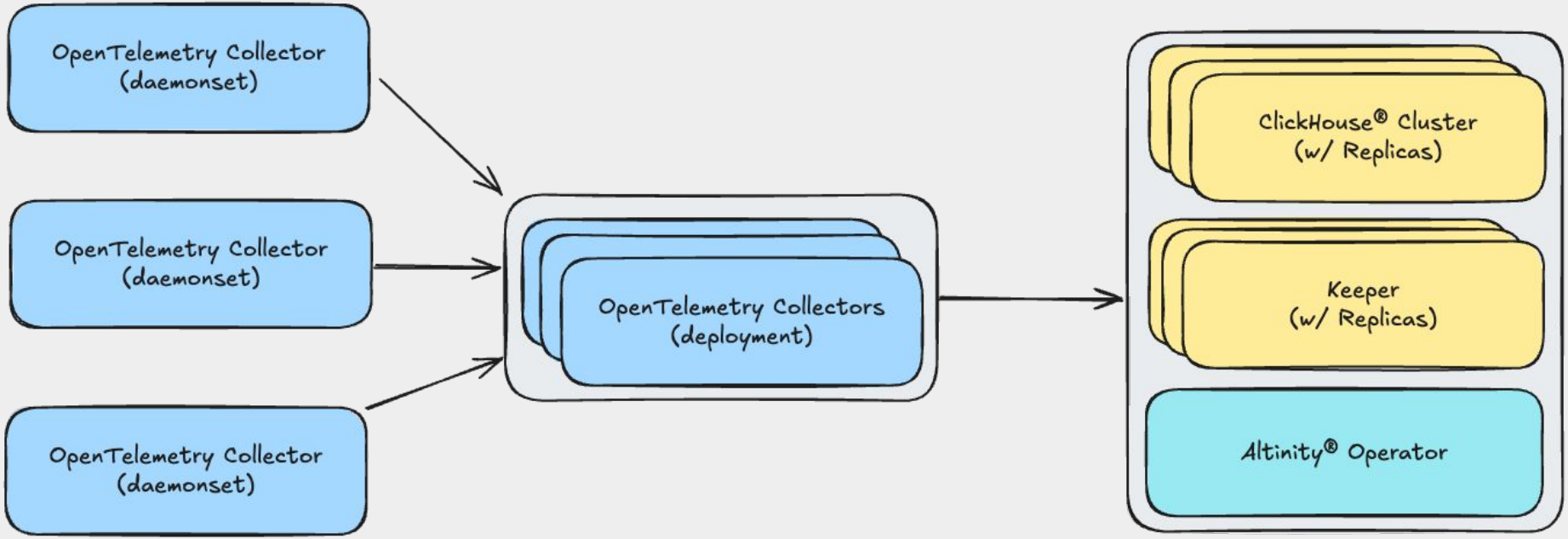
Kafka Connector

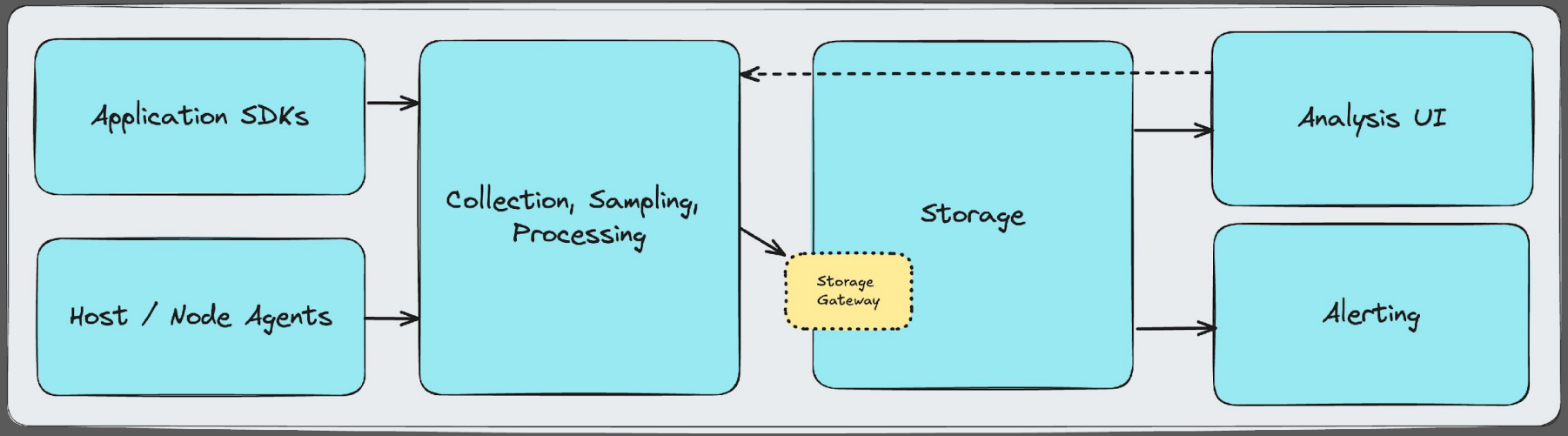
Coroot, SigNoz, ClickStack

Quesma

Iceberg + Parquet...


OTel Collector
SQL
ClickHouse
SQL
Grafana


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


Application SDKs
Host / Node Agents
Collection, Sampling, Processing
Storage Gateway
Storage
Analysis UI
Alerting

coroot :~#

default

search for apps and nodes

last hour

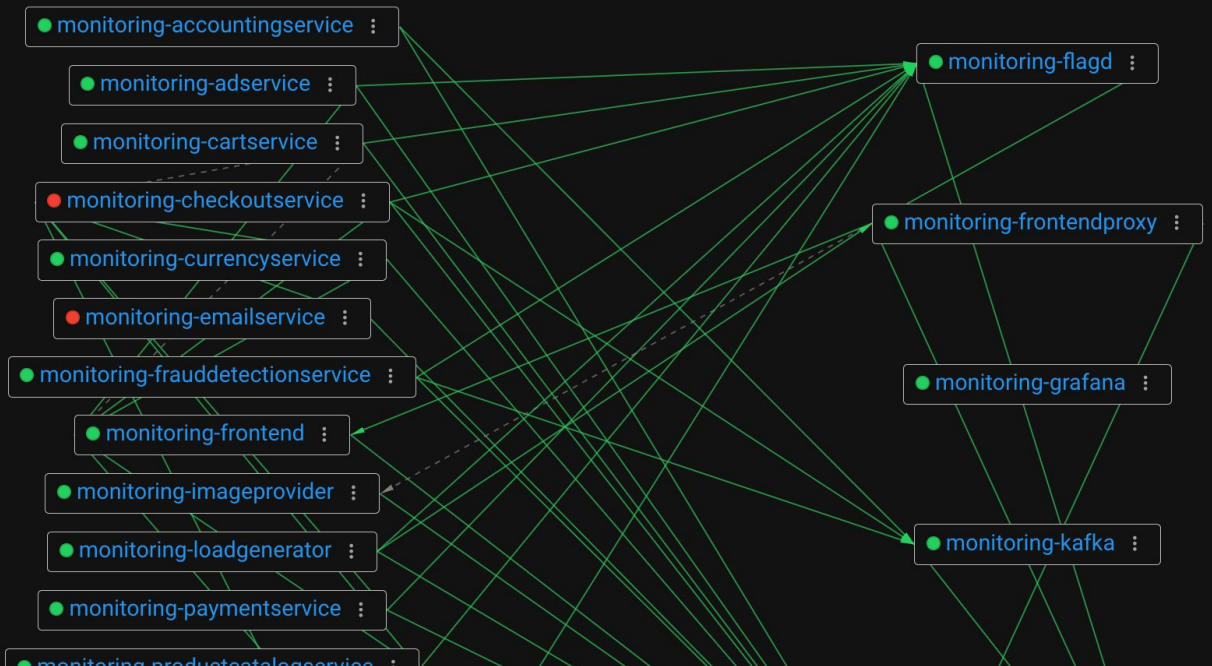
Overview

HEALTH | SERVICE MAP | TRACES | NODES | DEPLOYMENTS | COSTS

search

namespaces: monitoring

application | control-plane | monitoring | +




SigNoz


coroot :~#


ClickStack®

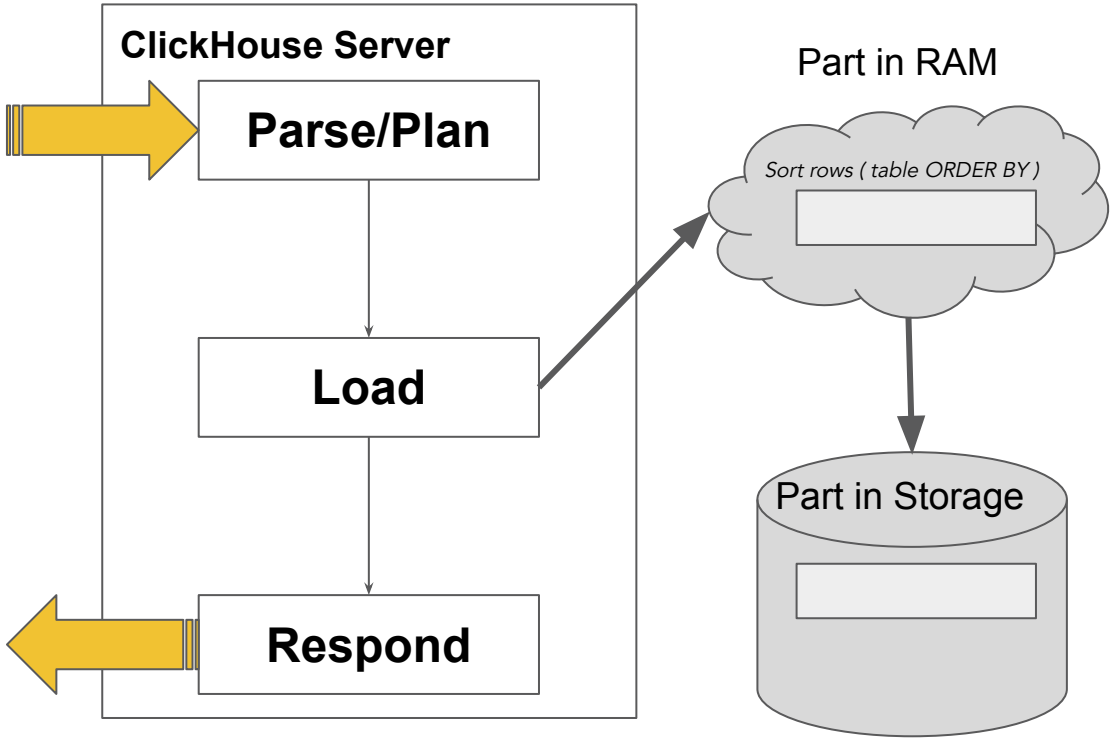
ClickHouse Architecture

How queries are processed

ClickHouse Architecture: Processing an Insert

```
INSERT INTO sdata
VALUES
(15, 'TEMP', . . .),
(15, 'TEMP', . . .)
```

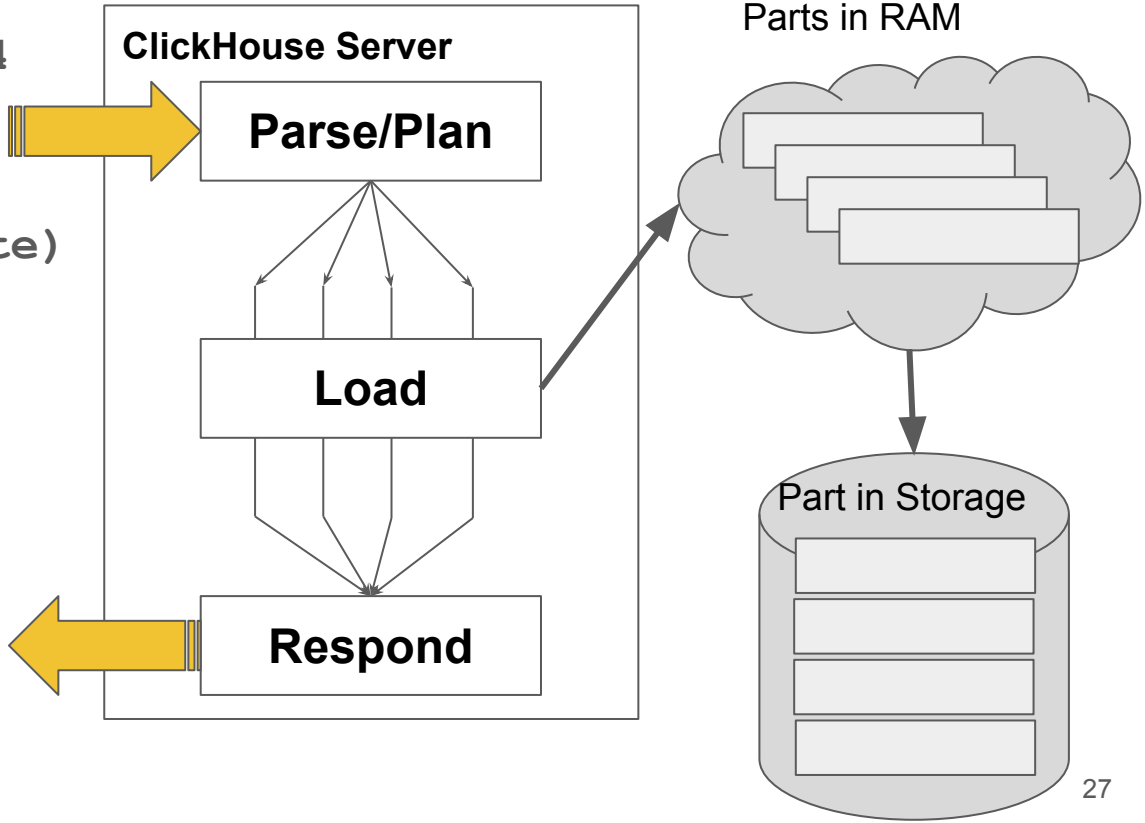
```
2 rows in set. Elapsed: 0.003 sec.
```



How can we make this more efficient? Parallelize!

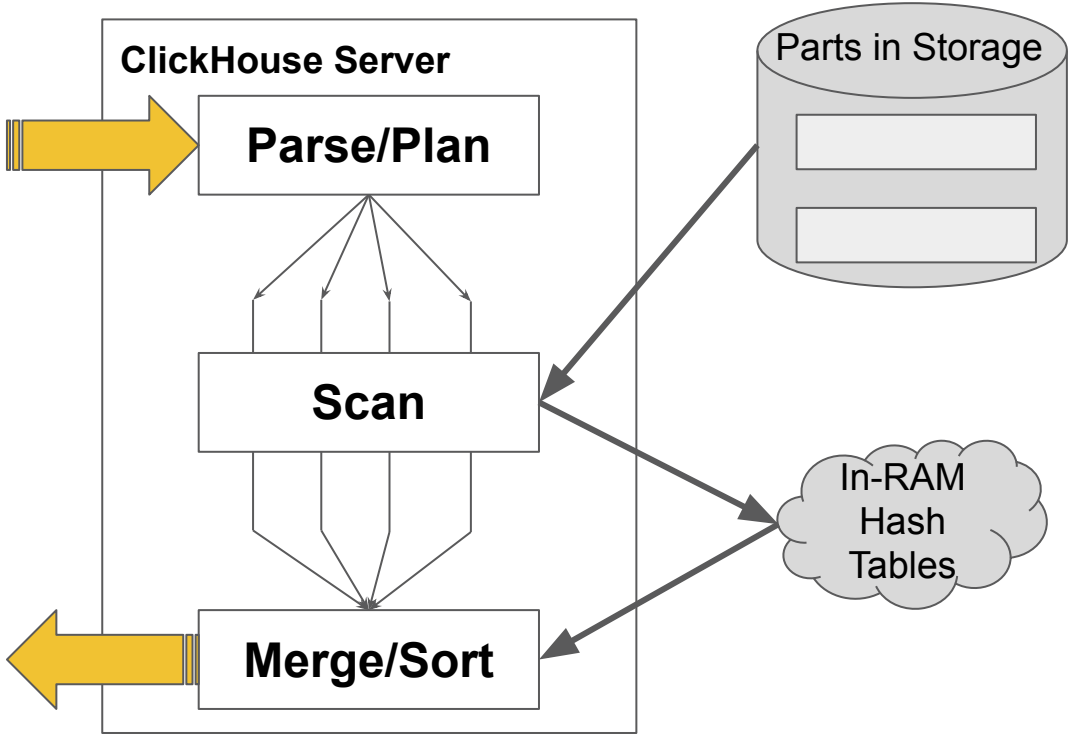
```
set max_insert_threads=4
insert into ontime_test
select * from ontime
  where toYear(FlightDate)
  between 2000 and 2001
```

```
2 rows in set. Elapsed: 0.003 sec.
```



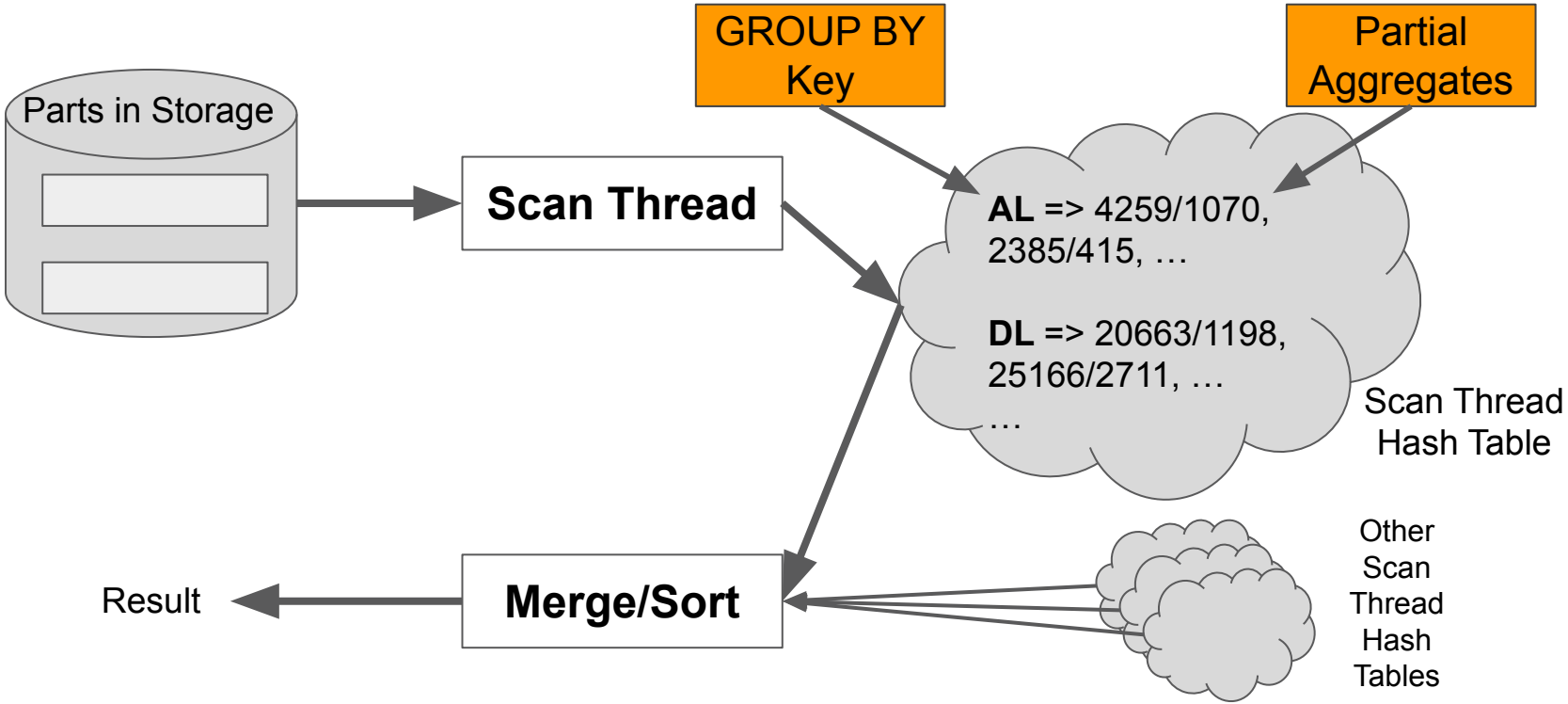
How does ClickHouse process a query with aggregates?

```
SELECT Carrier,
  avg(DepDelay)AS Delay
FROM ontime
GROUP BY Carrier
ORDER BY Delay DESC
```




```
┌─Carrier─┬──────────────Delay─┐
│ B6      │ 12.058290698785067 │
│ EV      │ 12.035012037703922 │
│ NK      │ 10.437692933474269 │
. . .
```

How does a ClickHouse thread do aggregation?



Observability for ClickHouse

Built-in metrics, logs, and ***traces***

System Tables

- Metric Log
- Part Log
- Query Log
- Query Views Log
- Trace Log
- Distributed Traces*

```
SHOW CREATE TABLE system.opentelemetry_span_log;
---
CREATE TABLE system.opentelemetry_span_log
(
    `hostname` LowCardinality(
String) COMMENT 'The hostname where this span was captured.',
    `trace_id` UUID COMMENT
'ID of the trace for executed query.',
    `span_id` UInt64 COMMENT
'ID of the trace span.',
    `parent_span_id` UInt64 COMMENT
'ID of the parent trace span.',
    `operation_name` LowCardinality(
String) COMMENT 'The name of the operation.',
    `kind` Enum8(
'INTERNAL' = 0, 'SERVER' = 1, 'CLIENT' = 2, 'PRODUCER' = 3, 'CONSUMER' = 4) COMMENT 'The SpanKind of the span. INTERNAL — Indicates that the
span represents an internal operation within an application. SERVER — Indicates that the span covers server-side handling of a synchronous RPC
or other remote request. CLIENT — Indicates that the span describes a request to some remote service. PRODUCER — Indicates that the span
describes the initiators of an asynchronous request. This parent span will often end before the corresponding child CONSUMER span, possibly
even before the child span starts. CONSUMER - Indicates that the span describes a child of an asynchronous PRODUCER request.',
    `start_time_us` UInt64 COMMENT
'The start time of the trace span (in microseconds).',
    `finish_time_us` UInt64 COMMENT
'The finish time of the trace span (in microseconds).',
    `finish_date`
Date COMMENT 'The finish date of the trace span.',
    `attribute` Map(LowCardinality(
String), String) COMMENT 'Attribute depending on the trace span. They are filled in according to the recommendations in the OpenTelemetry
standard.',
    `attribute.names` Array(LowCardinality(
String)) ALIAS mapKeys(attribute),
    `attribute.values` Array(
String) ALIAS mapValues(attribute)
)
ENGINE = MergeTree
PARTITION
BY toYYYYMM(finish_date)
ORDER BY (finish_date, finish_time_us, trace_id)
SETTINGS index_granularity =
8192
COMMENT
'Contains information about trace spans for executed queries.'


1 row in set. Elapsed: 0.034 sec.
```

Built-in Spans Table

System table created dynamically

Enable tracing via setting (on query or user profile) or request headers

Using a TraceID Header:

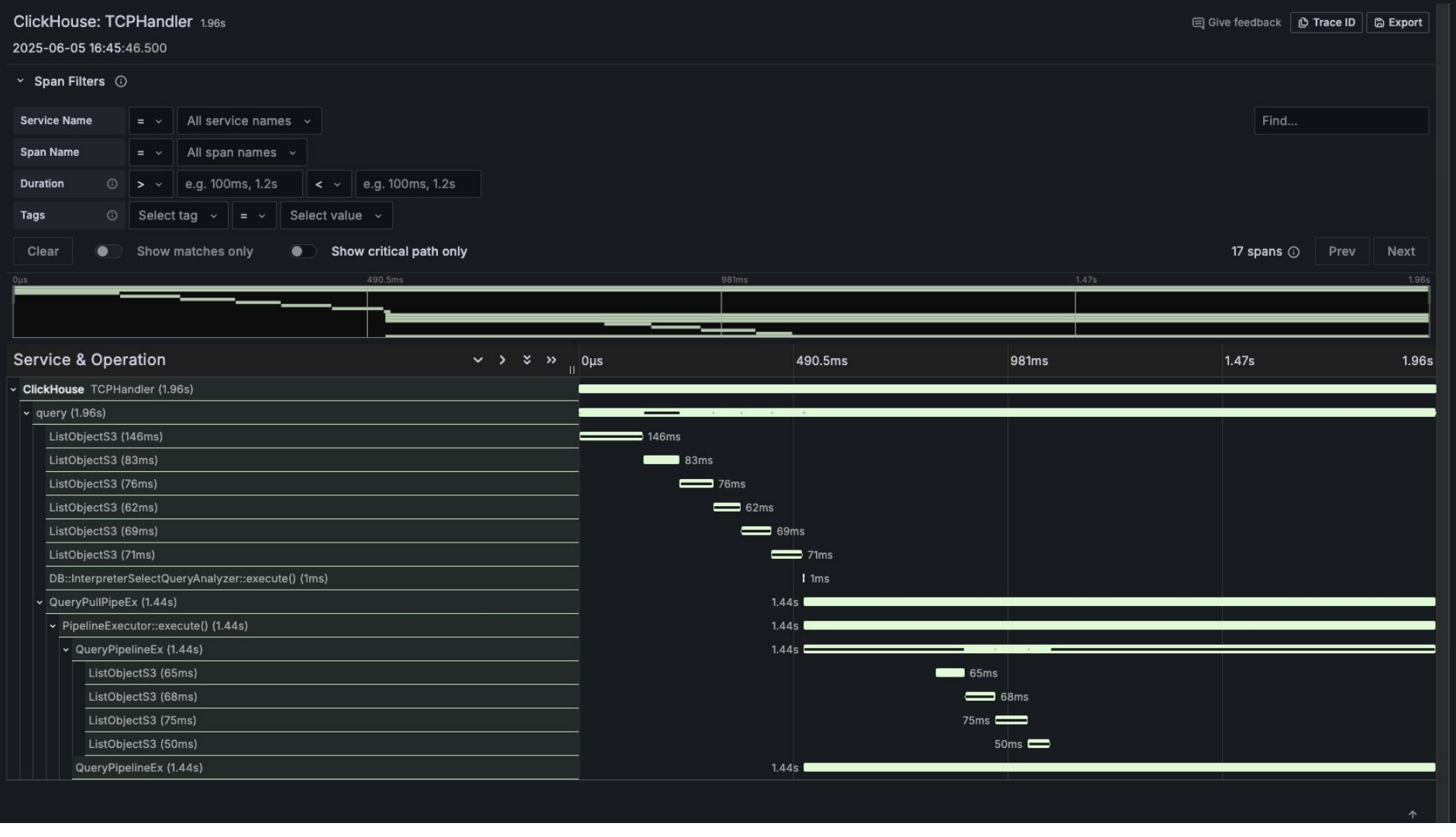
```
$ clickhouse-client \
   --opentelemetry-traceparent \
   "00-4bf92f...929d0e0e4736-00f067aa0ba902b7-01"
```

Always Initialize a Trace:

```
SET opentelemetry_start_trace_probability = 1;
```

Demo

Tracing ClickHouse Queries


ClickHouse: TCPHandler 1.96s
2025-06-05 16:45:46.500
Give feedback
Trace ID
Export
Span Filters
Service Name
All service names
Span Name
All span names
Duration
e.g. 100ms, 1.2s
e.g. 100ms, 1.2s
Tags
Select tag
Select value
Find...
Clear
Show matches only
Show critical path only
17 spans
Prev
Next
0µs
490.5ms
981ms
1.47s
1.96s
Service & Operation
ClickHouse TCPHandler (1.96s)
query (1.96s)
ListObjectS3 (146ms)
ListObjectS3 (83ms)
ListObjectS3 (76ms)
ListObjectS3 (62ms)
ListObjectS3 (69ms)
ListObjectS3 (71ms)
DB::InterpreterSelectQueryAnalyzer::execute() (1ms)
QueryPullPipeEx (1.44s)
PipelineExecutor::execute() (1.44s)
QueryPipelineEx (1.44s)
ListObjectS3 (65ms)
ListObjectS3 (68ms)
ListObjectS3 (75ms)
ListObjectS3 (50ms)
QueryPipelineEx (1.44s)
146ms
83ms
76ms
62ms
69ms
71ms
1ms
1.44s
1.44s
1.44s
65ms
68ms
75ms
50ms
1.44s

Client-Side Tracing

Supported ClickHouse Clients:

- Java
- Golang
- Python

Call to action!

Please help improve the OpenTelemetry Auto-Instrumentation Libraries

Final Thoughts

- Distributed tracing is awesome
- Built-in observability is awesome

Thank You!

🦋 @joshleecreates.bsky.social

🐘 @joshleecreates@hachyderm.io

🧳 linkedin.com/in/joshuamlee

💬 altinity.com/slack

Connect with me