data + ClickHouse

Singapore Meetup

July 11, 2024

Thank you to our host!

INFOCOMM
MEDIA
DEVELOPMENT
AUTHORITY

Tech Talks

- **The State of SQL-based Observability**
  Pradeep Chhetri, Site Reliability Engineer @ ClickHouse

- **ClickHouse: Powering Coinhall's Real-Time Blockchain Data Platform**
  Aaron Choo, Co-Founder & CTO @ Coinhall

- **Panel Q&A**

The State of SQL-based Observability

July 11, 2024

ClickHouse

Pradeep Chhetri

Site Reliability Engineer @ ClickHouse

in pradeep-chhetri-61a80935

𝕏 p_chhetri

chhetripradeep

- I love playing with computers, trying out new softwares and databases.
- In my free time, i enjoy watching chess and football.

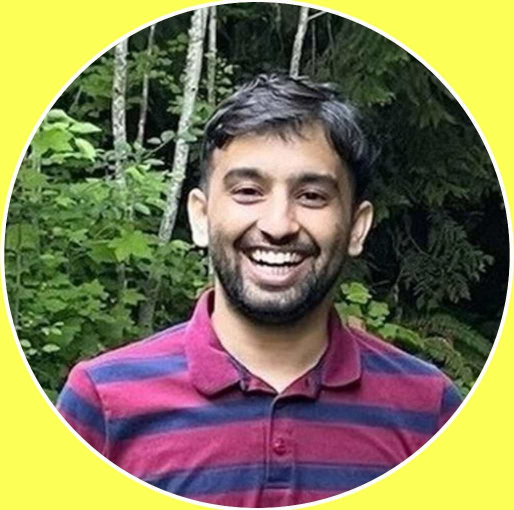


Table of contents

01 Introduction

02 Challenges

03 Deployments

04 Common Considerations

05 Demo

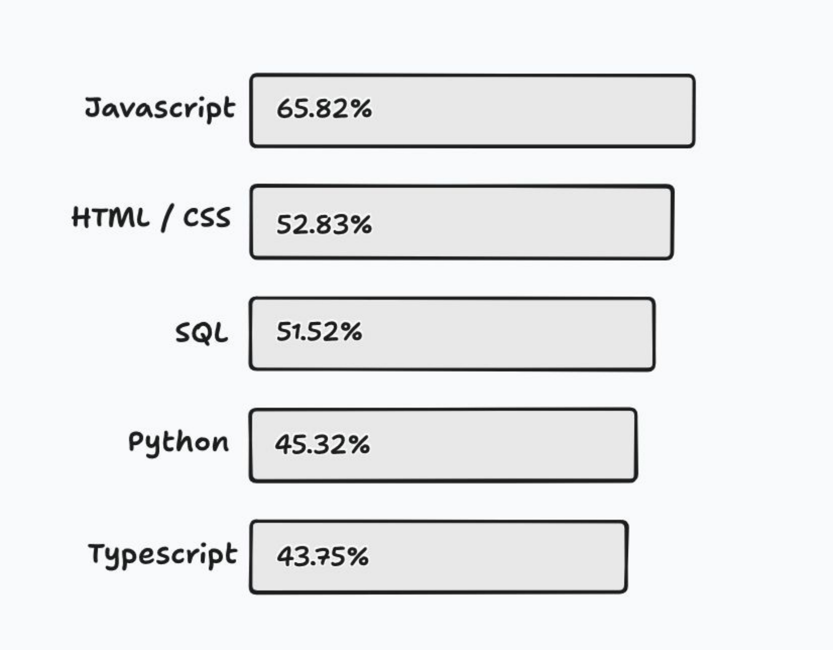
06 Questions

ClickHouse

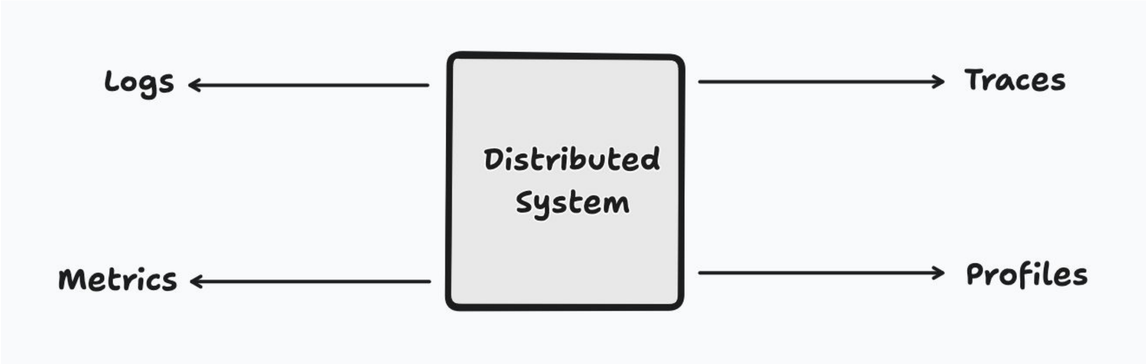
Introduction

SQL

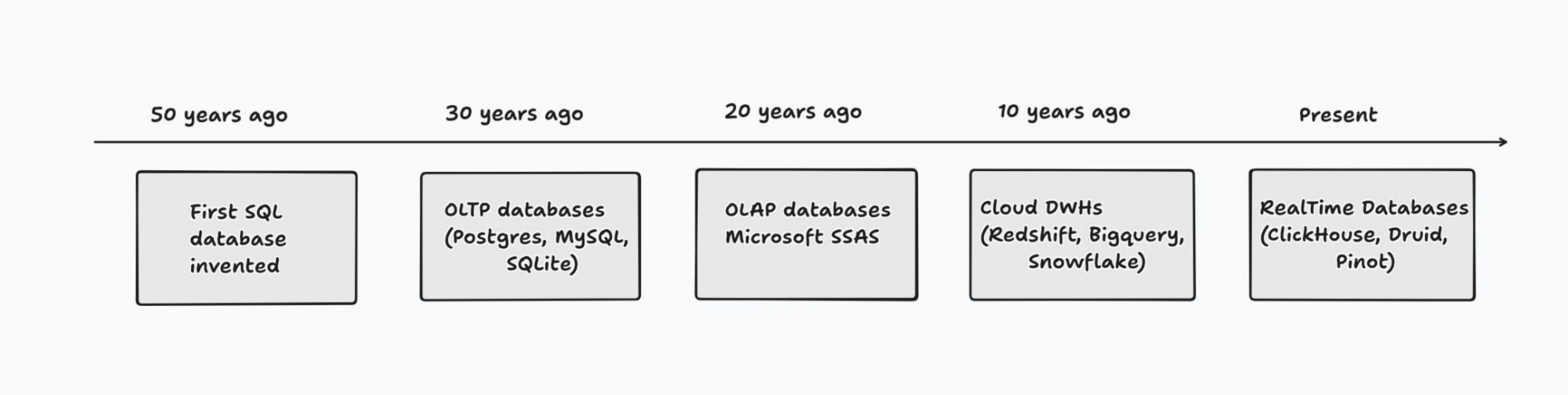
3rd most popular programming language



Observability



Evolution of SQL Databases



50 years ago: First SQL database invented

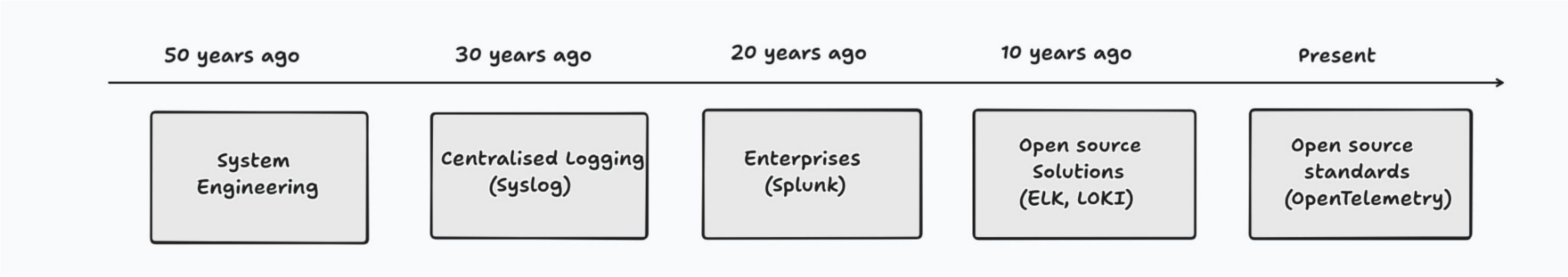
30 years ago: OLTP databases (Postgres, MySQL, SQLite)

20 years ago: OLAP databases Microsoft SSAS

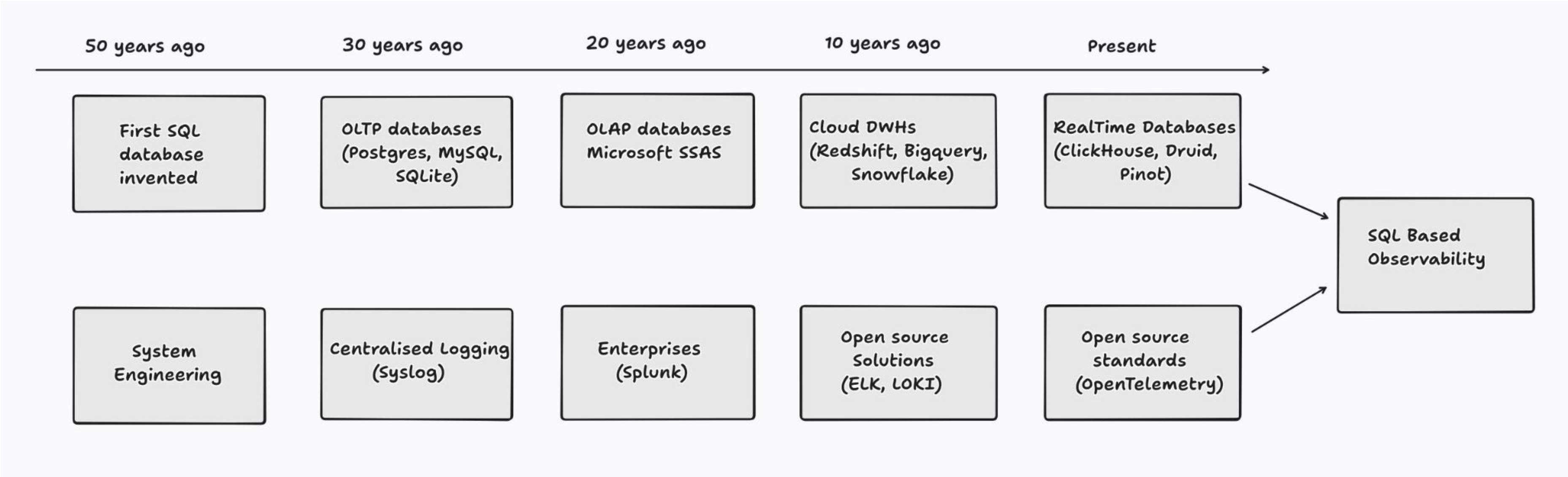
10 years ago: Cloud DWHs (Redshift, Bigquery, Snowflake)

Present: RealTime Databases (ClickHouse, Druid, Pinot)

Evolution of Observability


50 years ago
30 years ago
20 years ago
10 years ago
Present
System Engineering
Centralised Logging (Syslog)
Enterprises (Splunk)
Open source Solutions (ELK, LOKI)
Open source standards (OpenTelemetry)

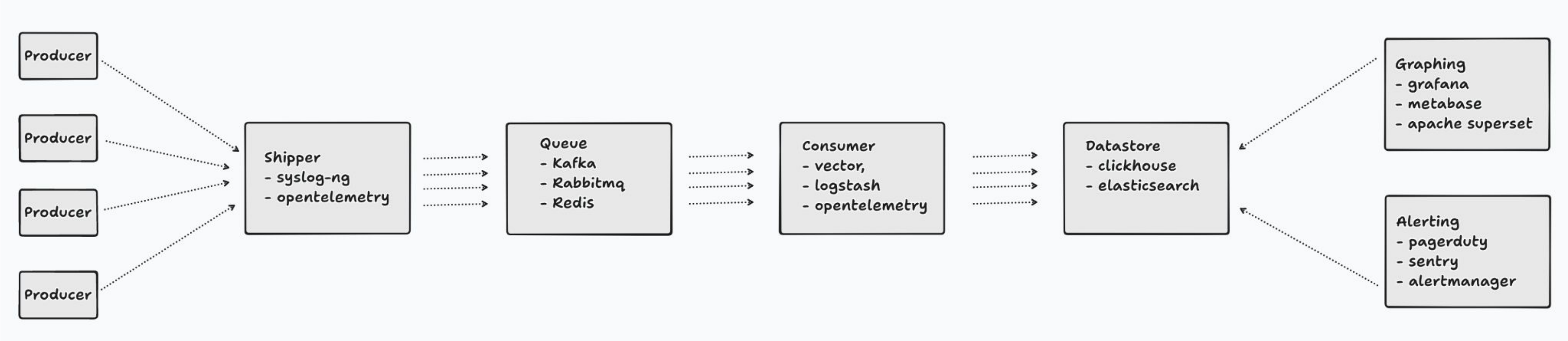
Overlap of SQL Databases & Observability


50 years ago
30 years ago
20 years ago
10 years ago
Present
First SQL database invented
OLTP databases (Postgres, MySQL, SQLite)
OLAP databases Microsoft SSAS
Cloud DWHs (Redshift, Bigquery, Snowflake)
RealTime Databases (ClickHouse, Druid, Pinot)
SQL Based Observability
System Engineering
Centralised Logging (Syslog)
Enterprises (Splunk)
Open source Solutions (ELK, LOKI)
Open source standards (OpenTelemetry)

ClickHouse

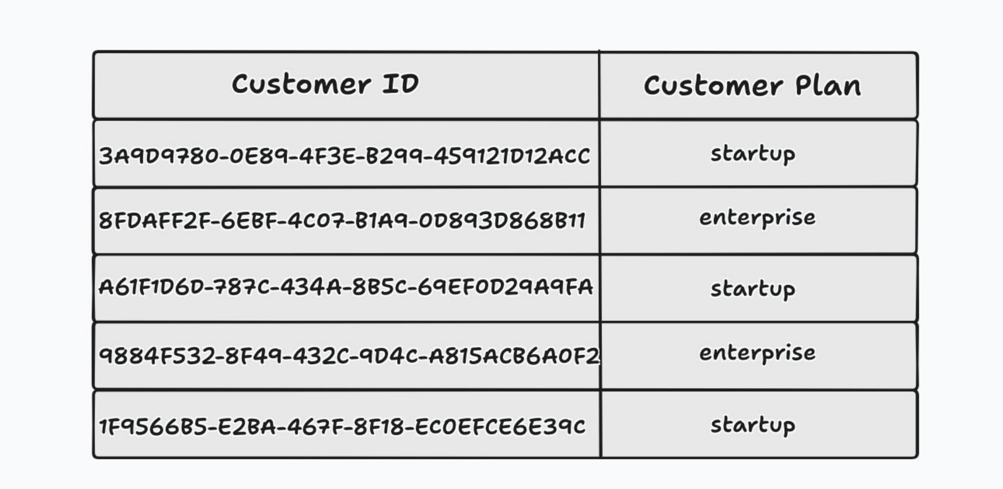
Challenges

Observability Pipeline



Challenges with Observability

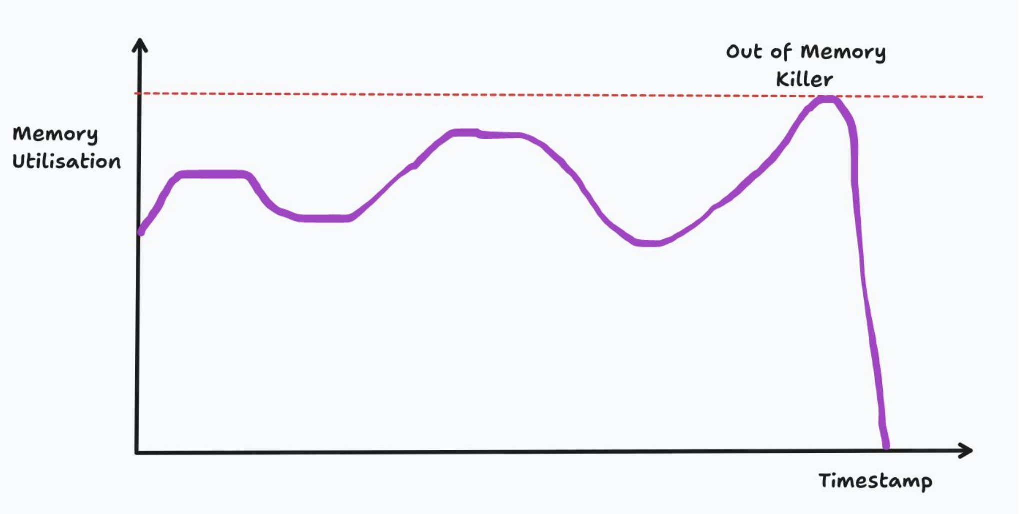
High Cardinality



Customer ID	Customer Plan
3A9D9780-0E89-4F3E-B299-459121D12ACC	startup
8FDAFF2F-6EBF-4C07-B1A9-0D893D868B11	enterprise
A61F1D6D-787C-434A-8B5C-69EF0D29A9FA	startup
9884F532-8F49-432C-9D4C-A815ACB6A0F2	enterprise
1F9566B5-E2BA-467F-8F18-EC0EFCE6E39C	startup

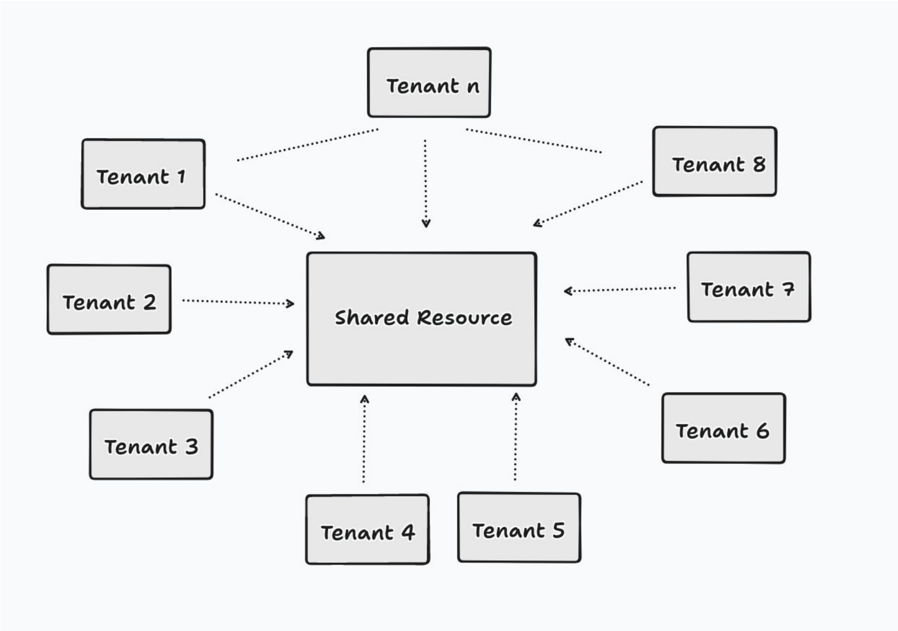
Challenges with Observability

Heavy Resource Utilization



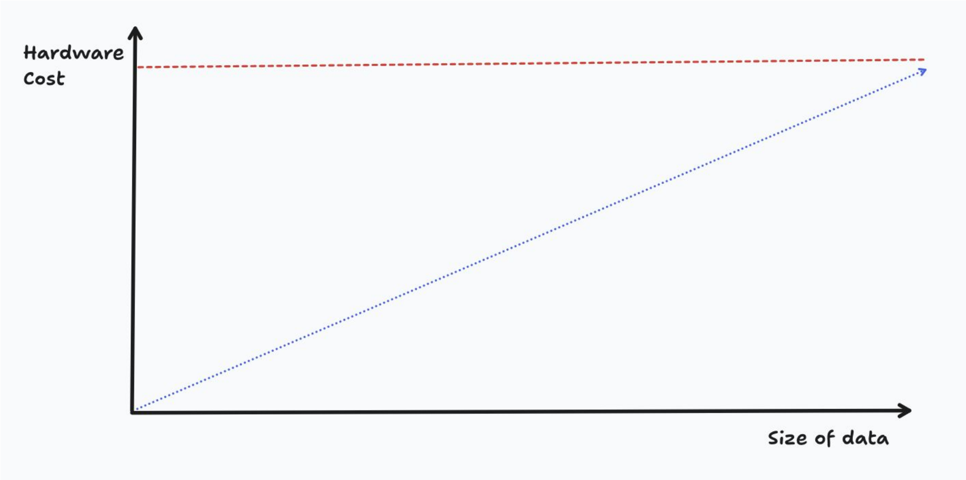
Challenges with Observability

Multi Tenancy Issues



Challenges with Observability

Resources Cost



Solutions for Observability Challenges

Infinite Cardinality Support

Optimized Resource Utilization

Compute & Storage Separation

Support for Quota, Priority, Resource Management

Efficient Data Compression

Scale easily from 1 byte to 1000 petabytes

“Observability is just another data problem.”

What is ClickHouse?

open-source

Developed since 2009
Open sourced in 2016
35k+ Github stars
1k+ contributors
300+ releases

column-oriented

Best for aggregations
Files per column
Sorting and indexing
Background merges

distributed

Replication
Sharding
Multi-master

OLAP database

Analytics use cases
Aggregations
Visualizations
Mostly immutable data

ClickHouse

Real-world Deployments

HTTP & DNS Analytics Platform

Architecture

Shippers → Kafka → ClickHouse

Wins

Efficient Ingestion & Compression

Improved Throughput & Latency

https://blog.cloudflare.com/http-analytics-for-6m-requests-per-second-using-clickhouse

CLOUDFLARE

Log Analytics Platform

Architecture

Log shippers → Kafka → ClickHouse → Kibana

QueryBridge to convert lucene to sql queries

Wins

Speed of ingestion, cost control

Tradeoffs

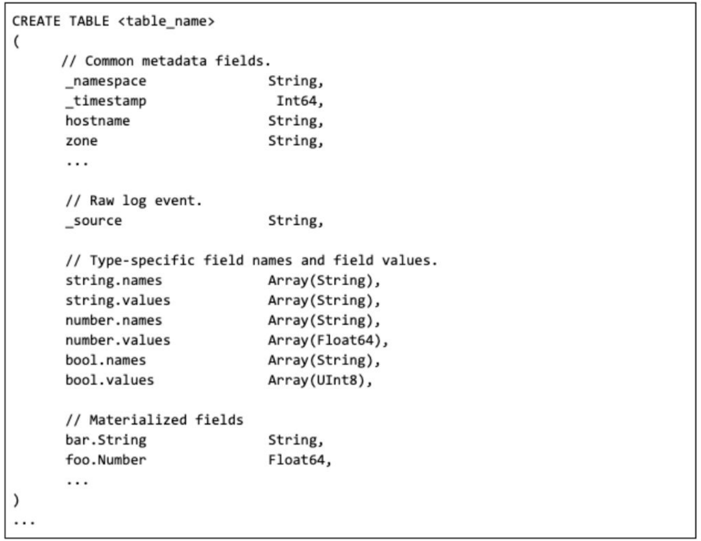
Stack administration, UI development

https://www.uber.com/blog/logging/

https://presentations.clickhouse.com/meetup40/uber.pdf

Uber

Log Analytics Platform



```
CREATE TABLE <table_name>
(
     // Common metadata fields.
      _namespace                String,
      _timestamp                 Int64,
      hostname                  String,
      zone                      String,
      ...

      // Raw log event.
      _source                   String,

      // Type-specific field names and field values.
      string.names              Array(String),
      string.values             Array(String),
      number.names              Array(String),
      number.values             Array(Float64),
      bool.names                Array(String),
      bool.values               Array(UInt8),

      // Materialized fields
      bar.String                String,
      foo.Number                Float64,
      ...
)
...
```

Adopting OLAP store for tracing

Architecture

OpenTelemetry → ClickHouse → Kibana

Wins

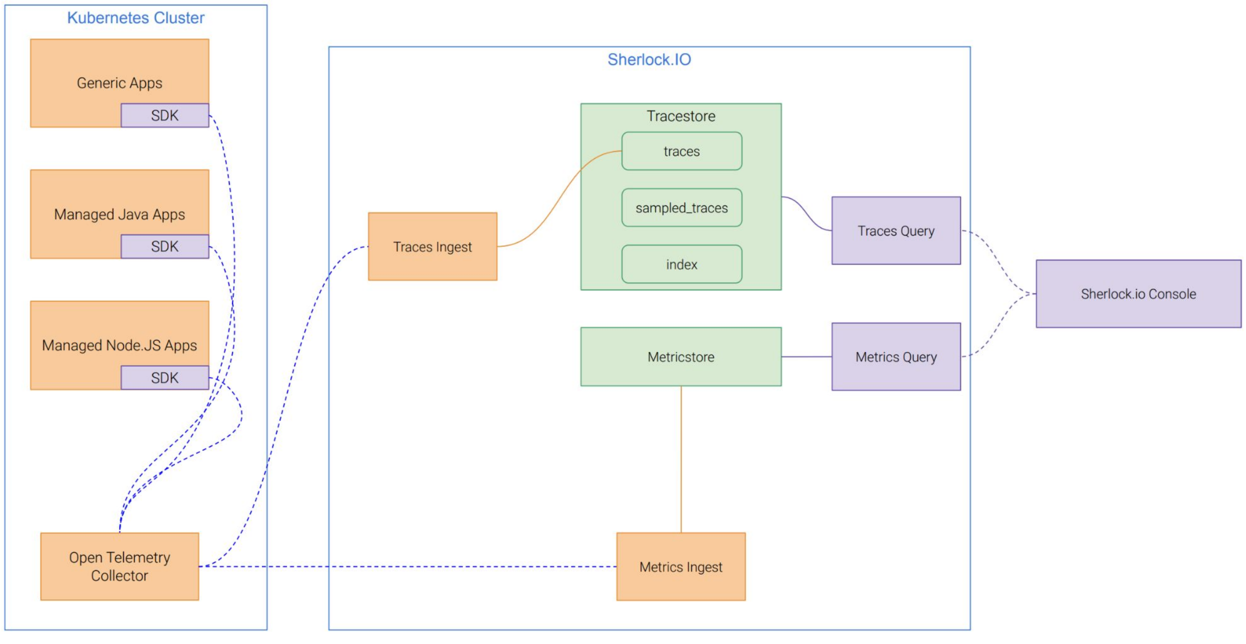
Data compression, open source licensing

Tradeoffs

Managing tiered OTel collectors

https://kccnceu2024.sched.com/event/1YeNu

Adopting OLAP store for tracing



7 © 2021 eBay. All rights reserved.

Dogfooding ClickHouse across o11y

Architecture

OpenTelemetry → ClickHouse → Grafana

Wins

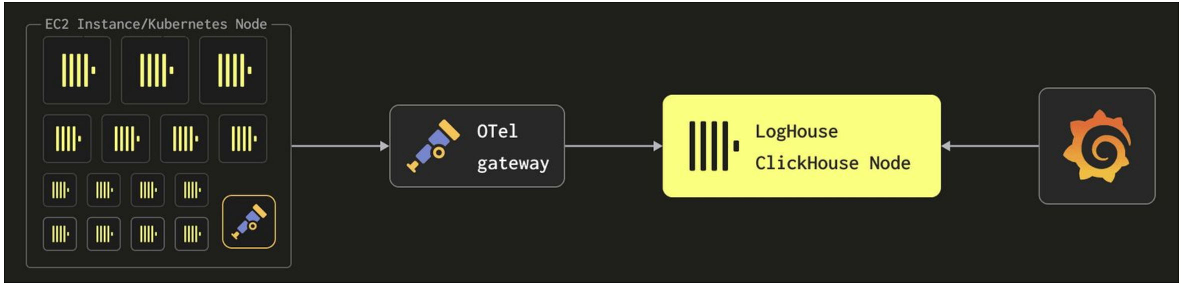
Granular log retention, Saved money on Datadog

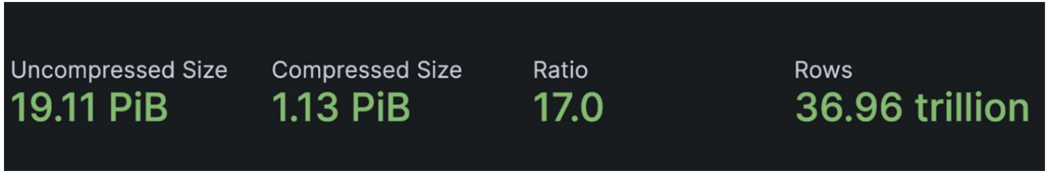
Tradeoffs

1.5 FTEs to build stack

https://clickhouse.com/blog/building-a-logging-platform-with-clickhouse-and-saving-millions-over-datadog

Dogfooding ClickHouse across o11y





Uncompressed Size	Compressed Size	Ratio	Rows
19.11 PiB	1.13 PiB	17.0	36.96 trillion

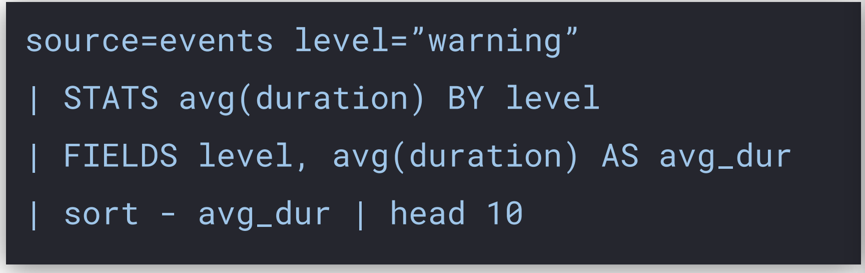
ClickHouse

Common considerations

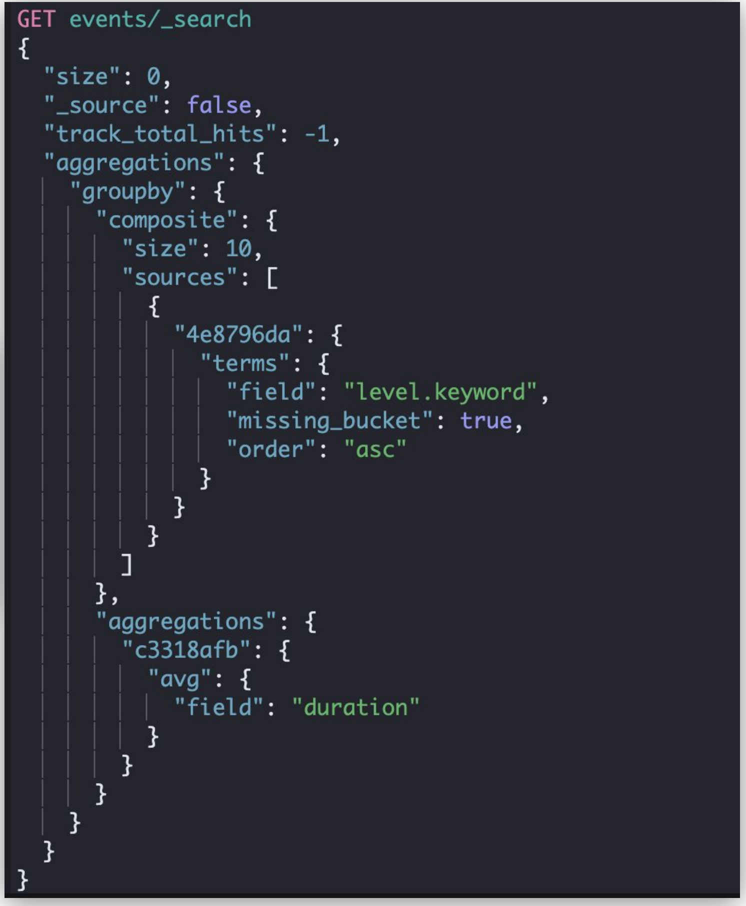
Query language considerations

“SQL is not compact enough compared to domain-specific query languages”

A simple query ?

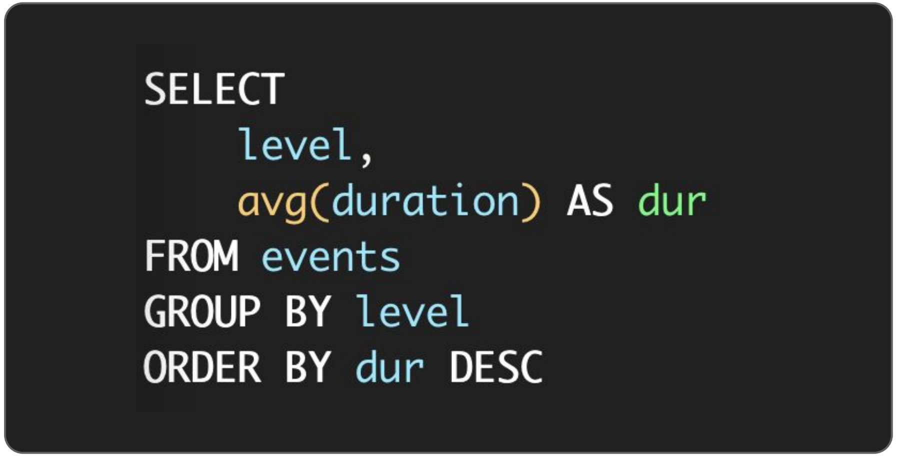


```
source=events level=”warning”
| STATS avg(duration) BY level
| FIELDS level, avg(duration) AS avg_dur
| sort - avg_dur | head 10
```



```
GET events/_search
{
  "size": 0,
  "_source": false,
  "track_total_hits": -1,
  "aggregations": {
    "groupby": {
      "composite": {
        "size": 10,
        "sources": [
          {
            "4e8796da": {
              "terms": {
                "field": "level.keyword",
                "missing_bucket": true,
                "order": "asc"
              }
            }
          }
        ]
      },
      "aggregations": {
        "c3318afb": {
          "avg": {
            "field": "duration"
          }
        }
      }
    }
  }
}
```

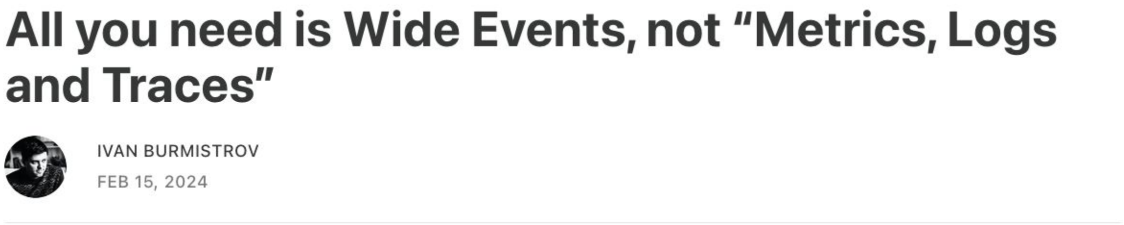
In good old SQL



```sql
SELECT
    level,
    avg(duration) AS dur
FROM events
GROUP BY level
ORDER BY dur DESC
```

Schema considerations

It helps to stop thinking about *“metrics”, “logs” and “traces”* separately and just think them as *“wide events”*



All you need is Wide Events, not “Metrics, Logs and Traces”

IVAN BURMISTROV

FEB 15, 2024

https://isburmistrov.substack.com/p/all-you-need-is-wide-events-not-metrics

https://news.ycombinator.com/item?id=39529775

Visualization considerations

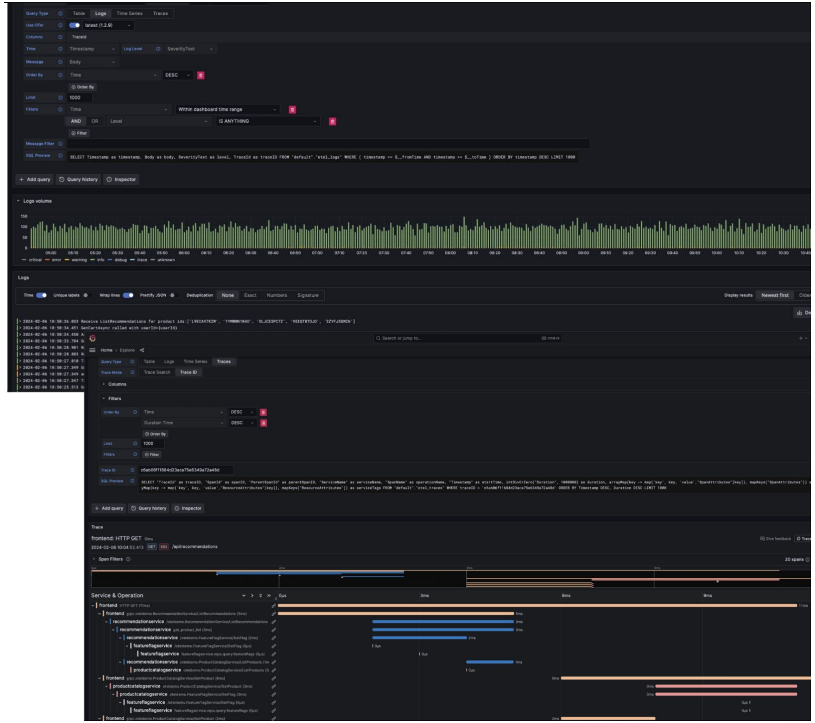
Grafana

Apache Superset

Perses

Metabase

Build your own



Use OTel: latest (1.2.9)

Columns: TraceId

Time: Timestamp | Log Level: SeverityText

Message: Body

Order By: Time DESC

Limit: 1000

Filters: Time Within dashboard time range; AND Level IS ANYTHING

```
SELECT Timestamp as timestamp, Body as body, SeverityText as level, TraceId as traceID FROM "default"."otel_logs" WHERE ( timestamp >= $__fromTime AND timestamp <= $__toTime ) ORDER BY timestamp DESC LIMIT 1000
```

Time ● Unique labels ○ Wrap lines ● Prettify JSON ○ Deduplication: None | Exact | Numbers | Signature

```
2024-02-06 10:50:36.033 Receive ListRecommendations for product ids:['L9ECAV7KIM', '1YMWWN1N4O', 'OLJCESPC7Z', '9SIQT8TOJO', '2ZYFJ3GM2N']
2024-02-06 10:50:34.451 GetCartAsync called with userId={userId}
2024-02-06 10:50:34.450
2024-02-06 10:50:33.784
2024-02-06 10:50:28.901
2024-02-06 10:50:28.883
2024-02-06 10:50:27.810
2024-02-06 10:50:27.349
2024-02-06 10:50:27.349
2024-02-06 10:50:27.347
2024-02-06 10:50:25.513
```

Query Type: Traces

Trace Mode: Trace ID

Order By: Time DESC; Duration Time DESC

Limit: 1000

Trace ID: c6ab06f11684d23aca75e6349a72a48d

```
SELECT "TraceId" as traceID, "SpanId" as spanID, "ParentSpanId" as parentSpanID, "ServiceName" as serviceName, "SpanName" as operationName, "Timestamp" as startTime, intDivOrZero("Duration", 1000000) as duration, arrayMap(key -> map('key', key, 'value',"SpanAttributes"[key]), mapKeys("SpanAttributes")) a
yMap(key -> map('key', key, 'value',"ResourceAttributes"[key]), mapKeys("ResourceAttributes")) as serviceTags FROM "default"."otel_traces" WHERE traceID = 'c6ab06f11684d23aca75e6349a72a48d' ORDER BY Timestamp DESC, Duration DESC LIMIT 1000
```

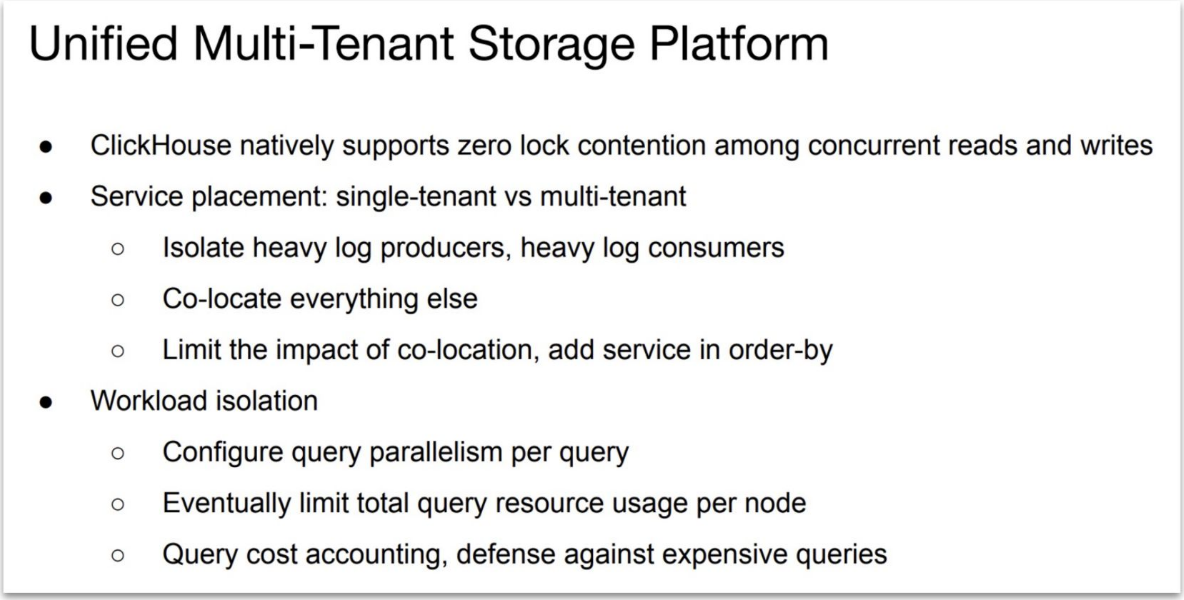
frontend: HTTP GET 12ms

GET 500 /api/recommendations

Multi-tenancy considerations

Example: Uber

Consider datastore ability to limit resources by table, user, session



Unified Multi-Tenant Storage Platform

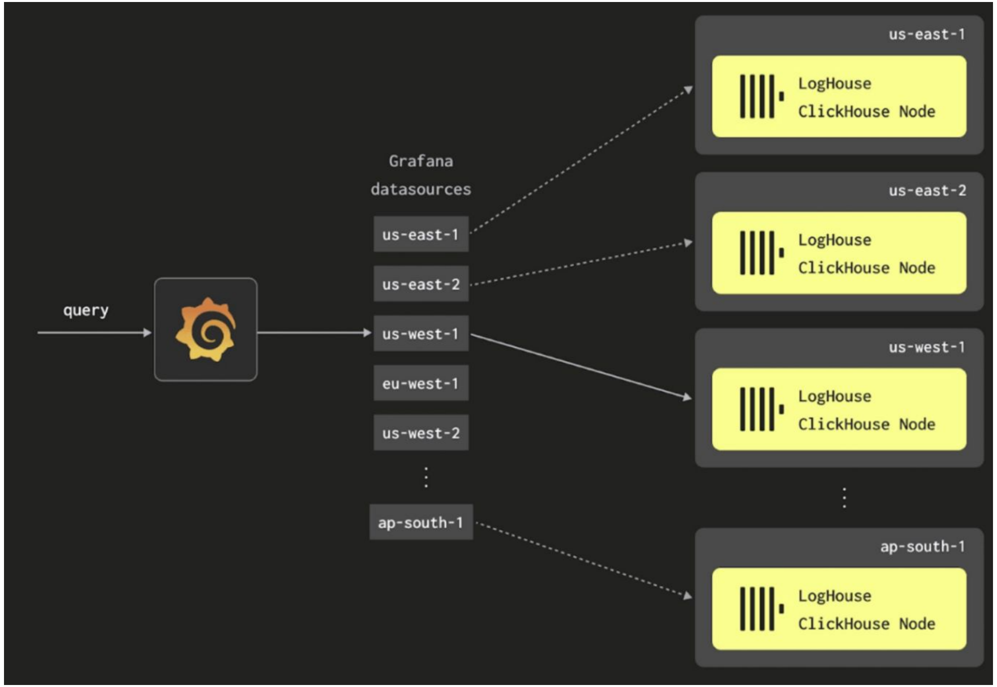
- ClickHouse natively supports zero lock contention among concurrent reads and writes
- Service placement: single-tenant vs multi-tenant
  - Isolate heavy log producers, heavy log consumers
  - Co-locate everything else
  - Limit the impact of co-location, add service in order-by
- Workload isolation
  - Configure query parallelism per query
  - Eventually limit total query resource usage per node
  - Query cost accounting, defense against expensive queries

Multi-region deployment considerations

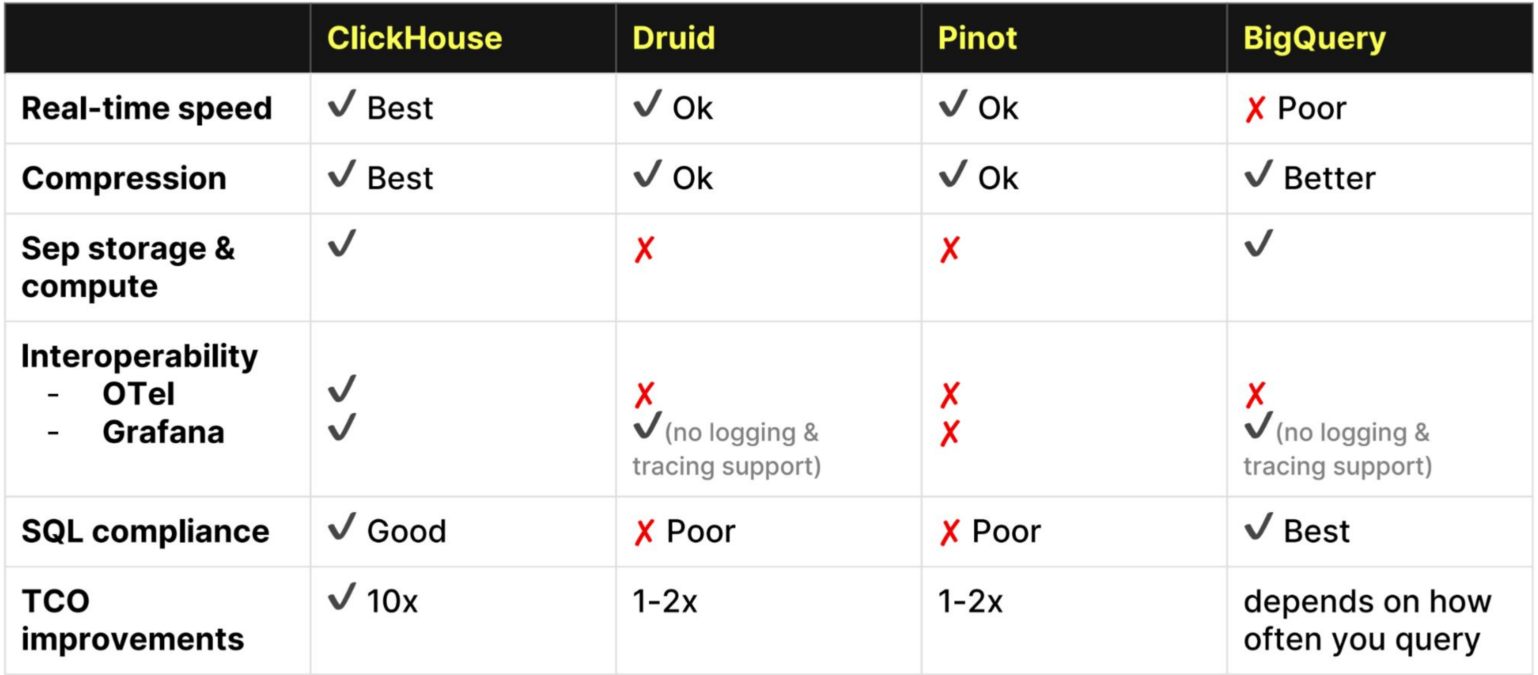
Example: ClickHouse Inc.

Per region data collection, cross-region queries

Resilient to AZ outage but not region outage



Choice of analytical datastore matters



	ClickHouse	Druid	Pinot	BigQuery
Real-time speed	✓ Best	✓ Ok	✓ Ok	✗ Poor
Compression	✓ Best	✓ Ok	✓ Ok	✓ Better
Sep storage & compute	✓	✗	✗	✓
Interoperability - **OTel** - **Grafana**	 ✓ ✓	 ✗ ✓ (no logging & tracing support)	 ✗ ✗	 ✗ ✓ (no logging & tracing support)
SQL compliance	✓ Good	✗ Poor	✗ Poor	✓ Best
TCO improvements	✓ 10x	1-2x	1-2x	depends on how often you query

https://benchmark.clickhouse.com

ClickHouse

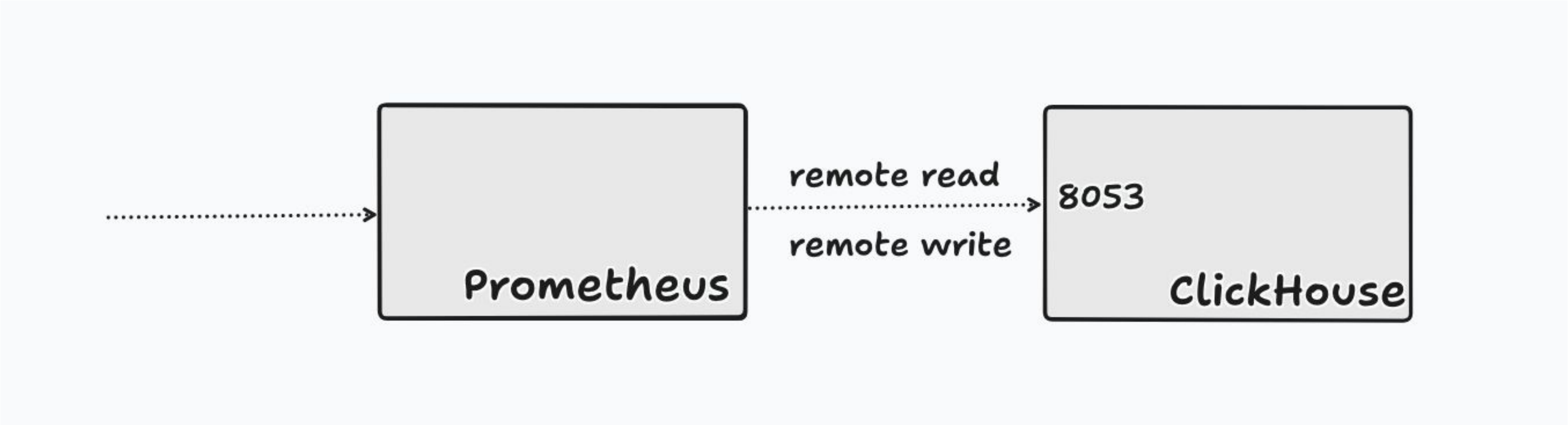
Demo

ClickHouse

ClickHouse Timeseries Engine

ClickHouse Timeseries Engine

Pull Request: https://github.com/ClickHouse/ClickHouse/pull/64183



ClickHouse

Questions