ClickHouse

How We Built ClickStack

An open source, OpenTelemetry-native Observability stack

Mike Shi - Cofounder of HyperDX, Observability Product @ ClickHouse

What is ClickHouse?

Open source

- Development started 2009
- Production 2012
- OSS 2016
- #1 analytics database on DB-Engines

Column-oriented

- Best for aggregations
- Files per column
- Sorting and indexing
- Background merges

Distributed

- Replication
- Sharding
- Multi-master
- Cross-region

OLAP database

- Analytics use cases
- Aggregations
- Visualizations
- Mostly immutable data

ClickStack

An Open Source Observability Stack

- An opinionated, end-to-end Observability stack built on ClickHouse
  - Data ingestion
  - Schemas
  - User interface and workflows
- All components are Open source
- Native OpenTelemetry (OTel) support
- Democratizes ClickHouse Observability, lowering the barrier for teams of any size
- Deploy and run in minutes anywhere

A Brief History of Observability

Search

1970s — Grep

2000s — Splunk

2010s — Elastic

Metrics

2008 — Graphite

2012 — Prom

2017 — Thanos

Increasing Volume & Cardinality

ClickHouse

Challenges with search

Search engines

You know for search, just not aggregations

- Fast for “needle in a haystack” searches
- Not optimized for frequent writes
- Poor performance when “zooming out” for trends with aggregations at scale

✓ Cost efficient

✓ Fast search

✗ Fast aggregations

Challenges with time series database

Metric stores

Built for metrics first

- Limited search
- Stronger aggregation performance unless higher cardinality
- But requires pre-aggregation - losing ability to root cause

✓ Cost efficient

× Fast search

✓ Fast aggregations

Can we avoid tradeoffs?

Mystery data store

Built for observability

- Lightweight indexes for search
- High performance aggregations on high cardinality data
- Use object storage for cheap and infinite storage

✓ Cost efficient

✓ Fast search

✓ Fast aggregations

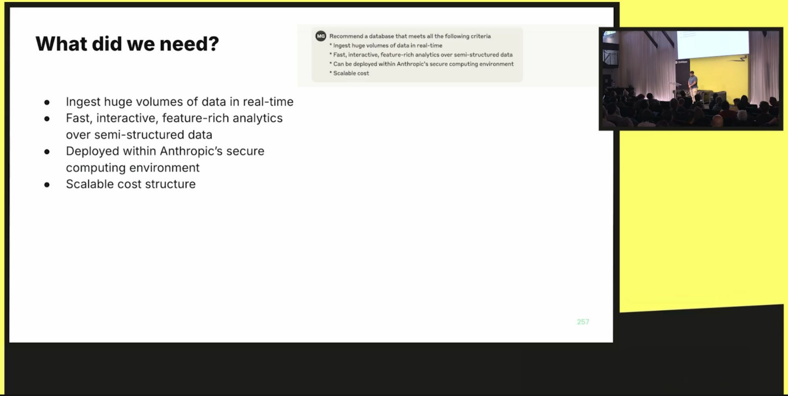
???

There's actually a lot of things we wanted!

- ✓ Resource efficient
- ✓ Petabyte-scale production tested
- ✓ Compute/storage separation
- ✓ Fast search
- ✓ Fast aggregations
- ✓ Full text search
- ✓ Fast aggregations
- ✓ High compression ratios
- ✓ Expressive query language
- ✓ No log size limits
- ✓ Supports structured wide events
- ✓ Schema on read
- ✓ Object storage support
- ✓ Open source
- ✓ Flexible indexing strategy

What were others doing?

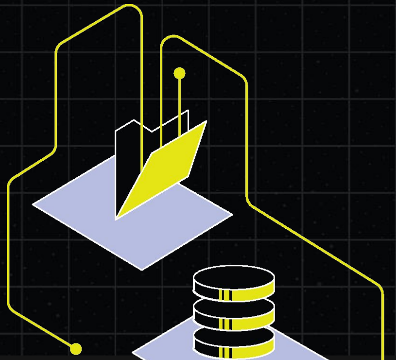
NETFLIX



ANTHROP\C

ClickHouse

01 Going Columnar



Columnar storage

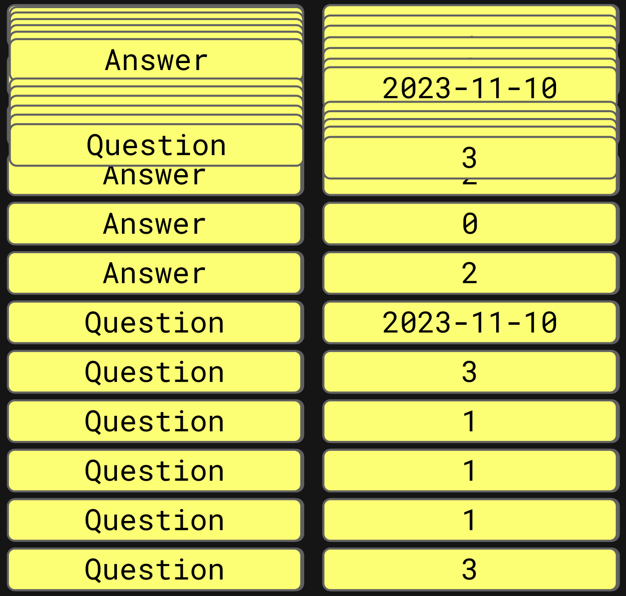
Row based

PostType	CreationDate
Answer	2023-11-09
Question	2023-11-10
Answer	2023-11-12
Question	2023-11-13
Question	2023-11-14
Answer	2023-11-15
Question	2023-11-15
Answer	2023-11-15
Question	2023-11-16
Answer	2023-11-17
Question	2023-11-18
Answer	2023-11-13

vs

Column based

`ORDER BY (PostType,CreationDate) CODEC(Delta)`



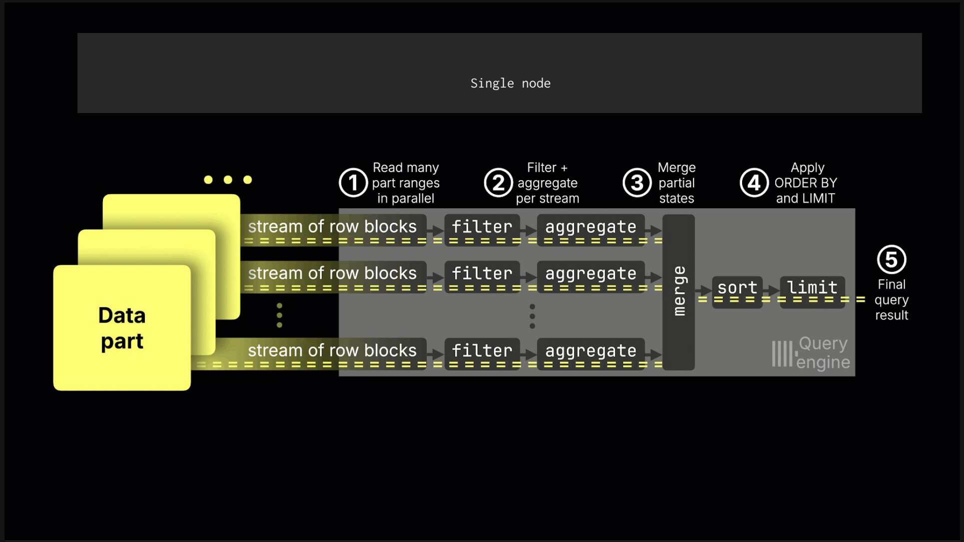
Benefits

- High Compression
- Only read columns in query
- Reduced I/O

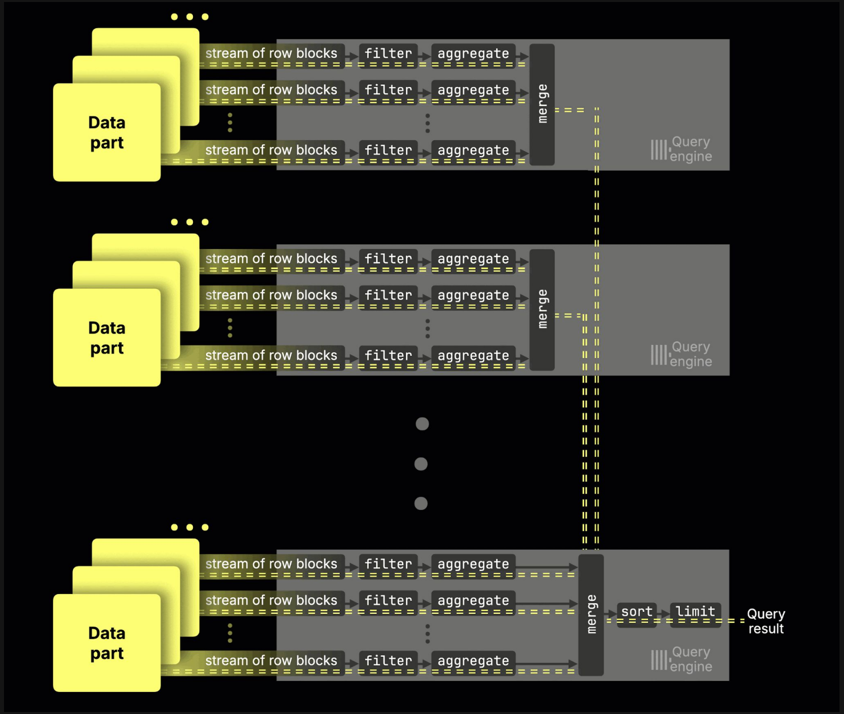
ClickHouse

Parallel vectorized data processing

Utilizing the full resources available



Multi node

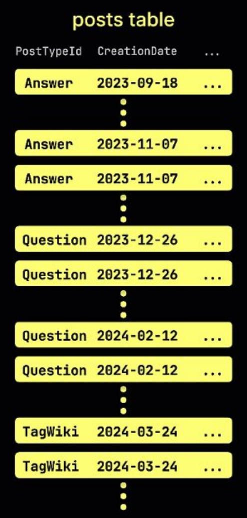


ClickHouse

Sparse Primary Indices

Provides an index over our columnar storage

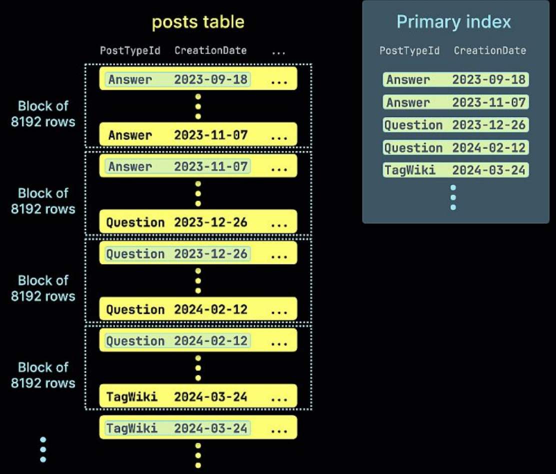
```
CREATE TABLE stackoverflow.posts
(
    `Id` Int32 CODEC(Delta(4), ZSTD(1)),
    `PostTypeId` Enum8('Question' = 1, 'Answer' = 2,
    'Wiki' = 3, 'TagWikiExcerpt' = 4,
    'TagWiki' = 5, 'ModeratorNomination' = 6,
    'WikiPlaceholder' = 7, 'PrivilegeWiki' = 8),
    `AcceptedAnswerId` UInt32,
    `CreationDate` DateTime64(3, 'UTC'),
    `Score` Int32,
    `ViewCount` UInt32 CODEC(Delta(4), ZSTD(1)),
    `Body` String,
    `OwnerUserId` Int32,
    `OwnerDisplayName` String,
    `LastEditorUserId` Int32,
    `LastEditorDisplayName` String,
    `LastEditDate` DateTime64(3, 'UTC') CODEC(Delta(8), ZSTD(1)),
    `LastActivityDate` DateTime64(3, 'UTC'),
    `Title` String,
    `Tags` String,
    `AnswerCount` UInt16 CODEC(Delta(2), ZSTD(1)),
    `CommentCount` UInt8,
    `FavoriteCount` UInt8,
    `ContentLicense` LowCardinality(String),
    `ParentId` String,
    `CommunityOwnedDate` DateTime64(3, 'UTC'),
    `ClosedDate` DateTime64(3, 'UTC'),
    INDEX view_count_idx ViewCount TYPE minmax GRANULARITY 1
)
ENGINE = MergeTree
PARTITION BY toYear(CreationDate)
ORDER BY (PostTypeId, toDate(CreationDate))
```



Sparse Primary Indices

Exploiting the index for fast filtering

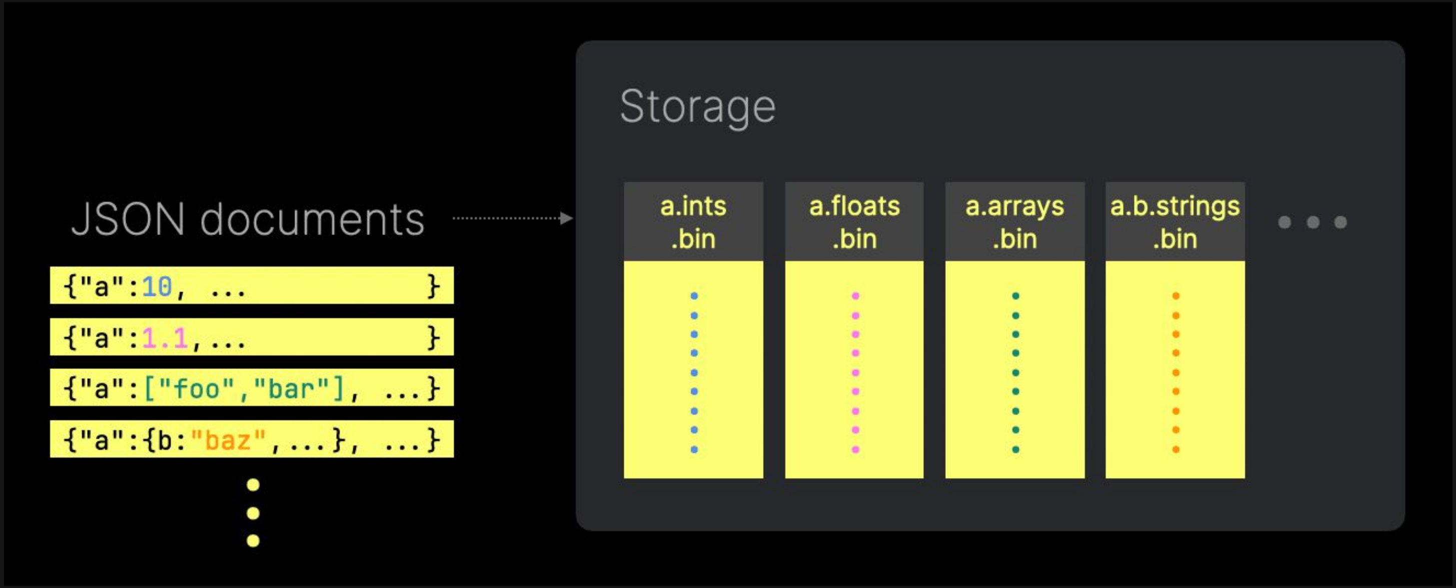
```
EXPLAIN indexes = 1
SELECT count()
FROM stackoverflow.posts_ordered
WHERE (CreationDate >= '2024-01-01') AND (PostTypeId = 'Question')

┌─explain──────────────────────────────────────────────────────────────────────────┐
│ Expression ((Project names + Projection))                                        │
│   Aggregating                                                                    │
│     Expression (Before GROUP BY)                                                 │
│       Expression                                                                 │
│         ReadFromMergeTree (stackoverflow.posts_ordered)                          │
│         Indexes:                                                                 │
│           PrimaryKey                                                             │
│             Keys:                                                                │
│               PostTypeId                                                         │
│               toDate(CreationDate)                                               │
│             Condition: and((PostTypeId in [1, 1]), (toDate(CreationDate) in [19723, +Inf])) │
│             Parts: 14/14                                                         │
│             Granules: 39/7578                                                    │
└──────────────────────────────────────────────────────────────────────────────────┘

13 rows in set. Elapsed: 0.004 sec.
```



ClickHouse

Wide structured events 🤝 JSON type

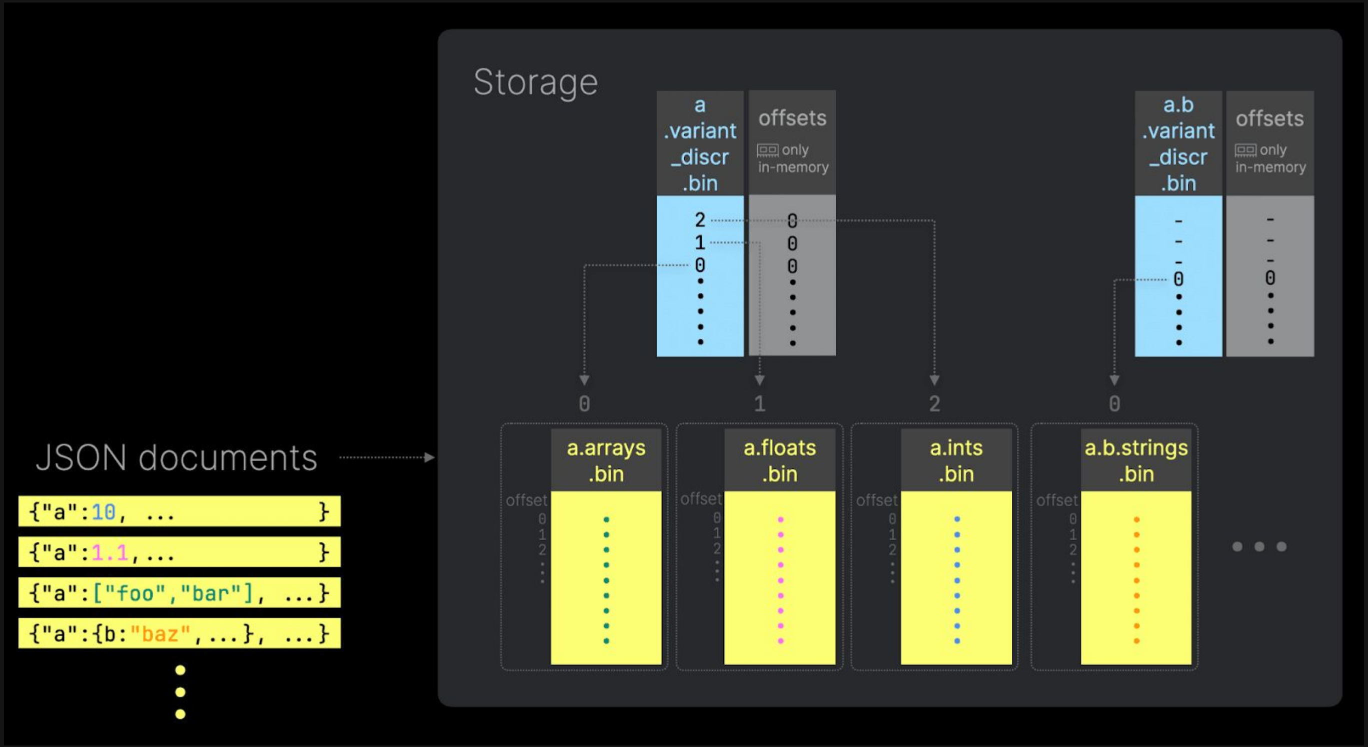
Dynamic columns per field



ClickHouse

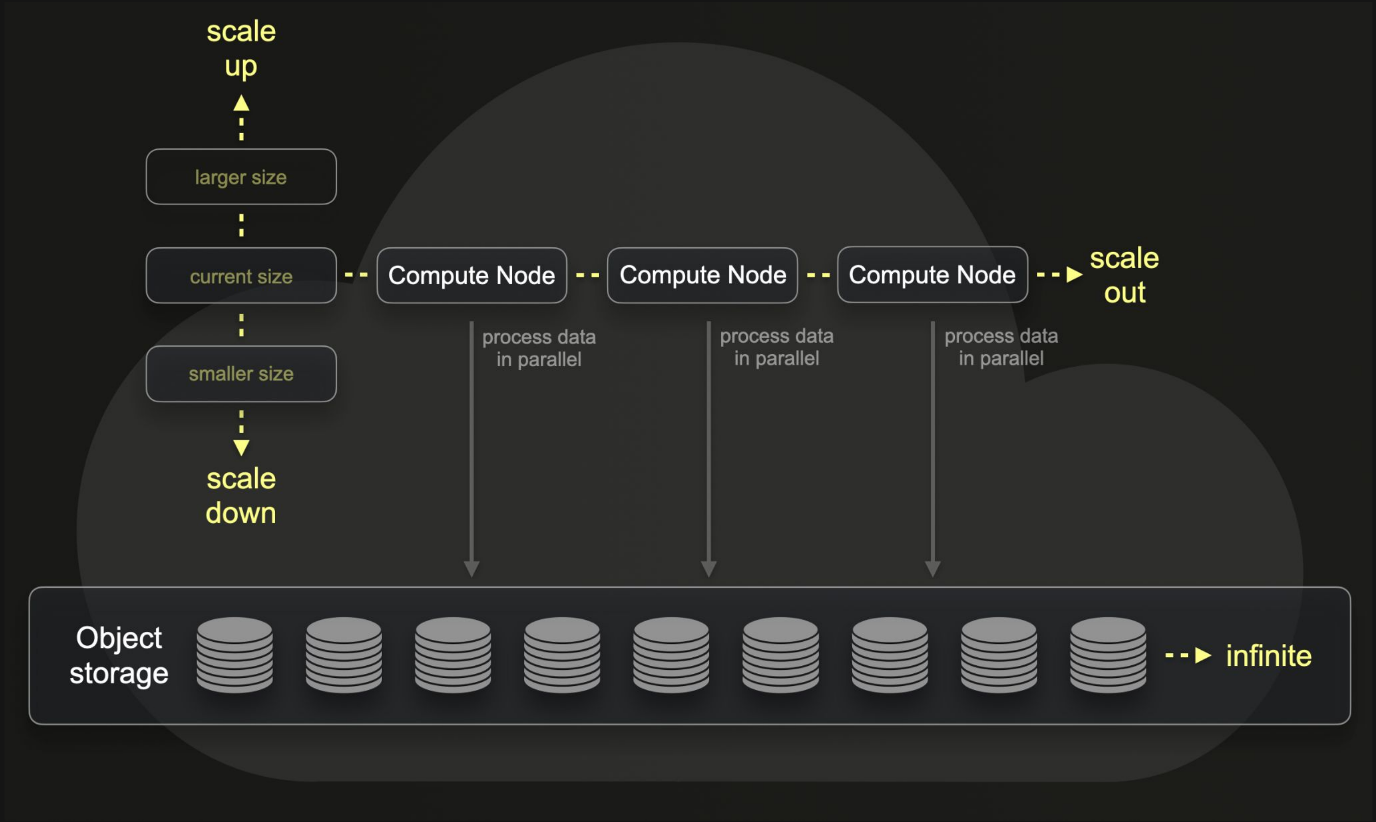
Wide structured events 🤝 JSON type

With type delineation



ClickHouse

S3 storage



Benefits

- Low cost retention
- Isolate read/write paths
- Scale compute dynamically for investigations

Why columnar stores for O11y

SQL native
SQL is the lingua franca for data, familiar, universal, and easy to build on

Schema on read/schema on write
Schema on write with JSON type, Schema on read with parsing functions

Fast writes and space efficient indices
Bloom filters and full text search

Fast selective filtering
With sparse primary key index

Fast aggregations
Over high cardinality immutable data

Cost-efficient and scalable
Industry-leading compression and scalability

02 Why OpenTelemetry?

Why OpenTelemetry?

The de facto standard = the commoditization of Observability instrumentation

- **One Instrumentation approach → Many Backends:** Send data to any observability tool (e.g. ClickStack, Datadog, Prometheus).
- **Unified Telemetry:** Logs, metrics, traces in a single consistent format, easier to correlate and analyze
- **Broad Language Support:** SDKs for Java, Python, Go, JS, .NET, and more - allowing easy instrumentation across services and layers with ease
- **Vendor neutral = no Lock-In: you own your data and stay in control**
- **Open Standard, Growing Community: Backed by the CNCF and major vendors**

ClickHouse

Beyond OpenTelemetry

Bring your own schema and ingestion tooling

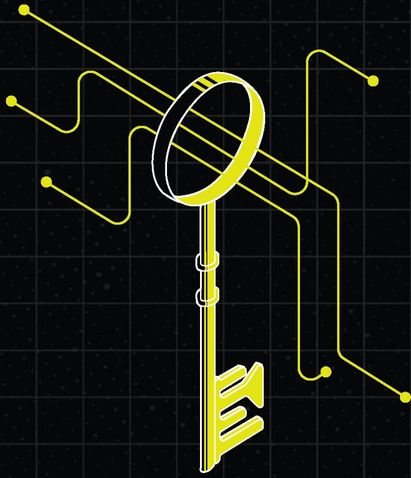
- OTel Native != OTel exclusive
- We decided to support any event which can be represented as a row with both labels and metrics.
- The term **"wide events"** is increasing popular. These aim to unify logs, metrics, and traces into a single structured record, enabling simpler, high-cardinality observability at scale.

We just need a **DateTime** and some **default columns to select**.


```json
{
  "timestamp": "2025-08-18T09:30:00Z",
  "service": "checkout-service",
  "trace_id": "abc123def456",
  "user_id": "user_789",
  "region": "eu-west-1",
  "status_code": 200,
  "latency_ms": 152,
  "cpu_usage": 0.42,
  "error": false,
  "memory_mb": 312,
  "endpoint": "/api/v1/orders",
  "http_method": "POST",
  "cart_size": 4,
  "payment_provider": "stripe",
  "retry_count": 0,
  "release_version": "1.3.7"
}
```


ClickHouse

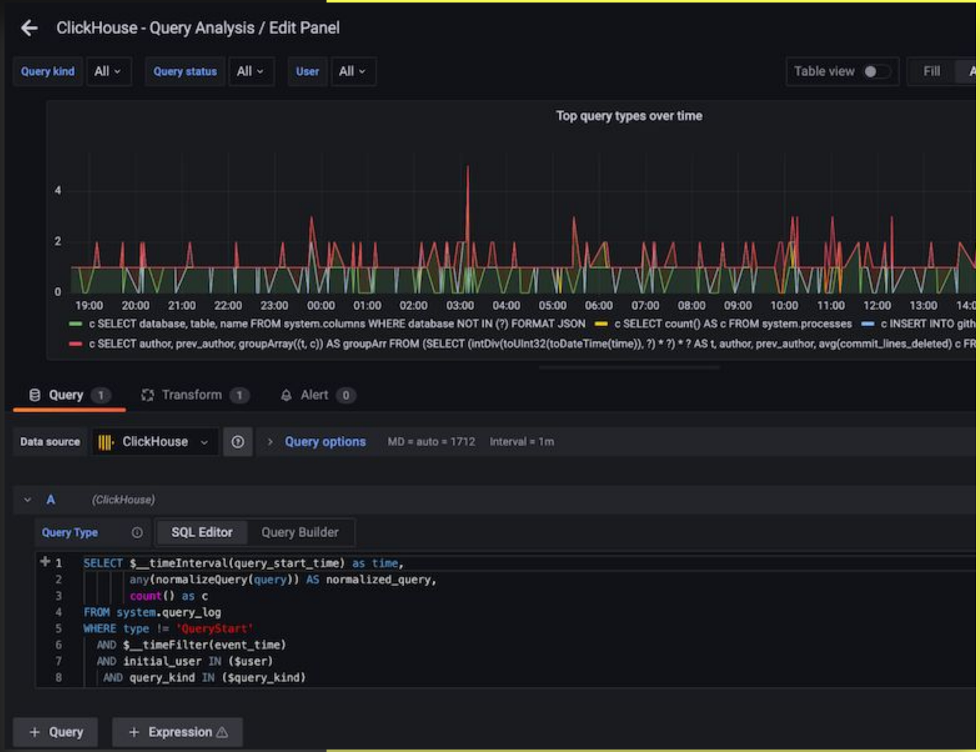
03 Developer Experience



Why we need a dedicated inter

Not Everyone is an O11y Expert

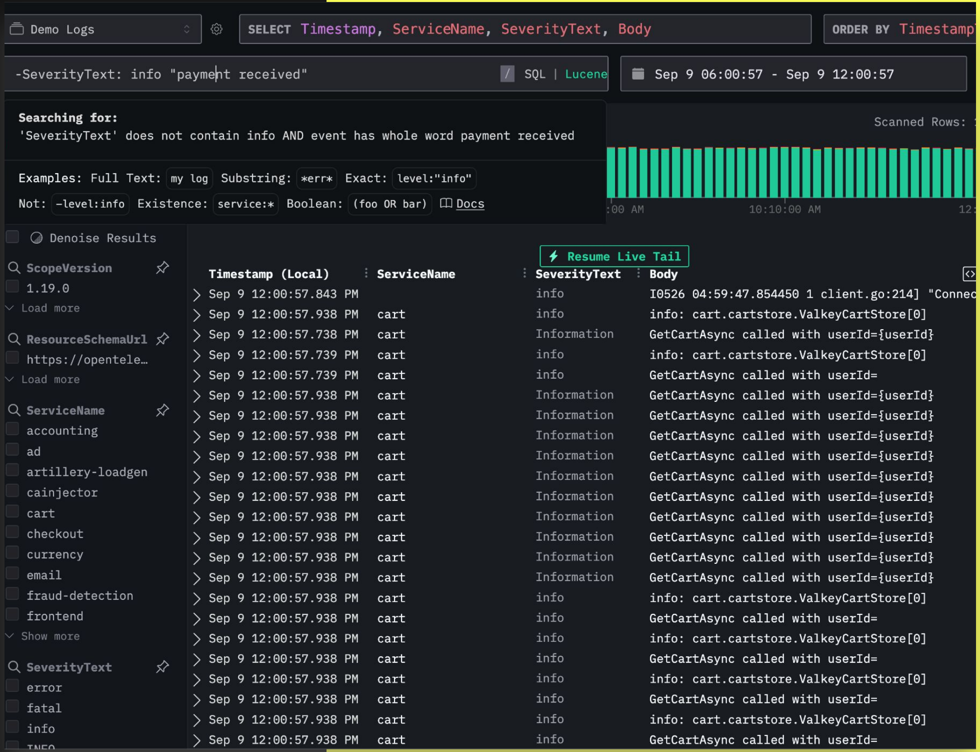
- We need to build for every day developers
- We can't always expect users to write optimal SQL queries and understand internals
- SQL is great for deeper analysis but not for o11y ad hoc investigations



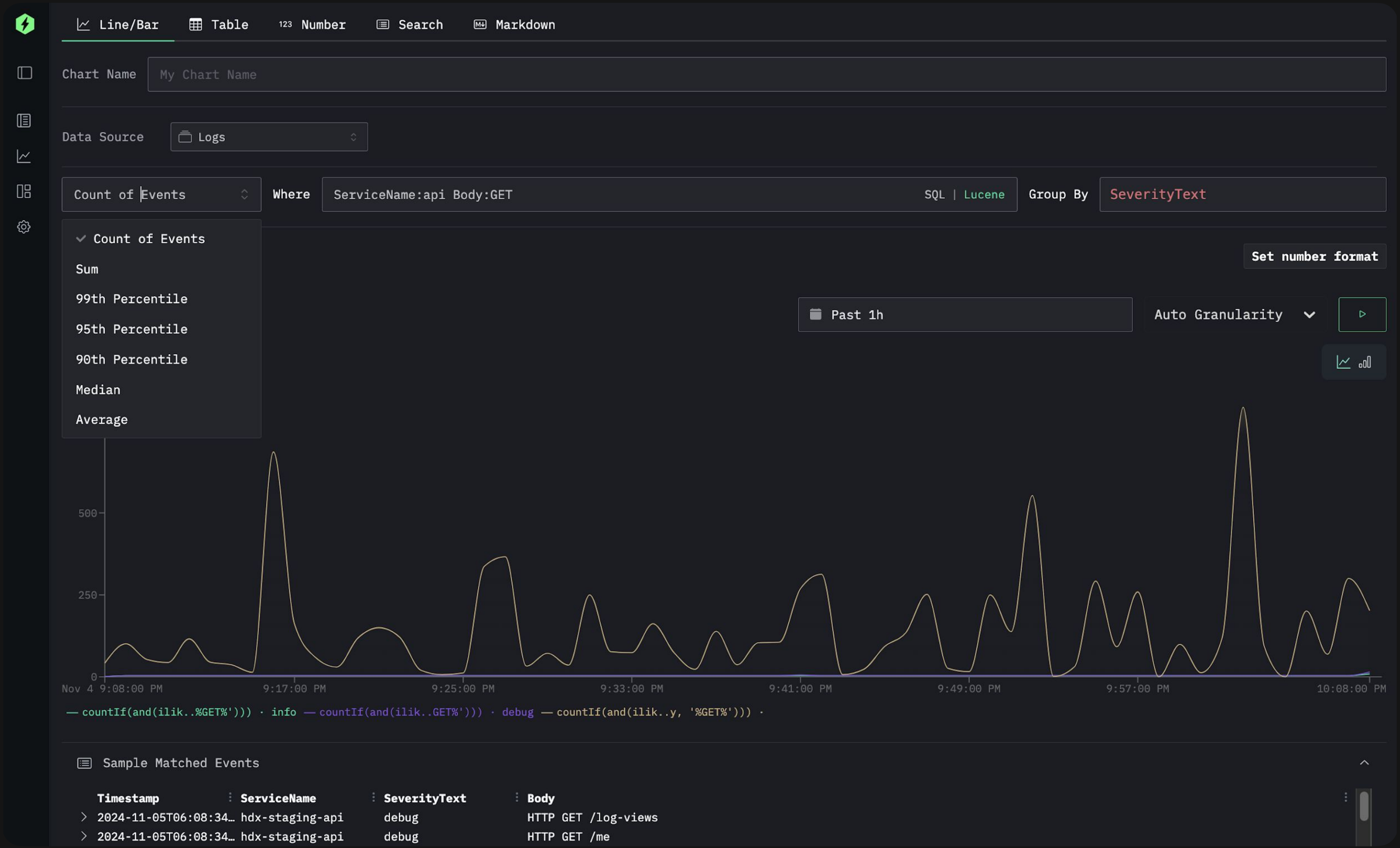
Hide complexity

Build for every day developers

- Lucene Syntax (Converts to SQL)
- English Query Explainer
- UI Filters
- Escape Hatch to SQL



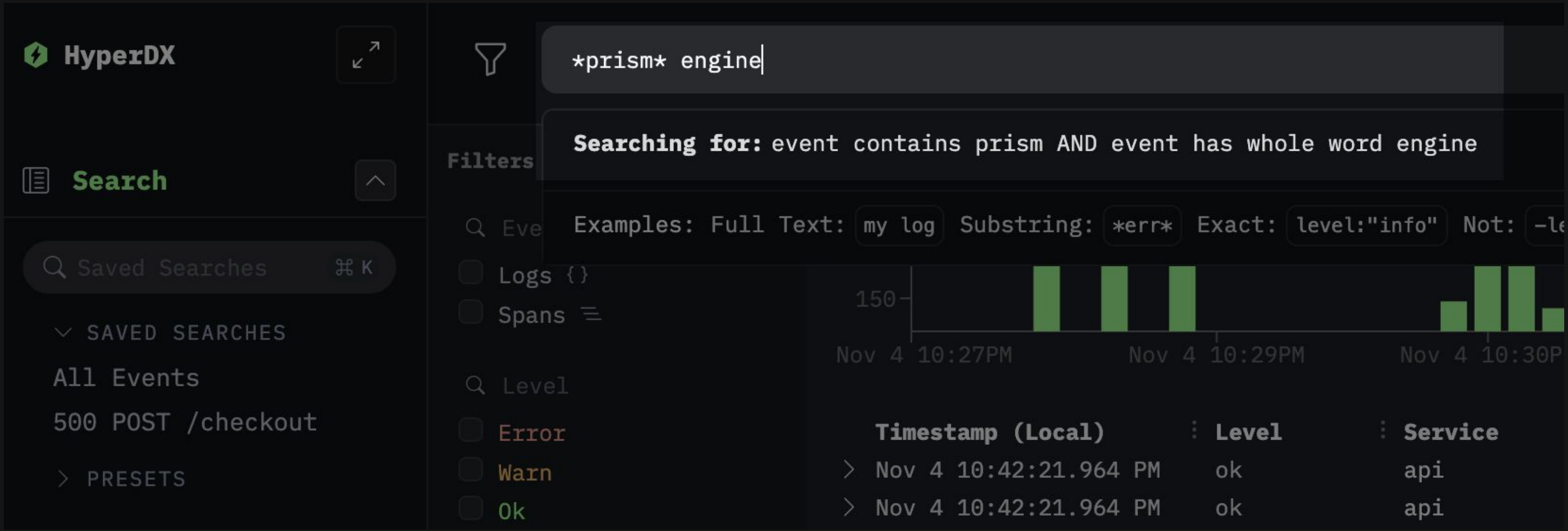
ClickHouse


Line/Bar
Table
Number
Search
Markdown
Chart Name
My Chart Name
Data Source
Logs
Count of Events
Where
ServiceName:api Body:GET
SQL | Lucene
Group By
SeverityText
Count of Events
Sum
99th Percentile
95th Percentile
90th Percentile
Median
Average
Set number format
Past 1h
Auto Granularity
500
250
0
Nov 4 9:08:00 PM
9:17:00 PM
9:25:00 PM
9:33:00 PM
9:41:00 PM
9:49:00 PM
9:57:00 PM
10:08:00 PM
Sample Matched Events
Timestamp
ServiceName
SeverityText
Body
2024-11-05T06:08:34… hdx-staging-api debug HTTP GET /log-views
2024-11-05T06:08:34… hdx-staging-api debug HTTP GET /me

But also fast observability needs optimizations that run database-deep

Database and UI invariably are tightly coupled

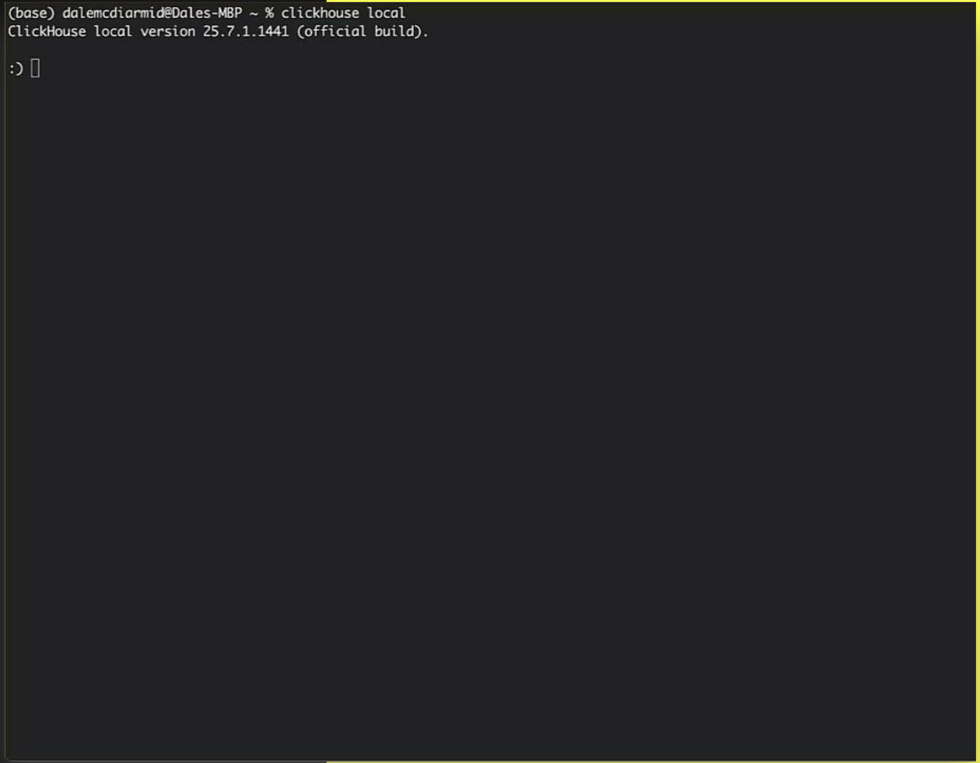
Index Design 🤝 UX Design



Many small optimizations

Align ordering with primary key

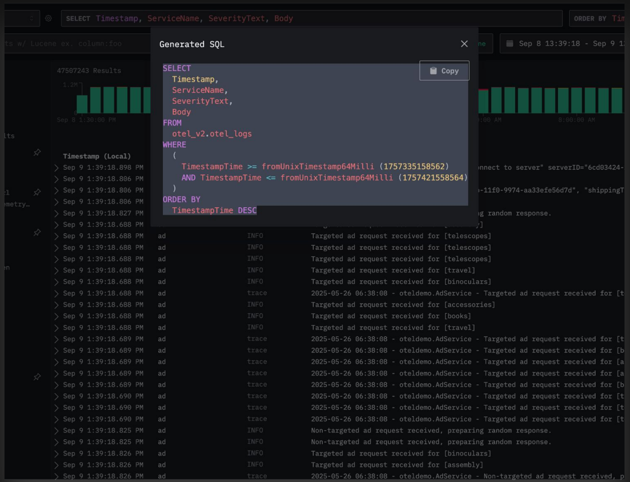
- `(toStartOf[Minute|Hour](Timestamp), ServiceName)` for primary key
- HyperDX queries ordered by raw Timestamp only, misaligned with sorting key → more data scanned
- Optimization missed: ClickHouse couldn't apply `optimize_read_in_order`, slowing down "latest results" queries with small LIMITs
- UI introspects the table and ensures `ORDER BY toStartOf[Minute|Hour](Timestamp) DESC, Timestamp DESC)`



ClickHouse

Many small optimizations

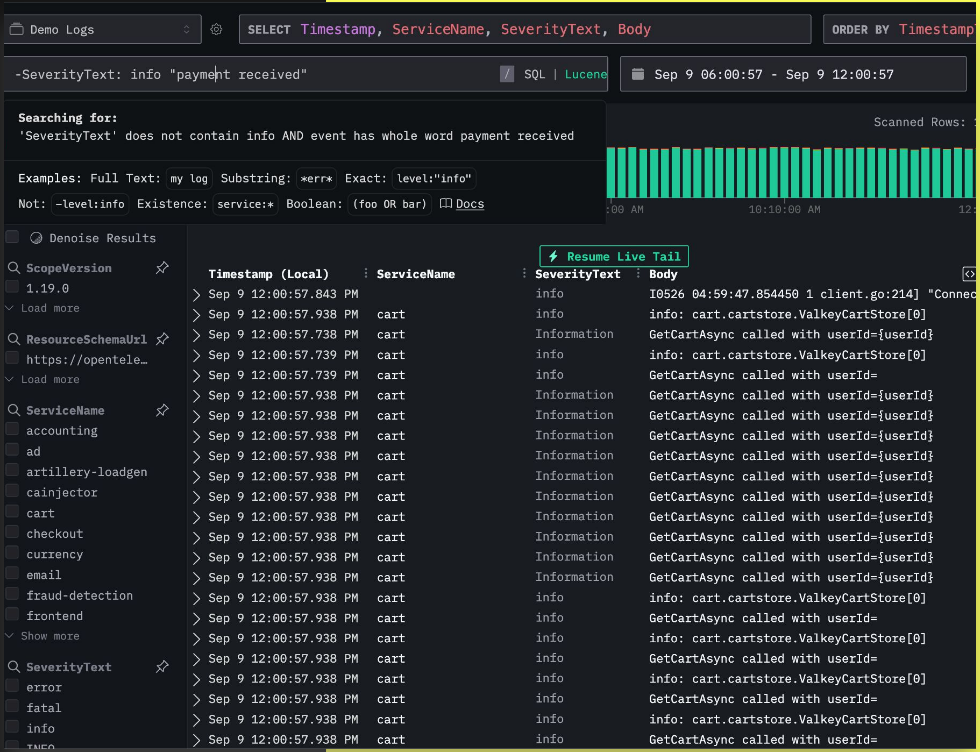
Chunk wide time ranges



Chunk the requests and execute until sufficient results

Summary

- Observability has evolved beyond search engines and dedicated stores.
- Column-oriented databases are the ideal foundation for observability.
- SQL remains the lingua franca for data exploration and adoption.
- But to fully exploit column-stores, UIs must be tightly coupled with the database analyzer - delivering fast performance and efficient workflows.



ClickHouse

Thank You

Public demo

play-clickstack.clickhouse.com

Get started docs

Say hello! michael.shi@clickhouse.com

ClickHouse