

ClickHouse - A Dataflow Template for Batch Ingestion

About Me



Bentsi Leviov

Senior Software Engineer
ClickHouse

- Passionate about technology, startups and finance
- Maintainer of multiple CH integrations
 - Apache Spark/Glue
 - Apache Beam/DataFlow
 - Apache Flink
 - dbt
 - Tableau
 - PowerBI
- B.S.c in Computer Science
- MBA from TAU & NYU

Table Of Contents

01

ClickHouse Introduction

High-performance analytical database

02

Apache Beam ClickHouseIO

Native Beam connector for ClickHouse

03

BigQuery <> ClickHouse DataFlow Template

Batch pipeline for data transfer

04

ClickHouseIO RM

Local ClickHouse for Beam integration tests

05

Future Improvements

Enhancements, coverage, and streaming

01

ClickHouse Intro

The fastest analytical database



Speaks SQL fluently



Processes data very fast



Highly efficient storage



Easily scalable to any size

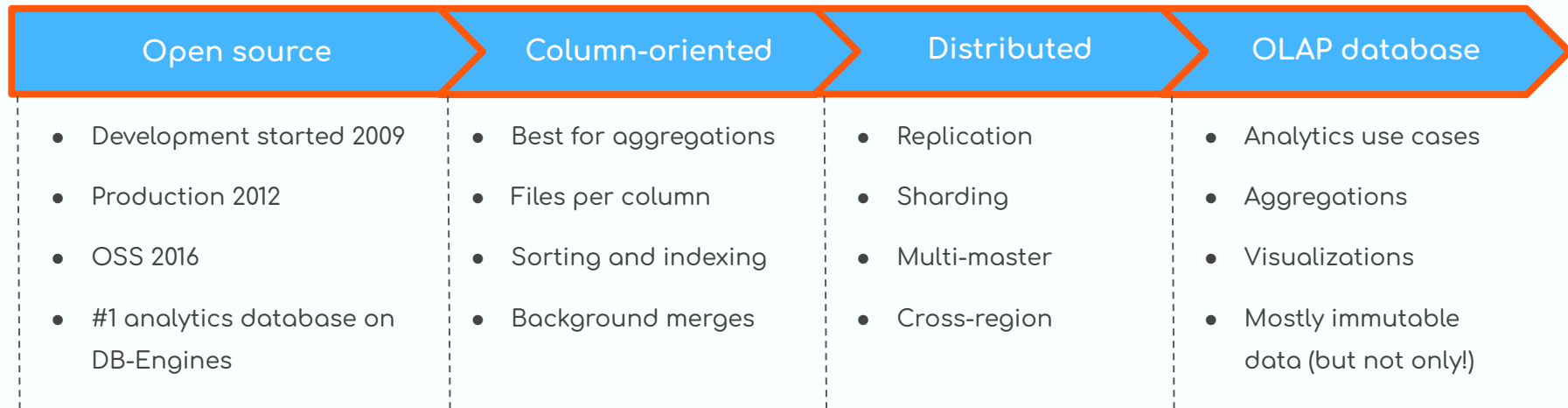
What is ClickHouse?

ClickHouse is an **open-source**,
columnar **OLAP database**

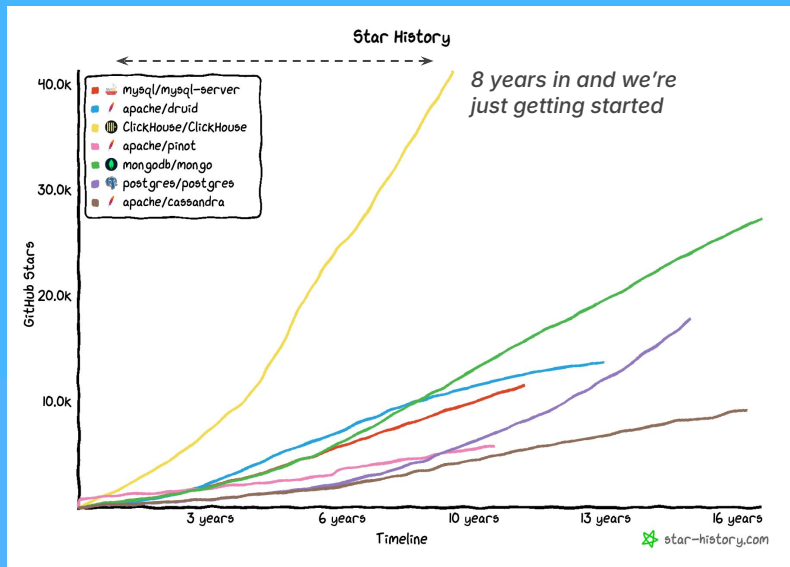
Designed for **blazing fast**
analytics of massive
volumes of data



ClickHouse In A Nutshell



ClickHouse momentum



ClickHouse Open Source

- 41k+ stars
- 10k+ Slack members
- 1.6k contributors
- 250k+ community members

The world's most popular analytics database trusted by

Meta

NETFLIX

Bloomberg



NVIDIA

Microsoft

GitLab

ERICSSON

Walmart

Deutsche Bank

CISCO

lyft

servicenow

ebay

CLOUDFLARE

ANTHROPIC

deepseek

COMCAST

SONY

IBM

ROKT

Spotify

instacart

Vercel

ClickHouse Use Cases

Real-time analytics

- Applications, Dashboards, APIs
- Customer facing
- Interactive
- Querying ClickHouse directly

Data warehousing

- Business data for internal use
- Use any BI software on top (Tableau, Power BI, Superset, Metabase)
- Read data directly from Iceberg, Parquet, Delta Lake, etc.

Observability

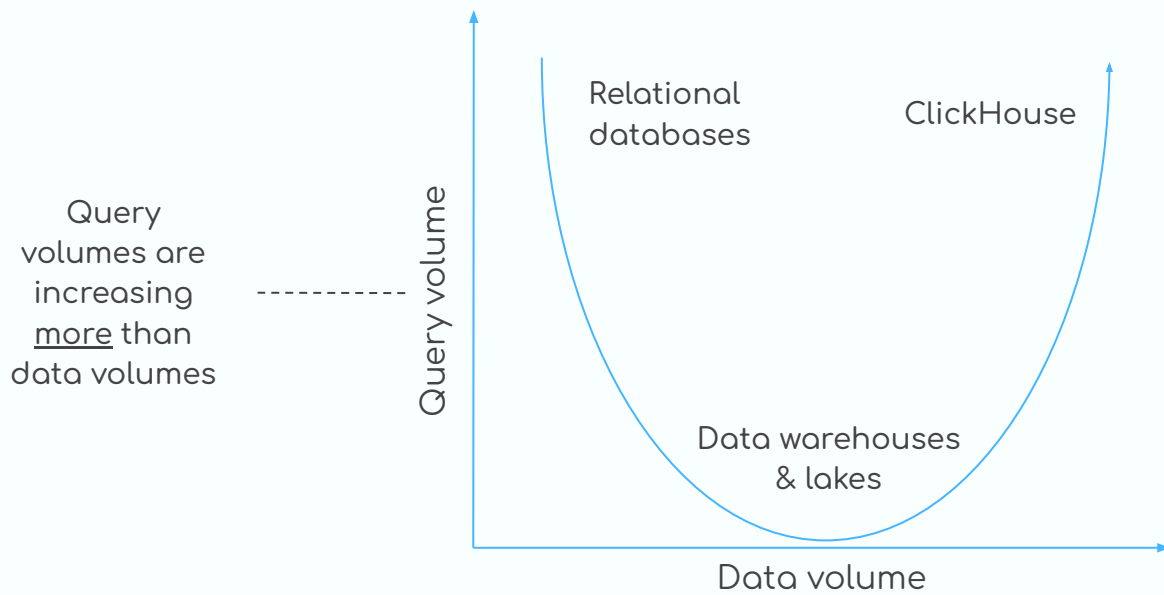
- Logs, metrics, traces, session replays & errors
- ClickStack (OpenTelemetry + ClickHouse + HyperDX)
- Unify on SQL as the query language

AI / machine learning

- Feature Store
- Vector Search
- LLM Observability
- MCP Server

ClickHouse Use Cases

Skating to where the puck is going



02

ClickHouseIO

Native Beam connector for ClickHouse

ClickHouseIO - Key Features



ClickHouse JDBC Client

- Based on ClickHouse official Java client.
- Uses efficient RowBinary format and inserts data in large block.



Batching & Reliability

- Configurable retries via FluentBackoff with exponential backoff support.
- Buffer rows in-memory for microbatch inserts.



Schema & Types

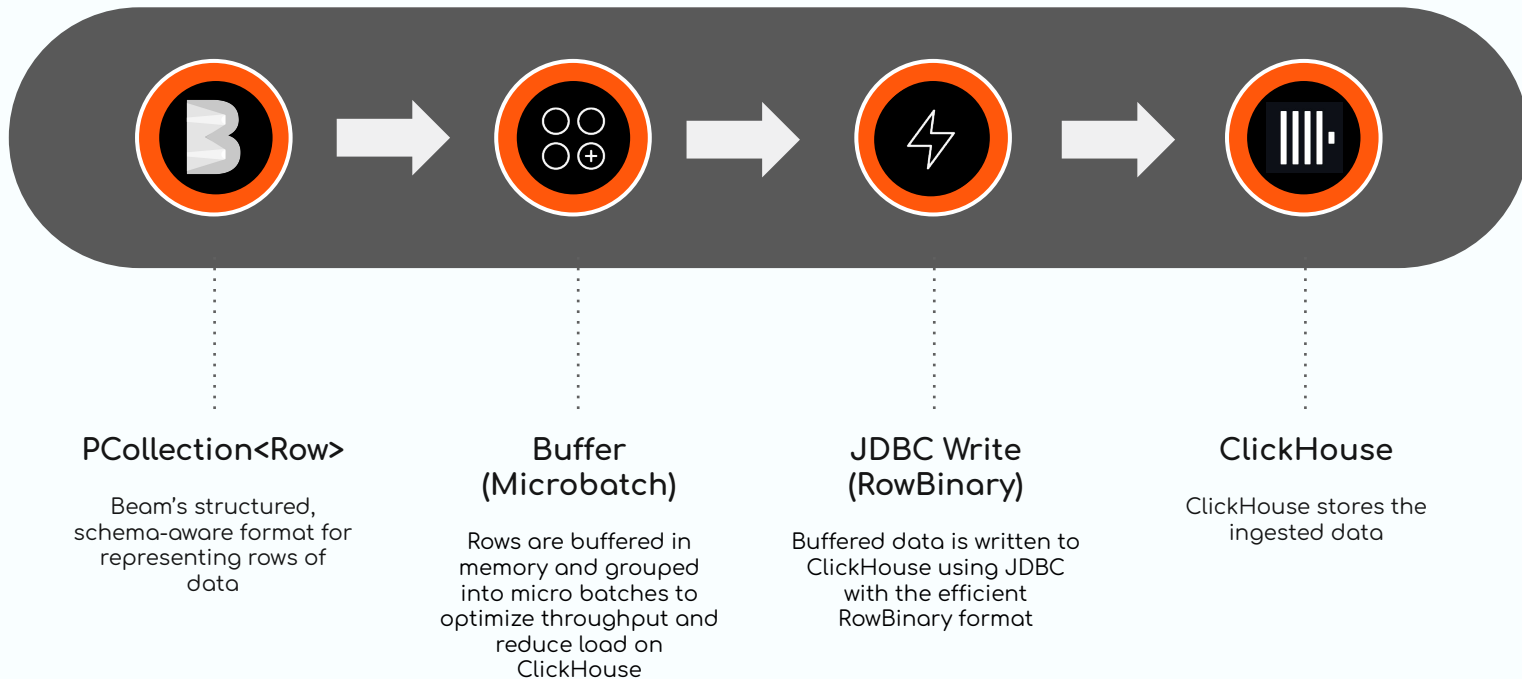
- Convert Beam Row schema to ClickHouse schema automatically.
- Supports ClickHouse Column aliases, default types and materializations.



Configurations

- `maxInsertBlockSize`, `insertDistributedSync`, `insertQuorum`, `insertDeduplicate`, `maxRetries`

ClickHouseO Internals



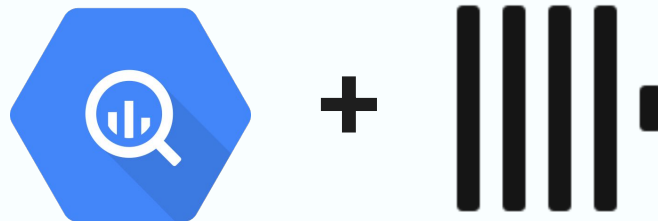
03

BigQuery <> ClickHouse

A DataFlow batch template for data transfer

BigQuery <> ClickHouse

- BigQuery is great for large-scale analytics, but not always ideal for low-latency serving¹
- BigQuery is great for large-scale analytics, but some workloads benefit from ClickHouse's lower cost-per-query, especially for frequent dashboard access²
- A DataFlow template would enable easy, automated data integration to use ClickHouse as a real-time query service on Top of BigQuery Data



1. <https://clickhouse.com/comparison/bigquery>
2. <https://clickhouse.com/blog/clickhouse-bigquery-migrating-data-for-realtime-queries>

Implementation Limitations

BigQueryIO

- BigQuery and ClickHouse have different type systems (e.g., TIMESTAMP vs. DateTime64)
- Not all types are 1:1 mappable, requires lossy or opinionated conversions
- Users must often review or adjust schemas manually

ClickHouseIO

Build for `TableRow`

- Currently supports only `PCollection<Row>`
- Requires full Beam Schema to be present and compatible
- No native support yet for TableRow, Avro, or other untyped formats

First Step

Read From BigQuery

- Read data either by table name or custom SQL query using Beam's native BigQueryIO.
- The data is retrieved as TableRow objects (a flexible map-like structure).

```
/*  
 * Step #1: Read from BigQuery. If a query is provided then it is used to get the TableRows.  
 */  
PCollection<TableRow> tableRows =  
    pipeline.apply(  
        name: "Read From Big Query",  
        BigQueryConverters.ReadBigQueryTableRows.newBuilder()  
            .setOptions(options.as(BigQueryToClickHouseOptions.class))  
            .build());
```


Second Step

Convert TableRow to Beam's Row

- Since ClickHouseIO only accepts PCollection<Row>, we must convert TableRow into Beam Row format.
- This requires schema inference and field-level type conversion, including handling differences like date/time types and arrays.
- The Beam Schema is constructed from the ClickHouse table metadata to ensure compatibility.

```
/*  
 * Step #2: Transform TableRow to Row  
 */  
PCollection<Row> rows =  
    tableRows  
        .apply(  
            name: "Convert to Beam Row",  
            ParDo.of(new TableRowToBeamRowFn(beamSchema, clickHouseSchema)))  
        .setRowSchema(beamSchema);
```

Third Step

Write to ClickHouse

- The converted Beam Row objects are written to ClickHouse using the native ClickHouseIO.Write.
- Several advanced options can be configured, such as insert_quorum, insert_deduplicate, and max_insert_block_size.
- The pipeline assumes the ClickHouse table already exists with a matching schema.

```
/*
 * Step #: Write to ClickHouse
 */

ClickHouseIO.Write clickHouseWriter =
    ClickHouseIO.write(clickHouseJDBCURL, options.getClickHouseTable());

if (options.getMaxInsertBlockSize() != null) {
    clickHouseWriter.withMaxInsertBlockSize(options.getMaxInsertBlockSize());
}

if (options.getInsertDistributedSync() != null) {
    clickHouseWriter.withInsertDistributedSync(options.getInsertDistributedSync());
}

if (options.getInsertQuorum() != null) {
    clickHouseWriter.withInsertQuorum(options.getInsertQuorum());
}

if (options.getInsertDeduplicate() != null) {
    clickHouseWriter.withInsertDeduplicate(options.getInsertDeduplicate());
}

if (options.getMaxRetries() != null) {
    clickHouseWriter.withMaxRetries(options.getMaxRetries());
}

// Step 3: Write data to ClickHouse
rows.apply( name: "Write to ClickHouse", clickHouseWriter);
```

Launching the Template

DataFlow Console

The screenshot shows the 'Dataflow / Create job' interface. The left sidebar contains navigation links: Overview, Monitoring, Jobs, Pipelines, Workbench, and Snapshots. The main content area is titled 'Create job from template'. It features a 'Dataflow templates' section with a sub-section 'Job builder' (Create custom jobs with the builder form and YAML editor). Below this, there are three input fields: 'Job name' (with a note 'Must be unique among running jobs'), 'Regional endpoint' (set to 'us-central1 (Iowa)' with a dropdown arrow), and 'Dataflow template' (set to 'BigQuery to ClickHouse' with a dropdown arrow). A descriptive paragraph follows: 'The BigQuery to ClickHouse template is a batch pipeline that ingests data from a BigQuery table into ClickHouse table. The template can either read the entire table or read specific records using a supplied query.' Below this is the 'Host' section with a 'ClickHouse JDBC URL' field and a note: 'The target ClickHouse JDBC URL, in the format "jdbc:clickhouse://host/port/schema". Any JDBC option could be added at the end of the JDBC URL. For example, "jdbc:clickhouse://localhost:8123/default"'. The 'Required Parameters' section includes a 'Username for ClickHouse endpoint' field (note: 'The ClickHouse username to authenticate with.') and a 'ClickHouse Table Name' field (note: 'The target ClickHouse table name to insert the data to.'). At the bottom, there is an 'Encryption' section with a 'Google-managed encryption key' option (note: 'Keys owned by Google').

gcloud

Built for me, not for you

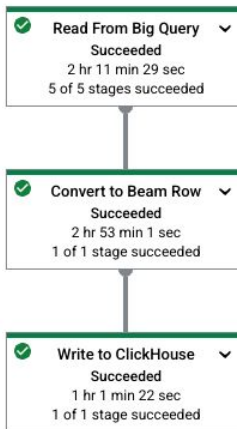
```
gcloud dataflow flex-template run
"bigquery-clickhouse-`date
+%Y%m%d-%H%M%S`\"
--template-file-gcs-location
"gs://clickhouse-dataflow-templates/bigquer
y-clickhouse-metadata.json" \
--parameters query="select * from
`ch-integrations.test_dataset.bitcoin`
limit
5",jdbcUrl="jdbc:clickhouse://***.us-east1.
gcp.clickhouse.cloud:8443/default?ssl=true&
sslmode=NONE",clickHouseUsername="default",
clickHousePassword="**",clickHouseTable="cr
ypto bitcoin transactions deduped"
```

Full Flow

Job steps view

Graph view

CLEAR SELECTION






04

ClickHouseIO RM

Local ClickHouse for Beam integration tests

ClickHouseResourceManager.java

- Extends Beam's `TestContainerResourceManager` to spin up a real ClickHouse instance via Testcontainers.
- Provides a consistent API to:
 -  Create and clean up ClickHouse tables
 -  Insert and query test data
 -  Validate table schemas and row counts
- Encapsulates all lifecycle and JDBC connection logic, reducing boilerplate in test code.

```
/**
 * Client for managing ClickHouse resources.
 *
 * <p>The class is thread-safe.
 */
public class ClickHouseResourceManager extends TestContainerResourceManager<GenericContainer<?>> 19 usages  ⬆ Benti
    implements ResourceManager {

    private static final Logger LOG = LoggerFactory.getLogger(ClickHouseResourceManager.class); 14 usages

    private final Connection connection; 7 usages

    // 8123 is the default port that ClickHouse is configured to listen on
    private static final int CLICKHOUSE_INTERNAL_PORT = 8123; 1 usage

    private static final String DEFAULT_CLICKHOUSE_CONTAINER_NAME = "clickhouse/clickhouse-server"; 1 usage

    private static final String DEFAULT_CLICKHOUSE_CONTAINER_TAG = "23.8"; 1 usage

    private final String jdbcConnectionString; 2 usages

    final List<String> managedTableNames = new ArrayList<>(); 4 usages

    ClickHouseResourceManager(Builder builder) throws SQLException { 1 usage  ⬆ Bentsi Leviav
        this(/* clickHouseConnection*/ null, buildContainer(builder), builder);
    }
}
```

05

Future Improvements

ClickHouseIO + Templates

Disclaimer

This is for informational purposes only and does not constitute a commitment to deliver any specific features or functionality. The development, release, and timing of any features or functionality described herein are subject to change and remain at the sole discretion of ClickHouse



- Upgrade Java client version
- Support more types (all those available by the java client)
- Expose Java client settings
- Expand supported data formats
- Improve streaming capabilities
 - Dead-letter Handling (DLQ)
 - Flush buffer based on time
 - Beam metrics and Observability enhancements

DataFlow Templates

1

PubSub To ClickHouse

Stream data from Pub/Sub directly into ClickHouse to power real-time analytics with low-latency ingestion

2

GCS To ClickHouse

Load batch data from Google Cloud Storage into ClickHouse

[Feature Request]: Create Template for Pub/Sub to ClickHouse #4

Open

BentsiLeviav opened on Jan 28 Member

Related Template(s)

ClickHouse

What feature(s) are you requesting?

Description

Develop a Google Dataflow Flex template to stream data from Pub/Sub topics to ClickHouse.

Objectives

- Build a Dataflow pipeline template to stream data from Pub/Sub topics to ClickHouse.
- Support for common Pub/Sub formats such as JSON and Avro.

Create sub-issue

[Feature Request]: Create Template for GCS to ClickHouse #3

Open

BentsiLeviav opened on Jan 28 Member

Related Template(s)

ClickHouse

What feature(s) are you requesting?

Description

Develop a Google Dataflow Flex template to read data from GCS Storage (GCS) to ClickHouse.

Objectives

- Build a Dataflow pipeline template to read data from GCS and write to ClickHouse.
- Ensure support for common file formats (e.g., CSV, Parquet, Avro, JSON).

Create sub-issue

Let's Connect!



 **slack**
from @Salesforce



ClickHouse
GitHub



My LinkedIn



All ClickHouse
Events



We are hiring!



ClickHouse
Academy



My GitHub

Thank You

Bentsi Leviav
Senior Software Engineer, ClickHouse