

Weekly

W1

Sensor data → data collection layer → cleaning and integration layer → storage layer (relational + time series database) → service layer (API) → visualization layer.

W2

Progress

List of open-source data warehousing solutions

- Apache Hive
- Apache Druid
- ClickHouse
- Apache Kudu
- Greenplum
- Apache Pinot
- PostgreSQL
- StarRocks
- Apache Doris

Evaluation Criteria

- Cost : Includes licensing fees (if applicable), hardware and infrastructure costs, as well as operational and maintenance expenses.
- Performance: Measures the efficiency and speed of a data warehousing solution in handling queries
- Scalability: Evaluates how well the data warehouse can accommodate growth

- Usability: SQL support, user interface, etc.
- Community Support: documentation and contributions from users and developers.
- Ecosystem Integration: Evaluates the ability of the data warehouse to connect and integrate with other technologies, such as ETL tools, BI tools, and data ingestion frameworks.
- Data Modeling Capabilities: Looks at how well the data warehouse can support different data models, such as star schema, snowflake schema, or data lakes.

W3

<https://benchmark.clickhouse.com/>

<https://benchant.com/ranking/database-ranking>

<https://celerddata.com/blog/what-you-should-know-before-using-clickbench>

1. Core Positioning & Use Cases

System	Core Focus	Use Cases
Hive	Hadoop-based offline data warehouse	Large-scale batch ETL, offline analytics (e.g., log processing, data lake storage)
Druid	Real-time time-series analytics	Real-time monitoring, ad analytics, user behavior analysis (low latency + high concurrency)
ClickHouse	High-performance OLAP columnar database	Wide-table aggregations, log/behavioral analytics (extremely fast single-table queries)
Kudu	Real-time updatable storage engine	Real-time data updates + analysis (e.g., IoT sensor data, financial transactions)
Greenplum	Enterprise MPP data warehouse	Complex ETL, business intelligence (BI), hybrid workloads (PostgreSQL-compatible)
Pinot	Low-latency real-time OLAP engine	Interactive real-time analytics (e.g., clickstream analysis, like LinkedIn's use case)
PostgreSQL	General-purpose RDBMS (supports OLTP/OLAP)	Transaction processing, medium-scale analytics (with extensions like TimescaleDB)
StarRocks	High-performance MPP OLAP engine	High-concurrency real-time analytics, ad-hoc queries (e.g., e-commerce dashboards)

Doris	Lightweight MPP OLAP engine	Real-time reporting, mid-scale analytics (simplified fork of StarRocks)
--------------	-----------------------------	---

2. Architecture & Performance

System	Storage Model	Query Latency	Throughput	Scalability	Real-time Ingestion
Hive	Row/Column (ORC)	Minutes	High	High (via HDFS)	No (batch-only)
Druid	Columnar + inverted index	Sub-second	Medium	High (distributed)	Yes (streaming)
ClickHouse	Columnar	Seconds	Very High	Moderate (sharding via ZooKeeper)	Yes (batch/stream)
Kudu	Columnar	Seconds	High	High (distributed)	Yes (real-time updates)
Greenplum	Row/Columnar	Minutes	High	Moderate (MPP scaling)	Yes (batch)
Pinot	Columnar + inverted index	Sub-second	High	High (auto-sharding)	Yes (streaming)
PostgreSQL	Row-based (column extensions)	Seconds-minutes	Medium	Limited (replication)	Yes (transactional)
StarRocks	Columnar	Sub-second	Very High	High (elastic)	Yes (streaming)
Doris	Columnar	Seconds	High	High (distributed)	Yes (batch/stream)

3. Key Features

System	Strengths	Weaknesses	Data Updates	Transactions
Hive	SQL compatibility, mature ecosystem	High latency, not for interactive	Overwrite-only	No
Druid	Low latency, high concurrency	Complex pre-aggregation, weak joins	No	No
ClickHouse	Blazing-fast scans, high compression	Poor multi-table joins, no transactions	Limited (via ALTER)	No
Kudu	Real-time updates, Hadoop integration	Limited storage scale, needs compute engines (e.g., Impala)	Yes	Partial (row-level)

Greenplum	Complex query optimization, ACID	Costly scaling, weak real-time	Yes	Yes (full ACID)
Pinot	Fast multi-dimensional aggregations	Complex setup, smaller community	No	No
PostgreSQL	Versatile, extensible	Performance degrades at scale	Yes	Yes (OLTP)
StarRocks	High concurrency, vectorized engine	Young ecosystem, fewer optimizations	Yes	Partial (batch)
Doris	Simple setup, MySQL compatibility	Fewer features vs. StarRocks	Yes	Partial (batch)

4. Ecosystem & Community

- **Hive:** Mature, tightly integrated with Hadoop (HDFS, Spark).
- **Druid:** Active community, cloud-native, works with Kafka/Flink.
- **ClickHouse:** Rapidly growing ecosystem (tools like Superset).
- **Greenplum:** Strong enterprise support, legacy migration-friendly.
- **PostgreSQL:** Largest community, rich extensions (TimescaleDB, Citus).
- **StarRocks/Doris:** Active Chinese community, easy for local users.

5. Recommendations

- **Real-time analytics:** Druid, Pinot, StarRocks (sub-second latency).
- **Batch processing:** Hive, Greenplum.
- **High concurrency:** ClickHouse, StarRocks, Doris.
- **Frequent updates:** Kudu, Greenplum, PostgreSQL.
- **Hybrid workloads:** PostgreSQL (with extensions), Greenplum.
- **Lightweight real-time:** Doris (simpler than StarRocks).

Summary

- **Prioritize raw speed:** ClickHouse (single-table), StarRocks (multi-table).
- **Ecosystem maturity:** Hive (Hadoop), PostgreSQL (extensions).
- **Real-time updates:** Kudu, Greenplum.
- **Low latency + concurrency:** Druid, Pinot, StarRocks.

Tool/Project	Open-Source Status	Paid Version/Commercial Support	Key Differences Between Free and Paid
Apache Hive	Fully open-source (Apache 2.0 License)	No official paid version, but third-party commercial support (e.g., Cloudera, Hortonworks)	Free version is fully functional; paid support offers enterprise services (e.g., technical support, optimization)
Apache Druid	Fully open-source (Apache 2.0 License)	Commercial versions available (e.g., Imply, CelerData)	Free version is fully functional; commercial versions offer enhanced features (e.g., management tools, support)
ClickHouse	Fully open-source (Apache 2.0 License)	Commercial version (ClickHouse Cloud)	Free version is fully functional; commercial version offers managed services, management, and support
Apache Kudu	Fully open-source (Apache 2.0 License)	No official paid version, but third-party commercial support (e.g., Cloudera)	Free version is fully functional; paid support offers enterprise services
Greenplum	Open-source (Apache 2.0 License)	Commercial version (VMware Tanzu Greenplum)	Free version is fully functional; commercial version offers advanced features and support
Apache Pinot	Fully open-source (Apache 2.0 License)	Commercial versions available (e.g., StarTree)	Free version is fully functional; commercial versions offer extended features and managed services
PostgreSQL	Fully open-source (PostgreSQL License)	Commercial support available (e.g., EnterpriseDB)	Free version is fully functional; commercial versions offer advanced features, support, and management tools
StarRocks	Partially open-source (Community Edition under Apache 2.0 License)	Commercial version (StarRocks Enterprise)	Community Edition has limited features; Enterprise Edition offers more features (e.g., data security, cluster management)

Apache Doris	Fully open-source (Apache 2.0 License)	Commercial support available (e.g., CelerData)	Free version is fully functional; commercial versions offer extended features and support
---------------------	---	---	--

W4

key components of a data warehouse stack: data ingestion, storage, processing, querying, and visualization.

Tech Stack:

1. Data Ingestion:

- **Apache Kafka** (streaming) + **Debezium** (CDC for PostgreSQL).
- **Telegraf** (IoT sensor data collection).

2. Processing:

- **Apache Flink** (real-time stream processing).
- **PostgreSQL PL/pgSQL** (stored procedures for lightweight transformations).

3. Storage:

- **TimescaleDB**
- **Apache IoTdb**
- **Influx db**

4. Query & Visualization:

- **Grafana** (time-series dashboards) or **Apache Superset** (general analytics).
- **Trino**

W5

some key points:

1. **Single Node vs. Cluster:** The open-source version of InfluxDB can run on a single node, making it suitable for small applications and development environments. However, in production environments, especially those requiring high availability and load balancing, users typically prefer a clustered setup.
2. **High Availability and Distributed Features:** The open-source version does not natively support high availability clustering. If you need high availability with the open-source version, you often have to manage replication and failover manually.

Data Processing Pipeline Architecture

1. Data Sources

- **IoT Sensors:** Devices such as temperature sensors, humidity sensors, and air quality monitors generate time-series data, critical for monitoring environmental conditions.
- **Logs or Monitoring Data:** Data generated by various devices are logged for analysis, which can include performance metrics, error logs, and operational data.

2. Data Ingestion

- **Telegraf:**
 - *Purpose:* A lightweight agent for collecting and sending metrics and events from IoT devices to various data stores, including IoTDB.
 - *Functionality:* Telegraf can be configured with different plugins for input (collecting data) and output (sending data). It supports a wide array of protocols and data sources, making it versatile for IoT scenarios.
- **Kafka:**
 - *Purpose:* A distributed event streaming platform capable of handling trillions of events a day, predominantly used for building real-time data pipelines and streaming applications.
 - *Functionality:* Kafka acts as a message queue, facilitating the transfer of data from IoT devices to storage systems, and supports protocols like MQTT for lightweight communication between sensors and data storage solutions.

3. Data Storage

- **InfluxDB:**
 - *Type:* Time-series database optimized for fast, high-availability storage and retrieval of time-series data.
 - *Features:*
 - Supports a SQL-like query language called InfluxQL for data queries.

- Can handle high write loads and is capable of compression and downsampling, which is crucial for time-series analytics.
- Ideal for applications requiring real-time analytics and visualization.
- **ClickHouse:**
 - *Type:* A high-performance columnar OLAP database management system designed for analytical workloads.
 - *Features:*
 - Supports real-time data ingestion and querying, ideal for analytical workloads involving complex queries.
 - Allows aggregating large datasets quickly, making it suitable for log data analysis and processing large volumes of data efficiently.

4. Data Processing

- **SQL Queries (IoTDB):**
 - *Purpose:* IoTDB's query language allows users to perform analytics on stored time-series data.
 - *Functionality:* Users can execute queries such as calculating averages, filtering data based on temperature thresholds, and fetching historical data for analysis.
- **Apache Flink:**
 - *Type:* A stream processing framework that enables real-time data processing and analysis.
 - *Features:*
 - Capable of processing data in a fault-tolerant way and supports stateful computations, making it powerful for complex event processing.
 - Integrates well with various data sources and sinks, including IoTDB, allowing users to perform calculations and aggregations on real-time data streams.

5. Data Analysis & Visualization

- **Grafana:**
 - *Purpose:* A popular open-source analytics and monitoring platform used for visualizing time-series data.
 - *Features:*
 - Provides powerful dashboards to visualize trends over time, displaying real-time metrics from IoTDB or other data sources.

- Supports various alerting mechanisms and plugins to enhance capability and integration with other data sources.
- **Apache Superset:**
 - *Purpose:* A modern data exploration and visualization platform that enables users to create interactive dashboards.
 - *Features:*
 - User-friendly interface for generating reports, and enables data exploration through SQL-based queries.
 - Supports a variety of data sources including IoTDB for complex data analysis and visualization.

6. Monitoring & Maintenance

- **Prometheus:**
 - *Purpose:* An open-source monitoring system and time-series database, particularly well-suited for recording real-time metrics.
 - *Features:*
 - Can monitor the health of data pipelines, alerting users to potential issues, and facilitating efficient resource management.
- **Apache Airflow:**
 - *Purpose:* An orchestration tool for managing workflows, allowing users to schedule and monitor complex data pipelines.
 - *Features:*
 - Useful for automating data ingestion, processing, and moving tasks, providing a clear representation of data pipelines through a web interface.
- **Trino:**
 - *Type:* A distributed SQL query engine designed for running interactive analytic queries across various data sources.
 - *Features:*
 - Allows queries spanning multiple databases, making it ideal for federated data analysis, and integration with existing data storage systems in an organization.

Steps in the Data Processing Pipeline Example

1. Data Collection Interface:

- Configure **Telegraf** plugins and parameters to collect data from IoT devices (e.g., temperature sensors).

2. Data Transmission:

- Use MQTT or Kafka to transfer data collected from devices to IoTDB.

3. Data Storage Configuration:

- Create an appropriate data model in IoTDB to store sensor data.
- For example, create a time-series to store device data: `root.building.sensor.temperature`.

4. Data Querying and Analysis:

- Use IoTDB's query language to execute SQL queries to retrieve specific time period sensor data.
- For example, to query the average temperature over the past hour:sql

Copy

```
SELECT AVG(temperature) FROM root.building.sensor.temperature WHERE time >=
'2023-01-01T10:00:00' AND time <= '2023-01-01T11:00:00';
```

5. Visualization:

- Connect Grafana to IoTDB to create time-series charts that track temperature changes in real time.

6. Monitoring and Feedback:

- Set up monitoring and alerting metrics, for instance, using Prometheus, to send alerts when the temperature exceeds a certain threshold.

W6

1. Core Requirements Analysis

Requirement	Details
Data Type	Structured time-series data (timestamp + sensor values)

Data Frequency	Low-frequency writes (every 10 minutes; e.g., 10 sensors per floor ⇒ ~600 data points per building per hour)
Data Characteristics	Multi-dimensional tagging (e.g., <code>BuildingA-5F-Room101-Temperature</code>), hierarchical query support
Deployment	Hybrid edge (local servers in buildings) + cloud storage
Long-Term Needs	Historical data compression, aggregation by floor/room, anomaly detection (e.g., temperature spikes)

2. InfluxDB vs. IoTDB Comparison

Feature	InfluxDB	Apache IoTDB
Write Load	Handles low-frequency writes, but underutilizes resources	Optimized for low/high-frequency hybrid scenarios, lower energy consumption
Data Modeling	Flat schema (simulate hierarchy via tags, e.g., <code>location=BuildingA/5F</code>)	Native tree structure (e.g., <code>root.BuildingA.5F.Room101.Temperature</code>)
Edge Deployment	Requires Telegraf, higher resource usage	Lightweight edge version (<100MB RAM), ideal for embedded devices
Query Efficiency	Strong time-range aggregation, weaker hierarchical filtering	Path-based queries (e.g., <code>root.BuildingA.5F.*.Temperature</code>)
Storage Cost	Default compression ratio ~3:1	TsFile compression up to 10:1, more cost-effective long-term
Alerting	Requires external tools (e.g., Kapacitor)	Built-in triggers for localized rule execution

3. Recommended Solution: Apache IoTDB

Key Advantages

1. Native Hierarchical Data Model

- Directly maps to building sensor hierarchy: `Building → Floor → Room → Sensor`.
- Example paths:

```
root.SmartBuilding.BuildingA.5F.ConferenceRoom101.Temperature
root.SmartBuilding.BuildingA.5F.Corridor.Noise
```

2. Edge-Cloud Synergy

- Deploy IoTDB edge nodes locally to aggregate data before syncing to the cloud, **reducing network dependency**.

- Edge version supports offline data recovery.

3. Storage Efficiency

- Example: 10,000 sensors × 144 daily entries = ~520M entries/year.
- IoTDB's TsFile reduces storage to ~10GB (50%+ savings vs. InfluxDB).

4. Industrial Protocol Support

- Built-in MQTT integration for direct sensor connectivity (e.g., Modbus-to-MQTT gateways).

5. Analytics Scalability

- Future analytics (e.g., "noise spikes on weekdays 9 AM–6 PM") via Spark/Flink integration.





5. Implementation Architecture

```
[Sensors] --(MQTT)--> [Edge IoTDB] --(Sync)--> [Cloud IoTDB Cluster]
|
| -- [Local Alerts] // e.g., temperature >30°C triggers notification
```

hierarchical modeling, edge efficiency, and storage savings.

ref

Criteria \ TSDBs	IoTDB	InfluxDB	OpenTSDB	KairosDB	TimescaleDB
Basic Features					
OpenSource	+	+	+	+	+
SQL-like	+	+	-	-	++
Schema	tree-based, tag-based	tag-based	tag-based	tag-based	relational
Writing out-of-order data	+	+	+	+	+
Schema-less	+	+	+	+	+
Batch insertion	+	+	+	+	+
Time range filter	+	+	+	+	+
Order by time	++	+	-	-	+
Value filter	+	+	-	-	+
Downsampling	++	+	+	+	+
Fill	++	+	+	-	+
LIMIT	+	+	+	+	+
SLIMIT	+	+	-	-	?
Latest value	++	+	+	-	+
Advanced Features					
Align by time	++	+	-	-	+
Compression	++	+-	+-	+-	+-
MQTT support	++	+	-	-	+-
Run on Edge-side Device	++	+	-	+-	+
Multi-instance Sync	++	-	-	-	-
JDBC Driver	+	-	-	-	-
Standard SQL	+	-	-	-	++
Spark integration	++	-	-	-	-
Hive integration	++	-	-	-	-
Writing data to NFS (HDFS)	++	-	+	-	-
Flink integration	++	-	-	-	-
Write and Read Performance					
Scalable Writes	++	+	?	+	+
Raw Data Query	++	+	?	+	+
Aggregation Query	++	+	?	+	+
Downsampling Query	++	+	?	+-	+-
Latest Query	++	+	?	+-	+

Legend	
++	 big support greatly
+	 support
+-	 support but not very good
-	 not support
?	unknown

W7

Clickhouse

Install through docker

docker pull yandex/clickhouse-server

run it

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\Hedy> docker run -d `
>> --name clickhouse-server `
>> -v clickhouse_data:/var/lib/clickhouse `
>> -p 8123:8123 `
>> -p 9000:9000 `
>> yandex/clickhouse-server
Unable to find image 'yandex/clickhouse-server:latest' locally
latest: Pulling from yandex/clickhouse-server
ea362f368469: Pull complete
38ba82a23e2b: Pull complete
9b17d04b6c62: Pull complete
5658714e4e8b: Pull complete
6bde977a0bf8: Pull complete
39053b27290b: Pull complete
762d3d237065: Pull complete
Digest: sha256:1cbf75aabe1e2cc9f62d1d9929c318a59ae552e2700e201db985b92a9bcabc6e
Status: Downloaded newer image for yandex/clickhouse-server:latest
ed31cf112fec52923f5adc12e5066cca007364cb22c54f97f6b73e5ef285a925
PS C:\Users\Hedy> docker exec -it clickhouse-server clickhouse-client
ClickHouse client version 22.1.3.7 (official build).
Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 22.1.3 revision 54455.

ed31cf112fec :) |
```

Connect to Clickhouse

`docker exec -it clickhouse-server clickhouse-client`

verify if it's runing

CREATE DATABASE test;

SHOW DATABASES;

```
Windows PowerShell

ed31cf112fec :) CREATE DATABASE test
CREATE DATABASE test
Query id: 477c304b-8ca8-4071-b7db-41daad4fc698
Ok.
0 rows in set. Elapsed: 0.007 sec.
ed31cf112fec :) SHOW DATABASES
SHOW DATABASES
Query id: 3022cc5c-735c-427c-bd7f-6d45189d1c2a
+-----+
| name |
+-----+
| INFORMATION_SCHEMA |
| default |
| information_schema |
| system |
| test |
+-----+
5 rows in set. Elapsed: 0.002 sec.
ed31cf112fec :) |
```

Create Table

```
Windows PowerShell

CREATE TABLE tmp_sound_levels
(
  `seq` UInt32,
  `timestamp` String,
  `a_weighted_sound_level` Float64,
  `a_weighted_equivalent_sound_level` Float64,
  `a_weighted_maximum_sound_level` Float64
)
ENGINE = MergeTree
ORDER BY seq
Query id: fd5a15f3-8c2e-4c53-b93e-bf064278b315
Ok.
0 rows in set. Elapsed: 0.033 sec.
ed31cf112fec :) select * from tmp_sound_levels
SELECT *
FROM tmp_sound_levels
Query id: 11322149-a8b2-4d12-bb6c-0cd1ea116d6c
Ok.
0 rows in set. Elapsed: 0.022 sec.
ed31cf112fec :) |
```

Import from csv

```
Windows PowerShell
seq timestamp a_weighted_sound_level a_weighted_equivalent_sound_level a_weighted_maximum_sound_level
1 2025/2/23 0:01 38 33.9 50.5
2 2025/2/23 0:51 46.1 40.9 56.9
3 2025/2/23 0:41 55 46.3 61.2
4 2025/2/23 0:31 53.8 46 57.6
5 2025/2/23 0:21 56.2 43.4 59.5
6 2025/2/23 0:11 47.1 35.8 56.1
7 2025/2/23 0:01 43.7 32.3 47.3
8 2025/2/23 0:51 42.5 33.1 55.2
9 2025/2/23 0:41 45.2 41.7 55.6
10 2025/2/23 0:31 54.3 44.9 56.2

10 rows in set. Elapsed: 0.020 sec. Processed 3.17 thousand rows, 160.48 KB (153.88 thousand rows/s., 7.79 MB/s.)

ed31cf112fec :) SELECT COUNT(*) FROM tmp_sound_levels;

SELECT COUNT(*)
FROM tmp_sound_levels

Query id: 9f34deab-ba5a-4f55-b445-385ed85ec6bf

count()
3171

1 rows in set. Elapsed: 0.067 sec.

ed31cf112fec :) |
```

```
Windows PowerShell

FROM tmp_sound_levels

Query id: 9f34deab-ba5a-4f55-b445-385ed85ec6bf

count()
3171

1 rows in set. Elapsed: 0.067 sec.

ed31cf112fec :) SELECT
    AVG(a_weighted_sound_level) AS avg_sound_level,
    MAX(a_weighted_maximum_sound_level) AS max_sound_level,
    MIN(a_weighted_equivalent_sound_level) AS min_equivalent_sound_level
FROM tmp_sound_levels;

SELECT
    AVG(a_weighted_sound_level) AS avg_sound_level,
    MAX(a_weighted_maximum_sound_level) AS max_sound_level,
    MIN(a_weighted_equivalent_sound_level) AS min_equivalent_sound_level
FROM tmp_sound_levels

Query id: 873e5bf8-dc74-4c32-8c28-6518b7bae34c

avg_sound_level max_sound_level min_equivalent_sound_level
48.3399243140965 94.5 28.7

1 rows in set. Elapsed: 0.145 sec. Processed 3.17 thousand rows, 76.10 KB (21.84 thousand rows/s., 524.25 KB/s.)
```

Doris

Installed


```
PS D:\Doris> docker-compose up -d
time="2025-03-25T23:43:44-03:00" level=warning msg="D:\\Doris\\docker-compose.yml: the attribute `version` is obsolete,
it will be ignored, please remove it to avoid potential confusion"
[+] Running 2/2
 ✓Container doris-fe-1   Started      1.3s
 ✓Container doris-be-1   Started      1.5s
PS D:\Doris> |
```

Pros and Cons of ClickHouse

Pros:

- **Extreme performance:** Optimized for large-scale data analytics with exceptional single-table query performance
- **High throughput:** Supports billions of rows per second ingestion

Cons:

- **Steeper Learning Curve:** May take time to get familiar with its unique query language features.
- **Complex Configuration:** Cluster setup can be intricate for new users.

Pros and Cons of Doris

Pros:

- **User-Friendly:** Simple SQL syntax makes it easy to learn, especially for MySQL users.
- **Concurrency:** Better concurrent query support compared to ClickHouse

Cons:

- **Relatively New Ecosystem:** Community and documentation resources are less extensive than ClickHouse.
- **Resource intensive:** Higher memory consumption