DATA+AI SUMMIT
BY databricks

APACHE XTABLE: INTEROPERABILITY BETWEEN OPEN TABLE FORMATS

Dipankar Mazumdar & Kyle Weller, Onehouse.ai
13th June 2024

SPEAKER BIOS

Dipankar Mazumdar

- Staff Developer Advocate @ Onehouse.ai
- Contributor @ Apache Hudi, XTable
- Prev: Data Architecture, Visualization, ML

in/dipankarmazumdar/

@dipankartnt

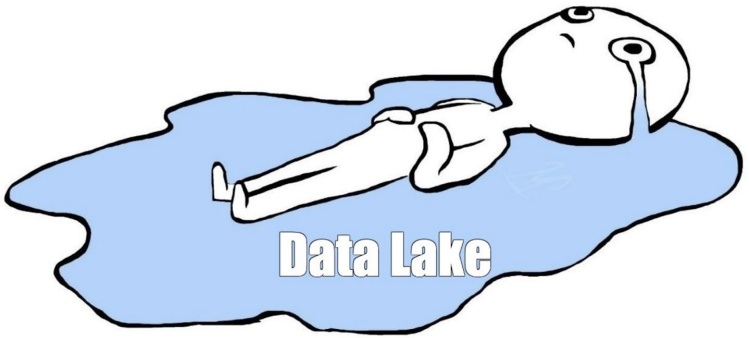
Kyle Weller

- Head of Product @ Onehouse.ai
- Prev: Product Manager - Azure Databricks, Azure ML, Bing Search

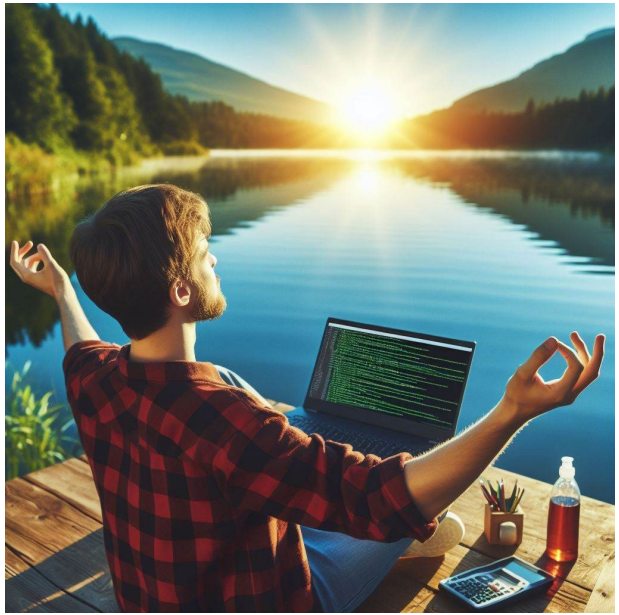
in/lakehouse/

@KyleJWeller

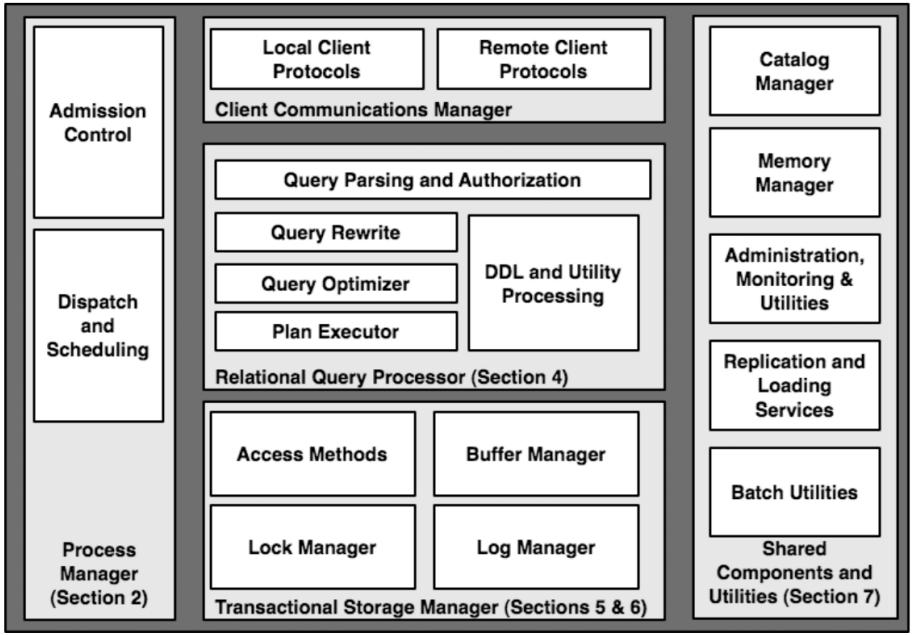
Data Lakes...



or

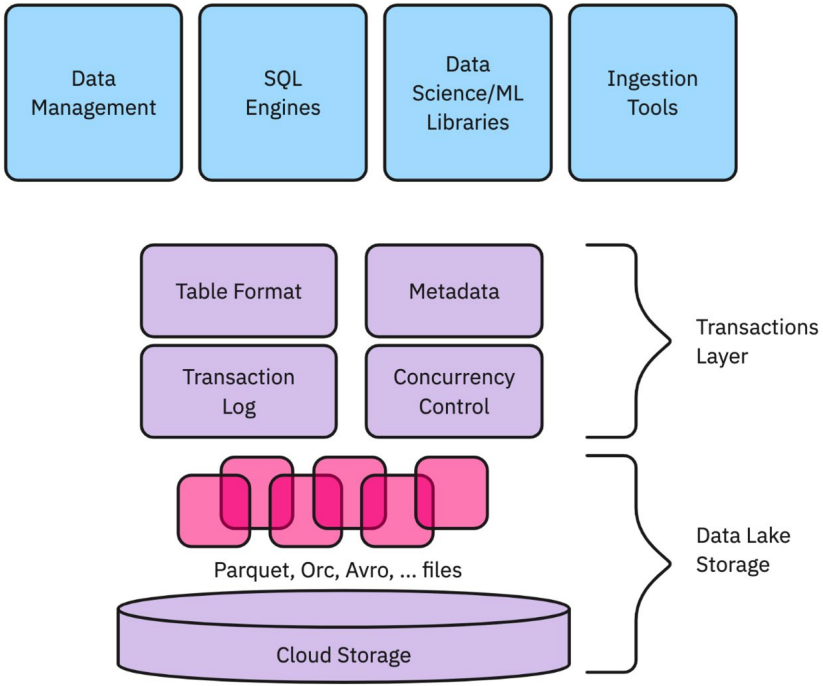


Architecture of a Database System



S3 Data Lake Storage

DATA LAKEHOUSE - UNBUNDLING OF THE DBMS



A lakehouse has the following key features:

- **Transaction support:** In an enterprise lakehouse many data pipelines will often be reading and writing data concurrently. Support for ACID transactions ensures consistency as multiple parties concurrently read or write data, typically using SQL.
- **Schema enforcement and governance:** The Lakehouse should have a way to support schema enforcement and evolution, supporting DW schema architectures such as star/snowflake-schemas. The system should be able to reason about data integrity, and it should have robust governance and auditing mechanisms.
- **BI support:** Lakehouses enable using BI tools directly on the source data. This reduce staleness and improves recency, reduces latency, and lowers the cost of having to operationalize two copies of the data in both a data lake and a warehouse.
- **Storage is decoupled from compute:** In practice this means storage and compute us separate clusters, thus these systems are able to scale to many more concurrent use and larger data sizes. Some modern data warehouses also have this property.
- **Openness:** The storage formats they use are open and standardized, such as Parquet and they provide an API so a variety of tools and engines, including machine learning and Python/R libraries, can efficiently access the data **directly**.
- **Support for diverse data types ranging from unstructured to structured data**: The lakehouse can be used to store, refine, analyze, and access data types needed for many new data applications, including images, video, audio, semi-structured data, an text.
- **Support for diverse workloads:** including data science, machine learning, and SQL an analytics. Multiple tools might be needed to support all these workloads but they all rely on the same data repository.
- **End-to-end streaming:** Real-time reports are the norm in many enterprises. Support for streaming eliminates the need for separate systems dedicated to serving real-tim data applications.

ORIGIN STORIES

2017 — open sourced — Uber

```
+ # Hudi
+ Hudi (pronounced Hoodie) stands for `Hadoop Upserts anD Incrementals`. Hudi manages storage of large analytical datasets on [HDFS](http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html) and serve them out via two types of tables

  * **Read Optimized Table** - Provides excellent query performance via purely columnar storage (e.g. [Parquet](https://parquet.apache.org/))
  * **Near-Real time Table (WIP)** - Provides queries on real-time data, using a combination of columnar & row based storage (e.g Parquet + [Avro](http://avro.apache.org/docs/current/mr.html))
```

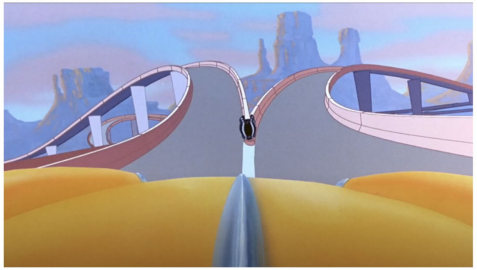
2018 — open sourced — N

```
+ ## Iceberg
+
+ Iceberg is a new table format for storing large, slow-moving tabular data. It is designed to improve on the de-facto standard table layout built into Hive, Presto, and Spark.
+
```

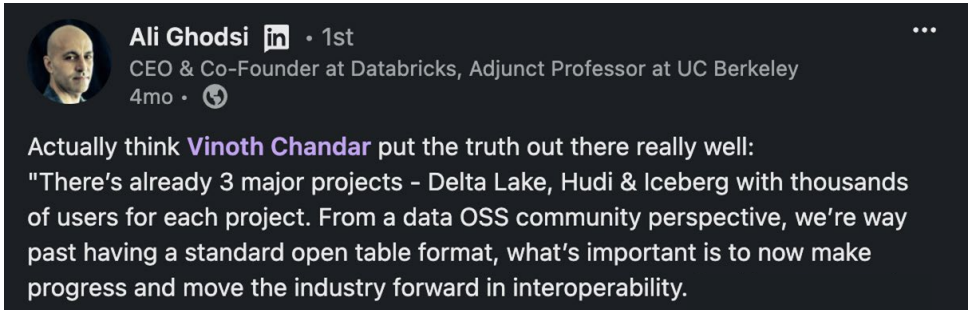
2019 — open sourced

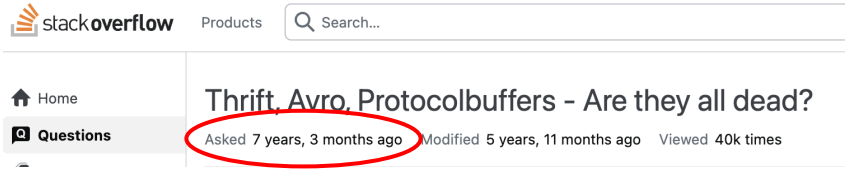
```
- Delta Lake Core is .... (copy text from delta docs)
+ Delta Lake is a next-generation engine built on top of Apache Spark. Delta Lake provides ACID transactions, optimized layouts and indexes, and execution engine improvements for building data pipelines to support big data use cases: batch and streaming ingests, fast interactive queries, and machine learning. Specifically, Delta offers:
```

🌶 (/ 4 4! +%| 4B? S ; L? >CP? LA? HNV



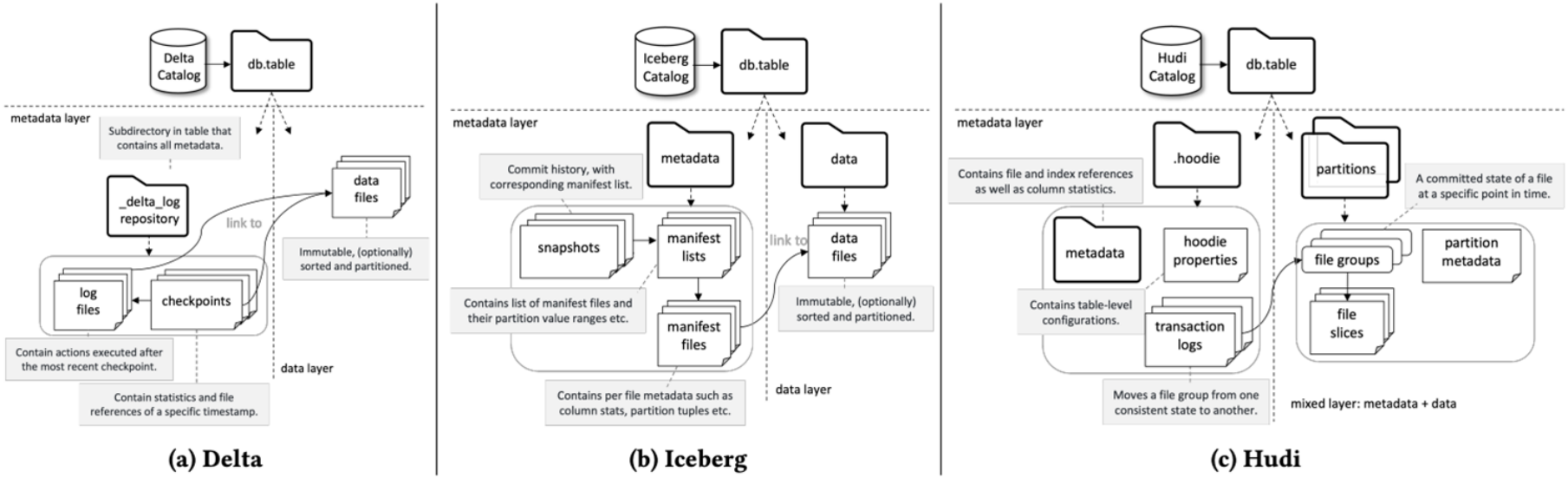
- Technical vision and goals are divergent
- The community needs are specialized
- All three projects are on fast growth trajectories
- New table formats are gaining traction: Apache Paimon, YOHB?





TECHNICAL FUNDAMENTALS

- Metadata abstractions on files in cloud object storage
- Tables with SQL semantics and schema evolution
- ACID transactions
- Updates and deletes (merge/upsert)
- Data layout optimizations for performance tuning



HOW IT LOOKS ON CLOUD STORAGE

- Fundamentals of table formats Hudi, Delta, Iceberg are not that different
- Each has a special metadata layer on top of parquet files

Apache hudi

```
s3_bucket/my_table/
 |- .hoodie/
 |     |- hoodie.properties
 |     |- metadata/
 |- file_1.parquet
 |- file_2.parquet
 |- file_N.parquet
```

DELTA LAKE

```
s3_bucket/my_table/
 |- _delta_log/
 |     |- 000000.json
 |- file_1.parquet
 |- file_2.parquet
 |- file_N.parquet
```

ICEBERG

```
s3_bucket/my_table/
 |- metadata/
 |     |- v1.metadata.json
 |     |- snap-9fa1-2-16c3.avro
 |     |- 0d9a-98fa-77.avro
 |- file_1.parquet
 |- file_2.parquet
 |- file_N.parquet
```

Amazon S3 | Google Cloud Storage | Azure Data Lake Storage | HDFS

WHICH FORMAT SHOULD I CHOOSE?

Choose Apache Hudi if:

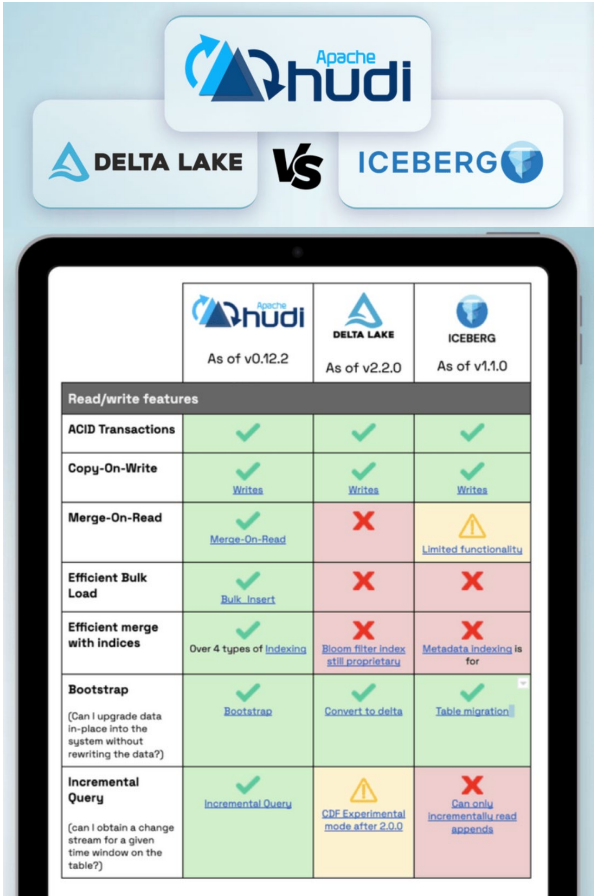
1. Mutable data - GDPR Deletes, Updates
2. CDC workloads
3. Low latency requirements
4. Large ETL pipelines - perf/cost w/ incremental ETL

Choose DELTA LAKE if:

1. Best Databricks experience
2. Needs fastest premium Spark with Photon
3. Wants an “easy-to-get-started” table format

Choose ICEBERG if:

1. Trino or Athena writes
2. Snowflake writes
3. Not sensitive to performance
4. Partition evolution



	Apache Hudi As of v0.12.2	DELTA LAKE As of v2.2.0	ICEBERG As of v1.1.0
Read/write features			
ACID Transactions	✓	✓	✓
Copy-On-Write	✓ Writes	✓ Writes	✓ Writes
Merge-On-Read	✓ Merge-On-Read	✗	⚠ Limited functionality
Efficient Bulk Load	✓ Bulk_Insert	✗	✗
Efficient merge with indices	✓ Over 4 types of Indexing	✗ Bloom filter index still proprietary	✗ Metadata indexing is for
Bootstrap (Can I upgrade data in-place into the system without rewriting the data?)	✓ Bootstrap	✓ Convert to delta	✓ Table migration
Incremental Query (can I obtain a change stream for a given time window on the table?)	✓ Incremental Query	⚠ CDF Experimental mode after 2.0.0	✗ Can only incrementally read appends

WHICH FORMAT SHOULD I CHOOSE?

Choos Apache hudi **if:**

1. Mutable data - GDPR Deletes, U tes
2. DC workloads
3. L latency requirements
4. Larg TL pipelines - perf/cost w/ incre ental ETL

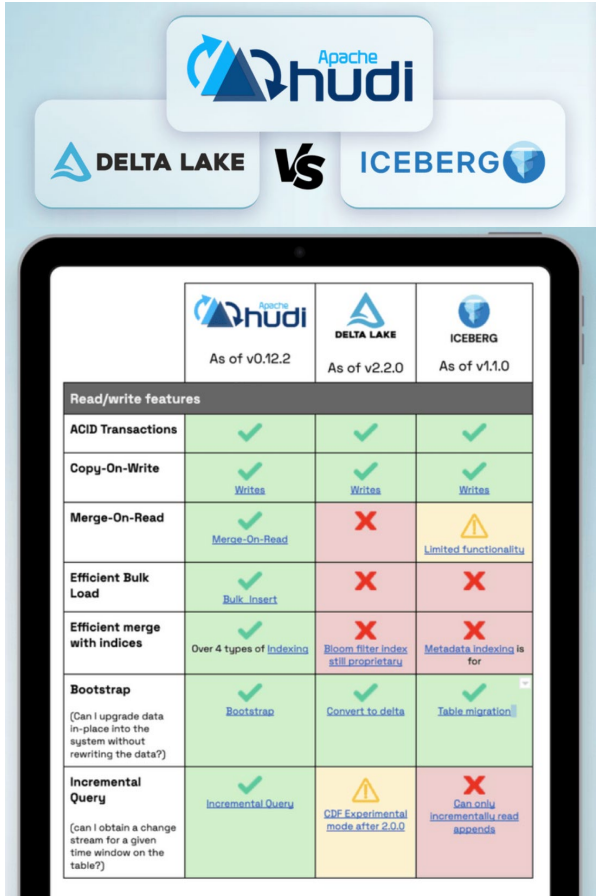
Choose DELTA **if:**

What if you could work across all 3?

2. Needs fastest pre in Spark with Photo
3. Wants an "easy-to-g started" table fo at

Choose ICEBERG **if:**

1. Trino or Athena writes
2. Snowflake writes
3. ensitive to performan
4. Parti



	Apache hudi As of v0.12.2	DELTA LAKE As of v2.2.0	ICEBERG As of v1.1.0
Read/write features			
ACID Transactions	✓	✓	✓
Copy-On-Write	✓ Writes	✓ Writes	✓ Writes
Merge-On-Read	✓ Merge-On-Read	✗	⚠ Limited functionality
Efficient Bulk Load	✓ Bulk Insert	✗	✗
Efficient merge with indices	✓ Over 4 types of Indexing	✗ Bloom filter index still proprietary	✗ Metadata indexing is for
Bootstrap (Can I upgrade data in-place into the system without rewriting the data?)	✓ Bootstrap	✓ Convert to delta	✓ Table migration
Incremental Query (can I obtain a change stream for a given time window on the table?)	✓ Incremental Query	⚠ CDF Experimental mode after 2.0.0	✗ Can only incrementally read appends

EXAMPLE BENEFITS OF MIX-AND-MATCH

Writing

Choose Apache Hudi writing w/ EMR (Spark)

1. Fastest writes for mutable workloads
2. Most flexible tuning parameters for ingestion

Choose DELTA LAKE writing w/ Fabric:

1. Easy-to-get-started out of the box
2. Makes data available to the entire Azure portfolio

Choose ICEBERG writing w/ BigQuery

1. Only table format supported for writes
2. Partition evolution

Reading

Choose DELTA LAKE reading w/ Databricks

1. Get fastest queries with Photon acceleration
2. Great experience for Data Science

Choose ICEBERG reading w/ Snowflake

1. Only supported table format in Snowflake
2. Decouple data storage using external tables

Choose Apache Hudi reading w/ DataProc (Spark)

1. Fast record level indexes for point queries
2. Powerful secondary indexing capabilities for Spark

INTRODUCING Apache XTable™ Incubating

⭐Celebrate by adding a little star ⭐

https://github.com/apache/incubator-xtable

apache / incubator-xtable

Type / to search

Code | Issues 55 | Pull requests 16 | Discussions | Actions | Projects | Security | Insights

incubator-xtable Public

Edit Pins | Watch 23 | Fork 82 | Starred 591

main | 19 Branches | 1 Tags | Go to file t | + | Code

wuchunfu and jcamachor Add NOTICE c09cce5 · 3 hours ago 188 Commits

.azure-pipelines/workflows	Enforce ASF license header in non-java file...	4 days ago
.github	Enforce ASF license header in non-java file...	4 days ago
.mvn	Enforce ASF license header in non-java file...	4 days ago
api	Enforce ASF license header in non-java file...	4 days ago
assets/images	Updated folder name to avoid ambiguity (#...	4 months ago
	Enforce ASF license header in non-java file...	4 days ago

About

Apache XTable (incubating) is a cross-table converter for lakehouse table formats that facilitates interoperability across data processing systems and query engines.

xtable.apache.org/

apache-iceberg | delta-lake | apache-hudi

Readme

Apache-2.0 license

>600 GH Stars ⭐

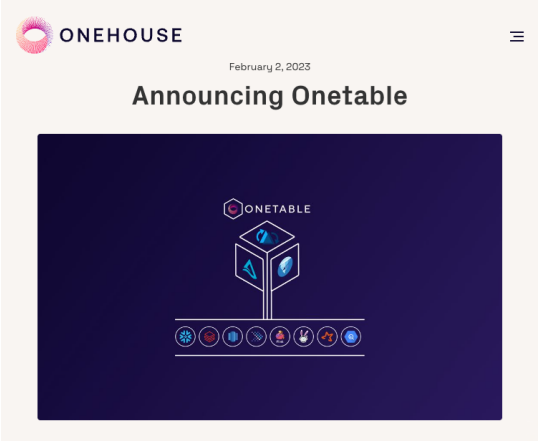
⌐] U
&I LEM

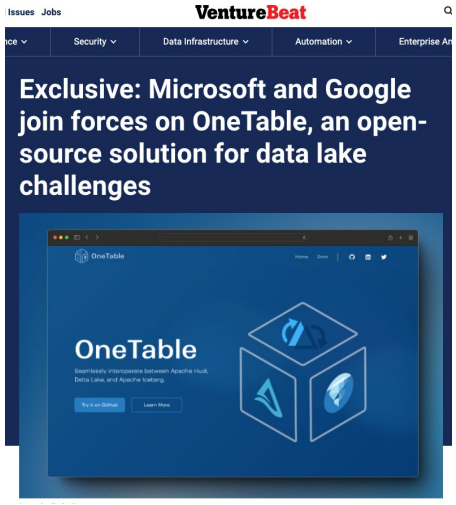
OneTable
@OnetableOSS

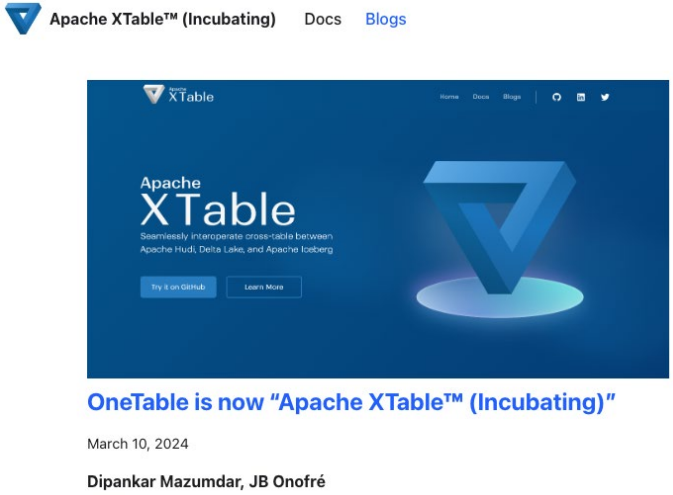
Today we made it on the TOP PAGE of Github Repos Trending worldwide: github.com/trending 🚀

#OneTable #apachehudi #apacheiceberg #deltalake

Apache XTable™ TIMELINE







Onehouse announces OneTable

OSS Co-Launch with Microsoft, Google, Onehouse

Donation to ASF and incubation as Apache XTable

Feb 2023 → **Nov 2023** → **Mar 2024**

ONEHOUSE

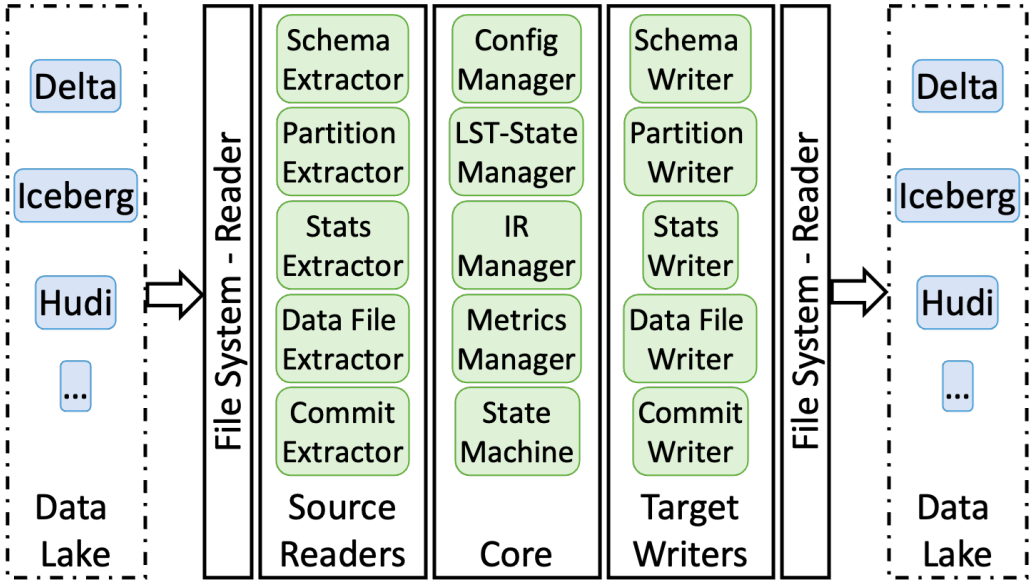
Microsoft Google ONEHOUSE

Microsoft CLOUDERA Walmart ONEHOUSE dremio Adobe Google

XTABLE ARCHITECTURE

Components:

1. Source Reader
1. Target Writer
1. Core Logic

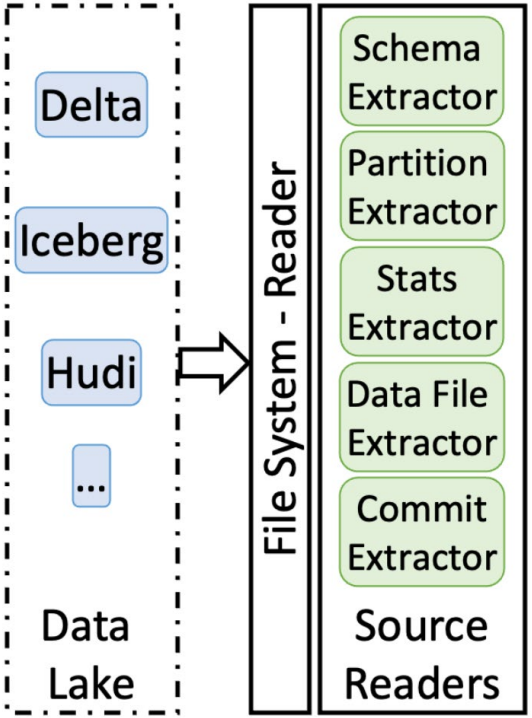


XTABLE ARCHITECTURE

Components:

1. Source Reader:

- LST-specific modules responsible for reading metadata from source table
- Operates using a pluggable file system
- Extracts info (schema, partition, transactions) & translates into XTable's internal representation



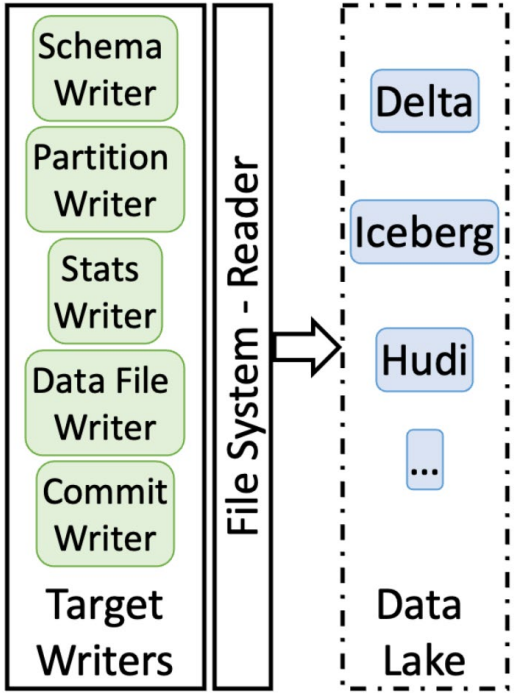
XTABLE ARCHITECTURE

Components:

2. Target Writer:

- takes the internal representation of metadata & accurately maps it to the target format's metadata structure

- includes re-creating schema, transaction logs & partition details in the new format



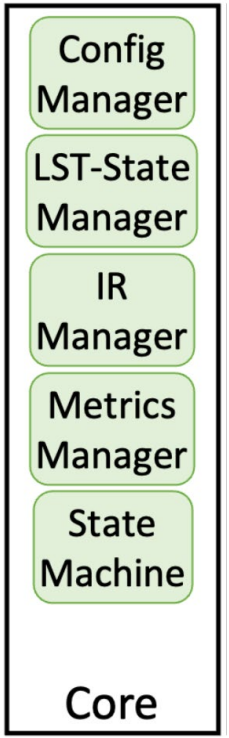
XTABLE ARCHITECTURE

Components:

3. Core Logic:

- CPU of XTable

- orchestrates the entire translation process
  - initialization of all components
  - managing sources/targets,
  - Handling tasks like caching for efficiency, state management for recovery & incremental processing, and telemetry for monitoring



HOW IT WORKS?

1: Choose your “source” format
2: Choose your “target” format(s)
3: XTable translates the metadata layers

Read your table as any of the formats

Hudi | Delta | Iceberg

yaml

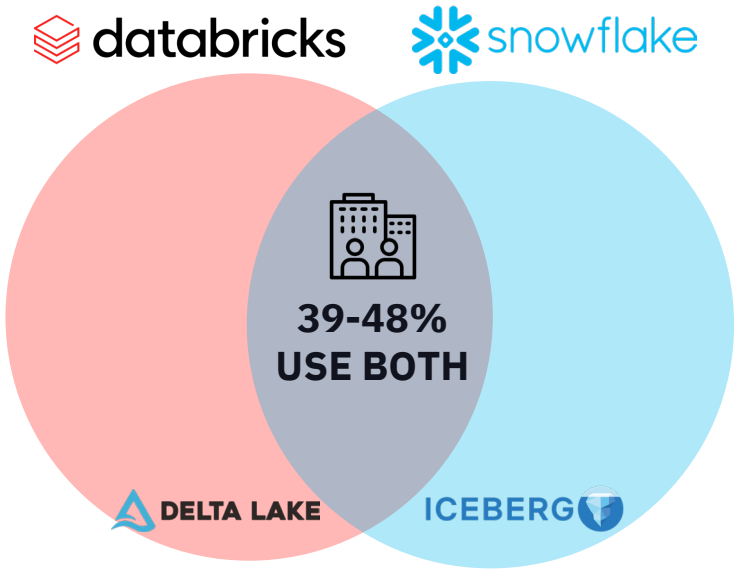
```yaml
sourceFormat: HUDI
targetFormats:
  - DELTA
  - ICEBERG
datasets:
  -
    tableBasePath: s3://path/to/hudi-dataset/people  # replace this with gs://path/to/hudi-dataset/people if yo
    tableName: people
    partitionSpec: city:VALUE
```

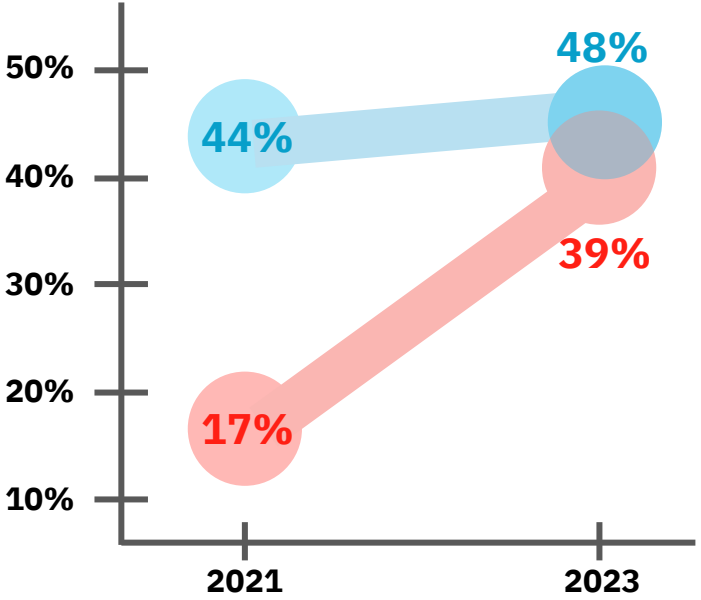
```
s3_bucket/my_table/
 |- .hoodie/
 |     |- hoodie.properties
 |     |- metadata/
 |- file_1.parquet
 |- file_2.parquet
 |- file_N.parquet
```

```
s3_bucket/my_table/
 |- .hoodie/
 |     |- hoodie.properties
 |     |- metadata/
 |- _delta_log/
 |     |- 000000.json
 |- metadata/
 |     |- v1.metadata.json
 |     |- snap-9fa1-2-16c3.avro
 |     |- 0d9a-98fa-77.avro
 |- file_1.parquet
 |- file_2.parquet
 |- file_N.parquet
```

```
# any of these work on the same table
spark.read.format("hudi")
spark.read.format("delta")
spark.read.format("iceberg")
```

A TALE OF TWO…

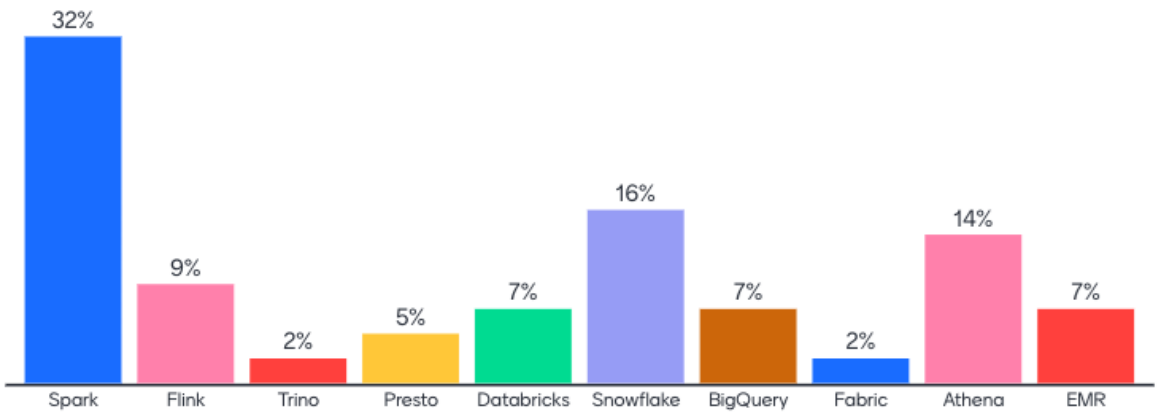

databricks
snowflake
39-48%
USE BOTH
DELTA LAKE
ICEBERG


Overlap Growth 2021-2023
50%
40%
30%
20%
10%
44%
48%
17%
39%
2021
2023

Join at menti.com | use code 5780 3382

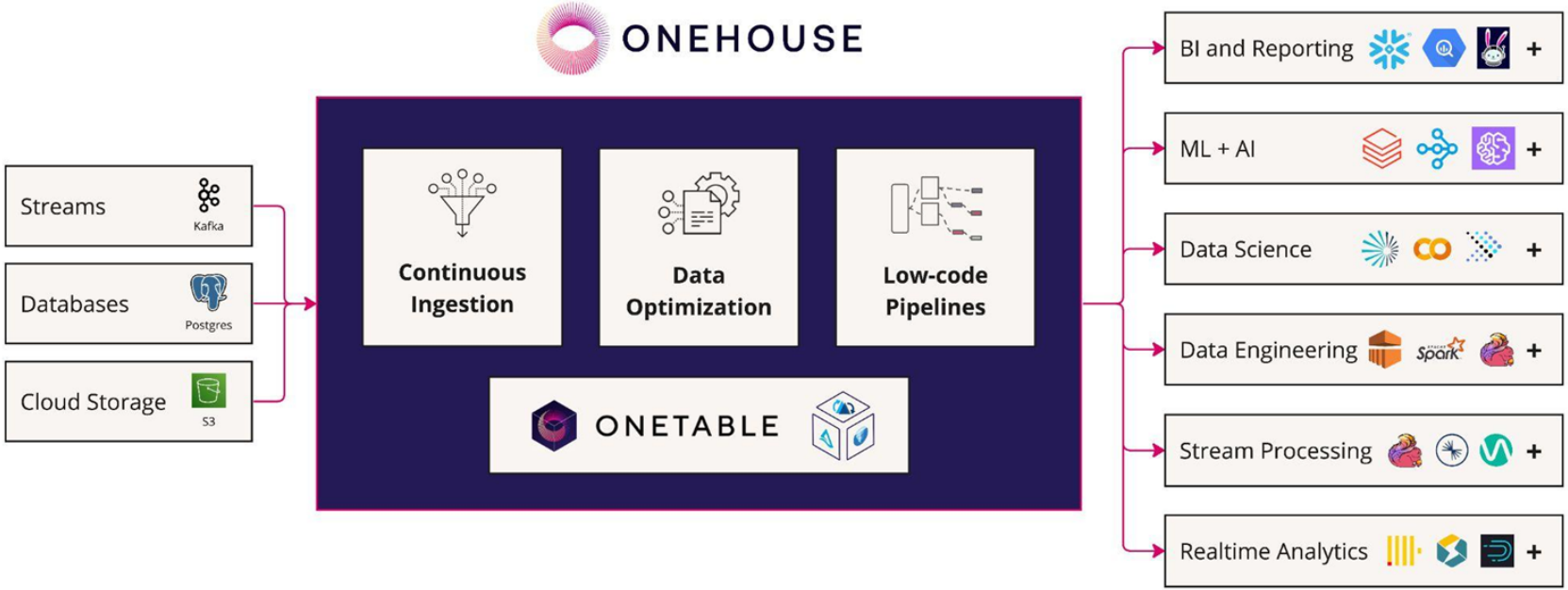
Mentimeter

Which Query Engines Do You Use?



Apache XTable™ WHY BUILD IT?

Whitepaper



Our Goal = Universal Data Lakehouse

Demo Time!

VISION FOR THE FUTURE

Goals

Seamless and efficient interoperability
Eliminate data silos
Project sustainability and evolution

Features

Real-time and transparent replication in any direction
Accurate and lossless model
Extensibility and flexibility

Community

Neutral and inclusive: Vendors, Cloud providers, Users
Graduate ASF Incubation

JOIN THE COMMUNITY

Initial Committers

- Tim Brown : *Onehouse*
- Vinish Reddy: *Onehouse*
- Ashvin Agrawal : *Microsoft*
- Jesus Camacho Rodriguez : *Microsoft*
- Anoop Johnson : *Google*
- Stamatis Zampetakis : *Cloudera*
- Hitesh Shah : *Adobe*
- Jean-Baptiste Onofré : *Dremio*
- Baljinder Singh : *Walmart*
- Vamshi Gudavarthi : *Onehouse*
- Vinoth Chandar: *Onehouse*

Contributors, Users, Mentors

Microsoft
Google
dremio
ONEHOUSE
snowflake
databricks
Walmart
Adobe
CLOUDERA

APACHE INCUBATOR

Apache XTable™

ROADMAP

Current Status

- Supported formats: ***Apache Hudi, Apache Iceberg, and Delta Lake***
- Tested with: Apache Spark, Trino, Microsoft Fabric, Databricks, BigQuery, Snowflake, Redshift, and more
- Features: on-demand incremental conversion, copy-on-write, catalog integration, change-history

Roadmap (6-12 months)

- ***Merge-on-Read*** (delete vectors)
- Apache Paimon (incubating)
- ***Performance***, ***efficiency***, and resiliency
- Deployment: as-a-service and in-memory
- Native engine integration

Roadmap (long term)

- Multi-writer (duplex)
- Synchronized commit timestamp
- Feature parity (superset)
- New technology stack
- Support new formats & versions
- Data Sharing
- Catalog

Apache XTable™ - LET'S BUILD TOGETHER

Github: https://github.com/apache/incubator-xtable

Docs : https://xtable.apache.org/docs/how-to

Twitter : https://twitter.com/apachextable

LinkedIn : https://www.linkedin.com/company/apache-xtable/

Mailing List : dev-subscribe@xtable.apache.org

Thank You!