DATA+AI
SUMMIT 2022

From 24h to 15min

How Robinhood Built a Streaming Lake House to Bring Data Freshness from 24h to <15min

Balaji Varadarajan
Senior Staff Engineer, Robinhood Markets
Apache Hudi PMC

Vikrant Goel
Engineering Manager, Robinhood Markets

ORGANIZED BY databricks

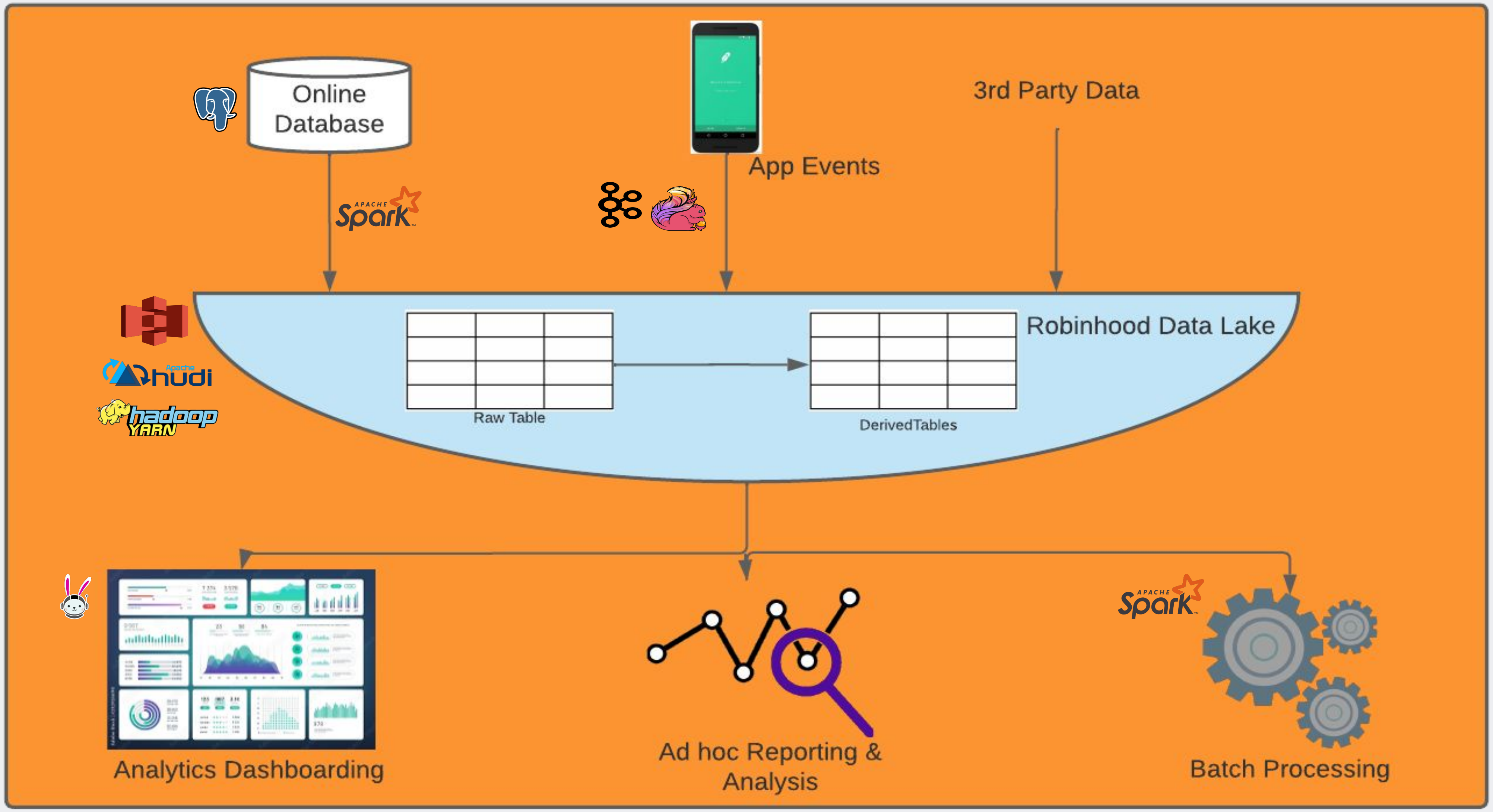
Agenda

- Legacy Data Lake and Ingestion Framework

- Deep Dive - Change Data Capture (CDC)
    - Design
    - Lessons Learned

- Deep Dive - Data Lakehouse Ingestion
    - Apache Hudi
    - End to End Setup

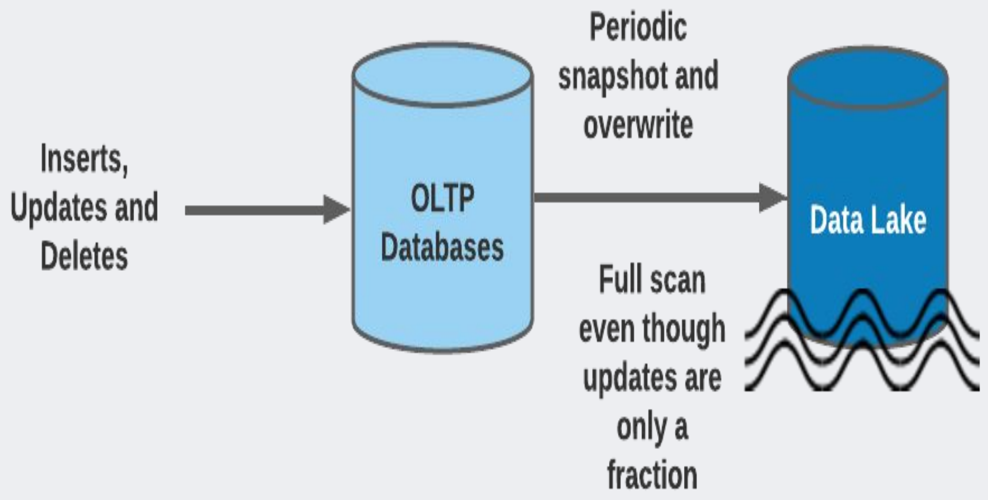
Data Lake



Online Database
App Events
3rd Party Data
Robinhood Data Lake
Raw Table
DerivedTables
Analytics Dashboarding
Ad hoc Reporting & Analysis
Batch Processing

Daily Snapshots

- Daily snapshotting of tables in RDBMS (RDS)
- High Read & Write amplifications
- Dedicated Replicas to isolate snapshot queries
- Bottlenecked by Replica I/O
- 24+ hours data latency

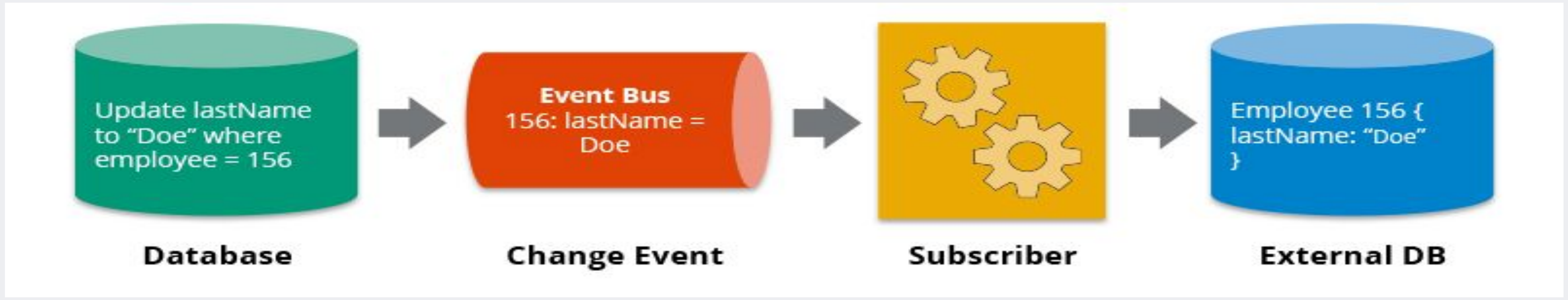

Inserts, Updates and Deletes
OLTP Databases
Periodic snapshot and overwrite
Data Lake
Full scan even though updates are only a fraction

Need Faster Intraday Ingestion Pipeline

$$\Big\downarrow$$

Unlock Data Lake for business critical applications

Change Data Capture


Update lastName to "Doe" where employee = 156
Event Bus 156: lastName = Doe
Employee 156 { lastName: "Doe" }
Database
Change Event
Subscriber
External DB

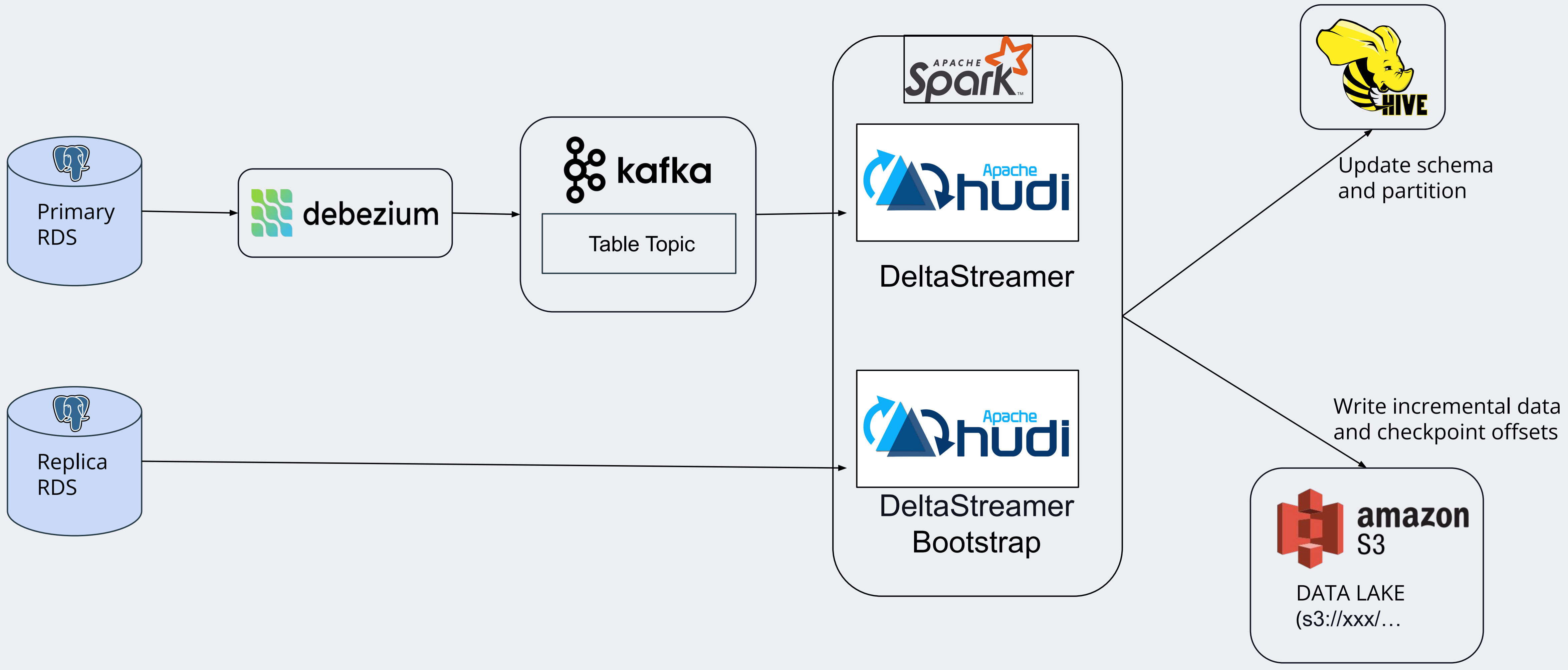
- Each CRUD operation streamed from DB to Subscriber
- Merge changes to lake house
- Efficient & Fast -> *Capture and Apply only deltas*

Deep Dive
CDC using Debezium

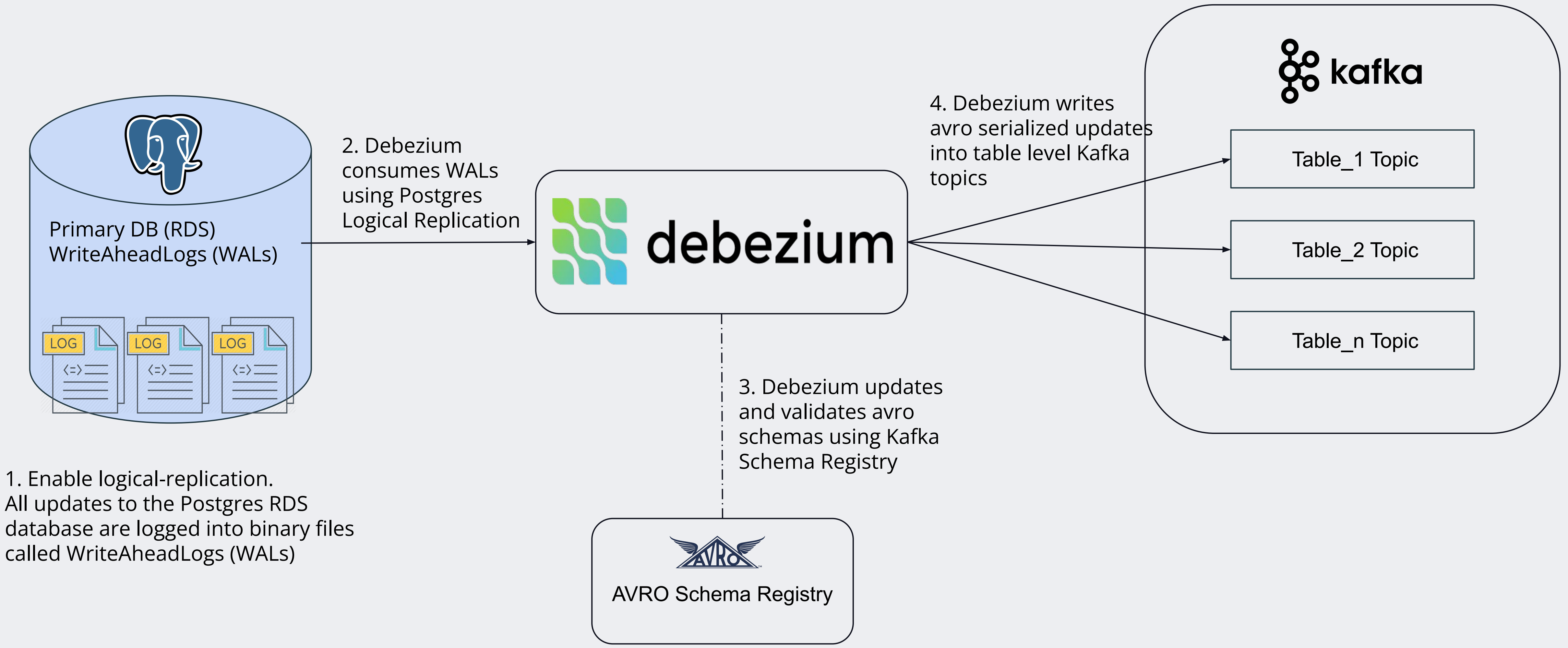
Debezium

- Open source & distributed Kafka-Connect Service for change data capture
- Support CDC from diverse RDBMS (Postgres, MySQL, MongoDB, etc.)
- Pluggable Sinks through Kafka

High Level Architecture



Debezium - Zooming In



Primary DB (RDS)
WriteAheadLogs (WALs)
LOG
LOG
LOG
2. Debezium consumes WALs using Postgres Logical Replication
debezium
4. Debezium writes avro serialized updates into table level Kafka topics
kafka
Table_1 Topic
Table_2 Topic
Table_n Topic
3. Debezium updates and validates avro schemas using Kafka Schema Registry
AVRO
AVRO Schema Registry

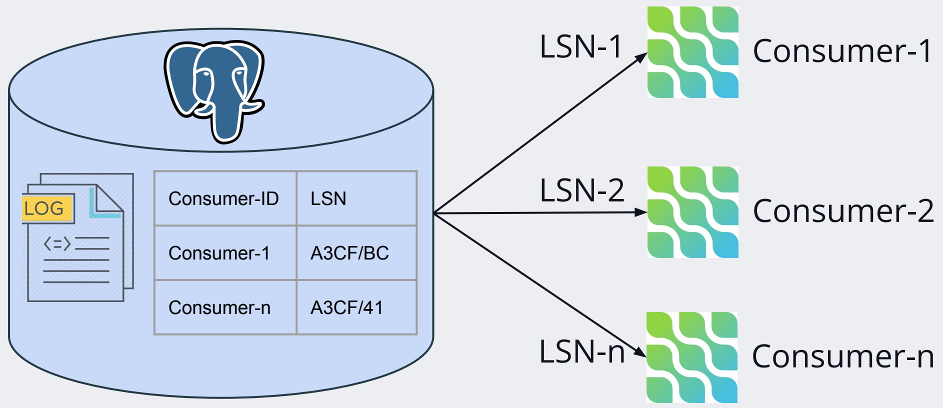
1. Enable logical-replication.
All updates to the Postgres RDS database are logged into binary files called WriteAheadLogs (WALs)

DATA+AI SUMMIT 2022

Why did we choose Debezium over alternatives?

	Debezium	AWS Database Migration Service (DMS)
Operational Overhead	High	Low
Cost	Free, with engineering time cost	Relatively expensive, with negligible engineering time cost
Speed	**High**	**Not enough**
Customizations	Yes	No
Community Support	Debezium has a very active and helpful Gitter community.	Limited to AWS support.

Debezium: Lessons Learned



Consumer-ID	LSN
Consumer-1	A3CF/BC
Consumer-n	A3CF/41

Postgres Primary:
- Publishes WALs
- Record LogSequenceNumber (LSN) for each consumer

1. Postgres Primary Dependency

ONLY the Postgres Primary publishes WriteAheadLogs (WALs).

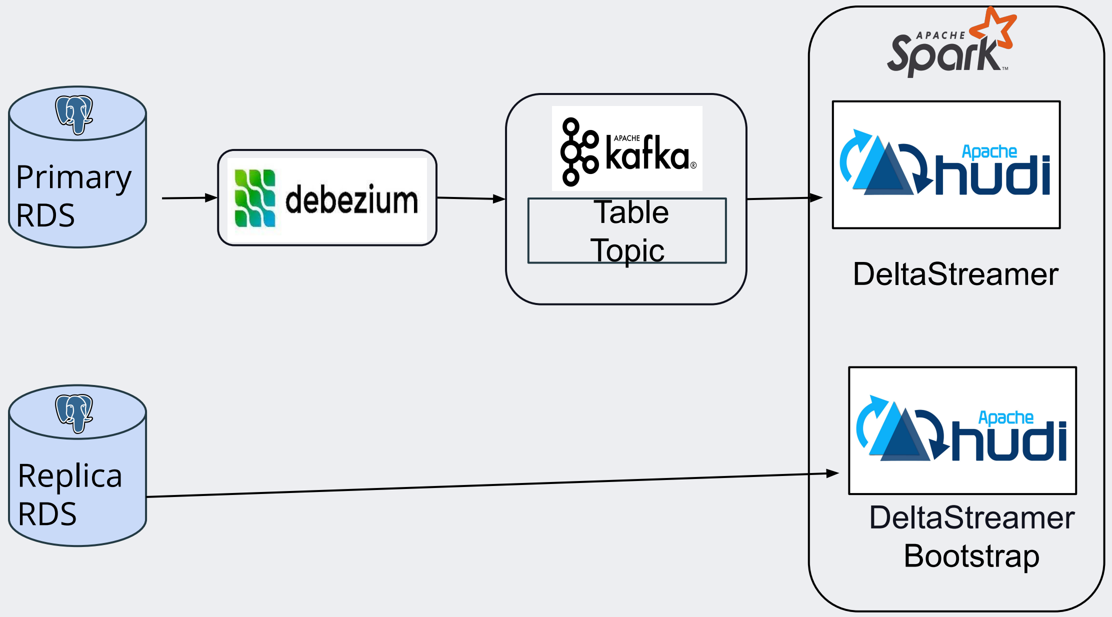
Disk Space:

- If a consumer dies, Postgres will keep accumulating WALs to ensure Data Consistency
- Can eat up all the disk space
- Need proper monitoring and alerting

CPU:

- Each logical replication consumer uses a small amount of CPU
- Postgres10+ uses pgoutput (built-in) : Lightweight
  Postgres9 uses wal2Json (3rd party) : Heavier
- Need upgrades to Postgres10+

Debezium: Lessons Learned



2. Initial Table Snapshot (Bootstrapping)

Need for bootstrapping:

- Each table to replicate requires initial snapshot, on top of which ongoing logical updates are applied

Problem with Debezium:

- Debezium processes data row at row level
- Large tables are slow
- Too much pressure on Kafka Infrastructure and Postgres primary

Solution using Hudi Deltastreamer:

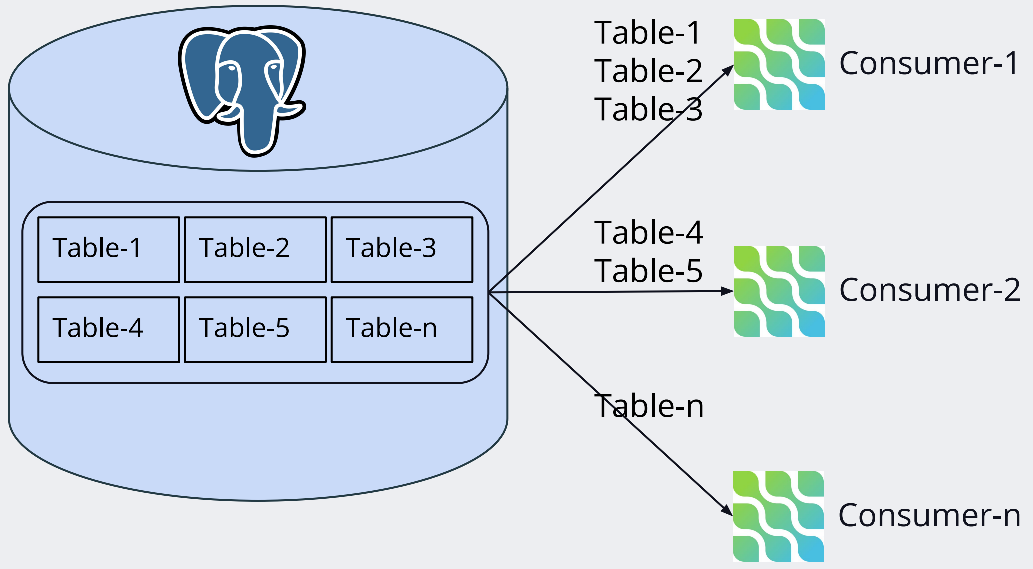
- Custom bootstrapping framework using partitioned and distributed spark reads
- Can use read-replicas instead of the master

Debezium: Lessons Learned

3. AVRO vs JSON

	AVRO	JSON	JSON + Schema
Throughput (Benchmarked using db.r5.24xlarge Postgres RDS instance)	Up to **40K mps**	Up to **11K mps.** JSON records are larger than AVRO.	Up to **3K mps.** Schema adds considerable size to JSON records.
Data Types	- Supports considerably high number of primitive and complex data types out of the box. - Great for type safety.	Values must be one of these 6 data types: - String - Number - JSON object - Array - Boolean - Null	Same as JSON
Schema Registry	Required by clients to deserialize the data.	Optional	Optional

Debezium: Lessons Learned



4. Multiple logical replication streams for horizontal scaling

- Multiple large tables can overwhelm a single Debezium connector
- Split the tables across multiple Debezium connectors
  Total throughput = throughput_per_connector * num_connectors
- Each connector does have small CPU cost

DATA+AI SUMMIT 2022

Debezium: Lessons Learned

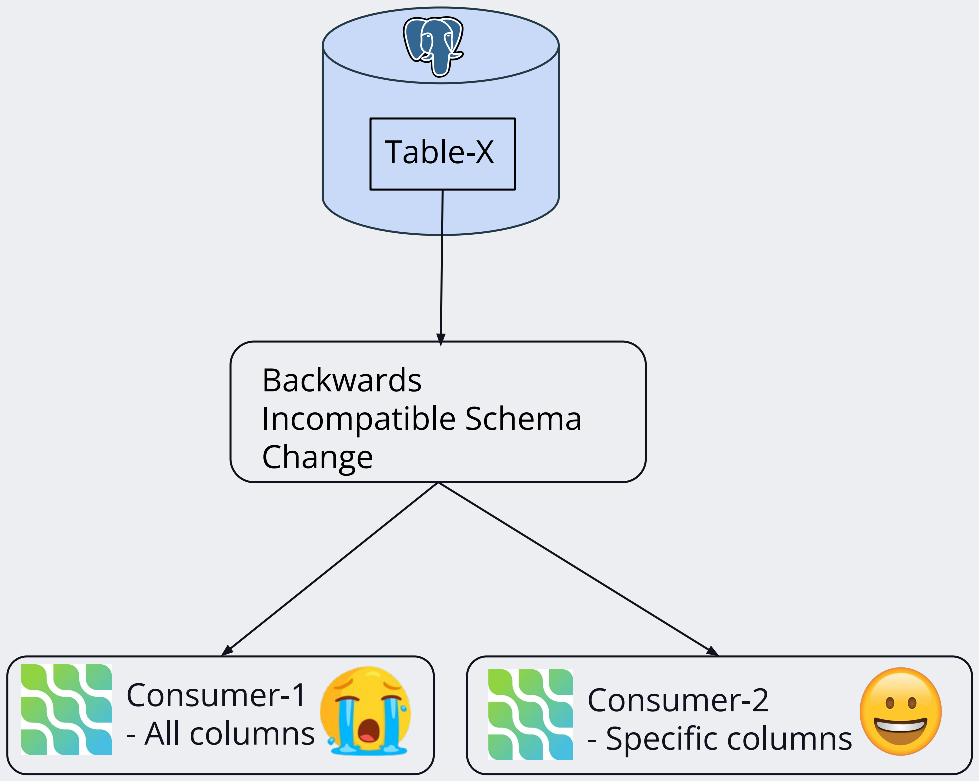


Table-X
Backwards Incompatible Schema Change
Consumer-1 - All columns
Consumer-2 - Specific columns

5. Schema evolution and value of Freezing Schemas

Failed assumption: Schema changes are infrequent and always backwards compatible.

- Examples:
  1. Adding non-nullable columns (Most Common 99/100)
  2. Deleting columns
  3. Changing data types
  4. Can happen anytime during the day #always_on_call

How to handle the non backwards compatible changes?

- Re-bootstrap the table

Alternatives? Freeze the schema

- Debezium allows to specify the list of columns per table.
- Pros:
  - #not_always_on_call
  - Batch the changes for management window
- Cons:
  - Schema is temporarily out of sync

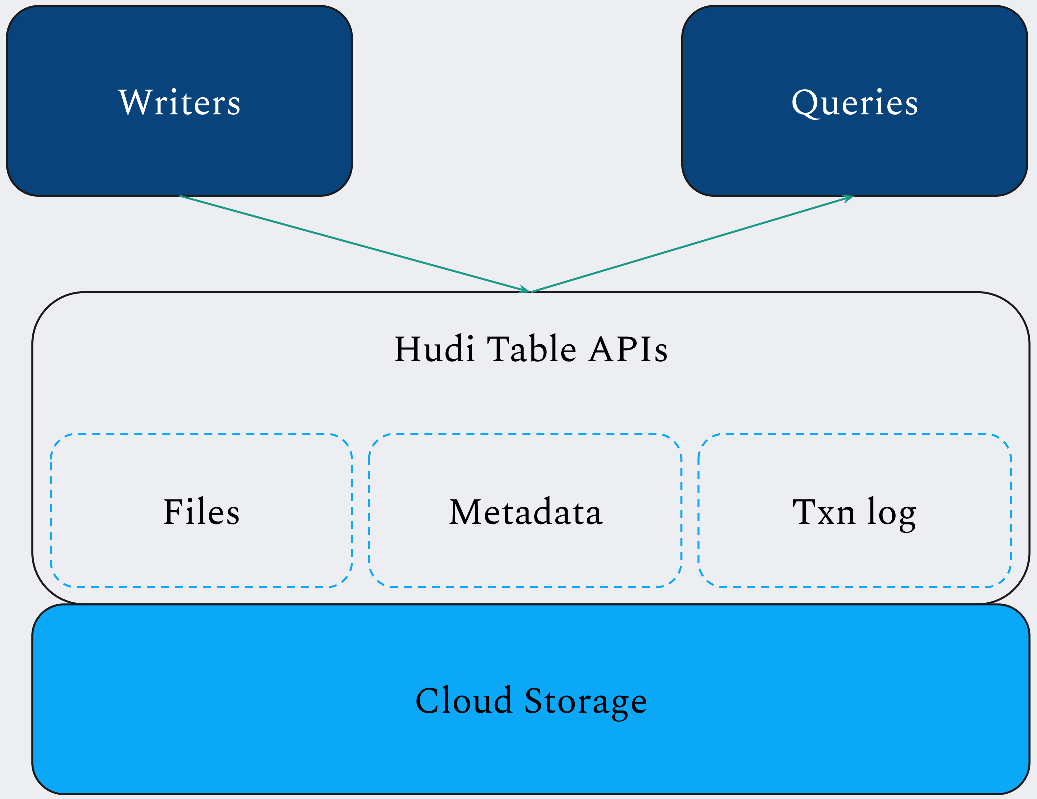
Deep Dive Lakehouse

Lakehouse - Requirements

- Transaction support
- Scalable Storage and compute
- Openness
- Direct access to files
- End-to-end streaming
- Diverse use cases

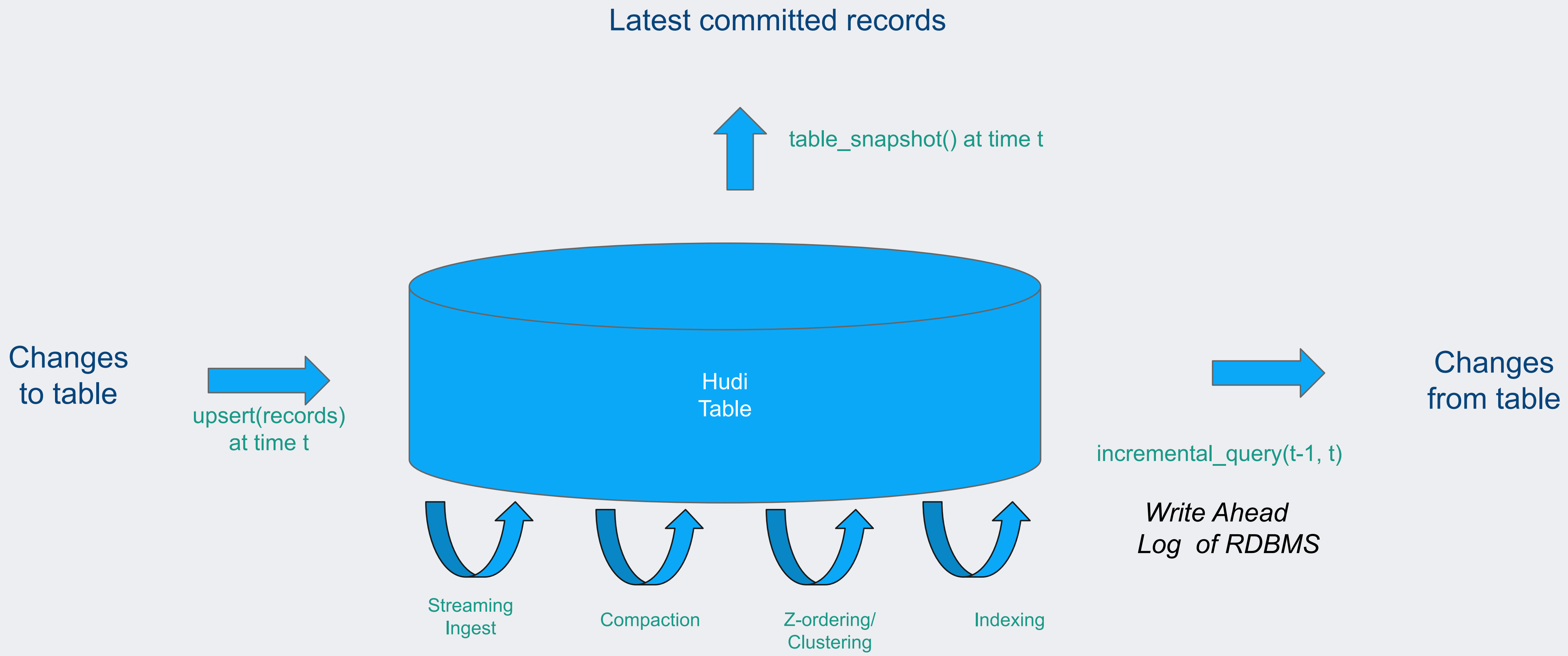
Apache Hudi - Introduction

- Transactional Lakehouse pioneered by Hudi
- Serverless, transactional layer over lakes.
- Multi-engine, Decoupled storage from engine/compute
- Upserts, Change capture on lakes
- Introduced Copy-On-Write and Merge-on-Read
- Ideas now heavily borrowed outside

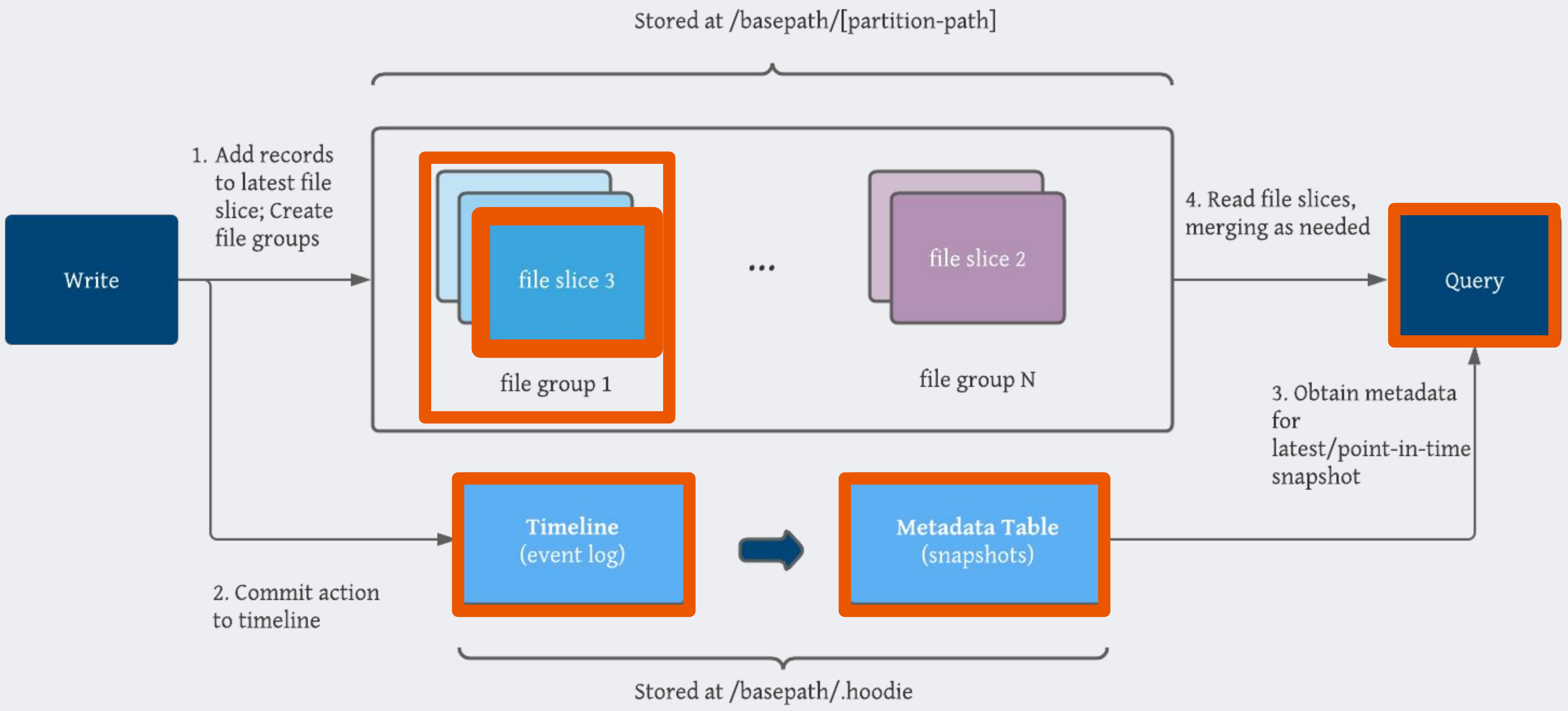


https://eng.uber.com/hoodie/ Mar 2017

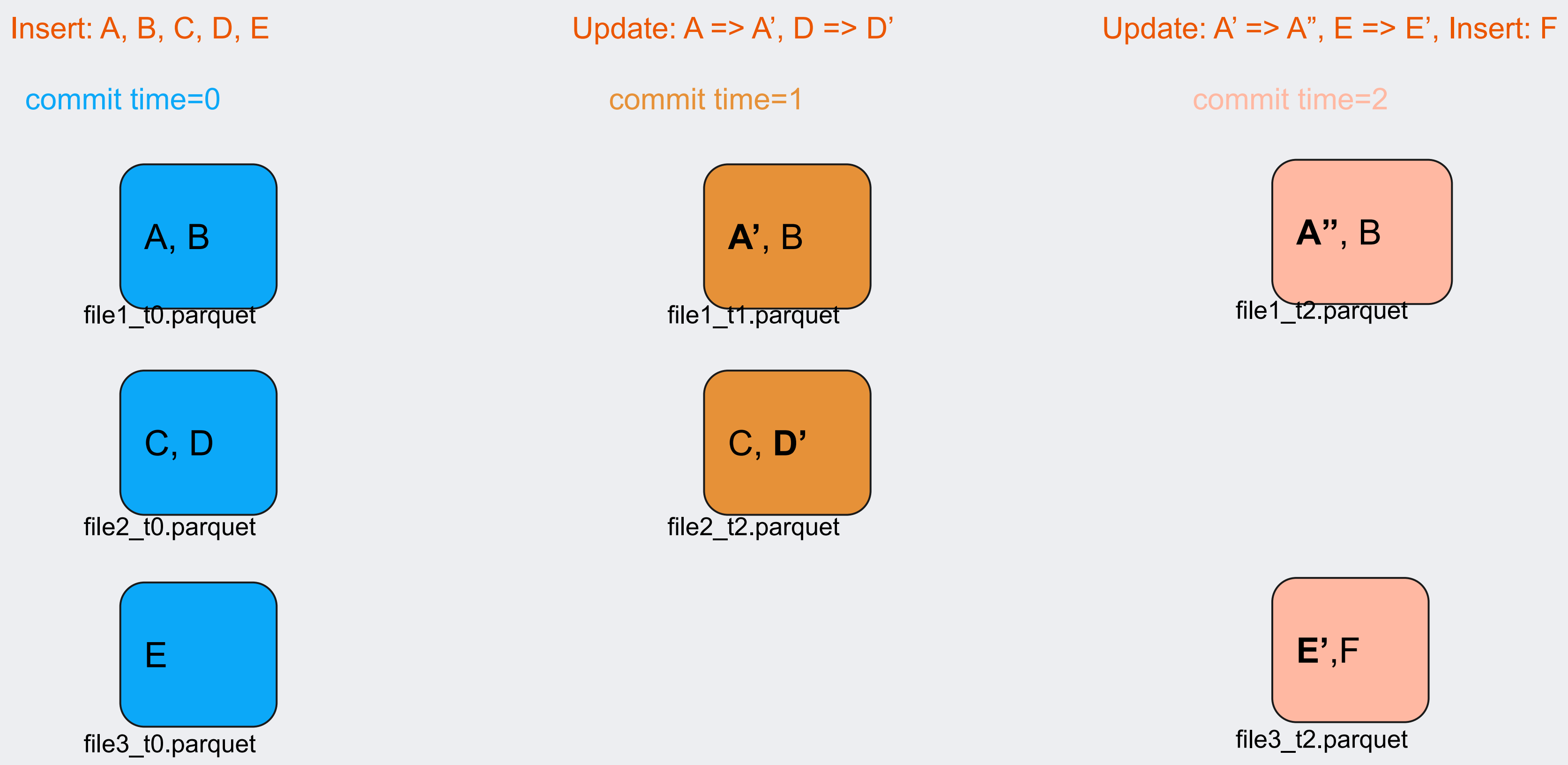
Apache Hudi - Upserts & Incrementals



Apache Hudi - Storage Layout

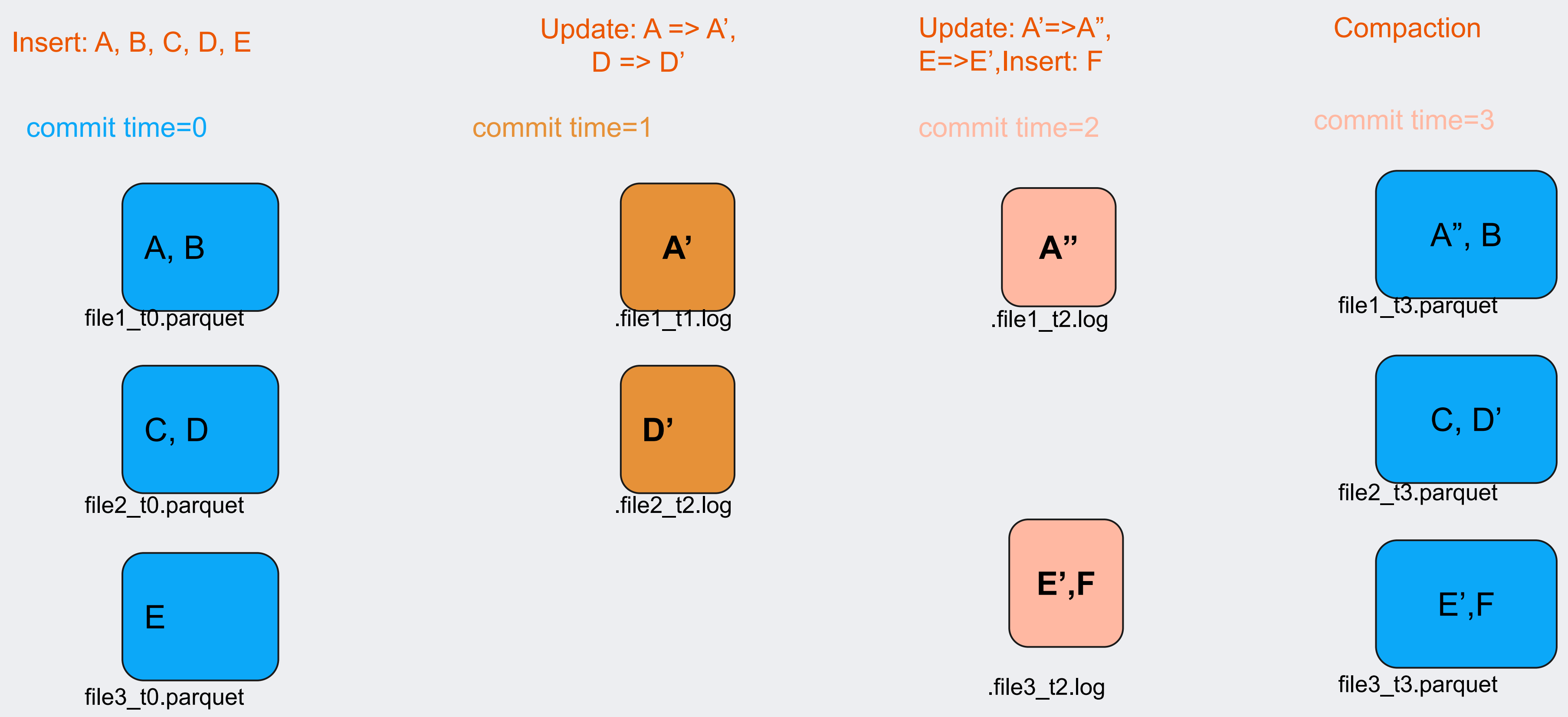

Stored at /basepath/[partition-path]
Write
1. Add records to latest file slice; Create file groups
file slice 3
file group 1
...
file slice 2
file group N
4. Read file slices, merging as needed
Query
3. Obtain metadata for latest/point-in-time snapshot
Timeline (event log)
Metadata Table (snapshots)
2. Commit action to timeline
Stored at /basepath/.hoodie

Apache Hudi - Copy-On-Write Table



Snapshot Query	A,B,C,D,E	A’,B,C,D’,E	A”,B,C,D’,E’,F
Incremental Query	A,B,C,D,E	A’,D’	A”,E’,F

Apache Hudi - Merge-On-Read Table



Snapshot Query	A,B,C,D,E	A',B,C,D',E	A",B,C,D',E',F	A",B,C,D',E',F
Incremental Query	A,B,C,D,E	A',D'	A",E',F	A",E',F
Read Optimized Query	A,B,C,D,E	A,B,C,D,E	A,B,C,D,E	A",B,C,D',E',F

Apache hudi

The Community

Pre-installed on 5 cloud providers

aws Google Cloud Alibaba Cloud Tencent Cloud HUAWEI

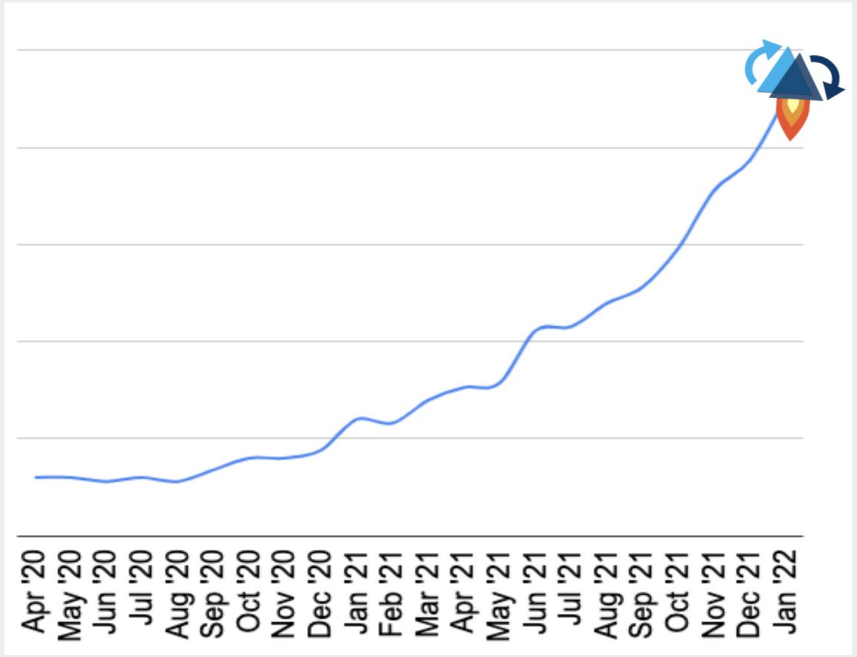
Diverse PMC/Committers

Uber aws lyft Robinhood zendesk ByteDance Tencent Cloud 蚂蚁金服 ANT FINANCIAL DV DoubleVerify

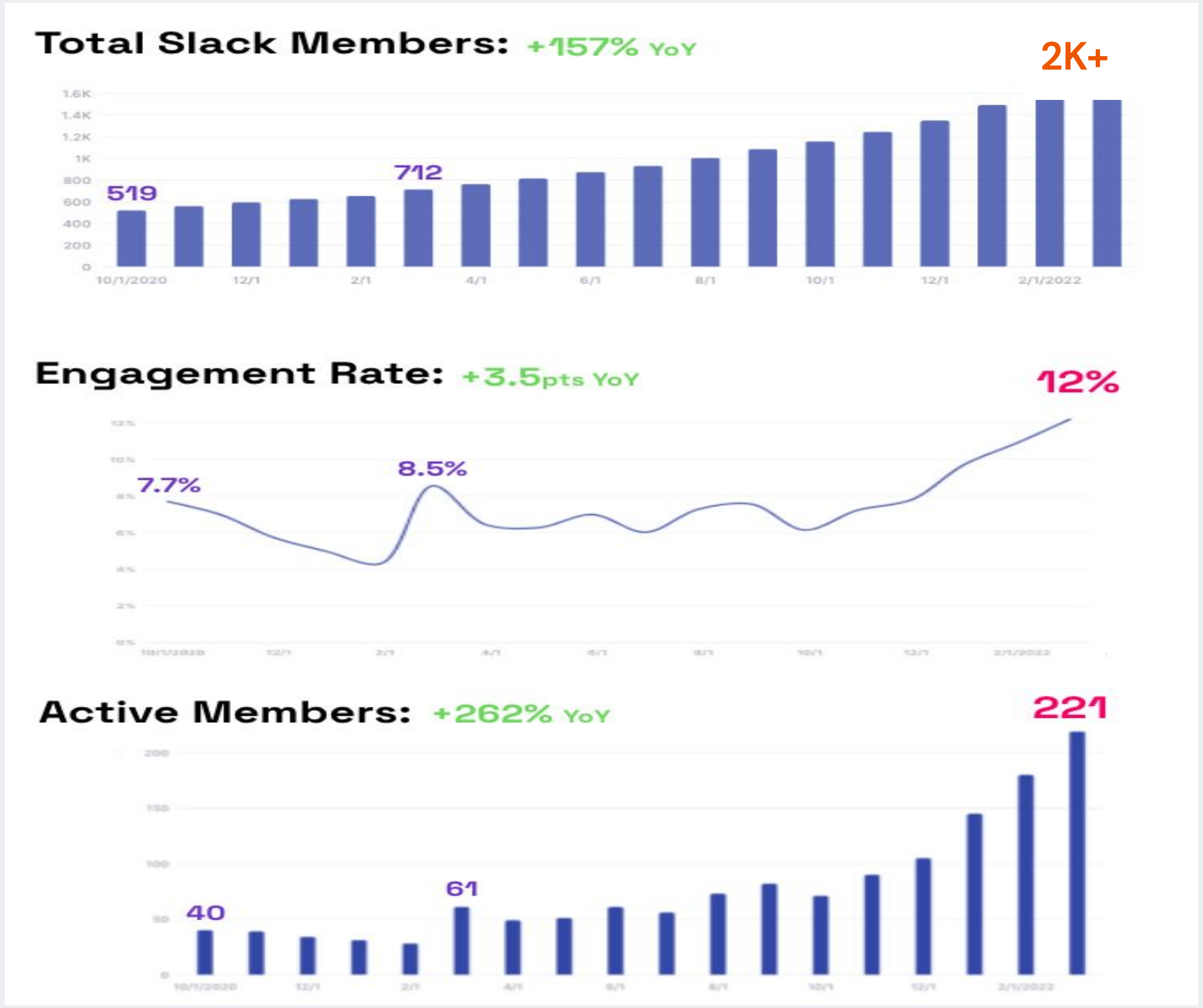
Rich community of participants

ebay Walmart udemy Disney+ Bai du 百度 Roku zoom shopify amazon Morgan Stanley SK telecom SAMSUNG zomato HSBC Analytics Vidhya Alipay H3C 新 华 三 集 团

800B+ Records/Day (from even just 1 customer!)	2000+ Slack Members	1M DLs/month (400% YoY)
1000+ GH Engagers	225+ Contributors	20+ Committers

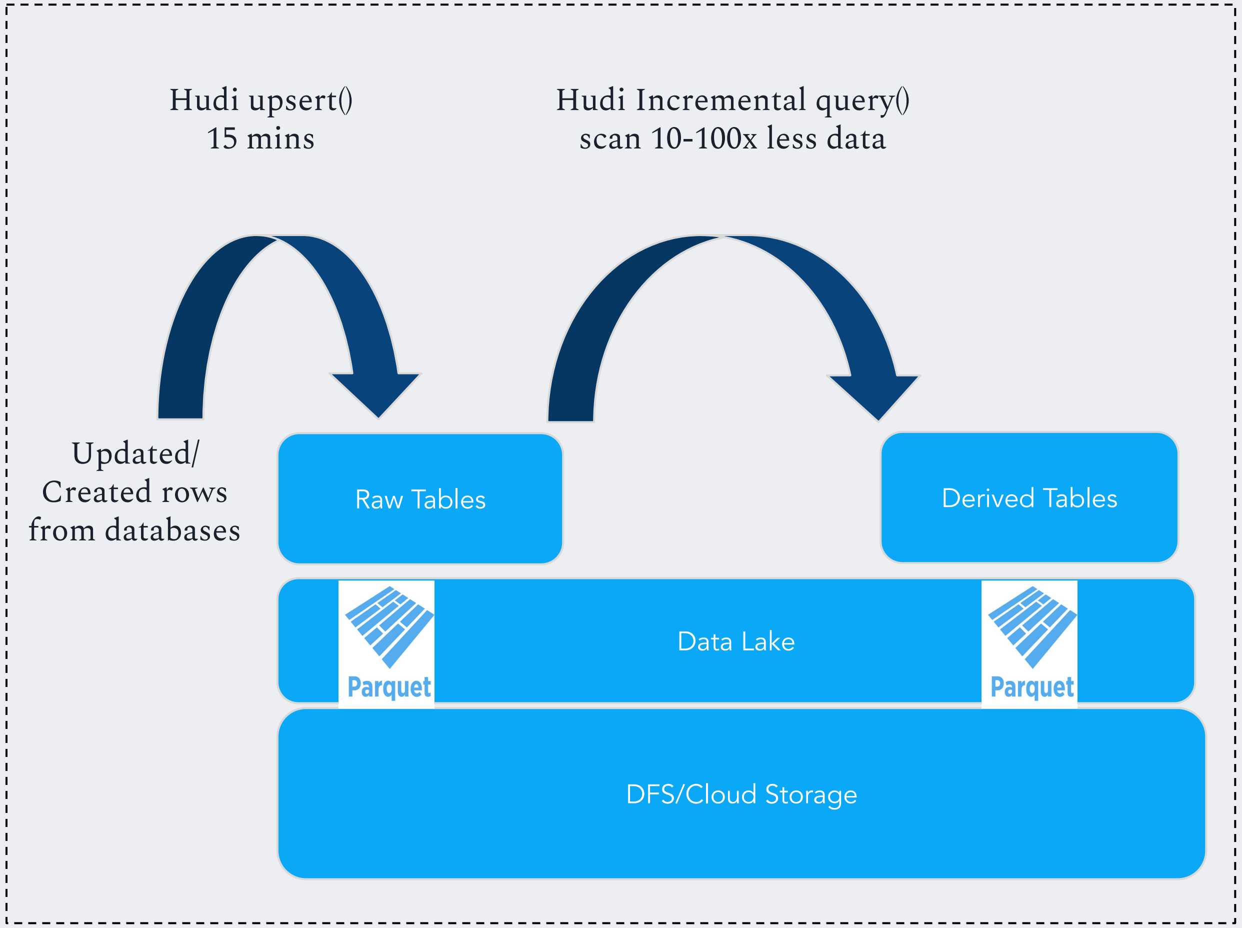


Apache Hudi - Community



Apache Hudi - Relevant Features

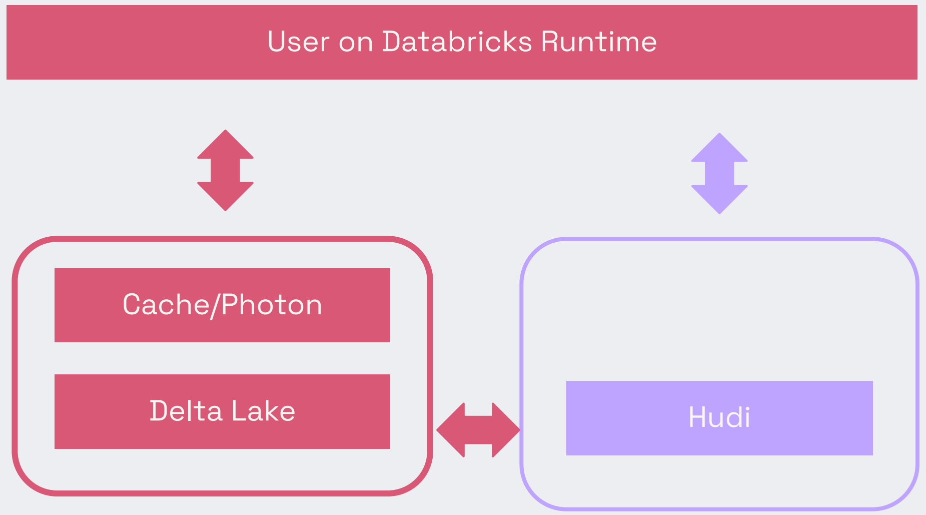
- Database abstraction for cloud storage/hdfs
- Near real-time ingestion
- Incremental, Efficient ETL downstream
- ACID Guarantees



Apache Hudi + Databricks = Best of Both Worlds!

- We believe
  - Hudi offers the most complete lakehouse storage platform.
  - Databricks offers great Spark experience, but cache/photon don’t work natively on Hudi.

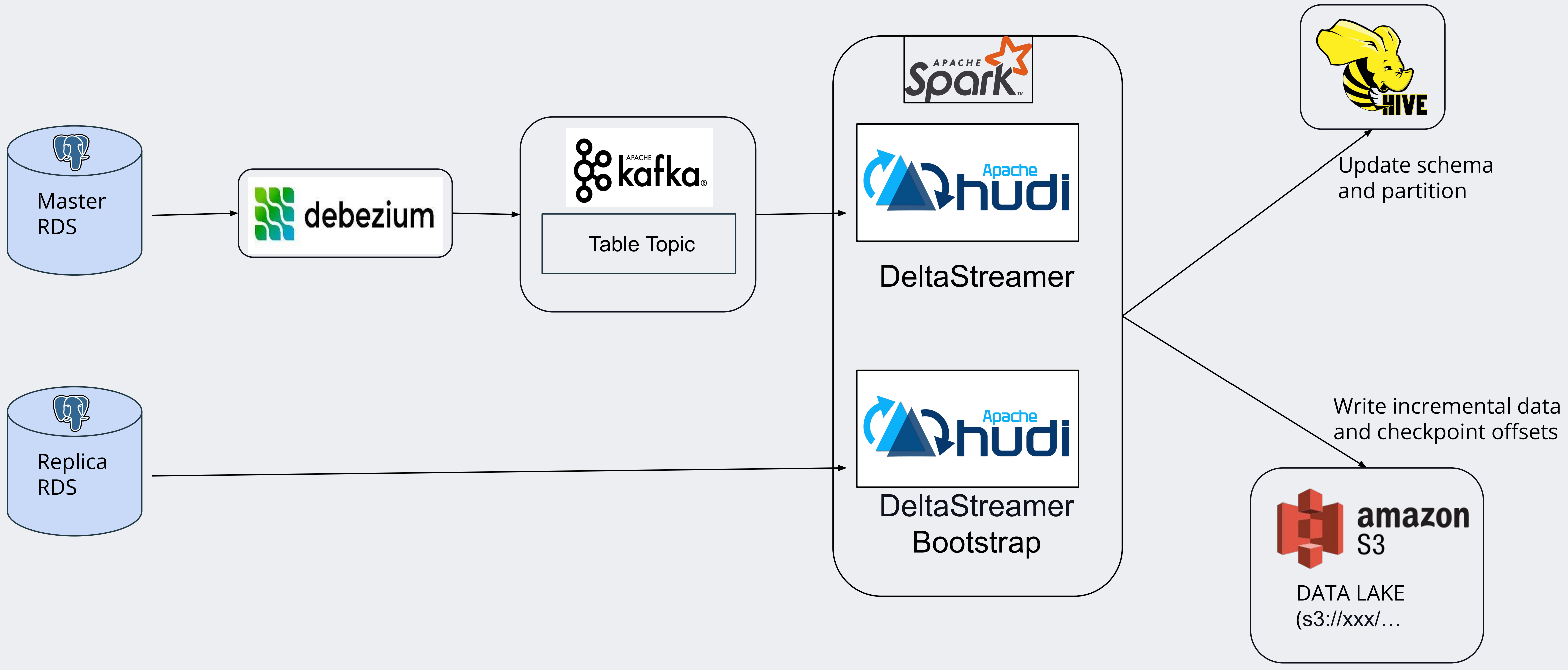
- Hudi supports pluggable metadata syncing - BigQuery, Hive, Glue, DataHub, …
- WIP - A bridge between Hudi and Delta Lake.
  - Query Hudi table snapshot as a Delta Lake table



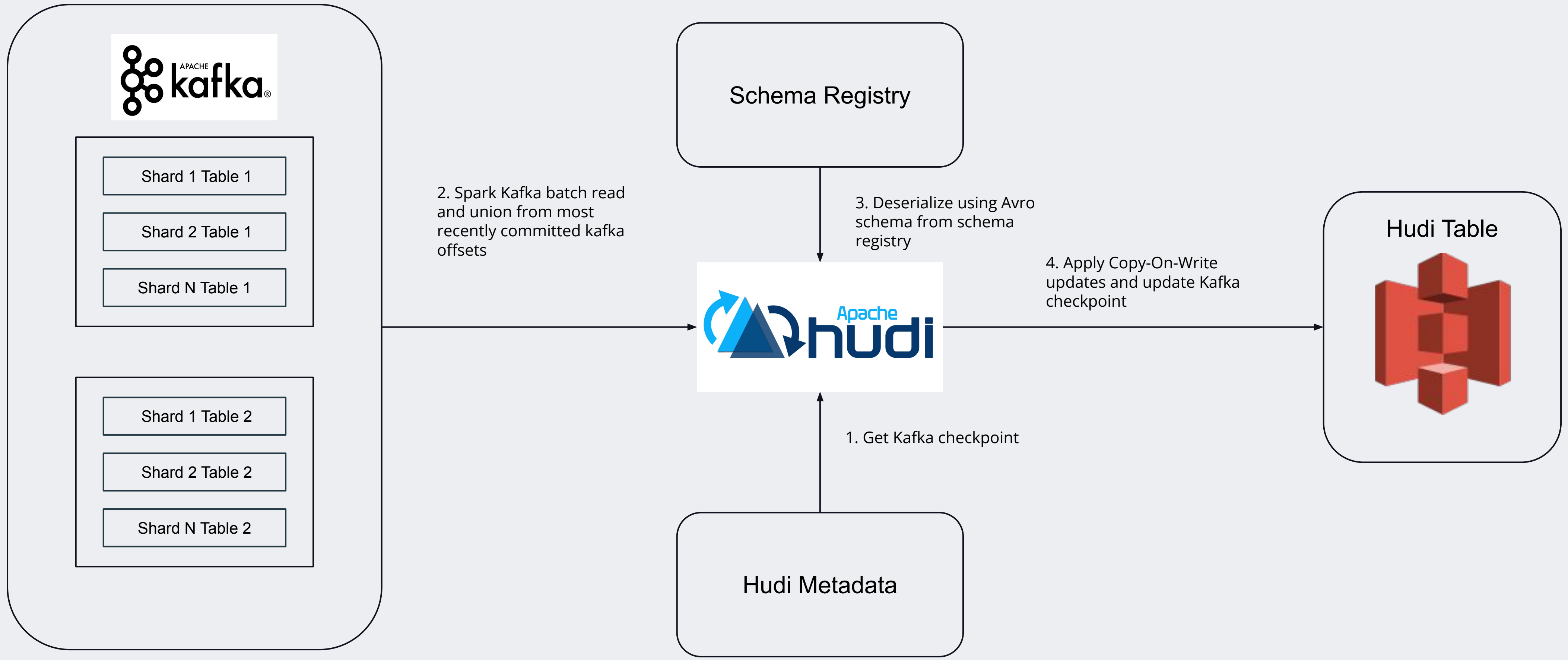
DATA+AI SUMMIT 2022

Deep Dive Ingestion

Recap - High Level Architecture



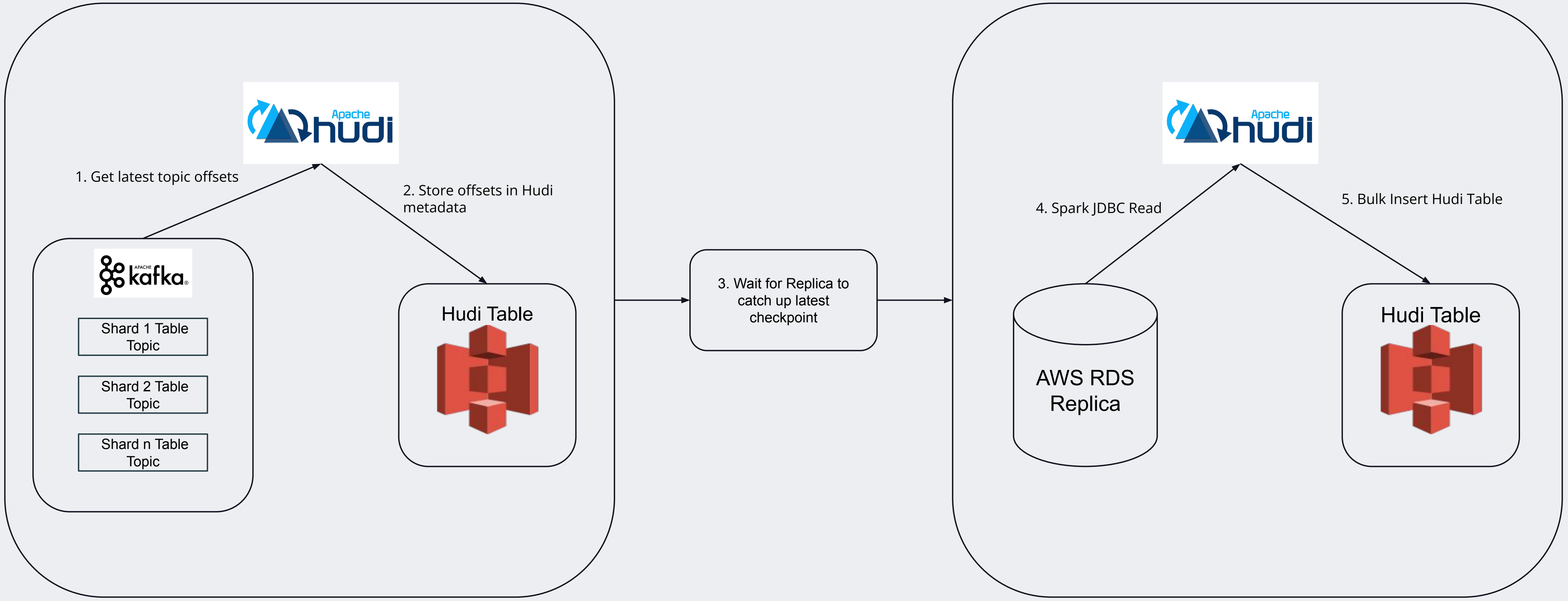
Data Lake Ingestion - CDC Path



Apache kafka
Shard 1 Table 1
Shard 2 Table 1
Shard N Table 1
Shard 1 Table 2
Shard 2 Table 2
Shard N Table 2
2. Spark Kafka batch read and union from most recently committed kafka offsets
Schema Registry
3. Deserialize using Avro schema from schema registry
Apache hudi
1. Get Kafka checkpoint
Hudi Metadata
4. Apply Copy-On-Write updates and update Kafka checkpoint
Hudi Table

DATA+AI SUMMIT 2022

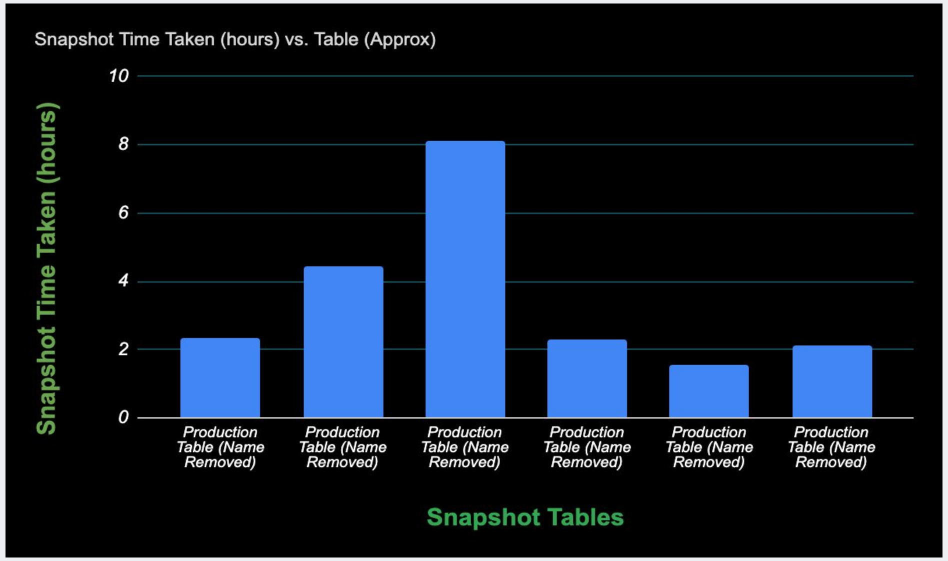
Data Lake Ingestion - Bootstrap Path

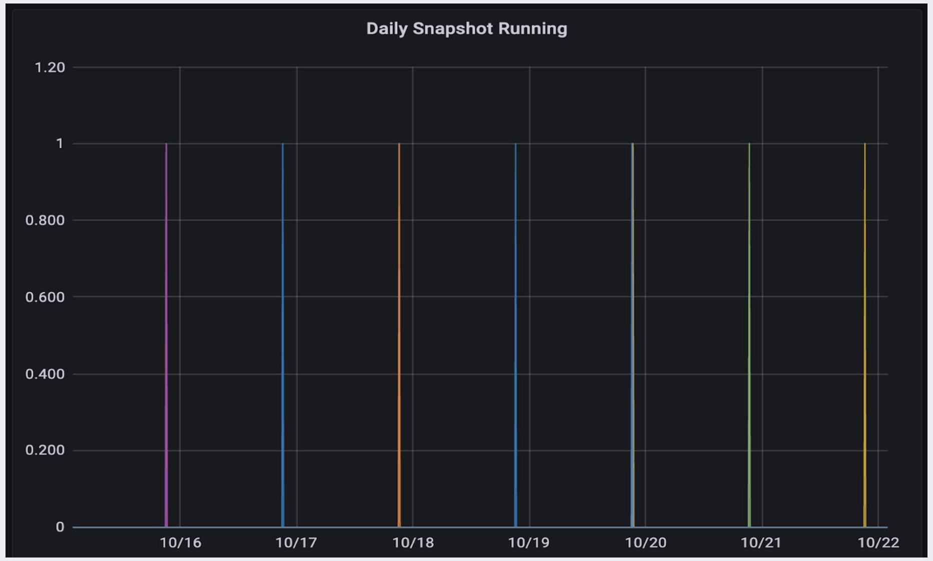


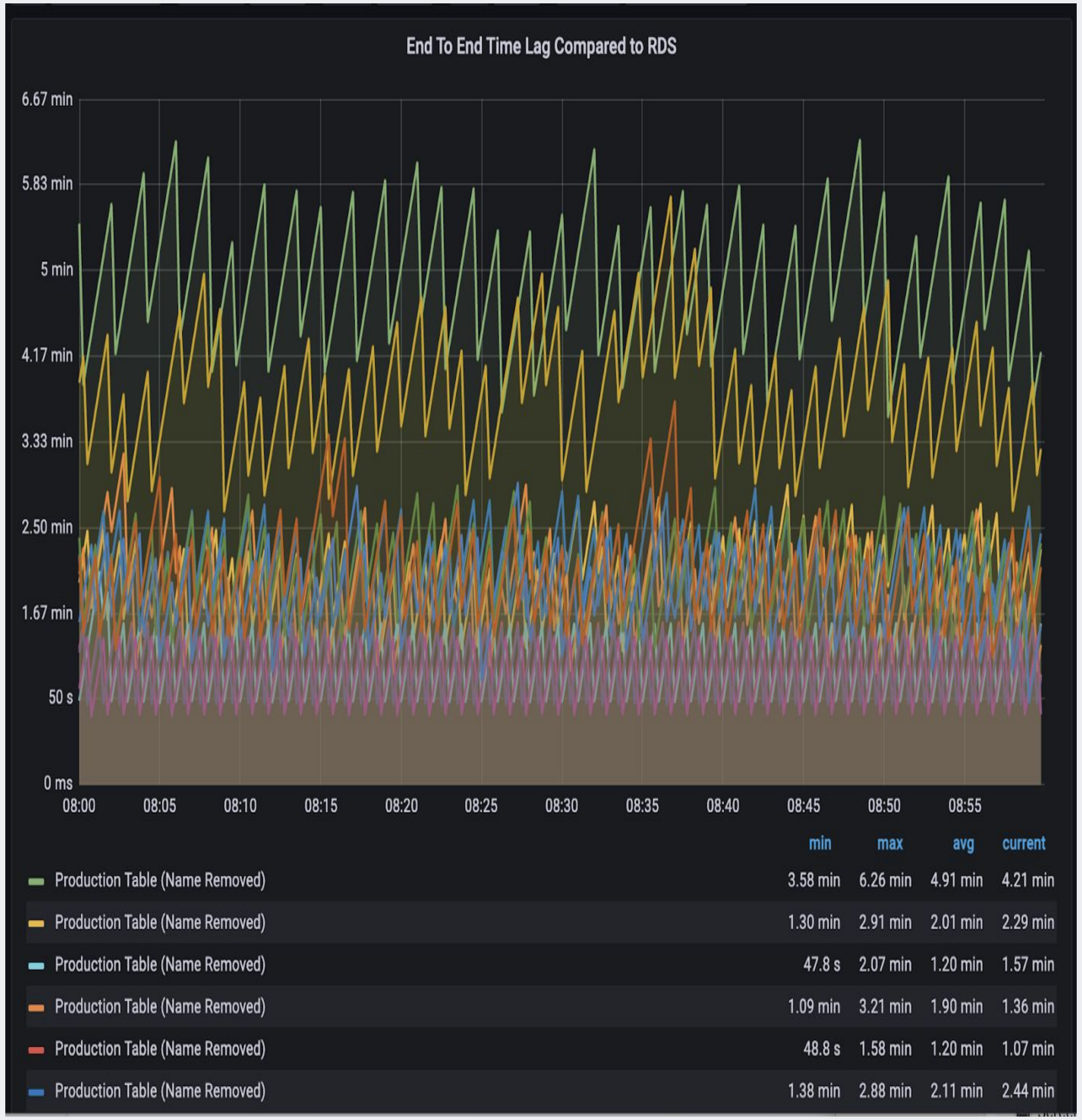
Running Ingestion at Scale

- 4000+ tables
- Automation, Self Healing
- Tiered SLAs - Provisioning and Isolation
- Pre-Commits and Validation for Quality Checks
- Monitoring & Alerting

Improved Freshness







	min	max	avg	current
🟩 Production Table (Name Removed)	3.58 min	6.26 min	4.91 min	4.21 min
🟨 Production Table (Name Removed)	1.30 min	2.91 min	2.01 min	2.29 min
🟦 Production Table (Name Removed)	47.8 s	2.07 min	1.20 min	1.57 min
🟧 Production Table (Name Removed)	1.09 min	3.21 min	1.90 min	1.36 min
🟥 Production Table (Name Removed)	48.8 s	1.58 min	1.20 min	1.07 min
🟦 Production Table (Name Removed)	1.38 min	2.88 min	2.11 min	2.44 min

DATA+AI SUMMIT 2022

Thank you!

We’ll hangout at back of room for any QA.

Balaji Varadarajan

Senior Staff Engineer, Robinhood Markets

Apache Hudi PMC

Vikrant Goel

Engineering Manager, Robinhood Markets