The Apache Lakehouse Ecosystem and Apache Beam's Role in It

JB Onofré <jbonofre@apache.org>

BEAM SUMMIT NYC 2026

JB Onofré

- Director of the Apache Software Foundation, member for a long long time :)
- Committer/PMC member on 20+ Apache projects (Apache ActiveMQ, Apache Beam, Apache Polaris, ...)
- Mentor/champion on Apache Incubator projects (podlings)
- Open Source Architect at Dremio, leading the OSPO team

The Lakehouse Promise

Why the industry converged on a new architecture …

Two Worlds - One Architecture

Data Lake	Data Warehouse	Lakehouse
✓ Low cost (object storage)	✓ ACID transactions	✓ **Open formats on object storage**
✓ Unlimited scale	✓ Fast SQL queries	✓ **ACID + schema evolution**
✓ Any format / any data	✓ Strong governance	✓ **Time travel & rollback**
✗ No transactions (ACID)	✗ Expensive & siloed	✓ **Unified batch + streaming**
✗ No schema enforcement	✗ Proprietary formats	✓ **Direct BI + ML access**
✗ Poor BI / SQL performance	✗ Weak ML / streaming support	✓ **Vendor-agnostic**

True story from every data team ever:

"We have a data lake."

"Great! What's in it?"

"...files."

This is why we need table formats.
A lake without a map is just a swamp.

Open Table Formats

Apache Iceberg, Apache Hudi, Delta Lake

What open table formats add to plain Apache Parquet

ACID Transactions

Serializable isolation for concurrent reads & writes on cloud object storage

Schema Evolution

Add / rename / drop columns without rewriting the entire table

Time Travel

Query any historical snapshot by timestamp or version ID

Partition Evolution

Change partition strategy without migrating existing data

Incremental Reads

Efficiently consume only new or changed files since last checkpoint

Metadata Catalog

Rich metadata layer enables predicate pushdown and file pruning

Choosing a format: key trade-offs

	Apache Iceberg	Apache Hudi	Delta Lake
Governance	Apache (vendor-neutral)	Apache (vendor-neutral)	Linux Foundation
Write model	Copy-on-Write / Merge-on-Read	Copy-on-Write / Merge-on-Read	Copy-on-Write default
Streaming strength	Strong (append + CDC)	Native CDC / upserts	Good (DLT integration)
Catalog integration	REST, Apache Polaris, …	Hive, AWS Glue	Unity Catalog
Beam Managed I/O	✓ Production ready	Community / in-progress	Via Spark runner
Best for	Broad, vendor-neutral stacks	High-velocity CDC & upserts	Databricks-heavy orgs

Choosing between Apache Iceberg, Apache Hudi & Delta Lake

is a bit like choosing a JavaScript framework.

Apache Iceberg

The one everyone agrees is architecturally correct
but took more time to adopt, but strongest

≈ TypeScript

Apache Hudi

Born from real production pain at massive scale.
Opinionated. Fast. Battle-tested.

≈ React

Delta Lake

Backed by one big vendor, works great in their ecosystem, lock-in?

≈ Angular

Apache Iceberg: layered architecture

Catalog	Single pointer to current metadata file
Metadata JSON	Snapshot log, schema history, partition spec
Manifest List (Snapshot)	Points to all manifests for a snapshot
Manifest Files	Lists of data files + column stats
Data Files	Parquet / ORC / Avro in object storage

Why this matters:

- Snapshot isolation — readers never see partial writes
- Time travel = query any historical snapshot
- Metadata drives file pruning — skip irrelevant files
- Manifest files enable incremental Beam reads

The Ingestion Problem

Getting data into lakehouse tables, reliably, at scale

Quick distributed system quiz:

"Exactly-once delivery"

in a distributed system means...

A

Every message is processed exactly once

B

At-least-once + idempotent operations (the real answer)

C

A lie we tell each other to sleep at night

(C is also technically correct. We'll handle that with idempotent writes.)

Moving data in is harder than it looks

The challenges

Exactly-once semantics
Duplicate events corrupt analytics

Schema evolution
Upstream schemas change without notice

Mixed workloads
Batch backfills alongside live streams

Multi-table fan-out
One source, many target tables

Runtime lock-in
Flink cluster today, Spark tomorrow?

The naive approaches

Custom Spark jobs
Brittle, tightly coupled to one runtime

Kafka Connect S3 sink
Lands files, but no table semantics (Iceberg Connector is interesting)

Airbyte / Fivetran
Great for batch; limited streaming control

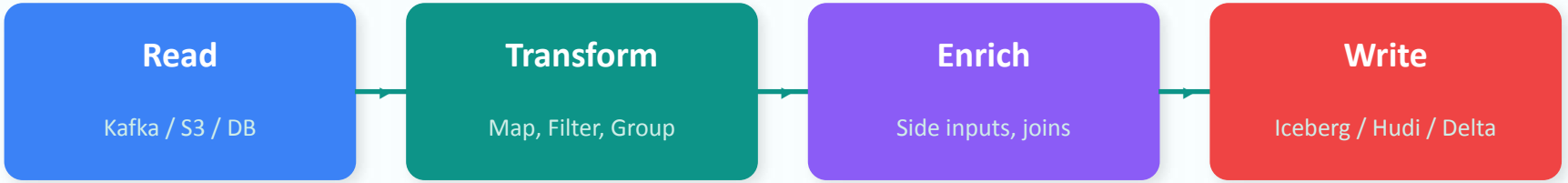
Hand-rolled Flink jobs
Powerful but per-format boilerplate

Copy-paste pipelines
Code duplication, no portability

Apache Beam's fit

Unified model, Runner abstraction, Managed I/O

One model: batch, streaming, and everywhere in between



Same pipeline code runs on any runner:

Apache Flink | Apache Spark | Google Dataflow | Direct Runner | Samza / Hazelcast

The Beam portability principle:

Write your pipeline logic once in the Beam model. Choose your execution engine (runner) at deploy time — or switch without touching pipeline code.

Managed I/O: Beam's lakehouse connector layer

What is Managed I/O?

- A unified API surface for I/O connectors in Beam
- Configuration-driven, not code-driven
- Handles schema inference, checkpointing & exactly-once
- Iceberg connector: production-ready read + write
- Supports both bounded (batch) and unbounded (streaming) sources
- Compatible with all Beam runners

Java — write to Iceberg via Managed I/O

```
// Configure Managed Iceberg write
Map<String, Object> config = Map.of(
  "table", "prod.events.clickstream",
  "catalog_name", "my_rest_catalog",
  "triggering_frequency_secs", 60
);

pipeline
  .apply(KafkaIO.read()...)
  .apply(Convert.toRow(...))
  .apply(Managed.write(
      Managed.ICEBERG)
    .withConfig(config));
```

Beam is like a universal power adapter.

Your pipeline code is the device.
Flink, Spark, and Dataflow are the wall sockets in different countries.
Beam makes sure you don't fry your laptop.

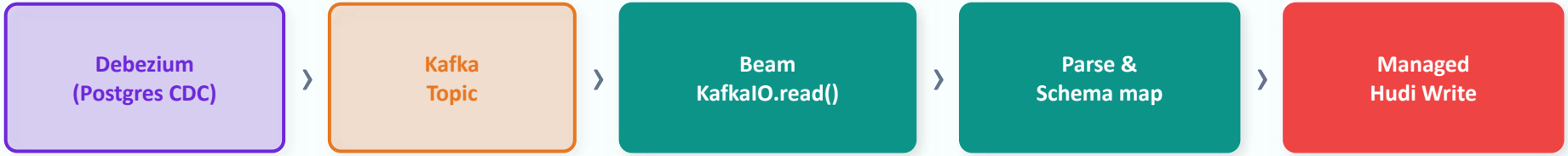
No more: *"We migrated from Flink to Spark and had to rewrite everything."*

👋 **Goodbye vendor lock-in.**

Practical Pipelines Patterns

Real patterns teams use today

Pattern 1: CDC streaming into Apache Hudi



Record key strategy

Use primary key as Hudi record key for deterministic upserts

Partitioning

Partition by event_date for efficient time-range scans

Exactly-once

Beam checkpointing + Hudi idempotent writer = no duplicates

Latency

Sub-minute with MoR (Merge-on-Read) write mode in Hudi

Pattern 2: Batch backfill into Apache Iceberg

Parallelism

Beam auto-parallelizes file reads; Reshuffle prevents fusion to distribute write workers

Snapshot isolation

All writes go to a new Iceberg snapshot; readers see a consistent table until the commit

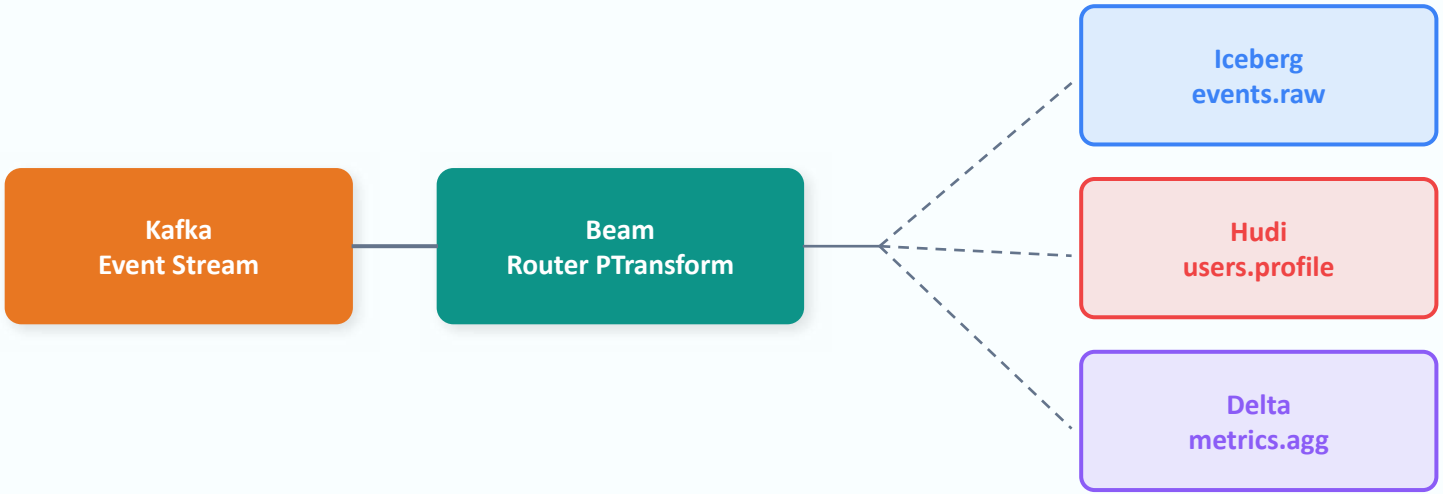
Schema detection

Beam Row type + Iceberg schema registry — handle schema drift without pipeline restarts

Incremental mode

Use Iceberg's snapshot log to read only new snapshots on subsequent runs — no full scan

Pattern 3: Multi-table fan-out in a single pipeline



Why Beam?

- ✓ One pipeline
- ✓ One deploy
- ✓ Shared I/O state
- ✓ Single schema registry
- ✓ One checkpoint

Complementary, not competing

Apache Beam

Programming model & portability layer

- Runtime-agnostic pipelines
- Unified batch + streaming model
- Managed I/O connector ecosystem
- Ideal for multi-target ingestion
- Write once, run anywhere

When portability, multi-runtime, or mixed batch+stream in one pipeline matters

Apache Flink

High-throughput stateful stream processor

- Lowest latency streaming
- Complex stateful operations
- Native Kafka source integration
- Mature Iceberg sink (Flink SQL)
- Best for pure streaming workloads

When you need sub-second latency, heavy stateful joins, or native Flink SQL

Apache Spark

Large-scale batch & micro-batch engine

- Massive batch throughput
- Spark Structured Streaming
- Rich ML/analytics ecosystem
- Native Delta Lake integration
- Best for large-scale transformations

When you need giant batch transformations, Delta Lake, or Spark ML integration

How to choose: a practical decision guide

Need portability across runtimes?

→ Start with Beam

Sub-second latency, heavy stateful joins?

→ Flink (or Beam on Flink runner)

Batch backfill > 10 TB?

→ Spark (or Beam on Spark runner)

Writing to Iceberg today?

→ Beam Managed I/O is production-ready

CDC upserts into Hudi?

→ Beam (community Hudi sink) or Flink

Databricks-heavy org?

→ Delta Lake + Spark Structured Streaming

Mixed batch + streaming in one job?

→ Beam's unified model wins

Already invested in Flink SQL?

→ Keep it; Beam wraps it as a runner

QUESTIONS?