streambased.

Streambased: Unifying Kafka and Iceberg into a single logical view.

Zero-copy, zero-lag access across hot set and cold set – without always-on ingestion jobs.

Challenge:
Kafka and Apache Iceberg are optimised for very different workloads: Kafka for low-latency, append-only streaming; Iceberg for large-file, versioned analytical queries. Bridging them with ETL pipelines forces a choice between data freshness and operational simplicity that only compounds as data volumes grow.

Solution:
Streambased eliminates that trade-off by acting as a zero-copy data virtualisation layer that translates Kafka topics into Iceberg-compatible tables at query time and exposes full Iceberg history back through the Kafka protocol. No data movement ahead of query time; no ingestion pipelines to maintain; no accumulation lag. The result is a single, continuously consistent view of data from the latest millisecond in Kafka to years of history in Iceberg, queryable by any standard Iceberg engine or Kafka consumer.

Use cases span any domain where data latency and decision accuracy are inversely correlated: fraud detection, underwriting, demand forecasting, IoT failure prediction and market response. Analytics quality is also bounded by data scope: ETL bottlenecks often force teams to exclude high-volume operational streams and miss the subtle signals that live in telemetry, clickstream, transactional records and third-party feeds. Removing the ingestion barrier makes the full breadth of available data analytically accessible without a pipeline project first, which shortens time to value for new sources and radically increases the surface area of analytically addressable problems.

Challenge: Kafka and Iceberg are not designed to work together.

Kafka messages arrive in small batches (typically 16 KB). Iceberg performs best when data is written in large file chunks (optimally 128 MB to 1 GB). This fundamental mismatch is the root cause of most Kafka-to-Iceberg integration problems.

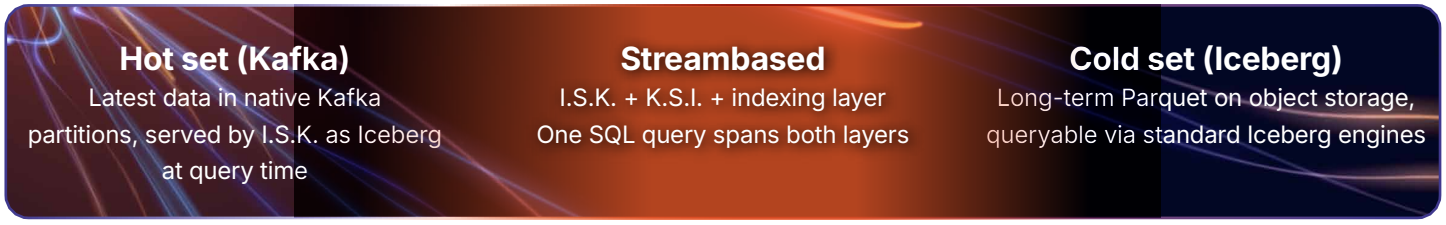
Core trade-offs in ETL copy-based integration

Issue	Root cause	Operational impact
Ingestion lag of 15 minutes or more	Kafka messages (~16 KB) must accumulate to efficient Iceberg file sizes (~512 MB or more) before committing; no data is queryable during this window.	**Live Kafka data unavailable to SQL engines during the accumulation period; minimum latency floor regardless of pipeline health.**
Small-file proliferation	Each Kafka flush creates a new Parquet file; at typical streaming rates, millions of sub-optimal files accumulate.	**Degraded read performance as every file must be opened and closed per query; bloated object storage costs; compaction jobs needed.**
Snapshot explosion	Every Iceberg write creates a snapshot; high-frequency streams generate millions of metadata entries.	**Query planning overhead grows with snapshot count; rollback operations slow; manual expiration jobs must be scheduled.**
Duplicated data storage	ETL holds a full copy of data in both Kafka and Iceberg with independent retention policies.	**Significant storage cost duplication; schema skew risk; two sources of truth create consistency questions.**
Ongoing Iceberg admin overhead	Iceberg is a format specification, not a managed service; compaction, snapshot expiry and partition evolution must be externally scheduled and executed.	**Requires dedicated Spark or Trino jobs to be built, monitored and recovered; operational complexity scales with data volume.**
Schema governance breakdown	ETL pipelines enforce no contract between Kafka and Iceberg schema; when a field type changes on one side (eg boolean to integer), the other side is completely unaware and continues writing against the old definition.	**Silent data corruption or loss eg the Iceberg column retains the original type, the semantic change goes uncaptured, and once Kafka retention expires the correct data is gone for ever.**

Solution: Streambased zero-copy data virtualisation.

Rather than copying Kafka data into Iceberg ahead of query time, Streambased projects it as a logical view. At query execution, the relevant Kafka data is fetched, translated into Iceberg format in memory and merged with cold Iceberg history before being returned to the query engine. The approach maps onto the established Lambda architecture but replaces the complexity of combining two data layers with a single SQL UNION across two Iceberg-compatible views.

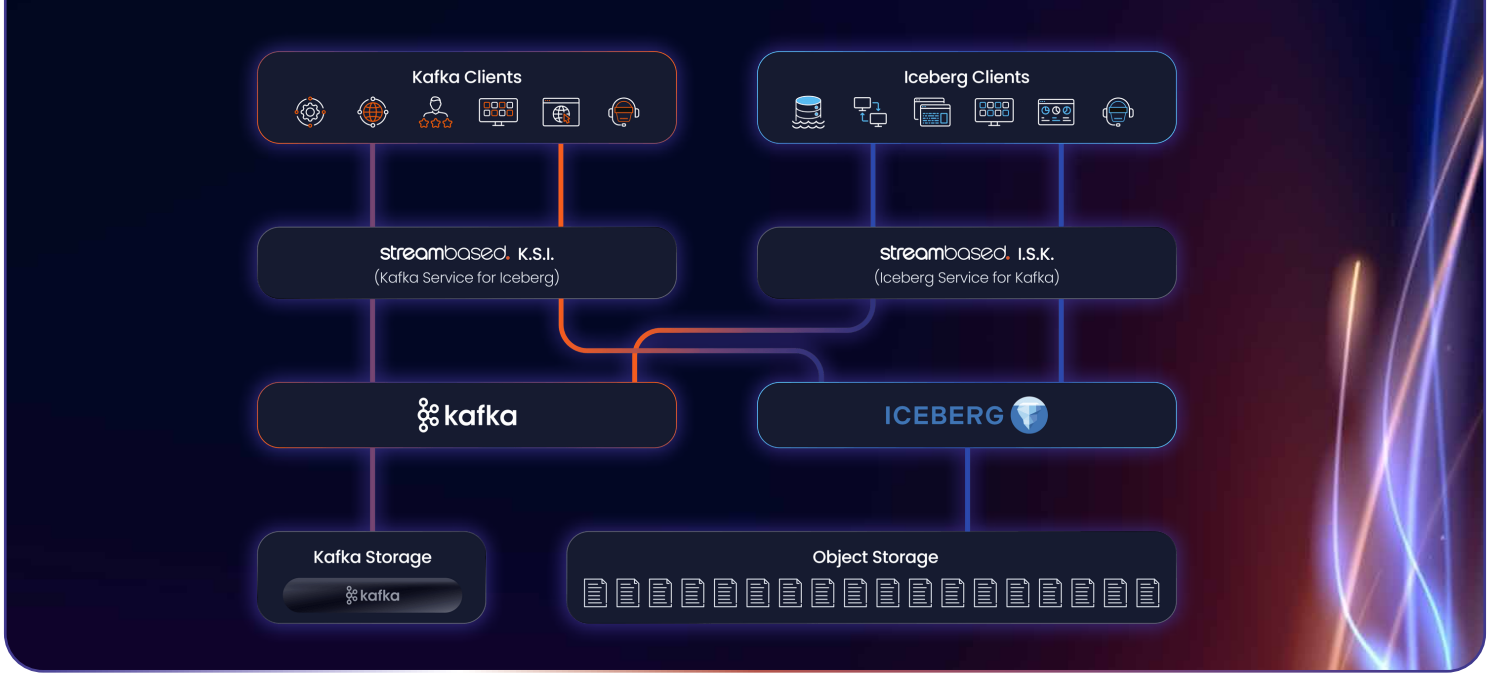
Architecture: hot set, cold set and merged set



Kafka consumers, microservices, observability platforms

SELECT * FROM isk.merged.sometable

Spark, Trino, Dremio, Snowflake, Databricks, DuckDB



What Streambased is (and what it isn't)

Streambased is a stateless virtualisation layer that sits above existing Kafka and Iceberg deployments, requiring no changes to broker configuration, data migration or dedicated ingestion jobs. It's not a database, sink connector or ETL system.

What Streambased is

- An abstraction layer that creates composable, unified views above Kafka and Iceberg.
- A query-time protocol translator: I.S.K. (Iceberg Service for Kafka) exposes Kafka topics as Iceberg-compatible tables; K.S.I. (Kafka Service for Iceberg) exposes Iceberg history via the Kafka wire protocol.
- Read-only by design: Streambased does not write into Kafka or Iceberg as part of its query path.
- A complement to existing infrastructure: it deploys alongside current ETL jobs, warehouses and processing engines with no cutover required.

What Streambased isn't

- Not a database, but a composable approach that brings together traditional database components while avoiding tight coupling.
- Not a sink connector: data does not need to be physically landed before it is accessible.
- Not an ingestion pipeline: there are no always-on jobs consuming from Kafka to pre-materialise files.
- Not a replacement for Kafka or Iceberg: it adds a unified access layer on top of both.

Key technical capabilities

I.S.K. (Iceberg Service for Kafka)

- Translates Kafka topics into Iceberg table projections at query time with no physical data movement or pre-staging.
- Eliminates compaction, snapshot expiry and partition evolution overhead for hot data: Kafka owns data layout; Iceberg metadata is ephemeral and imposed at query time.
- Zero-lag guarantee: data is fetched directly from Kafka at query execution; no accumulation window.
- Compatible with any Iceberg-compatible query engine including Spark, Trino, Dremio, Athena, Snowflake and Databricks via external Iceberg tables.
- Schema evolution in Kafka (field additions, deletions, type widening) is immediately reflected in the Iceberg view with no replay or rewrite required.

K.S.I. (Kafka Service for Iceberg)

- Kafka-protocol proxy enabling standard Kafka consumers to fetch from Iceberg cold storage transparently.
- Federates consumer operation across hot (Kafka) and cold (Iceberg) sets, presenting a single logical offset timeline.
- Enables streaming applications to access full historical depth without the cost of extended Kafka retention.

Deployment

- Overlay architecture: Streambased requires only the standard public Kafka consumer and admin APIs; no broker modification.
- Compatible with Confluent Cloud, Amazon MSK, Aiven, Redpanda and on-premises Apache Kafka 2.5.0 or above.
- All components are stateless and scale horizontally; no single points of failure.
- Hot-to-cold migration runs as a standard Iceberg INSERT... SELECT. Controlled by you, trigger at quiet times, manually or by configurable thresholds.
- Existing ETL jobs and downstream engines continue to operate unchanged alongside a Streambased deployment; no big-bang cutover required.

Technical specifications

Note: latency and throughput figures are workload-dependent. Results vary with data volume, query shape and underlying Kafka and Iceberg configuration.

Platform	Description
Kafka compatibility	Apache Kafka wire protocol; Kafka 2.5.0 and above; all managed and self-hosted distributions.
Iceberg compatibility	Apache Iceberg v2+ open table format; compatible with all major Iceberg-aware query engines.
Query engines supported	Spark, Trino, Dremio, Athena, Snowflake (external Iceberg tables), Databricks, DuckDB and any other Iceberg-compatible engine.
Storage backend	Any S3-compatible object store (Amazon S3, MinIO and equivalents).
Iceberg catalog	Compatible with existing Iceberg catalogs.
Kafka Schema	Derived from Schema Registry.
Schema evolution	Field additions, deletions and type widening reflected immediately in Iceberg view; no replay required.
Deployment model	Runs alongside existing Kafka and Iceberg infrastructure.
Scaling	All components stateless; horizontal scaling.
Security	Inherits and enforces all Kafka ACLs, authentication mechanisms (SASL/SCRAM, SASL/GSSAPI, mTLS) and Iceberg catalog security; no separate access control layer required.
Consistency	Read-your-writes; overlapping data between Kafka and Iceberg deduplicated at query time; ACID where underlying systems provide it.
Monitoring and management	Slipstream GUI is the metrics viewing system, providing alerting, index management and cluster administration.
Data freshness	Zero lag for hot set: data available at the millisecond it is produced in Kafka; cold set at standard Iceberg snapshot freshness.

How Streambased compares

Capability	Connector based (Kafka Connect / sink)	Connector Managed (Tableflow)	Tiered Storage (Buffstream / Ursa)	Streambased (Projection / zero copy)
Data freshness	>15 min (approx.)	Minutes (approx.)	Seconds (approx.)	**Zero lag**
Single source of truth	No (two copies)	No (two copies)	Single source of truth	**Yes**
Iceberg admin burden	Manual	Automated	Automated	**Eliminated**
Works with existing Kafka	Yes	Vendor lock-in	Replace broker	**Yes (any)**
Kafka reads from Iceberg (K.S.I.)	No	No	Partial	**Yes**
Deployment model	Kafka connector	Managed cloud only	Replace broker	**Overlay layer; no broker change**

Getting started

Streambased deploys as a stateless overlay on your existing Kafka cluster. No broker changes, no data migration, no pipeline cutover. A typical proof of value runs against a defined scope with an agreed use case in two weeks.

Contact hello@streambased.io or visit streambased.io

streambased.

streambased.io