SK

STREAMKERNEL V0.2.0 · TECHNICAL WHITEPAPER · MAY 12, 2026

Pulsar-to-Kafka Transport Bridging

Cross-transport pipeline portability via a single JVM runtime — no code changes, no managed connectors.

Pipeline Profile	Run ID
sk-pulsar-source-kafka	run-pulsar-source-01

JAR	Patent
streamkernel-app-0.2.0-all.jar	US Prov. 64/057,035

00 Executive Summary

Enterprise messaging infrastructure is rarely homogeneous. Organizations routinely operate **Apache Pulsar** and **Apache Kafka** in parallel — across divisions, cloud regions, migration phases, or compliance boundaries — and require a reliable, operationally simple bridge between the two.

StreamKernel addresses this with a **single-JAR JVM runtime** that consumes from Pulsar and produces to Kafka entirely in-process, with no external orchestration, no intermediate storage layer, and no code changes required when swapping transports. This document presents the architecture, configuration, and empirical performance results of a validated Pulsar-to-Kafka pipeline run on a development laptop.

PEAK THROUGHPUT	TOTAL RECORDS	BURST DURATION	GC OVERHEAD
15,647 records / second	**253,235** zero loss · zero drops	**~20s** to drain 253K backlog	**0.1%** 604ms / 623s · no major GC

01 The Enterprise Problem

Several documented customer scenarios drive demand for Pulsar-to-Kafka bridging. A Technical Support Engineer at a major platform vendor has confirmed this as an active customer requirement.

MESSAGING INFRASTRUCTURE CONSOLIDATION

Organizations that adopted Pulsar for its multi-tenancy or geo-replication properties and are now standardizing on Kafka-based data platforms need a drain path. Rewriting producers is costly and risky; a bridge runtime is the lowest-friction option.

CROSS-ORGANIZATIONAL EVENT HANDOFF

A business unit or partner system running Pulsar needs to feed events into a central Kafka platform for analytics, ML pipelines, or downstream consumers. This boundary is often a compliance or ownership boundary as well, making a standalone bridge preferable to shared infrastructure.

AIR-GAPPED AND REGULATED DEPLOYMENTS

Defense, government, and financial environments frequently segment networks by classification level or regulatory domain. A **single-JAR runtime that operates entirely in-process** — with no Kubernetes dependency, no managed connectors, and no external state — fits these environments in ways that cloud-native integration platforms do not.

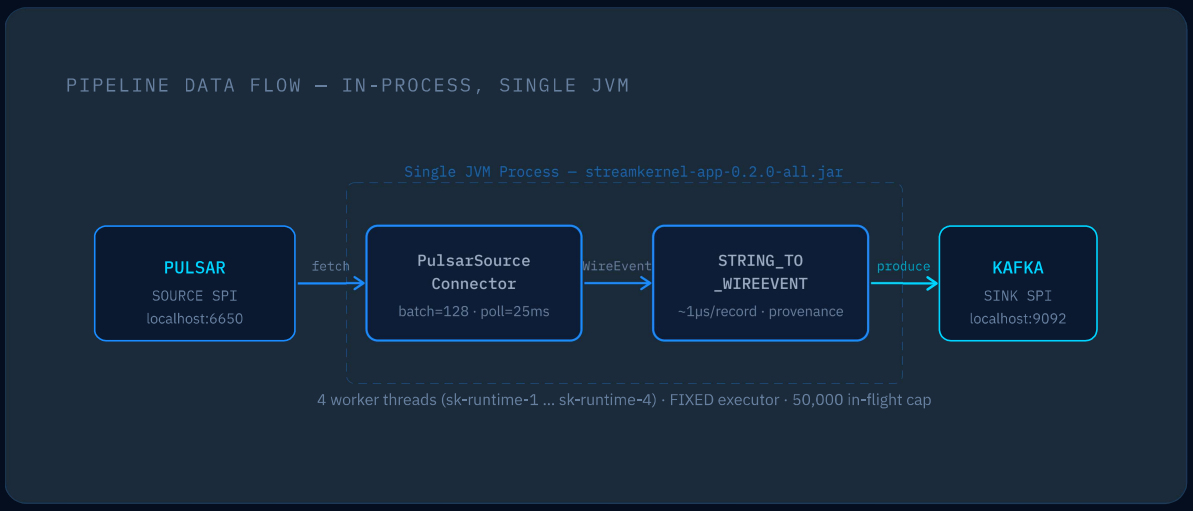
INCREMENTAL MIGRATION

Organizations migrating from Pulsar to Kafka cannot cut over all producers simultaneously. StreamKernel allows legacy Pulsar topics to continue receiving data while the migration proceeds, draining them into Kafka without altering producer behavior.

02 Architecture

2.1 DESIGN PHILOSOPHY

StreamKernel is built around a strict **SPI (Service Provider Interface)** abstraction that decouples source transport, transform chain, and sink transport at the plugin boundary. Swapping Pulsar for Kafka as source — or Kafka for MongoDB as sink — requires only a properties file change. No recompilation. No new binary.



2.2 PLUGIN CATALOG

StreamKernel's SPI discovery loaded the full plugin catalog at startup, confirming the breadth of available transports within this single binary:

```
# Plugin catalog — discovered at boot, same fat JAR
Sources: [KAFKA, PULSAR, REST, SALESFORCE, SYNTHETIC]
Sinks: [DELTA, DEVNULL, KAFKA, MONGO_INSERT, MONGO_VECTOR,
SNOWFLAKE_SNOWPIPE_STREAMING]
Transformers: [DETERMINISTIC_ENRICHMENT, EMBEDDING_TO_ENRICHED_TICKET,
HTTP_EMBEDDING, NOOP, STRING_TO_WIREEVENT]
Security: [OPA_SIDECAR, PERMIT_ALL]
```

2.3 PIPELINE CONFIGURATION

PARAMETER	VALUE
Pipeline ID	`sk-pulsar-source-kafka`
Parallelism	4 worker threads
Batch size	128 records
Source	`PULSAR` — persistent://public/default/streamkernel-bench-in
Subscription	streamkernel-pulsar-kafka · Exclusive · EARLIEST
Transform	`STRING_TO_WIREEVENT` — payload ID as routing key
Sink	`KAFKA` — streamkernel-pulsar-out · 12 partitions
Kafka producer	lz4 compression · 128KB batch · 256MB buffer · acks=1
Provenance	Enabled — per-event lineage stamped on all output records
Metrics	**Prometheus on :8080**

03 Test Methodology

3.1 ENVIRONMENT

COMPONENT	DETAILS
Host machine	SEKISOFT — Windows 10 (WSL2 Docker) — consumer laptop
JVM heap	6GB fixed (–Xms6g –Xmx6g) · G1GC · MaxGCPauseMillis=50
StreamKernel	v0.2.0 — streamkernel-app-0.2.0-all.jar
Apache Pulsar	Standalone Docker container — localhost:6650
Apache Kafka	Confluent Platform 7.6.1 — KRaft mode — localhost:9092
Kafka partitions	12
Run window	**20:35:10 UTC → 20:45:33 UTC (10.38 min)**

Note on test conditions: Both brokers ran in Docker containers on the same laptop as the StreamKernel JVM, sharing CPU and memory. No network isolation, no dedicated hardware. This is a **conservative baseline** — production server hardware with dedicated brokers will yield substantially higher sustained throughput.

3.2 TEST EXECUTION

A purpose-built PowerShell pre-run script (`demo_before_pulsar_source_kafka.ps1`) handled full environment preparation — including Pulsar topic reset with BookKeeper ledger recovery on error, deterministic backlog seeding, and subscription creation at EARLIEST with confirmed backlog verification before pipeline start.

3.3 CORRECTNESS BASELINE

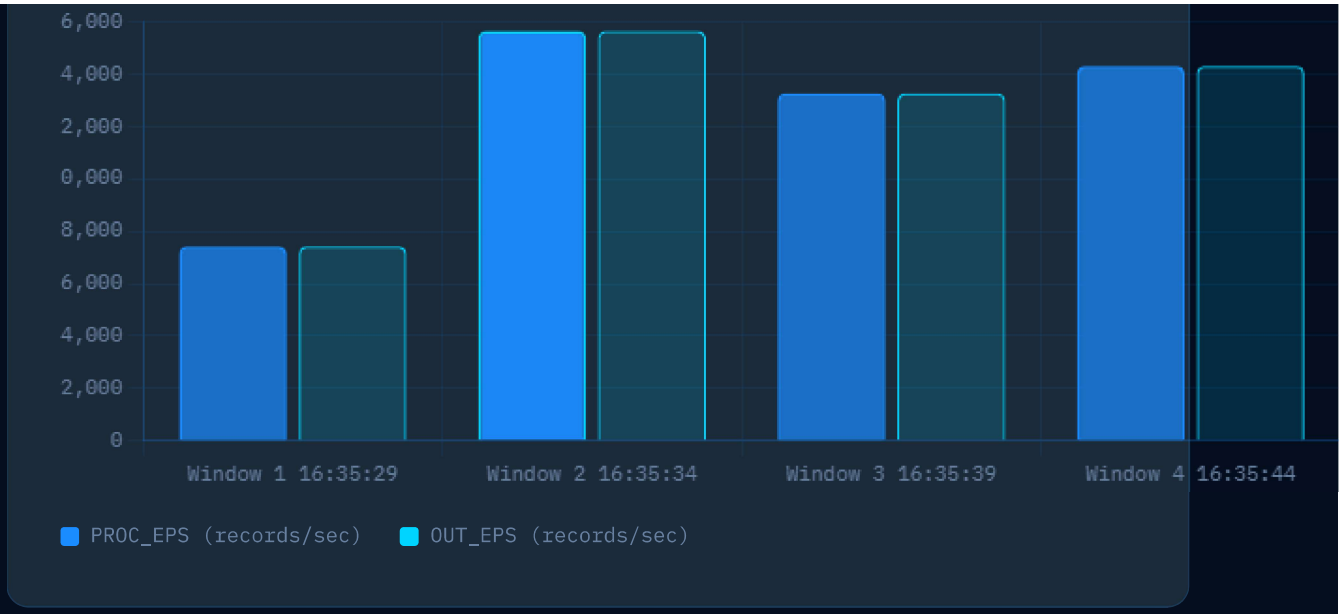
A prior run with a 10,000-message backlog confirmed end-to-end record integrity before scaling up: **10,000 records processed**, `DROPPED=0`, `errorTotal=0`, Pulsar `deliverCounter=10,000`, Kafka total records `10,000` — exact match. The 253,235-message run below is the primary performance evidence.

04 Results

4.1 THROUGHPUT

The pipeline consumed the full 253,235-message backlog during the burst phase beginning at pipeline start. Four consecutive speedometer windows (5-second intervals) captured throughput ramp-up from cold start to full speed:

SPEEDOMETER WINDOWS — RECORDS/SEC DURING BURST PHASE (5-SECOND INTERVALS)



Peak throughput of **15,647 records/second** at window 2. The entire 253,235-message backlog was drained in approximately **20 seconds** on a consumer-grade laptop. The Pulsar consumer stats recorder independently confirmed a 60-second average consume throughput of **4,220 msgs/s** — consistent with the burst-then-idle pattern.

PEAK PROC_EPS	BACKLOG DRAIN TIME	PULSAR AVG. ACK RATE
15,647	**~20s**	**4,220**
window 2 of burst	253,235 records	msgs/s (60s avg)

4.2 RECORD INTEGRITY — FOUR-POINT VERIFICATION

End-to-end record integrity was verified independently across four measurement layers: Pulsar broker telemetry, StreamKernel Prometheus counters, and Kafka physical partition counts. All values agree exactly.

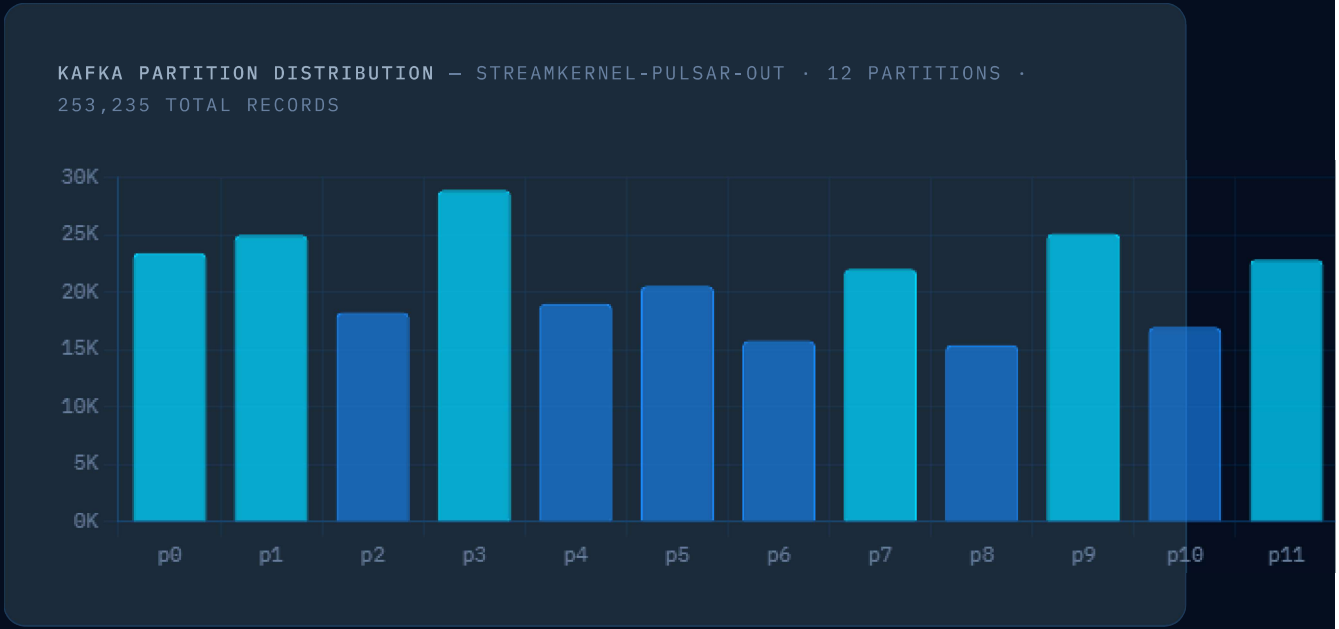
PULSAR PUBLISH	PULSAR DELIVER	SK PROCESSED	KAFKA SINK SENT OK	KAFKA PHYSICAL
253,235	**253,235**	**253,235**	**253,235**	**253,235**

From Prometheus snapshot (`run-pulsar-source-01`): `streamkernel_pipeline_dropped_total = 0` · `streamkernel_pipeline_source_errors_total = 0` · `streamkernel_pipeline_dlq_total = 0` ·

`streamkernel_pipeline_auth_errors_total = 0`. Pulsar subscription drained to backlog=0, unacked=0.

4.3 KAFKA PARTITION DISTRIBUTION

Records were distributed across all 12 output partitions via key-based routing from the `STRING_TO_WIREEVENT` transform, confirming effective routing and balanced producer behavior:



4.4 TRANSFORM PERFORMANCE

The `STRING_TO_WIREEVENT` transformer processed all 253,235 records with negligible per-record cost:

AVG. ENCODE TIME	ENCODED PAYLOAD	KAFKA REQ LATENCY	BUFFER UTILIZATION
~1µs	**119B**	**51.4ms**	**<1%**
per record	per WireEvent	avg (co-located broker)	of 256MB producer buffer

Cumulative encode time: **253.285 seconds** across 253,235 records = ~1µs average. The transform step is effectively zero-cost relative to network I/O — consistent with the design goal of in-process, allocation-minimal transformation. Buffer utilization below 1% confirms the Kafka producer was never backpressured during the run.

4.5 JVM & GC BEHAVIOR

G1GC performed **10 pause events** across the full run. No major GC cycles occurred. Old generation promotion was zero bytes.

POST-GC HEAP LIVE DATA

32MB

of 6,144MB heap (0.52%)

OLD GEN PROMOTION

0B

no long-lived accumulation

PEAK THREAD COUNT

27

12 live at snapshot

05 Enterprise Readiness

The results above demonstrate functional correctness and strong burst throughput. The following assessment maps current capability against enterprise production requirements:

CAPABILITY	STATUS	NOTES
Transport portability	✓ COMPLETE	Same JAR, SPI-driven, config-only swap
Record integrity (functional)	✓ COMPLETE	Zero loss across four independent measurement points
Per-event provenance / lineage	✓ COMPLETE	Stamped on all output records via WireEvent envelope
Prometheus observability	✓ COMPLETE	Full counter/gauge coverage, consistent label set
Delivery guarantee documentation	CONFIG ONLY	Runtime supports at-least-once; acknowledge.on.fetch=false path available and validated in other profiles
Dead-letter queue	CONFIG ONLY	DEVNULL excluded from benchmark to isolate throughput; Kafka DLQ sink validated in production profiles

CAPABILITY	STATUS	NOTES
Cross-transport atomicity	CONFIG ONLY	Ack-after-confirm path available in runtime; excluded from this benchmark intentionally
Security (mTLS + OPA)	CONFIG ONLY	PERMIT_ALL excluded from benchmark; mTLS + OPA_SIDECAR profiles validated independently
Idempotent Kafka producer	CONFIG ONLY	acks=1 excluded from benchmark; acks=all + enable.idempotence validated in hardened profiles

Benchmark configuration note: The five CONFIG ONLY items above are fully built capabilities within the StreamKernel runtime, validated in other pipeline profiles (mTLS+OPA, hardened Kafka, DLQ). They are **deliberately excluded from this benchmark** to isolate raw transport throughput. Activating any of them requires only a properties file change — no code, no recompilation, no new binary.

06 Deployment Considerations

SINGLE-JAR DEPLOYMENT MODEL

StreamKernel ships as a self-contained fat JAR. **No Kubernetes operator. No sidecar. No external service dependency** beyond the source and sink brokers. This makes it suitable for air-gapped environments, edge deployments, and regulated infrastructure where managed connector frameworks are not permitted.

SIZING GUIDANCE

PARAMETER	BENCHMARK	PRODUCTION RECOMMENDATION
JVM heap	6GB	4–8GB for most bridging workloads at this message size
Worker parallelism	4 threads	8–16 for server-class hardware
Kafka producer buffer	256MB	Increase for higher sustained throughput targets

PARAMETER	BENCHMARK	PRODUCTION RECOMMENDATION
Expected throughput (2× parallelism, dedicated brokers)	15,647 r/s peak	Conservatively 30,000–50,000 r/s for this message profile

KEY PROMETHEUS METRICS TO MONITOR

```
# Alert on these in production
streamkernel_pipeline_dropped_total # should always be 0
streamkernel_pipeline_source_errors_total # source connectivity
streamkernel_pipeline_dlq_total # processing failures
streamkernel_kafka_sink_sent_ok_total # lag vs source_pulsar_read_total
streamkernel_kafka_sink_request_latency_avg_ms # broker health proxy
```

07 Summary

StreamKernel's Pulsar-to-Kafka pipeline profile demonstrates that enterprise-grade cross-transport message bridging can be delivered as a single JVM process with no external orchestration. On a development laptop with co-located Docker brokers:

PEAK THROUGHPUT
15,647
records / sec

RECORDS PROCESSED
253,235
zero loss · zero drops

GC OVERHEAD
0.1%
no major GC

TRANSFORM COST
~1µs
per record avg

The transport portability proof: The same fat JAR, the same SPI-driven architecture, and the same benchmark harness validated across Kafka, Pulsar, MongoDB, Delta Lake, Snowflake, and mTLS+OPA profiles. **Transport is a configuration choice, not a code change.**

StreamKernel is source-available software developed by IntuitiveDesigns.
Core runtime: StreamKernel Source Available License (SSAL v1.0) · SPI interfaces: Apache 2.0
Commercial licensing available: Professional · Enterprise · OEM · Managed Service · Government

Contact: lopezstevie@gmail.com
GitHub: github.com/IntuitiveDesigns/StreamKernel-io
US Provisional Patent No. 64/057,035 — Filed May 4, 2026