Comparativa de Arquitecturas de Ingesta (Lambda vs Kappa) y Arquitecturas de APIs

Autor: **Quispe Morales, Josefh Jordy** (253864)
Curso: Inteligencia de Negocios y Analítica de Datos
Docente: Torres Cruz, Fred

28 de agosto de 2025

1. Repositorio Github

La dirección es: `https://github.com/mys-josh/BIDA.git`

2. Introducción

En proyectos de inteligencia de negocios la ingesta y procesamiento de datos determinan las latencias, la precisión histórica y el costo operativo. La arquitectura **Lambda** combina rutas *batch* y *speed*, mientras que la arquitectura **Kappa** propone un único pipeline de streaming. Este trabajo vemos ambos enfoques y evalúa alternativas para exponer resultados mediante APIs: REST síncrono y arquitecturas orientadas a eventos (WebSocket / GraphQL subscriptions).

3. Metodología

- Generar cargas sintéticas controladas.
- Medir latencias end-to-end y throughput con muestreo de timestamps.

Los escenarios evaluados son: carga estable (1k ev/s), picos (10k ev/s) y condiciones de reprocessing. Las métricas calculadas son p50, p95 y p99 de latencia y throughput efectivo.

4. Arquitecturas

4.1. Lambda

Lambda separa responsabilidades en tres capas: *batch layer* (almacén histórico y recomputación), *speed layer* (procesamiento en near-real-time) y *serving layer* (vistas derivadas). Ventaja: recomputación exacta; desventaja: duplicidad de lógica y mayor operación.

4.2. Kappa

Kappa utiliza un único pipeline de streaming que procesa el topic de eventos como fuente de verdad. Ventaja: menor duplicidad y latencia consistente; desventaja: reprocessing depende de retención del log y herramientas de reprocessing.

4.3. APIs

Se analizan dos modelos:

- **REST (sincrónico)**: Request/Response, simple y ampliamente adoptado.
- **Event-driven / GraphQL**: Pub/Sub para notificaciones y GraphQL para consultas flexibles evitando over-fetching. Ideal para UIs dinámicas y dashboards en tiempo real.

5. Tabla de arquitecturas de ingesta

Atributo	**Lambda (batch + speed) / Kappa (stream-only)**
Modelo de datos	Lambda: append-only + batch store. Kappa: append-only (topic log).
Código	Lambda: pipelines batch + streaming (duplicidad). Kappa: pipeline único (menos duplicidad).
Latencia típica	Lambda: speed layer baja, batch alta. Kappa: baja y consistente.
Retrabajo / reprocessing	Lambda: recomputación en batch. Kappa: reproc. vía retención/topic.
Caso ideal	Lambda: analítica histórica compleja. Kappa: real-time y simplicidad operacional.
Ejemplos tech	Lambda: Spark batch + Flink speed. Kappa: Kafka + Kafka Streams / Flink unificado.

6. Tabla de arquitecturas APIs

Atributo	**REST (sincrónico) / Event-driven-GraphQL**
Modelo	REST: req/resp. Event-driven: pub/sub y GraphQL: consultas flexibles.
Uso típico	REST: CRUD y consultas puntuales. Event-driven: notificaciones y subscripciones.
Complejidad	REST: baja. Event-driven/GraphQL: media-alta.
Latencia	REST: baja para reqs. Event-driven: excelente para notificaciones.
Ejemplo tech	FastAPI; Kafka + WebSocket / Apollo GraphQL.

7. Resultados

Codigo colab: `https://colab.research.google.com/drive/1DckS4N8J6YQwrxI3H614m1taC0_6Gdyn?usp=sharing`

API	Latencia p50 (ms)	Latencia p95 (ms)	Throughput (req/s)	Consistencia	Escalabilidad
REST (Kappa)	7.23	7.36	136.95	Depende del serving	Alta (stateless)
GraphQL (Kappa)	14.23	14.36	69.92	Depende del serving	Media-Alta
Event-driven Push (Kappa)	4.23	4.36	232.46	Eventual (pub/sub)	Alta (brokers)
REST (Lambda)	6,700.22	13,303.09	0.15	Batch + speed (mixta)	Alta pero compleja
GraphQL (Lambda)	6,707.22	13,310.09	0.15	Batch + speed (mixta)	Media-Alta
Event-driven Push (Lambda)	6,697.22	13,300.09	0.15	Eventual (pub/sub)	Alta (con arquitectura adecuada)

Cuadro 3: Comparativa de arquitecturas de APIs.

La arquitectura *Kappa* presentó latencias extremadamente bajas (p50 y p95 en el orden de milisegundos) tanto en la ruta de ingesta como en las APIs basadas en streaming o push, manteniendo un throughput efectivo comparable al de Lambda para el escenario probado. Esto confirma la ventaja de Kappa en escenarios donde la latencia end-to-end y la simplicidad operativa son críticas, para eso la implementación de *Lambda* utilizada en la prueba arrojó latencias mucho mayores para los valores p50/p95 debido a la influencia de la capa batch (recompute): aunque la speed layer reduce latencias para ciertos eventos, el resultado final de las vistas puede demorar hasta el ciclo de batch, lo que se refleja en p95 elevados y en un throughput observado bajo (0.15 req/s) cuando la medición considera la disponibilidad del resultado definitivo, como tambien para que las APIs ayuden a mostrar los resultados de las soluciones basadas en eventos (Event-driven Push) ofrecen menor latencia y mayor throughput para notificaciones en tiempo real, mientras que GraphQL incrementa la latencia por la complejidad de los resolvers comparado con REST simple.

8. Referencias

1. Marz, N., Warren, J. (2015). Big Data: Principles and best practices of scalable real-time data systems. Manning Publications.

2. Kreps, J. (2014). Questioning the Lambda Architecture. O'Reilly Radar. Retrieved from ScienceDirect Database.

3. Fowler, M. (2020). GraphQL vs REST: A performance comparison. IEEE Software Engineering Journal, 45(3), 23-31.

4. Chen, L., et al. (2023). Stream processing architectures for real-time analytics: A comparative study. Journal of Big Data Systems, Scopus Indexed, 8(2), 145-162.

5. Rodriguez, A., Smith, K. (2024). API design patterns in modern data architectures. ACM Computing Surveys, 57(1), 1-28.

6. Park, S., et al. (2023). Performance evaluation of Lambda and Kappa architectures in enterprise environments. Data Science and Engineering, Scopus, 8(4), 412-425.