APACHE KAFKA, APACHE PULSAR, DATA AT REST, DATA IN MOTION, DATA LAKE, DATA WAREHOUSE, EVENT STREAMING, KAPPA ARCHITECTURE, LAMBDA ARCHITECTURE

Kappa Architecture is Mainstream Replacing Lambda

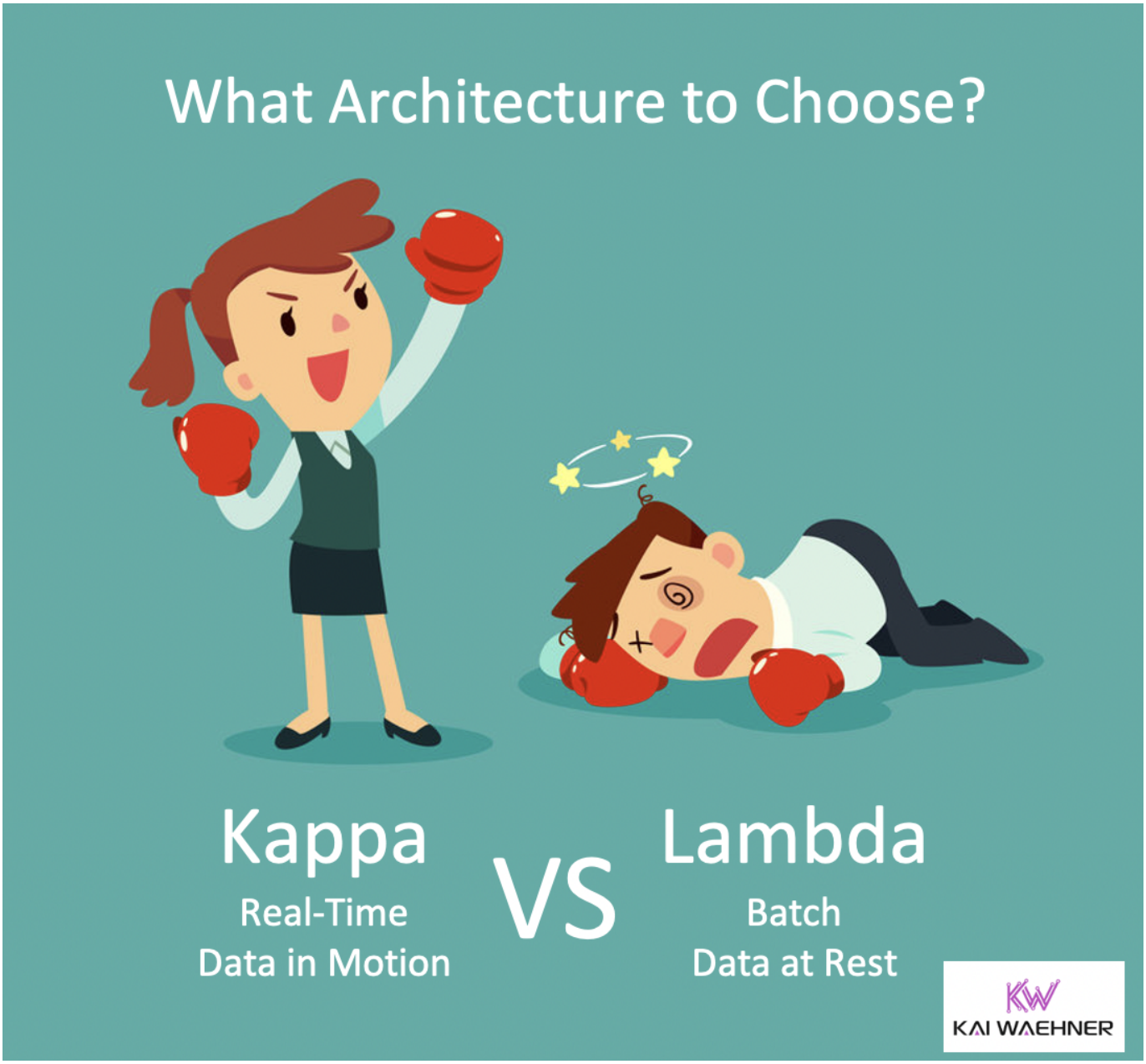
17 minute read

Real–time data beats slow data. That's true for almost every use case. Nevertheless, enterprise architects build new infrastructures with the Lambda architecture that includes separate batch and real–time layers. This blog post explores why a single real–time pipeline, called Kappa architecture, is the better fit. Real–world examples from companies such as Disney, Shopify, Uber, and Twitter explore the benefits of Kappa but also show how batch processing fits into this discussion positively without the need for Lambda.

By  KAI WAEHNER · 23. September 2021

Real–time data beats slow data. That's true for almost every use case. Nevertheless, enterprise architects build new infrastructures with the Lambda architecture that includes separate batch and real–time layers. This blog post explores why a **single real–time pipeline, called Kappa architecture**, is the better fit. Real–world examples from companies such as **Disney, Shopify, Uber, and Twitter explore the benefits** of Kappa but also show how batch processing fits into this discussion positively without the need for Lambda.

This post is heavily inspired by Jay Kreps' article "Questioning the Lambda Architecture" from 2014 (!) and maps his thoughts to the real–world situation in 2021. Today, almost every business solution, data storage and analytics provider, and business application leverages event streaming and asynchronous, truly decoupled event–based communication paradigms for data processing. For that reason, many move from Lambda to Kappa architectures.



A Modern Enterprise Architecture

A modern enterprise architecture offers cloud–native characteristics: Flexibility, elasticity, automation, true decoupling between different applications, and real–time capabilities (where needed).

Microservices, Data Mesh, and Domain-driven Design for True Decoupling

Let's quickly explore the buzzwords to understand how most people build modern enterprise architectures today:

- **Domain-driven Design (DDD)** enforces strict boundaries between service communication and a decentralized application landscape.
- **Microservices** enable building flexible, decoupled applications with different programming languages and communication paradigms.
- **Data Mesh** allows to architect services around data. Data is the product in a data mesh. Self-service capabilities and federation enable business units to focus on their business problem.

My blog post "Microservices, Apache Kafka, and Domain-Driven Design" explored this discussion in more detail (even though the buzzword "data mesh" did not exist at the time of writing). TL;DR: **An event-driven streaming infrastructure such as Apache Kafka uniquely enables proper decoupling and real-time data processing** (contrary to traditional web service / REST / HTTP-based microservice architectures and contrary to traditional messaging systems (MQ, ESB). The blog post about Kafka vs. MQ/ETL/ESB might also be helpful to learn more.

Real-time Data Beats Slow Data, but NOT Always!

Think about your industry, business units, problems you solve, and innovative new applications you build. **Real-time data beats slow data**. This statement is almost always true. Either to increase revenue, reduce cost, reduce risk, or improve the customer experience.

Data at Rest means to store data in a database, data warehouse, or data lake. This way, **data is processed too late in many use cases** — even if a real-time streaming component (like Kafka) ingests the data. The data processing is still a web service call, SQL query, or map-reduce batch process away from providing a result to your problem.

Don't get me wrong. **Data at rest is not a bad thing**. Several use cases such as **reporting (business intelligence), analytics (batch processing), and model training (machine learning) work very well with this approach**. But real-time beats batch in almost all other use cases.

I analyzed the relation between data at rest and data in motion and how this point of view regarding the enterprise architecture changed with the cloud-first strategy of most companies in the blog post "Serverless Kafka in a Cloud-native Data Lake Architecture".

The de facto standard for real-time data processing is Apache Kafka. Hence, the covered real-world examples in this post use Kafka.

With this context in mind, let's revisit Lambda architecture.

The Lambda Architecture

Nathan Marz coined the Lambda architecture: A data–processing architecture designed to **handle massive quantities of data by taking advantage of both batch and stream–processing methods**.

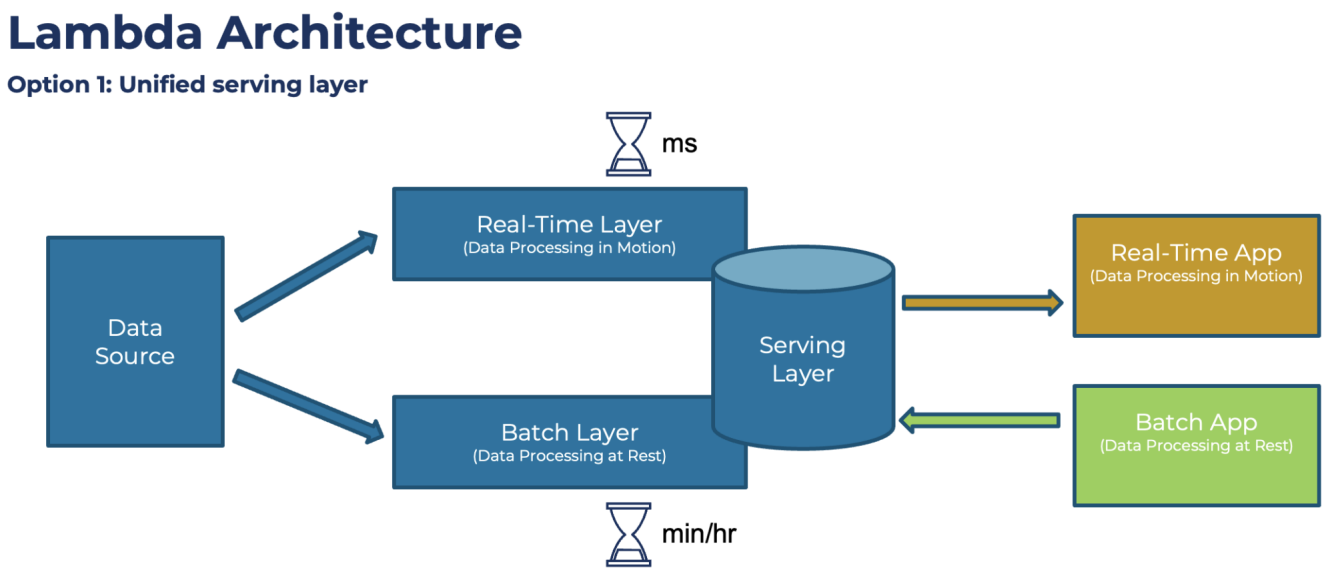
Lambda architecture includes **batch, speed, and serving layers**. This approach enables processing data in real–time but also easy re–processing of batched static datasets. The problem with out–of–order data is also solved.

This approach attempts to balance latency, throughput, and fault–tolerance by using batch processing to provide comprehensive and accurate views of batch data while simultaneously using real–time stream processing to provide views of online data. **The rise of lambda architecture is correlated with the steady growth of big data, real–time analytics, and the drive to mitigate the latencies of map–reduce.**

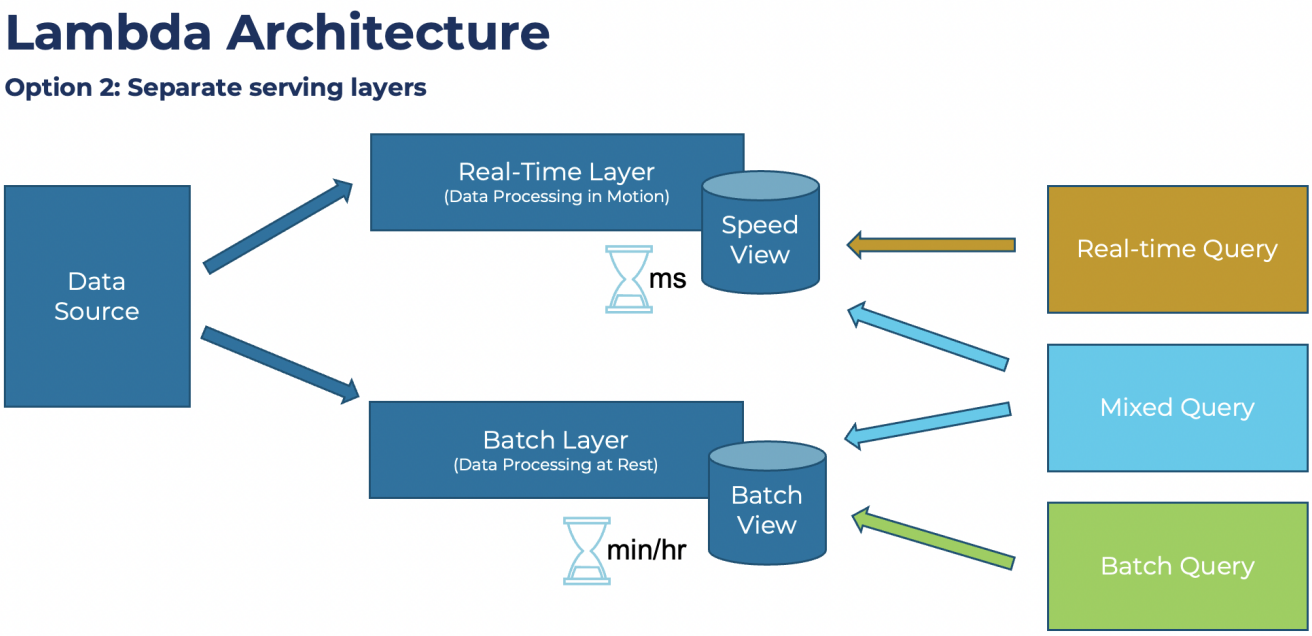
Two Options for a Lamba Architecture

The web discusses **two different approaches** to Lambda architecture.

The initial approach provided a **unified serving layer**. A unified serving layer joins the real–time and batch layer:



Another alternative is **two separate serving layers**. One layer is for real–time consumption, the other one for batch consumption:

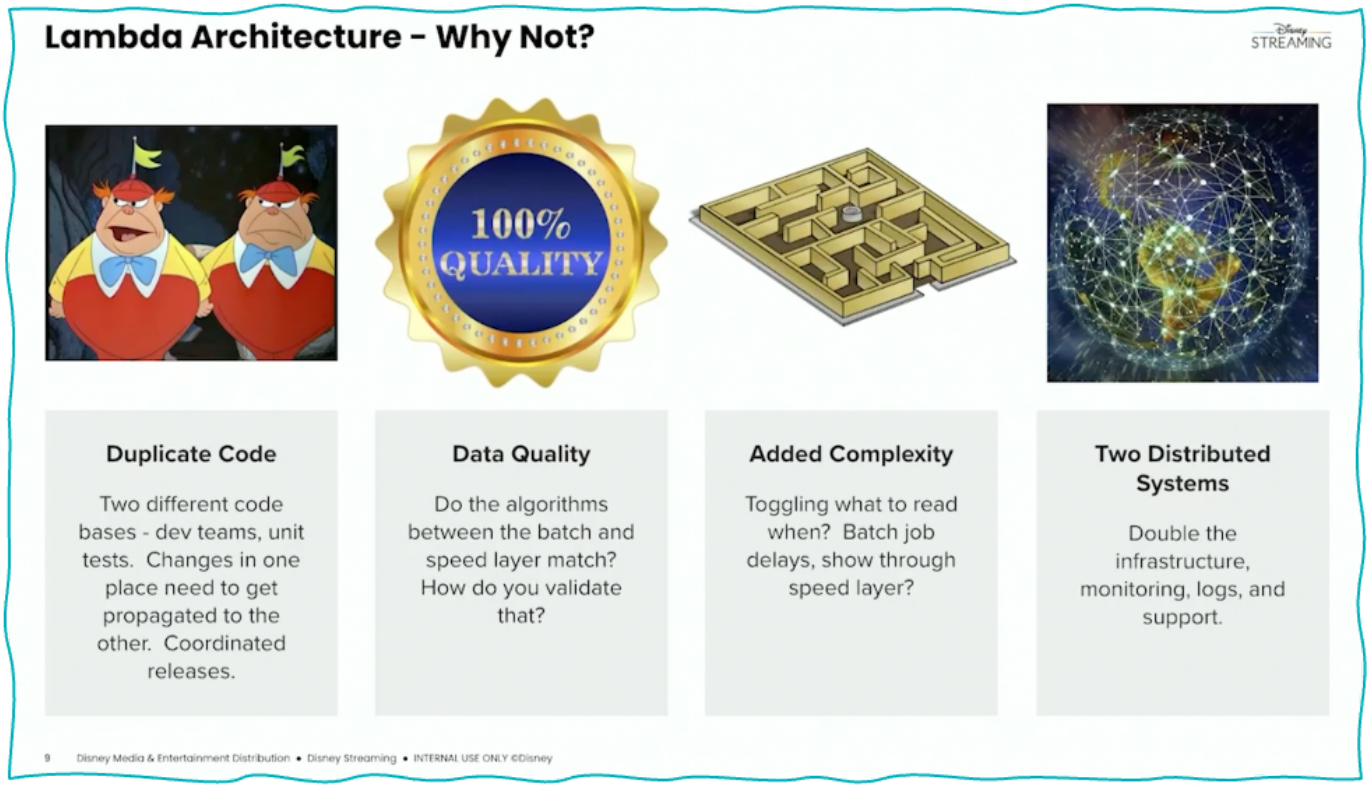


I see the second option much more in the field. In the end, both have the same concept of building two separate layers for data ingestion and processing.

Issues with the Lambda Architecture

The **Hadoop vendors heavily pitched the Lambda architecture to deploy and operate a super complex infrastructure with many big data frameworks**. Today, I only hear the pain of enterprises complaining about this complexity and the missing business value. No surprise that most of these vendors did not survive or have a very confusing and unclear future product strategy.

Disney has summarized the concerns with the Lambda architecture on one slide:



The **batch and streaming sides each require a different codebase** that must be maintained and kept in sync so that processed data produces the same result from both paths. Additionally, with batch, speed, and serving layers, everything needs to be processed (at least) twice. That increases the cost and operations efforts of storage, network, and compute.

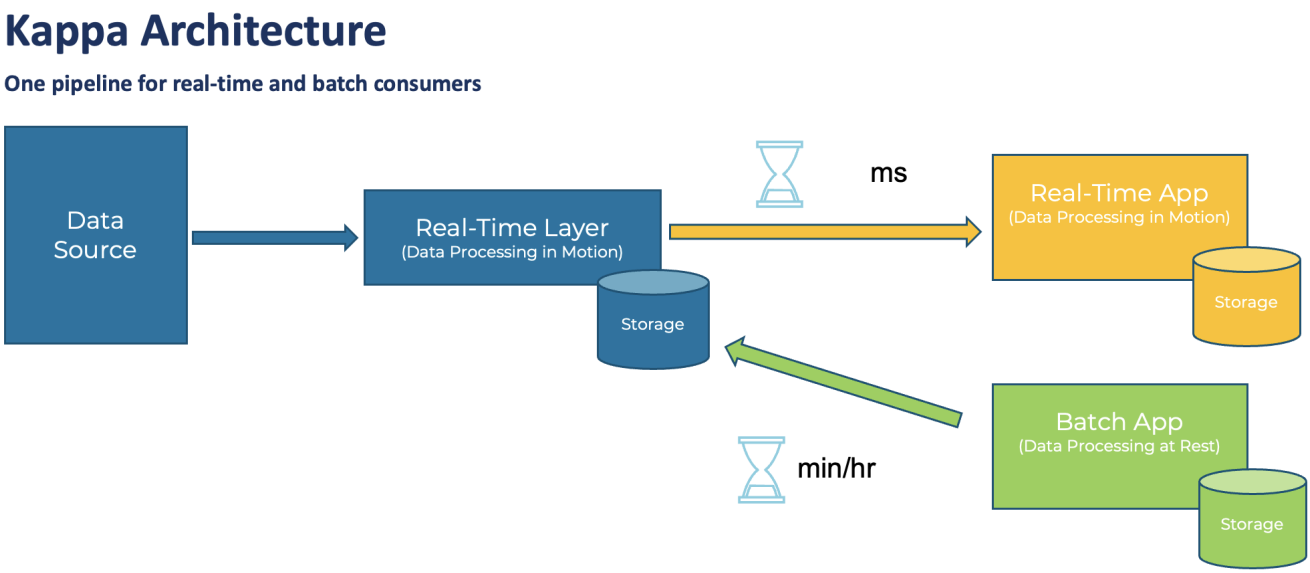
Jay Kreps had similar arguments when he proposed the Kappa architecture in 2014 (!), already: "The problem with the Lambda Architecture is that maintaining code that needs to produce the same result in two complex distributed systems is exactly as painful as it seems like it would be".

So, what's different in Kappa architecture?

The Kappa Architecture

The **Kappa architecture is a software architecture that is event–based and able to handle all data at all scale in real–time for transactional AND analytical workloads**.

The central premise behind the Kappa architecture is that you can perform both real–time and batch processing with a **single technology stack**. The heart of the infrastructure is streaming architecture. First, the event streaming platform log stores incoming data. From there, a stream processing engine processes the data continuously in real–time or ingests the data into any other analytics database or business application via any communication paradigm and speed, including real–time, near real–time, batch, request–response.



Unlike the Lambda Architecture, in this approach, you only do re–processing when your processing code changes, and you need to recompute your results. And, of course, the job doing the re–computation is just an improved version of the same code, running on the same framework, taking the same input data.

Benefits of the Kappa architecture

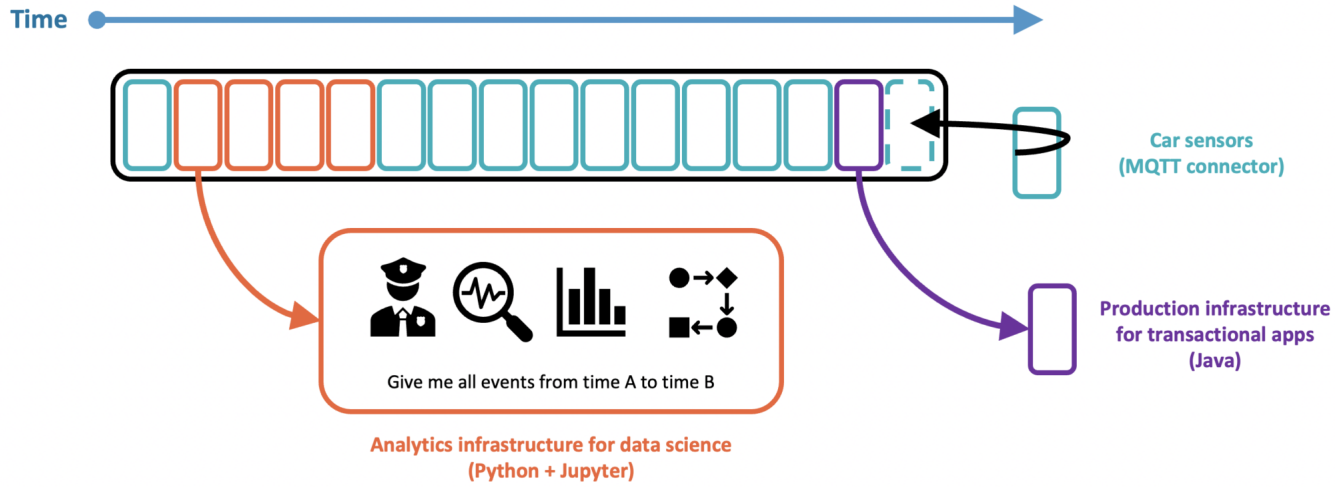
The Kappa architecture has several benefits:

- Handle all the use cases (streaming, batch, RPC) with a single architecture
- One codebase that is always in synch
- One set of infrastructure and technology
- The heart of the infrastructure is real–time, scalable, and reliable
- Improved data quality with guaranteed ordering and no mismatches
- No need to re–architect for new use cases

TL;DR: **The Kappa architecture leverages a single source of truth focusing on simplicity in the enterprise architecture**. People can develop, test, debug, and operate their systems on a single processing framework for BOTH real–time and batch systems. To be clear: The leading system for some applications can still be another system. For instance, the leading system for ERP is still SAP, while the source of truth for consumers is the Kafka log.

Kappa for Transactional and Analytical Workloads

Contrary to a data lake, event–streaming–powered **Kappa architectures enable transactional workloads in addition to analytical workloads** too.

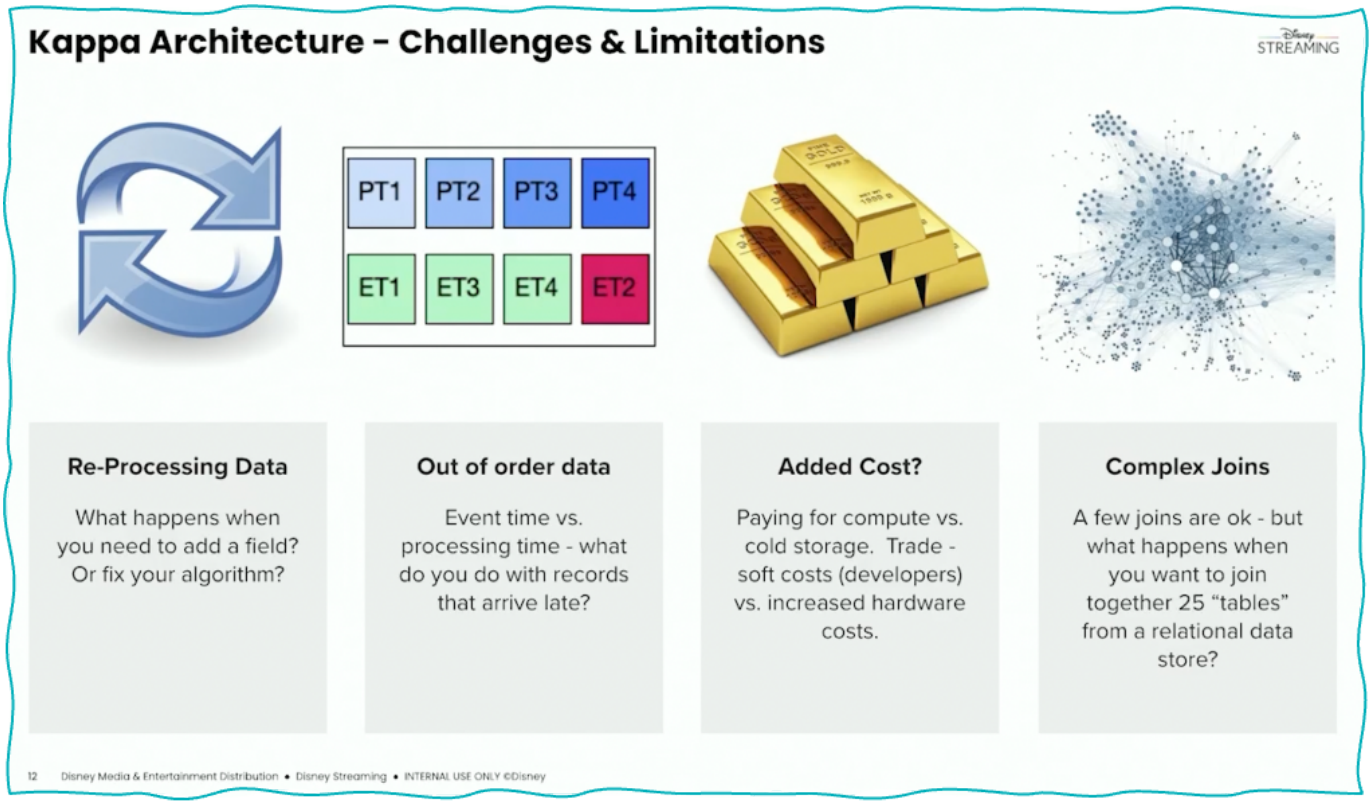


For instance, Kafka and its ecosystem support exactly–once semantics so that you can build your next payment platform for aftersales or customer interactions with mission–critical SLAs, low latency, and fault–tolerance built–in. Independently, the data science team consumes historical events for finding insights in a batch process using machine learning.

Kappa is NOT a free lunch!

The Kappa architecture sounds too good to be true? Well, a basic rule of thumb is still valid: **Use the right tool for the job!**

Event streaming is a paradigm shift. A big bang migration will not work. Here are a few lessons learned from Disney about introducing the Kappa architecture:



As a big bang does not work, a good way is to rethink data and databases. Martin Kleppmann called it "turning the database inside out". Let's look at this approach and how it helps to leverage the Kappa architecture in combination with other databases and analytics platforms.

The Inside and Outside Perspective to Solve the Kappa Challenges

Turning the database inside out is a **new thinking of the enterprise architecture**. The heart of the infrastructure is event–based and real–time. Where needed, you consume the events in batch or store them in additional storage and analytics tools with their concepts and paradigms after they consumed the events.

The inner perspective of Kappa: The central nervous system

Think of an event streaming platform like Kafka:

- **Data availability/retention:** Compacted Topics, Tiered Storage
- **Data consistency and fault–tolerance:** Exactly–once semantics, Multi–Region Clusters, Cluster Linking
- **Handling late–arriving data:** Event time and processing time are different. State management in the streaming application, proper data sinks, replay with guaranteed ordering, and timestamps.

- **Data reprocessing and backfill:** Dynamic clusters (ideally a serverless cloud offering or at least a cloud–native self–managed cluster), stateful applications (Kafka Streams, ksqlDB, external stream processing framework like Apache Flink).
- **Data integration:** Kafka Connect for sources and sinks, clients for any language, REST Proxy (real–time but also batch and RPC

An **event streaming platform provides many characteristics to built a Kappa architecture**. However, it is **not a silver bullet**. Additional databases and analytics tools are mandatory for some use cases. For instance, Kafka does not scale well for dynamic bursty workloads. Complex SQL queries and joins also need another database.

The outer perspective of Kappa: The applications and data stores

Think of any business application, data storage, or analytics platform:

- **Data Consumption**: Consume the data from the central nervous system. Consume the data at your speed (real–time, near real–time, batch, RPC).
- **Data Storage**: Store the data in your storage as long as you need it (in–memory, short–term storage, long–term storage).
- **Data Processing**: Process the data for your use case (real–time notification, indexing into your query engine, a batch process for reporting or model training, etc.). Complex processing is not doable in the event streaming platform (e.g., complex joins, intensive compute with batch algorithms).

The discussion "Can Apache Kafka be used as a database?" is also helpful to understand both perspective and the trade–offs of using Kafka as data storage.

Cost-Efficient and Scalable Kappa Architectures

A huge problem of realizing the Kappa architecture in the real world was **storing vast volumes of data in an event streaming platform**. This approach was **costly and had scalability issues** at the Terabyte or Petabyte scale. On the contrary, data lakes were designed for vast volumes from the beginning. Hadoop and HDFS were used on–premise in the early phases. The public cloud enabled the migration to fully–managed object storage such as AWS S3 or Google Cloud Storage to make data lakes even more scalable and cost–efficient for big data.

One approach is to reduce the data stored in the event streaming platform. **Infinite retention leveraging log compaction is a viable approach to reduce the storage size**. However, compacted

topics shrink data sets and only store the latest value for each message key. Hence, this workaround is not applicable for every use case.

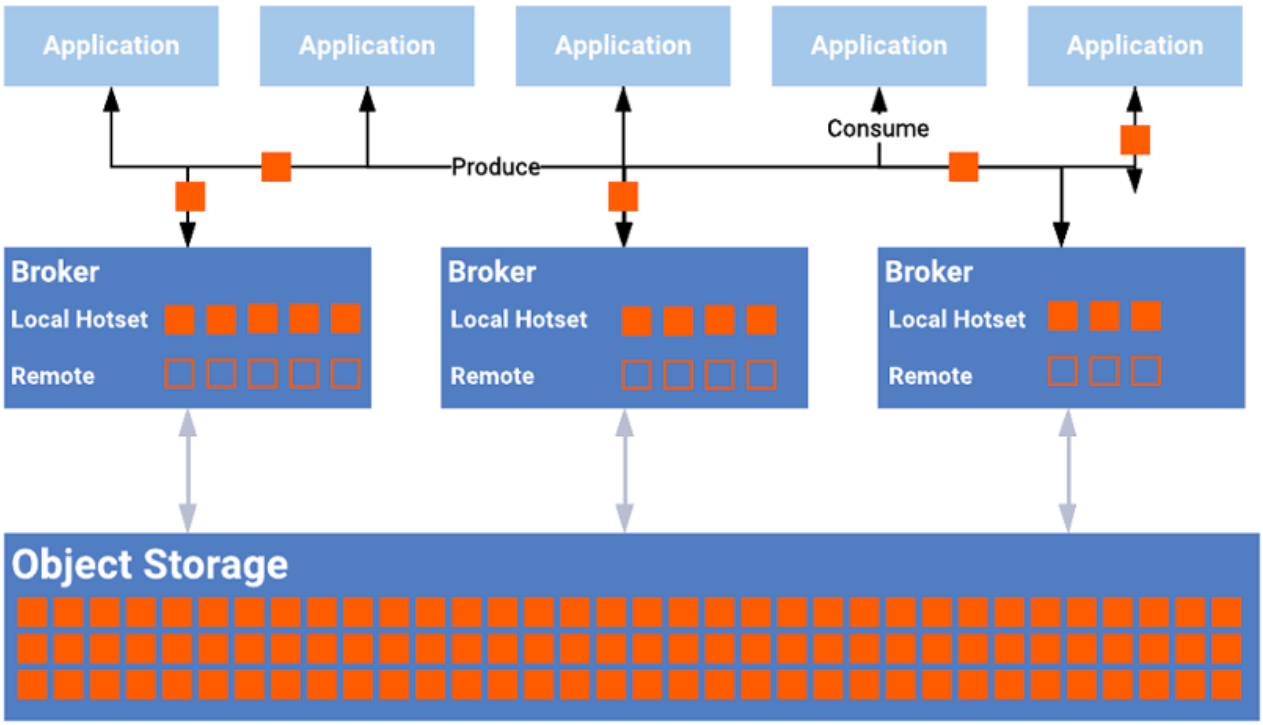
Another workaround I have seen a lot in practice is building a **"streaming data lake" with Kafka as a streaming layer and object storage for long-term storage**. The bi-directional integration was built with Kafka Connect and sink and source connectors. This was actually the main reason why Confluent built an S3 Source connector for Kafka Connect in addition to its heavily used S3 Sink connector.

Tiered Storage for Event Streaming

The good news is that streaming platforms evolved. **Tiered Storage allows decoupling storage from computing in event streaming platforms** such as Kafka or Pulsar.

Tiered Storage is a game-changer for Kappa architectures. It manages the storage without a performance impact on real-time consumers. Additionally, this enables a **very cost-efficient and elastic Kappa architecture without the need for a traditional data lake**. Uber talks about the motivation and benefits of Tiered Storage for Kafka (KIP-405) in a recent Kafka Summit talk.

Kappa architectures are very flexible regarding the underlying storage technology. While Uber uses Hadoop's HDFS as storage, Confluent went another way: Confluent Tiered Storage for Kafka is based on the S3 interface to leverage object storage and works for both public cloud provider object stores such as AWS S3 or GCS, and on-premise object stores such as PureStorage or MinIO for Kubernetes.



In other words: Tiered Storage for Kafka can leverage the same modern data storage as modern cloud data lakes (or as AWS calls it today: Lake House). Hence, the Kappa architecture provides the best of both worlds: Real–time data processing plus cost–efficient and scalable long–term storage for replaying historical data.

Real-World Examples for a Kappa Architecture

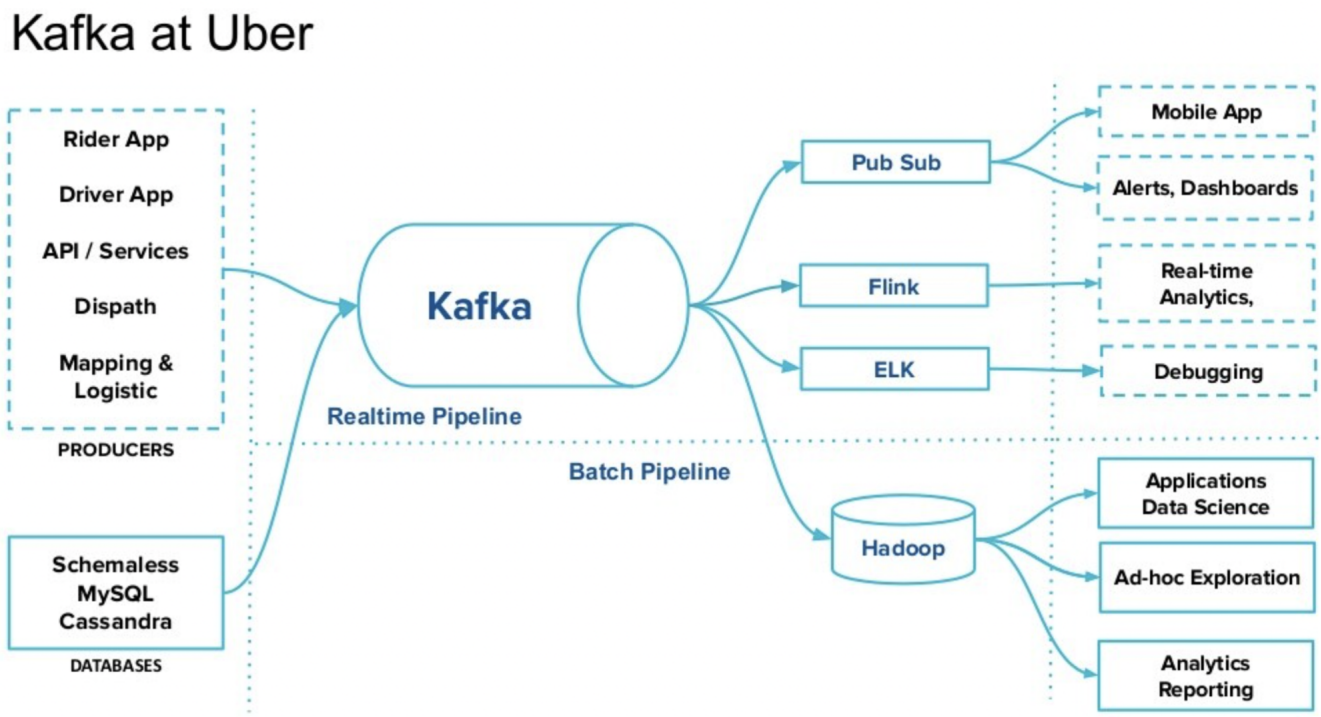
The above was a lot of theory. Let's recap: Real–time data beats slow data in most use cases. But batch processing is still needed and will not go away.

Let's now look at a few real–world examples of Kappa architectures at Uber, Shopify, and Disney.

Kappa at Uber for Trillions of Messages and Petabytes per Day

Uber is a very prominent tech giant. They talk a lot about their software architectures and deployments regularly in public. **Uber is one of the most significant Kafka users** in the world. In the meantime, they **process over 4 trillion msgs and 3PBs per day**.

As a perfect fit for this blog post, Uber presented at a recent Kafka Summit about their Kappa architecture:



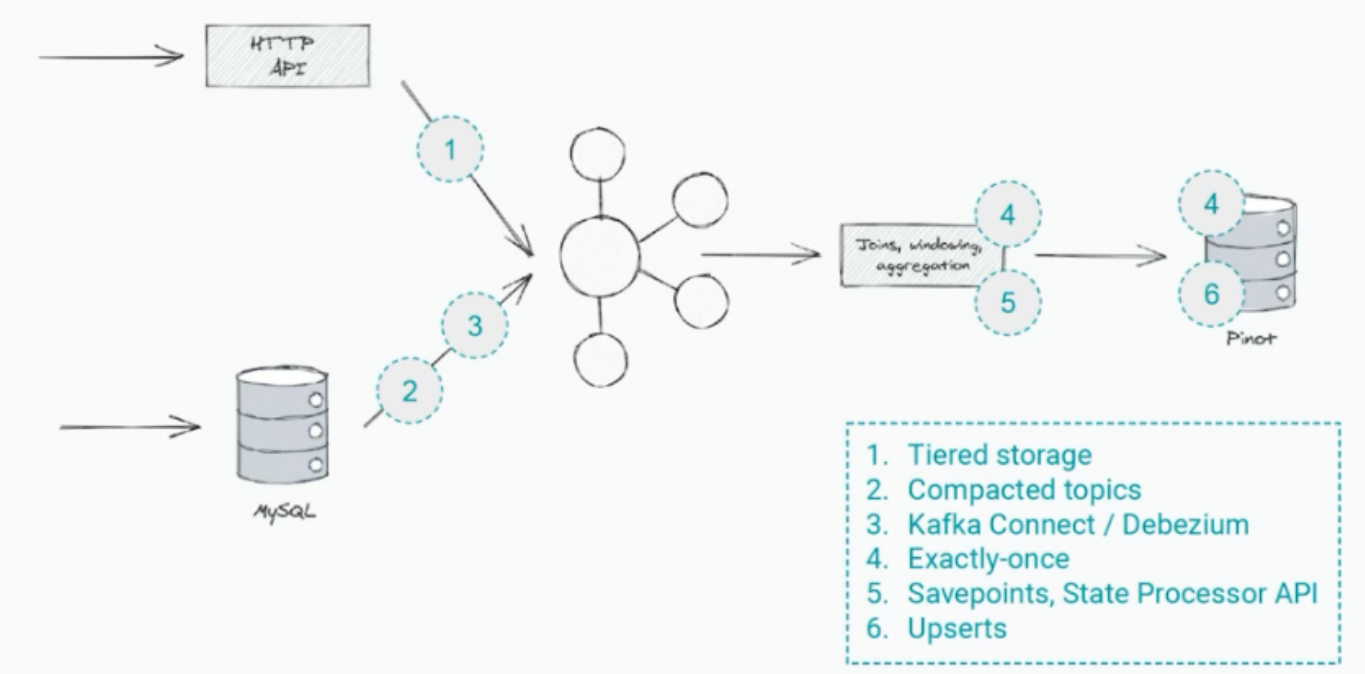
As you can see, Uber's architecture evolved precisely to what I described in the above sections. The **central nervous system is a Kafka–based real–time infrastructure**. Uber still has batch pipelines. Uber also provides APIs (e.g., to mobile apps). And — no surprise — they also have traditional SQL and NoSQL databases, business intelligence reporting tools, dashboards, and much more.

Uber's architecture shows the massive **benefits of Kappa:** The heart of the infrastructure is real–time, scalable, fault–tolerant, and reliable. **A single pipeline for everything. No need for a Lambda architecture! Kappa enables transactional and analytical workloads.** Each microservice in the data mesh can use its technology and communication paradigm for each application.

Kappa at Shopify for Stateless and Stateful Data Streaming

Shopify presented their Kappa architecture in a recent **Kafka Summit talk: "**It's Time To Stop Using Lambda Architecture**"** The session covered the concerns of Kappa architecture and how Shopify solved them with different building blocks. The three key components are the log (Kafka), streaming framework (Kafka Streams and Apache Flink), and data sinks (any real–time consumer or data store).

Here is one example of a stateful Kappa scenario at Shopify:



Shopify discussed the **core building blocks** of their Kappa architecture:

The Log (Kafka)

- Durability with Topic Compaction and Tiered Storage
- Consistency via Exactly–Once Semantics (EOS)
- Data Integration via Kafka Connect

- Elasticity via dynamic Kafka clusters

Streaming Framework (Kafka Streams / Flink)

- Reliability and scalability
- Fault tolerance
- State management

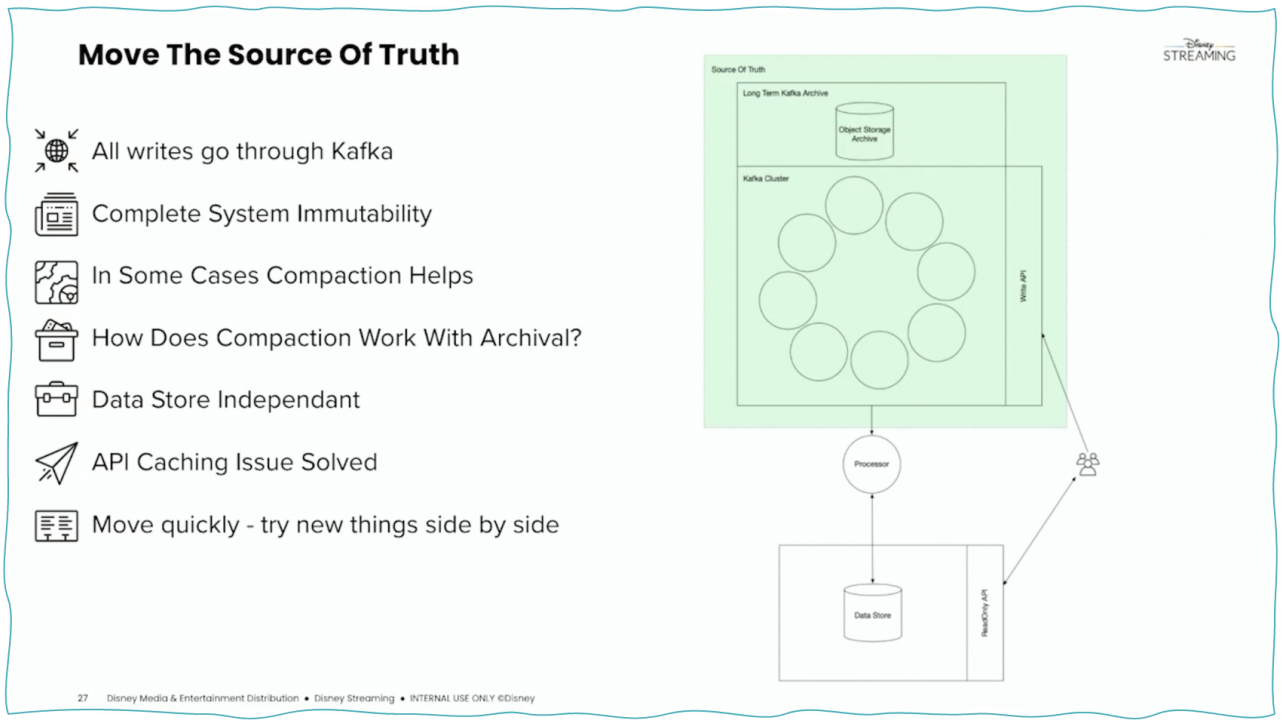
Data Sinks

- Real-time consumers
- Update/upsert for simplified design, for instance, RDBMS, NoSQL, Compacted Kafka Topics
- Append-only storage (i.e., no update), for instance, regular Kafka Topics, Time Series databases

Kappa at Disney as Single Source of Truth

Disney's Kafka Summit talk "Big Data Kappa" is very inspiring. It probably includes the most lessons learned and trade-offs of a real-world Kappa deployment. I encourage you to watch the on-demand video—many insights and guidance for building your own Kappa Architecture.

All data writes at Disney go through Kafka as the source of truth. The following screenshot shows the concept. The green box is the Kafka cluster, including Tiered Storage as the single source of truth. Any application consumes the data from Kafka for further processing and optional external storage.

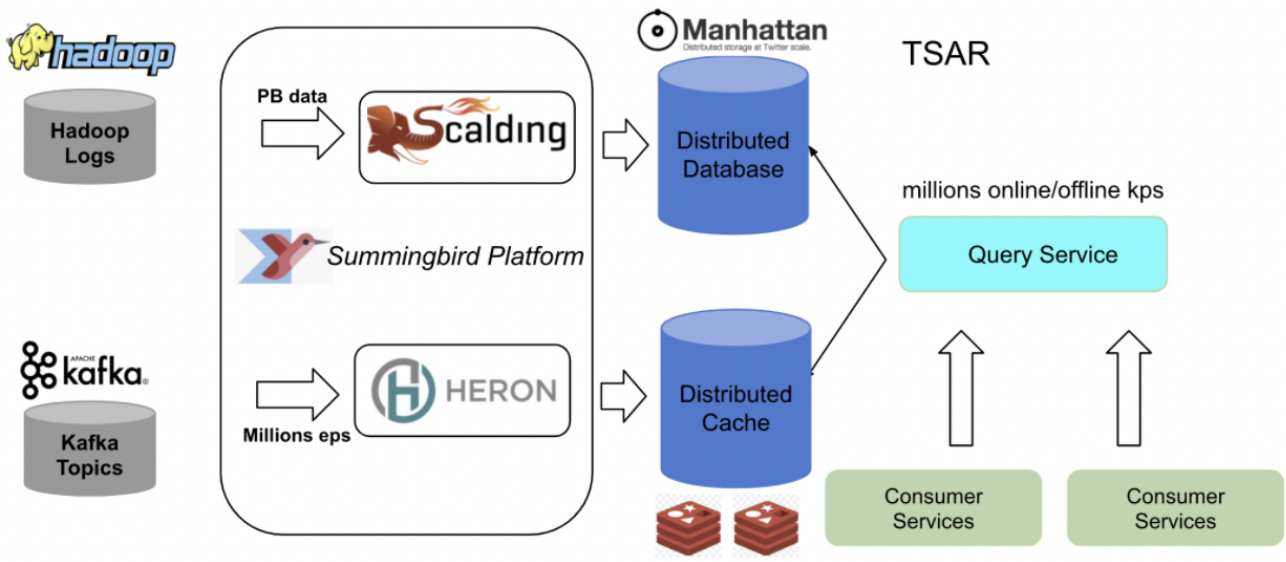


Disney **solves the following problems** with its Kafka–based Kappa architecture:

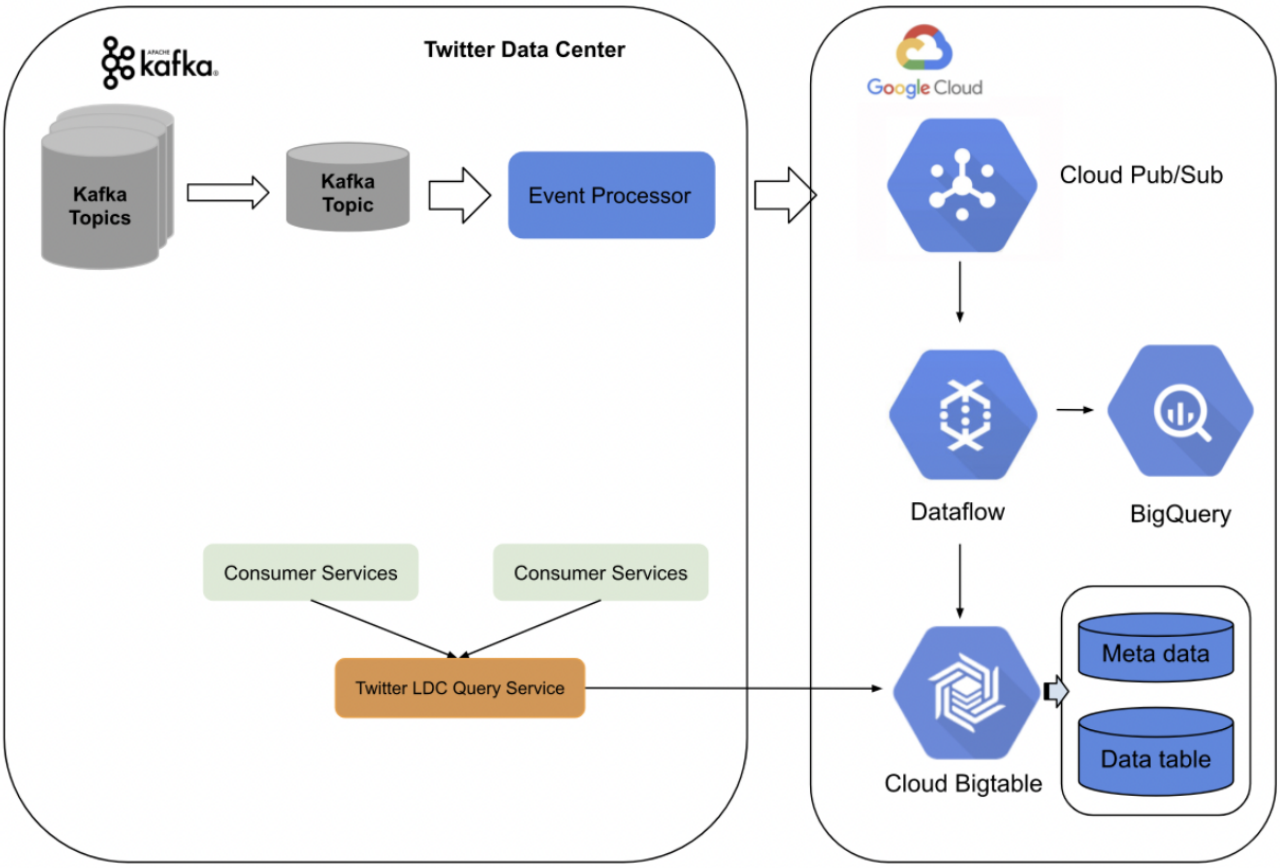
- Keep it simple (Kiss)
- Reduce Code Duplication
- Decreasing End To End Latency
- Full System Immutability
- Avoiding Data Discrepancies
- Ability to move laterally between storage systems
- Everyone wants their answers faster

Kappa at Twitter for Migration from Lambda Architecture

Twitter processes approximately **400 billion events in real–time and generates petabyte (PB) scale data every day**. The on–premise architecture with Hadoop and Kafka using the Lambda architecture was not efficient enough:



Therefore, Twitter **migrated to the cloud on GCP with Kafka using the Kappa architecture**:

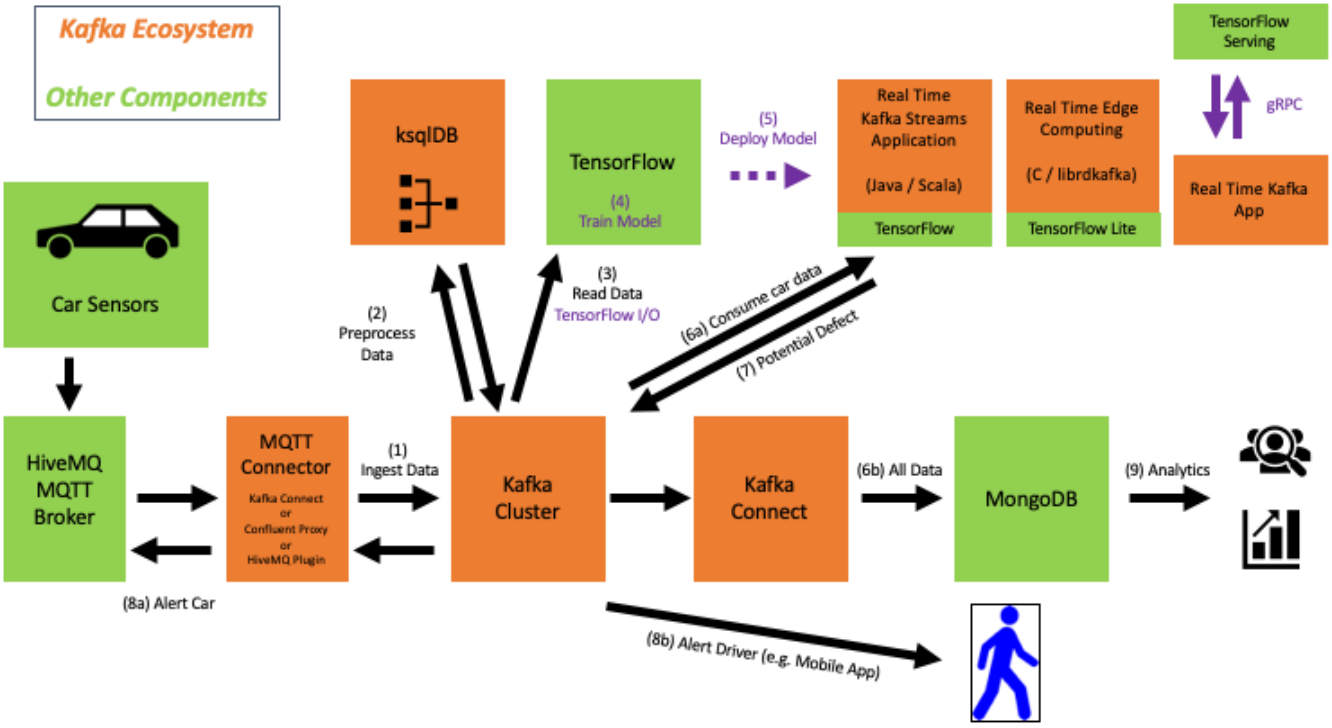


With the new **hybrid architecture** on both Twitter Data Center and Google Cloud Platform, they "are able to **process billions of events in real–time and achieve low latency, high accuracy, stability, architecture simplicity, and reduced operation cost** for engineers" as Twitter quotes in their detailed blog post about their Lambda to Kappa migration.

Example Project: Kappa for Machine Learning including Model Training, Scoring, and Monitoring

After real–world examples from Uber, Shopify, and Disney, I want to share one more **practical code example**: A technical demo connecting to 100,000 IoT devices to do streaming machine learning.

The use case is about integrating **tens or hundreds of thousands of IoT devices and processing the data in real–time**. The demo use case is predictive maintenance (i.e., anomaly detection) in a connected car infrastructure to predict motor engine failures:



The implemented Kappa architecture provides a single real–time infrastructure for various very different use cases and processing paradigms:

- **Real–time data ingestion at high throughput from IoT devices via an MQTT proxy**: Integration with millions of interfaces, in this case, simulated vehicles.
- **Batch processing for model training**: The TensorFlow Python application from the data scientist consumes historical data from the Kafka log to train analytic models.
- **Real–time stream processing for model scoring**: The Java–based streaming application is powered by Kafka Streams / ksqlDB and operated by the production engineer with mission–critical SLAs and low latency.
- **Near–real time ingestion into the digital twin for analytics**: Kafka Connect ingests the data into different databases and applications, in this case, a MongoDB Atlas cloud service.
- **Synchronous request–response / RPC communication for mobile app integration and transactional workloads**: The Confluent REST Proxy (or any other web / mobile proxy) sends real–time alerts to humans.

The whole infrastructure is cloud–native. It runs on Kubernetes and can be deployed in a data center or on any hyperscaler. The following blog post explains the demo in detail: IoT Live Demo — 100.000 Connected Cars with Kubernetes, Kafka, MQTT, TensorFlow. The code is available in the Github project.

Kappa under the Hood of Next Generation Software Products and SaaS Offerings

Software companies have the same challenges as end-users like Uber, Shopify, or Disney. Hence, no surprise that software vendors move to Kappa architectures and real-time capabilities as the heart of their infrastructures.

This section shows a few **examples of software vendors that moved to event-based architectures, event streaming, and asynchronous external interfaces** within their next-generation software offerings.

Once again: This does NOT mean that everything within these products is real-time or event-based, but only if the related components provide real-time capabilities, then you can provide a real-time interface for internal or external consumers.

Business Solutions (Salesforce, SAP, Slack, et al.)

Business solutions provide **customer interactions, logistics, manufacturing, internal communication, and many other use cases**. No surprise that real-time data beats slow data. For this reason, most modern business solutions moved from less flexible and less scalable communication paradigms to event-based interfaces. Instead of using files, web service APIs, or manual changes, communication happens via **event-driven APIs internally and externally**.

A few examples across different business solutions:

- **Salesforce**: The internal "platform events" architecture heavily relies on Apache Kafka for decoupled real-time data processing at scale. External APIs like the integration with Salesforce's proprietary sObject datatype moved from SOAP and REST web service to Streaming API PushTopics, Enterprise Messaging Platform Events, and Change Data Capture Events.
- **SAP**: Instead of relying on its legacy proprietary interfaces such as BAPI and iDoc, SAP moved to event-based APIs in their next-generation SAP S/4 Hana ERP platform. The blog "SAP integration options for Apache Kafka" shows the mess of numerous legacy interfaces and alternative modern event-based integration options.
- **Slack**: Being a messaging platform by nature, it is no surprise that the heart of their core backend infrastructure leverages event streaming. Slack's data streaming team focuses on providing Kafka as a Service for the company at the scale of trillions of messages per day across

dozens of clusters in Amazon data centers. For the front–end, Slack's current architecture leverages a service mesh built with Envoy and WebSockets.

Databases, Data Warehouses, Log Analytics

Data storage and analytics vendors are **traditionally batch technologies for long–term storage, dashboards, reporting, and interactive queries**. The heart of most solutions is still a batch system for analytics workloads. That's the core business of these products and services.

Nevertheless, almost all of these vendors went into **(near) real–time business due to customer demand**. Hence, event–based integration capabilities and near real–time ingestion, processing, and analytics are becoming more prevalent. Some examples:

- **MongoDB**: "Change Streams" allow applications to access real–time data changes from the document–based NoSQL datastore.
- **Snowflake**: "Snowpipe" can help organizations seamlessly load continuously generated data into the cloud data warehouse.
- **Elasticsearch**: "Data Streams" lets you store append–only time series data across multiple indices while giving you a single named resource for requests. Data streams are well–suited for logs, events, metrics, and other continuously generated data to ingest data into the Elastic search engine.

These solutions have in common that they move from batch to near real–time ingestion into their data store or data lake. Nevertheless, they still store and analyze data at rest. Hence, this is **complementary but not an alternative to event streaming**.

New entrants into the market try to differentiate from the above data storage vendors by providing a real–time infrastructure at its core. A great example is **Rockset**, a scalable real–time analytics platform in the cloud. As it is a native real–time solution, Rockset natively integrates with event streaming platforms such as Apache Kafka.

Event Streaming

Event Streaming platforms are event–based by nature. They process data in motion continuously. Therefore, **the central nervous system of a Kappa architecture has to be an event streaming platform**. Period.

For a **comparison of frameworks like Kafka and Pulsar, plus reviewing the differentiators from platform vendors and SaaS providers** such as Confluent, Cloudera, Red Hat, Amazon MSK, Azure Event Hubs, etc., please check out this comparison of event streaming platforms.

Event streaming will be one serverless component in a cloud–native data lake architecture in many future enterprise architectures.

It is worth noting that **event streaming and the above–discussed business solutions and data storage and analytics vendors are complementary, not competitive!** For instance, Confluent partners with business solutions such as Salesforce, database vendors such as MongoDB and Elastic, data–warehouses such as Snowflake, and cloud providers such as AWS or Azure to provide Source, Sink, and Change Data Capture (CDC) connectors powered by Kafka Connect. The fully managed Confluent Cloud service even provides the end–to–end integration as part of the serverless offering in the public cloud.

Video Recording: Kappa vs. Lambda Architecture

I covered the discussion around "Kappa vs. Lambda" in a **40–minute video recording**, too. Enjoy:

Kappa is the New Black for the Enterprise Architecture

Real–time data beats slow data. After reading this article, think about your industry, business unit, and projects again. If real–time data processing improves your customer experiences, increases your revenue, or reduces your cost and risk, then why wait? **The Kappa architecture provides enormous benefits and a much simpler infrastructure than the Lambda architecture**.

Having said this, batch processing and other data storage and analytics services are not going away. **Kappa and event streaming are complementary, and no silver bullet for every problem**. For more details, check out the article "Can Apache Kafka replace a database?" — that article emphases this statement and explores the trade–offs.

Event streaming is the foundation of Kappa architecture. There is no way around this. Apache Kafka is the de facto standard for event streaming and the choice in real–world Kappa architectures. If you still need or want to evaluate your own event streaming platform, continue with the Kafka vs. Pulsar comparison or the general comparison of competitive event streaming vendors and cloud solutions.

Did you already Kappa architecture? Or do you still rely on or even prefer Lambda architectures? What are your experiences and opinions? Let's connect on LinkedIn and discuss it! Stay informed about new blog posts by subscribing to my newsletter.

SIGN UP

We don't spam! Read our privacy policy for more info.
If you have issues with the registration, please try a private browser tab / incognito mode. If it doesn't help, write me: kontakt@kai-waehner.de

RELATED TAGS

ARCHITECTURE BATCH CLOUD DATA LAKE DATABRICKS DELTA LAKE HADOOP KAFKA KAPPA LAKE HOUSE LAMBDA OBJECT STORAGE OPEN SOURCE PULSAR REAL TIME S3 SPARK STREAM PROCESSING STREAMING ANALYTICS

Kai Waehner

builds cloud–native event streaming infrastructures for real–time data processing and analytics

4 COMMENTS

SURAJ BISHT says:
24. September 2021 at 1:36

Thanks Kai for a good article
I differ with above opnion, most of the Kappa architecture and implementation highlighted in second half (Uber, IoT etc) are actually closer to Lambda architecture where data is forked at Kafka and individual recieving stream may have their own database which is against kafka architectue to have single data truth. Only major difference in latest trend is that Kafka layer is acting as a data distributer and otignal data sources need to fork data separately to streaming and batch system.
Thanks,
Suraj

Log in to Reply

Kai Waehner says:
24. September 2021 at 8:50

I think your comment is not very different from my explanations and this article. Yes, Kafka is used to forking data. The difference to Lambda is that the heart of the infrastructure is real–time and scalable (but also storage for decoupling and supporting batch consumers). I recommend reading the article from Martin Kleppmann about turning the database inside out. That explains this approach very well.

Log in to Reply

John says:
30. March 2023 at 20:26

Thanks Kai, that was a terrific read. Can you comment on the suitability of Kappa for smaller organisations (where they might only be generating say <1 gigabyte of data per day)? Or would it be overkill and too costly.

Log in to Reply

Kai Waehner says:
31. March 2023 at 8:11

Kappa architecture is not related to the volume of data. Most of our customers use Kafka (and Kappa) for analytical AND transactional workloads. The following blog post might be helpful: https://www.kai–waehner.de/blog/2022/03/09/analytics–vs–transactions–api–data–streaming–with–apache–kafka/

Log in to Reply

LEAVE A REPLY

You must be logged in to post a comment.

YOU MAY ALSO LIKE

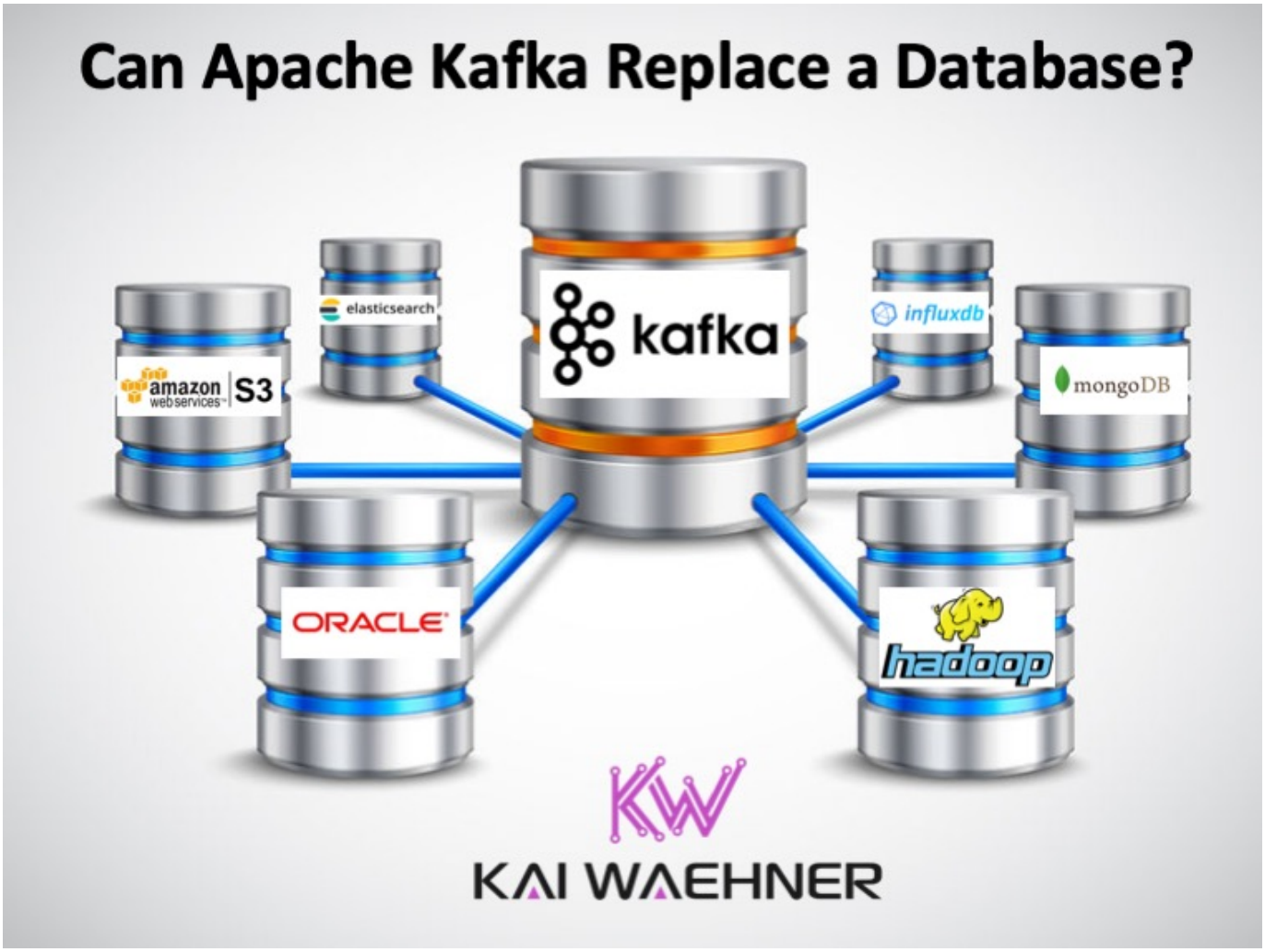
Error Handling via Dead Letter Queue in Apache Kafka

APACHE KAFKA, DATA STREAMING, DEAD LETTER QUEUE, DESIGN PATTERN, MESSAGE QUEUE

By KAI WAEHNER · 30. May 2022 · 3 comments

Recognizing and handling errors is essential for any reliable data streaming pipeline. This blog post explores best practices for implementing error handling using a Dead Letter Queue in Apache Kafka infrastructure. The options include a custom implementation, Kafka Streams, Kafka Connect, the Spring framework, and the Parallel Consumer. Real–world case studies show how Uber, CrowdStrike, Santander Bank, and Robinhood build reliable real–time error handling at an extreme scale.

READ MORE >



Can Apache Kafka Replace a Database?

ANALYTICS, APACHE KAFKA, ARCHITECTURE, BIG DATA, CONFLUENT, DATABASE, INTEGRATION, KAFKA CONNECT

By KAI WAEHNER · 12. March 2020 · 15 shares · 2 comments

Can and should Apache Kafka replace a database? How long can and should I store data in Kafka?…

READ MORE >

15

TECHNOLOGY EVANGELIST

Kai Waehner

builds cloud–native event streaming infrastructures for real–time data processing and analytics

SUBSCRIBE TO MY NEWSLETTER

Stay informed about new blog posts!

Email Address *

First name

Last name

SIGN UP

We don't spam! Read our privacy policy for more info.
If you have issues with the registration, please try a private browser tab / incognito mode. If it doesn't help, write me: kontakt@kai-waehner.de

END-TO-END INTEGRATION

FEATURED POSTS

Apache Kafka, KSQL and Apache PLC4X for IIoT Data Integration and Processing

Apache Kafka vs. Middleware (MQ, ETL, ESB) – Slides + Video

Deep Learning Example: Apache Kafka + Python + Keras + TensorFlow + Deeplearning4j

CATEGORIES

Select Category

TAG – CLOUD

ANALYTICS APACHE APACHE CAMEL APACHE KAFKA AWS

BIG DATA BUSINESSWORKS CLOUD CLOUD-NATIVE CONFLUENT

DATA STREAMING DEEP LEARNING DOCKER EAI EDGE

ENTERPRISE APPLICATION INTEGRATION ENTERPRISE SERVICE BUS ESB

EVENT STREAMING FLINK HADOOP HYBRID IBM IIOT

INTEGRATION IOT J2EE JAVA JEE KAFKA KAFKA CONNECT

KAFKA STREAMS KSQL KUBERNETES MACHINE LEARNING

MICROSERVICES MIDDLEWARE OPEN SOURCE REAL TIME SOA

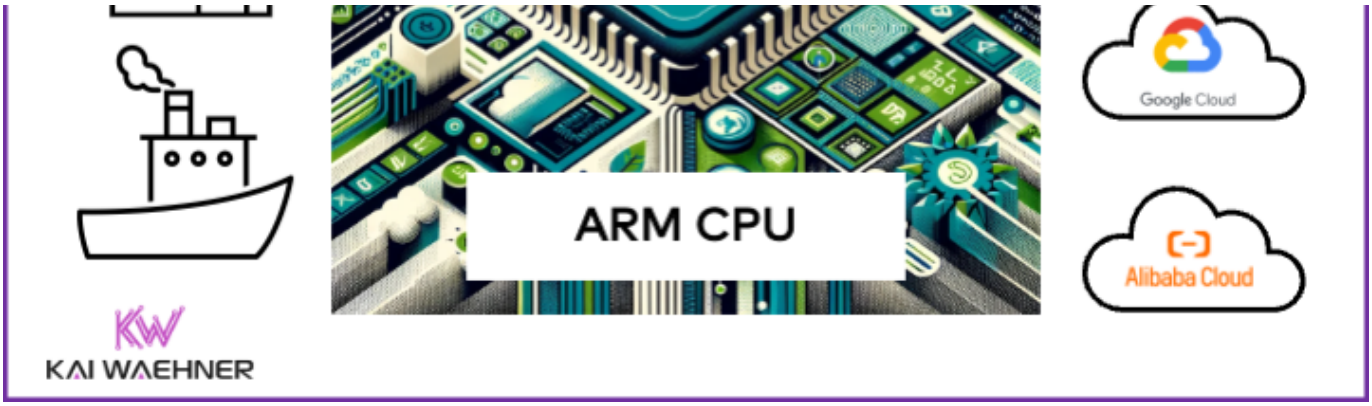
STREAMBASE STREAMING ANALYTICS STREAM PROCESSING TALEND

TIBCO

APACHE KAFKA, JAVASCRIPT, NODEJS, OPEN SOURCE

JavaScript, Node.js and Apache Kafka for Full-Stack Data Streaming

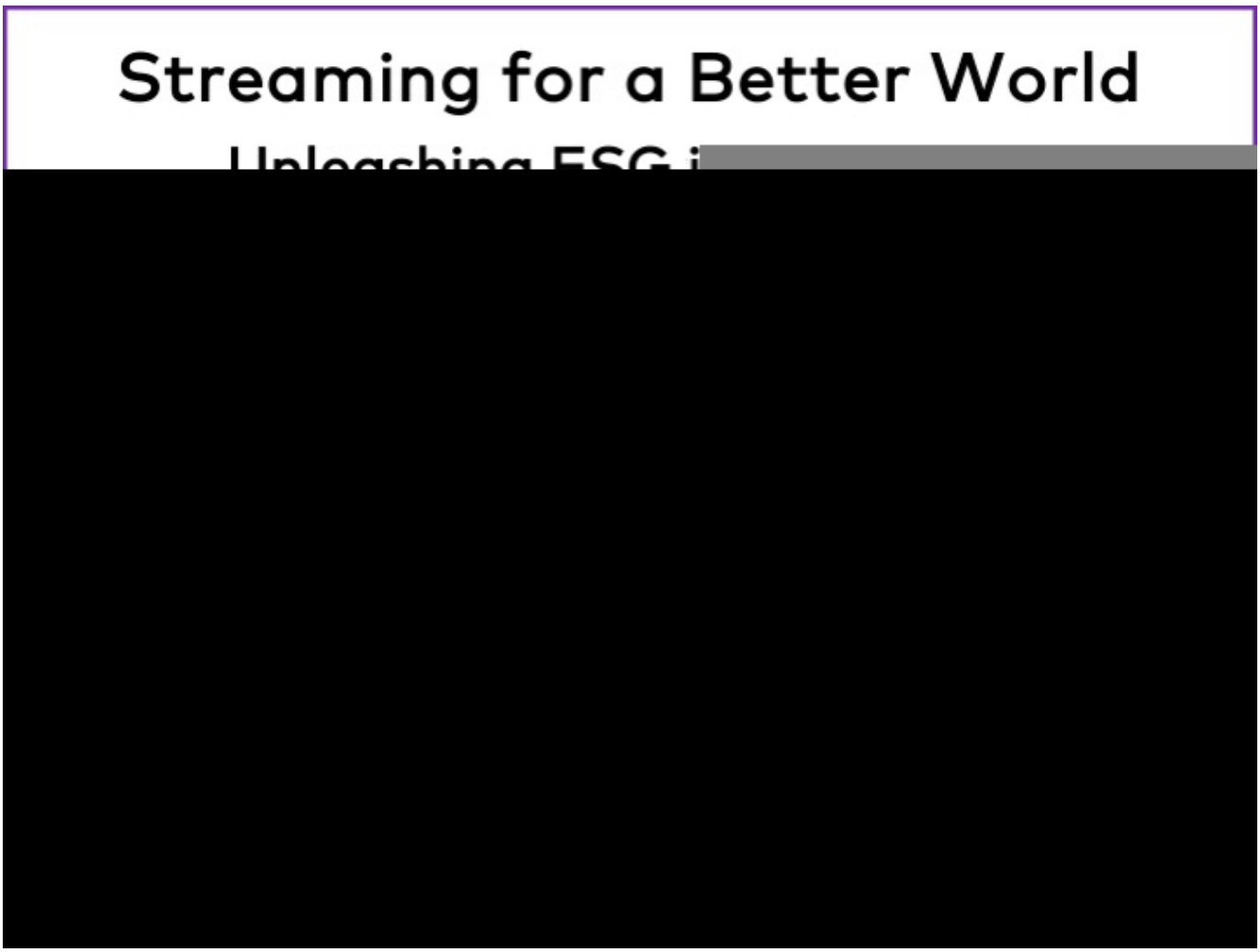
KAI WAEHNER · 4. March 2024 · No comments



APACHE KAFKA, ARM, CLOUD, EDGE, HYBRID CLOUD

ARM CPU for Cost-Effective Apache Kafka at the Edge and Cloud

KAI WAEHNER · 22. February 2024 · No comments



APACHE FLINK, APACHE KAFKA, DATA STREAMING, ESG

Green Data, Clean Insights: How Kafka and Flink Power ESG Transformations

KAI WAEHNER · 10. February 2024 · No comments

APACHE FLINK, APACHE KAFKA, ARTIFICIAL INTELLIGENCE, GENAI, LANGCHAIN, MACHINE LEARNING, PYTHON

GenAI Demo with Kafka, Flink, LangChain and OpenAI

KAI WAEHNER · 29. January 2024 · No comments