Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2023, 10(6):284-289**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Streamlining Data Ingestion with Apache Kafka and Databricks

Ravi Shankar Koppula

Satsyil Corp, Herndon, VA, USA
Ravikoppula100@gmail.com

Abstract: In this research paper, we delve into the intricacies of streamlining data ingestion using Apache Kafka and Databricks. As organizations continue to generate massive amounts of data, the need for real-time data ingestion and processing has become paramount. Traditional batch processing methods are no longer sufficient to meet the demands for timely insights and decision-making. Apache Kafka, a distributed streaming platform, coupled with Databricks, a unified analytics platform, offers a robust solution for real-time data ingestion and analytics. This paper explores the architecture, key features, and integration of Apache Kafka with Databricks. We discuss best practices for optimizing data ingestion, including data serialization, compression, monitoring, and alerting. Through this examination, we aim to provide a comprehensive understanding of how to effectively manage and utilize continuous data streams, ensuring scalability, efficiency, and accuracy in data-driven environments.

Keywords: Apache Kafka, Databricks, Data Ingestion, Real-time Analytics, Streaming Data, Data Serialization, Data Compression, Unified Analytics Platform, Monitoring and Alerting, Big Data.

Introduction

Most organizations, big or small, generate data continuously. Organizations invest significantly in analytics to derive insights from this data and to make data-driven decisions. Most of this data is ingested as batch data and stored on the data lake using different ingestion mechanisms. Once it is on the data lake, multiple datasets are processed asynchronously or as daily aggregation, joined, or filtered to create a golden dataset which is referred to as the "single source of truth". This is rather an outdated architecture considering that most organizations now report on data in real-time through KPIs, scorecards, and other visualizations. This has created the need for real-time streaming data ingestion solutions with minimal latency [1]. Organizations need an architecture wherein data is ingested as a stream and all-consuming analytics are continuously rerun whenever there are updates instead of periodically rerunning them through batch jobs.

A streaming architecture operates on the idea that at any time there will be one or more datasets referred to as continuously changing datasets, and at any time they can either be fixed snapshots of the dataset at a particular time or it is unknown how it can be fixed. Some classic examples of continuously changing datasets are funds available in an ATM, current temperature of a city, stock prices, flight information, etc. Streams of such continuously changing datasets can be generated by live sensors, data feeds, logs, and other changing data sources. Stream processing engines process these datasets over time and generate derived datasets. Like batch processing engines, this architecture is also comprised of an architecture where messages are ingested by a producer and transported to subscribers via a message broker. Streaming data ingestion is implemented on apache kafka along with stream analytics on databricks [2]. The architecture is visualized as follows:

Overview of Data Ingestion

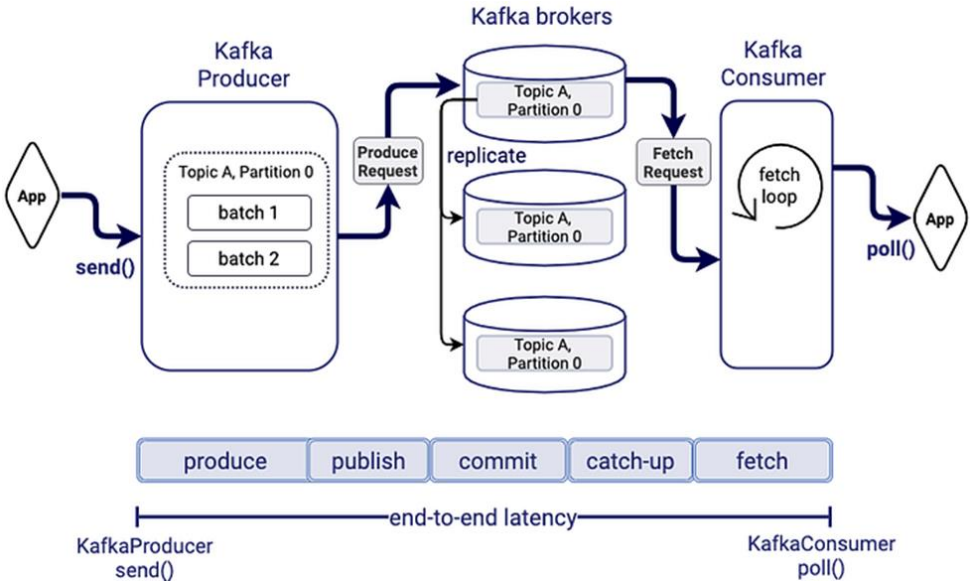
The vast majority of data is generated by cells in linked databases, which results in a large amount of data commonly referred to as the "data deluge" [1]. Cell records include a wide variety of different "types" of information that have been produced in unusual lengths, as well as different levels of quality. It becomes nearly

impossible to keep clean, one entry unit of data that can be used as a basis for assessment in the data deluge period, as more data sources emerge and progress. For this examination and research, the linked datasets generated by Open Corporates will be scrutinized. Open Corporates publishes information about companies registered at the Companies House, which is the Registrar of Companies for England and Wales, Scotland, and Northern Ireland. These companies have a unique identifier called a company number.

A small number of database entries in the linked dataset contain company number attributes that are malformed, resulting in incorrect data entry for that specific entry. Using a company registry as an example, an intelligent data pipeline can be extrapolated that ingests data into a data warehouse, sanity-checks them, and finally enriches the imperfect data with information from an external web service. The data enrichment strategy is developed based on a web service provided by the "Companies House" organization, which provides information about companies registered in the UK.

Apache Kafka: Key Concepts

In Kafka, consumers are applications that subscribe to one or more topics. A consumer group can consist of multiple applications, and each message sent to a topic will be consumed only once by these applications. A topic and a partition are uniquely identified by their name. Every message appended to a partition gets an incremental, unique offset, which is a monotonically increasing identifier that can be used to retrieve the message [2].



Topics and Partitions

Topics in Kafka are the high-level flow information entities (a stream of messages - events) where producers categorize the events they generate. Each Topic is identified by a name and, if not configured otherwise, events are written to it in a round-robin manner. Unlike other messaging systems, by default topics' events are not removed when they are consumed. Kafka does not store events in RAM but in disk, accessible through an index, with a configurable retention policy (hours, days, or size). This allows a reprocessing of events generated in a past time. Also, this architecture provides a distributed log, where consumers can go through the log at their own pace when required. The absence of forwarding (as with message queues) allows consumers to consume the same event multiple times; hence it is entirely possible to forward the same event to multiple unmatched front-end systems. All these features address a data-streaming architecture need, allowing the distributed processing of events made by different consumer systems [2].

Partitions provide load balancing and fault tolerance. Each Topic can be divided into several partitions and each partition can have several replicas. By having more partitions, the load of the log can be distributed among several Brokers. Each partition is totally ordered (within the partition, each event has an immutable sequence ID - offset) and replicated (a partition replication is a copy of the partition that is kept in an entirely different Broker). The consumer side is simpler: there is no message acknowledgment to the Producer. Once a message has been appended to the log, the system guarantees that it will be delivered to the consumers. Acknowledged

messages are not removed from the log. Brokers act as a peer-to-peer system, meaning that all start, stop, data forwarding, and log file storage operations are equally handled.

Producers and Consumers

In an Apache Kafka deployment, data is produced and consumed by external applications called producers and consumers. A producer publishes data to a Kafka topic; a consumer subscribes to the topic and reads from it. There can be many producers and consumers for a topic; they do not need to be aware of each other's existence. Producers and consumers can be an external application or another Kafka topic, allowing for complex topologies to be constructed. Producers and consumers can be implemented in many programming languages using the producer and consumer APIs provided with the Kafka distribution. When clients are written in languages other than Java, the client can take advantage of the performance and scalability of Kafka [3]. Data from Kafka topics is read, written, and transformed via connectors. Kafka Connect comes bundled with source connectors to pull data into Kafka from other systems, such as relational databases, and sink connectors that remove data from Kafka and push it to other systems. Kafka Connect makes deploying connectors very easy. Developers provide a file containing configuration properties. This file is then uploaded to a Connect worker (a stand-alone process that runs the connectors); the Connect worker will then start the connector in its own thread. Endpoints provided by the Connect worker can be used to monitor and control running connectors. Kafka Connect REST interface has been designed to provide good ways to integrate with other systems. For example, a connector can retrieve and write back data to the same RDBMS from which it ingests or writes to a NoSQL database to store data read from an RDBMS or other systems [4].

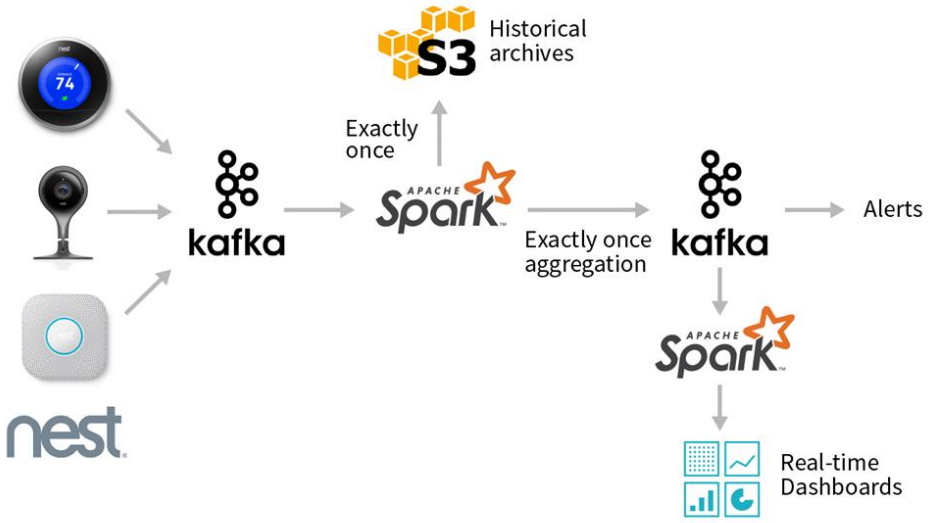
Databricks: Key Features

A Unified Approach to Cloud Data Management is suggested with Databricks, the Data and AI Platform. All data, analytics and AI are managed on an open and unified platform, making it possible to integrate data from all sources, build and train AI algorithms at scale and share insights in real time. Databricks is provided on all major clouds and integrates with hundreds of third-party technologies, making it easy to put the platform at the heart of an organization’s data strategy. All data is integrated, analytics simplified and AI made accessible for everyone. The unified and open platform provides a better chance of success than fragmented alternatives where integration and open standards have low priority [1].

Databricks is a self-service platform for a wide range of use cases; including batch and streaming data ingestion data processing jobs, ETL workloads, running complex machine learning experiments, serving prediction scores in real-time and providing dashboards for BI tools. It enables organizations to use a consistent paradigm and simplify operations. Databricks allows viewing workloads and their dependencies in one place and managing performance with pool queues. The ability to isolate workloads on different clusters per job and instantly spin up new clusters on demand improves performance and resource allocation.

Unified Analytics Platform

The Databricks Unified Analytics Platform is a comprehensive and sophisticated relationship analytics environment-integrating administrative, development, and user interfaces with a resilient computational fabric. It is designed to leverage existing big data sources or ingest new sources of big data for use within Spark, while also supporting the publishing of new big data sources resulting from the analytics efforts of Spark [1]. All the workload and data interfaces are unified, providing a common platform for spark on cloud or bare metal, and all other big data solutions. The Unified Analytics Platform’s simple interface describes what tables get processed and the operations to perform on them, while the execution model automatically deploys and configures all replicas of the tables-complex streaming queries or transformations.



Ingestion is a critical aspect of most data management systems, and its importance is magnified for big data systems. A comprehensive approach to universal ingestion for big data management systems is a uni-component big data management system. It should enable a system to easily grow its big data size, data source complexity, number of users, data processing computational load, and knowledge complex and scale. Most importantly, it should do so without great complexity. A uni-component big data management system automatically, seamlessly, and efficiently handles all these expanding aspects. With this approach, the ingestion capabilities of existing and newly developing systems are analyzed against the different metrics comprising universal ingestion.

Integration with Apache Kafka

The robust integration capabilities of Databricks with Apache Kafka present a powerful opportunity for businesses to streamline the process of data ingestion from a variety of sources, including clickstream events, IoT sensors, and enterprise applications [1]. This integration smooths the handoff of data from source systems to Databricks, with an option for built-in schema evolution, ensuring that the data conforms to an expected structure for analytics. The outcome is accelerated time to insight. Apache Kafka is a widely used message broker system devised to stream events from production systems, acting as a central hub where messages are held until they are consumed [2]. This decouples message producers from consumers, allowing them to be developed independently. Besides being a robust platform for building data pipelines, Kafka is also commonly used as a platform for building enterprise event notification frameworks, exposing an API to publish/subscribe topic-based events around which services can be built.

Best Practices for Streamlining Data Ingestion

Streamlining data ingestion process can help reduce latency and increase efficiency in a data-driven organization. By implementing various optimization techniques, it is crucial to continuously improve the data ingestion process for enhanced performance and scalability. Fine-tuning the process will not only maximize the utilization of available resources and technologies but also enable seamless integration of data from multiple sources. This comprehensive approach ensures a smoother, more robust data flow, maximizing the potential for valuable insights and informed decision-making.

Consider using Avro/ProtoBuf/Thrift as serialization formats when ingesting data into Kafka to keep the size of data records reasonable. json is also highly recommended if you are planning to keep the data in the JSON format. An interesting observation is that the data records' size does not exhibit a linear growth pattern with the number of fields, especially in the case of json. Surprisingly, even with less than 100 fields, the data record size becomes a regretful 48~50 KiB, not expanding much further even when adding more fields. When it comes to data compression, the gz format is suggested if you are utilizing Kafka-Connect-HDFS as the sink connector. On the other hand, if you are employing Kafka-Connect-Spark to avoid test failures on Databricks, snappy might prove to be a wise choice. Its efficient compression capabilities are well-suited for such scenarios. [5]

Data Serialization and Compression

Effective data serialization is crucial in the context of data ingestion pipelines with datastores because it ensures smooth processing and prevents parsing exceptions when binary data is streamed over the network. By properly serializing the data, it can be efficiently converted into a format that can be easily ingested by the receiving side, even if it expects text instead of binary. This is particularly important in scenarios where real-time data processing is required, as fast and accurate conversion of data is essential for timely analysis and decision-making. Another important aspect of data processing is data compression, which can greatly reduce the bandwidth usage on the network. By compressing the data before transmission, the size of the data can be minimized, leading to faster transfer times and reduced resource usage. This is especially beneficial when dealing with large datasets or when network bandwidth is limited. In addition, efficient data compression enables more efficient storage utilization, as compressed data requires less space on disk or in memory. In the case of Kafka consumer throughput, the choice of message key and value can play a significant role. By using relevant message keys such as user ID, session ID, or device serial number, the consumer can more efficiently process and filter the incoming messages based on these keys. This can greatly improve the overall throughput and performance of the Kafka consumer, enabling faster and more efficient data ingestion. Additionally, careful consideration should be given to the serialization and compression techniques used in conjunction with the chosen message keys, to ensure optimal performance in terms of both speed and resource usage. In conclusion, effective data serialization and data compression are essential components of data ingestion pipelines. They not only ensure smooth data transfer and prevent parsing exceptions but also optimize network bandwidth usage and storage utilization. Additionally, the choice of message keys in Kafka consumer throughput can greatly impact the overall performance and efficiency of the system. Therefore, it is crucial to consider these aspects when designing and implementing data ingestion processes. By taking a holistic approach and considering all these factors, organizations can achieve faster and more efficient data ingestion, leading to improved data processing capabilities and better business insights. [2]

Apache Avro and Protocol Buffers are recommended data serialization formats because they work natively with Kafka and both provide a Java API that supports (de)serialization for binary data. Avro works well with complex nested data structures on top of already defined primitives. Avro schemas are specified using JSON notation and compatible with DDL notation in Hive [5].

Monitoring and Alerting

When implementing monitoring and alerting mechanisms, the following best practices should be considered: monitoring key metrics such as the number of ingested records, processing time, and failure count, establishing thresholds for alerts to prevent alert fatigue, periodically reviewing and updating alert configurations, aggregating alerts to reduce noise, correlating alerts to identify root causes, and implementing runbooks that provide prescribed solution paths for issues.[6][1]

Conclusion

In the evolving landscape of big data, efficient data ingestion and real-time processing are critical for deriving timely insights and making informed decisions. This paper has explored the synergistic integration of Apache Kafka and Databricks, two powerful platforms that together facilitate streamlined data ingestion and advanced analytics. Apache Kafka, with its robust distributed streaming capabilities, and Databricks, with its unified analytics environment, provide a comprehensive solution for managing continuous data streams. We have detailed the key concepts of Apache Kafka, emphasizing its role in real-time data streaming and its architectural components, such as producers, consumers, topics, and partitions. Similarly, the discussion on Databricks highlighted its features, including its ability to seamlessly integrate with various data sources and analytics tools, thereby enhancing the overall data processing workflow.

Best practices for optimizing data ingestion were thoroughly examined, focusing on data serialization, compression, and the importance of implementing effective monitoring and alerting mechanisms. These practices ensure that the data ingestion process is efficient, scalable, and resilient, capable of handling the dynamic nature of big data environments. The combination of Apache Kafka and Databricks offers a robust framework that supports the ingestion, processing, and analysis of real-time data. This integrated approach not only enhances the performance and scalability of data-driven applications but also provides a flexible and

efficient means of managing continuous data streams. As organizations continue to generate and rely on vast amounts of data, the adoption of such advanced data ingestion and analytics solutions will be crucial for maintaining a competitive edge and achieving operational excellence.

By leveraging the capabilities of Apache Kafka and Databricks, businesses can effectively manage their data pipelines, ensuring that data is ingested and processed in real-time, thereby facilitating faster decision-making and enabling more agile and responsive business operations. The insights gained from this study underscore the importance of adopting modern data architectures and practices to meet the growing demands of the digital age.

References

[1]. X. Wang and M. J. Carey, "An IDEA: An Ingestion Framework for Data Enrichment in AsterixDB," 2019.

[2]. C. Martín, P. Langendoerfer, P. Soltani Zarrin, M. Díaz et al., "Kafka-ML: Connecting the data stream with ML/AI frameworks," 2022.

[3]. B. Lawlor, R. Lynch, M. Mac Aogáin, and P. Walsh, "Field of genes: using Apache Kafka as a bioinformatic data repository," 2018. ncbi.nlm.nih.gov

[4]. S. Singh Sandha, M. Kachuee, and S. Darabi, "Complex Event Processing of Health Data in Real-time to Predict Heart Failure Risk and Stress," 2017.

[5]. O. C. Marcu, A. Costan, G. Antoniu, M. S. Pérez-Hernández et al., "Towards a Unified Storage and Ingestion Architecture for Stream Processing," 2017.

[6]. A. Akanbi and M. Masinde, "A Distributed Stream Processing Middleware Framework for Real-Time Analysis of Heterogeneous Data on Big Data Platform: Case of Environmental Monitoring," 2020. ncbi.nlm.nih.gov