International Journal of Advanced Science and Technology
Vol.109 (2017), pp.35-42
http://dx.doi.org/10.14257/ijast.2017.109.04

Apache Kafka: Real Time Implementation with Kafka Architecture Review

Javed Ahmed Shaheen

M. Phil Computer Science, Computer Science Department, Virtual University Faisalabad, Pakistan
javedmatyana@yahoo.com

Abstract

Unluckily today's general architectures for the real time data and its processing at extent suffer from too much complexity: let we say, lot of technologies that need to be darned and operated together, and each individual technology is often complex by itself. Let we have to desire to publish and subscribe, streams of records then Apache Kafka is similar to a message queue or we can say it is an enterprise messaging system. Let we have to store streams of records in a fault-tolerant way: Kafka process streams of records as they occur. Kafka is better for applications of two broad classes one is structuring a real time streaming of data pipelines, means consistently obtain data b/w systems or applications and constructing real-time streaming applications that make over the streams of data. Kafka can do these things as it run as a cluster on one or more servers. The Kafka cluster accumulates streams of records in categories called topics while every record has a key, a value, and a timestamp. In this paper we have discussed Apache Kafka architecture and in last we have illustrate some Kafka applications in Big data era to solve problems through streaming.

Keywords: *APACHE KAFKA, KAFKA APIs, KAFKA Architecture, KAFKA REAL Time Application*

1. Introduction

Apache Kafka is developed by the Apache Software Foundation written in Scala and Java, it is an open source, stream processing platform with aim to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. If we discuss about its layers, its storage layer is basically a scalable publish or it subscribe message queue architected as a distributed transaction log, [1] which is making it highly valuable for enterprise infrastructures to process streaming data. Kafka connects with external systems to import or export data through Kafka Connect and it also makes available Kafka Streams, called Java stream processing library. The design is closely influenced by transaction logs [2]. Figure 1 is showing the Kafka ecosystem mean how producers push message to Kafka cluster and how consumers pull that message from broker of Kafka and the figure is also depicting on zookeeper which is used for managing and coordinating Kafka broker: ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.

Received (December 10, 2016), Review Result (March 7, 2017), Accepted (December 2, 2017)

ISSN: 2005-4238 IJAST

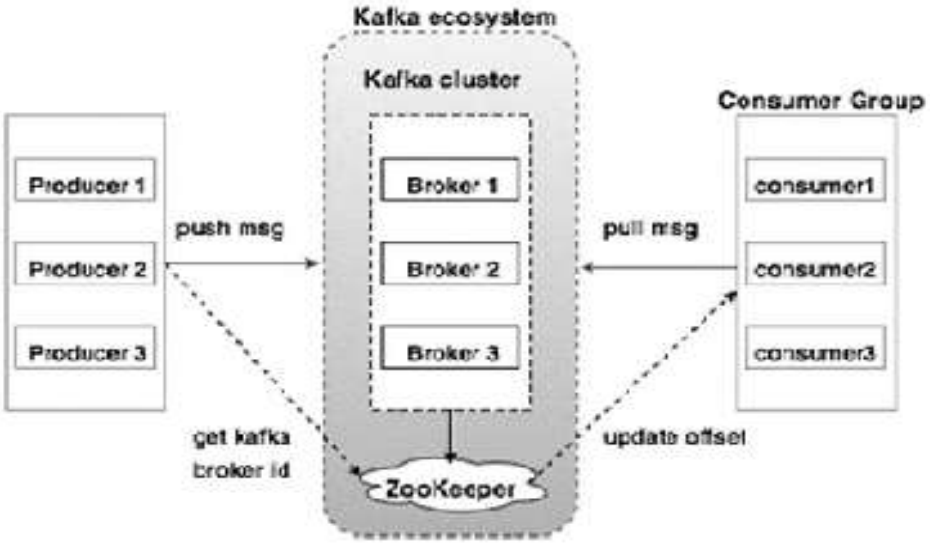


Figure 1. Kafka EchoSystem

2. APACHE Kafka Architecture:

Kafka is actually a store which stores messages come from processes (one or many) called producers. The data or messages are then partitioned in different partitions within various Topics. In this Topic's partition the messages are indexed and stored together with a timestamp. On the other end, other processes called Consumers can inquire messages from these partitions. Kafka which is working b/w these producers and consumers runs on a cluster of one or more servers and the partitions can be distributed across cluster nodes. Apache Kafka efficiently processes the real-time, streaming data when implemented along with Apache Storm, Apache HBase and Apache Spark. Figure 2 is showing basic architecture of Kafka.

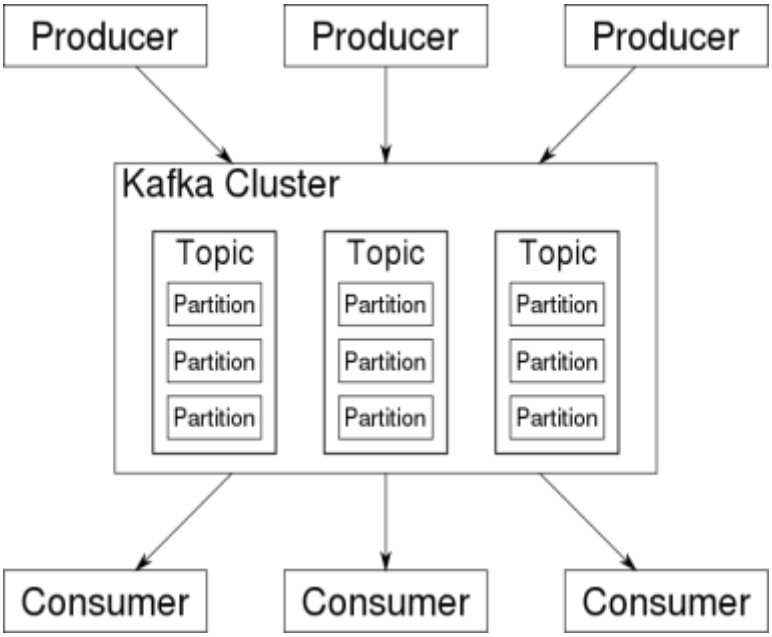


Figure 2. Apache Kafka Architecture

Kafka is deployed as a cluster on multiple servers, so Kafka handles its entire publish and subscribe messaging system with the help of four APIs *i.e.*, producer API, consumer API, streams API and connector API. Its ability to deliver massive streams of message in a fault-tolerant fashion has made it replace some of the conventional messaging systems like JMS, AMQP, *etc*. The major terms of Kafka's architecture are topics, records, and brokers. Topics consist of stream of records holding different information. On the other hand, Brokers are responsible for replicating the messages.

3. Kafka APIs:

There are four major APIs in Kafka.

3.1. Producer API:

It permits the applications to publish streams of records.

3.2. Consumer API:

It permits the application to subscribe to the topics and processes the stream of records.

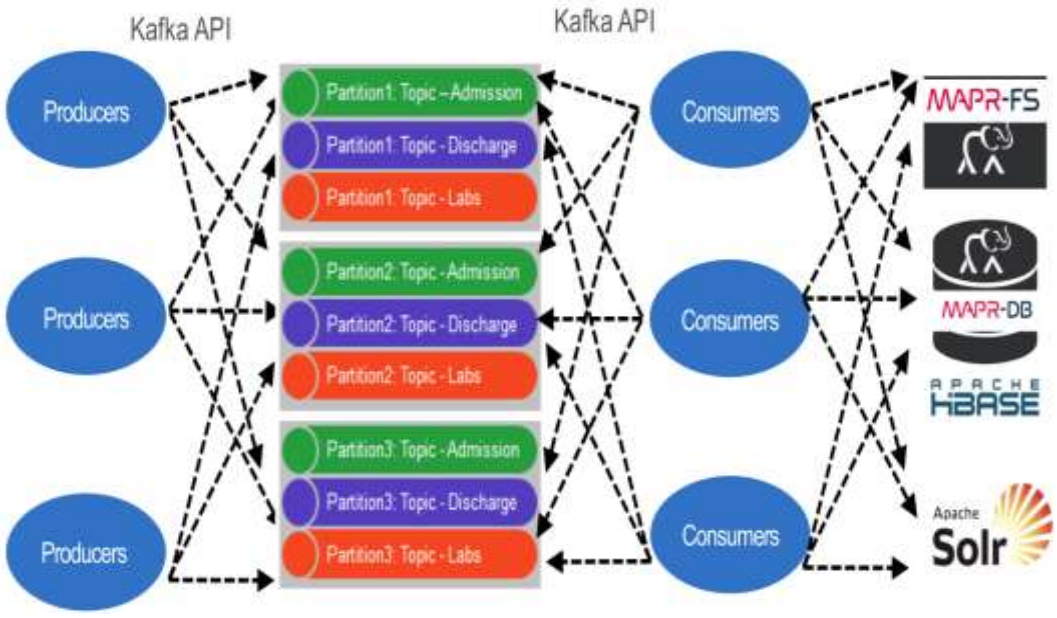
3.3. Streams API:

This API converts the input streams to output and produces the result.

3.4. Connector API:

It executes the reusable producer and consumer APIs that can link the topics to the existing applications.

Figure 3 is showing how different Producers are connected and streaming message to consumers.



4. Kafka ZooKeeper:

Before starting Kafka server we must have to start ZooKeeper as, ZooKeeper to do leadership election for Kafka broker and topic partition pairs in Kafka cluster and Zookeeper also provides multiple features for distributed applications like distributed configuration management, self election / consensus building, coordination and locks.

Kafka with ZooKeeper is used to manage service discovery for Kafka brokers of the Kafka cluster. ZooKeeper actually sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joins, a Broker dies, a topic was removed or a topic was added, *etc*., and ZooKeeper also offers an in-sync view of Kafka Cluster configuration.

5. Kafka Producer/Consumer and Topic:

Kafka producers write to Topics. Kafka consumers read from Topics. A topic is associated with a log which is data structure on disk. Kafka appends records from a producer(s) to the end of a topic log. A topic log consists of many partitions that are spread over multiple files which can be spread on multiple Kafka cluster nodes. Consumers read from Kafka topics at their cadence and can pick where they are (offset) in the topic log. Each consumer group tracks offset from where they left off reading. Kafka distributes topic log partitions on different nodes in a cluster for high performance with horizontal scalability. Spreading partitions aids in writing data quickly. Topic log partitions are Kafka way to shard reads and writes to the topic log. Also, partitions are needed to have multiple consumers in a consumer group work at the same time. Kafka replicates partitions to many nodes to provide failover and Kafka Producers, Comsumers and Topics have shown above in Figure 2.

6. Kafka Brokers:

A Kafka cluster is made up of Kafka Broker or multiple Brokers. Each Kafka Broker has a unique number called ID. Kafka Brokers have Topic log partitions. After connecting to a broker, bootstraps a client to the entire Kafka cluster after failover, we have to start with at least three to five brokers. A Kafka cluster can have 01, 10, 100, or 1,000 brokers in a cluster if needed. Kafka broker has shown above in Figure 1.

7. Kafka Connect

Kafka has a built-in framework called Kafka Connect for writing sources and sinks that either continuously ingest data into Kafka or continuously ingest data in Kafka into external systems. The connectors themselves for different applications or data systems are federated and maintained separately from the main code base.

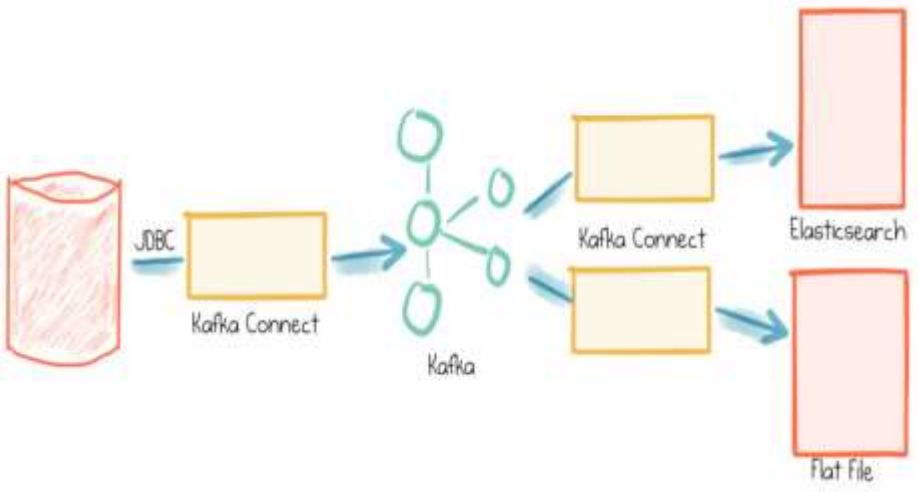


Figure 7. Stream Data from a Database (MySQL) into Apache Kafka® and from Kafka into both a Text File and Elasticsearch

8. Kafka Applications:

Kafka is broadly used in cloud by organizations in some combination of virtual private clouds say 34%, public clouds 52% and on premises 57%. Nearly one-third (32%) of respondents who use Kafka in the cloud have at least 6 Kafka applications in the cloud. Kafka is used in number of different ways for different use cases. In this year and in last year we found a surge of companies adopting Kafka streaming platforms to build mission critical, real-time applications that power their core business – all the way from small to large-scale use cases that handle millions of events per second. Now, companies used Kafka for "microservices". Microservices involve in many independent services. The goal of Microservices is broader than simply running them across different machines. It's about facing up to a world that is, itself, inherently distributed. Not in some narrow technical sense, but rather as a broad ecosystem composed from many people, many teams and many programs, all of which need the agility that microservices affords them. In current year, organizations use Apache Kafka: two-thirds (66%) use it for stream processing and three out of five (60%) use it for data integration. The most common use case for Kafka is data pipelines (81%), while half (50%) are already using it for microservices.

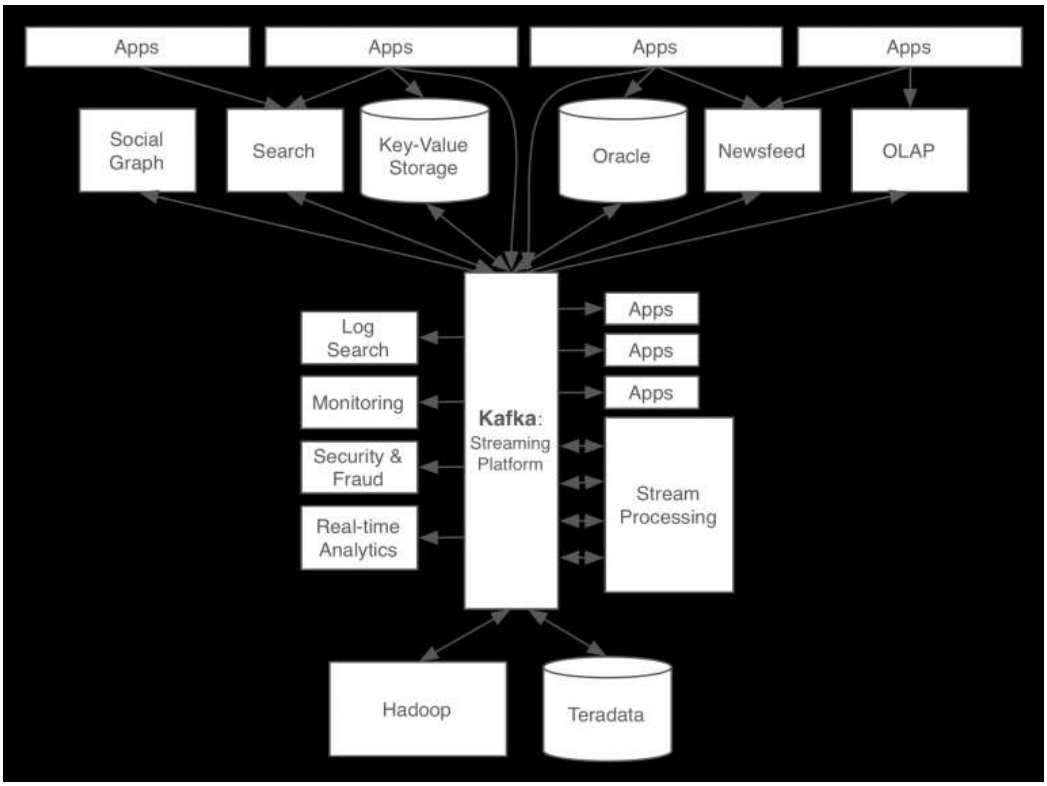


Figure 8. A Streamcentric Data Architecture Around Apache Kafka

In this diagram, Kafka acts as a universal pipeline for data and in this diagram it is showed that every system can feed into central pipeline or be fed by it *i.e.*, Apache Kafka; applications or stream processors can tap into it to create new, derived streams.

8.1. Kafka Data Pipeline:

In Figure 7 it is described that how to stream data from a database say MySQL into Apache Kafka and from Kafka into both a text file and Elastic search and do this all with the Kafka Connect API.

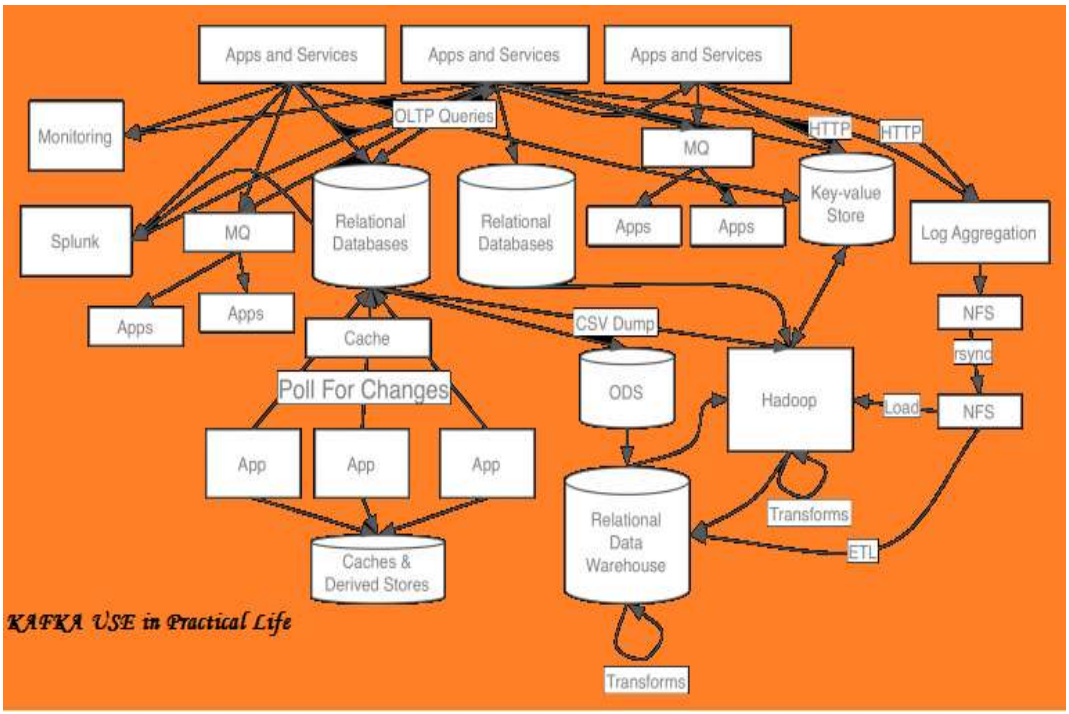


Figure 9. Apache Kafka Used with Different Application like HADOOP etc

9. KAFKA REAL TIME APPLICATIONS:

Here we have discussed some use cases for which Apache Kafka is used:

9.1. Messaging

Kafka works glowing as a substitute message broker which is used for a variety of reasons. Kafka has well throughput and built-in partitioning with replication, and fault-tolerance which makes it a good solution for large scale message processing applications. In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency and often depend on the strong durability guarantees Kafka provides.

9.2. Website Activity Tracking:

The original application of Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds means site activity like page views, searches, or other actions users may take is published to central topics with one topic per activity type. These feeds are available for subscription for a range of use cases including real-time processing, real-time monitoring, and loading into Hadoop or offline data warehousing systems for offline processing and reporting. Activity tracking is often very high volume as many activity messages are generated for each user page view.

9.3. Metrics

Kafka is often used for operational monitoring data. This indulges aggregating statistics from distributed applications to produce centralized feeds of operational data.

9.4. Log Aggregation

Kafka is also used as an alternative to log aggregation solution. Log aggregation Combines physical log files for servers and places them in a central processing location.

Kafka extracts file details and provides clearer extraction of log or event data as a as a stream of messages. This allows processing of low latency response time and easier support for multiple data sources and distributed data consumption. Compared with to log-centric systems such as Scribe or Flume, Kafka offers equally good performance, stronger durability guarantees due to replication, and much lower end-to-end latency.

9.5. Stream Processing

Kafka users are using Kafka to process data in processing pipelines of multiple stages, and raw input data are put into use in Kafka and then added, enriched or converted into new themes for subsequent consumption or tracking. For example, in order to use news articles, a workstation can scanned the content of the article in its RSS content on "articles"; Additional processing can normalize or reduce this content and publish the content of the pure article to a new topic; the last run may try to present this content to users. These processing pipelines create real-time data streams based on individual themes. According to 0.10.0.0, Apache Kafka has a light but powerful streaming library called Kafka Stream to perform data processing as described above. Apart from Kafka Streams, alternative tools for the development of open source script include Apache Storm and Apache Samza.

10. Conclusion

After many year struggling it has to focus on building a system that would focus on modeling streams of data that allow us to transport streams of data to all the systems and applications that needed them as well as build rich real-time applications which is intended to Apache Kafka. With the Kafka widespread integration into enterprise-level infrastructures, monitoring and Kafka performance at scale has become an increasingly important issue. Kafka monitoring and end-to-end performance requires tracking metrics from brokers, consumer, and producers, in addition to monitoring ZooKeeper which is used by Kafka for coordination among consumers[14][15]. Presently, several monitoring platforms are used to track Kafka performance, either open-source, like LinkedIn's Burrow, or paid, like Datadog. We have discussed how building a tool specifically designed around Kafka allows for stronger guarantees, better scalability, and simpler operationalization compared to other general purpose data copying tools. We have also discussed how combining Kafka Connect and Spark Streaming and to manage the complexity of building, maintaining, and monitoring large scale data pipelines

References

[1] “Mirror of Apache Kafka at GitHub”, github.com. Retrieved **(2017)** March 6.
[2] “Open-sourcing Kafka, LinkedIn's distributed message queue”, Retrieved **(2016)** October 27.
[3] “Cryptography and Protocols in Hyperledger Fabric”, (PDF), **(2017)** January, Retrieved 2017-05-05.
[4] “Kafka at HubSpot: Critical Consumer Metrics”.
[5] C. Park and A. Shankar, “Netflix: Integrating Spark at Petabyte Scale”.
[6] B. Svingen, “Publishing with Apache Kafka at The New York Times”, Retrieved 2017-09-19.
[7] S. Sudhakaran, “PayPal: Creating a Central Data Backbone: Couchbase Server to Kafka to Hadoop and Back (talk at Couchbase Connect 2015)”, Couchbase. Retrieved 2016-02-03.
[8] “Shopify - Sarama is a Go library for Apache Kafka”.
[9] “Concurrency and at Least Once Semantics with the New Kafka Consumer”.
[10] J. Baer, “How Apache Drives Spotify's Music Recommendations”.
[11] P. Hechinger, “CTOs to Know: Meet Ticketmaster's Jody Mulkey”.
[12] “Apache Kafka for Item Setup”, Medium.com. Retrieved 2017-06-12.
[13] “Streaming Messages from Kafka into Redshift in near Real-Time”, Yelp. Retrieved 2017-07-19.
[14] “Monitoring Kafka performance metrics”, 2016-04-06. Retrieved 2016-10-05.
[15] E. Mouzakitis, “Monitoring Kafka performance metrics’, datadoghq.com. Retrieved 2016-10-05, (2016-04-06).

Author

Javed Ahmed Shaheen is presently working as SST (Computer Science) in PSED BWN; he has done his MSCS (Networking) from Virtual University of Pakistan Lahore. He has practical experience of Network installation.