International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 3, May -June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

# Kafka: A Distributed Event Streaming Framework for Micro Service Based Applications

Gowtham H N*, Deepika Dash**

*(Computer Science and Engineering, R V College of Engineering, Bengaluru, India
Email: gowthamhn.cs17@rvce.edu.in)

** (Computer Science and Engineering, R V College of Engineering, Bengaluru, India
Email:deepikadash@rvce.edu.in)

----------------------------------------************************----------------------------------

## Abstract:

Big data processing involves models and techniques to process huge volumes of data increasing at an exponential rate that has made it difficult for traditional data management tools to process. Data streaming is a popular technique that enables real time processing of data for real-time data analysis. Currently Apache Kafka is one of the most popular data streaming framework that provides fast, highly scalable and distributed processing of big data. It is a Produce-Consume messaging service known for its high throughput, low latency and high reliability. In this paper we analyze the architecture, working and performance of Kafka.

*Keywords* **—Big Data, Data Streaming, Apache Kafka, Distributed Messaging.**

----------------------------------------************************----------------------------------

## I. INTRODUCTION

Data stream processing refers to processing of data in motion as it is produced instead of the traditional storing in a database and query later process.[1] An application listening to a data stream can react by triggering an action, to a message as soon as it is received. A messaging system allows transferring data from one application to another, which is of extreme necessity for distributed systems especially micro service-based applications. This way the application can solely focus on processing data without worrying much about how the data is shared among the different computes. Apache Kafka is one of the most popular open-source data streaming or messaging platform used by hundreds of companies across the globe. It was originally developed by LinkedIn to process their real time log data written mainly in Java and Scala. Its attractive features are high speed, scalability, distributed processing, durability, low latency, high throughput and reliability.

## II. ARCHITECTURE

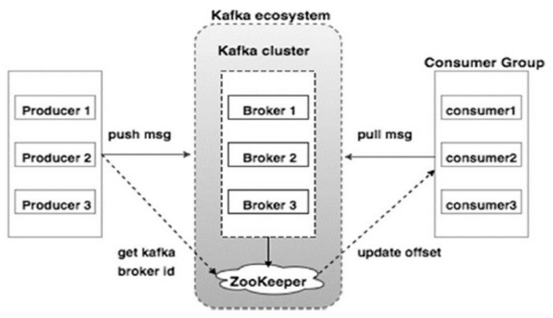


Fig. 1Apache Kafka Architecture

Kafka is based on a Publish-Subscribe architecture[3]. It consists of servers and clients that communicate with each other. The servers are a collection of systems that can span across multiple data centres. If any of the servers go down, other servers take responsibility and ensure good fault tolerance. Clients enable applications to publish or subscribe to Kafka. Kafka Clients are available for various languages such as Java, Scala, C/C++, Python etc.,

The main components of Kafka are as follows[2],

- Topics: A Kafka topic is a virtual group of similar ordered messages.
- Producers: A producer is a client or application that produces messages to Kafka topics.
- Consumers:A consumer is a client or application that consumes messages from Kafka topics.
- Partitions: Topics are divided into partitions and placed on separate machines of a cluster so that a topic can be parallelized.
- Consumer groups: A group of consumers where each partition is consumed by exactly one consumer.
- Brokers: A broker is a Kafka server.
- Clusters: A cluster is a group of brokers that work together with additional features such as redundancy, load balancing and reliable fail-over.
- Partitions: Topics are divided into several partitions in order to achieve parallelism.
- ZooKeeper: ZooKeeper is a distributed service synchronizer that is used to track the status of nodes in the cluster.

## III. WORKFLOW

Before we begin producing and consuming messages using Kafka, we need to configure Kafka Eco-system starting from brokers to clients. These configurations should be provided as a key value pairs. Once the configurations are set, zoo-keeper and Kafka server can be started. A basic Kafka environment is set up and running now. Applications can publish and consume messages from Topics created.

### A. *Creation of topics*

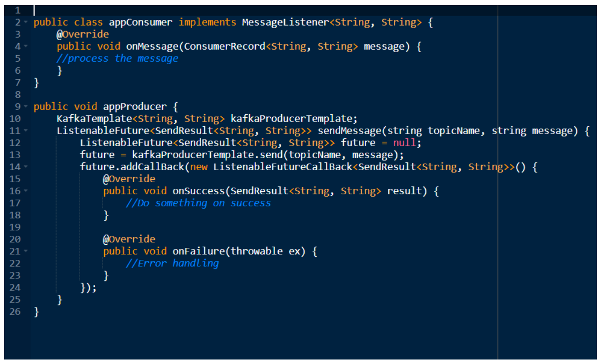
The messages that Kafka works with are divided into categories called topics. It is analogous to a folder in which for similar messages are stored. Since Kafka is a distributed system, these topics are divided and replicated across multiple nodes of a cluster. When creating a Kafka topic details such as the replication factor, number of partitions, topic name, zookeeper address needs to be specified.

### B. *Setting up clients*

Kafka provides an API interface for all its clients.In this paper we'll consider a Java Spring Boot Application for demonstration of Kafka Producer and consumer setup and their workflow. Although the clients can use any of the several servers of a cluster, the address of any one of the servers belonging to the cluster should be mentioned under the bootstrapserver field for initial connection. A single producer can publish messages to multiple Kafka topics and a single consumer can subscribe to multiple topics as well.

### C. *Kafka in action with the application*

Once all the setup is done, we are ready to push and consume messages from applications.Fig 2 shows a sample code for publishing and consuming events with a java spring boot application.



```java

public class appConsumer implements MessageListener<String, String> {
    @Override
    public void onMessage(ConsumerRecord<String, String> message) {
    //process the message
    }
}

public void appProducer {
    KafkaTemplate<String, String> kafkaProducerTemplate;
    ListenableFuture<SendResult<String, String>> sendMessage(string topicName, string message) {
        ListenableFuture<SendResult<String, String>> future = null;
        future = kafkaProducerTemplate.send(topicName, message);
        future.addCallBack(new ListenableFutureCallBack<SendResult<String, String>>() {
            @Override
            public void onSuccess(SendResult<String, String> result) {
                //Do something on success
            }

            @Override
            public void onFailure(throwable ex) {
                //Error handling
            }
        });
    }
}
```

Fig. 2Sample code to publish and consume message in a spring boot application.

An application based on a micro-service architecture will have several component services running independently that interact with each other through Kafka[2]. One benefit that can be reaped out of Kafka is the horizontal scalability. With Kafka as the data store the application can be distributed and horizontally scaled easily by just adding more brokers and compute instances.Fig. 3

shows how a system that is horizontally scaled to use 6 compute instances.

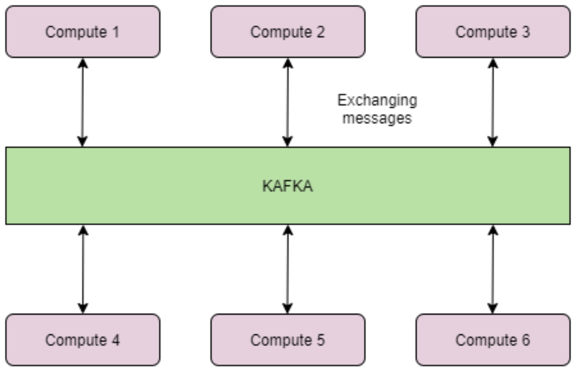


Fig. 3Horizontal scaling with Kafka.

These compute instances can be grouped into consumer groups so that exactly one of these processes a particular event. In this fast pace IT world migrating applications from legacy technologies is a difficult task. Although Kafka will not be able to completely replace traditional databases, it acts as a powerful complement. Kafka provides seem-less integration with legacy databases such as SQL as well as NoSQL databases. The scalability of an application using SQL database could be limited. With the help of Kafka this limit could be extended to a higher number of compute instances hence improving the response time and fault tolerance of the application.

## IV. COMPARISION WITH RABBITMQ

RabbitMQ is another popular framework that allows applications to send and receive messages within its components. While both have completely different architectures, they serve very similar purpose. The main difference between the two is the data structure in which the messages are stored. Kafka uses an append only log structure whereas the RabbitMQ uses a traditional queue structure. Most of the time applications have the use case of reading a message, processing it, and publish another message. For such an use case, Kafka with its Kafka streams feature is well suited. Although RabbitMQ provides a sense of consistency by keeping track of read status of messages, Kafka overshadows RabbitMQ with its performance. It provides higher throughput with limited amount of resource which is an important factor in big data processing. Kafka also provides a history of the data stream. When an application needs support for legacy protocols such as AMQP, RabbitMQ is useful. But with new applications that require complex routing, event sourcing or multi-line pipelined stages of processing of messages, Kafka is the preferred choice[4]. However, it is important to know RabbitMQ can be used for a lot of the use cases best suited for Kafka, but only with the help of other tools like Apache Cassandra.

## V. PERFORMANCE EVALUATION

The performance of Kafka when producers and consumers are run separately is evaluated here. The experiments were done with a single Kafka server. The main performance evaluation parameter considered was the throughput compared against the size of the message[5]. From fig 4. it can be observed the number of messages processed concurrently drops as the message size increase. However, the total throughput in kbps is still increased as the message size increases. In most of the micro-service based applications the size of the messages or events being exchanged is less than 1kb. Hence the throughput obtained from Kafka is pretty good.

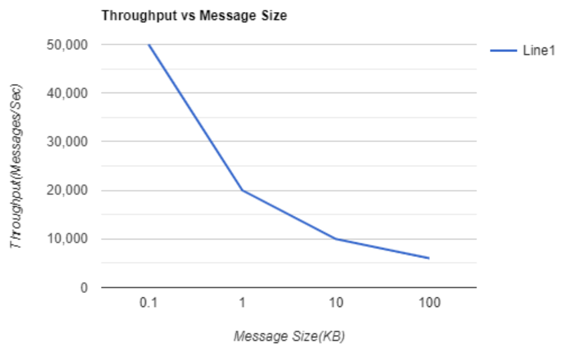


Fig. 4Throughput(Messages/sec) vs Message size.

## VI. CONCLUSIONS

In this work, the emphasis has been on answering questions such as What? How? and Why? Kafka. An overall idea about Kafka, its architecture, setup and how it can be used over its alternatives is provided along with a sample code snippet. Evaluation of performance based on other parameters as well as tuning of performance is left for future work. With the continuous developments and updates, there is no doubt that Kafka will form an integral part of micro service-based applications.

## ACKNOWLEDGMENT

We would like to thank all the faculty of RV College of Engineering and others for their knowledge transfer and the support provided for this work on Kafka.

## REFERENCES

[1] B. R. Hiraman, C. Viresh M. and K. Abhijeet C., "A Study of Apache Kafka in Big Data Stream Processing," 2018 International Conference on Information , Communication, Engineering and Technology (ICICET), 2018, pp. 1-3, doi: 10.1109/ICICET.2018.8533771.

[2] R. Shree, T. Choudhury, S. C. Gupta and P. Kumar, "KAFKA: The modern platform for data management and analysis in big data domain," 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 2017, pp. 1-5, doi: 10.1109/TEL-NET.2017.8343593.

[3] Shaheen, J. A.. “Apache Kafka: Real Time Implementation with Kafka Architecture Review.” International journal of advanced science and technology 109 (2017): 35-42.

[4] Dobbelaere, P., &Esmaili, K.S. (2017). Kafka versus RabbitMQ. ArXiv, abs/1709.00333.

[5] P. Le Noac'h, A. Costan and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications," 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 4803-4806, doi: 10.1109/BigData.2017.8258548.

[6] B. Yadranjiaghdam, N. Pool and N. Tabrizi, "A Survey on Real-Time Big Data Analytics: Applications and Tools," 2016 International Conference on Computational Science and Computational Intelligence (CSCI), 2016, pp. 404-409, doi: 10.1109/CSCI.2016.0083.