Development of Kafka Messaging System and its Performance Test Framework using Prometheus

Tejas V, Kiran V

Abstract: ***In today's era of internet, communication plays a vital role in every aspect of daily activity, as the number of internet users increases an enormous volume of data is generated. This motivates to develop and use a messaging system that collects and analyzes large volume of generated data. Kafka is a distributed messaging system based on publish-subscribe model with high throughput and fault-tolerant mechanism. Around 40% of fortune 500 companies use Kafka messaging system for processing data. Kafka is used to stream real-time data and it is also used to process the data received from IOT sensors. This paper discusses about the procedure for developing and deploying the Kafka messaging system into the working environment based on container technology. The secure communication is enabled by configuring the Kafka with two security mechanisms. The performance of the developed system is evaluated considering QoS parameters for different test scenarios using Prometheus tool.***

Keywords: Communication, data, Internet of Things, Kafka, QoS parameter

I. INTRODUCTION

Owing to the rapid growth of distributed computing, the implementation and growth of distributed systems has become ever more profound in recent years. For instance, high volumes of messages are produced continuously in some sizeable distributed systems. To process these messages effectively, developers need to find an efficient way to rapidly capture and distribute terabytes of messages. This requirement demands a kind of robust business model. In the article [1] the challenges faced by the remote server in data accumulation were identified and a Netty framework was used to adopt Kafka as an intermediate layer to achieve asynchronous communication. The test were conducted by using a single message size which is not the practical case. The system working performance were evaluated by running the producers and consumers separately [2]. The overall load experienced by Kafka system was monitored using Zookeeper service to ensure the continuous service availability [3]. The potential advantage of the Containerization technology over the previously existing VM technology is discussed in paper [4][5]. Considering the advantages of the Containerization methods [6], Kafka deployment is made using the Docker and Containarization mechanism [7]. The schematic view of the Kafka model is shown in Fig 1.

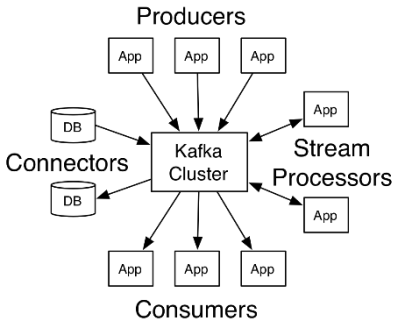

Fig 1: Schematic view of Kafka [1]

The model is based on implications that producers initially produce messages, and the produced messages are consumed later by the consumers. Kafka serves as a link between producers and consumers so that they can interact well with each other.

II. THEORY AND METHODOLOGY

A. Kafka Architecture

Kafka is a distributed publish-subscribe messaging system and a powerful queue that can accommodate a large amount of data and allows users to transfer messages from one endpoint to another. Kafka is suitable for consumption of both offline and web communications. The architectural view of Kafka is show in Fig 2. Kafka is built by making use of Zookeeper's synchronization service. Kafka messages remain on the disk and are repeated within the cluster to avoid loss of data.

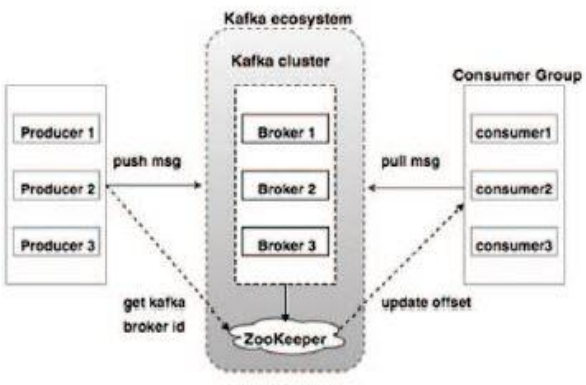

Fig 2: Architectural view of Kafka [2]

Revised Manuscript Received on May 21, 2020.

***** Correspondence Author

Tejas V*, department of Electronics and Communication Engineering, R V College of Engineering. Bengaluru, India. Email:tejasteju122@gmail.com

Dr. Kiran V, department of Electronics and Communication Engineering, R V College of Engineering. Bengaluru, India., Email: Kiranv@rvce.edu.in

The different terminologies associated with Kafka are:

1. Kafka Broker: In order to achieve load balance Kafka cluster consists of number of Kafka brokers. Due to stateless feature of Kafka brokers, Zookeeper services are used to preserve their cluster state.

2. Zookeeper: Kafka makes use of zookeeper service to manage the cluster. Zookeeper is used within the cluster to organize and handle Kafka brokers. Whenever a new broker is added into the Kafka cluster or when the available broker goes into failure state the zookeeper notifies the producers and consumers about the same.

3. Topic: A topic is the name of the category/feed name into which the messages are entered during publishing event and read from during the subscribing event. All the messages within the Kafka are organized into topics. There is further flexibility in topic creation by specifying the number of partitions within a single topic.

4. Producers: The messages are published into the Kafka topics by the producers. The messages can be produced to one or more topics simultaneously. The data is sent into the Kafka brokers. Producers can also write into a particular partition within a topic by specifying the partition offset value within a topic

5. Consumers: Consumers are the subscriber of the messages from the Kafka topics. Consumers read the published messages from the Kafka brokers by subscribing into the relevant topic.

6. Connectors: Connectors are used in stream processing application. It serves as an intermediate link for relaying stream of data from Producers and delivering the stream of data to Consumers.

B. Methodology Adopted

The development of the Kafka messaging system is based on Docker and Containerization technology. Before the emergence of containerization technology, Virtual machine (VM) had been the technology of choice for server resource optimization. Since each VM includes an Operating System and a virtual copy of all the hardware needed by the OS, VMs need substantial amount of RAM and CPU resources and increases software development life cycle. Containarization technology is preferred to overcome the drawbacks of Virtual Machine technology. A Docker container image is a lightweight, standalone, executable software package that consists everything required to run an application including code, runtime, machine tools. During runtime container images created become containers. Containers are available for both Linux and Windows based applications. The Kafka messaging system is installed on the kubernetes [7] cluster via helm charts. Helm manages Kubernetes resource packages through Charts. A chart is a accumulation of files organized in a specific directory structure to create a Kubernetes application. The two security mechanism are implemented to safeguard the confidential information exchanged through the Kafka cluster. After the development and deployment phase the Kafka system is monitored and its performance is evaluated for QoS parameters by integrating Kafka with Prometheus/grafana monitoring tool. The Kafka is monitored at regular intervals for resource optimization to ensure continuous service availability.

C. Experimental Details

The diagram, shown in the Fig. 3, depicts the experimental design of the research carried out in this paper work and represent the Kafka system operation process.

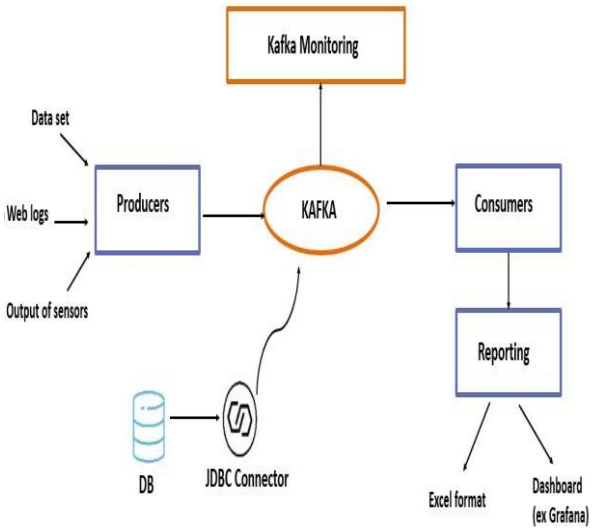

Fig. 3: Schematic representation of Kafka system operation

In order to deploy the Kafka system on the Kubernetes cluster through containerization method a Docker [8] file is created from which the docker image is extracted. The running instance of the docker image becomes containers which are deployed on the cluster. Helm is used to manage Kubernetes resource through charts. The chart is a collection of file organized in a source-tree structure to create a Kafka application on the kubernetes cluster. Once the Kafka is deployed on the kubernetes cluster the Kafka system is integrated with the Prometheus/Grafana monitoring tool as shown in Fig 4.

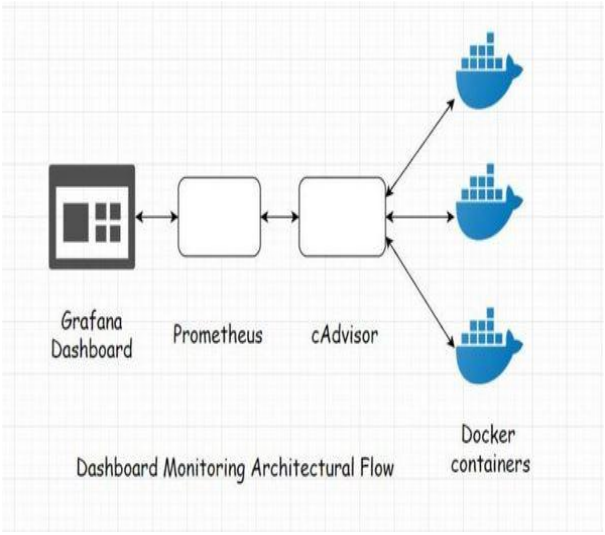

Fig 4: Integration of Kafka with Prometheus monitoring tool

The messages produced into the Kafka brokers can be from particular database, logs collected
from a particular website or the data received from the sensors. The data stored in Kafka can be subscribed at any instant of time. The performance is evaluated for Kafka by considering QoS parameters such as throughput, latency, error rate using the monitoring tool. The Fig 5 describes the work flow of JMX (Java Management Extensions) with Kafka. To collect the MBeans (Managed Beans) value the JMX port has to be set. A query is made with the name MBeans to obtain the metrics value that acts as a performance indicator for Kafka.

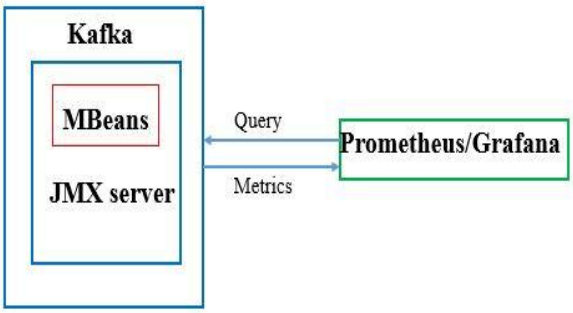

Fig 5: JMX port connection method [2]

The query associated with Metric is summarized in Table1.

Table I: JMX Queries

Metrics name	Query
Bytes in per sec	kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec
Bytes out per sec	kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec
Total time (ms)	kafka.network:type=RequestMetrics, name=TotalTimeMs,request={Produce|FetchConsumer}
CPU usage	rate(cpu_seconds_total(mode="idle",instance))
Error rate	rate(kafka_network_requestmetricscount(name="Error per second"))

III. RESULTS AND DISCUSSSION

Once the cluster is created and the cluster is in running state, the Kafka system is deployed on the running cluster using helm command. The release name and namespace must be specified during the Kafka deployment to uniquely identify the instance within the cluster. Once the installation of the Kafka system is made on the cluster. The deployment status of the Kafka is verified using Helm list command. The performance of the messaging system is assessed by altering the number of records produced and consumed from the Kafka system. Various QoS parameters are considered for the performance indication. The results were observed for four different test scenario. The result of one of the test scenario for processing 10000 messages is shown in the Fig [6]-[12]

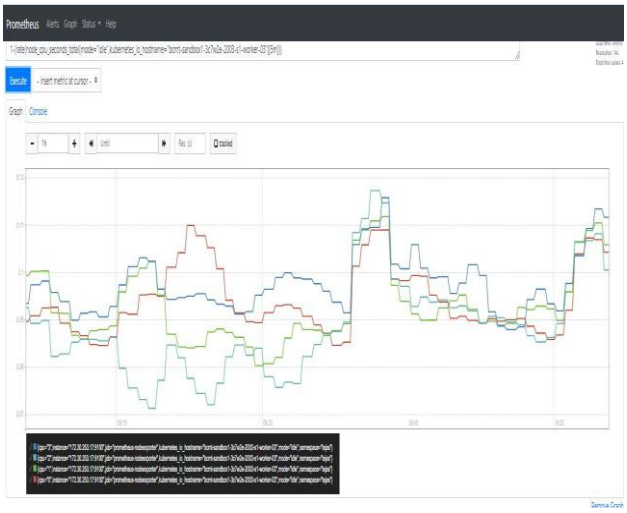

Fig 6: CPU usage graph 10000 messages processed

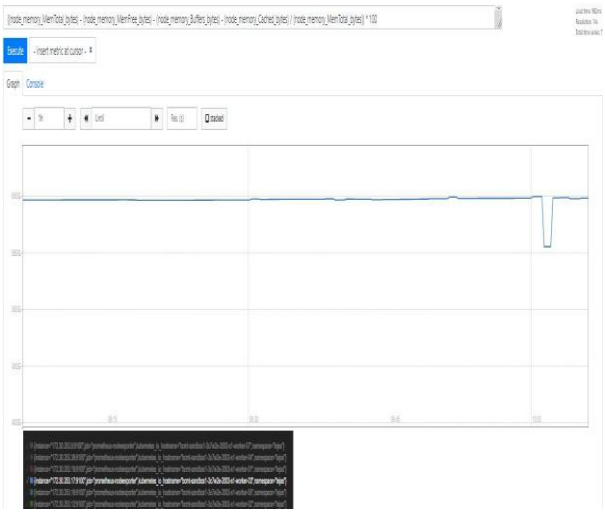

Fig 7: Memory usage graph for 10000 messages processed

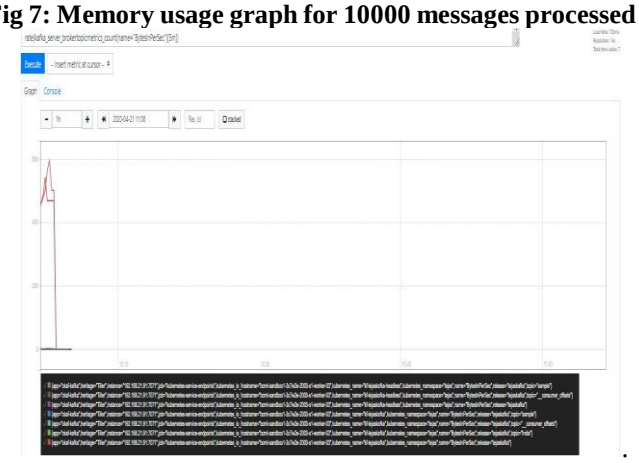

Fig 8: Bytes in per second for 10000 messages processed

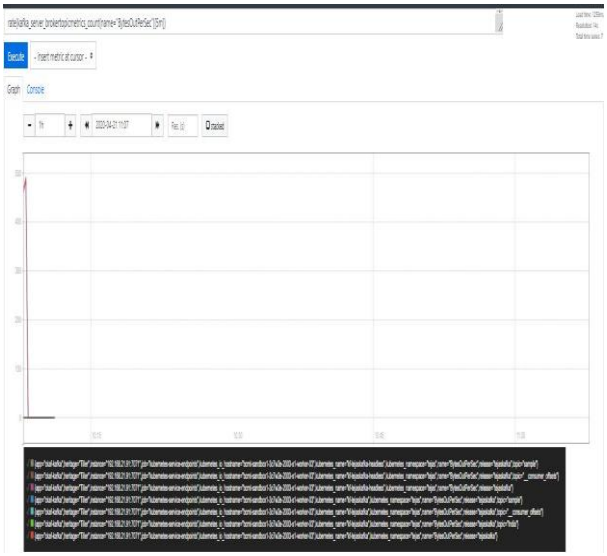

Fig 9: Bytes out per second for 10000 messages processed

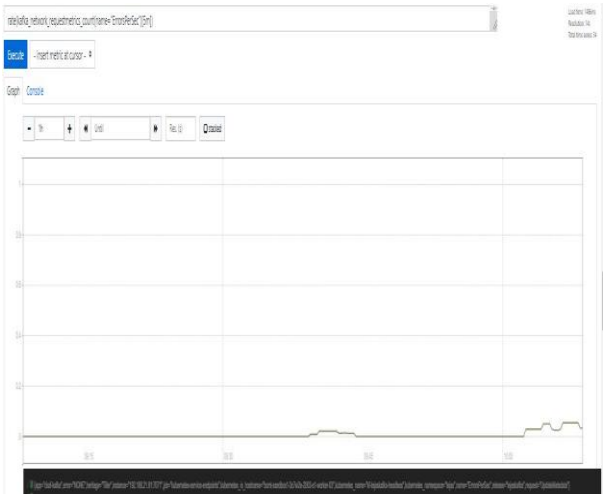

Fig 10: Error per second for 10000 messages processed

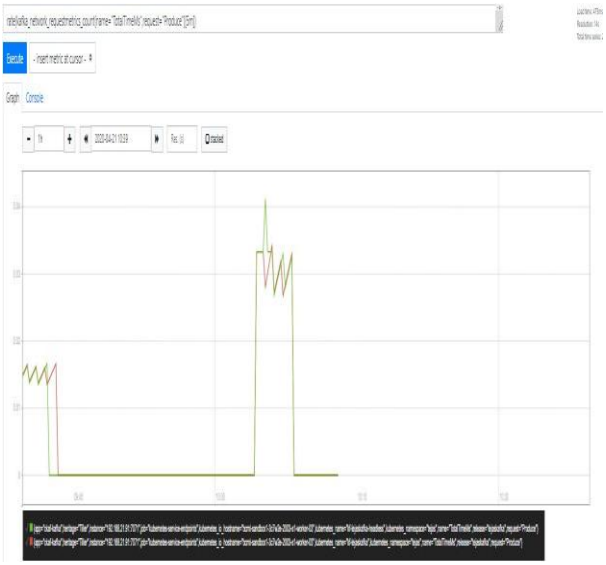

Fig 11: Total processing time for 10000 messages processed

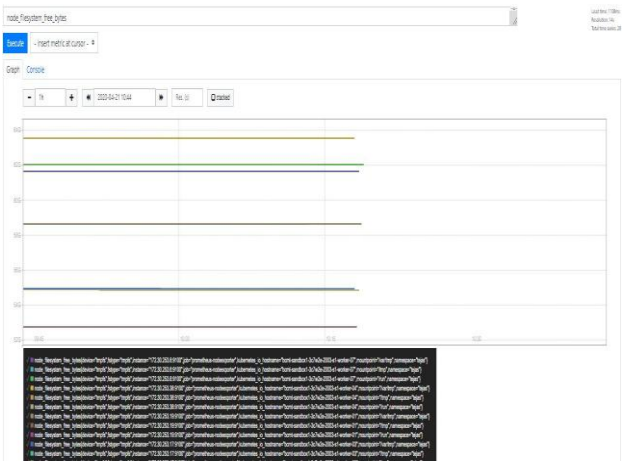

Fig 12: Disk usage graph for 10000 messages processed

The entire metrics evaluation for all the four test cases is summarized in the Table II

Table II: Performance results for different test cases

Metrics	Case1	Case2	Case3	Case4
Topic name	Rvce	Bangalore	Karnataka	India
Number of Messages	100	1000	5000	10000
CPU usage(m core)	0.065	0.075	0.11	0.12
Memory usage(MB)	0.008	0.05	0.09	0.1
Disk available (Gigabytes)	55	54.5	54.3	54.2
Bytes In per sec	5	50	250	500
Bytes Out per sec	6	50	270	510
Total Time(ms)	0.004	0.0045	0.015	0.04
Error per sec	0	0	0.2	0.2

From the results tabulated in Table II it can be concluded that the developed messaging system is a high throughput, fault tolerant system with negligible latency by observing the bytes in/sec, bytes out/sec , total processing time metrics .

The other metrics such as CPU usage, Memory usage, Disk usage must be monitored continuously to optimize the resources based on demand to ensure continuous service availability.

IV. CONCLUSION AND FUTURESCOPE

This project was intended to develop a high throughput, low latency, fault tolerant messaging system and evaluate the performance of the messaging system considering different QoS parameters. This type of messaging model offers advantage over the traditional messaging model in terms of operational throughput and reduced latency. The deployment of the Kafka is based on containarization technology which further ensures resource optimization and portability across different operating platforms and clouds. In order to ensure secure communication two security mechanisms such as SASL and SSL mechanisms are implemented. From the obtained results it can be concluded that the developed Kafka system has high throughput and negligible delay while processing huge volumes of messages. However during the research process it was found that the development of Kafka requires lot of manual work. As future work in order to ease the development work and reduce human errors during development more practical tools can be involved and integrated into Kafka. The further in depth analysis of Kafka could be made by considering other QoS parameters such as purgatory size, input output wait ratio etc.

ACKNOWLEDGMENT

The writers would like to thank Mr. Ramesh K Sridharamurthy, Technical Manager, NOKIA Networks and solutions for his support and assistance during the entire work process.

REFERENCES

1. Z. Yuan, B. Pang, Y. Du, X. Liu, J. Yao and C. Kong, Design and Implementation of Internet of Things Message Subscription System Based on Kafka, *IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, Chongqing, China, 2019, pp. 603-606.
2. H. Wu, Z. Shang and K. Wolter, Performance Prediction for the Apache Kafka Messaging System, *IEEE 21st International Conference on High Performance Computing and Communications*, Zhangjiajie, China, 2019, pp. 154-161
3. M. Song, G. Luo and E. Haihong, A Service Discovery System based onZookeeper with Priority Load Balance Strategy, *IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, Beijing, 2016, pp. 117-119.
4. S. Singh and N. Singh, Containers & Docker: Emerging roles & future of Cloud technology, *2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Bangalore, 2016, pp. 804-807
5. V. Medel, O. Rana, J. Á. Bañares and U. Arronategui, Modelling Performance & Resource Management in Kubernetes, *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, Shanghai, 2016, pp. 257-262
6. G. Kambourakis, A. Rouskas and S. Gritzalis, Using SSL/TLS in authentication and key agreement procedures of future mobile networks, *4th International Workshop on Mobile and Wireless Communications Network*, Stockholm, Sweden, 2002, pp. 152-156.
7. P. Le Noac'h, A. Costan and L. Bougé, A performance evaluation of Apache Kafka in support of big data streaming applications, *IEEE International Conference on Big Data (Big Data)*, Boston, MA, 2017, pp. 4803-4806
8. Preeth E N, F. J. P. Mulerickal, B. Paul and Y. Sastri, *Evaluation of Docker containers based on hardware utilization, International Conference on Control Communication & Computing India (ICCC)*, Trivandrum, 2015, pp. 697-700.

AUTHORS PROFILE

Tejas V, is currently pursuing his post-graduation degree in Communication systems from the Department of Electronics and Communication Engineering, R V college of Engineering, Bengaluru, India. He obtained his Bachelor of Engineering degree from department of Electronics and Communication Engineering, BNM institute of Technology, Bengaluru in 2017. He has previously published a paper on smart electronics driver assistance aid at IEEE conference. His area of interest include Communication systems, Image processing, Computer Networks

Dr. Kiran V, is working as an Associate professor, Department of Electronics and Communication Engineering, R V college of Engineering, Bengaluru, India. He completed his Master of technology degree on Digital Electronics and Communication later completed his PhD on Communication and signal processing in the year 2019 from Visveswaraya Technological University, Belagavi, India. He has a teaching experience of 14 years in various subjects such as Analog communication, Information theory and decoding, Data communication. He has presented 13 International conference papers, 6 National papers and his work has been published in 7 journals.