Modern Messaging Queues - RabbitMQ, NATS and NATS Streaming

Poojya J Bhat, Priya D

Abstract—Distributed messaging structures shape the core of massive microservice architecture , cloud native applications and data streaming as they are used to communicate between different application services. With actual-time crucial programs there is a developing need for well-constructed messaging platform this is fault tolerant, has low latency and scalable. This paper surveys various message broker that are in vogue today. These modern message brokers have their own adavantages and disadvantages that have come up lately. There is need for the comparative study to decide which broker is most suitable for a specific appalication. An in-depth study is required to decide which features of a messaging system meet the needs of the application. The paper outlines information about three messaging systems – RabbitMq, Nats and Nats-Streaming and explores the features they offer as well as their performance under varied testing workloads..

Keywords: NATS, NATS-Streaming, RabbitMQ, distributed messaging systems

I. INTRODUCTION

Distributed Message Brokers are normally used to decouple separate ranges of a software program structure. They are of great help in conversation between these tiers asynchronously, through the use of the pub-sub paradigm and also request-reply paradigm[1] Dispersed frameworks are currently across the board and have more than a large number of substances. They are continually developing. This underlines the requirement for cross stage specialized technique that can adjust to various conventions and is dynamic in nature. Point to point and synchronous communication do not adapt to the dynamic nature of applications. A versatile and approximately coupled framework, for example, the distribute buy in framework [2] is appropriate for the present market needs. Another comparative framework are message lines. Continuous site following, information examination, ongoing blast in IoT gadgets and logging has expanded the interest for profoundly accessible , deficiency lenient, informing frameworks. They structure the middleware foundation for microservices , cloud-based applications and enormous information streaming. Message Brokers along with in-memory database like redis are go to in log handling and live streaming.

A. Publish/Subscribe.

Publish-subscribe pattern is a mechanism where consumers subscribes to one or more topic/subject agreed upon by both parties (producer and consumer) and the producer publishes the message into those topics/subjects through broker servers.

Endorsers tuning in regarding a matter get messages distributed regarding that matter. On the off chance that the supporter isn't effectively tuning in regarding the matter, the message isn't gotten. Supporters can utilize the trump card tokens, for example, * and > to coordinate a solitary token or to coordinate the tail of a subject. Fig.1 shows the diagram of the publisher-subscriber design with the publishers, subscribers and broker(server). Key to these system is they are decoupled in terms of Time, Space and Synchronization[2]. So the producer and subscribers can be active at any time and hey are unkonwn to each other. They are utilized for extortion identification, reconnaissance, aviation authority and algorithmic exchanging.

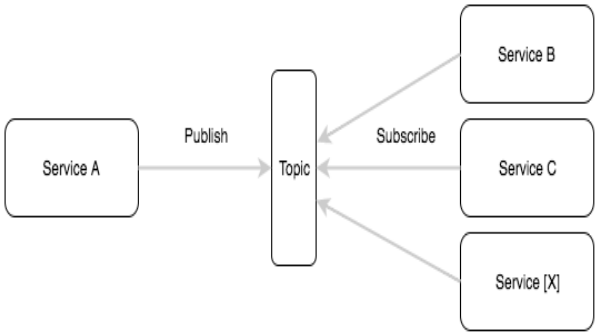
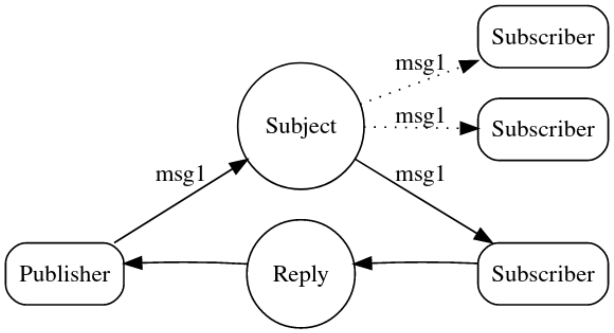


Fig.1 Publisher-Subscriber System

B. Request/Replay

A request is sent and the application either looks out for the reaction with a certain break or gets a reaction non-concurrently. The expanded unpredictability of current frameworks requires highlights, for example, area scale up, downsize and straightforwardness. Numerous innovations need extra segments, sidecars and intermediaries to achieve the total list of capabilities. Fig.2 shows the diagram of the request-replay design. A request is distributed on a given subject with an answer subject, and respondents tune in regarding that matter and send reactions to the answer subject. Answer subjects are one of a kind subjects called inbox that are powerfully coordinated back to the requester, paying little heed to area of either party.

Revised Manuscript Received on June 30, 2020.
Poojya J Bhat, Student, R V College of Engineering, Bangalore, India.
Priya D, Assistant Professor, R V College of Engineering, Bangalore, India.



II. OVERVIEW

The famous low dormancy Message Brokers incorporate Amazon Kinesis, NATS Streaming, RabbitMQ [8],Apache Kafka [9], Redis [10], , Microsoft occasion centers and Google pub/sub.

These frameworks are used in systems which has enormous measure of information handling, which must be tended to by a sturdy, all around organized information gushing structure. The accompanying section gives a concise picture about Rabbit MQ, NATS and NATS Streaming followed by examination of these three structures.

A cutting edge informing framework needs to help different correspondence designs, be secure as a matter of course, bolster numerous characteristics of administration, and give secure multi-tenure to a genuinely shared foundation. A cutting edge framework needs to include:

- Secure of course interchanges for microservices, edge stages and gadgets.
- Secure multi-tenure in a solitary dispersed correspondence innovation.
- Straightforward area tending to and disclosure.
- Strength with an accentuation on the general soundness of the framework.
- Convenience for dexterous turn of events, CI/CD, and activities, at scale.
- Exceptionally adaptable and performant with worked in load adjusting and dynamic auto-scaling.
- Predictable character and security instruments from edge gadgets to backend administrations.

A. RabbitMQ

RabbitMQ is a message-queueing software called a message broker or queue manager. It is a software where queues can be defined, applications may connect to the queue and transfer a message onto it. RabbitMQ, and messaging in general, uses some jargon I.e, Producer / Sender, Consumer / Receiver and Queue /(large message buffer) - Producers and consumers can send data to or receive data from one or more queue.

It is publicly released and consolidates Advanced Message Queuing Protocol (AMQP). It empowers consistent nonconcurrent message-based correspondences between applications. The core idea in the messaging model in RabbitMQ is that the producer never sends any messages directly to a queue. Producer can only send messages to an exchange. Exchange receives messages from producers on one side and on the other side it pushes them to queues. The messages shipped are approximately coupled, for example the sender and the collector frameworks need not be fully operational simultaneously.

RabbitMQ is written in Erlang Programming language. It is lightweight and can be sent on cloud. It is language skeptic [7]. It bolsters dialects like – Ruby, Python, C, C++ .NET and so forth. It tends to be sent and utilized across different working frameworks.

Messages are imparted over TCP associations. Significant parts associated with RabbitMQ are Publisher, Consumers, Exchange and Queues. A coupling is a "connect" or a standard that chooses the course of a message to a specific line. Each message that gets distributed in line contains a payload and a steering key. The directing key chooses the specific line where a message should be conveyed.

AMQP 0-9-1 (Advanced Message Queuing Protocol) is an informing convention that empowers adjusting client applications to speak with accommodating informing middleware representatives.

The AMQP 0-9-1 Model has the accompanying perspective on the world: messages are published to *exchanges*, frequently contrasted with post workplaces or letter boxes. Exchanges at that point circulate message duplicates to *queues* using rules called *bindings*. . At that point the broker either convey messages to consumers bought in to lines, or consumers get/pull messages from queues on request.

AMQP 0-9-1 is a programmable convention as in AMQP 0-9-1 substances and directing plans are fundamentally characterized by applications themselves, not an agent director. As needs be, arrangement is made for convention activities that proclaim lines and exchanges, characterize ties between them, buy in to line, etc. This gives application engineers a great deal of opportunity yet additionally expects them to know about potential definition clashes. Practically speaking, definition clashes are uncommon and regularly show a misconfiguration. Applications proclaim the AMQP 0-9-1 substances that they need, characterize fundamental directing plans and may decide to erase AMQP 0-9-1 elements when they are not, at this point utilized.

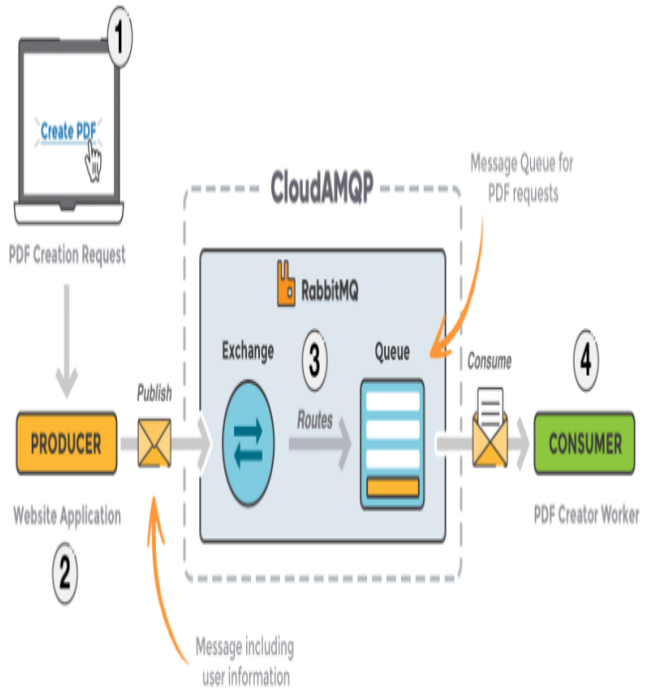


Fig.3 AMQP workflow

At the point when the application associates (AMQP association establishment) two stages happen which are disconnected to the designer. All the customer libraries will initially open. And afterward, they will take part in AMQP handshaking. Inside, in the server, RANCH will deal with all the TCP subtleties. The acceptors handels all the association demand that comes in.The peruser is the procedure that handles approaching messages from the attachment. The second that is begun, it'll will give a connection.start. It will give an association start crude back to the customer. The customer will send back connection.start_ok. Some portion of that connection.start_ok contains the accreditations - username and secret word of the application. At that point there's association tuning. Furthermore, from that point forward, the dealer will send a connection.open_ok. Rather than sending messages to lines legitimately, makers send messages to exchanges. Exchanges at that point send messages to lines whose coupling key matches the messages' directing key. The line, thus, sends messages to consumers who are bought in to it. This plan of exchanges and lines in AMQP makes the message passing straightforward. Figure 3 shows the work process of AMQP. A producer doesn't have to keep up any state about its messages' consumers. Correspondingly, a consumer doesn't have to keep up state about the producerss. The choices of which line the message ought to be directed to is chosen by the exchange rather than the application, which makes evolving message-steering and conveyance rationale simpler contrasted with an informing framework where the application settles on such choices.

AMQP has a few highlights that make it speaking to application designers. The most basic highlights incorporate a decision of exchange types, message ingenuity, and disseminated specialists. There are four sorts of exchanges characterized in AMQP, including direct exchange, topic exchange, fanout exchange, and header exchange Figure 4 gives the short knowledge into the design of RabbitMq just as the exchange types.

• Direct exchange sends messages to various lines/queues as indicated by the routing key. Messages are load adjusted between consumers rather than lines.

• Topic exchange sends messages to every lines/queues whose linking/binding key matches routing key. It is used to deploy publish/ subscribe model.

• Fanout exchange acts like a broadcaster and does not consider routing keys. The exchange just sends received messages to all queues which are bound to the exchange.

• Header exchange uses message headers to route message to queues instead of routing key.

AMQP additionally offers solid dependability ensures. For each exchange and line in the representative, messages can be persevered in circle by flipping the toughness banner..

As far as disseminated intermediaries, AMQP gives bunch, organization and scoop strategies to accomplish high accessibility and message specialists circulation. Hubs inside the bunch are reflections of one another, with the exception of lines which are nearby to every hub. Alliance and scoop models are utilized to permit messages over a WAN.

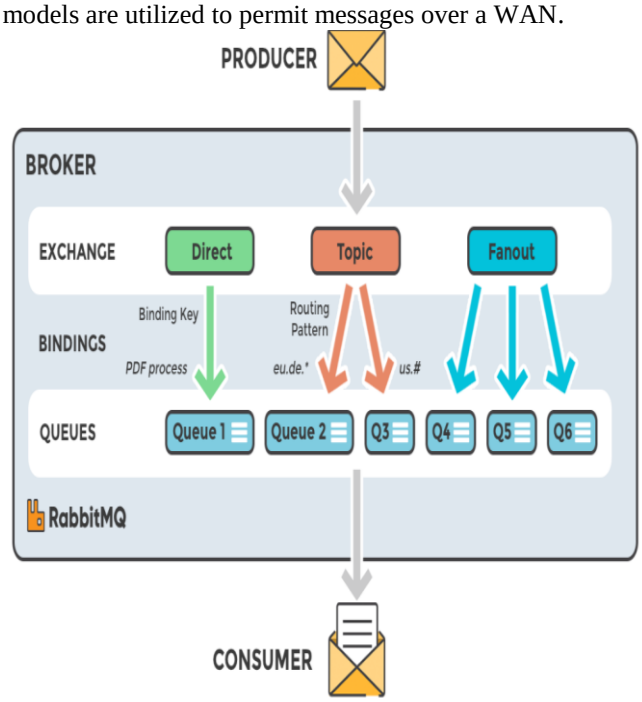


Fig 4. RabbitMQ Architecture

B. NATS

NATS informing empowers the trading of information that is sectioned into messages among PC applications and administrations. These messages are tended to by subjects and don't rely upon arrange area. This gives a reflection layer between the application or administration and the hidden physical system. Information is encoded and encircled as a message and sent by a distributer. The message is gotten, decoded, and prepared by at least one supporters. Nats adopts the strategy of being engaged, taking care of the issues of execution and accessibility while remaining amazingly lean. It discusses being "consistently on and accessible", and utilizing a "fire and overlook" informing design. It's effortlessness, center

and lightweight qualities make it a prime contender for the microservices environment.

NATS is written in Go language that is open-source cloud-based informing framework, lightweight and is kept up by Synadia Group. Endorsers effectively listening onto these subjects get the messages. NATS server known as gnastsd accommodates adaptability by cutting off memberships if there is a break in association with the server. Different highlights of NATS incorporate bunched method of servers and a consistently on dial tone for pub/sub framework.

NATS makes it simple for projects to impart across various cloud suppliers, on-premise frameworks dialects and situations. Customers associate with the NATS framework, for the most part by means of a solitary URL, and afterward buy in or distribute messages to subjects.

C. NATS Streaming

NATS Streaming is an information spilling administration for NATS server. The NATS Streaming server is a customer of NATS that can stack an inserted NATS Server at runtime, or interface with a current NATS server, by means of setup or order line choices. Directs are the subjects in NATS Streaming in which customers get information and makers send information to be placed in message logs. Figure 5 gives the brief insight into the architecture of NATS & NATS Streaming.

Features	**RabbitMQ**	**NATS/NATS Streaming**
Developed in the year	2007	2015
Language used for development	Erlang	Go
Models supported	pub/sub and Message queue	pub/sub, request/replay and Message queue
Message Delivery	At most once and At least once	At most once - NATS/NATS Streaming At least once - NATS Streaming
Message storage	In-memory/Disk	In-memory/Disk
Latency	Low-medium	High
Distributed Units	Queues	Channels
Throughput	Medium to high	High
Languages supported	30	12
Protocols supported	AMQP, MQTT and STOMP	Google Prtocol Buffer

Table I Feature Comparison Table

Nats Streaming gives extra highlights, for example, at any rate once conveyance of message, upgraded message convaention utilizing Google convention cushions, message diligence that is valuable for message replay and tough memberships. At-least-once message delivery, Flow control (rate matching/limiting), Message replay by subject and Durable subscribers form the core feature set of NATS Streaming.

Sadly, it doesn't bolster trump card coordinating. Solid memberships basically implies that in the event that a customer were to restart, at that point the server will begin conveyance with the soonest message that is unacknowledged by that endorser.

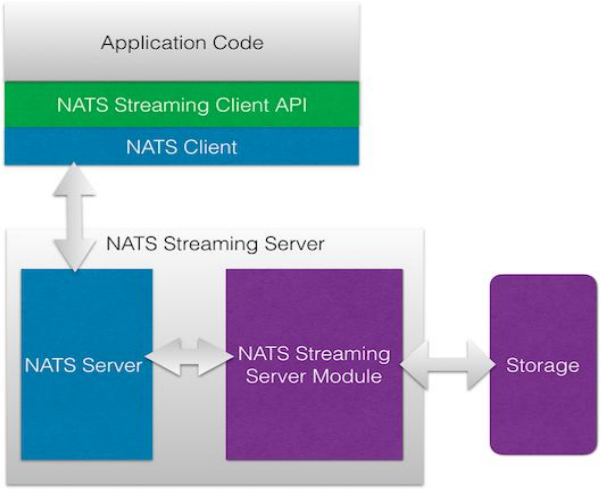


Fig.5. NATS & NATS Streaming Architecture

III. COMPONENTS OF SIMULATOR

The three advancements portrayed above are well known and give contending likenesses that it gets hard to pick the correct system. [10],[11] give an inside and out investigation of Quality of Service (QOS).

A. Throughput

Throughput is characterized as the quantity of messages per unit time that can be sent moved among makers and buyers. RabbitMQ sits tight for affirmations (ACKs) for each message and doesn't do clump preparing, along these lines diminishing throughput. But, RabbitMQ has the element to kill ACKs. The examinations completed in [13] bolster this end. NATS Streaming is quickly advancing with the most recent April 2019 discharge and is including broad highlights for elite cloud local applications. Further research should be completed to record the varieties in execution of RabbitMq and NATS Streaming.

B. Message delivery

Message assurance or conveyance frames the essence of nature of administration.'Precisely once' conveyance happens when message is gotten just one time. 'At any rate once' conveyance happens when message is sent in any event once yet can likewise get sent on numerous occasions. This is valuable for recuperation from disappointment. 'At most once' is where the message could possibly get conveyed. This gives a high throughput. This requires costly calculations. RabbitMQ gives all things considered once and at any rate once conveyance. NATS gives all things considered once and in any event once conveyance [13].

C. Latency

The deferral in process is named as dormancy/Latency. RabbitMQ is fit for giving low-idleness. NATS streaming gives a high greatness of idleness when contrasted with RabbitMQ [2]. This can mostly be ascribed to the way that NATS streaming API interfaces with NATS customer API to get in contact with the NATS server which initiates delay because of the circuitous way. The analyses completed in [10,15] infer that for at-most once and at any rate once conveyance in RabbitMQ the dormancy doesn't fluctuate much for medium burden.

D. Message Persistence

It is the capacity to hold the messages so they are accessible significantly after a representative restart. NATS Streaming gives message diligence either in memory or utilizing level documents. This is a configurable choice for NATS. RabbitMQ has diligence as an alternative and can be put away in-memory or on plate [12]. Regardless of whether a line is set as strong in RabbitMQ it doesn't guarantee message steadiness.

E. Message Ordering

For RabbitMQ, requesting is available inside a queue. For numerous queue supporters the requesting can be kept up by the utilization of predictable hash trade. NATS Streaming gives endorsers messages in the request they were distributed by a solitary distributer yet doesn't ensure request conveyance if there should be an occurrence of different distributers.

F. Scalability

It characterizes the flexibility of a framework to oblige a developing number of undertakings, for example, makers/poducers or purchasers/consumers or messages. RabbitMQ underpins bunching [18] where in numerous hubs go about as a solitary message representative. This is helpful to adjust the remaining burden and scale the framework to deal with huge number of messages. NATS Streaming gives the grouping mode, however this is just for HA. Versatility isn't all around upheld by NATS, and can be accomplished by utilizing the booking components gave by Go language.

G. Availability

It is the capacity of a framework to augment its uptime. The framework needs to give adaptation to internal failure to high accessibility (HA). NATS Streaming backings grouping and adaptation to non-critical failure mode for HA. In bunching mode, information is repeated onto diverse group hubs utilizing Raft Consensus calculation though in FT mode information is put away on shared capacity. In any case, to pick between the two modes, one ought to know that grouping has execution overhead on the grounds that an affirmation is sent when information is repeated on all hubs. In [16], It is seen that for a solitary line, execution is the most elevated and as the line is reflected the presentation is influenced, however this is a tradeoff for better adaptation to internal failure.

IV. BENCHMARK TEST

Benchmark Tests are tabulated for sending a prtobuf message of size 3KB with different combination of languages and number and publisher-subscribers.

Table 2 Nats Benchmark Test

(Time in Microseconds)	Single CPP Pub	Single Node Pub	Single Python Pub	Single Go Pub
Single CPP Sub	207	563	543	214
Two CPP Sub	242	627	583	249
Single Node Sub	787	693	672	532
Single Python Sub	613	717	613	497
Single Go Sub	324	584	610	203

Table 3 RabbitMq Benchmark Test

(Time in Microseconds)	Single CPP Sub	Single Node Pub	Single Python Pub	Single GO pub
Single CPP Sub	70	1035	375	410
Two CPP Sub	100	2908	1441	416
Single Node Sub	1042	3976	2688	1935
Single Python Sub	426	2986	1684	667
Single GO Sub	49	2734	402	455

V. DISCUSSION

The Various examinations of highlights of Message Broker are quickly spoken to in Table I and Benchmark Test for Latency in Table II and Table III. As found in the table there are not many similitudes between the Message Brokers which can additionally be concentrated to increase significant experiences. While planning cloud local arrangement, continuous preparing, enormous information, web of things or microservices arrangements facilitated on cloud the table should be alluded as it contains important experiences which when overlooked may make framework fizzle or result in a poor structure which ought to be stayed away from in programming advancement consistently. Kafka as named by makers is a spilling stage fit for taking care of enormous measure of information progressively or close to ongoing. RabbitMQ is a Message Broker that comes as nonexclusive as conceivable which is one of the purpose behind its extraordinary ubiquity and wide scope of utilizations. RabbitMQ has the ability to course the messages as required with the assistance of trades and lines which is truly necessary on account of Remote Farming or Remote Driving on a quite gigantic scope can likewise be utilized in web of vehicles where heading is significant. Tremendous Financial partnership depend on RabbitMQ for message conveyance ensures for what it's worth RabbitMQ's specialty to offer typses of assistance in territory where information is generally significant and should work with no misfortune in information. With the ACK exchange assurance of RabbitMQ, all messages are destined to be sent over to customers, this component is helpful for monetary exchanges. NATS Streaming is moderately new and is valuable for lightweight applications. It has a little help network that is developing quickly with the new discharges. The discharges comprise of an ever increasing number of highlights advancing it into a superior informing system that can rival Kafka.

VI. CONCLUSION AND FUTURE ENHANCEMENT

The element correlations are recorded succinctly in Table I. The table likewise records a portion of the comparative highlights between RabbitMq and NATS/NATS Streaming, that can be explored further by examination with other informing structures. The component correlations talked about, ought to be contemplated when structuring a venture arrangement or building up a disseminated IT framework. The fast look at the tables gives peruser with clearness on what Message Broker meets their requirements. Thus, perusers are to allude this paper when they need a fast presentation into the universe of Message Brokers and to take a gander at the assortment it brings to the table to the different areas of Assembling, E-Commerce, Finance, IOT and Big Data. The paper can likewise be perused by educated perusers when they are confounded regarding which Message Broker fits well for their utilization case or which Message Broker they should pick with extravagance of decision.

It isn't apparent at start however Message Brokers hold the way in to the Low Latency correspondence which is genuinely necessary in this developing universe of enormous information, it frames the center of large information gushing and preparing, constant examination, IOT and microservices area as an ever increasing number of information is produced and applications are worked to deal with and break down the information to pick up important bits of knowledge. Future work on the paper is further plunge further into the individual message worldview and track their progressions with each new discharge. Additionally, concentrating any new Message Broker that show up in the realm of Message Brokers to contrast those with the drivers of the business to give the perusers clearness on what Message Broker best suits their utilization case.

REFERENCES

1. R. Rocha, L. L. Ferreira, C. Maia, P. Souto and P. Varga, "Improving the performance of a Publish-Subscribe message broker," 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain, 2019, pp. 91-92.
2. R. Rocha, L. L. Ferreira, C. Maia, P. Souto and P. Varga, "Improving the performance of a Publish-Subscribe message broker," 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC), Valencia, Spain, 2019, pp. 91-92.
3. P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. “The Many Faces of Publish/Subscribe” ACM Computing Surveys (CSUR), vol. 35, pp. 114–131, 2003
4. Gracioli, Giovani & Dunne, Murray & Fischmeister, Sebastian “A Comparison of Data Streaming Frameworks for Anomaly Detection in Embedded Systems” 1st International Workshop on Security and Privacy for the Internet-of-Things (IoTSec),2018
5. S. Stoja, S. Vukmirovic and B. Jelacic, "Publisher/Subscriber Implementation in Cloud Environment," *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Compiegne, 2013, pp. 677-682.
6. Banavar, Guruduth & Deepak Chandra, Tushar & Strom, Robert & C. Sturman, Daniel. “A Case for Message Oriented Middleware.”,*DISC Proceedings*,Slovak Republic,1999, pp. 1-18.
7. B. R. Hiraman, C. V. M and C. Karve Abhijeet, "A Study of Apache Kafka in Big Data Stream Processing," *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, Pune, 2018, pp. 1-3.

8. Jay Kreps, Neha Narkhede, Jun Rao, "Kafka: A Distributed Messaging System for Log Processing",*at NetDB workshop*,2011
9. RabbitMQ,https:// 2019.
10. Apache Kafka website, April 2020. Available online: https://kafka.apache.org/.
11. Redis website, April 2020. Available online: https://redis.io/.
12. P. Bellavista, A. Corradi and A. Reale, "Quality of Service in Wide Scale Publish—Subscribe Systems," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1591-1616, 2014.
13. F. Araujo and L. Rodrigues, "On QoS-aware publish- subscribe," *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, Vienna, Austria, 2002, pp. 511-515.
14. P. Dobbelaere and K. S. Esmaili. "Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations" In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, pages 227–238. ACM, 2017.
15. NATS, https://nats.io/documentation/, 2019
16. RabbitMQ, https://2019
17. John, Vineet & Liu, Xia. A Survey of Distributed Message Broker Queues.2017, arXiv:1704.00411
18. T. Treat. Benchmarking NATS Streaming and Apache Kafka, https://dzone.com/articles/benchmarking-nats-streaming-and-apache-kafka, 2016
19. T. Treat. Benchmarking Message Queue Latency,201s6
20. M. Rostanski, K. Grochla and A. Seman, "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ," *2014 Federated Conference on Computer Science and Information Systems*, Warsaw, 2014, pp. 879-884
21. S. Skeirik, R. B. Bobba and J. Meseguer, "Formal Analysis of Fault-tolerant Group Key Management Using ZooKeeper," *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, Delft, 2013, pp. 636-641.
22. Martin Toshev, "Learning RabbitMQ" Birmingham, UK: Packt Publishing, Ltd, 2015
23. B. R. Hiraman, C. V. M and C. Karve Abhijeet, "A Study of Apache Kafka in Big Data Stream Processing," *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, Pune, 2018, pp. 1-3.
24. S. Patro, M. Potey and A. Golhani, "Comparative study of middleware solutions for control and monitoring systems," *2017 Second International Conference on Electrical, Computer andCommunication Technologies (ICECCT)*, Coimbatore, 2017, pp. 1- 10.
25. Kamburugamuve, Supun & Fox, Geoffrey. (2016). "Survey of Distributed Stream Processing." 10.13140/RG.2.1.3856.2968.
26. Kamburugamuve, Supun, Leif Christiansen and Geoffrey C. Fox. "A Framework for Real Time Processing of Sensor Data in the Cloud." *J. Sensors* 2015 (2015): 468047:1-468047:11.
27. Z. Wang *et al*., "Kafka and Its Using in High-throughput and Reliable Message Distribution," *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, Tianjin, 2015, pp. 117-120.
28. 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)*, Boston, MA, 2017, pp. 4803-4806.

AUTHORS PROFILE

Poojya J Bhat (pujyabhatt@gmail.com) pursuing MTech in Software Engineering in RV College of Engineering, Bengaluru. Working as an intern in Atonarp Micro-Systems pvt lmt, Bengaluru. Also have been part of RVCE placement cell as Student Placement Co-Ordinator Industrial Visit: 27th Nov 2018 to Green Data Center at EGL Business park ,IBM India Pvt Ltd., IBM coordinator: Makala V Reddy. Attended Webinar: TCS Cisco WebEx Events 29th November 2018 Hosted by Abhishek and Dhruv Sharma on Evolution of business 4.0 and Impact on software quality engineering.

Priya D (priyad@rvce.edu.in) is a professor of Information Science and Engineering Department at RV college of Engineering, Bengaluru. And their research work is

1. Faculty Advisor for RVCE "Coding Club". As a part of coding club, many consultancy projects were executed and hackathons were also organized by the club under the supervision of faculty advisors. The Club is working on many consultancy projects in the field of Machine Learning, IoT and Android app development.

2.Team member of IoT Application development under Cisco Center of Execellence.

3.Completed and awarded Certificate for the course on "FUNDAMENTALS OF ACCELERATED COMPUTING WITH CUDA C/C++" by NVIDIA.

4.Resource person for "Machine Learning & IoT" for VTU college students across Karnataka. About 80 participants were present. It was held on 22nd & 23rd July 2019.

5.Resource Person for "IoT Application development" for M.Tech students of Mechanical department, RVCE held on 08/03/19.

6.Resource Person for "IoT Application development using Nodemcu ESP8266" for BCA students of National college, 10th Jan 2019, Jayanagar.

7. Guiding students in various AI related UG and PG projects.