ISSN: 2582-8118 ***Volume 1, Issue 2; March 2021***

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYSIS IN SCIENCE AND ENGINEERING

Web: https://www.iarj.in/index.php/ijrase/index

7. Monitoring Microservices in the Distributed Environment

Pranali Prashant Ugale

MTECH Student, Department of Computer Engineering and Information Technology, College of Engineering Pune, Pune, India.

S. K. Gaikwad

Assistant Professor, Department of Computer Engineering and Information Technology, College of Engineering Pune, Pune, Maharashtra, India.

ABSTRACT

It is a hurdle to monitor the huge number of microservices which are might place in different regions physically. It is very difficult to find a point of failure in such a big environment. Again it's very complicated to find out what going inside in a specific actively running service. This research paper proposed architecture for monitoring such services with less computational and space overhead on the production site (on those machines which are actively serving customers using microservices). Proposed architecture aims to work efficiently, also provides the best performance in a real-time environment so that failure investigation will become easy to handle. This architecture is highly reliable and highly scalable without downtime can be safely used in the real-time production environment for monitoring microservices. The proposed architecture can handle a high workload with horizontal scalability.

KEYWORDS

Cloud, Kubernetes, docker, big data, kafka, monitoring, container, Open distro for elasticsearch, highly-available, Scalable.

Introduction

In computer science, earlier internet companies used to run large programs with many modules on a single system. Which increases the load on that system. Even when number of requests increase they need to add many servers with a whole replica of that application.

The number of big cloud service providers switch to another architecture because it's not possible to use monolithic architecture due to exponential increase in number of the user of internet and size of applications even grows.

But some of them like stack overflow still follow a monolithic one because of some of its advantages like the speed of execution, simplicity in management, very few network calls as compared to the new architecture model.

Nowadays it's impossible to run whole giant code on a single system and scale that as per need so we divide the whole module into many operational services which can be easily scaled up and scale down as per need. This service-oriented architecture is called microservice architecture. This kind of architecture is more economical. So here many machines are involved in the deployment part as per need. Single service can be run on many machines or a single machine will run few services.

This kind of new architecture comes with its pros and cons too. This will might be an economical solution but here is the problem handling too many machines and managing them is too difficult for service providers. Monitoring all of those machines will become a big hurdle as the application grows more and more and we scale them up. In such an environment handling errors or failures becomes very complex. Even finding them will become a critical part. Even if we design any solution to monitor such architecture we need to make sure that these things will not affect real servers which are used in production for serving customers.

Sometimes for designing perfect architecture we need to understand the whole picture in a hierarchy. Now we all aware of the term cloud. What exactly cloud is? It could be a collection of the number of data centres that are connected via a network. Datacentre have many racks, each of them may have a large no of servers. Each server will act as an individual virtual machine. Each virtual machine can have one or more containers which will allow us to run one or many small services. So our whole deployment part can have one or many such servers. Which are might be too away from each other physically so monitoring them collectively could be a big challenge.

Challenge: Monitoring

As discussed in the above part of the paper we need millions of containers to run a single big application. Logging will help us to understand what goes inside the container. When we encounter any issues those logs will help users to understand the real problem which leads to failure. If this monitoring is not planned well will cause a lot of issues.

Literature Review

Lei Chen, Jian Liu (2020) [1] have done a study to find a way for collecting logs in a distributed environment. This paper is mainly deal with only for those applications which use docker container. This paper provides a way to collect logs but overhead generated using this technique will be more. SAMAN B (2017) [2]. His research has been conducted to find a way to monitor microservices. This paper also analysis performance. Application is already deployed using a framework spring. They used Kieker to monitor that application. In the first section of his paper, he explains why there was a need of switching from monolithic architecture to microservice architecture. This method will be useful for only very few architectures. This solution will work in very few case studies.

Wang Haitao, Zhao Jing (National Institute of Standardization, Beijing) (2019) [3] proposed a really good method for collecting logs in a distributed architecture. This architecture uses Kubernetes to handle many containers but suggests having a log collecting agent on those machines which are in production. This will create too much overhead so this solution is not optimized to use in the real-time production environment. Apart from this, they discuss Kafka which will play important role in architecture. Ren and zang (2020) [4] introduced a new way of monitoring that is smart enough will generate an alarm if failure is detected or investigate. They use SpringBoot. This architecture is used under only microservices architecture. This method again does not discuss clearly how the proposed architecture will affect overall performance.

Zamfir, M. Carabas (2020)[5] research on which is the best possible way to manage collected logs. They have used elastic search for handling large no of data which is generated from many machines will show us how it is efficient to store in an elastic search. But the approach does not mention performance analysis and unable to prove efficiency. Heavy shippers will lead things more complex by increasing the load on production systems. Nicola Dragoni, Schahram Dustdar (2018) [6] proposed a real-world case study that will show us how performance parameters will change drastically after switching from traditional monolithic to microservice architecture. This paper discusses every possible aspect which is related to performance. How things changed during this shift is mentioned. This study will help us for analysis of performance in our architecture. Alecsandru Patra¸scu, Victor-Valeriu (2014) [7] propose a way for logging. This way is generic which deals with virtualization layers in the cloud environment. Performance parameter in analysis using EC2. Calculated performance parameter properly. Vlad-Andrei Zamfir, Mihai Carabas(2019)[8] propose a monitoring system that will work for many architectures. This architecture will be useful for many microservices and even can be used in multi-virtualization. This architecture will help in understanding performance parameters. This architecture uses MYSQL as centralizing database. This will might lead to lesser performance during querying data from a database.

Research Gaps

Ongoing research on monitoring microservices having some loopholes which makes them a wrong choice to follow in a production environment directly. After a detailed study and observation of loopholes present in existing systems are identified and stated below.

Most of the researches not covering the whole architecture as per need in the production environment. [1][5][6].

Few solutions are great but produce more overhead which is not expected in a real-time environment. [1][4][5]. Some research paper only deals with specific architecture and tools and resources which are not open source [2][4][8].

Most important parameter is to figure out the computational complexity of our solution and overhead on an overall system which is not properly estimated in many research. [1][3][4][5].

only detection of failure is not necessary when we are the service provider. So we need to make the sure failure of any node will not cause the whole system to collapse. Management of handling such a scenario is necessary for a service provider. [1][2][5].

Handling edge cases in every component of the proposed solution are necessary which is not mentioned which will lead to the collapse of whole architecture under unfavourable conditions [1][3]. None of the architecture guaranty high availability as well as scalability with performance.

Objectives of Research

- Proposed architecture finds optimize way to monitor microservices in a distributed environment with less overhead on those systems which are serving customers.
- Which reduces computational power for monitoring in the production environment.
- Which provides real-time results than existing architecture.
- Resulting solution only includes open-source resources for building.
- Proposed architecture can be used in any cloud environment like with public or private cloud.
- Proposed solution is highly available and scalable at any time.

Proposed Work:

In the case of monolithic architecture, application deployment is not at all a trivial event that demands proper attention because it will either work or fail. Which was like just set it and then ignore it because the management part was not difficult. Very less amount of things need to monitor.

A deployment part is a periodic event. Before the virtualization concept was coined users need to have a physical machine with the dedicated operating system on it to deploy an application. But as soon as things started evolving virtualization came up. And physical systems replace with the virtual machine. But still, infrastructure dependency was there.

That means if two different code demands two different operating systems then it was difficult to run those codes on the same machine. After containers came into existence even that dependency was not there. Containers are isolated themselves from everything in an environment. We can have many containers on the same machine with even different needs of assets.

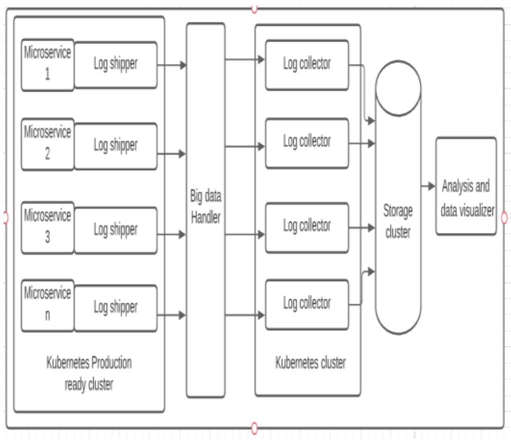

Fig. 1. Abstract view of proposed architecture

In case of the microservice environment along with container deployment managing those thousands or more no of containers is another important task. Which needs something that will control many containers. That will take care of the number of replicas we have and centralize the control manager. To solve this issue we are using Kubernetes.

As per the above simple architecture of the proposed work. It consists of many components currently we are keeping things simple to understand the full view of the whole architecture. Kubernetes will help us in managing containers. Kubernetes works great with Docker containers or any other. We can keep track of the number of replication which will guarantee availability.

In many of the researches, we have found that they directly use logstash on their worker nodes which is a completely wrong practice to be followed in the actual real-world especially when you are an application service provider. Consuming computational power of worker nodes and even run the heavy process on them will create a burden on the overall infrastructure. So we decide to use a very lightweight shipper on those worker nodes.

After conducting tests we have decided how to parse logs. There are two options available to parse them on worker node or use extra machines to do that. Again it depends on how much resources it will consume to have this burden on important worker nodes.

We use Kafka to Handel big data coming from Kubernetes. We have conducted few tests of the rate of generation of the log over the rate of ingestion. Then we process streams of data

After filtering, shipping will be done using another Kubernetes cluster that contains few instances of log collector logstash. We have all logs in our elastic search index. We can query and accordingly access those logs for monitoring. We can have a visual tool for real-time monitoring.

The important part about this research is it uses all open source technologies. One can deploy this architecture in any cloud environment.

The monitoring part of microservice is as important as deployment so needs to be designed by keeping everything in mind like how to provide high availability or having stable solution or it must not create any vulnerability or security issue for deployment. The proposed solution is the completely reliable and most important thing it is not at all creating burden on original architecture which will be used for providing service to end-user.

Implementation:

This section contains a detailed view of a proposed architecture as per production.

Kubernetes Setup:

Why Kubernetes is the best choice: Kubernetes will help us in managing containers. Kubernetes works great with Docker containers.

We can keep track of the number of replication which will guarantee availability. Kubernetes will assist us in handling failures too.

In Kubernetes, we have a central entity and even some worker nodes on which we can deploy our microservices.

Kubernetes is the place where containers are deployed. The most important thing about Kubernetes is we can deploy applications in a highly available production-ready cluster.

What exactly these buzz words mean? We can call any cluster production-ready if that cluster is serving customers in any way, that could be a single node cluster that is hosting a blog site webpage or it could be a larger cluster with millions of machines like Netflix or large enterprise application.

There is a trade-off in setting up a production-ready cluster because it completely depends on a business requirement.

Few parameters need to consider:

- To find an optimized way to monitor microservices in a distributed environment with less overhead on important systems.

- All components present in that cluster must be highly available
- All the components must be in a secure environment.
- They must be controlled from a single point if possible.

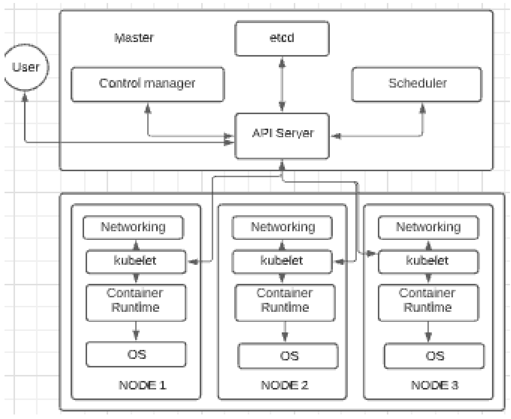

Fig. 2. Components of Kubernetes

What exactly a term highly available means and how to achieve that? The answer is if instances greater than equal to 1 if fail at any point in time, must not lead to failure of the whole cluster. Just like that machines greater than equal to 1 if fails at any point of time must not lead to failure of the whole cluster.

The solution for this is to make a cluster failure resilient. To achieve that according to the proposed architecture. There must be a single replica of everything present in a cluster at any point. People often get confused between the multi-master Kubernetes cluster and the highly available one. So, it might possible that a multi-master Kubernetes cluster fails due to the failure of other components.

There are many ways to set up a cluster but we need to find a balance between different things like security, high availability, and other things which are impacting cluster health. There are many parameters one of them is security. It must have a basic security-related thing like TLS secured communication everywhere.

Using certificated/identities can use a CSR API. You can disable anonymous authentication.

Enforce RBAC and node authorizers. Just make sure to handle dashboard and helm properly can use access control mechanism and roles. Set a restrictive pod Security Policy as a default.

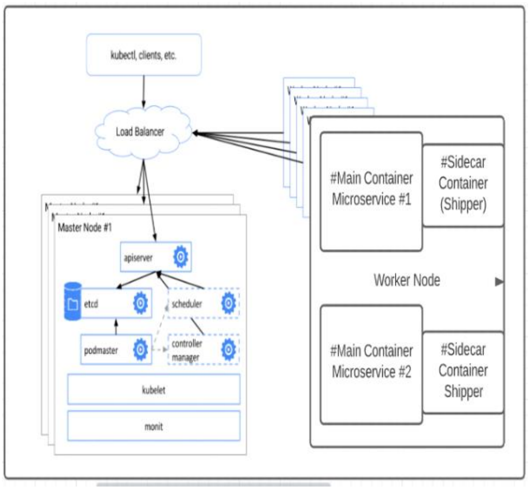

Fig. 3. Architecture: HA setup Kubernetes

One of the important parts here is an API server or a web server that talks to its database etcd where all metadata stores. API server can be scaled horizontally. The controller manager and scheduler only talk to the API server. The controller manager helps to maintain the actual state the same as a decided one. Scheduler helps to find pods and bind them. On the node side, we have a base operating system and a container runtime could be a docker or anything. The node side also contains kubelet which is a node agent talks to the container runtime about what containers need to execute.

Kubernetes is responsible to handle failure case of containers by self-healing property that is by restarting containers if it fails. Kubernetes can scale containers so make it a scalable component. After deployment of the main container, we are using a sidecar container that shares some volumes with the main container in next part will discuss the use of those volumes and how those volumes can be used in a few failure scenarios.

Dedicated Lightweight Log Shipper

Why a lightweight shipper? As we want to observe what going inside our deployed microservices. It's difficult to observe logs in large Kubernetes one by one. So here in this architecture, we have something called a Sidecar container which shares few volumes with the main container where microservices are deployed.

Now we need to make sure that we are not imposing more load on real nodes which are serving customers need. So, we want something lightweight enough that will take less space and will not demand many resources.

There are many solutions available one possible solution is to design one lightweight log shipper or we can pick the best available log shipper from open-source technologies. One can have Logstash which is the wrong choice. Logstash is a heavy shipper which demands more resources than a simple beat. Logstash is a more powerful technology that is not build to just pump logs so it's best to avoid that from the production node. Here we need to keep in mind few things that according to the official document from Kubernetes one pod must have a single container because the sidecar will always demand resources too, which is allocated to the main container. So here it's important to choose technologies wisely. In this case, our job is to just pump important logs from the main container nothing else no computation, no aggregation, no filtering.

Implementing a log shipper is the best way we are using in this architecture just to make sure the sidecar container is asking for few resources. Building a log shipper can be a tedious task we are adding two processes in log shipper one will take care of actual shipping that is moving data from the main container to Kafka and the other process will take of the failure condition. This second process will also ensure that even if the sidecar container fails it will start reading from the same place where it left by accessing the pointer file which is present in the volume of the main container.

We have sidecar containers with customized log shippers which are highly available as is its control by the Kubernetes manager. Failure scenario of sidecar will be handled by Kubernetes and pointer file, which is tracking offset part of watched files.

How to handle big data?

Now we have many sidecar container pumping logs. How to handle this much amount of BIG DATA. In this case, we can't use directly Logstash. Logstash could not handle this much data coming with high speed and can crash. The solution to this problem is to have something like a buffer that can either handle big data or it can process data that is coming in the black box with high frequency and big volume. There are many big data handling tools are available under open-source license few of them are just designed to handle stream processing.

We can have Apache spark, Fling, Kafka, and Akka streams. Apache Kafka can handle volume and velocity can work with logs efficiently [9], due to some amazing features like high availability, security, and performance. We have selected apache Kafka as a buffer in between log shipper and Logstash which will aggregate and filter data due to its best performance as compared to others[9].

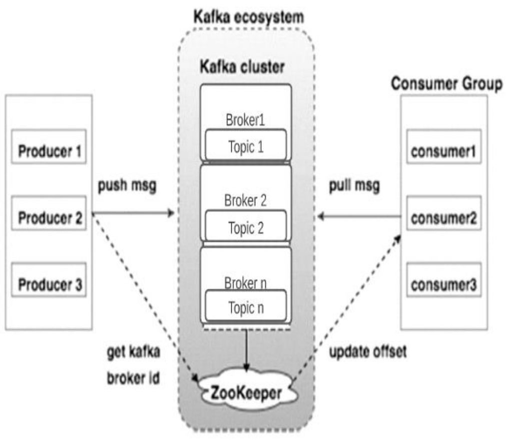

Fig. 4. Apache Kafka ecosystem

What exactly apache Kafka is and how it will be useful in this architecture?

Here problem is to store events. Events have some state and description. It has a timestamp like when it happened. The primary idea is event is an indication of the time when that thing took place. We use an ordered entity for that called as called a log. Now log has states what happens and when that happens. Apache is Kafka is built to manage logs using a historical term called a topic. The topic contains an ordered collection of logs stored durably. Topics can be small or could be too large there is nothing economic about their structures or they can remember data for a long time or they can remember data for just few seconds. Each thing in a topic represents a partial thing from an event.

Another use case of Kafka in microservice architecture each service is very small they talk to each other through Kafka topics sometimes. Each of these services can consume massage from a Kafka topic and do some computation and produce massage to another Kafka topic it will give that to another microservice and could be used for further analysis. This basic usage of Kafka can be used in this architecture for saving Logstash to handle BIG DATA. Kafka connect is another powerful entity to connect systems or a search cluster it could be a database. To integrate all of them a search cluster, SaaS product, systems we have an entity called Kafka connect. It's an ecosystem for open source even commercial products.

We are using another component with apache Kafka to achieve scalability and high availability. It is depending on the Zookeeper which will act as a managing entity. Apache zookeeper is used to coordinate functions and nodes served by apache Kafka.

Apache Kafka is built on the publishing and subscribe model. for processing logs we have designed filters as per log type which can process logs and send them back to respective partitions. Each partition is subscribed by one or more consumer groups which will then consumed by the log collector agent.

Shipping big data to storage is another hurdle we used a fully scalable part here we have many instances of logstash which will forward processes log to storage. An important part is any instance can fail at any time to handle failure we are using a full Kubernetes cluster with many instances of logstash as a part of the consumer group of a Kafka. In this subtopic, every component is failure resilient and scalable. Kafka can be controlled by apache zookeeper and logstash instances by Kubernetes.

Storage and Analysis

The next part of this architecture is the data warehouse that is where data should be stored. There are again multiple choices to select the database or a system that can be used to store data. We have done few tests with few available options like influxdb, arango db, Apache drill, and then selected Open distro for Elasticsearch. Which is used to store data in this architecture. There are many reasons behind selecting that database. The first reason is that it has extremely powerful search capabilities than other available choices.

There is one issue while using Elasticsearch that is currently pop up. Elasticsearch changed its license to SSPL from Apache 2.0 so it's no more a pure open-source tool to use after its version change. As per the problem statement of this paper, every technology must come under the pure open-source domain. The solution is Open distro for Elasticsearch built by amazon. Amazon is providing it under license Apache 2.0 so we have used them in our architecture.

Open distro for Elasticsearch also has its plugins which come under the same license that is apache 2.0. Open distro for Elasticsearch is going to provide its upcoming versions under the same license apache 2.0.

Open distro for Elasticsearch (ODFE) is an open-source technology that is distributed and highly scalable. It has powerful and efficient searching capabilities. It is not like a traditional database even people often compare it with relational databases.

It can handle large volumes of data efficiently which will make it more powerful. It has full-text search capabilities. Open distro for elastic search cannot only stores log but also it can analyze them too.

We can query just like typical databases. We can use SQL for it with open distro for Elasticsearch and for normal Elasticsearch OSS image we have DSL queries. Queries that take 10 seconds to execute with others take 10 milliseconds in executing them against OPDF.

It stores data in partitions that are spread across multiple nodes of a cluster. It provides data replication via partitions we call shards. Shards are available across many nodes in Elasticsearch.

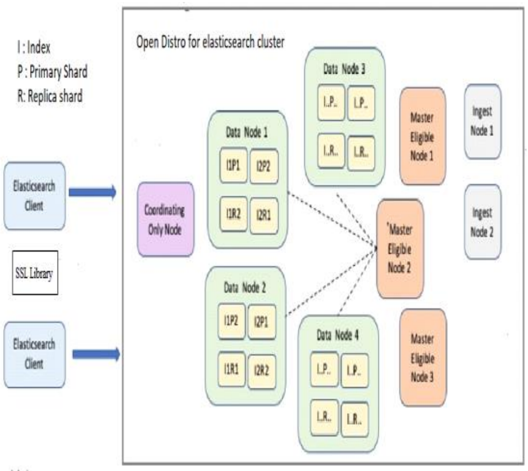

Fig. 5. ODFE highly available cluster setup

Setting up a cluster is not difficult but it's difficult to set up a highly available cluster that is production-ready. 'n' node elastic search cluster with minimum 3 master nodes and minimum 3 data nodes can become a highly available cluster. Why 3 because if any master node fails still it can continue. If even if any data node fails we still have data due to the replication of partitions. Here is an issue we must need to achieve a quorum to proceed or to elect a new master. So we can set configure a cluster to avoid **split brain** issues by imposing a condition that it should not work if a minimum of more than half of nodes are available.

As Elasticsearch and Kibana are products from the same community license changes applied on Kibana too. So again, the solution for this problem is to use a forked version from amazon. Kibana is a visualization tool. Along with aws plugins, it can become more than a visualization tool. Like security plugin from open distro will make Elasticsearch and Kibana both powerful and secure. It also provides access control and a role-based mechanism for index. It has great monitoring capabilities than the usual normal version of Elasticsearch. It provides alerting mechanism if we set a trigger.

We are using the machine learning capabilities of ODFE to create an anomaly detector. A machine-learning algorithm “Robust Random Cut Forest Based Anomaly Detection on Streams” [9] helps us to find the anomaly. Kibana is used for visualization and analysis of every activity inside the container. We can have the kibana server installed on any server that can connect to ODFE cluster securely with a node to node encryption policy.

Conclusion

This architecture will assure that this is the best possible way to monitor the microservices environment in the real-time production environment. Which will use less computational power than the existing architecture available. This stable solution is more reliable and highly scalable as per need. Resulted architecture provides low overhead on an important worker node in the Kubernetes cluster. This proposed architecture will work in the real-time production environment for providing effective cloud service. The proposed architecture is completely based on open source technologies that can handle a large amount of data and allow the user to securely monitor the application

References:

1. L.Chen, J. Liu, M. Xian and H. Wang, "Docker Container Log Collection and Analysis System Based on ELK," 2020 International Conference on Computer Information and Big Data Applications (CIBDA) 2020, pp. 317-320, doi: 10.1109/CIBDA50819.2020.00078
2. Barakat, Saman, Monitoring and Analysis of Microservices Performance.Journal of Computer Science and Control Systems. (2017) 10. 19-22.
3. Xinming Lai1, HaitaoWang, Jing Zhao, Fan Zhang, Chao Zhao and GangWu. IOP Conference Series: jorurnal of computer science Volume 688, Issue 3,2019
4. Y. Jiang, N. Zhang and Z. Ren, "Research on Intelligent Monitoring Scheme for Mi-croservice Application Systems," 2020 International Conference on Intelligent Trans-portation, Big Data Smart City (ICITBS), Vientiane, Laos, 2020, pp. 791-794, doi: 10.1109/ICITBS49701.2020.00173
5. V. Zamfir, M. Carabas, C. Carabas and N. Tapus, "Systems Monitoring and Big Data Analysis Using the Elasticsearch System," 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2019, pp. 188-193, doi: 10.1109/CSCS.2019.00039.
6. M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen and S. Dustdar, "Microservices: Migration of a Mission Critical System,"(2018) in IEEE Transactions on Services Computing, doi: 10.1109/TSC.2018.2889087.
7. A. Pătraşcu and V. Patriciu, "Logging framework for cloud computing forensic environments," 2014 10th International Conference on

Communications (COMM), Bucharest, 2014, pp. 1-4, doi: 10.1109/ICComm.2014.686666.

8. A. Noor et al., "A Framework for Monitoring Microservice-Oriented Cloud Applications in Heterogeneous Virtualization Environments," 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 2019, pp. 156-163, doi: 10.1109/CLOUD.2019.00035.
9. Sudipto Guha, Nina Mishra, Gourav Roy, Okke Schrijvers: Robust Random Cut Forest Based Anomaly Detection on Streams. ICML 2016: 2712-2721
10. Jay Kreps, Neha Narkhede, Jun Rao and Linkedin Corp “Kafka: a distributed messaging system for log processing.” NetDB’ 11 2015.