International Journal of Current Research and Review
DOI: http://dx.doi.org/10.31782/IJCRR.2021.13507

Scopus®

Original Research

IJCRR
Section: Healthcare
ISI Impact Factor (2019-20): 1.628
IC Value (2019): 90.81
SJIF (2020) = 7.893

Analysis of Kubernetes for Distributed Healthcare System Development using COVID-19 Healthcare App

Prassanna J, Mohanty S, Jinturkar PS, Pandey P

School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai Campus, Tamilnadu, India.

ABSTRACT

Introduction: Distributed computing is a field of computer science which deals with the study of distributed systems A system which has communication and coordination with each of its nodes and which interacts with each other to achieve a common target which is to effectively compute the computation. These capabilities are conducive to implementing a systematic and efficient COVID-19 tracking application which can be accessed and worked on by numerous entities.

Objective: To provide information about a client-server architecture which is a platform for managing and maintaining containerized workloads and services that forms a base for automation.

Methods: A Kubernetes cluster IS created with a calico pod network along with the main drivers of Kubelet, Kubeadm and Kubectl. Secure Shell (SSH) protocol is used for secured shell and data management and authentication between the client and server.

Results: We have performed and distributed our tasks in such a way to show the developers that multiple tasks can be performed at the same time using Kubernetes orchestration platform and used to parallelize multiple tasks. This increases the efficiency of the machine and the performance of the system becomes much faster.

Conclusion: A system which has communication and coordination with each of its nodes and which interacts with each other to achieve a common target which is to effectively compute the computation. One such application which helps in the distribution of tasks and helps do the computation is Kubernetes. It is based on a client-server architecture which is a platform for managing and maintaining containerized workloads and services that forms a base for automation.

Key Words: Kubernetes, Microservices, Docker, Kubeadm, Kubelet, Kubectl, COVID-19 Tracking

INTRODUCTION

Kubernetes (otherwise called k8s or "Kube") is an open-source compartment coordination stage that mechanizes a considerable lot of the manual procedures associated with conveying, overseeing, and scaling containerized applications. As such, you can bunch together gatherings of hosts running Linux compartments, and Kubernetes causes you to effectively and productively deal with those groups. Kubernetes bunches can traverse across on-premise, open, private, or half and half mists. Hence, Kubernetes is a perfect stage for facilitating cloud-local applications that require quick scaling, similar to ongoing information gushing through Apache Kafka. Containers and container stages give unquestionably more points of interest over customary virtualization. Segregation is finished on the kernel level without the necessity of any foreign programming software, so containers are undeniably progressively effective, quick, and lightweight. Docker is presently the premier famous containerized stage. Docker provides us with a platform for parallelizable and distributed computing of these containerized applications but still, a problem of coordination and schedule of those containers appeared. Kubernetes is a system for managing different containerized applications across multiple hosts, providing basic mechanisms and functionalities for deployment, maintenance, and scaling of various and different types of applications.

Virtualization permits better use of assets in a physical server and permits better adaptability because an application can be included or refreshed effectively, lessens equipment expenses. Each Virtual Machine is a complete computerized

Corresponding Author:
Prassanna J, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai Campus, Tamilnadu, India.
Email: prassanna.j@vit.ac.in

ISSN: 2231-2196 (Print) **ISSN:** 0975-5241 (Online)
Received: 14.10.2020 **Revised:** 23.11.2020 **Accepted:** 26.12.2020 **Published:** 03.03.2021

machine on which we can run and execute our normal programs as we do on our computers, running every one of the segments, including its working framework over the virtualized equipment. Now, in the containerized environment, it acts like Virtual Machine's (VM), yet they have loosened up confinement resources to make use of the OS amid the application. Accordingly, containers are viewed as lightweight. Similar to a Virtual Machine, a container possesses its filesystem, CPU, memory and process space. As they are decoupled from the hidden foundation, they are versatile across mists and OS conveyances.

Containers have become a necessity in today's world due to:

1. Administration disclosure and burden adjusting:
   Kubernetes can uncover a container utilizing the Domain Name System (DNS) name or utilizing their address. In the event where traffic in a container is too high, Kubernetes can adjust and accommodate the system traffic till the arrangement is steady.
2. Capacity Organization:
   Kubernetes permits you to naturally mount a capacity arrangement of our decision, for example, nearby stockpiles, open cloud suppliers.
3. Computerized rollouts and rollbacks:
   We as users can completely control the execution of the containerized applications which we are going to deploy and we can change the execution state into an ideal state or general state depending on the need and requirement. We can mechanize Kubernetes and expel those containers which are not required and keep those which are necessary.

The last few years have seen a rise in popularity of cloud services due to their scalable, tractable and solitary nature. At the same time, contemporary devices now can run containerized microservices without demanding high power. In place of these developments, a plethora of research has been conducted into hosting services in containers and deploying it in clusters. A Kubernetes compatible cluster container can enhance the workload placement across the cloud and clusters.[1]

The aim of developing this architecture is to deploy flexible modules which are deployed and scaled independently to constitute a cloud-based application. Kubernetes provides a set of foundations upon which we can access the methods to deploy, maintain, scale and repair containers which host the microservice. Kubernetes performs abstraction of the underlying convoluted system of the microservices and their maintenance. Upon investigation, it was established that despite being explicitly stated in the official documentation, Kubernetes is more appropriate for public clouds rather than private ones.[2]

Kubernetes aim to gauge the applicability of the Kubernetes mechanism as a way to create and host a Kubernetes cluster. The purpose of this cluster was to enable easy deployment and scalability of bioinformatics workflows. The user could allocate resources and provide certain front end functionalities (like logging) seamlessly. The network is aimed to grow in a horizontal manner such that it will create slaves if the resources available are not sufficient.

The goal of Kubernetes networking is to turn containers and Pods into bona fide "virtual hosts" that can communicate with each other across nodes while combining the benefits of VMs with a microservices architecture and containerization. Consistently, get to the administrators has progressed from security to enabling specialist of business improvement by engaging relationship to finish their occupations securely, gainfully and cost-effectively, at long last transforming into a key device for danger and asset the board in affiliations.[3] Table 1 delineates the differences between the traditional distributed systems and Kubernetes in terms of key factors such as workload distribution, storage management, scalability, system requirement and security.

Table 1: Differences between the traditional distributed systems and Kubernetes

S. No	ATTRIBUTE	TRADITIONAL PERIOD	KUBERNETES PERIOD
1	WORKLOAD DISTRIBUTION	Earlier, there was no system for distributing the work services among multiple machines and there became a necessity for scheduling, distributing and managing resources.	Using SSH protocol, master, as well as worker nodes, are connected and work gets distributed across the nodes and resource allocation becomes much easier.
2	STORAGE MANAGEMENT	Earlier, to store or mounting a storage system in our machine required a large amount of money, as well as data, was handled by data centres putting the security of the data in question.	Kubernetes allows us to mount storage according to the users choice and necessity. The user can store the data in public, private or hybrid mode depending upon the authenticity of the data.

Table 1: ***(Continued)***

S. No	ATTRIBUTE	TRADITIONAL PERIOD	KUBERNETES PERIOD
3	SCALABILITY	There was a problem with scaling the resources as per the user needs and requirements. Scaling the resources required a lot of labour and investment of money. Also, if one node failed, the entire cluster deployment would fail to work.	Kubernetes offers containers for distribution of work and we can scale the resources to as much as we want. Scaling of the resources will allow proper distribution and resource allocation of work. Even if one container fails, Kubernetes automatically restarts the container so that cluster is not damaged
4	SYSTEM REQUIREMENT	There has been a lot of debate that the resources require a large number of resources and consume a lot of CPU and memory (RAM). So, every time we would need to work on extremely high configuration computers so that the work takes place efficiently.	Although, there is a basic system requirement and RAM issues here as well, the requirement of CPU and RAM for a particular container is handled by the user. Kubernetes automatically fits and manages the resources in such a way that the nodes are utilized to their fullest.
5	SECURITY	There has always been a problem of storing passwords, tokens and other security-related information in a secure environment where no tampering of important information can take place. Data security has been an emerging issue which needs to be addressed.	Kubernetes allows us a storage system to store and manage sensitive information such as tokens and passwords. The data remains secure here and no tampering of data can take place. The user can update and deploy their secure data whenever necessary.

Kubernetes consists of some major components without which the cluster would not be instantiated. Kube adm is an essential component and is used for the maintaining and power housing the entire cluster, We make use of Kubectl to control the Kubernetes clusters on the numerous centres. Kubelet is, in essence, an architecture which operates on each node and allows the flow of information between each of the nodes and the master depending upon the requirement.[4]

Secure Socket Shell is a UNIX based security protocol used for accessing any of the machines which are near or distant. To make the communication between the different pods of the master and the slave possible through a secure shell, we use SSH protocol. So, to access the pods, it can be done through service which is a stable point for a set or a collection of different pods.[5]

The protean nature of Pod which is featured in the following model structure(Kubernetes organization demonstrates) was evaluated and it was found out that pods can be scaled up and down depending upon the usage and the user requirement. If the user wants more scalability, Kubernetes provides the option of scaling it to as many node resources as much required. Generally, a container enabled virtualization model is adopted for cloud environments. This model makes use of Dockers along with Kubernetes for a multi-host Docker container system. An important aspect of such an environment is that Kubernetes must monitor the resource requirements and/or consumption level of the running applications. It must dynamically adjust the resources allocated to the containers to provide optimum performance. At the moment Kubernetes provides a rudimentary dynamic resource allocation process which is limited to only the CPU consumption. This paper aims to develop a generic platform to efficaciously deploy a dynamic resource- provisioning based on Kubernetes.[6]

The majority of solutions for improving efficiency in the data centre industry is hardware-based. A software-based solution must be found instead of a hardware one. This thesis considers a comprehensive outlook of the data centre and takes into consideration the virtual and physical infrastructure. The aim is to surrogating a comprehensive scheduling system in place of the ubiquitous present scheduler and has the regard of the software and hardware models.[7]

Kubernetes provides the opportunity to deploy and maintain containerized modules within a Platform-as-a-Service (PaaS) cloud service. This is achieved through Kubernetes' functionality to deploy numerous 'pods' across multiple machines which allow scalability of everchanging workload. Multiple dockers are supported by each pod and are responsible for hosting services (like I/O and file system) associated with the pod. Kubernetes provides a much-needed approach which enables scalability and resilience of the deployed application thus augmenting the elasticity of the cloud platform.[8]

Bhutada et al.[9] demonstrated the sections that ought to be considered for the improvement of game-plans subject to microservices, showcasing how the arrangement resting on microservices is portrayed, perceiving the fragments associated with the running in the coursed enlisting circumstances, elucidating the security show germane and associating the parts that merge the checks and strategies related with the structure subject to Microservices. This research formulates a novel method to automize the process of forming and monitoring Kubernetes Federation using TOSCA (Topology and Orchestration Specification for Cloud Applications). This is accomplished by embracing the organization cost that automizes the displaying and launch of cloud applications. It additionally includes the organization of cloud uses of the TOSCA and checks the scaling parts of the framework.

Monolithic Architecture [10]is exceptionally mind-boggling to comprehend by the recently joined colleague. It has over-burdened IDE and web-based containers too, Scaling and Constant improvement are troublesome. To beat every one of these constraints, Microservices appeared. Our proposed strategy is additionally founded on Microservices.

Kubernetes or K8s is universally accepted as the standardized tool for deploying containerized applications. The fact that it is applicable for public, private and hybrid cloud only serves to further this assertion. This paper serves to address the problem of selection of nods for pod deployment. This is established through the use of combining the ant colony algorithm and particle swarm optimization algorithm. The node with the minimal objective function is selected to deploy the pod.[11]

Cloud computing uses the internet and a centralized remote architecture to provide services which handle data which are later provided on-demand. Virtualization is the keystone to deploying a cloud service, these VMs contain and deploy the cloud workload. Efficient allocation of VM leads to an improved resource allocation.[12,13]

Kubernetes can be defined as a set of building blocks which work in conjunction to deploy, maintain and scale containerized microservices. It provides an abstraction of the complexities and nuances of microservice architecture. This paper evaluates Availability Management Framework (AMF) service which enables high availability and in certain cases increases the service outage.[14]

Tasking scheduling is an important aspect which needs to be resolved to balance the workload on a cloud-based server. A Threshold Based Multi-Objective Memetic Optimized Round Robin Scheduling technique[15] is proposed in this paper to resolve this problem. This technique employs a burst detector to gauge the workload and perform any necessary measures.

The standard cloud service providers such as AWS require high amounts of money and will not function without a high-speed internet connection. To counter these constraints, we aim to make an offline Kubernetes network will provide a more scalable service and can be expanded as per user needs and is completely open-source. The proposed system will contain a master node which will be connected to multiple client nodes which provide unique functionalities. Users will be able to avail those functionalities through the master node. The system will support four applications (Node JS, Angular JS, Mongo DB) which will be distributed over two client systems deployed on distinct ports on the clients.

Proposed System Architecture

The proposed framework is as per the following: Proposed System Architecture. As per monolithic architecture, the use of Microservices has become extremely necessary for all. The system architecture of our project is also based on Microservices. Technologies included in the proposed system are:

- Microservices
- Docker
- JWT Token
- Kubernetes

Figure 1 shows the architecture diagram of Kubernetes. As seen in Figure1 Kubernetes follows a client-server architecture. There exists a master installed on one machine and the node on separate virtual machines.

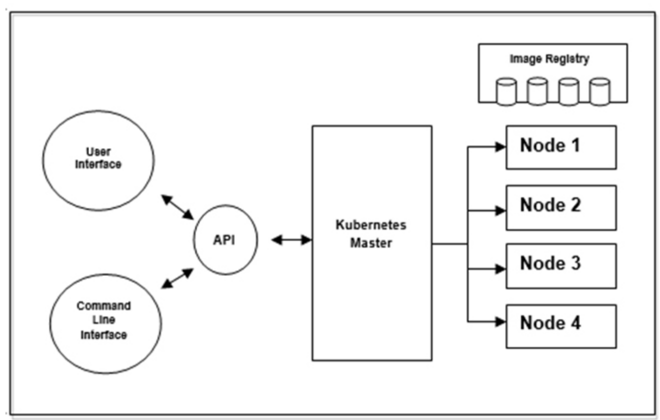

Figure 1: Architectural diagram of Kubernetes.

The use of different Virtual Machines and containerized pods has made the job of the Operating System easier and multitasking among the master machine and the worker nodes becomes easier. Figure 2 shows us a Pod Diagram depicting the workload of the master has been divided into worker nodes and depending upon the pod requirement for each of the worker nodes, we have divided the work among containers to make it a containerized deployment.

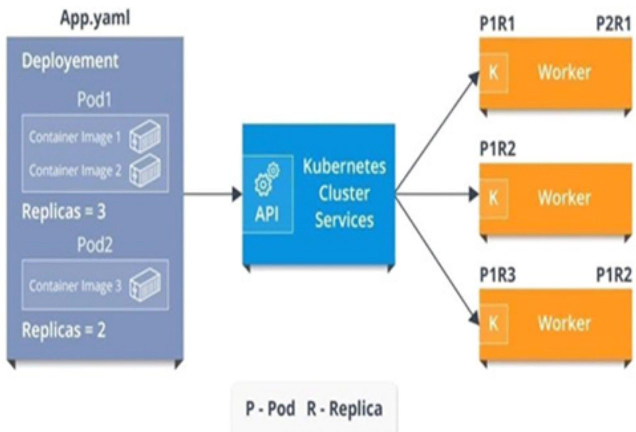

Figure 2: POD Diagram.

MATERIALS AND METHODS

Step 1: The first thing to do would be to form the Kubernetes Cluster in the master system. Execute the following code to do so:

#kubeadminit--Episerver-advertise-address=192.168.56.101 --pod-network-cidr=192.168.0.0/16

Here our master node has initialized successfully into our cluster.

Step 2: As fore mentioned, execute the following code as a non- root user from the output:

$ mkdir-p$HOME/.kube$sudocp-i/etc/kubernetes/admin.conf $HOME/.kube/config $ sudochown

$(id -u):$(id -g) $HOME/.kube/config

Step 3: To envision whether or not all the pods are working or not we have to use the command,

$ kubectl get pods -o wide --all-namespaces

Step 4: The subsequent order of business is to setup the dashboard. Run the following snippet of code to setup the dashboard:

$kubectlcreate-f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes- "https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml"dashboard.yaml

After the master has been deployed into the cluster, it provides a command for the other scalable nodes to join the cluster. To control the cluster in a way such that any user can operate onto the cluster and scale up and down the resources depending upon the requirement, we need a web UI, which is the Kubernetes dashboard. To initialize the dashboard, we need to give the token key which we obtain when the nodes join the clusteralong with the master.

Step 5: During the course of this step, we will construct the service account for the Dashboard and obtain the credentials.

Note: It is imperative that these code snippets are run in anovel terminal otherwise the kubectl command may stop.

List of commands to execute:

The purpose of this code is to manufacture a service account forthe dashboardinsidethe Defaultnamespace.

$ kubectl create serviceaccount dashboard -n default.

This will ensure that the service dashboard is created and all the cluster rules are added to it

$ kubectl create clusterrolebinding dashboard-admin -n default --clusterrole=cluster-admin—serviceaccount=default:dashboard.

After running this code snippet we obtain the token necessary for dashboard login.

$ kubectl get secret $(kubectl get serviceaccount dashboard -o jsonpath="{.secrets[0].}"

After applying the kubeadm command and providing the master machine IP, the nodes join the cluster.

RESULTS AND DISCUSSION

Figure 3 shows the deployments which have been performed of some sample applications as well as the CPU and memory usage of our machine on which the applications are performed. The service status ensures that the deployments are up and running and are ready for execution. The dashboard status as well as the deployment status of the master machine as well as the node is clearly shown. Then our Nginx docker and AngularJS application can be deployed through webpage, where we provide the IP of the client and the port number specified at the dashboard.

Figure 3: Deployment workload status.

Figure 4 shows us to see our NodeJS application, COVID-19 Patient Monitoring System deployment throughwebpage, where we provide the IP of the client and the port number specified at the dashboard. We can add, delete and display patient information. MongoDB is used as backend database.

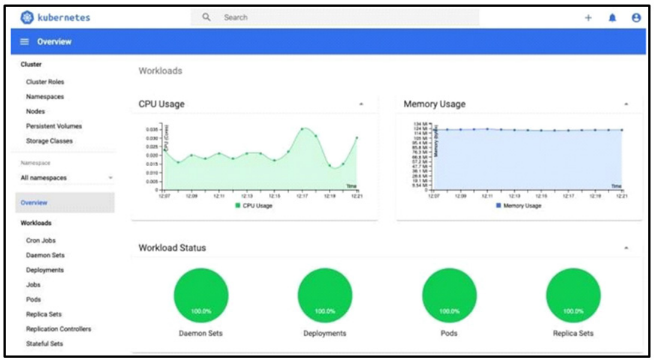

Figure 4: Deployment of COVID-19 patient monitoring system.

Figure 5 shows us that Kubernetes on our machine is platform dependent and as the number of CPU's increase, our requests increase.

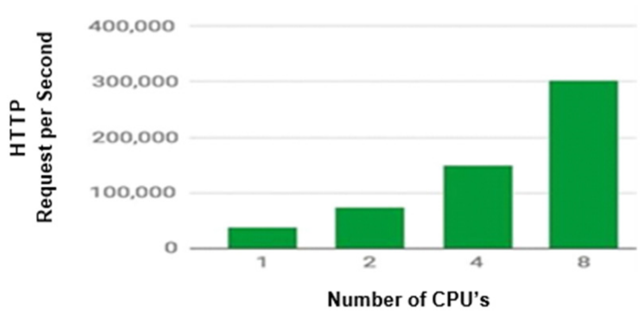

Figure 5: Number of CPU vs requests.

Fig. 6 shows us the CPU utilization per pod. As the number of CPU's of the machine increases, the effective pod utilization increases. Therefore, a high specification system is recommended for performing the cluster operations if we want to perform on local Virtual Machines.

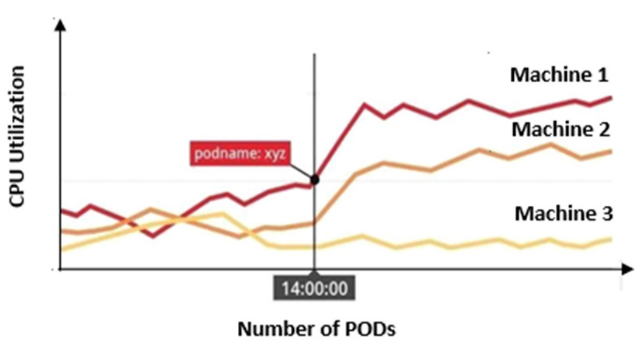

Figure 6: CPU utilization per POD.

CONCLUSION AND FUTURE WORK

We have achieved a high performance service provider using Kubernetes cluster and hosted our own cloud service using Calico pod network. In a brief span of time, Kubernetes has developed and created into an financial powerhouse. Because it offers shifted benefits, numerous companies see to create items and administrations to meet an ever-increasing require. Kubernetes has a capacity to work on both public and private cloud and has made it one of the top choice apparatuses for the businesses that work with hybrid clouds. If this proceeds, we would be able to indeed see more companies contributing in Kubernetes administration framework. Also, this has been performed on our local machine. We would like to host the cluster in cloud in future, so that the cluster becomes platform independent and independent of the system state.

ACKNOWLEDGEMENT

Authors acknowledge the immense help received from the scholars whose articles are cited and included in references of this manuscript. The authors are also grateful to authors / editors / publishers of all those articles, journals and books from where the literature for this article has been reviewed and discussed.

Conflict of Interest:

The Author(s) declare(s) that there is no conflict of interest.

Source of Funding:

Nil.

Individual Author's Contribution:

All authors have contributed equally.

REFERENCES

1. Goethals T, De Turck F, Volckaert B. FLEDGE: Kubernetes compatible container orchestration on low-resource edge devices. In: Internet of vehicles : technologies and services toward smart cities. 6th International Conference, IOV 2019, Proceedings. Springer; 2020. p. 174–189.
2. D. Jaramillo, D. V. Nguyen, R. Smart. Leveraging microservices architecture by using Docker technology. SoutheastCon 2016, Norfolk, VA, 2016, p. 1-5.
3. "Microservices" martinfowler.com. [Online]. Available: https://martinfowler.com/articles/microservices.html. [Accessed: 01-Oct 2020].
4. Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, et al. Microservices: Yesterday, Today, and Tomorrow. In: Mazzara M., Meyer B. (eds) Present and Ulterior Software Engineering. Springer, Cham. 2017.
5. Newman S. Building Microservices: Designing Fine- Grained Systems. O'Reilly Media, Sebastopol. 2015.

6. Amaral M, Polo J, Carrera D, Mohomed I, Unuvar M, Steinder M. Performance Evaluation of Microservices Architectures Using Containers. In: IEEE 14th International Symposium on Network Computing and Applications, Cambridge, MA, 2015, pp. 27-34.
7. Townend P, Clement S, Burdett D, Yang R, Shaw J, Slater B, et al. Invited Paper: Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes. In: IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco East Bay, CA, USA, 2019, pp. 156.
8. Docker - Build, Ship, and Run Any App, Anywhere. Available: https://www.docker.com/. [Accessed: 01 Oct 2020].
9. Bhutada S, Jyothi KK. Enhancing Security to the MicroService (MS) Architecture by Implementing Authentication and Authorization (AA) Service using Docker and Kubernetes. In: International Journal of Innovative Technology and Exploring Engineering (IJITE), pp. ISSN: 2278-3075, Volume-8, Issue-6S3, April 2019
10. Kubernetes. Available: https://kubernetes.io/. [Accessed: 24 Oct. 2020].
11. Zhang WG, Ma XL, Zhang JZ. Research on Kubernetes' Resource Scheduling Scheme. In Proceedings of the 8th International Conference on Communication and Network Security (ICCNS 2018). Association for Computing Machinery, New York, NY, USA, 144–148.
12. Mevada A, Patel H, Patel N. Enhanced Energy Efficient Virtual Machine Placement Policy for Load Balancing in Cloud Environment. Int J Curr Res Rev 2017;09(06):50-53.
13. Prassanna J, Venkataraman N. Adaptive regressive holt–winters workload prediction and firefly optimized lottery scheduling for load balancing in cloud. Wireless Netw 2019:10:1-19.
14. Vayghan LA, Saied MA, Toeroe M, Khendek F. Microservice Based Architecture: Towards High-Availability for Stateful Applications with Kubernetes. In: IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), Sofia, Bulgaria, 2019, pp. 176-185.
15. Prassanna J, Neelanarayanan V. Threshold Based Multi-Objective Memetic Optimized Round Robin Scheduling for Resource Efficient Load Balancing in Cloud. Mobile Netw Applic 2019;24(4):1214-1225.