2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus) | 978-1-6654-0476-1/20/$31.00 ©2021 IEEE | DOI: 10.1109/ElConRus51938.2021.9396553

Internet of Things Services Orchestration Framework Based on Kubernetes and Edge Computing

Daniil Ermolenko
The Bonch-Bruevich State University of Telecommunication
St. Petersburg, Russian
daniil-ermolenko@mail.ru

Claudia Kilicheva
The Bonch-Bruevich State University of Telecommunication
St. Petersburg, Russian
Bambosik@yandex.ru

Ammar Muthanna
The Bonch-Bruevich State University of Telecommunication
St. Petersburg, Russian
ammarexpress@gmail.com

Abdukodir Khakimov
Peoples' Friendship University of Russia (RUDN University)
Moscow, Russian
khakimov-aa@rudn.ru

***Abstract*— Presented work is fanalysis of how the microservices paradigm can be used to design and implement distributed edge services for Internet of Things (IoT) applications. Basically, IoT is a platform where integrated services are associated with the common network, thus all devices are able to gather and exchange data among each other. Typically, monolithic user mobility research services are developed for the unified ETSI MEC system reference architecture centers. ETSI MEC considers microservices as a tool for breaking monolithic applications into a set of loosely coupled distributed components. It is expected that this architecture will facilitate the dynamic adaptation during the application execution. However, increased modularity can also increase the burden on orchestration and system management. In MEC, user hardware is connected through gateways to microservices running on the edge host.**

There are three levels in each of the edge systems: 1) microservices perform a logical operation with components for motion track analysis, 2) movement foresight and 3) outcome visualization. The distributed service is realized with Docker containers and calculated on actual world adjustment with low capacity edge servers and real user mobility information. The results demonstrate the fact that the edge perspective of low latency may be encountered in this sort of implementation. The integration of a software creation technology with a standardized edge system supplies respectable basis for subsequent development. The paper considers the application of the boundary computing architecture and Kubernetes as an orchestration and management of network applications.

Keywords*— *Edge computing; Kubernetes; IoT; Clustering; Orchestration

I. INTRODUCTION

Today, communication networks have become a mandatory part of our lives. Previously, networks were created to connect people through machines, but now they are designed to connect machines and similar devices. Implementation of the Internet of things has a significant impact on every aspect of human activity, turning objects of everyday life into communication devices. It mainly happens because of the IoT ability to provide convenient, efficient, and feasible solutions for various applications such as healthcare, transportation, security, and finance. According to the research of Department Statista, the number of connected devices will reach more than 50 billion by 2030.

The above statements make it impossible for the current network infrastructure to maintain a huge increase in traffic and provide a service with the best QoS score. In developing future network systems, it is necessary to adapt existing network architectures to increasing needs, as well as design and develop new management capabilities to help meet the stringent requirements of future use cases.

It is the concept of edge computing that contributes to the development of a flexible network architecture of a new generation of networks. Edge computing is a new computational paradigm in network construction that brings the resources of a server, as a data center on a smaller scale, closer to the end devices. This approach helps unlock the full potential of high-performance access networks for ultra-low latency and high transfer rates, and increases resilience to problems in the underlying networks and data centers.

Today, Multi-access Edge Computing (MEC) is a standardized approach of the European telecommunications standards Institute (ETSI) for accessing peripheral computing at the network level. MEC, running at the core network and access levels, is the ideal solution for most use cases. However, there are still some issues that need to be addressed: the first is related to a vulnerability in network access, and the second is due to a high load on access networks and MEC servers. This is a major obstacle in large-scale IoT use cases, where multiple sensors can produce large amounts of data, or when critical system functionality is experiencing network access problems. Implementation of the concept is currently possible through the use of network function virtualization (NFV) and software-configurable networks (SDN) technologies.

This article is about exploring the possibility of bringing some boundary functions to the local level as virtualized and dynamically deployable components that use local hardware. Also, there is a development of a model multi-level network architecture using the concept of boundary computing, network function virtualization, network interaction management using the structure of software-configurable networks, and application orchestration. For the research, a prototype of a local edge network based on virtualized microservices was

978-1-6654-0476-1/21/$31.00 ©2021 IEEE

implemented. This model is presented as Docker containers and deployed using the Kubernetes orchestration system on the cluster's working nodes.

II. Related works

The paper [1] is dedicated to the implementation of IoT edge architecture, giving an accurate pattern about it. The other part of the work interpreters background and the newest form of this field. Futher the chosen use case was portrayed. In the practice, the authors carried out a test sample built on Docker containers orchestrated by Docker Swarm, using Alpine basis way and multi-phase-builds. Through the use of Alpine basis way joint with multi-stage-builds, it made it permissible to decrease the basis way amount from a few hundreds of MBs to a few tens of MBs retaining the runtime expense at a viable level. Consequently, the maintaince unfolding and initiation period per nanoservice were lowered from minutes to less than 60 seconds and operation initiation moment was compressed into the most of the instances from tens of seconds to just about a few seconds. This article effectively introduced the attainability work of deployment of virtualized nanoservices at the localized stage of IoT systems.

A report [2] is all about researching the investigation of launching various Intrusion Detection and Prevention System (IDPS) equipment satisfiability. A structure conforming the supplying safety services for IoT mechanism, centralized by a fog host, was put forward. This method involved a user to receive a preferable safety services that is capable of ensuring security and privacy aspects regardless of the service which were initially signed to. Besides, the proactive essence of the service supplying fog-based playground enhances the scaling of IoT annexes. The proposed composition is confirmed with emulating malevolent flow stream and analysing the productivity of the edge host. Although, the features and abilities of the orchestrator rely on the essence of the facility kind.

III. Problem statement

Microservice architecture is a newly formed concept in the area of software models that involve splitting a monolithic application into separate processes that implement a single function. This approach has the following advantages: 1) reduced complexity through the use of small applications; 2) declarativeness at deploying and orchestrating components; 3) flexibility in the design of various structures and tools; 4) predictable scalability; 5) improving the service's fault tolerance.

The microservice architecture is important in implementing edge computing networks in view of efficient resource management and the ability to scale a single component to increase performance and fault tolerance.

Scalability can be enhanced by various methods of introducing light and functional objects. Therefore, microservices are universal for designing, implementing, and deploying distributed IoT services. The problem of orchestration of modern computing systems is the heterogeneity of modern services, the complexity of orchestration and management of functional units, and the establishment of interconnection between them [3]. Modern MEC standards also require early troubleshooting of failures and their consequences for high availability of edge computing applications. However, these problems can be solved by using comprehensive solutions for deploying and developing platforms for deploying edge computing networks with a set of tools to ensure the continuous development cycle and integration of services, as well as management and orchestration of computing and network resources. Currently, microservices are implemented using containerization technology, which involves virtualization at the operating system level.

Actually, Docker has become a key solution in the development of containerization technology. It standardized the packaging and delivery of applications, and also took the microservices paradigm and established the best practice in creating docker images where there is only one workflow that performs one function of the application.

Docker provides security, manual management of computing resources, and supports overlay type networks to enable container networking between servers. The next step in the development of containers was the standardization of runtime (Container Runtime Interface) and networking (Container Networking Interface) interfaces that made it possible to separate the runtime plane from control and provided compatibility between different enterprise solutions. The paper considers a model network of edge computing based on Docker container and Kubernetes orchestrator.

IV. Proposed system

Today, in the telecommunications sector, one can observe the gradual introduction of new technologies in the organization of network and computing infrastructure, as well as new service design patterns. The new concept of NET-2030 networks, developed by the International Telecommunication Union (ITU), is still in development, but it is already becoming clear that the future in the telecommunications industry is aimed at blurring the line between cellular communications and the fixed network, developing general principles for building services, providing high availability of applications, both geographically and technically, as well as providing even higher speeds and quality of service indicators in comparison with the existing infrastructure. Software-defined Networking and Network Function Virtualization should be considered as technologies for network infrastructure technologies.

IoT devices are becoming more important in everyday life and their number is growing every year. To sustain this phenomenon requires new approaches to the design of network computing infrastructures and the management of network services. In the last few years, Application of the Internet of Things were integrated like a bunch of tiny, autonomous microservices. Their structure, however, is relatively fresh concept in software world. Also, microservices paradigm lies in a field of traditional service-oriented architecture (SOA) extension. Every microservises may represent with the ultralight container that can might be applied by a variety of user. For example, taking into account the smart city version, assets are spread inside the network, making sure, microservices are situated near the terminal device, which the IoT annex requests. Microservices, to a greater extent,

determine the new structure of network construction. Factors such as low latency, bandwidth, energy efficiency and cost must be considered when designing future MEC frameworks.

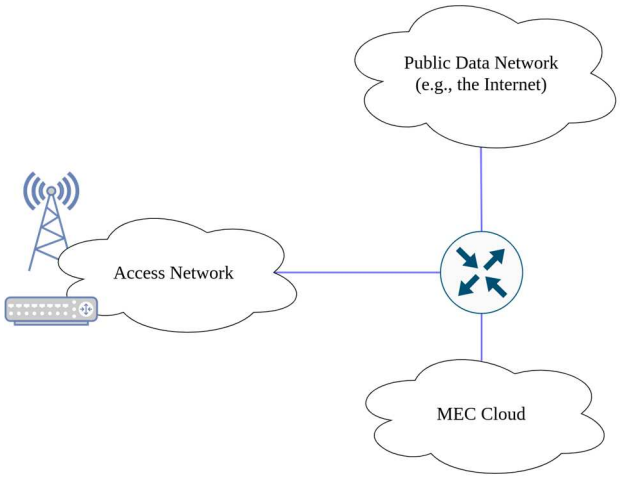

Fig 1. Concept of MEC and Internet.

It was the above technologies that became key in the development of edge computing, microservices determined the approach to service deployment, the synergy of software-defined networks and virtualization of the network function laid the basic principles in the development and operation of edge services. ETSI (European Telecommunications Standards Institute) became the regulatory body for standardization and specification supply. A working group of engineers from well-known telecommunications companies has so far defined a Multiple Access Edge Computing (MEC) technology. MEC allows to achieve closeness of the service to the user, low round-trip latency, minimize the use of public network bandwidth, and most importantly, provide devices with computing power for processing information and storage for its storage [4]. MEC assumes use with different types of access networks (both mobile and fixed), which clearly identifies MEC as a key technology in the construction of NET-2030 networks. Figure 1 shows a simple example of attaching an edge cloud to a public network.

To manage and orchestrate IoT applications, a microservices orchestrator must be used to ensure the stability and scalability of the operating system. Today, when using a microservice architecture, containerization technology is actively used, which makes it possible to provide greater efficiency in the use of computing resources, predictable scalability and convenience in developing microservice applications within the framework of virtualization of network functions. To overcome the complexity of orchestration and management of services with this approach to building edge computing systems, it is proposed to use Kubernetes as a basic element, and use OCI-compatible container management tools as a virtualization platform. In this paper, we focus on two main aspects of multi-access edge computing (MEC) technology: deployment (provision of work) and operation (provision of scalability and adaptation of the system).

A. Proposed framework of edge cloud computing for IoT services

Figure 2 shows the general architecture of the proposed edge cloud framework for IoT applications. This framework combines SDN and NFV MANO architectures to provide a stable infrastructure for deploying an application, enabling security services, and providing operational capabilities for maintenance, orchestration, and application scalability. This infrastructure of network and computing resources allows for new approaches to deploying applications with the necessary computing resources.

An edge cloud consists of a gateway for the access network, a firewall that provides initial traffic filtering to match connected services, and a Kubernetes cluster that provides the server and compute infrastructure of the cloud. Also, due to the flexibility of the Kubernetes architecture, the system provides entry points for data storage, including the use of distributed file systems.

A Kubernetes cluster mainly consists of a control plane and a work plane. The control plane can be represented as a single master node, or master control components combined into a cluster to achieve increased fault tolerance of the entire system. Control plane includes such components as: etcd, API server, Controller-manager, Scheduler.

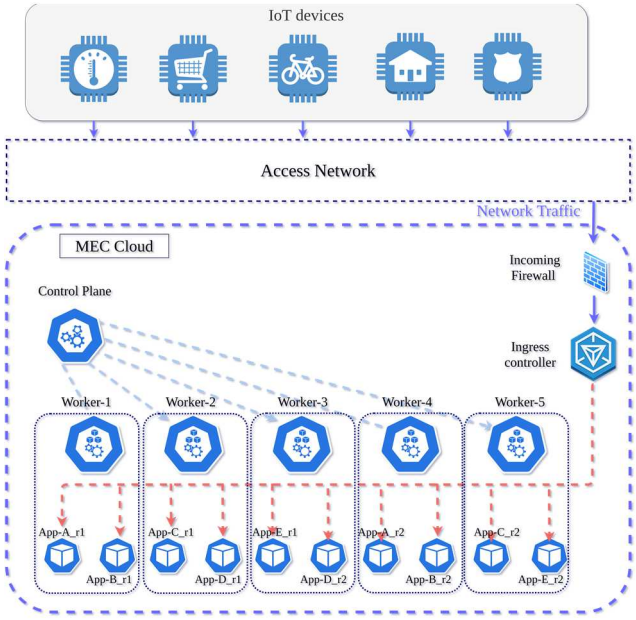

Fig 2. Proposed framework of implementation MEC cloud.

Etcd is a key-value cluster storage containing all information about the cluster, all settings, object manifests, secrets. The API server is the central part of the control plane. This component processes requests from other components of the cluster, working over the REST protocol using connection encryption mechanisms. The controller-manager consists of several controllers of the management processes in the cluster. Let's describe the three main controllers. The Node controller is responsible for notifying about the state of the node and reacting when it is unavailable. The Replication controller maintains the correct number of pods for each ReplicaSet on

the system. The endpoint controller anchors the Endpoint object that connects the kube service and the pod. The Scheduler is responsible for assigning pods to a node and allocating resources, it takes into account QoS, Affinity / anti-affinity, requested resources.

The application instances themselves are deployed on worker nodes, and the key components of the system are kubelet and kube-proxy, which run on all machines in the cluster, including the managers. Kubelet provides interaction and control with the virtualization infrastructure, and kube-proxy implements the service entity in the cluster, we will talk about this abstraction below.

B. Entities and interconnection nodel in framework, model flow traffic

To understand the organization of the network in kubernetes, it is necessary to consider the objects associated with ensuring the passage of network traffic to the application.

Service

Services allow you to associate an application or an instance of it with a specific IP address and DNS name. They are required to publish the part of the application responsible for receiving / transmitting traffic outside the cluster. Service types:

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName
- Headless

Ingress

Ingress is used to receive requests from the Internet, describes the rules for accessing the application via the http protocol. The abstraction specifies the host name, for example my.host.foo, which corresponds to the server name, and to which port of the service to send traffic. When receiving an http request for hostname, the cluster sees that there is such an ingress with the requested hostname, according to the description, a decision is made that traffic should be sent to the appropriate service with the correct port.

Kubernetes itself does not provide Ingress functionality; this requires a special module called Ingress controller.

The Ingress Controller is based on a proxy server that maps services and pod endpoints to ensure traffic flows directly. The proxy server configuration is compiled from the ingress description. At the moment, ingress controllers can distribute udp and tcp traffic, that is, they work at the 4th level, acting as a universal traffic arbiter.

kube-proxy is a cluster component that looks at the API Server, stands on all nodes, manages network rules on the nodes, and actually implements Service. It is implemented based on iptables or ipvs. Network rules define how traffic flows from service to pod. The chain of rules for the ClusterIP type is shown in figure 3

Analyzing the chain of rules, we can conclude that kube-proxy implements a round robin algorithm for passing traffic in the operation of the service, which ensures a uniform load between instances bound to the service object.

```
KUBE-SERVICES
  -d 1.1.1.1/32
  -p tcp
  -m comment --comment “mynamespace/myservice:http cluster IP”
  -m tcp --dport 80
-j KUBE-SVC-UT8A43GJFBEDGO3V

-A KUBE-SVC-UT8A43GJFBEDGO3V
  -m comment --comment “mynamespace/myservice:http”
  -m statistic
    --mode random --probability 0.500000000
  -j KUBE-SEP-MMYWB6DZJI483RW

-A KUBE-SVC-UT8A43GJFBEDGO3V
  -m comment --comment “mynamespace/myservice:http”
-j KUBE-SEP-J33WXLDEJI483RW

-A KUBE-SEP-MMYWB6DZJI483RW
  -p tcp
  -m comment --comment “mynamespace/myservice:http”
  -m tcp
-j DNAT
  --to-destination 10.102.3.49:80

-A KUBE-SEP-J33WXLDEJI483RW
-p tcp
  -m comment --comment “mynamespace/myservice:http”
  -m tcp
-j DNAT
  --to-destination 10.102.0.93:80
```

Fig 3. Iptables rules list.

CNI – Container Network Interface

Container Network Interface is a Cloud Native Computing Foundation project consisting of specifications and libraries for writing plugins that configure network interfaces in Linux containers. Kubernetes implements a CNI interface to create a network overlay that routes traffic between pods of different nodes [5]. Some plugins for Kubernetes:

- Flannel provides a simple and easy way to set up a Layer 3 network infrastructure. Has support for jumbo frames, is capable of working in VXLAN mode.
- Calico is a Layer 3 virtual network that has the ability to enforce policies. Supports IP-IP tunnel and VXLAN.
- Cilium - Uses the Linux kernel BPF and XDP mechanisms for network traffic control.
- ovn-kubernetes - Provides an overlay network built on Open vSwitch (OVS) and Open Virtual Networking (OVN) with support for both Linux and Windows.

To build a network, I suggest using the CNI flannel plugin in host-gateway mode. This mode is the least resource-intensive, but requires network integration into one L2-segment. Flannel provides routing between nodes and cluster pods through the use of routing rules and control of virtual network interfaces. the flanneld container working in the kube-system space provides ip addresses and prevents network conflicts.

Figure 4 shows a network model that describes traffic flow in the edge cloud.

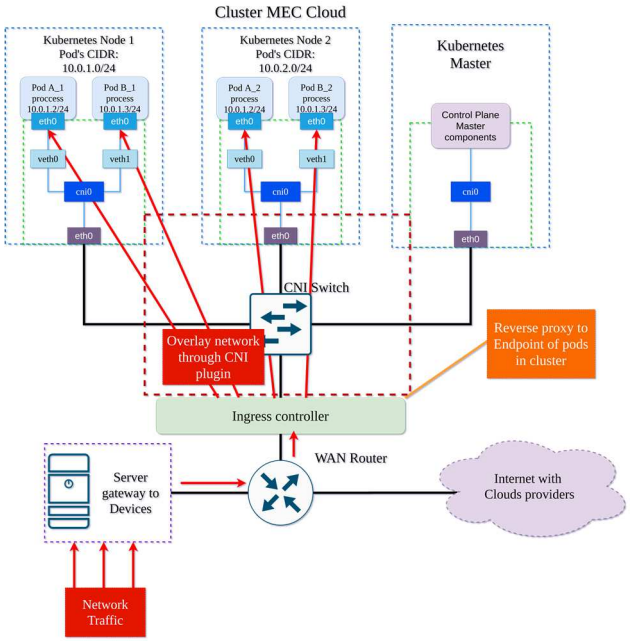

Fig 4. Traffic flow network.

V. IMPLEMENTATION AND EXPERIMENTS RESULTS

The purpose of the experimental part of this article is to study the operation of the IoT device protocol in the simulation implementation of the boundary local cloud. Http was chosen as the protocol of IoT devices. We chose it because there are many test methods for it, it is the basic protocol for building IoT applications, and the protocol methods are used in the REST design pattern, which is focused on working with resources. To study the performance of the model network, we will conduct two comparative experiments, where we will measure the time to receive a response from the server.

For testing, a script was written in python. The script describes sending get requests with creating a session to the server, for this, the aiohttp and asyncio libraries were used to ensure multithreading in the formation of requests, thereby increasing the performance of requests per second compared to the classic urllib library. Requests are generated in the range from 10 to 50 requests per second in increments of 10. Also, while the traffic generator is running, the iperf3 tool is launched on both ends of the connection (client-server) to load the communication channel with tcp connections. The client was launched with the following parameters: 100 parallel requests and 100 kB read / write buffer length. These parameters reflect the actual load from IoT devices. The duration of each step is 10 minutes, which minimizes the error when averaging the response time.

A. *Experiment 1, apache server deployed in one instance*

Client is Notebook, which creates a load of tcp connections via iperf3 tool and generates http get requests to the server.

Logically the client and the server are in the single LAN segment, which is implemented using a bridge connection of the main interface of the laptop. Figure 5 shows a diagram of a local area for the experiment.

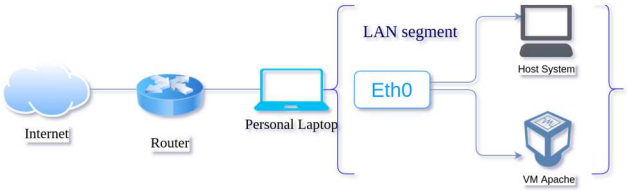

Fig 5. First experiment, apache server deployed in one instance.

B. *Experiment 2, apache server deployed in a Kubernetes cluster.*

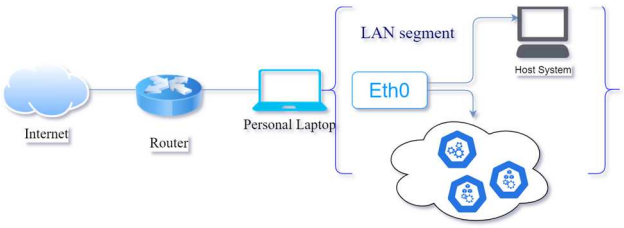

Fig 6. Experiment 2.

The implementation of the local network section is similar to the first experiment. The cluster represents three nodes that act as virtualbox. The control node and the two workers are linked by cni flannel plugin, which runs in 'host-gw' mode. An ingress controller from the kubernetes community is used to access the web server. The traffic is evenly distributed between the two Apache replicas, which are distributed to each working node in the cluster. Figure 6 illustrates the LAN diagram for the second experiment.

Laptop configuration. Asus x507ma laptop, Intel Pentium N5000 processor, 8 GB DDR4-2400 single rank single channel RAM, 480 GB SSD on MLC memory cells. Operating system Fedora Linux 32, kernel version 5.8.4, VirtualBox 6.1.

C. *Results*

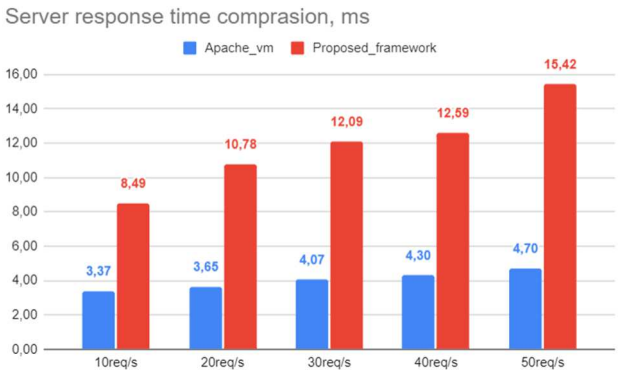

Fig 7. Results of testing first deployment scenario.

As mentioned above, the purpose of the experiment was to determine the response time from the server. At each stage of testing, packets were captured by the tcpdump utility. Further, wireshark determined the time since request for each request, and then calculated the average value of the obtained parameter throughout the entire testing phase. Figure 7 provides a

comparison chart of the average server response time for the two experiments.

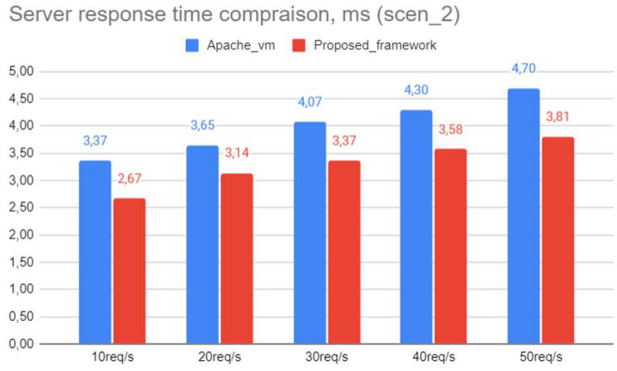

Fig 8. The second option for testing the proposed framework

Analysis of the results of the two experiments indicates that the apache application deployed in the Kubernetes cluster is inferior to the classic apache version on one server. But as you can see, the results do not increase linearly, this is due to the fact that iperf3 requires a large amount of CPU time to create parallel requests, so the testing does not reflect the result that we would like to see. To reduce the load on the system and get more performance from the experimental client and server, you can use minikube. Minikube provides a more compact but less flexible solution for deploying a local Kubernetes cluster out of the box.

Let's carry out the second scenario of testing the proposed framework, the cluster configuration remains the same as in the first scenario. For this, the ingress controller was enabled, and the apache web server was launched in two replicas. The cluster virtual machine is deployed using the kvm hypervisor, 4 vCPUs, 3 GB RAM. This configuration requires less vcpus and value of memory.

Analyzing the results obtained, we can conclude that load balancing gives a positive result to reduce response time (figure 8), and using orchestration and Kubernetes management, you can easily scale and change the parameters of cluster applications. For example, the value of the number of replicas is set in the manifest file of the deployment object, which is the control plane and takes as a declarative description for the application to work. Also, in the configuration of the http server, the configmap mechanism was used to transfer the configuration inside the container with the application.

VI. CONCLUSION

In this article, we have proposed a framework for deploying an edge cloud of IoT applications based on the Kubernetes orchestrator and microservice manager. During operation and testing, the framework showed satisfactory results in terms of performance. When deploying an edge cloud, light scaling can be applied depending on the tasks and tasks, additional object abstractions for configuration can be applied, and a data warehouse can be added. Edge computing has tremendous implications for large scale global IoT deployments. The distributed application approach to a single compute cluster and a reliable data link brings the concept of an all-encompassing unified data center, bringing the edge cloud together as well as the edge cloud and the side within the public network. Future work will be to provide a framework that can optimize and efficiently distribute heterogeneous traffic for a variety of applications that require high server response.

REFERENCES

[1] Islam J., Harjula E., Kumar T., Karhula P., Ylianttila M.: Docker Enabled Virtualized Nanoservices for Local IoT Edge Networks. CSCN 2019, DOI: 10.1109/CSCN.2019.8931321

[2] Imrith V. N., Ranaweera P., Jagurnath R. A., Liyanage M.: Dynamic Orchestration of Security Services at Fog Nodes for 5G IoT. ICC 2020, DOI: 10.1109/ICC40277.2020.9149019

[3] Ateya A.A., Muthanna A., Koucheryavy A. 5G framework based on multi-level edge computing with D2D enabled communication. 20th International Conference on Advanced Communication Technology (ICACT) conference proceedings. 2018. C. 507-512.

[4] Mobile Edge Computing (MEC); Framework and Reference Architecture./ 26.03.2020 gs_MEC003v010101p

[5] Marko Luksa Kubernetes in Action: Manning Publications Co. 2018. 594 c. ISBN: 9781617293726

[6] Recommendation ITU-R M.2083: IMT Vision, “Framework and overall objectives of the future development of IMT for 2020 and beyond,” Sep. 2015.

[7] ITU-R, “Minimum requirements related to technical performance for IMT-2020 radio interface(s),” Nov. 2017.

[8] 3GPP TS 28.554, “Management and orchestration; 5G end to end Key Performance Indicators (KPI)”, Ver. 2.0.0, release 15, Sep 2018.

[9] Wang, G., Zhao, Y., Huang, J., Wang, W.: The controller placement problem in software defined networking: a survey. IEEE Network, 31(5), pp.21-27, (2017).

[10] Chen, M., Ding, K., Hao, J., Hu, C., Xie, G., Xing, C., Chen, B.: LCMSC: A lightweight collaborative mechanism for SDN controllers. Computer Networks, 121, pp.65-75, (2017).

[11] Zhang, L., Wang, Y., Li, W., Qiu, X., Zhong, Q.: A survivability-based backup approach for controllers in multi-controller SDN against failures. In 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 100-105, IEEE, (Sep. 2017).

[12] Song, P., Liu, Y., Liu, T., Qian, D.: Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers. Science China Information Sciences, 60(3), p.032202, (2017).

[13] Li, L., Xu, Q.: Load balancing researches in SDN: A survey. In 7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC), pp. 403-408, IEEE, (July 2017).

[14] Koponen, T., Casado, M., Gude, N., Stribling, J., Nicira, Inc.: Distributed control platform for large-scale production networks. U.S. Patent 8,830,823, (2014).

[15] Tootoonchian, A., Ganjali, Y.: HyperFlow: A distributed control plane for OpenFlow. In Proceedings of the 2010 internet network management conference on Research on enterprise networking, pp. 3-3, (April 2010).

[16] Yazici, V., Sunay, M.O., Ercan, A.O.: Controlling a software-defined network via distributed controllers.arXiv preprint arXiv:1401.7651, (2014).

[17] Krishnamurthy, A., Chandrabose, S.P., Gember-Jacobson, A.: Pratyaastha: an efficient elastic distributed SDN control plane. In Proceedings of the third workshop on Hot topics in software defined networking, pp. 133-138, ACM, (Aug. 2014).

[18] Koerner, M., Kao, O.: Multiple service load-balancing with OpenFlow. In IEEE 13th International Conference on High Performance Switching and Routing (HPSR), pp. 210-214, IEEE, (June 2012).

[19] Yao, H., Qiu, C., Zhao, C., Shi, L.: A multicontroller load balancing approach in software-defined wireless networks. International Journal of Distributed Sensor Networks, 11(10), pp. 454159, (2015).