Live migration of containerized microservices between remote Kubernetes Clusters

Kiranpreet Kaur, Fabrice Guillemin, Francoise Sailhan

▶ **To cite this version:**

Kiranpreet Kaur, Fabrice Guillemin, Francoise Sailhan. Live migration of containerized microservices between remote Kubernetes Clusters. INFOCOM WKSHPS 2023: IEEE International Conference on Computer Communications - Workshop, May 2023, Hoboken, NJ, United States. pp.114-119, 10.1109/INFOCOMWKSHPS57453.2023.10225858 . hal-03466765v2

HAL Id: hal-03466765
https://hal.science/hal-03466765v2

Submitted on 3 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Live migration of containerized microservices between remote Kubernetes Clusters

Kiranpreet Kaur[1,2], Fabrice Guillemin[1], Francoise Sailhan[3]
[1] Orange Labs, 2 Avenue Pierre Marzin, Lannion, France.
email: {firstName.lastName}@orange.com
[2] Cedric Laboratory, CNAM Paris, 292 rue St Martin, Paris, France.
email: {firstName.lastName}@cnam.fr
[3] Labstic Laboratory, IMT-Altantique, Brest, France.
email: {firstName.lastName}@imt-atlantique.fr

***Abstract*—The recent adoption of cloud native technologies by telecommunication industry is accompanied by the incoming development of Network Functions that are containerized and packaged as light-weighted microservices. In order to efficiently orchestrate Cloud-Native Network Functions (CNFs), thorough migration strategies should be supported to place and migrate the CNFs. In this paper, we present a testbed illustrating the migration of pods between remote K8S cluster. As a use case, we consider the migration of a network function belonging to an opensource 5G core network (namely, Magma).**

***Index Terms*—Multi-cluster migration; Live migration; Containers**

I. Introduction

The microservices paradigm has recently gained popularity in the telecommunications industry with the emergence of Network Function Virtualization (NFV). Complex monolithic network functions, which were so far hosted on dedicated hardware, are now preferably decomposed into ready-to-use and easy-to-manage microservices. This evolution also benefits from the emergence of container-based technologies, which have been popularised with the wide spread of cloud infrastructures, notably those based on Kubernetes (K8S) clusters. The main advantage of CNF lies in the greater flexibility offered, for example, to create, update, migrate or delete CNF.

However, the need for an appropriate mechanism to migrate a chain of CNFs (i.e., decomposed into microservices) across distributed cloud infrastructures (and thus across multiple Kubernetes clusters) requires a critical attention that is the main focus of this work. We introduce a prototype that supports migration of some CNFs of a private 5G core network instantiated into a three-level cloud infrastructure (Figure 1). Our prototype is based on Kubernetes (K8S) [1], which is a popular open source container orchestration and management engine for automating the deployment, scaling and managing of containerized applications. In practice, core network functions are instantiated on a data center. If the data center hosting all the CNFs gets overloaded, some CNFs are migrated. For this purpose, the prototype relies on existing K8S technologies and does not modify the K8S orchestration platform: a new pod[1] is created at the destination node to host the migrated components. Then, the requests received at the old pod are transferred to the new one when this latter gets fully active. To handle such a container migration, further development is needed, especially with distributed K8S clusters. From a practical point of view, this is quite challenging as many community groups are running in the race of achieving an efficient multi-cluster migration mechanism. Thus, the proposed solution contributes as a migration technique that can be followed to support the migration at any layer of the virtualization infrastructure along with unlocking the tangible opportunities for industry and managing the life cycle of cloud-native functions using Kubernetes.

The organization of this paper is as follows: in Section II, we describe the cloud infrastructure along with the use case considered for illustrating the migration method. Section III gives the details and the motivation for implementing pod migration from one K8S cluster to another. Related works (Section IV) and concluding remarks (Section V) are further presented.

II. Architecture of the testbed

In the following, we describe the cloud infrastructure (§ II-A) that sustains the mobile core network (§ II-B).

A. Network and cloud

We consider a three-tiered cloud infrastructure (Figure 1), which today reasonably represents traditional Internet Service Provider (ISP) networks involving 2 Tier/3 Tier networks. These networks, which have a national and regional footprints, interconnect end users to Tier 1 backbone networks used to exchange international traffic. The associated cloud infrastructure reflects the architecture of ISP networks with their delivered services. The cloud infrastructure includes a (national) *central cloud*, a regional cloud (attached to a Point of Presence) referred to as *fog cloud*, and clouds closer to end user (*edge cloud*). Edge data centers host operator services (e.g., cloudRAN, firewall), Business to Business (B2B) services (e.g., enterprise network functions) or applications requiring intensive computing (e.g., cloud gaming, AR/VR, etc.). We further assume that each data centre hosts a K8S cluster;

[1]Several pods may be created to support the migration of large/distributed CNFs.

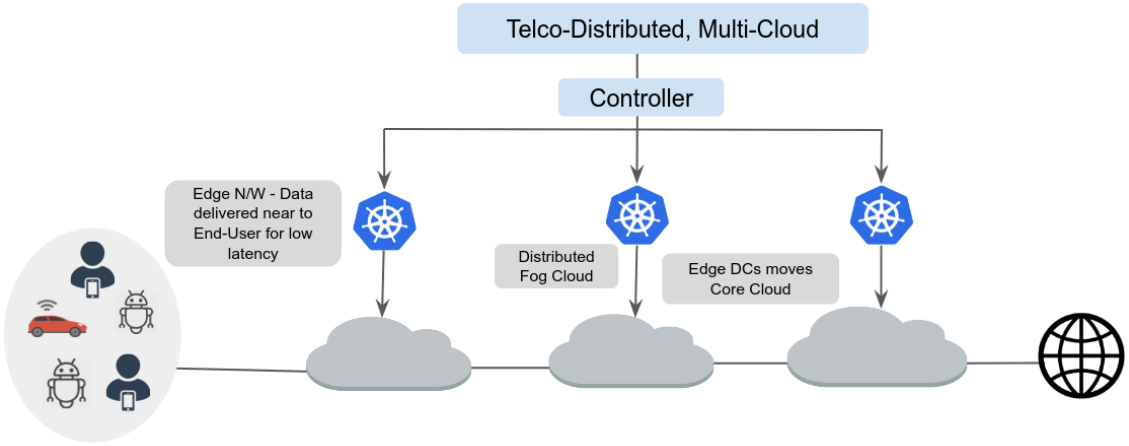


Fig. 1. Extending Kubernetes paradigm to other domains

the cloud infrastructure is therefore composed of a collection of distributed K8S clusters. Each cluster is characterised by specific resources capacity and bandwidth: by mean, the one farther to the end user is the cloud cluster, which has the largest centralised storage and compute resource, which offer high scalability and can be convincingly used on demand; followed by a fog cluster, which accumulates co-located nodes to reduce the distance between end-user devices and cloud data centres, and allows various functions to be easily moved to the end-user device for low-latency interactivity; finally, the edge cluster spreads accross edge nodes that are near the end-users and that provides comparatively lower latency at the cost of limited resource capacity compared to cloud and fog cloud.

B. 5G Mobile Core Network

We consider an open-source core network, implemented using Magma [2], which supports diverse radio technologies, including LTE, 5G and WiFi. Magma was originally designed to extend the coverage of mobile networks but today Magma is seen as an effective solution for building private 5G networks. Thanks to the multi-operator capability offered by the Federation Gateway (FEG), Magma could also be advantageously used in the context of TowerCos [3]. As depicted in Figure 2, the main components of the Magma architecture include:

- **Orchestrator (Orc8r)**: Orchestrator is a cloud service that provides a simple and consistent way of securely configuring and monitoring the wireless network. The orchestrator has 3 main functions: a Network Management System (NMS) that supports, e.g., configuration and basic monitoring capabilities, KPIs exposed through a REST endpoint, and a secure communication channel for communication between the various gateways.
- **Access Gateway (AGW):** This functions provides mobile packet core for both 4G and 5G services. It follows a distributed architecture enabling horizontal scaling with a radio access network (RAN) including e.g., eNodeBs and gNodeBs. With 5G, AGW deals with the User Plane Function (UPF), Session Management Function (SMF), and the Access and Mobility Management Function (AMF). These three functions make up the so-called Minimal Viable Core (MVC), which is the minimal set of functions required to establish PDU sessions in 5G. In the case of 4G, the MVC comprises the MME and S/PGW functions. There is no authentication function (AUSF, UDM, UDR): authentication is mocked by provisioning via the NMS the IMSIs to UEs authorized to connect to the network.
- **Federation Gateway (FeG):** This function integrates the MNO core network within Magma by providing standard 3GPP interfaces to existing MNO components (notably the HSS in 4G and the AUSF in 5G). It acts as a proxy between the Magma AGW and the operator's network and facilitates the delivery of core functions, such as authentication, data plans, policy enforcement, and charging to be compliant with an existing MNO network and the expanded network using Magma core.

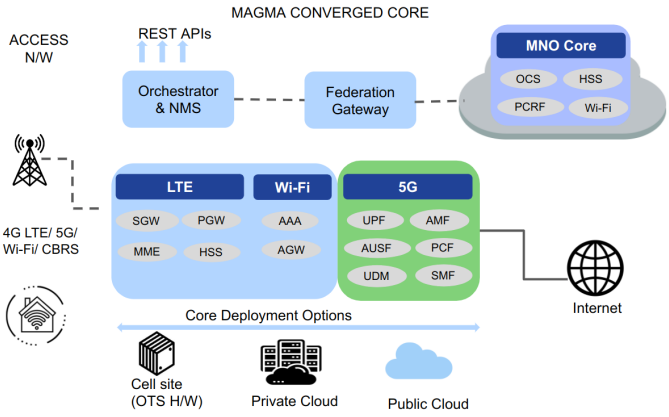


Fig. 2. Magma Architecture and its components.

C. 5G Mobile Core Network Setup

In practice, some networks functions of the Magma core network are containerized. In particular, the Magma Orchestra-

tor (Orc8r) is deployed in Kubernetes and divided into various helm charts. The orchestrator henceforth contains various pods and services that are deployed using Minikube [4] as defined in [5]. Similarly, Magma access gateway is divided into sub-functions to support LTE, WiFi and 5G core. Overall, related sub-functions are deployed on the same K8s edge cluster and are instantiated on Bare metal at different edge nodes (as depicted in Figure 3).

As new CNFs require to be placed near to the user, orchestrator's pod(s) should be moved from edge cluster to a cloud cluster to free space on the edge cluster hosting the Magma core. For that purpose, Orc8r needs to be migrated to another K8s cluster without interrupting the current communication with AGW.

III. Migration of a 5G Core Mobile Network Component

A. Design Rational

In addition to migrating orc8r, a key requirement is to ensure that (i) there is no disconnection between AGW and the migrated orc8r, which implies that the network traffic gets properly routed and (ii) that any chained service (including AGW) that communicates with orc8r is not affected by the migration. For this purpose, we rely on Traefik [6], which is a load balancer that appropriately routes the network traffic to the desired destination (in our case, the migrated orc8r). In particular, Traefik provides a ingress controller for each migrated network function that accepts the traffic from outside the destination cluster and forwards the traffic to the migrated network function. In addition, an external DNS server (herein Cloud DNS) is used to provide hostname resolution and in particular, to handle the redirection process at the DNS level rather than by proxying.

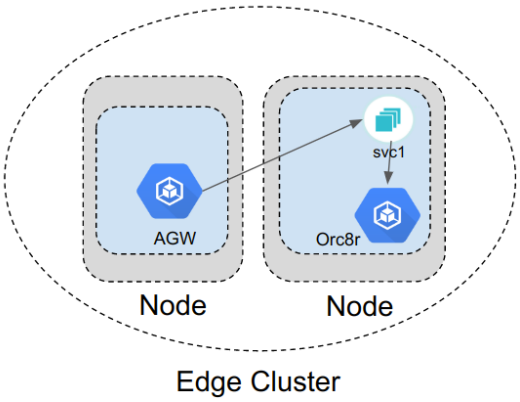


Fig. 3. Deployment of AGW and Orc8r on K8s cluster (Edge)

B. Migration Strategy - Step by Step

The methodology adopted [7] to migrate orc8r is as follows:

1) The migration process starts by copying the orc8r service that needs to be migrated and by setting up a new Kubernetes cluster (in a new node located e.g. in the cloud) in which the ocr8r copy is instantiated. For the sake of simplicity, the new instance of orc8r is named svc_2 while the original instance is named svc_1. In the configuration file, the namespace associated with svc_1 and svc_2 corresponds to *orc8r-ns*.
2) In the Traefik load balancer, a new IngressRoute is created to route the network traffic to the newly migrated service (svc_2 a.k.a. *orc8r-ns*) in the cloud cluster.
3) In Cloud DNS, a private DNS zone named $new.testnetwork.internal$ is created. In order to provide response to DNS clients that request name resolution for the newly migrated instance (svc_2 a.k.a. *orc8r-ns*), a record is added with the name $new.testnetwork.internal$ pointing to the load balancer IP address (located in a new cluster).
4) Finally, the old orc8r instance is deleted. During the migration, AGW continues communicating with the orc8r running in the cloud cluster, with essentially no downtime; the whole migration process remains completely transparent for AGW thanks to the DNS redirection and the routing performed by the load balancer.

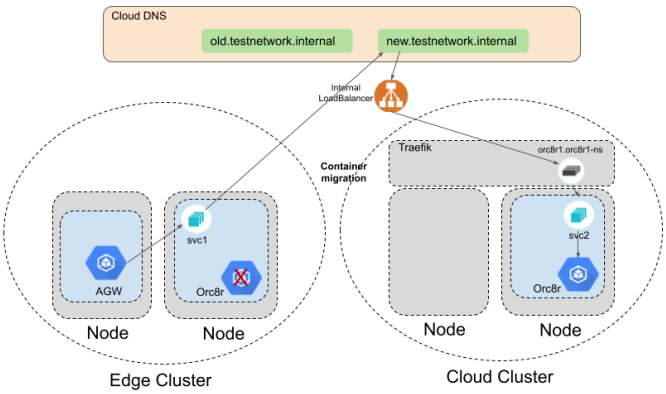


Fig. 4. Layout of Orc8r migration in Cloud cluster

C. Demonstration

We have used sample Kubernetes pods, namely AGW and Orc8r of Magma components and performed the manual migration steps instead of executing it through a controller. Two different VMs have been created on a server and Minikube is used to create a Kubernetes cluster on top of each VM. To illustrate the deployment of pods, we have reported some screenshots:

- Figure 5 shows the AGW and Orc8r pod deployment on the same cluster (namely, the edge cluster) through the controller VM.
- Further, Figure 6 shows the after migration view of the new running Orc8r pod on another minikube cluster (corresponding to cloud cluster) created on another VM.
- Likewise, an old orc8r pod has been deleted from the edge cluster after the activation of new orc8r pod (on cloud cluster) in Figure 7.

```
ubuntu@vm-4:~$ kubectl get pods -n orc8r-ns
NAME     READY   STATUS    RESTARTS   AGE
orc8r    1/1     Running   0          92s
ubuntu@vm-4:~$
ubuntu@vm-4:~$ kubectl get pods -n agw-ns
NAME    READY   STATUS    RESTARTS   AGE
agw     1/1     Running   0          36s
```

Fig. 5. Accessing the deployment of AGW and Orc8r on Edge cluster through Controller

```
ubuntu@vm-2:~$ kubectl get pod -n orc8r-ns
NAME     READY   STATUS    RESTARTS   AGE
orc8r    1/1     Running   0          14s
```

Fig. 6. Deployment of Orc8r on Cloud cluster

```
ubuntu@vm-1:~$ kubectl get pod -n orc8r-ns
NAME     READY   STATUS    RESTARTS   AGE
orc8r    1/1     Running   0          76m
ubuntu@vm-1:~$ kubectl delete pod orc8r -n orc8r-ns
pod "orc8r" deleted
ubuntu@vm-1:~$ kubectl get pod -n orc8r-ns
No resources found in orc8r-ns namespace.
```

Fig. 7. Deletion of Orc8r on Edge cluster

- A Traefik IngressRoute routed the network traffic to service svc_2 if the destination hostname matches with $orc8r1.orc8r1-ns$ or $orc8r1.orc8r1-ns.svc.local$ as shown in Figure 8.

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: svc1
spec:
  entryPoints:
    - web
  routes:
  - kind: Rule
    match: Host(`orc8r1.orc8r1-ns`) || Host(`orc8r1.orc8r1-ns.svc.local`)
    services:
    - kind: Service
      name: svc2
      namespace: orc8r-ns
      port: 80
```

Fig. 8. Demonstration of yaml file of Traefik IngressRoute

- Figure 9 represents the newly created ExternalName service pointing to the DNS name (new.testnetwork.internal) which resolves into load balancer IP.

```
kind: Service
apiVersion: v1
metadata:
  name: svc1
spec:
  externalName: new.testnetwork.internal
  type: ExternalName
```

Fig. 9. Demonstration of yaml file for Externalname service

The steps described above are currently triggered by hand. But a controller for automating these steps is under development.

D. Development of the controller

In addition to a database storing information related to services, network and cloud topology, the controller contains four main components: Monitoring, Analysis, Discovery and Execute components. Based on the metrics collected from all the Kubernetes clusters, it performs the analysis and triggers the migration script based on a set of rules (e.g., rule defining that the resource limit exceeded).

For this purpose, we rely on the Prometheus [8] and Grafana [9] that are deployed as containers on controller VM (that hence acts as a Prometheus master). Prometheus is a monitoring and event alerting tool and Grafana enables the visualization & analysis of the data provided by Prometheus. In practice, a kube-prometheus-stack [10] helm chart is installed on VM for each cluster that includes Grafana and Prometheus operator. Notably, minikube cluster is deployed on a separate VM which blocks the access to Prometheus and Grafana services that are deployed in a cluster. Thus, the services Type from ClusterIP need to be upgraded to NodePort type that exposes the respective service via static port on node's/VM's IP. Also, a Nginx reverse proxy server is set up to publish ports using the Prometheus UI and Grafana service. This allows the Prometheus controller to receive the metric data from the respective cluster (see Figures 10 and 11 that depicts the CPU usage and storage IO distribution of pods created on edge cluster).

Controller also has a full access to remote clusters using the kubeconfig file. In minikube cluster, this necessitates to create a ssh proxy between the controller and cluster VMs using the following commands:

ssh -L6443:192.168.59.103:8443 10.4.11.92 $-forVM-1$

ssh -L6444:192.168.59.110:8443 10.4.11.213 $-forVM-2$

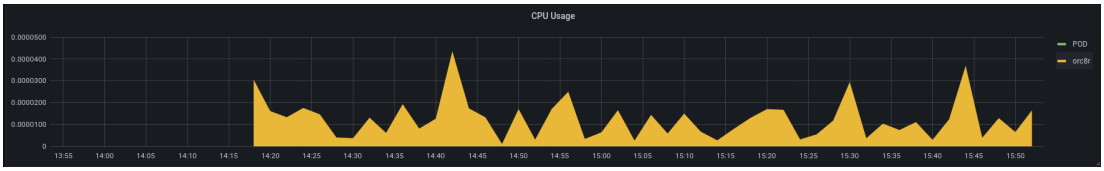
Then, using the command *kubectl config use-context* <*context-name*>, we can switch between the clusters.

Now, the controller has an access to the remote cluster in addition to metric data and can manage and handle the migration process based on the events or strategy introduced in [11].

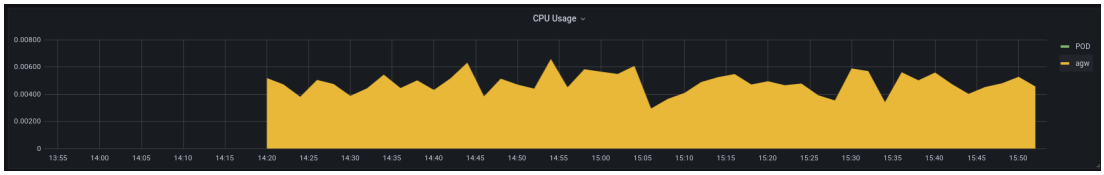
Whenever necessary, the script is executed by the controller to trigger the migration of pods from one cluster to another. In particular, the controller makes decisions and manages the load among different nodes of a cluster based on their available resources. Further, the proposed strategy follows a heuristic approach to choose an appropriate destination node. It also considers the ephemeral nature of containerized services and the distance between the data centers located on geo-distributed locations. Compared to the static placement approach, this migration testbed ensures an optimal placement of pods considering resource load and end-to-end latency.

IV. Related work

The work named CloudHopper [12] supports the live migration of interdependent containerized applications across different cloud providers (namely, Amazon Web Services,



(a) CPU usage of Orc8r



(b) CPU usage of AGW

Fig. 10. Grafana visualization of CPU resources by the Orc8r and AGW pods.

(a) Storage IO distribution of Orc8r

(b) Storage IO distribution of AGW

Fig. 11. Grafana visualization of Storage IO distribution of the Orc8r and AGW pods.

Google Cloud Platform and Microsoft Azure). It aims at reducing the downtime time and supports pre-copy migration, connection holding and redirection to keep the user unaware about the migration process.

Authors in [13] execute the docker container migration across data centers in cloud networks. While migrating the container, they leverage the hierarchy feature of image layers by avoid repetitive transmission of existing layers at the target node resulting in 57% lessened total migration time, 55% lower image migration time, and 70% of downtime on average in comparison to mentioned state-of-the-art. Further, a live stateful container migration solution has been proposed in [14]. It supports just-in-time zero-copy migration that allows the container at destination node to restart prior to transmission of whole states.

In [15], a geo-distributed fog network based approach is presented for stateful container migration within different nodes of Kubernetes cluster. To handle the time consuming process of large-sized container's disk states transmission, authors leveraged the layered structure of OverlayFS file system [16]. It transparently snapshots the pod volumes and transfers the snapshot content to the target server before actual container migration. Whereas, at the source end, the snapshot content becomes read-only and a new empty read/write layer is added on top.

Further, an open source tool - Velero [17] allows the migration of Kubernetes resources and persistent volumes. It utilizes the Kubernetes API instead of Kubernetes etcd to extract and restore the states in contrast to other tools. This API-oriented approach is beneficial in case a user does not have an access to etcd databases. Also, resources exposed by API servers are simple to backup and restore even for several etcd databases. To extend the backing up and restore functionalities of any type of Kubernetes volume, a program called restic [18] is required to activate. Velero release is available on GitHub [19].

A work in [20] implemented the container migration on edge infrastructure and designed it as a third party tool. They leveraged the layer structure of the docker container storage system to shorten the migration time. The strategy is to start preparing the target edge node prior to actual commencement of the migration process. Where the layers underlying the top layer of Docker image are transmitted in advance (as they remain unchanged during the container's whole life cycle) and only the top layer is transmitted during process execution.

The stateful live migration of containers across Mobile Edge Computings (MECs) in [21] follows the three-layered architecture - Base, Application and Instance layer. They aimed to place the service near to the user and minimize the overall migration time and service downtime. In the first steps, the base layer (excluding the application) composed of primary components (i.e., guest OS, kernel, etc.) is transmitted

on each MEC in order to avoid the transmission of the base layer for each migration request. Then, the application layer composed of idle applications and its data is passed on while migration is triggered and keeps the service running. Finally, the active states in the instance layer are transmitted after suspending the service. Therefore, the service downtime reflects the transmission of the instance layer. However, the detailed experimental explanation is not provided in the paper due to lack of space.

Moreover, the open-source solution - Cloudify [22] is a multi-cloud and edge orchestration platform that claim the the pod migration without interrupting the service from one node to another within the Kubernetes cluster.

A service-oriented architecture KubeVirt [23] provides additional functionality to Kubernetes. It supports the migration of VM instances (acted as a pod) from one host to another host within a cluster. To make it profitable for containers, the containerized applications can be run inside the VM. A release is available on GitHub [24].

Although the work in [25], [26] is considered to be a prototype, it enables the pod migration of stateful containers in Kubernetes. The release includes the additional commands to migrate and checkpoint running pods within single/multiple clusters through modified kubelet and customized cri. The work around Edge Multi Cloud Orchestrator (EMCO) [27] focuses on deploying edge microservices across multiple clusters. Compared to single cluster migration, multi-cloud migration requires various network configurations which also rely on distinct use-cases. For instance, the bursty traffic related use-case in [28] allows to decide the triggering of migration process. There are still limitations to solve specific to dynamic management of resources while keeping them alive and facilitating an automated management of composed microservice during their whole life cycle, including their instantiation, migration and termination. Also, it is required to tackle the connection alive during termination and re-establishment as a chain of microservices not only communicate with end-users but also with respective microservices that may be placed on the servers in different clusters.

V. Conclusion

Our objective is to perform a live migration of containerized network functions from one Kubernetes cluster to another to support the effective migration of 5G network functions. The key lesson learnt is that by implementing a relatively straightforward strategy involving DNS redirection and proper forwarding, a controller may trigger migration (on the basis of monitoring data provided by Prometheus) of pods or containers across distributed K8S clusters. In terms of performance, the migration time (in the testbed) is about a few tens of milliseconds, due to the creation time of pods. In real networks with large K8S clusters and significant propagation times, the migration time may be increased upto a few tens of seconds but would remain within acceptable bounds when compared with manual configuration.

References

[1] "Kubernetes, https://kubernetes.io," accessed: 05-Dec-2022.

[2] S. Hasan, A. Padmanabhan, B. Davie, and al., "Building flexible, low-cost wireless access networks with magma," *20th USENIX Symposium on Networked Systems Design and Implementation*, 2023.

[3] F. Guillemin and L. L. Beller, "Analysis of business opportunities for tower companies enabled by virtualization," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2022, pp. 163–168.

[4] "Minikube, https://minikube.sigs.k8s.io/docs/," accessed: 09-Dec-2022.

[5] "Magma Orc8r deployment on Minikube, https://magma.github.io/magma/docs/orc8r/dev_minikube," accessed: 09-Dec-2022.

[6] "Traefik, https://traefik.io/," accessed: 05-Dec-2022.

[7] "Migrating applications between Kubernetes clusters, https://medium.com/google-cloud/migrating-applications-between-kubernetes-clusters-8455cf1bfccd," accessed: 05-Dec-2022.

[8] "Prometheus, https://prometheus.io/," accessed: 13-Jan-2023.

[9] "Grafana,https://grafana.com/," accessed: 13-Jan-2023.

[10] "kube-prometheus-stack,https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack," accessed: 13-Jan-2023.

[11] "A microservice migration approach to controlling latency in 5G/6G network," 2022, submitted for publication.

[12] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2529–2538.

[13] B. Xu, S. Wu, J. Xiao, H. Jin, Y. Zhang, G. Shi, T. Lin, J. Rao, L. Yi, and J. Jiang, "Sledge: Towards efficient live migration of docker containers," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 321–328.

[14] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2137–2142.

[15] P. S. Junior, D. Miorandi, and G. Pierre, "Stateful container migration in geo-distributed environments," in *CloudCom 2020-12th IEEE International Conference on Cloud Computing Technology and Science*, 2020.

[16] N. Mizusawa, K. Nakazima, and S. Yamaguchi, "Performance evaluation of file operations on overlayfs," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2017, pp. 597–599.

[17] "Velero. https://velero.io/," accessed: 02-Dec-2022.

[18] "Restic. https://velero.io/docs/v1.7/restic/," accessed: 02-Dec-2022.

[19] "Velero GitHub. https://github.com/vmware-tanzu/velero," accessed: 02-Dec-2022.

[20] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2018.

[21] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Migrating running applications across mobile edge clouds: poster," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 435–436.

[22] "Cloudify [Official site]. https://cloudify.co/," accessed: 02-Dec-2022.

[23] "KubeVirt [Official site]. https://kubevirt.io/," accessed: 02-Dec-2022.

[24] "KubeVirt GitHub. https://github.com/kubevirt/kubevirt," accessed: 02-Dec-2022.

[25] "Podmigration-operator https://github.com/schrej/podmigration-operator," accessed: 02-Dec-2022.

[26] "Podmigration-operator [Extended version]. https://github.com/ssu-dcn/podmigration-operator," accessed: 02-Dec-2022.

[27] "Edge Multi-Cluster Orchestrator (EMCO). https://smart-edge-open.github.io/ido-specs/doc/building-blocks/emco/smartedge-open-emco/."

[28] V. Balasubramanian, M. Aloqaily, O. Tunde-Onadele, Z. Yang, and M. Reisslein, "Reinforcing cloud environments via index policy for bursty workloads," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–7.