2021 IEEE International Conference on Service-Oriented System Engineering (SOSE) | 978-1-6654-3477-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SOSE52839.2021.00019

Collaborative Cloud - Edge: A Declarative API orchestration model for the NextGen 5G Core

Oana-Mihaela Ungureanu
University Politehnica of Bucharest
Bucharest, Romania
oana.ungureanu30@gmail.com

Călin Vlădeanu
University Politehnica of Bucharest
Bucharest, Romania
calin@comm.pub.ro

Robert Kooij
Delft University of Technology
Delft, The Netherlands
r.e.kooij@tudelft.nl

***Abstract*—As the edge computing era matures, it encompasses a broader range of applications that will be able to opportunistically use both the edge and the cloud mainly determined by the location constraints or the mission-critical reasons. Moreover, the convergence of verticals with the next generation mobile network will lead to an imminent edge-cloud collaboration that needs to be carefully addressed by Mobile Network Operators (MNOs). This paper proposes a novel approach to orchestrate the next-generation 5G core running as Cloud-Native Network Functions (CNFs) closer to the edge network while offloading the computation power required to ensure on-demand processing to a central cloud infrastructure. In this manner, full service isolation and network segmentation can be achieved independently from the geographical region with a high degree of scalability. Our main objective is to automate the CNFs provisioning in a declarative API-style in order to easily manage the deployment of applications and services across multiple clusters. Therefore, in our setup we considered crucial to assess the benefits of running the 5G Service-Based Architecture (SBA) in the cloud with multi-tenancy capabilities and evaluate the performance in terms of latency, while keeping the network control at the edge.**

***Index Terms*—edge computing, CNF, SBA, multi-tenancy, network slicing**

I. Introduction

The cellular network architecture is evolving today at a high pace towards a distributed, decoupled and infrastructure agnostic ecosystem mostly due to the rapid growth of vertical industries (automotive, media and entertainment, gaming, healthcare, sensors, augmented reality etc.) that fully rely on the mobile network operators (MNOs) infrastructure to aggregate massive amounts of data and increased computation power as well as storage in the end-user proximity. The mobile communications ecosystem defined by 5GPPP [1] and ETSI [2] strives to accommodate the coexistence and isolation of all different tenants starting with the network slicing provisioning to the level of service quality by ensuring a NFV (Network Functions Virtualization) Management and Orchestration (MANO) [3] system that can be deployed and owned by each tenant.

In recent years, virtualization and network programmability of the Evolved Packet Core (EPC) became the de facto standard, together with the emergence of Network Functions Virtualization (NFV) and Software Defined Networking (SDN), thus numerous organizations tried to standardize the implementations of the MANO framework [4]. Among them are large open-source projects like ONAP [5], SONATA [6], OPNFV [7], Cloudify [8], Open Baton [9], Open Source MANO (OSM) [10] or OpenStack Tracker [11]. Papers [12] and [13] recall most of the projects that include the software components for the 4G and 5G mobile core network that can be run as Virtual Network Functions (VNFs).

The emergence of Cloud Native Computing Foundation (CNCF) [14] fostered the NFV evolution to a new paradigm, Cloud-native Network Function (CNF) and shifted the MNOs interest to run VNFs as cloud native applications in the public cloud. A successful migration path from a legacy monolithic architecture towards a cloud-native one does not address only the hardware decoupling of VNFs, but also a modular design through APIs and a new distributed architecture based at its core on automation and self-management. Since none of the aforementioned MANO frameworks [5]–[11] addresses the management of CNFs, a rightful candidate for the container orchestration is Kubernetes [15] to help MNOs modernize their costly infrastructure and transform it to a modular platform designed for multiple integrations with Operational Support Systems (OSS) for better performance and resiliency. Nevertheless, Kubernetes was initially designed for cloud data centers and many of the edge computing scenarios are not solved by native Kubernetes as they require a lightweight management plane at the edge.

The concept of *network segmentation* or *network slicing*, recently introduced a particular component to the 5G architecture called Multi-access Edge Computing (MEC) that was proposed by ETSI in 2014 [16] and can be deployed at the edge network closer to the end-users with the aim to address verticals and especially IoTs demands in terms of Ultra Low Latency Communications (URLLC), Enhanced Mobile Broadband (eMBB) and a higher computation power for massive Machine Type Communications (mMTC) [17]. Although most literature focuses on the containerization of the MEC [18]–[20], from the architectural point of view, the work is divided between hosting the 5G workloads either in the edge or central cloud. Even though a central cloud offers the availability of powerful computation and storage resources, it can generate significant delay when it comes to intensive processing operations. On the other hand, the edge cloud is very often in the proximity of the Radio Access Network (RAN) which leads to a decrease in delay and jitter,

2642-6587/21/$31.00 ©2021 IEEE
DOI 10.1109/SOSE52839.2021.00019

but with limited computation and storage capacity [21]. Cisco introduced in 2012 a new paradigm called ”fog computing” in order to reduce the burden of transferring data from the central cloud to the edge network [22]. Nevertheless, the main challenge remains to trace the border for the edge cloud and define the intersection of 5G with the public cloud. MNOs already have well-established business models as Internet Service Providers (ISPs) and they are attractive channels due to their existing network infrastructure. Nonetheless, the three main public cloud providers current initiatives mostly cover IoT scenarios, i.e., Amazon via AWS Wavelength [23], Microsoft via Azure for IoT [24] and Google Cloud via Global Mobile Edge Cloud [25], although none of them have yet offerings on their marketplaces for managed 5G-as-a-Service (5GaaS). Nevertheless, their interest recently shifted towards hosting the 5G SBA since AWS already announced integration support for the Open5GS project [26].

This paper presents a synergy between cloud and edge as it encompasses a flexible and programmable architecture of the 5G SBA running in the cloud that can be easily scaled out and each service managed independently at the edge where the business logic resides. According to the literature a complete implementation of an orchestration model for both MEC and the 5G packet core running in microservices, is currently lacking. On the other hand, a limited number of initiatives [27], [28] deal with the federation challenges encountered in multi-cluster and multi-cloud deployments. Our contributions are three-fold. First, we propose a declarative container orchestrator that uses APIs for the creation, configuration and management of the 5G SBA at the edge with computation and processing power offloaded to the public cloud. Secondly, our approach presents a collaborative cloud-edge model that ensures an isolation layer for the different verticals' needs with a granular monitoring and orchestration of the Network Functions (NFs). Consequently, we evaluate the proposed model in terms of response time latency for the inter-services communication and memory utilization of the control plane applications. In this paper, we also investigate the geo-distribution capabilities of edge computing, whether the existing container orchestration frameworks offer support for such deployments and equally succeed to provide full lifecycle management regardless of the location or even the infrastructure provider.

In Section II, we present the relevant open-source projects with various implementations of the MANO framework as well as the 5G virtualized mobile core and different MEC orchestrators. We also reference the papers that cover the main use cases for IoT and mobile devices. Section III describes the available Kubernetes orchestrators dedicated to the edge where we highlight the limitations of these tools in satisfying the multi-cluster/multi-cloud and federation scenarios. In Section IV, we compare two deployment models in terms of response time and memory utilization for the most intensive control plane applications. For both our setups we assess the control management deployed at the edge as well as from the 5G service mesh provisioning perspective. Section V presents our final conclusions regarding the proposed model and open topics for future research.

II. Related work

The body of literature on tracking offers of a plethora of open-source projects that have as objective either the development of the MANO framework or the virtualized mobile packet core. For instance, a popular open-source project is ONAP (Open Network Automation Platform) [5] initiated by the Linux Foundation to simplify the orchestration of the VNFs and to integrate the SDN module while concentrating on specific use cases for vCPE (virtual Customer Premises Equipment) and VoLTE (Voice over LTE). Linux Foundation also launched in 2014 another open-source project called Open Platform for Network Function Virtualization (OPNFV) [7] dedicated to the integration and implementation of VNF as well as the virtualization of 5G core. Another project that addresses an implementation of the NFV-MANO architecture in 5G is SONATA [6]. The applications and use cases considered are in the area of monitoring IoT devices, traffic optimization, virtual Content Delivery Network (vCDN) to increase the scalability and automate the virtualized Evolved Packet Core (vEPC) configuration.

Cloudify [8] is an orchestration project for the NFV developed in TOSCA language (Topology and Orchestration Specification for Cloud Applications) that describes how to orchestrate cloud based applications and employs an orchestration engine based on different topologies. A similar approach to Cloudify is employed by OpenStack Tacker [11], a project developed under OpenStack [34] umbrella. OpenStack Tacker is based on MANO architecture defined by the ETSI group and aims to build an orchestrator with VNF Manager role to create and manage NFs. In a joined effort to standardize the specifications on the NFV-MANO architecture, the Fraunhofer Fokus Institute and the Technical University Berlin contributed with the project OpenBaton [9]. Most of the developed components are designed as cloud-native applications. The Open Source MANO (OSM) [10] was initially introduced in 2016 under ETSI umbrella and has as contributors many companies, telecom operators as well as equipment providers and focuses on developing programmable interfaces to integrate different modules and components. Some of these scenarios are around “service chaining”, ”network segmentation” and orchestration of the MEC component. In paper [35] the authors propose a setup based on OpenAirInterface [29] virtualized EPC that uses OSM as a VNF orchestrator running on a private cloud.

The transition of MEC towards a cloud architecture is discussed in papers [36] and [37] due to the multitude of advantages such as storage power, backup, on-demand provisioning, scalability or multi-tenancy. Therefore, the cost of storing and processing could be shared between mobile operators and cloud providers. Paper [38] addresses a synthesis of the different scenarios that serve the MEC architecture, especially in the IoT industry where billions of devices are connected to the Internet. Among the benefits in adopting

TABLE I: Open-source projects for virtualized 4G and 5G mobile packet core

OSS Project	*Language*	*License*	*4G*	*5G*	*CNF*	*Active Contributors*
OpenAirInterface [29]	C	Apache v2.0	yes	yes	wip*	OpenAir Software Alliance EUROCOM
NextEPC	C	GNU AGPLv3	yes	wip*	wip *	NextEPC
corenet [30]	Python	GPL-2.0 License	yes	no	no	Corenet
openLTE [31]	C++	GNU AGPLv3	yes	no	no	openLTE
open5GS	C	GNU AGPLv3	yes	yes	yes	Open5GS
OMEC	C++	Apache v2.0	yes	yes	no	ONF, Intel, Deutsche Telekom, Sprint, AT&T
free5GC [32]	Go, C	Apache v2.0	no	yes	wip*	Free5C
srsLTE [33]	C++	GNU AGPLv3	yes	no	no	srsLTE
OpenNESS	Go	Apache-2.0	yes	yes	yes	Intel

* work in progress.

MEC for the IoT use cases is the high scalability, the offloading for computation processing, resource allocations as well as security. The authors of [39] present a study tracing the principal directions in IoT by analyzing the key technologies and the current challenges.

A considerable amount of work is mainly divided between different solutions to orchestrate NFVs in a dedicated cloud architecture presented in papers [12], [21] and virtualized functions developed for 4G and 5G mobile core. Table I summarizes the existing OSS projects mainly dedicated to the virtualization of mobile packet core according to the 4G specifications and shows few initiatives for the development of NFs for 5G SBA. The most common tool used for the virtualized 4G core is NextEPC [40] and its integration with SDN and MEC is thoroughly addressed in both [13] and [41]. The authors of [41] also propose an implementation of the MEC orchestrator based on Kubernetes that integrates the open-source project NextEPC. In a situation where an application requires to be moved from one MEC server to another, containers play a fundamental role in reducing the migration duration and decreasing the overall downtime. A similar approach for deploying MEC in a microservices architecture orchestrated by Kubernetes and integrated with NextEPC is proposed by authors of paper [13]. The main advantages for adopting this architecture are discussed from a DevOps perspective.

Among the projects that support both 4G and 5G VNFs as well as CNFs are Open5GS [42] that we will further consider in our setup presented in Section IV and OpenNess [43]. Open Network Edge Services Software (OpenNESS) is a tool to simulate the MEC architecture with the aim to provision CNFs. This project was developed in collaboration with Intel and it runs entirely on a microservices based architecture as it provides APIs for the community. The Edge Multi-Cluster Orchestration (EMCO) is a geo-distributed application orchestrator for Kubernetes also developed under OpenNESS project. The EMCO aims to automate the deployment of applications and services across clusters. It acts as a central orchestrator that can manage edge services and network functions across geographically distributed edge clusters.

From a container orchestrator perspective, several approaches exist in the literature and yet even few of them address the federation and multi-cluster challenges. The authors of [20] implement a testbed built on a different container orchestrator i.e., Docker Swarm [44] focusing on the vehicular communication scenario. Despite the fact that it does not address any of the IoT or mobile use cases, paper [27] presents a special lightweight resource container orchestrator, named Fledge that connects to a Kubernetes cluster in the cloud through a Virtual Kubelet [45] using OpenVPN [46]. In this manner multiple nodes with Fledge agents installed can be scaled and instantiated at the edge even though the orchestration relies on its own container networking implementation and no kube-proxy is deployed. Looking at the multi-cluster approach, paper [28] presents a federated approach for the edge computing container orchestration by analyzing the behavior and limitations of Kubernetes centralized control plane, whereas the edge sites are deployed in multiple micro data centers across different locations.

III. EDGE COMPUTING IN THE CLOUD LANDSCAPE

The crossway between ”fog computing” and ”edge computing” is properly highlighted in the Ph.D. dissertation [47] since both concepts pledge for bringing the processing power closer to the end-user device. However, edge computing focuses more on the use cases defined by the MNO’s whereas fog computing extends the applicability to broader areas such as smart cities or remote surveillance cameras. Both concepts integrate the SDN technology for traffic optimization and policy-based to improve the service quality. This section discusses some of the available solutions for Kubernetes at the edge in terms of low resources, geo-distribution and edge device management highlighting the pros and cons for each of the projects. We also evaluate and propose a multi-cluster and multi-cloud architecture that accommodates the deployment of 5G SBA in the public cloud.

A. Container orchestrators for the edge computing

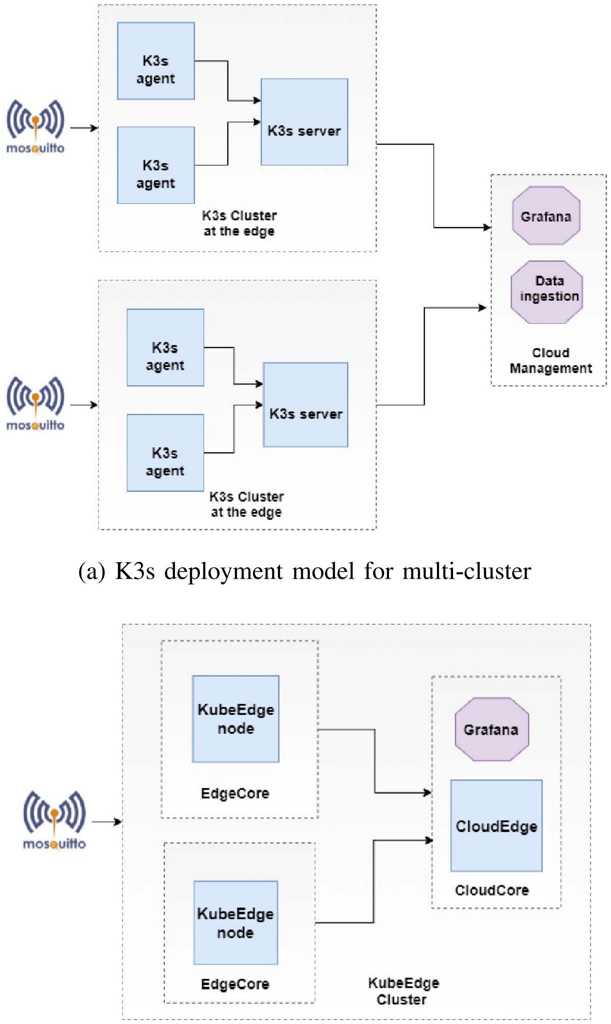
Kubernetes is already a standard for cloud-native applications, therefore we will consider the Kubernetes orchestrator as a reference in our research. In a Kubernetes orchestrator architecture the smallest entity in a cluster is called a *pod* as it can host an application or a process. One micro-service can be seen as a collection of pods and policies. A *job* can run multiple tasks to create one or more pods. At control plane level there is an *API server* responsible with updating the pods’ state, a *controller-manager* that monitors the cluster state and a *scheduler* that dictates to which computation nodes the pods are allocated. A *kubelet* agent runs on each node and a *kube-proxy* is responsible with the traffic policies [48].

Although Kubernetes was natively designed for cloud data centers, when it comes to Kubernetes at the edge several open-source projects exists to spin up clusters. For instance, *Kind* [49] is a tool to run a Kubernetes cluster into Docker containers developed by Kubernetes Special Interest Groups (SIGs). It supports multi-node and multi-cluster deployments and can be easily integrated with other Kubernetes-style APIs declarative tools, i.e., *ClusterAPI (CAPI)* [50]. *Kubedge* [51] is an open-source project under CNCF (Cloud Native Computing Foundation) built on Kubernetes with the aim to orchestrate IoT applications as it incorporates the business logic running at the edge. The *KubeEdge* architecture has two main components: *cloud* and *edge* with the corresponding modules *hub* and *edge*. The *EdgeHub* represents a web socket client responsible to communicate with the *CloudHub* component hosted in the cloud. The *EdgeController* is deployed in the cloud to control the state synchronization of the Kubernetes API Server with the nodes, applications and configurations of the edge.

Another tool dedicated to run Kubernetes at the edge is *K3s* [52] which comes with a lightweight version of Kubernetes. This tool was developed by Rancher Labs and it is mainly suitable for IoT use cases and edge technologies. The deployment is split into *K3s servers* and multiple agents that can run at the edge and unlike *KubeEdge* it does not require cloud-side communication. A production environment normally requires a cluster management solution on top of *K3s* for the application management, monitoring, alarming, logging and security across the cluster, but for the time being Rancher does not support this capability [53].

In Figure 1 we compare the *K3s* and *KubeEdge* architectural models for multi-cluster deployment. Both tools allow the event bus subscribe to a device data using an Message Queuing Telemetry Transport (MQTT) broker that collects messages from the clients. Mosquitto [54], the supported MQTT broker in *KubeEdge*, is a centralized message broker and serves as a gateway for the sensors. Figure 1 (a) illustrates the *K3s* main components, the *K3s server* agent that runs in different clusters at the edge and sends data to the cloud for monitoring and application management (i.e., Grafana [55]). In comparison to the *KubeEdge* deployment from Figure 1 (b), *K3s* only allows to run a full Kubernetes cluster on the edge and presents a decentralized model since each edge node requires additional deployment of the Kubernetes management cluster. On the other hand, the *CloudEdge* component specific to *KubeEdge* can provide centralized management only for one Kubernetes cluster at the edge. In order to ensure isolation and a centralized management of different Kubernetes cluster at the edge, we need to look at a multi-cluster management approach.

Kubernetes Federation v2 also named “KubeFed” [56] was one of the first projects developed under a SIG within CNCF organization. Despite the fact it cannot accommodate all scenarios, ”KubeFed” shares some capabilities for multi-cluster scenarios. The main difference when compared to multi-cluster is that federated clusters share parts of the configuration managed by a main entity represented by the host cluster in charge to propagate the configuration to the member clusters. The federated configuration along with the cluster specifications can be defined as a series of templates and policies.



(a) K3s deployment model for multi-cluster

(b) KubeEdge deployment model

Fig. 1: Overview of K3s and KubeEdge multi-cluster architecture

B. Multi-Cluster vs. Multi-cloud vs. Kubernetes Federation

The microservices architecture is implemented using containers. Compared to a monolithic design model where the entire system runs on a dedicated hardware, in a microservice-based architecture, different processes of an application can run in one or multiple containers. One of the primary advantages of cloud-native applications is their capability to be configured declaratively in an API-style in order to ensure a modular system. The methods in which they are implemented are indubitably dictated by the multiple integrated technologies i.e., orchestrator, operators, Custom Resource Definitions (CRDs), etc.

C. Multi-Cluster vs. Multi-cloud vs. Kubernetes Federation

One particular tool that follows these practices is the *ClusterAPI (CAPI)* [50] framework developed by the Kubernetes Special Interest Group (SIG) Cluster Lifecycle under CNCF. The CRDs are defined in *CAPI* though templates in order to extend the APIs exposed by Kubernetes API server and allow users to create new resources such as Kubernetes clusters and virtual machines (VMs) that host the nodes on which the cluster is running. Moreover, the CAPI controller is responsible to adjust the user application requirements. *CAPI* also supports the integration with multiple cloud providers either in a public cloud or in a private cloud. In order to scale an application across multiple data centers, cluster federation is required to replicate workloads in all member clusters. From a multi-cluster perspective, we identified several reasons to employ a *CAPI* approach. We summarize below the main benefits for running the next mobile generation containerized core in multiple clusters deployed across multiple clouds:

a) ***Full isolation:*** simultaneous network slicing to ensure on-demand resources that translate into QoS and latency for both IoT and mobile dedicated services. Different tenants in the cloud run separate slices of the mobile core corresponding to different clusters for each type of application. For example, an URLLC application can run in slice 1, eMBB application in slice 2 and mMTC application in slice 3. To follow software development compliance rules (i.e., DevOps), it is possible to separate development and staging environments by running them in different clusters isolated from production.

b) ***Multi-region:*** a best practice when running any type of application in the cloud is to provision it in different regions. This covers the availability and failover concerns as well as latency or geographical location in case of data protection and GDPR compliance.

c) ***Multi-cloud:*** adopting a multi-cloud infrastructure ensures disaster recovery capabilities and avoids the lock-in with a certain cloud provider.

d) ***Scalability:*** is no longer a concern across cloud deployments, even though the service limits of the Kubernetes can be a constraint and it can vary from one cloud provider to another (e.g. a maximum number of pods in a cluster).

In Figure 2 we illustrated the CAPI capabilities for multi-cluster and multi-cloud deployments. The ClusterAPI provider is designed to operate different third-party infrastructure vendors as well as the management of the clusters and its machines. In order to integrate with multiple cloud architectures, these controllers interrogate a component called *actuator* for the communication with the cloud provider. The main role of an actuator is to update the cluster state. Each cloud and on-premises environment has its own dedicated ClusterAPI providers that ensure cluster provisioning, i.e., CAPI provider for AWS *(CAPA)* whereas CAPI for vSphere *(CAPV)* is dedicated to the VMware provider [50]. The CAPI provider for AWS (*CAPA*) covers both Elastic Compute (EC2) deployment as well as the Elastic Kubernetes Service (EKS), i.e., *CAPA-EKS*. Hence, the mobile packet core can run in different tenants in the cloud or in the same tenant but in different virtual private clouds (VPC) corresponding to a different slice. The main role of a VPC is to provide network segmentation and security throughout policy configuration.

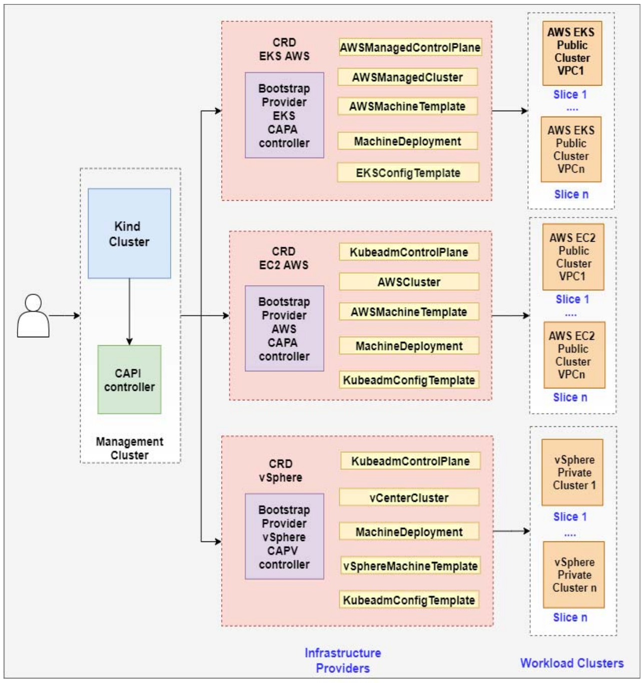
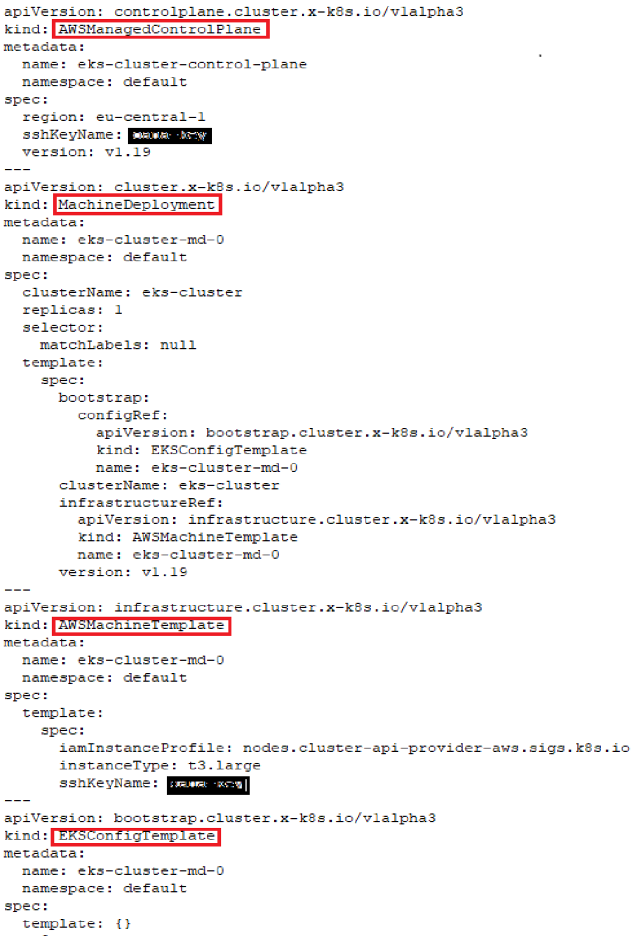


Fig. 2: Multi-cluster and multi-cloud architecture using CAPI

The *CAPI* framework consists of three concepts: the *management cluster* that stores the information of all cloud providers, the *bootstrap providers* used to install the Kubernetes control plane nodes and the *workload* or *tenant clusters* created as cluster resource objects associated with the cloud providers [57]. A user can define different types of resources in a declarative manner using Custom Resource Definitions (CRDs) that slightly differ based on the cloud provider infrastructure. Since the workload cluster is created in a declarative manner, we used the template shown in Figure 3 to display the provisioned CAPI resources for an EKS cluster. In comparison to AWS EC2 where the workload clusters are deployed directly on EC2 compute machines, EKS is a Kubernetes managed cluster service in AWS that introduces another layer of management in the containers operations. For this reason the resource definitions, i.e., CRDs differ from the AWS EC2 and vSphere configuration. The following resources are required for the provisioning of the EKS clusters [58]:

- **AWSManagedControlPlane** is the equivalent of *KubeadmControlPlane* used to declare the properties of the EKS control plane and its related AWS networking and Identity Access Management (IAM) roles.
- **AWSManagedCluster** represents a mechanism to integrate with *CAPI*, provides information about the



```
apiVersion: controlplane.cluster.x-k8s.io/v1alpha3
kind: AWSManagedControlPlane
metadata:
  name: eks-cluster-control-plane
  namespace: default
spec:
  region: eu-central-1
  sshKeyName: [redacted]
  version: v1.19
---
apiVersion: cluster.x-k8s.io/v1alpha3
kind: MachineDeployment
metadata:
  name: eks-cluster-md-0
  namespace: default
spec:
  clusterName: eks-cluster
  replicas: 1
  selector:
    matchLabels: null
  template:
    spec:
      bootstrap:
        configRef:
          apiVersion: bootstrap.cluster.x-k8s.io/v1alpha3
          kind: EKSConfigTemplate
          name: eks-cluster-md-0
      clusterName: eks-cluster
      infrastructureRef:
        apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
        kind: AWSMachineTemplate
        name: eks-cluster-md-0
      version: v1.19
---
apiVersion: infrastructure.cluster.x-k8s.io/v1alpha3
kind: AWSMachineTemplate
metadata:
  name: eks-cluster-md-0
  namespace: default
spec:
  template:
    spec:
      iamInstanceProfile: nodes.cluster-api-provider-aws.sigs.k8s.io
      instanceType: t3.large
      sshKeyName: [redacted]
---
apiVersion: bootstrap.cluster.x-k8s.io/v1alpha3
kind: EKSConfigTemplate
metadata:
  name: eks-cluster-md-0
  namespace: default
spec:
  template: {}
```

Fig. 3: Template used to provision the EKS workload cluster

AWSManagedControlplane state, any changes in the API server endpoints or failure domains.

- **EKSConfigTemplate** specifies the user data required when creating the EC2 instances for the worker nodes.

The ClusterAPI Provider for AWS (*CAPA*) recently added support for the EKS Cluster API Bootstrap Provider (*CAPA-EKS*) [59]. Nevertheless, this functionality is available only for experimental purposes and it is not recommended to use in production environments. The resources listed below are common to all three environments (AWS EKS, AWS EC2 and vSphere) :

- **Cluster** contains the specifications required by the infrastructure provider to create a Kubernetes cluster i.e., Classless Inter-Domain Routing (CIDR) blocks for pods and services (e.g. *AWSManagedCluster* for AWS Elastic Kubernetes Service (EKS), *AWSCluster* common for AWS EC2 and VMware vSphere.
- **Machine** is responsible for the minimal configuration of a Kubernetes node (e.g. kubelet version).
- **MachineSet** ensures the desired number of *Machine* resources are up and running at all times, similar to *ReplicaSet* from Kubernetes definition.
- **MachineDeployment** acknowledges changes to the *Machine* resources, by providing a rolling-out strategy between *MachineSets* configurations. A MachineDeployment orchestrates deployments over a pool of *MachineSets*.
- **MachineTemplate** corresponds to worker nodes that can be controlled and configured separately using the *AWSMachineTemplate* that follows the same in *MachineDeployment* specification.

A particularity encountered in AWS EC2 and vSphere environments is the *KubeadmControlPlane*, the equivalent of the *AWSManagedControlPlane* in AWS EKS which is accountable for the initialization of the control plane in the workload clusters.

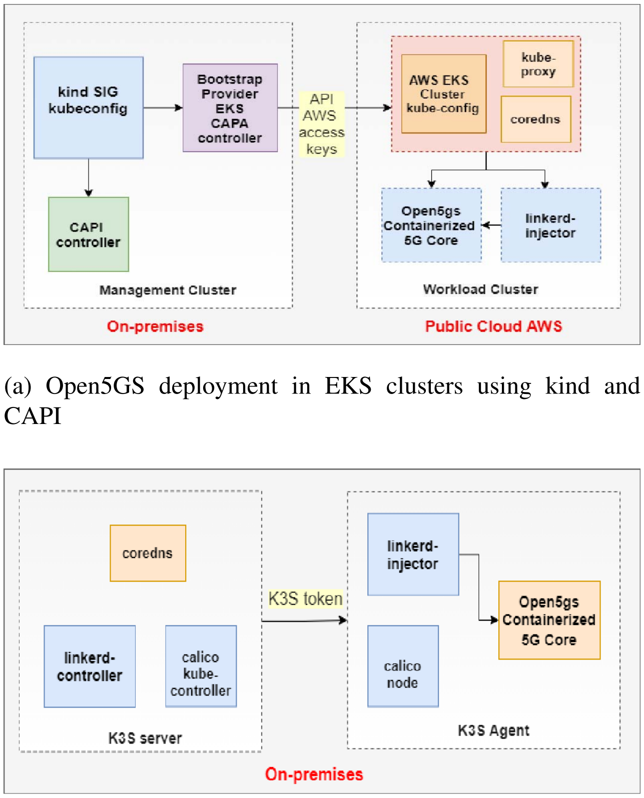
IV. Architecture and Setup Evaluation

In this Section, we compare the orchestration of the 5G packet containerized core by employing two of the Kubernetes orchestrators at the edge tools we discussed in the previous sections: *K3s* and *CAPI* based on the main cluster provisioning with *kind*. Both of our setups include the full stack of Open5GS deployed using Helm [60] which is a packet manager for Kubernetes. Moreover, we assess the two deployments in terms of response time and memory consumption based on the underlying infrastructure, on-premises vs public cloud.

A. Experimental setup - deploying the 5G NextGen core in Kubernetes clusters

In the first testbed (see Figure 4(a)), we deploy the management cluster and install *CAPI* in an on-premises environment running on VMware vCenter in a VM with 8 GB RAM and 4 CPUs. We use the *AWS bootstrap provider CAPA for EKS, i.e., CAPA-EKS*, thus the workload cluster was created on AWS public cloud and running in an EKS cluster, as well as the functions applications corresponding to the 5G packet core. In this scenario there are two different Kubernetes cluster-specific configurations: the *kind SIG kubeconfig* generated with *kind* that corresponds to the management cluster and a second configuration, the *kube-config* for the EKS cluster where the *workload cluster* resides. The CRDs and templates specific to the *CAPA-EKS* including the properties of the EKS control plane, the IAM roles and networking resourses are defined using an AWS provisioning template, i.e., *CloudFormation* stack. The communication between the management cluster and the workload cluster is ensured via *API AWS access keys*.

The second testbed (see Figure 4(b)) is running in an on-premises VMware environment that consists of a *K3s* cluster composed of a *K3s server* acting as a master node and a *K3s agent* as a worker node. The *K3s server* is running on a VM with 8 GB RAM and 4 CPUs, whereas we allocated 4 GB RAM and 4 CPUs for the VM's hardware specifications on which the *K3s agent* is running. In the *K3s* deployment, the worker node joins the *K3s server* through a *K3s token* created on the master node. The 5G network functions deployed using Open5GS are running on the *K3s agent* node. We



(a) Open5GS deployment in EKS clusters using kind and CAPI

(b) Open5GS deployment using K3S

Fig. 4: Testbed for 5G Core SBA deployed with Kubernetes orchestrators at the edge

deployed the Open5GS in a separate namespace to provide isolation from the rest of the workloads and additionally we run Calico [61], an SDN solution for containers, to enable strong isolation through declarative policies. Regarding the networking overlay, *K3s* comes by default with *flannel* [62], an open-source Container Networking Interface (CNI) and VXLAN as the default backend. Thus, we replaced the default CNI with *Calico* and installed the *calico kube-controller* on the *K3s server*, while the *calico node* is running on the *K3s agent*.

In Figure 4, both setups have in common a component called *linkerd-injector*. This is part of *Linkerd* [63], a popular **service mesh** tool deployed on Kubernetes cluster that allows and monitors communication between services and routes the traffic and the API calls between services/endpoints. The service mesh layer deployed on top of Kubernetes infrastructure role is to provide abstraction of the application business logic and mainly ensure service monitoring and observability. Another advantage is that it integrates with several monitoring and tracing tools (i.e., Prometheus [64] and Grafana) to allow network discovery, tracing and visualization between services, traffic flow or API latencies.

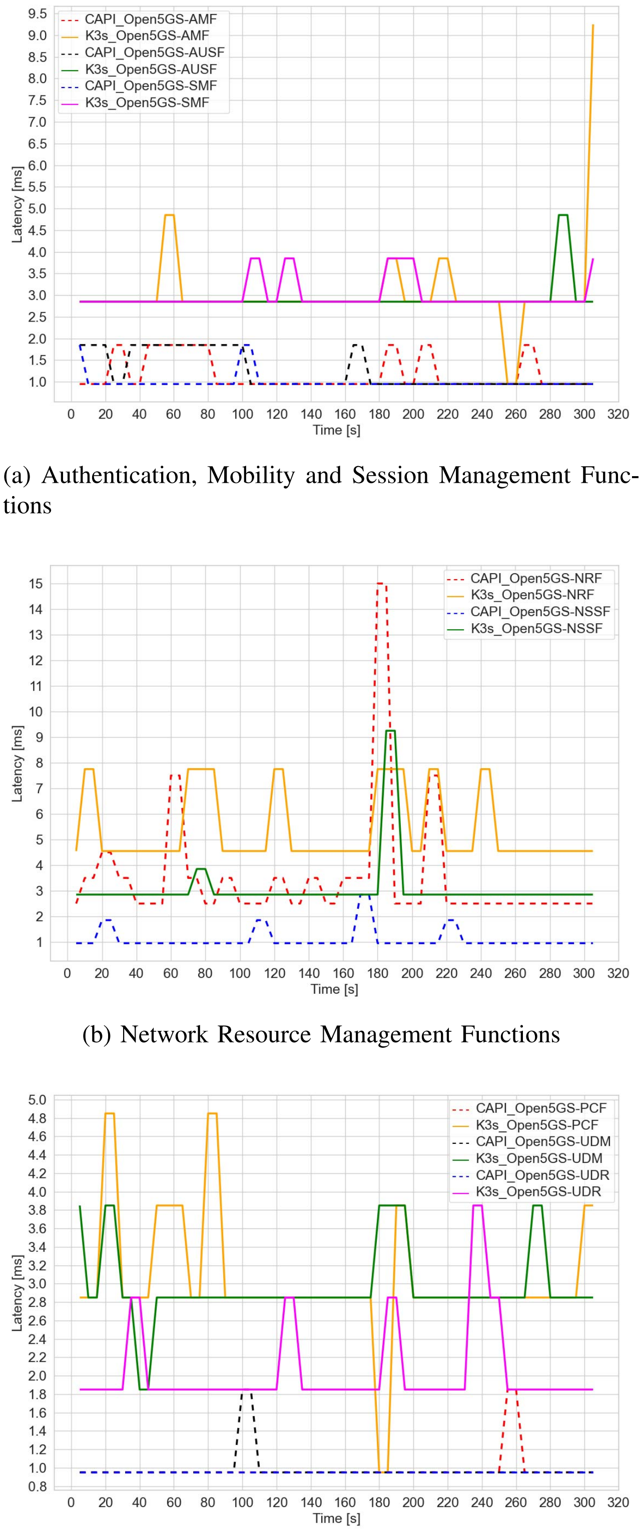
The Open5GS components mostly rely on HTTP/2 protocol to communicate, hence we used the ”inject” capability to add the *Linkerd* data plane proxy to the 5G application corresponding pods. Since the User Plane Function (UPF) is the only network function that uses GPRS Tunnelling Protocol (GTP-U), it will not have a corresponding *linkerd proxy* [65]. The UPF is responsible for the user plane packet routing and forwarding and interconnection with the Data Networking (DN).

B. Setup evaluation

Linkerd comes with a series of observability features for telemetry and monitoring. One metric specific to HTTP is the **response time latency**. This refers to the time it takes an application to perform an operation (e.g., processing a request, populating data, etc.) In service mesh terms, this is determined at the response level by measuring the amount of time the server takes to process each HTTP request. Latency is characterized by the percentiles of a distribution, commonly including the *p50* (or median), the *p95* (or 95th percentile), the *p99* (or 99th percentile) and it is expressed in milliseconds [63]. For example, a request latency *p95* value of 50 ms indicates that 95 out of 100 requests took 50 ms or less to process. The optimal values for end-to-end latency approved by 3GPP Release 16 [66] in the case of mMTC applications are under 50 ms (e.g. industrial IoT), whereas for critical URLLC applications (augmented/virtual reality, autonomous driving, etc.) can be in the range of 5-10 ms ([67]–[69]).

Figure 5 shows the latency, i.e., the *p95* response time for each of the deployed Open5GS network functions in the *CAPI* and *K3s* setups. In the *CAPI* deployments, the Open5GS applications are running in an *EKS workload cluster* hosted on AWS public cloud, whereas in the *K3s* deployment the *K3s agent* is hosted in an on-premises environment (see Figure 4). Nevertheless, to determine the end-to-end latency as defined by 3GPP Release 16 it is necessary to connect to the end-user through the RAN using a gNB and UE simulator (e.g UERANSIM [70]). This is currently not in the scope of this paper as we plan to address it in a later study. The main objective of this research is to evaluate the performance of the Open5GS containerized application inter process communication using application programming interfaces (APIs) for both our *CAPI* and *K3s* proposed deployments. Our results show better response time values for the *CAPI* setup compared to *K3s* which validates our assumption in regards to the benefit of running Open5GS in the public cloud since employing on-demand resources not only increases scalability but also leads to a faster inter-services communication.

In Figure 5 (a), we compare the latency values for three network functions: Access and Mobility Management Function (AMF), Authentication Server Function (AUSF) and the Session Management Function (SMF). The AMF is the function responsible for the authentication, connection and mobility management between the network and mobile device, whereas the AUSF performs the user authentication. The SMF handles the session management, IP address allocation, and control of policy enforcement. As a matter of fact, the latency for all



(a) Authentication, Mobility and Session Management Functions

(b) Network Resource Management Functions

(c) Policy Control and Data Management Functions

Fig. 5: Latency comparison of the Open5GS deployment using CAPI vs K3s

three network functions where Open5GS was deployed using *K3s* is $\sim$ 4-5 ms that translates into almost twice the response time obtained in the *CAPI* setup.

In Figure 5 (b), we display the values for the Network Resource Management Functions i.e., Network Repository Function (NRF) and the Network Slice Selection Function (NSSF). According to 3GPP Release 16, the NSSF's role is to select the Network Slice Instance (NSI) dedicated to the user. NSI is responsible for providing the hardware resources (e.g. computation, storage and networking resources) and the network functions for a network slice. The NRF performs network discovery and maintains a list with the available NFs along with the associated profiles. In terms of values, the NSSF deployed in the CAPI environment reports half the latency $\leq$ 2 ms in comparison to the *K3s* deployment while the response time for NRF in the case of CAPI slightly exceeds for a short interval the corresponding response time for *K3s* deployment. Nonetheless, this burst of latency does not affect the overall latency obtained for the NRF as AWS guarantees 99% service availability according to the SLA [71].

The latency values for the Policy Control and Data Management Functions are illustrated in Figure 5 (c). The Policy Control Function (PCF) objective is to ensure policy rules to the control plane functions while providing access to the subscription policies required to take decisions in in a Unified Data Repository (UDR). The latter stores the subscription information and is responsible to retrieve the structured data that can be exposed to a network function. The Unified Data Management (UDM) can also use the UDR to store and extract subscription information while its primary role is to generate authentication credentials and handle the user identification. We observe in our experiment that the latency values for PCF, UDR and UDM are significantly smaller ($\leq$ 2 ms) in the case of the Open5GS provisioned with *CAPI* in comparison to the *K3s* setup.

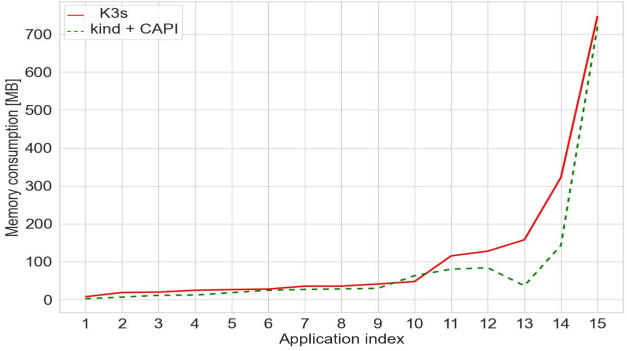


Fig. 6: Memory consumption for K3s server and CAPI Management Cluster deployed with kind

From a control plane resource consumption perspective, we evaluate the two setups in terms of memory utilized in the container runtime for both the *management cluster* in CAPI and the *K3s server*. Figure 6 shows the amount of RAM used per application instead of calculated per process and it is

Index	*K3s* applications	*kind* applications
1	docker-containerd	docker-proxy
2	local-path-provisoner	containerd
3	dockerd-current	kindnetd
4	kube-controllers	local-path-provisoner
5	metrics-server	kube-proxy
6	coredns	linkerd-controller
7	linkerd-controller	capi-controller
8	calico-controller	kube-scheduler
9	traefik	coredns
10	metrics-api	cainjector
11	grafana-server	kubelet
12	linkerd-proxy	kube-controller-manager
13	containerd	dockerd
14	prometheus	etcd
15	K3s-server	kube-apiserver

TABLE II: Application index corresponding to top most 15 intensive resource consumption control plane applications in both deployments - K3s vs kind

considered as the sum of **private** RAM and the **shared** RAM program processes. Table II illustrates the corresponding index of the top 15 most intensive control plane applications for both the *K3s* deployment and *kind*, respectively. For instance, the total amount of RAM required by *kindnetd* which is the default CNI plugin for *kind* (application number 3) is 11.5 MB, whereas in the case of K3s *calico kube-controller* it consumes 35.8 MB RAM. Nevertheless, for the *coredns* service which is the Kubernetes cluster DNS, memory consumption is similar, i.e., 30 MB. The two setups use *containerd* as the Kubernetes Container Runtime Interface (CRI). We can see in Figure 6 (application 11) that the *containerd* memory utilization in the case of *K3s* reaches 158 MB, almost twice the needed memory for *kind*, around 84 MB. The highest memory consumption value corresponds to the *kube-apiserver* 712.5 MB (in *Kind* deployment), whereas for *K3s*, the memory used for the *k3s-server* is 747.7 MB. An application required only by the *Kind* setup and specific to Kubernetes is *kubelet* which consumes 77.8 MB, whereas in the *K3s* deployment we installed *linkerd* that requires 150 MB. Even though the machines that host the two workloads have similar hardware specifications, our evaluation shows higher memory consumption rate for the *K3s* control plane deployment. These results indicate that even though *K3s* is considered a lightweight container orchestrator, it utilizes more memory in comparison to *kind*. On the other hand, the latency values are higher in the scenario where Open5GS is deployed using *K3s*. Hence, we can draw the conclusion that running 5G network functions on the edge is limited by the hardware requirements which are utilized on-demand in a cloud infrastructure.

V. Conclusions and Future Work

In this paper, we proposed a practical use case implementation of the 5G SBA in the context of edge computing for multi-cluster deployments and integration with third-party cloud providers. This paper aimed to provide an insight into the CNF open-source projects as we conducted a survey of the most relevant frameworks for the MANO ochestrator as well as a summary of the existing OSS projects for virtualized 5G packet core. We also evaluated the different container orchestration models designed for edge computing.

One of our main objectives was to compare two of the deployment models from a latency perspective when running the 5G CNFs and the memory resource consumption for the control plane. In this manner we were able to validate that the CAPI deployment using *kind* is a lighter edge container orchestrator than the other candidate. Moreover, the cloud-side deployment with centralized management on the edge shows better response time in comparison to the *K3s* edge deployment of the Open5GS. Another benefit of employing the *CAPI* solution is that it allows multiple geographically distributed clusters running on third-party cloud providers infrastructure. This translates into deploying edge services and network functions on different clusters spread across multiple clouds in order to ensure full isolation between the tenants.

As future work we are planning to test the communication between the 5G RAN and the 5G Core for the proposed deployment as well as the publish and subscribe capabilities for the IoT sensors. We would like to ensure that the proposed solution accommodates all scenarios for both telcos and IoT edge. Another item we want to assess in our future research is the security of our proposed setup from a cloud compliance management posture. In this paper, we deployed a *service mesh* within the cluster and analyzed the inter-service communications for the Open5GS deployment, thus we would like to further extend our work and evaluate the external traffic for multi-cluster and multi-cloud intensive operations according to our proposed multi-cloud architecture.

References

[1] S. Redana, Bulakci, C. Mannweiler, L. Gallo, A. Kousaridas, D. Navrátil, A. Tzanakaki, J. Gutiérrez, H. Karl, P. Hasselmeyer, A. Gavras, S. Parker, and E. Mutafungwa, “5G PPP Architecture Working Group - View on 5G Architecture, Version 3.0,” Jun. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3265031

[2] ETSI GS. ”Network Function Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework, Dec., 2015. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-EVE/001_099/005/01.01.01_60/gs_nfv-eve005v010101p.pdf

[3] ETSI GS NFV-MAN 001. ”Network Functions Virtualisation (NFV); Management and Orchestration,” Dec., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[4] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, “NFV and SDN-Key Technology Enablers for 5G Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[5] ONAP. Open Network Automation Platform, Accessed: Apr. 2021. [Online]. Available: https://www.onap.org

[6] SONATA. SONATA NFV Platform, Accessed: Mar. 2021. [Online]. Available: https://www.sonata-nfv.eu

[7] OPNFV. Open Platform for NFV, Accessed: Mar. 2021. [Online]. Available: https://www.opnfv.org

[8] Cloudify. Cloudify, Accessed: Mar. 2021. [Online]. Available: https://cloudify.com

[9] Open Baton. Open Baton, Accessed: Mar. 2021. [Online]. Available: https://openbaton.github.io

[10] ETSI. Open Source MANO, Accessed: Mar. 2021. [Online]. Available: https://osm.etsi.org

[11] OpenStack. OpenStack Tacker, Accessed: Mar. 2021. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[12] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, p. 107516, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620311786

[13] J. Haavisto, M. Arif, L. Lovén, T. Leppänen, and J. Riekki, "Open-source RANs in Practice: an Over-The-Air Deployment for 5G MEC," in *2019 European Conference on Networks and Communications (EuCNC)*, 2019, pp. 495–500.

[14] CNF. Cloud Native Computing Foundation, Accessed: Mar. 2021. [Online]. Available: https://www.cncf.io/telecom-user-group/

[15] Kubernetes. Kubernetes, Accessed: Mar. 2021. [Online]. Available: https://kubernetes.io

[16] ETSI MEC ISG. "Mobile Edge Computing (MEC); Technical Requirements ETSI", 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/01.01.01_60/gs_MEC002v010101p.pdf

[17] H. Huang, W. Miao, G. Min, and C. Luo, "Mobile Edge Computing for the 5G Internet of Things," 05 2019, pp. 143–161.

[18] J. Haavisto, M. Arif, L. Lovén, T. Leppänen, and J. Riekki, "Open-source RANs in Practice: an Over-The-Air Deployment for 5G MEC," in *2019 European Conference on Networks and Communications (EuCNC)*, 2019, pp. 495–500.

[19] J. Okwuibe, J. Haavisto, E. Harjula, I. Ahmad, and M. Ylianttila, "Orchestrating Service Migration for Low Power MEC-Enabled IoT Devices," 05 2019.

[20] N. R. Y. L. Bilel Cherif, Pascal Berthou, *Testbed for Multi-access Edge Computing V2X applications prototyping and evaluation*, 2020.

[21] S. Imadali and A. Bousselmi, "Cloud Native 5G Virtual Network Functions: Design Principles and Use Cases," 11 2018, pp. 91–96.

[22] X. Sun and N. Ansari, "Edgeiot: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[23] AWS. AWS Wavelength, Accessed: Mar. 2021. [Online]. Available: https://aws.amazon.com/wavelength

[24] Microsoft. Azure for IoT, Accessed: Mar. 2021. [Online]. Available: https://azure.microsoft.com/en-us/overview/iot

[25] Google. Global Mobile Edge Cloud (GMEC), Accessed: Mar. 2021. [Online]. Available: https://cloud.google.com/blog/topics/inside-google-cloud/google-cloud-unveils-strategy-telecommunications-industry

[26] AWS. Open source mobile core network implementation on Amazon Elastic Kubernetes Service, Accessed: Mar. 2021. [Online]. Available: https://aws.amazon.com/blogs/opensource/open-source-mobile-core-network-implementation-on-amazon-elastic-kubernetes-service

[27] T. Goethals, B. Volckaert, and F. De Turck, "Fledge: Kubernetes compatible container orchestration on low-resource edge devices," 11 2019.

[28] A. L. Karim Manaouil, "Kubernetes and the edge?" Inria Rennes-Bretagne Altlantique, Rennes, Tech. Rep., 2020.

[29] OpenAir Software Alliance EUROCOMcorene t. OpenAirInterface, Accessed: Mar. 2021. [Online]. Available: https://openairinterface.org

[30] Cellular Network Infrastructure. Corenet, Accessed: Mar. 2021. [Online]. Available: https://github.com/mitshell/corenet

[31] openLTE. openLTE, Accessed: Mar. 2021. [Online]. Available: http://openlte.sourceforge.net

[32] free5GC. free5GC, Accessed: Mar. 2021. [Online]. Available: https://www.free5gc.org

[33] srsLTE. srsLTE, Accessed: Mar. 2021. [Online]. Available: https://www.srslte.com

[34] OpenStack. OpenStack, Accessed: Mar. 2021. [Online]. Available: https://wiki.openstack.org

[35] T. Dreibholz, "A 4G/5G Packet Core as VNF with Open Source MANO and OpenAirInterface," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020, pp. 1–3.

[36] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[37] X. Sun and N. Ansari, "EdgeIoT: Mobile Edge Computing for the Internet of Things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[38] Q. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W. Hwang, and Z. Ding, "A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art," *IEEE Access*, vol. 8, pp. 116 974–117 017, 2020.

[39] S. Li, L. Xu, and S. Zhao, "5G Internet of Things: A survey," *Journal of Industrial Information Integration*, vol. 10, 02 2018.

[40] NextEPC. NextEPC, Accessed: Mar. 2021. [Online]. Available: https://nextepc.org/

[41] J. Okwuibe, J. Haavisto, E. Harjula, I. Ahmad, and M. Ylianttila, "Orchestrating Service Migration for Low Power MEC-Enabled IoT Devices," 05 2019.

[42] Open5GS. Open source project of 5GC and EPC (Open5GS) (Release-16), Accessed: Mar. 2021. [Online]. Available: https://open5gs.org

[43] Intel. Open Network Edge Services Software (OpenNESS), Accessed: Mar. 2021. [Online]. Available: https://www.openness.org

[44] Docker. Docker Swarm, Accessed: Mar. 2021. [Online]. Available: https://docs.docker.com/engine/swarm

[45] Virtual Kubelet. Virtual Kubelet, Accessed: Mar. 2021. [Online]. Available: https://virtual-kubelet.io

[46] OpenVPN. OpenVPN, Accessed: Mar. 2021. [Online]. Available: https://openvpn.net

[47] L. Cominardi, "Enhanced connectivity in wireless, mobile programmable networks," Ph.D. dissertation, Universidad Carlos III de Madrid, 2019.

[48] Kubernetes. Kubernetes K8S, Accessed: Mar. 2021. [Online]. Available: https://kubernetes.io

[49] Kubernetes SIGs. Kind, Accessed: Mar. 2021. [Online]. Available: https://kind.sigs.k8s.io

[50] Kubernetes SIGs. ClusterAPI, Accessed: Mar. 2021. [Online]. Available: https://cluster-api.sigs.k8s.io

[51] CNCF. KubeEdge, Accessed: Mar. 2021. [Online]. Available: https://kubeedge.io/en

[52] Rancher. K3s, Accessed: Mar. 2021. [Online]. Available: https://k3s.io

[53] KubeEdge to the left, K3S to the right. ProgrammerSought, Oct., 2020. [Online]. [Online]. Available: https://www.programmersought.com/article/6858882868/

[54] Eclipse Foundation. Eclipse Mosquitto, Accessed: Mar. 2021. [Online]. Available: https://mosquitto.org

[55] Grafana. Grafana, Accessed: Mar. 2021. [Online]. Available: https://grafana.com

[56] Kubernetes SIGs. Kubernetes Cluster Federation (KubeFed) , Accessed: Mar. 2021. [Online]. Available: https://github.com/kubernetes-sigs/kubefed

[57] G. Chandra, "Multi-Cloud and Multi-Cluster Declarative Kubernetes Cluster Creation and Management with Cluster API (CAPI — v1alpha3)," ITNEXT, Jul 24 2020. [Online]. [Online]. Available: https://itnext.io/multi-cloud-and-multi-cluster-declarative-kubernetes-cluster-creation-and-management-with-cluster-6df8efdc2a89

[58] R. Case. Introducing EKS support in Cluster API. Waveworks, Oct., 2020. [Online]. Available: https://www.weave.works/blog/introducing-eks-support-in-cluster-api

[59] Kubernetes SIGs. ClusterAPI Provider AWS, Accessed: Mar. 2021. [Online]. Available: https://github.com/kubernetes-sigs/cluster-api-provider-aws

[60] Helm. Helm, Accessed: Mar. 2021. [Online]. Available: https://helm.sh

[61] Project Calico. Calico, Accessed: Mar. 2021. [Online]. Available: https://www.projectcalico.org

[62] Flannel. Flannel, Accessed: Mar. 2021. [Online]. Available: https://github.com/flannel-io/flannel

[63] Linkerd. Linkerd, Accessed: Mar. 2021. [Online]. Available: https://linkerd.io

[64] Prometheus. Prometheus, accessed: Apr. 2021.

[65] G. Chandra, "Opensource 5G Core With Service Mesh," Gitconnected, Mar. 21, 2021. [Online]. [Online]. Available: https://levelup.gitconnected.com/opensource-5g-core-with-service-mesh-bba4ded044fa

[66] ETSI GS. 5G;System architecture for the 5G System (5GS) (3GPP TS 23.501 version 16.6.0 Release 16)," Oct., 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf

[67] S. R. Pokhrel, J. Ding, J. Park, O.-S. Park, and J. Choi, "Towards enabling critical mmtc: A review of urllc within mmtc," 2020.

[68] M. Siddiqi, X. Yu, and Joung, "5G Ultra-Reliable Low-Latency Communication Implementation Challenges and Operational Issues with IoT Devices," *Electronics*, vol. 8, p. 981, 09 2019.

[69] G. Americas, "New Services Applications with 5G Ultra-Reliable Low Latency Communications," 5G Americas Whitepaper, Tech. Rep., 2019.

[70] UERANSIM 5G Solutions. UERANSIM, accessed: Apr. 2021.

[71] AWS. Amazon Compute Service Level Agreement, accessed: Apr. 2021.