An Initial Approach to a Multi-access Edge Computing Reference Architecture Implementation Using Kubernetes

Ignacio D. Martínez-Casanueva(✉), Luis Bellido, Carlos M. Lentisco, and David Fernández

Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain
i.dominguezm@alumnos.upm.com

Abstract. The increasing demand of data and real-time analysis has given rise to edge computing, providing benefits such as low latency, efficient bandwidth usage, fine-grained location tracking, or task offloading. Edge computing based on containers brings additional benefits, facilitating the development and deployment of scalable applications adapting to changing market demands. But in order to enable edge computing in the telco industry, it is important that current standardization efforts are followed by software platforms implementing those standards. This paper proposes an approach to the design and implementation of an edge computing platform based on Kubernetes and Helm providing functional blocks and APIs as defined by ETSI in the Multi-Access Edge Computing (MEC) reference architecture. Although this proposal is still at a work-in-progress state, this paper describes the design and implementation of an open-source proof-of-concept scenario focusing on the lifecycle management of cloud native MEC applications. The resulting prototype shows the feasibility of this approach, that can be adequate to create a lightweight MEC demonstration platform for university laboratories and experimentation.

Keywords: Edge computing · Application virtualization · Platform virtualization

1 Introduction

Edge computing technologies, by moving cloud computing resources closer to the end users, bring many benefits such as low latency, efficient bandwidth usage, fine-grained location tracking, or task offloading. Edge computing based on containers rather than on virtual machines will help develop and deploy scalable applications that can adapt dynamically to meet changing market demands. But besides the computing resources to run applications, edge computing relies on management and orchestration services to manage the application lifecycle and orchestrate different services.

ETSI has already began the standardization work for Multi-access Edge Computing (MEC). This paper presents an initial approach to implementing the MEC reference architecture defined by ETSI, using a container-based computing infrastructure. We focus on the Kubernetes platform because it provides scaling, scheduling and fault-tolerance features that can be leveraged to address the lifecycle management of cloud native MEC applications. But platforms providing edge computing management and orchestration services for container-based edge computing, such as Akraino or Airship, are still not aligned with the ETSI standards.

The objective of this work is to analyze how Kubernetes can support MEC services following a clean slate approach, in which first the ETSI standards are analyzed, to then map the reference architecture to a design based on Kubernetes and supported by other open-source components. In the long term, we think this approach can result in a reference platform that can be simpler to deploy than the current edge computing platforms for university laboratories and experimentation.

The remainder of the paper is organized as follows. Section 2 provides an overview of the MEC reference architecture, an introduction of Kubernetes and Helm as container management tools, and a discussion of related work. Section 3 describes the HelmMEC proposal, a middleware that allows the deployment of containerized MEC applications on Kubernetes following the ETSI standard specifications. Section 4 provides a description of a proof-of-concept showing the feasibility of this proposal. Finally, conclusions are summarized in Sect. 5.

2 Background

In this section, ETSI standardization of the Multi-Access Edge Computing (MEC) paradigm is first introduced along with the MEC reference architecture. Then, the new concept of cloud native applications is presented as an approach that can be embraced to build applications within the scope of MEC. Then, the main technologies used as a basis of the proposal are introduced: Kubernetes as a container management solution and Helm as a package manager for cloud native applications. Finally, a brief discussion of related work is provided.

2.1 Multi-access Edge Computing (MEC)

Edge computing is a new paradigm that moves cloud resources closer to the end users. Due to the proximity of the consumers to the applications, the edge computing technology brings many benefits such as low latency, an efficient bandwidth usage, fine-grained location tracking, or task offloading [1]. ETSI standardized the Multi-access Edge Computing (MEC) term and it is currently working on providing specifications of a reference architecture and the different APIs used by its components. ETSI ISG MEC defines a reference architecture [2] for MEC systems as depicted in Fig. 1, with two levels: MEC system level and MEC host level.

The MEC system level is composed of the following functional blocks: Operations Support System (OSS), MEC Orchestrator (MEO) and User Application Life Cycle Management Proxy (UALCMP). The MEO is the core component in the MEC system

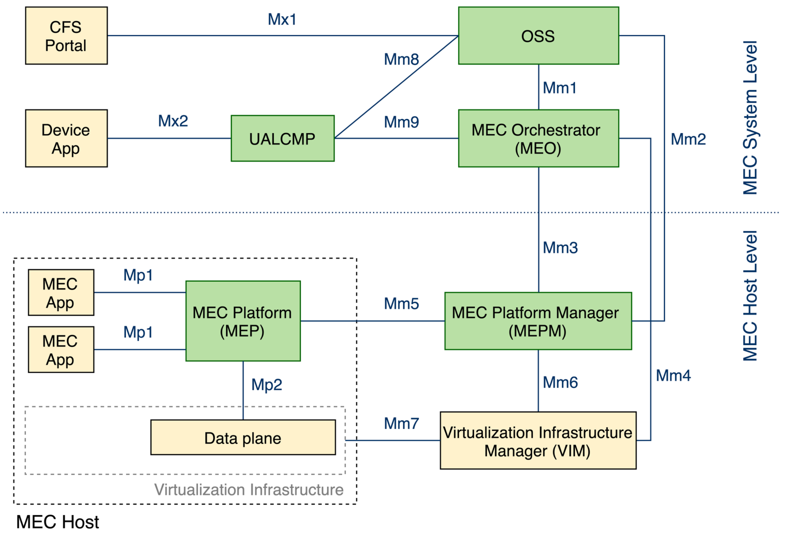

Fig. 1. ETSI MEC reference architecture

since it is responsible for keeping track of the provisioned MEC hosts, the available resources as well as MEC services. The MEO is also responsible for the management of MEC application packages, and the scheduling, relocation, and instantiation/termination of MEC applications deployed across the controlled MEC hosts. Customers or external device applications can interact with the MEO through the OSS, which is responsible for granting and forwarding requests to the MEO. Eventually, MEO forwards these requests to the MEPM, which realizes them by interacting with the target MEP and the VIM.

The MEC host level is composed of: MEC Platform Manager (MEPM), Virtualization Infrastructure Manager (VIM) and MEC host. The MEC host contains a virtualization infrastructure that supports the instantiation of MEC applications. These MEC applications are registered and configured by the MEC Platform (MEP) with the purpose of exposing and/or consuming services. The MEC applications may interact with each other and external users via a configurable data plane on the MEC host.

2.2 Cloud Native Applications

Due to the benefits that MEC provides, MEC technology opens an ecosystem for new and innovative applications that need to quickly adapt to changing market demands. In this respect, ETSI promotes the use of DevOps practices to deliver applications for MEC [3]. Development of cloud native applications has risen as one of the approaches followed by DevOps engineers.

Cloud native [4, 5] is a set of modern cloud techniques which aim at building scalable, dynamic applications than can quickly adapt to changing demands. Cloud native applications embrace microservices-based architectures rather than the common monolithic approach. The separation into independent services facilitates the development of applications as each microservice can evolve at a different pace. The package isolation and portability that containers provide makes them the perfect choice for virtualization as developers can just focus on implementing the business logic rather worrying about the underlying infrastructure. But building and deploying complex containerized applications requires container orchestration, which can be provided by open source orchestrators such as Docker Swarm [6] or Kubernetes (K8s) [7]. In this paper, we focus on the Kubernetes platform because it provides scaling, scheduling or fault-tolerance features that can be leveraged to address the lifecycle management of cloud native MEC applications.

2.3 Helm

Helm [8] is an open source project that enables application package management for cloud native applications that run on Kubernetes. Helm bundles the different Kubernetes resources of an application (e.g., Pods, ConfigMaps, Services, etc.) into a Helm chart. Helm charts are stored in a separate repository, and each chart may have different versions available.

Using Helm, application lifecycle management is made possible by registering releases of Helm charts in the Kubernetes datastore. Thus, when a user triggers a basic management task from Helm, e.g., *instantiate*, Helm combines the previous, current, and desired states of the application and applies changes in Kubernetes accordingly. Helm operates with the Helm client which realizes charts as one-time operations against Kubernetes. Upon issuing an operation over a given chart, the Helm client computes changes and makes sure the resulting requests of resources to the Kubernetes API are accepted. But, once this operation is finished, the Helm client no longer intervenes in the lifecycle management of the chart.

Flux Helm Operator [9] is an open source project that provides an implementation of a Kubernetes operator [10] for Helm. This project extends Kubernetes by defining the Helm chart as a new Custom Resource Definition (CRD) and implements an operator that runs Helm logic while watching for changes on Helm chart objects. Therefore, Flux Helm Operator can be used as the basis of an application lifecycle management solution for MEC applications based on Helm charts.

2.4 Related Work

Several open source platforms have been developed to provide the benefits of MEC computing technologies in different industrial and entertainment areas. Akraino [11] is an open source project of the Linux Foundation that implements MEC by using both OpenStack and Kubernetes as a VIM. Airship [12] is a project originally conceived for automatically deploying cloud infrastructures such as OpenStack. By means of Helm charts, Airship automates the installation of OpenStack services such as Neutron or Nova. But Airship can also be used to deploy MEC applications. However, the technical documentation provided by these open source projects does not specify how the MEPM and the MEO functions can be implemented in practice. This paper provides an initial approach to implementing these functions using Helm and Kubernetes.

ETSI has focused for some time in an alternative approach to provide MEC. ETSI has analyzed how the management and orchestration frameworks for MEC and for NFV, which share many similarities, can be integrated into a unified MECinNFV architecture [13]. From this perspective, MEC applications can be deployed as VNFs over existing NFV platforms such as Open Source Mano (OSM) [14]. However, it is still not possible to deploy a MEC application over an NFV platform as it is defined in the standards. MEC applications are defined in a different manner than virtual network functions and the functions performed by the MEO and the MEPM differ from the functions that the NFVO and the VNFM entities can carry out. Taking into account these limitations, Schiller et al. developed a generic VNFM based on JuJu, that can also play the role of the MEPM [15]. However, in this paper we discuss how Kubernetes can work not only as a VIM, but also as the core of the MEPM. Section 3 explains that Kubernetes natively provides lifecycle management, scheduling, fault, or configuration features that make it an intent-driven orchestrator that meets the needs of the MEPM.

3 Cloud Native MEC Applications Managed by Helm

Kubernetes main limitation for working as a MEPM block is that it is not aware of the concept of a MEC application. Besides, identifying what exactly is a MEC application becomes cumbersome due to Kubernetes's nature. Hence, to fulfill the desired role of the MEPM, an additional functional block that translates from MEC application management to Kubernetes resource management is required. Helm offers this functionality by mapping Helm charts with MEC applications. Flux Helm Operator project provides a mechanism that can be utilized to manage the lifecycle of a MEC application while defining it in a declarative way through Kubernetes. As a result, Helm Operator works as a generic MEPM for MEC applications that are declared as Helm charts.

The declaration of MEC applications in Kubernetes provides the means for an implementation of the MEPM's interfaces specified by ETSI in the reference architecture (see Fig. 2). ETSI GS MEC 010 [16] provides REST API specifications for the Mm1 and Mm3 reference points. The Mm1.AppPkgm and Mm3.AppPkgm APIs focus on MEC application package management whereas Mm1.AppLcm and Mm3.AppLcm focus on MEC application lifecycle management. Additionally, ETSI defines Mm3.AppLccn APIs at the MEPM for notifying the MEO about changes on the status of an application instance that are related to lifecycle management operations. In this paper we propose an implementation of the MEPM via a new layer that supports Mm3.AppPkgm, Mm3.AppLcm and Mm3.AppLccn APIs. This layer receives the name of HelmMEC.

3.1 HelmMEC

Figure 3 shows an overview of the proposed MEPM design, including the K8s Control Plane and the three main components of HelmMEC: HelmMEC proxy, Flux Helm Operator, and NoSQL database.

The HelmMEC proxy is a web server that implements the standardized REST APIs at the Mm3 reference point. It is responsible for handling requests through the Mm3 reference point and translating them into interactions with the K8s Control Plane via the K8s API.

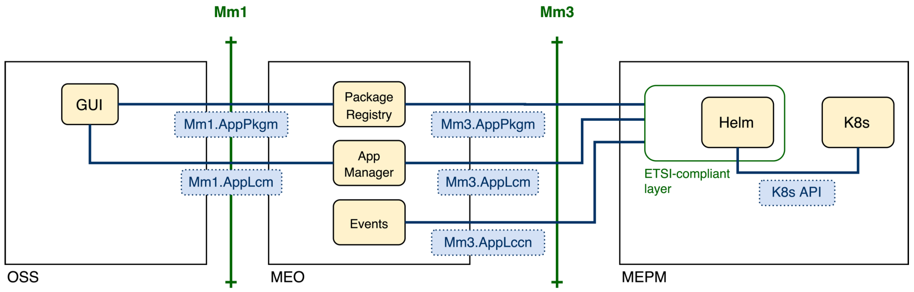

Fig. 2. REST APIs specified by ETSI for Mm1 and Mm3 reference points

The HelmMEC proxy uses a NoSQL database to store all the information related to MEC application instances. This information is necessary to run operations to create, delete, instantiate, and terminate MEC application instances. Each application instance has an appDId that is used by the HelmMEC proxy to identify which application package, i.e., Helm chart, is used by the application. The MEC application package information is managed at the MEO where the package appDId is associated with Helm chart-specific details.

The Helm Operator is the core component of HelmMEC as it provides the functionalities of a generic manager of Helm-based applications. Helm Operator achieves this by adding the HelmRelease CRD in Kubernetes, which will allow a declarative management of Helm-based applications via K8s API. The HelmMEC proxy interacts with the Helm Operator by configuring HelmRelease resources in Kubernetes. These configurations are the result of translating requests coming from the Mm3 REST APIs.

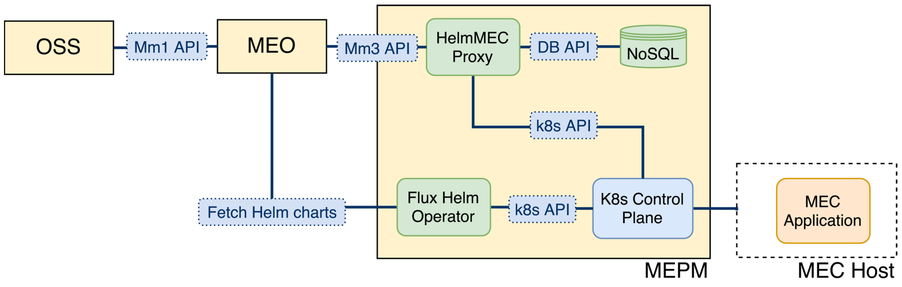

Fig. 3. HelmMEC internal components

As a result, the OSS can request lifecycle management operations on MEC application instances that will eventually trigger configurations in the Helm Operator. For example, in the case of instantiating a MEC application instance: the HelmMEC proxy creates a new HelmRelease in Kubernetes out from the package information of the application; the Helm Operator fetches the corresponding Helm chart information from the MEO, and ultimately orders Kubernetes the creation of the resources that compose the MEC application.

4 Proof-of-Concept Validation

In this section we introduce the prototype that has been developed to validate HelmMEC. The Virtual Network over linuX (VNX) [17] virtualization tool has been used to simulate a scenario that resembles a MEC system as depicted in Fig. 4. The installation and configuration of all the components has been automated with Ansible playbooks. The code of the prototype is available in our GitHub repository [18].

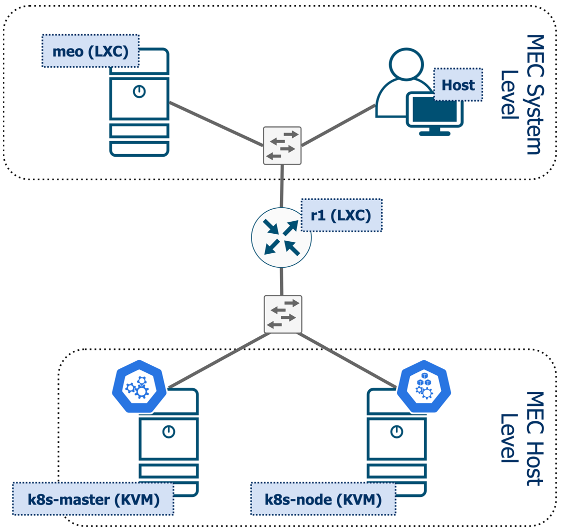

Fig. 4. MEC system prototype based on Kubernetes

At the MEC host level, a Kubernetes cluster is provisioned on two KVM virtual machines running Ubuntu 18.04 LTS image. The K8s cluster is configured using open source project Kubespray [19]. Kubespray is a collection of Ansible playbooks that enable installing production-ready Kubernetes clusters that can be easily customized with a variety of configuration parameters specified from Ansible. The HelmMEC components run as containerized services on the Kubernetes cluster. The HelmMEC proxy is based on a web server programmed with FastAPI framework [20] that partially implements the Mm3.AppLcm interface. An instance of MongoDB is deployed as the NoSQL database that stores application instance-related information. Flux Helm Operator runs as another Kubernetes service following Flux's guidelines.

An emulated gateway router (r1) provides connectivity between the MEC host level and the MEC system level. The router has been implemented by setting up a Linux Container (LXC) that runs Ubuntu 18.04 LTS. In addition, the BGP daemon BIRD [21] has been configured to BGP-peer with the Kubernetes cluster, thus enabling dynamic routing to services running in Kubernetes.

At the top of Fig. 4, the MEC system level components are displayed. This level contains the MEO which is deployed as a Linux Container (LXC) that runs an Ubuntu 18.04 image. The MEO is configured with a web server programmed using FastAPI web framework [20]. This web server acts as a MEC application package manager that partially implements the Mm1.AppPkgm interface. It relies on MongoDB as its NoSQL database for storing application package-related information. Additionally, the user's host machine is attached to the same network as the MEO, enabling users to interact with the MEC system as the OSS component.

As a result, the prototype allows users to request the MEO to onboard new Helm charts as MEC application packages, as well as directly request the MEPM to instantiate MEC applications from a chosen Helm chart.

5 Conclusions

We presented the HelmMEC middleware that leverages Kubernetes and Helm to realize MEPM functions. Helm provides package management features that fit the needs for cloud native MEC applications. By running Helm as a Kubernetes Operator, a promotion of generic applications as first-class objects in Kubernetes is achieved. This enables benefiting from Kubernetes lifecycle management capabilities for whole generic applications. HelmMEC proposal is aligned with ETSI reference architecture for MEC. The declarative definition of applications through Kubernetes, facilitates the integration with ETSI-specified APIs for the MEPM functional block.

The proof-of-concept presented in this work shows the feasibility of implementing a MEC system aligned with the reference architecture that is based on Kubernetes playing the roles of VIM and MEPM.

As a future work, the prototype will be extended to implement the remaining APIs for the reference points Mm1 and Mm3 as specified by ETSI. In addition, an elaborated association between MEC application packages and Helm charts should be addressed. The prototype will be a key element of initial experiments to support MEC based services on campus networks in the context of our research group projects.

Acknowledgement. This work was supported in part by the Spanish Ministry of Science and Innovation in the context of the ECTICS project (PID2019-105257RB-C21) and by the Spanish Ministry of Science, Innovation and Universities in the context of the Go2Edge project (RED2018–102585-T).

References

1. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing - a key technology towards 5G. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf. Accessed 30 Sept 2020
2. ETSI: Multi-access Edge Computing (MEC); Framework and Reference Architecture. ETSI GS MEC 003 V1.1.1 (2019)
3. Sabella, D., et al.: Developing software for multi-access edge computing. https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp20ed2_MEC_SoftwareDevelopment.pdf. Accessed 30 Sept 2020
4. Microsoft - Defining Cloud Native. https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition. Accessed 22 Sept 2020

5. Richardson, A.: What is cloud native and why does it exist? https://www.cncf.io/webinars/what-is-cloud-native-and-why-does-it-exist/. Accessed 24 Sept 2020
6. Docker Swarm mode overview. https://docs.docker.com/engine/swarm. Accessed 22 Sept 2020
7. Production-Grade Container Orchestration – Kubernetes. https://kubernetes.io. Accessed 08 June 2020
8. Helm. https://helm.sh/. Accessed 08 June 2020
9. Flux Helm Operator. https://docs.fluxcd.io/projects/helm-operator/en/stable/. Accessed 08 June 2020
10. Operator pattern – Kubernetes. https://kubernetes.io/docs/concepts/extend-kubernetes/operator/. Accessed 08 June 2020
11. Akraino. https://www.lfedge.org/projects/akraino. Accessed 22 Sept 2020
12. Airship: Automated OpenStack Deployment for Open Source Infrastructure. https://www.airshipit.org. Accessed 22 Sept 2020
13. ETSI: Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NFV environment. ETSI GR MEC 017 V1.1.1 (2018)
14. Open Source Mano. https://osm.etsi.org. Accessed 22 Sept 2020
15. Schiller, E., Nikaein, N., Kalogeiton, E., Gasparyan, M., Braun, T.: CDS-MEC: NFV/SDN-based Application Management for MEC in 5G Systems. Comput. Networks. **135**, 96–107 (2018). https://doi.org/10.1016/j.comnet.2018.02.013
16. ETSI: Multi-access Edge Computing (MEC); MEC Management; Part 2: Application lifecycle, rules and requirements management. ETSI GS MEC 010-2 V2.1.1 (2019)
17. Virtual Networks over linuX (VNX). https://web.dit.upm.es/vnxwiki/index.php/Main_Page. Accessed 08 June 2020
18. vnx-mec-k8s: VNX scenario that deploys a K8s-based MEC system. https://github.com/giros-dit/vnx-mec-k8s. Accessed 17 Sept 2020
19. Kubespray - Deploy a Production Ready Kubernetes Cluster. https://kubespray.io. Accessed 17 Sept 2020
20. FastAPI framework, high performance, easy to learn, fast to code, ready for production. https://fastapi.tiangolo.com. Accessed 21 Sept 2020
21. The BIRD Internet Routing Daemon Project. https://bird.network.cz. Accessed 22 Sept 2020