Implementing Multi-Access Edge Computing with Kubernetes

Vojdan Kjorveziroski[1*][0000-0003-0419-4300], Cristina Bernad Canto[2][0000-0001-9537-415X], Katja Gilly[2][0000-0002-8985-0639], and Sonja Filiposka[1][0000-0003-0034-2855]

[1] Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University (Skopje), Skopje, North Macedonia
{vojdan.kjorveziroski,sonja.filiposka}@finki.ukim.mk
[2] Department of Computer Engineering, Miguel Hernández University (Elche), Alicante, Spain
{cbernad,katya}@umh.es

Abstract. The introduction of novel telecommunication standards such as 5G and beyond, which offer increased throughput and more efficient network communication, serve as enablers for new technologies. One of these technologies is Multi-Access Edge Computing (MEC), with the potential to revolutionize existing computing architectures and their feature set as we know them today. By collocating compute nodes with mobile networking equipment, customers are offered reduced latency and increased privacy, compared to alternative scenarios where the traffic is indiscriminately routed to the cloud. However, the distributed nature of the mobile landscape, coupled with the huge number of involved parties, requires careful consideration and standardization before any rollout. A number of reference documents have been published in recent years with the aim of standardizing the communication interfaces of MEC. We discuss these standardization efforts and provide a description of a MEC architecture centered around the Kubernetes container orchestrator, replacing the concept of virtualized MEC applications with containerized instances which can be deployed as serverless functions. We also offer a use-case scenario leveraging the described component mapping, showcasing the scalability and load-balancing features of the proposed architecture.

Keywords: Multi-Access Edge Computing, Architecture Mapping, Kubernetes, Serverless Computing, 5G.

1 Introduction

Since the introduction of the cloud computing paradigm [1], this new concept of centralized and virtually unlimited computing capacity has had a major impact across the industry, as well as in people's everyday lives. Its initial success has served as an enabler for the development of new technologies, such as machine learning, augmented reality, virtual reality, and sensor networks [2]. However, the meteoric rise in the number of both industrial and personal internet of things (IoT) devices [3], combined with the increasing desire for latency sensitive and bandwidth intensive applications

by end-users, has introduced challenges to this architecture. One solution aimed at overcoming the network latency constraints stemming from the geographically distributed nature of the cloud is to introduce data preprocessing closer to the source devices – at the edge of the network [4]. This would not only reduce communication delays but would also decrease the traffic volume to the cloud, thus easing network congestion. With the introduction of such a multi-layer approach to computing, global cloud capacity would be utilized for high-performance computations and data archival, while leveraging the more resource constrained edge for simpler, yet more sensitive computation in terms of response time and privacy.

Even though edge computing has attracted a noticeable research interest, it is still a broad term and the question of where exactly to place the computing infrastructure remains. A number of commercial entities have developed edge sites, dedicated data centers in areas with large numbers of users [5]–[7]. Other strategies include the deployment of resource limited, general purpose devices which can process data from nearby sensors either independently or as a part of a cluster [8]. However, one of the more prominent initiatives today is the introduction of multi-access edge computing (MEC), previously known as mobile edge computing [9]. The main idea is to collocate compute infrastructure at mobile network sites together with the communications equipment, thus potentially solving both the network connectivity and data processing issues at the same time. The introduction of improved and faster communication protocols under the 3GPP umbrella, such as 5G and the resulting packetization of mobile networks, only contribute to the popularity of this approach.

In an effort to alleviate the fragmentation surrounding all new and emerging technologies, such as MEC itself, the European Telecommunications Standards Institute (ETSI) has published a reference MEC framework, which has been further supported by a reference architecture as well [10]. The goal of this paper is to map this reference architecture, which is primarily focused on virtualization, to a container based one, centered around the Kubernetes container orchestrator, while also offering an option to use serverless functions as the runtime environment for the deployed MEC applications. The main contributions of this work are: 1) Evaluation of containers and serverless functions as feasible runtime environments for MEC applications; 2) Mapping the components of the Kubernetes container orchestrator, along with help of additional software extensions to the reference ETSI MEC architecture, describing options for the deployment of serverless MEC applications. 3) Discussion of current initiatives within the Kubernetes community to tackle the edge of the network as a new computing frontier.

The rest of this paper is organized as follows: in section 2 we outline recently published work relevant to the topic of multi-access edge computing and the application of containers at the network edge. We then continue with section 3, where we first give a description of Kubernetes extensions for the network edge, before discussing the mapping of the entities presented in the reference ETSI architecture to the various components of the Kubernetes orchestrator. In section 4 we discuss the implications of the presented choices, as well as offer a reference use-case leveraging the architecture presented in the previous section. We conclude the paper with section 5, outlining open issues and directions for future research.

2 Related Work

The pioneering work in the multi-access edge computing landscape is done by the ETSI MEC industry specification group (ISG). A number of documents have been published so far, encompassing different aspects, ranging from use-case descriptions to technical API specifications. [11] presents a list of required and optional features that a MEC platform should support, augmented with a list of use-cases which make use of them. To aid real-world implementations and ensure cross-compatibility, [12] offers a technical API specification which should be implemented by the various entities present in the reference architecture.

Even though the initial focus of the initiative has been around virtualization and the use of virtual machines as runtime environments for MEC applications, the ISG has also published a document [13] exploring other options, many of which are more lightweight in terms of resource consumption and initial start up time. Several alternatives have been proposed, such as the use of unikernels [2] or Kata containers which provide better workload isolation using a hardware virtualization technology [9], [14], [15]. The serverless paradigm has also been identified as a potentially viable solution for MEC [16], with the first implementations already being available [17], [18].

Efforts to reuse existing products also need to be noted. [19] extends the concept of a MEC application to a Kubernetes environment, describing ways for easy deployment, ensuring cross-compatibility and easy sharing of already developed applications. The authors of [20] have also identified the Kubernetes orchestrator as a viable option for MEC and develop federated scheduling algorithms that can take into account individual application requirements and existing network conditions. While it is true that we also focus on Kubernetes, the goal of this paper is to map the existing and readily available Kubernetes components and extensions to the MEC reference architecture, without introducing any custom modifications.

Finally, taking into account the large number of ongoing initiatives, it is important to ensure efficient load-balancing even across disparate compliant infrastructures. [21] explores different mechanisms for request routing, many of which are centered around the use of the domain name system (DNS) protocol. The same strategy of achieving location transparency in a distributed edge infrastructure is also described in [17] and [22].

3 Architecture Mapping

The published MEC architecture by the ETSI ISG describes a hierarchical system comprised of two levels – the System Level and the Host Level. Even though an additional third level is present in the reference framework – Network, it is not described in the architecture. When it comes to Kubernetes, there are multiple ways in which its components can be deployed in an edge environment, with various implications to performance, network latency, and scalability. In order to reflect the reference architecture as closely as possible, we have chosen to use a hierarchy of federated Kuber-

netes clusters [20] at the edge, controlled by a central control plane using the KubeFed [23] solution. Alternative approaches include the use of a single flat cluster with distributed nodes, where an overlay network would be deployed on top of the existing telecom infrastructure to support inter-node communication. However, this would not only increase deployment complexity, but would also negatively impact the network performance, due to a lack of hierarchy and the need for continuous communication with the control nodes. The KubeEdge initiative [24] is yet another option, albeit it is currently focused on dealing directly with end-devices and acquiring data from their sensors, instead of general purpose MEC applications. It is the most lightweight approach of the three, but the lack of hierarchy and the limited number of workloads deployable on the edge nodes restrict its usage in a native MEC environment.

In the sections that follow we map the Kubernetes components to the entities present in the two levels of the reference architecture, beginning with the System Level. Closely following this discussion, Fig. 1 presents a visual representation of the relationship between the components and mapped entities.

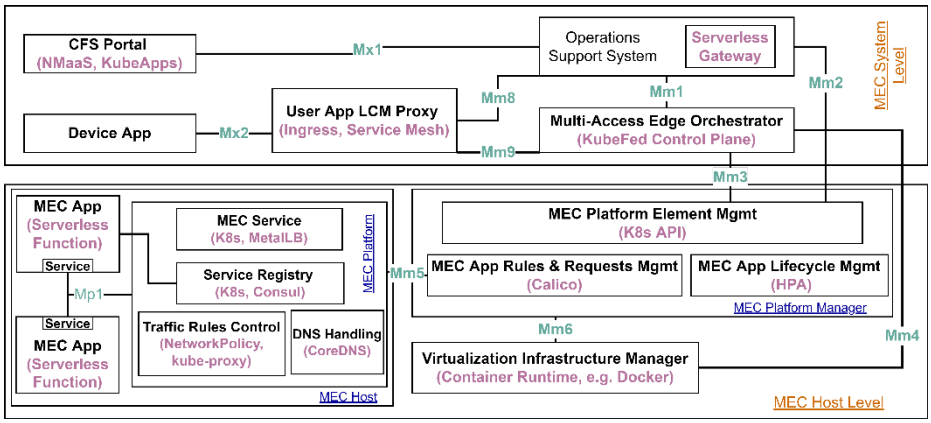

Fig. 1. Proposed mapping of software components to the reference MEC architecture

3.1 Multi-Access Edge Computing System Level

The MEC System Level is comprised of 5 entities. 4 of these are discussed below, while the Device App, representing client applications on end-user devices has been omitted due to it not being relevant to the Kubernetes architecture.

Multi-Access Edge Orchestrator (MEO). The MEO is the core component at the MEC System Level. It is represented by a central, highly available Kubernetes cluster which is hosted on dedicated infrastructure and has connectivity to all edge locations. Using the Mm3 reference point, it functions as a coordinator of the KubeFed federated remote clusters. The MEO, through its central role, maintains an overall view of the MEC system, including the current status of the remote clusters, deployed applica-

tions, services, overall topology, and available resources. Since it has direct control over the remote clusters, it can also take part in the on-boarding of new applications, set rules on where they should be deployed, the number of replicas that should be created across different locations, and any additional requirements that must be fulfilled in terms of available hardware or performance. The built-in KubeFed scheduler can schedule applications in a number of ways [25]: a) Equally distributing the replicas across all available federated clusters; b) Ratio-based scheduling, where a given percentage of replicas is located on one remote cluster, with the remainder on another; c) Label-based scheduling, taking into account remote cluster characteristics, for example availability of specific hardware, geographical location, or current conditions; d) Relocation between nodes, in cases of limited capacity – an application migration can be automatically triggered if lack of resources is detected in the existing environment. The migration can reschedule the pod either on a different node at the same edge location (within the same Kubernetes cluster) or on another Kubernetes cluster part of the federation that has enough resources available.

It should be noted that the MEO does not take part in the running of the applications, instead it sets global application scaling preferences, and it is up to the individual MEC platform managers (federated Kubernetes clusters) to fulfill these requests according to the available resources on their end.

Customer Facing Service (CFS) Portal. The customer facing portal should facilitate easy MEC application deployment by end-users. One major advantage of adopting Kubernetes in a MEC environment is the large number of existing integrations, some of which are in the form of application marketplaces, allowing seamless deployment of new applications through interactions with the Kubernetes API. Examples include the NMaaS project [26] which allows easy deployment of network monitoring applications on top of a Kubernetes cluster using a web front-end resembling an app store, or Kubeapps [27], which compared to NMaaS, provides lower levels of abstraction. In the serverless camp, the need for serverless function marketplaces has been identified from the start, and a number of initiatives have been created as a result [28], [29]. All of these existing solutions can easily be made compliant to the Mx1 reference point, allowing users not only to consume, but also share MEC application either packaged as serverless functions, long-running containers or any other runtime environment managed by the Kubernetes API [30].

User App Lifecycle Management (LCM) Proxy. The LCM proxy is an optional component which facilitates relocation of MEC applications between the MEC system and external infrastructures. Kubernetes or its extensions currently do not offer a way of seamless application transfer to third-party infrastructures not part of the federation, but research targeting other platforms is available. For example, VideoPipe [31] allows the use of nearby devices to create flexible video processing pipelines by using cross-platform JavaScript serverless runtimes. Adapting such a solution to a Kubernetes environment would potentially unlock the option of flexible application instance transfer from third party equipment, even end-user devices, to the more per-

formant MEC infrastructure. Another responsibility of the LCM Proxy is the authorization of requests from device applications. To support this, an API gateway or a centralized Ingress can be deployed, routing requests based on additional information, for example one present in the headers, and then leveraging the Mm8 reference point, handing-off these requests to the serverless platform controller co-located in the Operations Support System. In cases where a Kubernetes service mesh is used or an in-cluster Ingress controller, the centralized control plane of the MEO can control their behavior using the Mm9 reference point.

Operations Support System (OSS). The OSS is tightly integrated with the other functionality offered by the telecom provider, including subscriber and mobility information, which can drive the deployment and scheduling decisions of the MEO. When a serverless platform is used, the OSS can include a serverless gateway component which would intercept requests with the aim of optimizing the required number of replicas or pre-warming containers and passing this information to the MEO via the Mm1 reference point.

3.2 Multi-Access Edge Computing Host Level

The MEC Host Level is represented by federated Kubernetes clusters deployed on the network operators' edge infrastructure. Each Kubernetes cluster is comprised of a separate control plane (MEC Platform Manager) represented by Kubernetes master nodes responsible for coordinating the multiple worker nodes taking the role of MEC hosts. Additional details about the three main entities from this level are discussed in the subsections below.

Multi-Access Edge Computing Platform Manager. The Platform Manager is represented by the control-plane of the separate Kubernetes clusters deployed at each edge location which are then federated with the MEO. In this context, the MEO represents the global control plane of the whole MEC infrastructure, while each Platform Manager acts as a localized control plane which has the nearby MEC hosts under its control. The master nodes of the edge clusters sync information with the master nodes in the MEO using the Mm3 reference point which in this case is represented by the KubeFed API.

Taking into account the potentially resource constrained nature of the edge infrastructure, different Kubernetes distributions can be used, depending on the requirements. Work has already been done to offer lightweight Kubernetes distributions specifically aimed at the network edge, such as MicroK8s, or K3s, which can be used in this context. No matter the chosen Kubernetes flavor, the application instantiation behavior is abstracted away, as they all support the same image format and are compliant with the original Kubernetes API. Using such a hierarchical approach with both a central control plane in the form of MEO and a localized control plane, it is possible to realize the requirements of application and environment configuration, together with performance information gathering to support the scheduling tasks of the MEO.

Combining the native Horizontal Pod Autoscaler (HPA) [32] built into Kubernetes with the workload relocation due to lack of resources supported by KubeFed would allow adaptive horizontal scaling as a result of current resource usage among different edge locations.

Virtualization Infrastructure Manager (VIM). The VIM can be best represented by the container runtime present on each Kubernetes node at the edge. While Kubernetes does support different container runtimes, it is common for all of them to be able to allocate, manage, and release system resources in coordination with the kernel of the underlying operating system. Furthermore, pulling, verifying, and instantiating containers from container images, no matter whether they represent traditional long-living applications or elastic serverless functions, is also the role of this component.

When it comes to performance measurements, for each container running locally on the nodes, metrics can be natively exposed, such as consumed memory, processing time, and network bandwidth. Additional application level or platform level information such as number of successful requests can be implemented using third-party components, such as the popular Prometheus monitoring system [33], which can also be integrated with the aforementioned Kubernetes HPA. On-the-fly checkpointing and migration of running serverless functions can be realized using container techniques such as the one presented in [34], thus optimizing compute time during long running blocking operations, or in cases when resources are exhausted on a given node. It needs to be noted though that such workload migration is an optional feature, as per the reference architecture.

Internal network access, within the same edge cluster, as well as external, to other federated clusters including the internet, is provided by the Calico container network interface (CNI) plugin. This is made possible by the calico-node agent, running on every edge host as a container, supported by the VIM.

Multi-Access Edge Computing Platform. The MEC platform is part of the MEC host entity present in the MEC host level. It is represented by Kubernetes worker nodes which are part of the local edge clusters and are controlled by the MEC Platform Manager (Kubernetes control plane). Using native Kubernetes functionality, coupled with additional addons, the MEC Platform is responsible for implementing traffic rules, handling DNS queries, as well as service exposure and service registration. Leveraging the calico-node agents running on each node, together with kube-proxy, it is possible for the MEC platform to implement specific traffic rules received from the platform manager. These rules instruct the local data plane how to forward traffic and how to expose services. In cases where Calico is used as the container network interface (CNI) plugin, it is also possible to apply rules across multiple clusters, using federated networking policies, a feature supported by the plugin.

Whereas CoreDNS is the recommended DNS resolver in Kubernetes clusters, in a MEC environment it can also be repurposed to serve as a recursive resolver for mobile network users which are connected to the network segment where the particular MEC host is placed. As previously, the initial configuration itself, as well as the vari-

ous zones are managed by the responsible MEC Platform Manager in their entirety. Optionally, global rules can be defined in coordination with the MEO. This approach can act as one part of a DNS-based load-balancing strategy for access to the deployed MEC applications, explained in more details below. In terms of accessing the MEC applications, services are used for their exposure, accepting incoming traffic, and routing it to the appropriate instance. Kubernetes has multiple service publishing options, depending on whether access should be allowed only from within the cluster or from external sources as well. This behavior can further be augmented by additional load-balancing plugins such as MetalLB or PureLB [35], aiding the service publishing process.

In terms of persistent storage volumes which are required by stateful MEC applications, a number of container storage interface (CSI) plugins [36] are supported by Kubernetes, allowing integration with local or even remote storage solutions where the communication would be facilitated by the CSI plugin. These storage volumes, once provisioned, can be offered to the respective applications via the Mp1 reference point. It must be noted though that the implementation of a comprehensive storage strategy is out of scope of the MEC reference architecture.

In summary, by using the Calico overlay network and its local components on each node, it is possible to facilitate inter-cluster communication. In this way, MEC applications can be published via services, but can also in turn act themselves as consumers of different MEC applications hosted both in the same cluster or on some other cluster part of the federation.

Finally, the question of how to efficiently route requests when the user is moving from one location to another is something that is widely researched. The idea is to offer seamless mobility, while keeping the latency to the currently consumed MEC application as low as possible. For this purpose, a varied strategy can be used: 1) use split DNS with response policy zone and leverage the CoreDNS instance at each edge site as a recursive resolver that will prioritize access to locally deployed instances of the MEC applications. This is also one of the strategies explored by the MEC ISG [21], and is already implemented in existing MEC solutions. However, in our case, this approach does not solve the problem in cases when the nearest cluster is overloaded, and requests need to be dynamically balanced across different clusters; 2) To offer inter-cluster balancing, a distributed service discovery solution can be devised which can take into account additional properties, such as network load and physical distance to the infrastructure where the application is hosted. While Kubernetes does offer native service discovery, it is not sufficient for such advanced cases, and an additional service discovery component needs to be integrated. For example, Consul [37] or a similar tool can be deployed to augment the service discovery efforts. Using a federated Consul deployment with a global view of the available MEC services, the stored metadata about each service can be leveraged while routing the requests, issuing temporary redirects to the best candidate fulfilling the requirements.

3.3 Multi-Access Edge Computing Applications

The MEC applications themselves can be represented in a variety of ways, such as virtual machines, long-living containers, or serverless functions. When it comes to a serverless deployment, the functions can either be deployed independently or in groups, making use of the container runtime available on each of the worker nodes for instantiating the functions. By leveraging the Kubernetes API and the underlying components of the federated clusters, the functions themselves can interact with the MEC platform. In this manner, they can indicate their availability, current load, or expose any other application-level metrics relevant for their lifecycle and for the scheduling of new instances. Each MEC application, can also indicate the required resources for its uninterrupted execution, and when not explicitly stated, these can be set to default values [38].

4 Discussion

Repurposing existing components and leveraging the Kubernetes container orchestrator as a centerpiece of a distributed MEC strategy, backed by additional well-established addons, offers easy implementation and the possibility of integration with third-party systems. The architecture mapping discussed previously, as well as the various other components that have been included can be better described by a use-case scenario.

Using the customer facing portal, users are presented with an application catalog from which they can choose their desired application to be instantiated in the form of serverless functions across the MEC infrastructure. In some instances, depending on policy, this catalog can also include other types of applications, for example backed by long-running containers or virtual machines. This request is then forwarded to the Operations Support System via the Mx1 reference point, and the OSS can either accept it or deny it, depending on customer information available in other telecom provider's systems. If accepted, the request is passed onto the MEC Orchestrator via the Mm1 reference point, which in turn triggers the application instantiation on the edge by instructing the MEC Platform Managers how to proceed with the deployment via Mm3. Once the application request is received by the Kubernetes control plane at edge location, it schedules the required pods according to the rules specified by the Platform Manager, in accordance to locally available resources. At this point, the worker nodes which have been selected fetch the required container images, and with the use of the local container runtime start the pods, registering the service endpoint in the service registry, before exposing it. The DNS zone is also updated, allowing the DNS resolver to include the new service address in the DNS responses for the particular application. Should the MEC application become viral, additional replicas can be automatically added using Kubernetes HPA policies based on CPU, memory usage, or any other metric interpreted by a third-party component, such as requests per second. Since all network users are using the nearest CoreDNS server as their recursive resolver, requests for the application are automatically load-balanced between the application replicas present within the MEC hosts. However, in cases where the resources

of the given cluster are exhausted, new replicas can be further instantiated on other nearby clusters that have spare capacity. In such cases, the request routing component with the help of the service registry and the associated metadata with each service can route the request to some other cluster part of the federation that conforms to the set routing strategy. Since the issued redirects are temporary in nature, the possibility to dynamically change the destinations remains.

Should an application migration system such as the one described in [31] is implemented, then the User App Lifecycle Management Proxy can moderate requests for on-boarding applications which are currently executed on external infrastructures, or offloading applications which are running at some MEC Host within the federation, in a containerized runtime environment.

5 Conclusion

Using a set of existing software components, we have mapped the reference multi-access edge computing architecture introduced by the ETSI MEC ISG to an architecture comprised of federated Kubernetes clusters. Utilizing the official federation solution for this container orchestrator, KubeFed, a hierarchy of clusters can be provisioned, conforming to the requirements set out in the published reference documents. Alternative cluster federation solutions exist today [39], [40], but KubeFed has allowed us to achieve our goal of only reusing existing and well-established software components, without any additional custom extensions. In this way a scalable MEC architecture can be built, which should be immediately familiar to anyone with container orchestration experience, thus minimizing the adoption effort for telecom providers.

We have also provided a reference use-case which outlines the functionality of the included software components and their role within the wider MEC architecture. Through the description of scaling and request routing methods, we have also tackled the reference communication points, such as those connecting to external entities (Mx), facilitating the MEC platform functionality (Mp), and allowing component management (Mm).

In the future, it would be interesting to see how the promising KubeEdge project progresses, allowing severely resource constrained nodes to extend the cluster to remote edge locations. In fact, a number of MEC initiatives and special interest groups related to popular infrastructure software have emerged in recent years [41], attracting a sizeable interest. To better map these different initiatives, the ETSI MEC ISG has published a matrix of existing implementations [42], along with a description of which entities and reference points are implemented by each of them.

Even though mobile-edge computing is a rather new concept and development continues driven primarily by enabling technologies such as 5G, it is encouraging that standardization efforts continue. These standardization efforts are essential in avoiding further walled gardens without any cross-platform interoperability, as is the case with many popular initiatives which have unfortunately outpaced their standardization efforts.

Acknowledgement. The work presented in this paper has received funding from the Faculty of Computer Science and Engineering under the "SCAP" project.

References

[1] M. Armbrust *et al.*, 'Above the Clouds: A Berkeley View of Cloud Computing', 2009, Accessed: May 22, 2022. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[2] N. Kratzke, 'A Brief History of Cloud Application Architectures', *Applied Sciences*, vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.

[3] 'Global IoT and non-IoT connections 2010-2025', *Statista*. https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/ (accessed Jan. 14, 2022).

[4] G. Carvalho, B. Cabral, V. Pereira, and J. Bernardino, 'Edge computing: current trends, research challenges and future directions', *Computing*, vol. 103, no. 5, pp. 993–1023, May 2021, doi: 10.1007/s00607-020-00896-5.

[5] 'The Cloudflare Global Network | Data Center Locations', *Cloudflare*. https://www.cloudflare.com/network/ (accessed May 21, 2022).

[6] 'Key Features of a Content Delivery Network| Performance, Security | Amazon CloudFront', *Amazon Web Services, Inc.* https://aws.amazon.com/cloudfront/features/ (accessed May 21, 2022).

[7] 'Azure Front Door edge locations by abbreviation'. https://docs.microsoft.com/en-us/azure/frontdoor/edge-locations-by-abbreviation (accessed Jan. 14, 2022).

[8] M. Gusev and S. Dustdar, 'Going Back to the Roots—The Evolution of Edge Computing, An IoT Perspective', *IEEE Internet Computing*, vol. 22, no. 2, pp. 5–15, Mar. 2018, doi: 10.1109/MIC.2018.022021657.

[9] G. Nencioni, R. G. Garroppo, and R. F. Olimid, '5G Multi-access Edge Computing: Security, Dependability, and Performance', *arXiv:2107.13374 [cs]*, Jul. 2021, Accessed: May 20, 2022. [Online]. Available: http://arxiv.org/abs/2107.13374

[10] 'Multi-access Edge Computing (MEC); Framework and Reference Architecture'. ETSI MEC ISG. Accessed: May 19, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.02.01_60/gs_MEC003v020201p.pdf

[11] 'Multi-access Edge Computing (MEC): Use Cases and Requirements'. ETSI MEC ISG. Accessed: May 19, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/02.01.01_60/gs_MEC002v020101p.pdf

[12] 'Multi-access Edge Computing (MEC); MEC Management: Application lifecycle, rules and requirements management'. ETSI MEC ISG. Accessed: May 19, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/01002/02.01.01_60/gs_mec01002v020101p.pdf

[13] ‘Multi-access Edge Computing (MEC): Study on MEC support for alternative virtualization technologies’. ETSI MEC ISG. Accessed: May 20, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/027/02.01.01_60/gr_MEC027v020101p.pdf

[14] V. Aggarwal and B. Thangaraju, ‘Performance Analysis of Virtualisation Technologies in NFV and Edge Deployments’, in *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Jul. 2020, pp. 1–5. doi: 10.1109/CONECCT50063.2020.9198367.

[15] R. Perez *et al.*, ‘A Performance Comparison of Virtualization Techniques to Deploy a 5G Monitoring Platform’, in *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, Jun. 2021, pp. 472–477. doi: 10.1109/EuCNC/6GSummit51104.2021.9482570.

[16] V. Kjorveziroski, S. Filiposka, and V. Trajkovik, ‘IoT Serverless Computing at the Edge: A Systematic Mapping Review’, *Computers*, vol. 10, no. 10, Art. no. 10, Oct. 2021, doi: 10.3390/computers10100130.

[17] S. Yang, K. Xu, L. Cui, Z. Ming, Z. Chen, and Z. Ming, ‘EBI-PAI: Towards An Efficient Edge-Based IoT Platform for Artificial Intelligence’, *IEEE Internet Things J.*, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.3019008.

[18] C. Cicconetti, M. Conti, A. Passarella, and D. Sabella, ‘Toward Distributed Computing Environments with Serverless Solutions in Edge Systems’, *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 40–46, Mar. 2020, doi: 10.1109/MCOM.001.1900498.

[19] I. D. Martínez-Casanueva, L. Bellido, C. M. Lentisco, and D. Fernández, ‘An Initial Approach to a Multi-access Edge Computing Reference Architecture Implementation Using Kubernetes’, in *Broadband Communications, Networks, and Systems*, Cham, 2021, pp. 185–193. doi: 10.1007/978-3-030-68737-3_13.

[20] D. K. Bainbridge and C. Corporation, ‘Implementing Multi-layer Infrastructure Management for Multi-Access Edge Computing (MEC) Services Using Kubernetes’, p. 23, 2021.

[21] M. Suzuki, T. Miyasaka, D. Purkayastha, Y. Fang, Q. Huang, and J. Zhu, ‘Enhanced DNS Support towards Distributed MEC Environment’. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi-wp39-Enhanced-DNS-Support-towards-Distributed-MEC-Environment.pdf

[22] L. Baresi, D. Filgueira Mendonça, and M. Garriga, ‘Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture’, in *Service-Oriented and Cloud Computing*, Cham, 2017, pp. 196–210. doi: 10.1007/978-3-319-67262-5_15.

[23] *Kubernetes Cluster Federation*. Kubernetes SIGs, 2022. Accessed: May 22, 2022. [Online]. Available: https://github.com/kubernetes-sigs/kubefed

[24] KubeEdge, ‘KubeEdge’, *KubeEdge*. https://kubeedge.io/en/ (accessed May 22, 2022).

[25] *Kubernetes Cluster Federation - ReplicaSchedulingPreference*. Kubernetes SIGs, 2022. Accessed: May 23, 2022. [Online]. Available: https://github.com/kubernetes-

sigs/kubefed/blob/c26fab53cffd7fee991cff2c6660297a9a9f93f1/docs/userguide.md

[26] 'NMaaS Home - GÉANT'. https://www.geant.org:443/Services/Connectivity_and_network/NMaaS (accessed May 23, 2022).

[27] 'Kubeapps, deploy your applications in Kubernetes'. https://kubeapps.com/ (accessed May 23, 2022).

[28] 'AWS Serverless Application Repository - Amazon Web Services', *Amazon Web Services, Inc.* https://aws.amazon.com/serverless/serverlessrepo/ (accessed May 22, 2021).

[29] *OpenFaaS Function Store*. OpenFaaS, 2021. Accessed: May 22, 2022. [Online]. Available: https://github.com/openfaas/store

[30] 'KubeVirt.io', *KubeVirt.io*. https://kubevirt.io// (accessed May 20, 2022).

[31] M. Salehe, Z. Hu, S. H. Mortazavi, I. Mohomed, and T. Capes, 'VideoPipe: Building Video Stream Processing Pipelines at the Edge', in *Proceedings of the 20th International Middleware Conference Industrial Track*, Davis CA USA, Dec. 2019, pp. 43–49. doi: 10.1145/3366626.3368131.

[32] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, 'Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration', *Sensors*, vol. 20, no. 16, Art. no. 16, Jan. 2020, doi: 10.3390/s20164621.

[33] 'Prometheus - Monitoring system & time series database'. https://prometheus.io/ (accessed May 22, 2021).

[34] P. Karhula, J. Janak, and H. Schulzrinne, 'Checkpointing and Migration of IoT Edge Functions', in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, New York, NY, USA, Mar. 2019, pp. 60–65. doi: 10.1145/3301418.3313947.

[35] V. Kjorveziroski, A. Mishev, and S. Filiposka, 'Evaluating IPv6 Support in Kubernetes', in *2021 29th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, Nov. 2021, pp. 1–4. doi: 10.1109/TELFOR52709.2021.9653276.

[36] 'Container Storage Interface (CSI) for Kubernetes GA', *Kubernetes*, Jan. 15, 2019. https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/ (accessed May 20, 2022).

[37] 'Consul by HashiCorp', *Consul by HashiCorp*. https://www.consul.io/docs/k8s (accessed May 23, 2022).

[38] 'Configure Default Memory Requests and Limits for a Namespace', *Kubernetes*. https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/ (accessed May 23, 2022).

[39] 'Multi-Cluster Kubernetes. Simplified. | Admiralty'. https://admiralty.io/ (accessed May 22, 2022).

[40] 'Pipeline · Banzai Cloud'. https://banzaicloud.com/products/pipeline/ (accessed May 23, 2022).

[41] 'KubeEdge@MEC: Combining the Kubernetes ecosystem with 5G', *Cloud Native Computing Foundation*. https://www.cncf.io/blog/2021/07/20/kubeedgemec-combining-the-kubernetes-ecosystem-with-5g/ (accessed May 22, 2022).

[42] ‘MEC Ecosystem - MECwiki’. https://mecwiki.etsi.org/index.php?title=MEC_Ecosystem (accessed May 22, 2022).