D. Roman, H. Song, K. Loupos, T. Krousarlis, A. Soylu, and A. F. Skarmeta: *The Computing Fleet: Managing Microservices-based Applications on the Computing Continuum*. New and Emerging Ideas (NEMI) track at 19th IEEE International Conference on Software Architecture (ICSA 2022). *To appear*. DOI: 10.1109/ICSA-C54293.2022.00015.

The Computing Fleet: Managing Microservices-based Applications on the Computing Continuum

Dumitru Roman, Hui Song
SINTEF AS
Oslo, Norway
{first.last}@sintef.no

Konstantinos Loupos, Thomas Krousarlis
Inlecom Innovation
Athens, Greece
{first.last}@inlecomsystems.com

Ahmet Soylu
Oslo Metropolitan University
Oslo, Norway
ahmetsoy@oslomet.no

Antonio F. Skarmeta
University of Murcia
Mucia, Spain
skarmeta@um.es

***Abstract*—In this paper we propose the concept of "Computing Fleet" as an abstract entity representing groups of heterogeneous, distributed, and dynamic infrastructure elements across the Computing Continuum (covering the Edge-Fog-Cloud computing paradigms). In the process of using fleets, stakeholders obtain the virtual resources from the fleet, deploy software applications to the fleet, and control the data flow, without worrying about what devices are used in the fleet, how they are connected, and when they may join and exit the fleet. We propose a three-layer reference architecture for the Computing Fleet capturing key elements for designing and operating fleets. We discuss key aspects related to the management of microservices-based applications on the Computing Fleet and propose an approach for deployment and orchestration of microservices-based applications on fleets. Furthermore, we present a software prototype as a preliminary evaluation of the Computing Fleet concept in a concrete Cloud-Edge scenario related to remote patients monitoring.**

Keywords—Computing Fleet, Architecture, Computing Continuum

I. Introduction

Cloud computing has been the dominating paradigm in recent years. A critical factor for its success is that it releases enterprises and service providers from directly handling on-premises computers individually and allows them to use the Cloud as a single entity. The increasing amount of data to process and security and critical response time requirements are dragging computation from the Cloud to the Edge devices closer to the data sources [1]. This forces computation tasks to be handled directly on the individual Edge devices. This is highly challenging as Edge devices are heterogeneous, spatially distributed, and move continuously. As a result, "the cost of deploying and managing an Edge computing environment can easily exceed the purpose financial benefits", as estimated by Gartner[1]. This raises the question: *Can we use many Edge devices, along with multi-cloud resources, as a holistic abstract entity to provide advanced computing capacity at smart device level (Edge), from Edge-to-Fog-to-Cloud, while being device independent and providing advanced concepts (ad-hoc clouds, time-triggering, decentralised intelligence, etc.)?* In Cloud computing, such a holistic entity is achieved based on computer clustering: "Cloud OS" (OpenStack, Kubernetes) integrates all computers in a data center as a single system (cluster) [2]. For external users, a Cloud-native application appears to run on a single computer, while internally, the Cloud OS ensures that the application fully leverages the existence of many computers for better performance. Following this direction, we believe the question above is answered by providing "the clusters for the Edge". However, the traditional cluster concept assumes that the computers are roughly homogeneous, connected by stable fast networks, and under a unified management. This assumption is no longer valid in Edge computing and today's IoT, and therefore, the existing cluster-oriented Cloud OSs do not work directly on the Edge.

We thus propose a novel concept, the *Computing Fleet*[2], as the "new cluster" in the era of Edge computing. Like a cluster, a fleet is an abstract set of infrastructure elements across the Computing Continuum, viewed as a single system, regardless of their nature (e.g., IoT devices, Edge servers, drones). Differently from cluster elements, *the elements in a fleet have unique profiles* – different processing capabilities, hardware capacities, energy requirements, network connectivity, associate to different data sources/consumers, and different domain roles. Moreover, *a fleet is dynamic*, with devices joining and exiting, and their profiles and topology constantly evolve. An Edge-native application runs on a fleet as a whole and can fully leverage the infrastructure elements across the whole continuum, considering their unique profiles, maximizing the performance, fulfilling the required tasks, and adapting to changes. Consequently, a fleet consists of *dynamic elements hired opportunistically* based on application requirements. To fully utilize the Computing Fleet, it is required that next generation OSs not only go beyond the state-of-the-art Edge management techniques (by looking at all devices as a single system), but also extend the current concept of OS with learning capabilities to handle the open infrastructure. The architecture of Computing Fleet OSs is an essential aspect that drives the implementation of such systems.

In this paper, besides introducing the concept of the Computing Fleet, we propose a reference architecture with three layers of key system services needed to realize the fleet concept (Section II). The reference architecture is meant to support software architects in designing OSs for the Computing Fleet, ultimately facilitating the development and deployment of applications on the Computing Fleet. Furthermore, we focus on the important aspect of application deployment on the Computing Fleet, where we propose an approach for the deployment and orchestration of microservices-based applications (Section III). A software prototype is introduced as a preliminary way to evaluate the Computing Fleet concept, with a focus on the deployment aspects (Section IV). Finally, we discuss related work, summarize the paper, and provide an outlook (Section V).

II. Reference Architecture for Fleet Computing

Like Cloud OSs that abstract many computers as a single cluster, a Computing Fleet OS is a software system that manages many devices and provides services for stakeholders

[1] https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders

[2] The term 'fleet' is inspired by the navy domain, where vessels of various size and capacity operate as a single whole and follow the global combat strategy, whilst each undertaking its own vessel-specific function.

to use the devices together as a single fleet. The challenge is hiding the complexity and dynamicity of devices from the stakeholders, while exploiting the different device profiles, connections, and contexts to optimize the resource usage. For this purpose, we propose an architecture composed of components wrapped as distributed *agents* running on Edge devices and *aggregators* on selected devices.

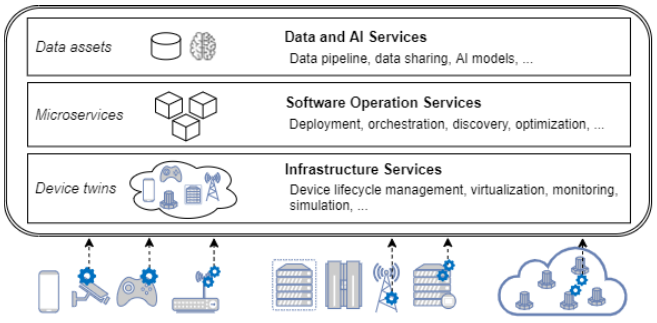
Data assets
Data and AI Services
Data pipeline, data sharing, AI models, ...
Microservices
Software Operation Services
Deployment, orchestration, discovery, optimization, ...
Device twins
Infrastructure Services
Device lifecycle management, virtualization, monitoring, simulation, ...

Figure 1. Computing Fleet Reference Architecture

The agents and aggregators are the physical components of the envisaged Computing Fleet OS, and they collaborate during runtime to provide fleet operating services in three layers, as shown in Figure 1:

- **Infrastructure Services layer**: Organizes the networked devices across the continuum as a fleet, through device registration and lifecycle management, network management, monitoring and simulation. This layer provisions virtualized resources to upper layers.
- **Software Operation Services layer**: Supports the operation of software applications on a fleet. This layer is in charge of the automatic orchestration and deployment of microservices-based applications, the discovery of services within and across applications, the federation of different fleets, and the optimization of service operation.
- **Data and AI Services layer**: This layer builds on top of the services to provide higher level support for building applications on data processing, AI, and digital twins.

Various stakeholders are expected to participate in such an ecosystem: infrastructure providers (own and maintain the devices and their connection and provide them as infrastructures to other stakeholders); service providers and application developers / DevOps (create and operate software applications in the software layer on top of the infrastructures); data scientists (manage, share, and utilize data and intelligence on top of the fleet infrastructures and applications).

<u>*Example Scenario*</u>. Figure 2 illustrates the proposed layers using a motivating example simplified from an eHealth system in the area of remote patient monitoring, where Edge devices / gateways are deployed at the patients' premises to collect patients' health-related data. The gateways collect data from local sensors and cameras and send the processed data to back-end services on the Cloud or local servers in nursing institutes. The DevOps team develops and operates a series of applications, such as the patient fall detection application.

In the infrastructure services layer, devices are combined into a private local "Cloud" (the fleet), which provides an overall picture of the whole system as device twins, provisions virtual resources from the fleet, and allows the high-level reconfiguration of the network.

In the application/software operation services layer, microservices, e.g., for video processing, are developed. The developers only need to deploy and/or update the entire application into the whole fleet, and the system automatically distributes the microservices into the devices, orchestrates them into a running application, and keeps adapting them in response to changes in the fleet (such as new devices enrolled), without intervention from the developers.

In the data and AI services layer, the data and AI assets such as the video stream and the ML function for fall detection are managed. This can include simplified creation, tuning and sharing of the data and the AI modules, together with the underlying microservices. It also provides the platform for setting up the federated learning to train the fall detection model in a distributed way.

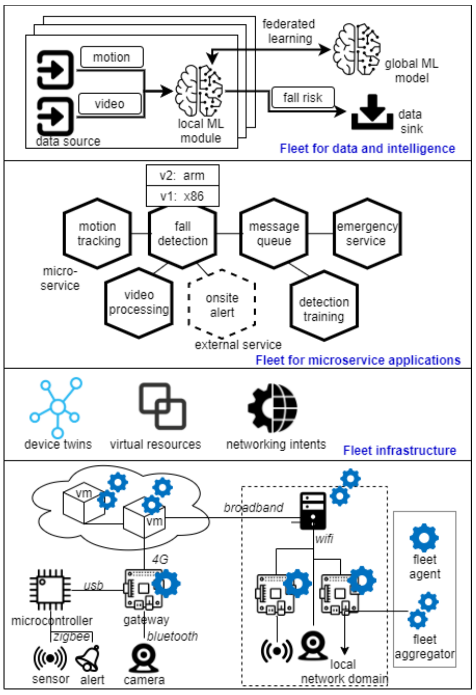

Figure 2. Example scenario for the Computing Fleet

In the next section, we dive deeper into the second layer and discuss various aspects and challenges related to software application deployment on the Computing Fleet. For the software application architects and developers of this eHealth application, the benefit of this layer is that they can focus on the design and development of application functions, without worrying about the heterogeneity and dynamicity of the Edge infrastructure underneath the application.

III. Microservice-based Applications on Fleets

Microservices architecture has become the predominant architectural style for developing Cloud applications, where each application is developed, deployed, and maintained as a collection of independent services. This architectural style is becoming a necessity in the context of Edge computing, where applications must make use of highly heterogenous devices. The Computing Fleet will have to support applications in the mainstream microservices architecture.

The example scenario in Figure 2 illustrates a sample application comprising of six microservices, as hexagons shown in the middle. Five are internal services (with solid borders), of which the application developers provide implementations. The *fall-detection* microservice has multiple implementations, with the same machine learning models optimized for ARM/x86 architectures, respectively. It uses an external microservice, *onsite-alert*, to intervene the patient when he risks falling (e.g., vibration on the smart watch).

Application developers should be able to deploy the entire application into the whole fleet instead of deploying each internal microservice into individual devices. Similarly, at runtime, an application always discovers the external service from the fleet without having to first decide from which devices, domains, or providers to find the services. Technically speaking, developers provide a global specification of the application (with all microservices and their connections) to the leading aggregator of the Computing Fleet, generally with the help of a CI/CD tool. When receiving the application specifications, the distributed fleet aggregators and agents work together to place the internal microservices into the proper devices: an aggregator decides which microservices should be deployed on each of its associated agents and marks this suggestion to the device twins. The agents regularly communicate to the aggregators to pull the deployment suggestions and then install the microservices accordingly. Such reverse control style ensures the scalability, flexibility, and correctness of deployment into distributed devices with a complex and unstable connection. After deployment, an internal service asks the nearest aggregator for the required external services, and the aggregator may collaborate with other aggregators to search for the service within the whole fleet. In this context, three main challenges need to be addressed: (i) How an agent *deploys* microservice implementations on its local devices, (ii) How the aggregators *orchestrate* multiple microservices to each associated devices, and (iii) How aggregators work together to *discover* external services from the whole fleet. The questions lead to a set of application-level fleet services, as illustrated in Figure 3.

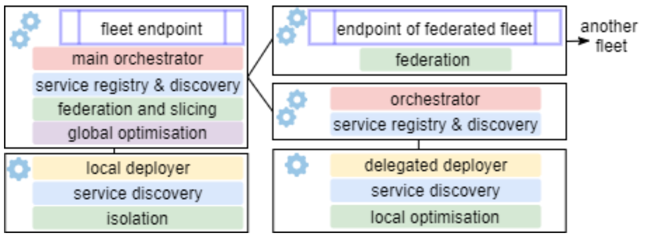

Figure 3. Fleet services for microservice applications

<u>Software deployment on local devices</u>: Fleet agents deploy the microservice implementations on the local devices. The key challenge is the existence of heterogeneous devices within a fleet. This can be approached towards a diversity-oriented Edge deployment using the Model-Driven Engineering method. It requires a new deployment modeling language that allows developers to specify the multiple functionally identical implementations for a microservice, together with the constraints and preferences of each variant, as part of the application specifications. The language can be based on existing deployment modeling languages (e.g., Kubernetes). Based on the deployment model, the deployment engines, distributed inside fleet agents, assess the context of itself and its neighboring devices to decide which variant to deploy, satisfy local constraints and approximate global goals, and maintain the fleet's diversity. Another challenge is how to support the devices with limited resources or Internet connections. This can be based on Kubernetes deployment model from containers to general artifact-platform pairs and implement a novel hierarchical deployment engine to use agents in powerful devices as delegates for the automatic deployment of microservices on the associated devices.

<u>Orchestration of internal microservices</u>: Some fleet aggregators contain orchestrators that work together to translate the design-time application specification into a runtime topology of service instances and infrastructure elements. At the core of this orchestration, we find the distributed knowledge base maintained by the mesh of orchestrators, including the device twins about the infrastructure and the current runtime topologies for existing applications. Based on the knowledge, the orchestrators search for the best runtime topology for the new application to meet multiple objectives such as global performance, service latency, and fault tolerance. This includes the proper distribution of computation among parallel devices (e.g., assigning a data processing logic to those gateways that connected to the relevant sensors) and the offloading of computation along far/near Edge and Cloud. The orchestrators need to continuously adjust the runtime service topology, or search for new ones, to adapt to the changes of the fleet itself or its environment. This will require novel techniques to reduce the searching time, especially when abrupt changes such as device mobility or failure incidents. The envisaged techniques include restoring and learning from previous searching and intermediate results, preserving alternative solutions obtained with the help of simulation, and nature-inspired decentralized optimization methods to approximate the optimal compositions. Novel data sharing mechanisms will also be needed to exchange knowledge and decisions for the resilience of the entire fleet despite potential failures.

<u>Discovery of external services</u>: Finding a reusable service in the Edge is more complicated than in the Cloud. In addition to functional matching and QoS, the context of services and hosting infrastructure must be considered and the different behaviors of service consumers. Two aspects are relevant in this context: (i) Novel organization of services description registration in the fleet. In theory, every aggregator should maintain the real-time knowledge of all the exposed services together with their contexts (device twins/orchestration status). We will need to approximate this setup at an affordable cost through a novel distributed architecture of service registry, a new mechanism for exchanging and synchronizing the service knowledge, and the continuous adaptation of the distribution of service knowledge to cope with fleet changes. (ii) Cognitive searching mechanisms: Historical data on network between service components, service demand, and user mobility will be used to create accurate prediction models for rapid discovery (reducing search space). The utility of the service discovery results will need to be monitored by calculating a weighted combination of solved requests, search precision, discovery latency, network overhead, resource usage. Learning models can be explored to increase the utility model.

<u>Fleet federation and slicing</u>: The deploy-orchestrate-discover trilogy from one fleet will need to be extended to multiple fleets, so that applications can be deployed across fleets (e.g., a fleet of homecare devices may need to be federated to a fleet of medical devices in the hospital to host applications that support the collaboration between the hospital and homecare service providers). A federated fleet from two fleets can be created by launching a new aggregator that connects to the two head aggregators from two existing fleets. Similarly, a sub-fleet can be created by launching a virtual aggregator connected to a subset of agents and aggregators. The new virtual aggregators can provide their endpoints to deploy applications on the federated or sliced fleets. One direction here would be to focus on the collaboration model between the aggregators, including knowledge sharing, orchestration, and discovery tasks. This

will require automatically changing the networks (routing, firewalls) according to the fleet federation and slicing.

Green meta-optimization: To solve potential conflicts between different optimization mechanisms, applications, and fleets, a meta-optimization module should be considered as a coordinator and arbiter. The meta-optimization introduces green aspects, such as energy consumption, into the overall picture. The core of the meta-optimizer is an extended knowledge base of available optimization approaches, together with their effects and impacts. Aiming at green objectives, the fleet knowledge base extends device and service profiles with power consumption, processing capacity, use of renewable energy sources, and others. It is built based on prior inputs and the history of application operation. The dynamic orchestration will then be able to consider the feedback from meta-optimizers when choosing the appropriate device to allocate and distribute the computational effort. Moreover, novel scheduling and green optimization techniques need to be introduced, such as resource allocation that balance computation load towards more power-efficient nodes and power management policies that put the devices in an eco-mode if needed. In addition to runtime application operation, architects and component suppliers can also use such meta-optimizers, e.g., providing references on the Lithium-Ion battery set up and the battery initial sizing and its re/charging operations, with the prediction of the workload and power consumption.

IV. Software Prototype and Preliminary Evaluation

To preliminary evaluate the Computing Fleet concept, we implemented an initial software prototype based on the reference architecture, focusing on the system services for fleet deployment and orchestration as outlined in the previous section. Following the concept of the Computing Fleet, a sample scenario using the prototype shows how a DevOps team can provide an entire application composed of microservices to a single fleet, without being concerned about how many devices exist, how to install software on different types of devices, and whether each device is online or not. To achieve this, the main challenge from the software deployment point of view is the *heterogeneity* and *dynamicity* of the fleet, which we address in this prototype. The prototype covers two layers of the reference architecture: in the infrastructure layer we reused the Azure IoT Hub for device registry and lifecycle management, by maintaining a device twin in the central Cloud for each Edge device; in the software operation layer, we developed two open-source tools for microservice orchestration and deployment:

- *Diversity-Aware Fleet-Oriented Microservice placement on Edge Devices (DivEnact)*[3]: It takes as input the microservices with alternative implementations and place them into different devices considering their contexts.
- *GeneSIS*[4]: Provides a generic way for deploying the microservices on different type of Edge devices, including those that do not have full-functioned OSs or direct internet connections.

We evaluate the fleet deployment prototype on the remote patient monitoring (RPM) system (introduced in Section II). The set-up includes 200 gateways (Raspberry Pi compatible devices), each of them connected to a set of medical sensors (via Bluetooth), and some of them connected to lower-end sensors or actuators (without Bluetooth support via microcontrollers). The gateways connect to backend services hosted in Azure via 4G. In this prototype, Edge devices are limited to gateways and microcontrollers that are used to connect sensors. Lower-end devices such as medical sensors are not included, since they do not allow deployment of general-purpose software. The testing scenario introduces a new access control mechanism into the RPM fleet. The scenario answers the following two questions:

1. *Can we deploy the application with alternative microservices on the fleet in a single action, without handling individually the heterogeneous devices in the fleet?*
2. *Can we handle the dynamicity of the fleet in an automatic way, so that developers do not need to follow up the change of the fleet after deployment?*

The scenario requires new hardware to be distributed and installed on the patient's side, i.e., a micro-controller board (Arduino Uno) connected to the RPM gateway via a USB port, and alternatively a remote Bluetooth-connected button that the patient must wear as a necklace. The software part includes three microservices: (M1) C code running on the Arduino board to listen to the remote button; (M2) Python code in Docker container to listen to button events via Bluetooth; and (M3) an updated version of gateway software to bind the button events with the authorization and authorization service in the Cloud.

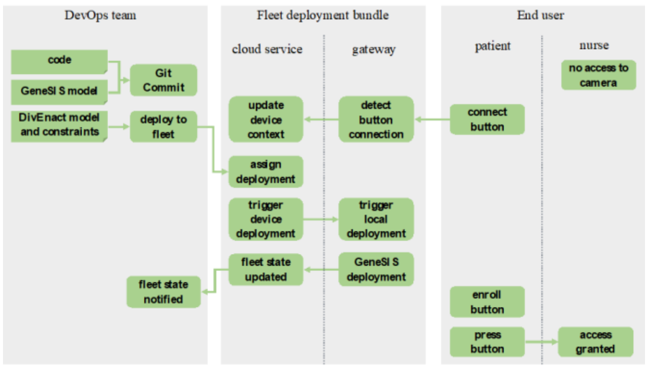

Figure 4. Fleet deployment of context-aware access control feature

Figure 4 shows how the fleet deployment bundle is used to realize this scenario. In the diagram, the corner-folded boxes represent the input artefacts, and the rounded rectangles are the activities. The arrows represent the causal links between activities, i.e., the target activities happen as a result of the source one. The vertical spatial relationship between activities roughly indicates their temporal relation, i.e., the activities placed higher in the diagram happens before the lower ones. The complete scenario can be seen in a screenshot video[5], and in the rest of this section, we briefly explain the main steps.

The DevOps team develops the code for the alternative microservice implementations, together with the deployment models in GeneSIS (for different types of the devices) and provides an application architecture model describing the relation between microservices and between microservices and the devices. Furthermore, they provide additional constraints that M1 and M2 should only be placed to gateways that are attached with Arduino buttons and Bluetooth buttons, respectively. For patients who did not choose to install any buttons, all the three services should not be deployed. Once

[3] https://gitlab.com/enact/divenact
[4] https://gitlab.com/enact/GeneSIS
[5] https://www.youtube.com/watch?v=a22YP4TB-To

the DevOps team release the code on GitHub, the fleet deployment action will be automatically triggered. At this moment, the developers do not need to know which patients have installed which types of buttons.

The deployment actions trigger the tools to perform actual fleet deployment work in two different places: the Cloud and the Edge gateways. The central Cloud service will first decide which gateways should be provisioned with the new software, based on the context of the devices. The assignment decision will be noted on the device twins of the relevant patients. The DivEnact agents on the gateways keeps checking the status of their device twins, and once a new deployment is updated in the device twin, the gateway will launch local deployment, by downloading the relevant deployment models, and utilize the GeneSIS engine to install the binary code or the container, depending on the implementation. The whole assignment and deployment process is automatic, without interaction from the developers, and therefore, the developers are shielded from the **heterogeneity of the fleet**.

Fleet assignment does not impact all the Edge devices at the same time: The DivEnact agent will keep monitoring the contexts of the device and update the device twin. The central DivEnact service will re-evaluate the assignment in a pre-defined interval (5 minutes in this case) and trigger device deployment if needed. From the patient's point of view, if they plug an Arduino button into their gateway, the microservices M1 and M3 will be automatically deployed in 2-7 minutes (local deployment takes in average 2 minutes in our experimental setup), and they can immediately start using the new access control feature, without waiting for the next deployment action triggered by the DevOps team. The whole process does not need interaction with the DevOps team, and therefore they are shielded from the **dynamicity of the fleet**.

V. Related work, Summary, and Outlook

Relevant related work for the Computing Fleet come primarily from the domain of Edge application management. Hao et al. [3] proposed a layered system architecture to organize many Edge devices to support applications, with the focus on workflows. Recent approaches on service placement in Fog and Edge [4] focus on the planning of optimal locations to run the application services, while simplifying the problem of deploying services to the devices. Automatic deployment support for heterogeneous IoT and Edge devices is rare, and in existing approaches (e.g., [5]), the coverage of devices with constrained resources or missing direct connection to the Internet is not addressed. They also lack the support of dynamicity of the Edge fleet. In summary, according to a recent survey [6], most of the existing approaches on managing Edge applications follow the direction of dividing applications into components (workflows or services) and assisting designer and developers in placing the components. The Computing Fleet concept follows the same direction, however, goes further to shield the architects and developers from the heterogeneity, distribution, and dynamicity of devices, and allows deployment of the entire application on a single entity, the fleet. Some Cloud vendors provide management support for a large set of IoT and Edge devices as a single fleet, such as the Azure IoT Edge [7] and AWS Greengrass [8]. These manage the dynamicity of the devices by tracking the lifecycle of devices and maintaining their runtime contexts. However, currently these commercial management services for Edge devices only provide basic support on service placement and deployment. A relevant concept in this context is the Edge Operating System [9], which aims at providing a softwarization platform for orchestrating and deploying network management services on Edge devices. In contrast to this Edge OS, the Computing Fleet considers applications that comprise general-purpose and business-oriented software services.

In summary, we introduced the concept of Computing Fleet as an abstract entity for representing groups of heterogeneous, distributed, and dynamic infrastructure elements across the Computing Continuum and proposed an associated reference architecture. We discussed key aspects related to management of microservices-based applications on the Computing Fleet and propose an approach for deployment of microservices-based applications. We presented a prototype to preliminary evaluate the proposed approach in a concrete Cloud-Edge scenario, which shows the feasibility of the approach in hiding the heterogeneity and dynamicity of fleets to the application developers.

Since the Computing Fleet concept is new, how it will actually be implemented and realized in practice is an open issue. We particularly investigated the application layer in this paper, however, all the other layers of the architecture require novel approaches and solutions in order to enable a fully-fledged architecture for the Computing Fleet. As part of future work, we plan to discuss various aspects and technology choices for all the architectural layers and develop a proof-of-concept for all the layers of the proposed architecture, in addition to identifying use cases from different domains for an overarching validation of the Computing Fleet concept.

Acknowledgments. This work received partial funding from the projects EC H2020 DataCloud (Grant nr. 101016835) and NFR BigDataMine (Grant nr. 309691) and FLEET (Grant nr. 309700).

References

[1] D. Kimovski, R. Mathá, J. Hammer, N. Mehran and H. P. R. Hellwagner, "Cloud, Fog or Edge: Where to Compute?," *IEEE Internet Computing,* 2021.

[2] O. Sefraoui, M. Aissaoui and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications,* vol. 55, no. 3, pp. 38-42, 2012.

[3] Z. Hao, E. Novak, S. Yi and Q. Li, "Challenges and software architecture for fog computing," *IEEE Internet Computing,* vol. 21, no. 2, pp. 44-53, 2017.

[4] F. A. Salaht, F. Desprez and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR),* pp. 1-35, 2020.

[5] P. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli and A. Solberg, "Advances in deployment and orchestration approaches for IoT-a systematic review," in *2019 IEEE International Congress on Internet of Things,* 2019.

[6] R. Mahmud, K. Ramamohanarao and R. Buyya, "Application management in fog computing environments: A taxonomy, review and future directions.," *ACM Computing Surveys (CSUR),* vol. 53, no. 4, pp. 1-43, 2020.

[7] D. Jensen, Beginning Azure IoT Edge Computing: Extending the Cloud to the Intelligent Edge, Apress, 2019.

[8] A. Kurniawan, Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning, Packt Publishing Ltd, 2018.

[9] A. Manzalini and N. Crespi, "An edge operating system enabling anything-as-a-service," *IEEE Communications Magazine,* vol. 54, no. 3, pp. 62-67, 2016.