IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 July 2023, accepted 11 August 2023, date of publication 21 August 2023, date of current version 25 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3307023

SURVEY

Automated Application Deployment on Multi-Access Edge Computing: A Survey

ÁLVARO SANTOS[1,2], JORGE BERNARDINO[1], (Member, IEEE), AND NOÉLIA CORREIA[2]
[1]Polytechnic Institute of Coimbra, Coimbra Institute of Engineering, 3030-199 Coimbra, Portugal
[2]Center for Electronic, Optoelectronic, and Telecommunications (CEOT), University of Algarve, 8005-139 Faro, Portugal

Corresponding author: Álvaro Santos (ans@isec.pt)

This work was supported by the Foundation for Science and Technology (FCT) of Portugal within the Center for Electronics, Optoelectronics, and Telecommunications (CEOT) under Projects UIDB/00631/2020 CEOT BASE and UIDP/00631/2020 CEOTPROGRAMÁTICO.

ABSTRACT The advent of technologies such as 5G makes it possible to improve the availability and quality of existing services and introduce new ones, such as enhanced mobile broadband, IoT applications, augmented reality, mission-critical services, cloud gaming, or smart infrastructure. It is now possible to get faster responses to a wide range of requests, but the addition of more users and services can still make normal operation difficult due to network congestion, bandwidth limitations, scalability issues, service differentiation, or security concerns. To solve this problem, and to help meet Service Level Agreements (SLAs), some services must be brought closer to the user. The Multi-access Edge Computing (MEC) initiative is a step in this direction, enabling cloud-like services to be moved closer to the end user, providing lower access latencies. However, the dynamic nature of edge computing environments, and the mobility of users, require the implementation of automated service delivery processes that can adapt to environmental conditions, such as the use of optimal policies tailored to the scenario. This article surveys the research focused on MECs, particularly those that use automated deployment mechanisms, namely Infrastructure as Code (IaC) tools. Today, these tools play a key role in automated deployment mechanisms, especially for maintaining optimal policies, which is still under investigation. The result of this assessment has been the identification of the relevance of IaC to these processes and the identification of future research directions.

INDEX TERMS Multi-access edge computing, infrastructure as code, automated deployments.

I. INTRODUCTION

The cloud offers several essential services for day-to-day use, including applications, storage, processing in high-performance systems, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Function as a Service (FaaS), Container as a Service (CaaS), only to mention some [1], [2], [3], [4], [5]. All of these services make it possible to increase responsiveness in a wide range of areas, with applications that are richer in functionality. However, centralization on the cloud has its drawbacks. One is latency, which is the time that elapses between the moment a request is made and the moment a response is received. Centralization can also raise some security and privacy concerns, increasing vulnerability, in addition to being a single point of failure, and having lack of transparency and compliance challenges. Other drawbacks include decontextualization of data, bandwidth contracts, and reduced overall performance. These concerns have led to a shift in the opposite direction, with researchers trying to bring the capabilities of the cloud closer to the sources of information. It is therefore necessary for information sources and their customers to have local access to services similar to those found in cloud datacenters, known as *the edge* [4].

The evolution of edge computing offers many benefits. By reducing the physical distance between data sources and processing resources, edge computing reduces latency, improves bandwidth utilization, and increases overall performance. This proximity enables faster response times, facilitates real-time analytics, and supports bandwidth-intensive

The associate editor coordinating the review of this manuscript and approving it for publication was Nurul I. Sarkar.

applications such as video streaming, immersive virtual reality experiences, and even computationally intensive IoT devices.

In light of the the above, the European Telecommunication Standards Institute (ETSI) has proposed a specification called Multi-access Edge Computing (MEC), which aims to structure and define a way to implement these micro/mini-clouds at the edge [6], [7]. Multi-access Edge Computing is an evolution of Mobile Edge Computing. It recognizes that devices can be more than just mobile. In this way, MEC is no longer restricted to the network of the supplier, but can also be made available at the offices of the customer [8]. This new form of deployment creates opportunities by offering new services, new customers, and interoperability between services and the MEC, while allowing application developers and content providers to benefit from ultra-low latency, high bandwidth for Quality of Service (QoS), and both security and privacy enhancements.

Although providers may choose different strategies and infrastructures, there are initiatives aimed at standardizing the models and protocols used, such as O-RAN [9], [10], from the Open RAN Alliance. This will contribute to a wider deployment of applications at the edge, reducing problems associated with high latency, reduced bandwidth and low throughput, while ensuring Quality of Service (QoS), information security and Quality of Experience (QoE) [4], [11], [12].

In view of the constant evolution of systems and customers, the need for a dynamic approach to MEC implementation processes is becoming increasingly apparent. To automate the process of deploying applications and services, there must be ways to describe the resources (applications, services and their dependencies) to be installed in each MEC. However, despite efforts to agree on the forms of implementation, the way in which the provisioning and deployment description is carried out is highly dependent on existing or future infrastructures. In this context, Infrastructure as Code (IaC) tools are particularly important, as they enable the description and automation of the provisioning and deployment processes without the need for step-by-step manual intervention [13], [14]. The aim of this article is to systematize knowledge about the automation of application deployment processes in edge computing environments, and related technologies, with a particular focus on IaC. To this end, the following research questions are defined:

RQ1: What methods exist for automating the deployment of applications in MEC environments?

RQ2: Which IaC methodologies are best suited for deployments in MEC environments?

The remainder of this article is structured as follows. Section II introduces the MEC concept, cloud services and the MEC standard defined by ETSI. Section III introduces the IaC concept and some of the most common IaC tools. Section IV presents the methodology adopted for related work search. Future research directions are presented in Section V, while Section VI concludes the article.

II. MULTI-ACCESS EDGE COMPUTING

The MEC is a type of network architecture that provides cloud computing capabilities, and an IT service environment, at the network edge [15], [16]. Its primary goal is to bring computing, storage, and network resources closer to end users and their devices. In doing so, MEC aims to reduce latency, improve real-time data processing, increase network efficiency, enable ultra-low latency applications, and drive innovation in areas such as IoT, Smart Homes, Smart Cities, Smart Industries, Augmented Reality, and Vehicle-to-Everything (V2X) communications. Unlike general edge computing, MEC specifically aims to provide standardized interfaces and open platforms for deploying and managing applications at the network edge.

The following subsections provide an introduction to cloud concepts and the MEC standard defined by ETSI.

A. CLOUD SERVICES

To meet the need for service availability, the cloud is increasingly being used for the provision of services and functionality on a large scale [17], [18], [19]. Depending on the required level of availability and the relevance of characteristics related to the location, security, and privacy of information, there are different forms of implementation, usually framed in public or private clouds. More recently, hybrid cloud models (complementing private cloud services with public cloud services) and multi-cloud (using cloud services from different service providers) have become increasingly important to make infrastructures more adaptable to existing needs and services more resilient [5], [20], [21]. In the context of MEC [6], [22], and edge computing [17] in general, the aim is to bring centralized cloud functionalities to the edge so that customers can take advantage of the functionalities while benefiting from response times that are more appropriate to the characteristics of the services being implemented [23], [24]. Given the market share of certain cloud service providers, it is foreseeable that at least some edge computing environments can be implemented with systems similar to those used in more traditional clouds, although adapted to the specifics of edge computing. This means that services from Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Alibaba Cloud, or open systems such as OpenStack, Kubernetes and others are expected to be available on the MEC servers. A discussion of these services can be found in [25], where an analysis of the characteristics of the most commonly used cloud services is performed and the existence of support for edge or on-premises services is confirmed.

B. ETSI STANDARD

In the current context of 5G evolution, bringing cloud capabilities closer to users will significantly reduce service access latency. This means that with the appropriate deployment of applications at the edge, users can benefit from improved QoE. But there are more advantages than just

latency reduction for the users consuming these services. By bringing processing and storage capabilities closer to the user, network congestion can be significantly reduced, benefiting all users.

However, problems can also arise from over-reliance on the edge [26]. The storage and processing capabilities at the edge are much more limited than those offered by the cloud, and may not be able to satisfy the number of local users who want to use them. The relevance of using the edge can also vary depending on the type of application. In [7], three types of services are identified: enhanced Mobile Broad Band (eMBB), Ultra Reliability and Low latency Communications (URLLC), and massive Machine Type Communications (mMTC). The eMBB applications are characterized by high bandwidth requirements, for example, video on-demand; the URLLC applications include applications with low response time requirements, such as Tactile Internet [27], Interactive Gaming, Augmented Reality, Virtual Reality, industry or automotive; and the mMTC services refer to applications that involve many devices, such as sensors, with low response time requirements, but not as low as the previous one. However, mMTC applications can cause limitations due to the number of devices that generate a large amount of data on the network.

Defining an architecture that is capable of migrating services and applications between the MEC and the cloud, or between MEC platforms, is critical to its success. Migration to the cloud may be justified not only by the need to access higher processing and storage capabilities, but also by the need to respond to concurrent application requests. The migration of services between MEC may be justified if the user has mobility. This may be the case, for example, for automotive and intelligent vehicle services, where it may not make sense to maintain an application in an MEC whose location or network connections become less reliable [28], [29].

The ETSI has defined a general architecture [6] for what a MEC should be, which presented in Figure 1. As shown, there is a clear need for management modules that allocate the essential resources to each application and ensure the functioning of the entire ecosystem. These MEC platforms are designed to make use of existing resources, some of which are already being used by operators to build communications networks. Therefore, opening up these services to customers cannot jeopardize the functioning of other services.

For the MEC to work properly, there must be services that enable it, such as:

- Discover network, users, capabilities and local services;
- Manage traffic, DNS, mobility, V2X, etc.;
- Registration of own services and discovery of locally available third-party services;
- APIs for interoperability between MEC systems and infrastructure, such as, Network Functions Virtualization (NFV) [30] and Software-Defined Networking (SDN) [31].

III. INFRASTRUCTURE AS CODE TOOLS

Cloud implementation solutions can rely on Graphical User Interfaces (GUI), dashboards, or specific task management applications, upon the creation of the first instances of services, or when production adjustments are needed.. However, these approaches become impractical when multiple instances need to be created. These are not suitable for automating the entire process. For this reason, platforms offer a Command-Line Interface (CLI) that allows services to be managed through commands, as well as the creation of sets of instructions and commands (scripts) for more agile task repetition.

For more complex and distributed applications, it will be beneficial to use languages that allow to describe the infrastructure, platforms, services and applications in a more abstract way, so that they can adapt to the different environments available. This is already possible with some traditional cloud services. These include the IaC or Application Description Templates, which allow the provisioning and deployment of applications and services to be described using scripts or Domain-Specific languages (DSL).

The initiatives and tools to accomplish an automated deployment are numerous. Among them one can find: Ansible, CAMEL, Chef, OpenStack Heat (HOT files), Puppet, Terraform, and TOSCA. These systems do not always work in the same way or have the same objectives. For example, Terraform is a system that is characterized as a tool, although it has a language for describing the needs and actions to be taken. On the other hand, the OASIS TOSCA standard is a declarative language that allows a fairly complete description of the support infrastructures, applications, services, and all the relationships between the constituent nodes, in a generic way and adaptable to the environments in which it is used. When using TOSCA, it is essential to have systems in the support infrastructure, such as orchestrators, that will use the descriptions to instantiate and execute the appropriate actions for the proper functioning of the service.

The following subsections provide more details on the above mentioned IaC tools and a comparative analysis of their characteristics. One of the points to be analyzed is the type of description, whether it is a more declarative or procedural approach, usually corresponding to the ability to be more or less abstract. Another point is the operating model, whether it includes effective execution tools or depends on other systems, and whether it runs only locally or follows, for example, a client-server model. In order for future systems to generalize the way descriptions are made, it is important to identify how the various constituent elements of a deployment are defined, and how they can be interrelated and ordered for effective deployment. It will also be important to identify the support that each tool provides for the post-deployment operations. In this regard, we intend to determine whether the tools include mechanisms for monitoring and subsequently acting on deployments to adapt them to the dynamic realities in which they will be used.

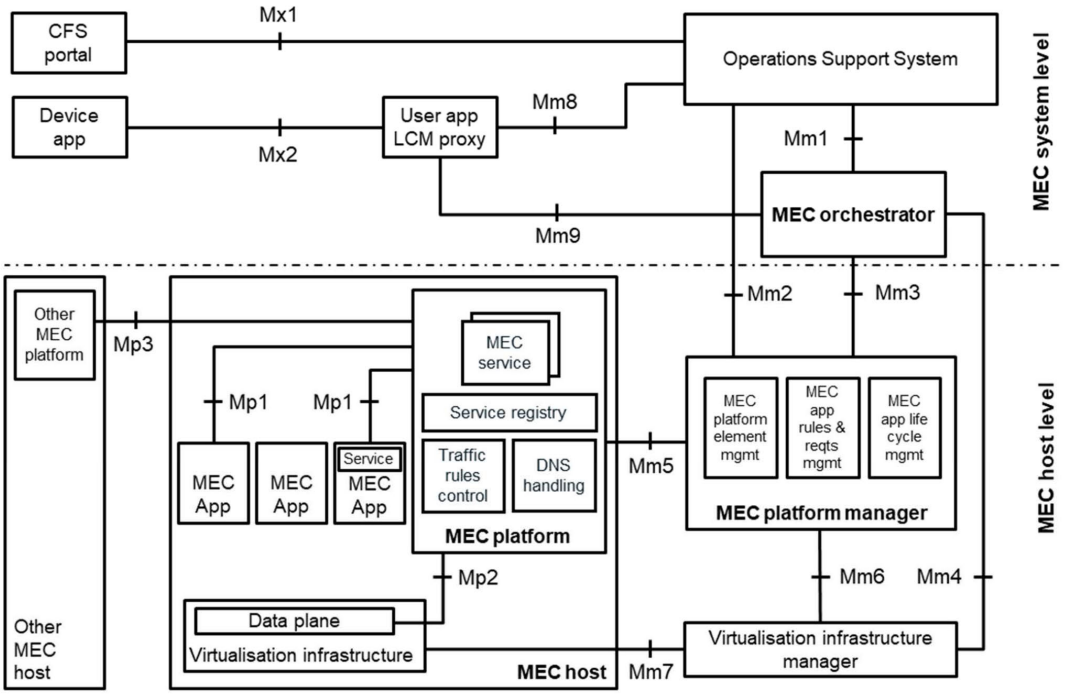

FIGURE 1. Multi-access edge system reference architecture [6].

A. ANSIBLE

This tool allows the automation of various processes, such as provisioning, application deployment, management and configuration, equipment updates, among others. However, Ansible can be used to automate any process because it allows for process descriptions that are not limited to specific functionalities, enabling interoperability with other tools or the execution of applications and scripts to extend its capabilities [32].

The specification of processes is done in a procedural way through YAML. Actions to execute processes are performed without the need to install any specific services or agents on client systems (targets), and SSH connections are used to execute tasks.

For compatibility with existing cloud platforms (e.g., AWS, GCP, Azure, OpenStack) and other systems (e.g., Cisco network equipment, Juniper, VMware virtualization systems), modules are available to configure most network services. New modules/scripts can be created (using e.g., python, ruby, bash) to act on other systems, with a very active community developing and sharing these scripts.

Ansible consists of a control node where the YAML procedures, called *playbooks*, are executed. There is also an *inventory* file that describes the target systems of the actions (where the scripts are executed). Unlike other systems that are configured remotely based on existing communication capabilities, such as a specific API, Ansible is based on local script execution, which means that instructions are executed locally on those systems after the script is transferred (e.g. Linux via SSH). This approach thus avoids the development and maintenance of APIs (see Figure 2).

The scripts allow the use of variables to pass essential parameters. For example, the IP address of a database server can be passed as a parameter to to access databases created by others. The scripts also allow conditional execution based on the success or failure of previous commands.

When compared to other tools, Ansible is not concerned with a more abstract organization of components and how they relate to each other, making it applicable to smaller projects. The main goal is to execute scripts that perform a set of tasks to provide the intended services, and the representation of a modular composition can be achieved through a good organization of the scripts. The relationships between the components should be defined by a correct sequencing of the execution of operations. Post-implementation management operations are performed through Ansible scripts triggered by external monitoring entities or manually by operators.

B. CAMEL

The Cloud Application Modelling and Execution Language (CAMEL) [33], [34] is a DSL that describes actions for deploying and managing applications and services at several levels: where services should be installed, the requirements to be fulfilled, the scalability methods that can be used, security configurations, the organization of the components and also their execution. CAMEL emerged from the PaaSage project [35]. The main objective of the project was to develop a multipurpose platform that would help not only in the design and execution of tasks during the development of

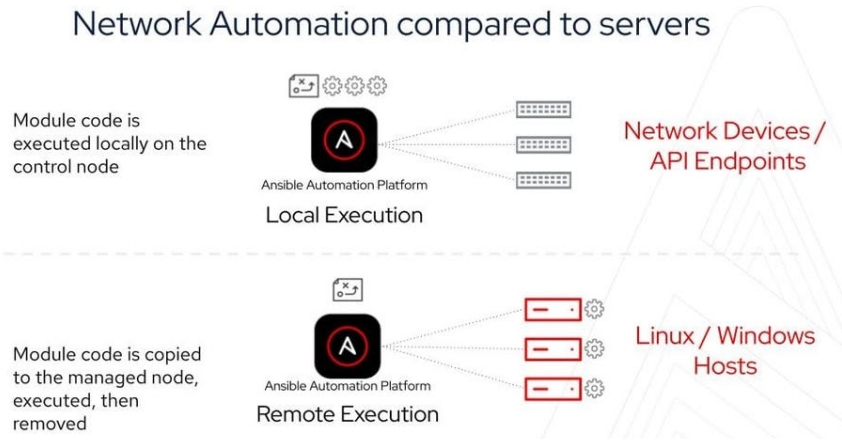

FIGURE 2. Network automation with Ansible [32].

multi-cloud services, but also in the deployment phase. This is a very versatile tool, multi-purpose and applicable to projects of different dimensions.

The DSL CAMEL [36], [37] was developed with several assumptions in mind: cloud provider independence, separation of concerns, reusability, abstraction. As it is only a language, it does not directly provide ways to implement the descriptions, so a whole system is needed that takes advantage of these descriptions to make them effective in the target systems. In this specific case, the tasks are developed in the context of the different modules that make up the PaaSage platform (as shown by the workflow in Figure 3). The platform interprets the descriptions of the desired objectives and plans how to achieve them. The necessary instructions are mapped into the existing mechanisms to interact with the components that support the application, such as servers or network devices. The main objective of the platform is to support IaaS models, although other models can be explored in its context.

In DSL CAMEL, a service is defined in a declarative way based on a file that describes all its requirements (system requirements and their dependencies) and policies too. The definition of the representative model of a service includes general information about the service (name, version, administrators, etc.), the intended model (servers, services, applications, and their relations/dependencies), operational requirements, metrics that must be respected, scaling and localization actions, among others.

All tasks involved in these processes can be adapted according to metrics collected by the PaaSage system, meaning that there will be a self-adaptive workflow (see Figure 3).

C. CHEF

The Progress Chef, or Chef for short [38], consists of a set of tools for automating the configuration of servers or cloud platforms, facilitating the management of the entire infrastructure of an application, service or company network. When compared to other tools, the main characteristic of Chef is that it works with a client/server architecture in which the server manages and schedules the instructions to be given to the clients. The clients are Chef agents installed on systems that will be the targets of the configuration (e.g. AWS supports Chef) and this operating model allows the implementation of IaaS or PaaS models. This compartmentalized approach allows for more decoupling, parallel developments in different places, and dealing with larger projects.

Chef enables the creation of *cookbooks* of *recipes* for the configuration of applications, services or infrastructure required by an enterprise. *Recipes* are described in a procedural way using Ruby.

The platform is known as Chef Infra and includes tools for creating, applying and managing the *recipes*. In addition to the tools for creating the *recipes*, as in the case of the Chef Workstation shown in Figure 4, there is an essential entity, the Chef Server, which manages the *cookbooks* and the information about the systems (clients) on which the *recipes* are applied. The *recipes* are sent from the server to the clients, which execute them in the context of an agent, the Chef Infra Client. This client can periodically check with the server to see if there are any updates to be applied. Chef has a *supermarket* where several *recipes* are available for the most common systems.

In addition to *recipes*, *cookbooks* can contain attribute definitions, *recipe* usage policies, templates for creating more complex scenarios (e.g., adapting configurations to different versions of an operating system), and other information. Chef Infra includes tools and agents for obtaining information about the status of services, to facilitate the management of the entire infrastructure. In short, the organization of the components required for a particular deployment is achieved through the appropriate structuring of *recipes* in the context of *cookbooks* that contain the set of procedures to be performed. Template files can be used to define basic *recipes*, allowing deployments to be adapted according to a set of input variables.

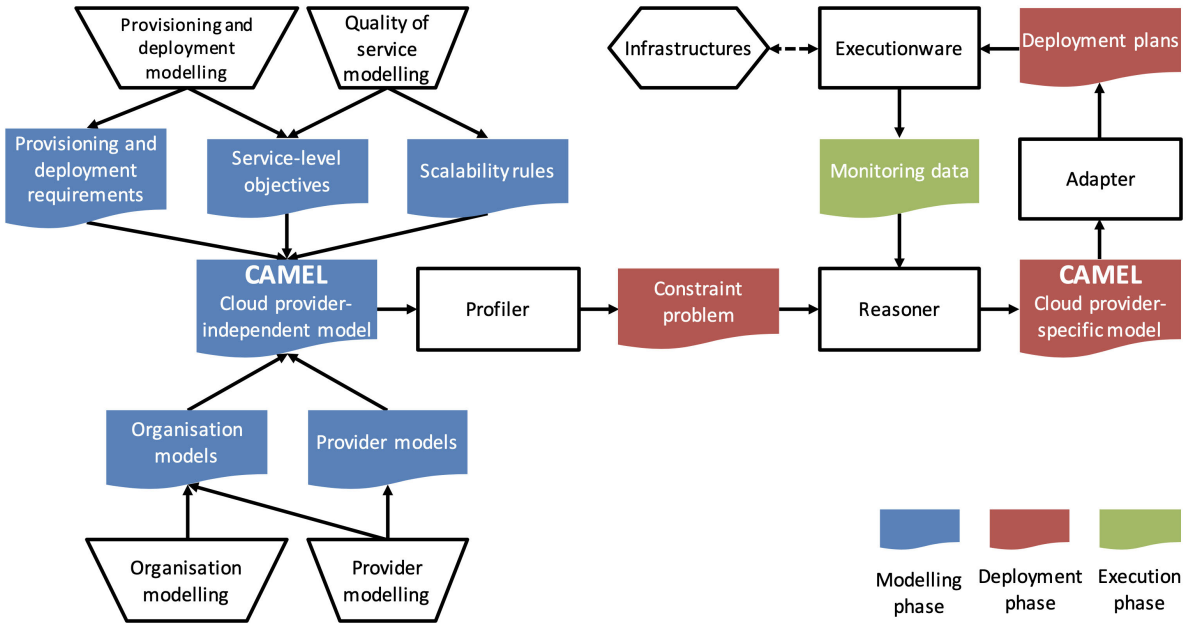

FIGURE 3. CAMEL models in the self-adaptation workflow [37].

FIGURE 4. Chef general workflow [39].

D. HEAT

Heat is the OpenStack module responsible for orchestrating service deployments [40]. As such, Heat's scope is limited to OpenStack. Initially, the descriptions were made as compatible as possible with the form of description used in AWS through CloudFormation. This form of description evolved and gave rise to Heat Orchestration Template (HOT) files [41], in which the intended behavior is performed declaratively. HOT files are described using the YAML language.

Among the various parts that can be configured, emphasis should be placed on the resources that relate to the actual services or systems to be installed, and the conditions that can affect these actions. Within the description of resources, it is possible to describe the dependencies between them, which allows the Heat orchestrator to outline a plan of actions to be taken. Its less abstract and decoupled approach makes this tool more suitable for projects of limited size, and with a very specific focus.

Heat runs on the OpenStack platform, which includes monitoring systems (e.g., Ceilometer) that allow triggering the execution of new models to change those in operation.

E. PUPPET

Puppet is defined as a tool for automating the process of installing services and systems [42]. Puppet operates on a client/server architecture. A server manages the operations to be performed on the target systems, which is done by agents installed on these systems. The agent (called *facter*) informs the server about the characteristics of the systems on which it is installed, as well as their status. The server collects this information in a catalog of services and systems, which it then uses to act in the most appropriate way, to make new deployments or adapt existing ones (see Figure 5). The connections between the client and the server are secured using SSL.

Service descriptions are done declaratively using Puppet files (`.pp`). These descriptions are implemented in Puppet DSL or Ruby, although the plans that are used by the orchestrator (called Bolt) can also be implemented in YAML [43]. Puppet files are organized into modules so that they can be reused. In this context, the Puppet Forge repository [44] is made available to the community with configurations suitable for most cloud systems and services on the market.

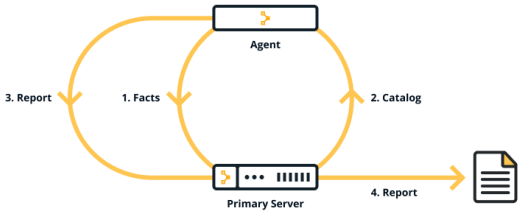

FIGURE 5. Puppet operation [45].

Resources can be of various types, including those that are built-in: *package*, *service*, *file*, *user*, *exec*, *group*, *notify*. Other types can be obtained from repositories or automatically included in certain releases [45]. Definitions allow to specify requirements for a given resource to be properly deployed, thus allowing to define dependencies between resources.

F. TERRAFORM

Terraform [46], [47] is classified as an IaC tool that allows provisioning and deployment in public or private clouds of computing instances (usually, virtual machines or containers), storage, networking, or other configuration needs to make available all the necessary infrastructure for a given application, falling under IaaS, PaaS and their derivative models. Terraform is a tool that is simple to install, as there is no need to install agents on client systems that are the target of configurations. This tool is primarily designed for the deployment of services rather than for their ongoing management. More specifically, when configurations are changed or a new plan is created, it will be executed taking into account the previous actions; the new deployment may require a prior shutdown of services or, in other cases, the use of vendor tools to make specific changes. In short, the adaptations are not a straightforward process, being more suitable for long-term deployments.

The configuration descriptions are in the form of text files (`.tf`). These files can be used multiple times in the same or different contexts. As text files, they can also be easily shared and managed through version control systems to facilitate team development. Once written, the configuration files are processed by the Terraform Provider, which generates an application plan according to the platform on which it will be available and its dependencies.When approved by the user, the plan is applied, allowing the deployment of the entire infrastructure.

Terraform currently allows working with more than 1700 providers, available in a Registry service provided by HashiCorp [48], where the most used public and private clouds are included (e.g., AWS, Azure, GCP, OpenStack, VMware, Kubernetes).

The configuration files just mentioned consist in the declarative representation of the infrastructure with a successive specification of blocks, each one representing a relevant element for the configuration, such as: resource, variable, provider. The language follows the format defined by HashiCorp itself, respecting the HCL syntax [49].

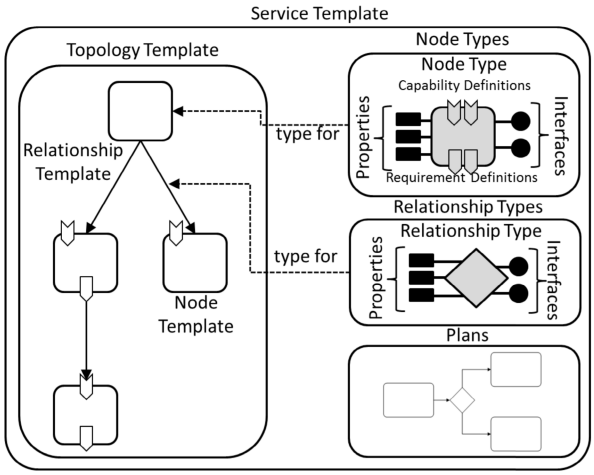

FIGURE 6. Structural elements of a TOSCA 1.0 *service template* and their relations [53].

G. TOSCA

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an initiative of the Organization for the Advancement of Structured Information Standards (OASIS) to define a generic language for describing cloud services and all the resources and dependencies required for their deployment [50]. The TOSCA descriptions are made in a declarative way and aim to be as platform independent as possible, ensuring portability of applications between systems.

YAML is used for the descriptions, improving readability and making the scripts easier to understand. At the time of writing, the standard is in version 1.3 [51] and version 2 is in preparation [52]. A TOSCA script is called a *service template* and contains all the information necessary to deploy the applications it describes. A *service template* may have dependencies on others to complete its descriptions.

A *service template* contains generic information (*metadata*) about the deployments, but also allows the definition of all application details. Element types can be described in an abstract and reusable way, and then application components are specified according to these types (see Figure 6).

The above scripts describe the types of nodes that are considered relevant to the deployment in question. These types (`node_types`) correspond to generic and reusable ways of describing the constituent elements of the configuration, from which they are concretized to suit the applications, in the so-called `node_templates`. The existing relationships between them are also defined. For instance, a node corresponding to a database engine requires a host (machine and OS) on which it will be installed. In this situation, there is a relationship between the nodes representing these entities, properly represented in the TOSCA language by `relationship_types` and `relationship_templates`. It is possible to define capabilities for descriptions (`capability_types`), and also any requirements that nodes or relationships may require.

In the `topology_template` section, the necessary instantiations of each type and the concrete relationships between them (e.g., `node_templates`, `relationship_templates`) are objectively described. Through the interfaces associated with each node or relationship, the necessary mechanisms that can lead to the realization (provisioning, installation, configuration) of the services in question are defined and initiated. For example, the actions to be performed when a node is created, started, or stopped.

At this stage, TOSCA is only designed for the deployment of applications, while their subsequent management is not a priority. However, in the context of interfaces and policies, events can be generated at any time that lead to a restructuring of the deployment, possibly in response to service scaling or relocation requirements.

Once the *service templates* have been created, they can be distributed via a TOSCA Cloud Service ARchive (CSAR) file, which includes the actual TOSCA descriptions and any additional resources required for the deployments (these can be scripts in other formats, virtual machines images, containers, etc.).

TOSCA emerges due to a lack of standardization in the way cloud services are operating. In fact, a lack of portability can potentially slow down further developments, so the idea behind the TOSCA standard is to render improvements in the deployment, termination, and any other management function of cloud applications. TOSCA provides mechanisms to describe all the nodes and relationships required for a deployment, but it does not provide tools to execute the procedures described in the descriptions. This task is left to the TOSCA orchestrators, which can be implemented in a suitable way for each objective. Some orchestrator examples are: xOpera (implements the TOSCA standard with the Ansible automation tool where Ansible *playbooks* can be used as orchestration actuators within the TOSCA interface operations) [54], [55], Cloudify (an extension to the TOSCA descriptions that does not follow the standard exactly) [56], Ubicity (an orchestrator whose implementation fully respects the standard) [57], *Turandot* (an implementation for Kubernetes) [58].

H. ANALYSIS OF IaC TOOLS

All the tools described (summarized in Table 1) allow the representation of suitable configurations for most services, but in some this is done more concretely while in others it is done more abstractly. According to [13], the best solution may be to complement the use of one tool with others. For instance, xOpera, which is based on TOSCA, uses Ansible to describe the implementation of TOSCA interface actions.

The languages used to describe deployments vary, although YAML is used in more systems. Puppet, the oldest of the platforms presented, in its latest versions already allows the use of YAML in parts of its definition (plans) to make the configurations easier. The main difference in terms of descriptions is that some follow the procedural model, with a predictable sequence of execution, while others have a declarative model. Those that follow the declarative model present alternatives for executing scripts in a more sequential manner, in order to translate essential steps of service configuration.

Due to the complexity that some applications can reach, the ability to organize entities in a more abstract way, and to identify their constituent components can facilitate their definition and subsequent maintenance. TOSCA, as a standard, is the one that can more accurately translate the complexity of the system into nodes and their relationships, although there are ways of doing this in the others. The cases where it may be more complicated to create a representation that allows the best visualization of the constituent nodes, their relationships or dependencies, are those that represent procedural models, such as Ansible and Chef. In these cases, it is possible to minimize these problems with a good organization of the scripts, allowing the separation of the configurations of each element into autonomous and easily identifiable files.

In declarative models there is always a way of specifying the dependencies between the various elements. This is a fundamental point so that the deployment is done correctly, and is achieved in all of them (e.g., it makes no sense to install a database engine and then install the operating system on which it will run).

In the procedural models, the metrics and operating conditions to be verified can be done in the traditional way as in a programming language (e.g., `if`, `when`, or similar). In the remaining models, there are ways to define them by specifying the requirements and capabilities of the resources, following the terms used in the context of TOSCA, but with similar solutions in the remaining formats (e.g., using of keywords like `constraints`, `conditions`, `required`, `provided`).

In terms of the types of cloud computing services supported, the tools mentioned allow support for IaaS environments and, directly or indirectly, PaaS. In the case of CAMEL, the documentation assumes that it only fits into the IaaS model.

All tools support the requirements of the first two phases of the service lifecycle as defined in the TOSCA 2.0 undergoing proposal [52]: Day 0 - Service Design (modeling the design of a solution); and Day 1 - Service Deployment (resource provisioning and service delivery). Regarding the operations for a post-deployment phase (third phase of the service lifecycle in TOSCA 2.0, designated by Day 2 - Service Management), not all tools allow these operations by themselves. Ansible depends on the existence of an external system (e.g. a monitoring service) that executes the correct scripts. Terraform also needs an external stimulus to change a deployment, but it preserves the previous state and the new deployment operations are planned taking into account the previous state and the goal. TOSCA includes mechanisms that can be triggered by external events (policies and notifications), but depends on the orchestrator implementation and external systems.

Configuration examples for the various IaC tools are available at: *https://github.com/ansualg/IaCTools-Examples*.

TABLE 1. IaC tools.

	Ansible	CAMEL	Chef	Heat	Puppet	Terraform	TOSCA
Creator	AnsibleWorks	EU Project	Progress	OpenStack	Perforce	HashiCorp	OASIS
Type	Procedural	Declarative	Procedural	Declarative	Declarative	Declarative	Declarative
System Model	Server and SSH	Server and APIs	Client/Server	Server and APIs	Client/Server	Server and plugins	IaC only
IaaS, PaaS	IaaS, PaaS	IaaS	IaaS, PaaS	IaaS (mainly)	IaaS, PaaS	IaaS, PaaS	IaaS, PaaS
Language (DSL)	YAML	CAMEL DSL	Ruby	YAML	Puppet DSL, Ruby	HCL	YAML
Files	*Playbooks*, *inventory*	CAMEL files	*Cookbooks*, *recipes*	HOT files	Puppet files	Terraform plan	*Service templates*, CSARs
Component representation	No[1]	Yes	Yes	Yes	Yes	Yes	Yes
Dependencies definition	No[1]	Yes	No[1]	Yes	Yes	Yes	Yes
Configuration abstraction	No[1]	Yes	No[1]	Yes	Yes[1]	Yes[1]	Yes
Policies or QoS	External	Yes	Yes	Yes	Yes	Yes[1]	Yes
Post-deployment operations	External	Yes	Yes	Yes	Yes	Yes	External

[1] Programmatically or based on the definition order

IV. RELATED WORK

This section provides an overview of the use of techniques and technologies related to the automation of application deployment processes in MEC, including service adaptation to server overloads, increase in requests, and user or service mobility. Given the importance of automation processes for the present work, where the set of IaC tools mentioned in Section III becomes particularly relevant, a search was conducted on works that mention at least one of these tools. Authors who give relevance to the IaC tools by referring to any of them show sensitivity to the importance of automating the deployment process. The search was carried out in the main repositories, ACM, IEEE *Xplore* and Scopus, filtering as follows:

Terms in metadata, title, abstract, keywords:
`"Multi-access Edge Computing" OR "MEC"`

At least one IaC:
`"Ansible" OR "Chef" OR "CAMEL" OR "Puppet" OR "Terraform" OR "TOSCA" OR ("Heat" AND "OpenStack")`

Publication date:
$2018.01.01 \leq date \leq 2022.12.31$

Type:
`Peer-reviewed publication`

The articles that met the defined criteria are summarized in Table 2. The contributions analyzed below are intended to provide valuable insights into the challenges and solutions associated with automating and adapting services in MEC environments. Understanding the chosen deployment platform, supporting scaling and migration, monitoring key characteristics and leveraging key technologies such as SDN, NFV, Network Slicing or Containers are critical for efficient and dynamic deployments. In addition, the use of IaC tools plays an important role in automating the deployment process.

A. APPLICATION DEPLOYMENT

It is envisaged that applications from different clients can coexist in a MEC and, therefore, operate in isolated environments that ensure their security and the security of other clients. According to ETSI, this can be achieved using Virtual Machines (VMs). However, deploying a VM is a heavy process, with high processing, memory, storage, and bandwidth requirements. Therefore, a more recent work has considered lighter deployment solutions based on containers or unikernel operating systems [79]. The literature tends to focus more on containers, which are also preferred for implementing Virtual Network Functions (VNF) due to their light weight, easy configuration and fast implementation.

Containers are seen as a lighter and more practical solution when instances are placed at the edge, for end-device representation. In [59], a container is created for each end-device (body camera), which is responsible for offloading tasks from its device. Ansible is used to build a testbed for the created scenario. In [60] and [61], the benefits associated with the speed of deployment when transitioning from VMs to containers are highlighted, in addition to the gains in flexibility. These works also take advantage of Ansible, but the authors point out that Ansible is very effective for rapid prototyping, although not so suitable when high performance is required. In [62], the transition from VMs to containers is identified as the best option for 5G services, while [63] points out containers as a future solution for service migration in the context of virtual mobile devices, given the improvements they can bring due to their reduced overload.

Regardless of whether containers or other solutions are used, there are several works that highlight the improvement in QoE when processing is placed closer to the client. In [64], the authors consider three cloud levels when deploying the application: main-cloud, mini-cloud and micro-cloud. The main-cloud refers to the traditional cloud located in large datacenters, the micro-cloud represents the services located at

TABLE 2. Related work.

Work	Processes			Monitoring[1]	SDN	NFV	Slicing	IaC Tools	Containers
	Deployment	Scaling	Migration						
[59]	Kubernetes	–	Stateless	c, l, p, j	–	–	–	Ansible, Puppet	Yes
[60]	Docker	–	Stateful	c, l, m, r	Yes	–	–	Ansible	Yes
[61]	Docker	–	Stateful	c, l, m	–	–	–	Ansible	Yes
[62]	5G-CARMEN	–	Yes	b, l	Yes	Yes	Yes[2]	Heat, TOSCA	Yes
[63]	OpenStack	Yes[2]	Yes	s	Yes	Yes	–	Ansible	Yes[2]
[64]	OpenStack	Yes[2]	Yes	d, l	Yes[2]	Yes[2]	–	Heat	–
[65]	Docker, VirtualBox	Yes	Yes	b, l	Yes	Yes	Yes	TOSCA	Yes
[66]	OpenStack	–	Yes[2]	b, l	Yes	Yes	Yes	Heat, TOSCA	Yes
[67]	OpenStack	Yes[2]	Stateless	c, h, l, r	–	Yes	–	TOSCA	Yes[2]
[68]	OpenStack	–	–	c, l, m	–	Yes	–	TOSCA	–
[69]	Opensatck	Yes	Yes[2]	e, l, o, u	–	Yes	–	Heat, TOSCA	Yes[2]
[70]	OpenStack	Yes	Yes	b, e, l, r	Yes	Yes	Yes	Ansible	Yes
[71]	Kubernetes, VMware	–	–	b, c, l	Yes	Yes	Yes	TOSCA	Yes
[72]	OpenStack, Kubernetes	Yes	Yes[2]	b, l	Yes	Yes	Yes[2]	Ansible, Heat, TOSCA	Yes
[73]	OpenStack, Kubernetes	Yes	Yes	h, l, t	Yes	Yes	Yes	TOSCA	Yes
[74]	Melodic	Yes[2]	Yes[2]	c, h, l, m, r	–	VNF	Yes	CAMEL	Yes
[75]	OpenStack[2]	–	–	l	Yes	Yes	Yes	Heat, TOSCA	Yes[2]
[76]	Kubernetes[2]	Yes	Yes	l, t	Yes	–	–	Ansible	Yes[2]
[77]	OpenStack	Yes	Yes	l, t	Yes	Yes	–	Heat	Yes[2]
[78]	OpenStack	Yes	–	c, e, m, o, q, u	Yes	Yes	Yes	Heat	Yes

[1] b - bandwidth, c - CPU load, d - delay, e - execution time, h - throughput, j - hop count, l - latency, m - memory, o - cost, p - priority, q - QoE, r - RTT, s - packet source, t - traffic, u - users or requests
[2] lightly mentioned

the edge, close to where they are needed, and the mini-clouds appear at an intermediate level (at the level of what can be considered Fog) that allows the aggregation of services from multiple micro-clouds. The study shows that response times can be reduced up to half by moving services from the cloud closer to the customer. OpenStack services were used to support the test environment, and these services were orchestrated using OpenStack Heat.

In addition to the choice between using VMs or containers, deploying applications at the edge requires flexible provisioning and organization of network resources so that access to the provided instances is transparent. Many authors highlight the use of SDN or NFV for this purpose (see Table 2). Some work focuses mainly on the benefits of implementing NFV and SDN services in MEC, individually or together, as in the case of [65]. The authors conduct a comparative study to demonstrate the benefits of using containers to deploy VNF compared to native or virtual machines. For example, according to the study, the Squid application consumes 16.8% of CPU when deployed via a VM, compared to 1.25% of CPU when deployed via containers. The difference in memory consumption is also significant. When 2GB usage limits were imposed on both VMs and containers, the VMs used all of the memory, while the container version used only 93.66MB. The experiments were carried out using a platform called LightMANO, and resources are advertised using TOSCA-based descriptors via the LightMANO REST interface.

The type of application can also have a strong impact on the choice of the most appropriate deployment model. As mentioned above, 5G applications are classified into the following three groups: eMBB, URLLC and mMTC. In [66], the authors consider the possibility of deploying applications in two possible locations depending on their type: Central Office or Edge. With this in mind, an algorithm is presented to choose among four possibilities: *i*) Central Office only; *ii*) Edge only; *iii*) Central Office and Edge in parallel (according to the level of resource usage); *iv*) Edge and Central Office sequentially (first Edge and then Central Office). The algorithm always starts with the "Central Office only" model, but depending on the resources available at the edge, it will offload to one of the other models (opting for "Edge only" when it has 100% of the available resources). The following ratios between cloud and edge deployments are shown: 10:0 for eMBB, 1.5:8.5 for URLLC and 7.8:2.2 for mMTC. The creation of the test environment was supported by OpenStack and Heat, but communication control was performed using OpenStack's Tacker and TOSCA.

In [67], a fully functional prototype of a MEC is presented, following the ETSI standards. The prototype was used to compare the availability of services when using a MEC with the availability of services when these are placed in the core of the cloud, in particular considering video streaming services. From the tests, the authors obtained core-to-edge and edge-to-core handover times of 73.3s and 29.0s, respectively. These times include the VM activation required to support the services. The throughput tests, which considered processing at the edge or core and the level of CPU utilization required for each, achieved equivalent throughput values for CPU utilization levels of 20% for the edge and 15% when forwarding traffic to the core. The authors conclude that there is still room for improvement. In terms of RTT, the edge values were around half of those achieved for the cloud when considering video requests to a server. It was also shown that the latency introduced by the MEC itself was negligible and that no losses were detected during handover. In this work, TOSCA is used as a model to describe the applications in the MEC. The prototype has been developed using OpenStack.

OpenStack is also presented in [68] and [69] as a solution for implementing MEC, through an extension called Automated Provisioning framework for MEC (APMEC). Deployments are performed using TOSCA templates, which are later translated into HOT to be orchestrated by OpenStack Heat. In this work, the OpenStack Tacker plays a crucial role in orchestrating the VNF essential for the operation of MEC services (a MEC service is defined as the combination of a MEC application and the set of virtualized network functions essential to its operation). APMEC enables VNF reuse and can therefore support more user requests. The results show more than 60% of accepted user requests with 30% lower routing costs compared to a baseline approach.

In [70], it is proposed to implement IoT applications by decomposing them into simpler and reusable functions, which are then made available through NFV. The network support for this architecture is based on SDN. In this way, it is intended to simplify the deployment of applications through an organization called IoT Service Slicing Functions. This architecture also facilitates the migration of applications between MEC, as well as scaling operations. Deployment is done using Ansible *playbooks*. The work also demonstrates the difference in performance when deploying using containers or VMs. The first solution, using Docker, takes three times longer than the second.

In [71], the concept of slices is also used to allow the sharing of services provided by the MEC owner to multiple MEC customers (who in turn provide services to their end-users). The proposal follows the MEC ETSI definition but introduces changes to support the deployment of APplication Slices (APS) based on a set of Application Component Functions (ACF), corresponding to Docker containers running on Kubernetes.The use of slices for each MEC customer allows isolation. The authors present a work-in-progress system and the results show that the increased load of one customer does not affect the overall service quality of the others. Experiments also show that results can be improved if services can be shared, rather than identical services from different slices competing with each other. In this case, the improvement in latency when accessing services is around 20%. The authors also point out that MEC APplication Slice Subnets (MAPSS) can be deployed using TOSCA, although some customization is required.

The work [72] presents a testbed for 5G and MEC environments based on open-source software is presented. The main contribution of this work is to gather information on how to set up testbed environments at zero cost using containers, Kubernetes and OpenShift, although VM deployment can also be enabled. Regarding the deployment of applications and VNF, it is said that the intention is to use TOSCA, which will then be translated into OpenStack Heat.

Another model found in the literature for service provisioning is based on private MEC implementations, in order to ensure lower response times and information privacy. This is the case of [73], which presents a developed platform, ECoreCloud, for the provision of 5G network services in ''smartfactory'' environments using private implementations of these services or using the infrastructure of the network operator. The authors explore the possibility of moving physical machine services associated with the control of factory equipment to the cloud, in order to make the services more flexible and take advantage of the cloud. This requires ensuring that adequate response times can be achieved for controlling the equipment. Several implementation models are discussed, from completely private services (for use in industry) to services shared with the public. It was shown that services requiring response times of up to 30 ms, could be handled through the services provided by ECoreCloud with times below this limit. In this work TOSCA is used to perform the VNF deployment.

A hybrid model is also explored in [74]. The article compares the RTT and throughput for both private cloud and hybrid implementation models. The private clouds used were NorNet Core (consisting of servers distributed across several universities and research institutions in Norway) and Simula (an implementation built at the Simula Research Laboratory based on OpenStack). Hybrid models were set up using Simula and AWS, and, also, NorNet Core and AWS.

The work [75] focuses on another dimension that should be taken into account in the deployment of services in MEC, when the mobility of users leads to the crossing of borders and the use of roaming services. In these situations, and also when trying to allocate resources during periods of high demand, it may be necessary to use techniques normally used in stock markets to set prices for the use of existing resources. Instead of relying on prices set by ''big providers'', the price is negotiated between the supplier and the customer. The setting of usage costs according to supply and demand, as well as the acquisition of futures and options contracts for the use of services, as in the stock market, should be a line of price application to be considered. The authors give an example where a new episode of a streaming series is released and rights can be purchased in advance for certain future dates. Eventually, this acquisition can be canceled by selling the rights to other interested parties and according to the prices of the moment. The provision of services may eventually be automated, and the importance of the existence of tools (OpenStack Heat and TOSCA are given as examples) for defining the services to be orchestrated and transferring the images for later instantiation is mentioned.

B. MIGRATION AND HANDOVER

MEC service relocation is an important application management mechanism and is essential to cope with increases in load on certain servers or to accommodate client mobility. In general, the overall objective is to ensure service continuity when systems are moved, while ensuring that usage policies, QoS and QoE are maintained.

Service migration can be done from the centralized cloud to the edge, to bring processing closer to customers and improve QoE, or from the edge to the cloud when there are not enough resources at the edge to meet demand. In [76],

the authors argue that the transition between the cloud and the edge should be transparent to the customer. SDN is used to route traffic to the appropriate application instances at the edge, or to the cloud when it is no longer appropriate to run them close to the client. By using SDN, applications can be made available at the MEC level without any customization. However, it is necessary to register the application at the SDN controller level, by specifying its IP and ports. To prove the viability of the proposal, a testbed is set up using Ansible to configure the nodes of the virtual network topology. According to the authors, even with a slow first request, the transparent approach is still significantly faster than any non-transparent solution that requires redirection via the cloud.

In [59], two different ways of deciding on service migration are considered: centralized and decentralized. The centralized decision takes into account data collected globally from the whole system. The decentralized decision is made by the edge devices based on the information exchanged between them. The QoS values evaluated were the CPU load, the priority of the task, its security level, or the number of hops between the end device and a candidate edge device, which reflects the path length and thus the latency. However, in such work, only the centralized version is used and only stateless applications are considered for the tests, i.e., the authors only consider migrations where it is not necessary to secure the execution state between the instantiations running at different edge locations. A testbed using Ansible was set up to perform the tests. The possibility of using Puppet to perform the deployment during load balancing operations is also mentioned. Simplified migration tests were carried out, simulating the failure of an edge device. Relatively low transfer values were obtained, measured from the time of failure in one system to the time of availability in another. However, the tests do not take into account the need for software or data transfer.

In [60] and [61], the authors propose an extension to the ETSI MEC [6] definition using container migration technologies, which is much easier and faster than VM migration. The differences between stateful and stateless migration are discussed, and it is pointed out that in stateful migration the protection and recovery of the so-called ''user context'' should be the responsibility of the application. The application is the one that knows what the transition of the user context between deployments in different hosts or MEC means. To make the whole process more flexible, it is proposed to store the state in a container, which will facilitate its transfer between MEC. The migration consists of two phases: the transfer of the application and the transfer of the ''user context'', both based on the migration of containers. In tests, migration processes of about 11 seconds with a stop time of 3 seconds were achieved. The test environment was set up and controlled using Ansible.

There are works, such as [63], that also assume the separation of data storage and processing locations. In this case, the data is centralized (although the existence of a local repository synchronized with the global one is foreseen), and its immediate transfer is not required. Migration needs are detected when requests for the same device appear from different routes. This work uses VMs, but the authors mention the need for containers to speed up migrations. Ansible is used to deploy and start the images at the new location. Processing is maintained in the old system while the new one is prepared in the new location. When the new image is available the processing is redirected using SDN. Tests show that using an SDN controller at the cloud results in migration times of 24 ms, while placing the controller at the edge results in migration times of 2 ms.

In situations of high mobility, such as in V2X environments, the existence of more efficient migration processes becomes even more important. In [62], the migration problem is studied for situations where the vehicles cross, for example, the borders between countries. Given the nature of the vehicles, migration is done in advance by deploying the necessary VNF in the MEC that is expected to be selected, using the movement of the vehicle as a reference. This work distinguishes between stateful and stateless migrations. However, the stateless solution was considered more suitable for implementation, given the huge diversity of domains to be reconciled by the management platform, to maintain continuity of service with low latency, making this management difficult. For this reason, this work also considers the maintenance of the VNF in the previous MEC, transferring only those that are considered more relevant. In the context of the project under development the authors indicate the use of TOSCA, YANG or OpenStack Heat (HOT files) for the orchestration of the solutions they intend to implement.

The study in [67] tested the handover of video services between the cloud and the edge, and no loss of service was detected. The time between the handover request and the time when the service is fully instantiated and delivering content was measured. This cloud-to-edge handover time was more than twice the edge-to-cloud time, but the tests performed didn't require any state transfer, which facilitates handover operations.

In [77], the provision of Content Delivery Network (CDN) services is studied, in the particular case of service provision when users are traveling on a train. The goal is to provide the service close to the user, but in the context of the work the location is quite variable. The system includes a monitoring module that makes it possible to know the current location and predict the future location based on the type of the transport. In this case there is no need to migrate applications, as the authors mention that the applications are already available in the MEC. Only the need to have the content (video) available according to the variable location of the users is considered. The results show a 10.9% reduction in network core load, while achieving a cache hit ratio of 99.8% (avoiding the need to fetch from central servers). These cache utilization levels were achieved as the number of users increased, and considering a small number of videos to watch (10 videos, for a total of 6.8 GB).The implementation of the testbed included the use of OpenStack Heat. SDN is used to redirect traffic to the appropriate MEC.

C. SCALING

The need to satisfy adequate QoE levels requires permanent monitoring of services. Increases in RTT values can be the result of increased latency due to mobility of client devices, as discussed in Subsection IV-B, but they can also be the result of an increase in load on servers. This increase can be due to several factors, such as an excess of available applications (possibly from multiple clients), but it can also be an internal application problem, or too many users. Scaling operations are essential to deal with these cases, and improve user QoE.

A CDN architecture, to be used by video streaming service providers, is proposed in [78]. This proposal takes into account what is known as CDN slicing. In this work, the systems are monitored to respond to overload due to an increase in the number of users, and the load on CPU, RAM and storage is taken into account. Two possible solutions are mentioned: vertical scaling and horizontal scaling. Vertical scaling corresponds to an increase in capacity (CPU, RAM, ...), while horizontal scaling corresponds to an increase in the number of instances (requiring load balancing systems for proper operation). In this work, the authors focus on vertical scaling. When an overload is detected, a new VM with more capacity is started while the previous one remains running. Only when the VM with more capacity is available will it start to provide the services, discarding the previous one. The algorithm also considers scaling down, when fewer resources are needed, making it possible to reduce processing capacity and opt for a more economical solution. OpenStack Heat is used to set up the test environment for evaluating the algorithm, and the tests performed show that it is possible to reduce the periods during which the user is exposed to low levels of QoE, as scaling up is performed as soon as a drop in QoE is detected.

An algorithm for the scaling of IoT services is presented in [70], whose implementation is done using IoT Service Slice Functions. These functions result from the decomposition of more complex services into simpler functions, implemented using VNF available in MEC and reusable between IoT devices. This makes the process of scaling some of these simplified functions easier.

D. DISCUSSION

The works presented address many of the difficulties that can arise when deploying applications in MEC. The motivations mentioned for deploying services at the edge, and in particular in MEC, include the need to reduce latency and improve user QoE. The preference for more flexible solutions is highlighted, and here the use of container-based deployments is repeatedly mentioned as a lighter and more flexible solution. VM-based deployments are perceived to be heavier and more time-consuming, making not only the deployment task but also the migration of services or scaling tasks more difficult.

Scaling operations are already common when adapting services to the demand, whether in the cloud, at the edge or on-premises, but now service migration is becoming particularly important in the context of edge deployments. Given the high mobility of users, it will be necessary to adapt the availability of services to maintain satisfaction levels. Mechanisms to implement service migration between MEC may also be required to achieve better QoS and QoE.

Depending on the services to be provided, different solutions are considered. In situations where the evolution of the user's location is predictable, as well as the content that will be required (e.g., video on demand), the necessary resources can be provided in advance. There are scenarios, considered in some works, where users may not benefit significantly from moving services to the edge, or the gains are relatively small. Migrating services between MEC to meet SLAs, policies, QoS or QoE requirements can involve heavy processes. The works analyzed mention stateful migrations, where the state of the application has to be transferred, and stateless migrations, where the state does not have to be transferred. In some works, the stateless option was chosen for the first prototype tests because it was considered to be easier to implement, but it will certainly not be a reality in most cases. In the works where stateful migration is mentioned, the option for defining user contexts based on containers appears to be the most reasonable. Migrating an entire VM, including its entire service state, will result in increased transfer and repositioning times in the target systems, which will affect the QoE of using these services. In situations where even a container-based solution is too heavy to meet service continuity requirements, it may be necessary to plan the transition to minimize service downtime as much as possible. One possible solution is to continue to provide the service from the old site until the new site is fully prepared. Another solution is to separate the state and data from the applications themselves, with the data possibly at a location other than the application site. The use of synchronization mechanisms with cloud services from the current/old site to the new site is also presented as a possibility. In any case, it is possible to verify through the works that the existence of smaller states and applications facilitates this task. In this sense, there are approaches based on the concatenation of smaller functions (VNF) that could facilitate all these processes. The scaling operations will also be easier in this situation.

In MEC, multiple applications with different requirements will certainly coexist, but it is essential that they do not interfere with each other or affect the overall use of the services. Some works point to slicing techniques as an important contribution to isolate applications or to share MEC resources between them.

Automation of the deployment, scaling and migration processes will be essential and IaC tools will certainly play a fundamental role. In the articles reviewed, the IaC tools traditionally used for deploying applications and services in the cloud were also used for the deployment in the different service delivery models: at the edge, on-premises or in hybrid models. The analyzed articles mention the use of IaC to set up the test environments, as well as a means of transferring

the configurations to be performed in the target systems (e.g., in the case of service migration). In the case of stateless migrations, the use of descriptive ways to indicate the configurations (to be made in the systems) allows the process to be simplified, with TOSCA being mentioned several times as the most abstract and generic model to cope with the heterogeneity and interoperability of the systems. Although TOSCA is a standard, its practical use is not yet widespread, as Ansible is often used as a tool for automating the installation of systems.

Monitoring resource usage at the edge is also quite relevant. In addition to service response times, the articles also analyze the load levels reached by the CPU and memory usage. With regard to response times, it is mentioned that comparative time monitoring between different solutions is carried out in order to select those that offer an appropriate level for the services to be provided.

From the works analyzed, it is possible to draw some conclusions about the research questions presented:

- *RQ1: What methods exist for automating the deployment of applications in MEC environments?*
  It is possible to verify that the automation of application deployment processes in MEC follows the same principles used for deployment in the cloud. However, the emphasis is on the need to make the processes lighter and more agile, taking into account the more limited resources at the edge, as well as the nature of the users, whose mobility requires greater dynamism and adaptation of the processes performed. For example, it highlights the option of deployments based on simpler VNF or containers rather than VMs, or the option of slicing techniques to help isolate applications that may share the same resources.
- *RQ2: Which IaC methodologies are best suited for deployments in MEC environments?*
  The reviewed articles consider IaC tools to automate deployment processes, and testbeds have been built to assess the validity of the proposals. The references used did not highlight the need for modifications when adapting them to MEC environments. Furthermore, the use of IaC (in particular those that allow a more generic and adaptable description) is identified to as an essential tool to enable deployments in different MEC systems.

V. FUTURE RESEARCH DIRECTIONS

The deployment of services in the MEC is intended to improve user QoE, but there are challenges that arise and need to be addressed so that the quality is not compromised. Some tasks related to application deployment, application migration between different points of the network, and service scaling to cope with variable usage intensities, were identified as research issues, and the following future research directions can be outlined:

- **Modular and distributed applications for dynamic systems**
  The use of shared and dynamic systems, such as those envisaged for MEC, will benefit from simpler and modularly organized applications, such as microservices [80], which allow only the essential to be executed at any given time, and freeing resources for use by other applications. The decomposition of applications into simpler, reusable modules allows them to be delivered to their users in a more reliable and distributed way. This way an application can run in a distributed manner, with modules available and used in different locations in the cloud, at the edge or on-premises. As the user moves between locations, modules can be made available more quickly, closer to where they are needed. However, between the time one module starts processing information and the time it is sent to the next module, the latter may have changed its location on the network, or an alternative module may have been assigned. Appropriate techniques are needed to address these issues.
- **Support for the creation of dynamic mashups based on RESTful paradigm**
  Distributed applications benefit from effective ways to connect their different modules, allowing for flexible application deployments. Distributing an application across multiple network points requires the ability to transfer state or data between modules. The output of one or more modules becomes the input of others. When using RESTful architectures, the state or results produced by one module can be directly transferred to another module defined in the execution plan, reducing the problems of state management and maintenance, particularly in mobile systems. To simplify the process of service discovery and certification of the modules used, the establishment of appropriate connections may be supported by more centralized service registries or may rely on the use of SDN, which allows traffic to be redirected to the appropriate locations. This type of implementation creates dynamic pipelines between the different modules, and these must be planned properly.
- **Mashup optimization**
  The way in which these processes would be carried out needs to be optimized so that applications do not lose reasonable levels of QoE and comply with contractual policies. Given the load that MEC can experience, it is important to continually optimize services as demand evolves or requirements are not met.

VI. CONCLUSION

This survey reviews existing research on MEC and the use of IaC as a mechanism for automated deployment. The improvements in QoE, QoS and SLA compliance that the MEC can provide are discussed, and key considerations are highlighted. The core features of IaC tools for automating the deployment processes and adapting them to the target systems, given the dynamic nature of these processes, are also emphasized. However, the variability of this process and the use of shared resources require special attention to ensure that global service quality levels are not compromised. Dynamic adaptation

of deployments, migration processes to support user mobility, and scaling techniques to accommodate varying number of users, become essential considerations. Therefore, the implementation of deployment optimization techniques is expected to play a critical role in advancing this field.

REFERENCES

[1] A. Puliafito, G. Tricomi, A. Zafeiropoulos, and S. Papavassiliou, "Smart cities of the future as cyber physical systems: Challenges and enabling technologies," *Sensors*, vol. 21, no. 10, pp. 1–25, 2021.

[2] H. Rajaei and F. Mirzaei, "IoT, smart homes, and zigbee simulation," *Simul. Ser.*, vol. 50, no. 3, pp. 77–86, 2018.

[3] R. Amadio, A. Isgandarova, and D. Mazzei, "Building a taxonomony of industry 4.0 needs and enabling technologies," EasyChair Preprint, 2021, no. 5621. [Online]. Available: https://easychair.org/publications/preprint/WJtF

[4] S. Hamdan, M. Ayyash, and S. Almajali, "Edge-computing architectures for applications: A survey," *Sensors*, vol. 20, no. 22, pp. 1–52, 2020.

[5] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, and B. Varghese, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1–12, 2019.

[6] *Multi-Access Edge Computing (MEC); Framework and Reference Architecture*, Standard GS MEC 003, ETSI, 2022.

[7] *Cloud RAN and MEC: A Perfect Pairing*, Standard 23, ETSI, 2018.

[8] X. Li, C. Guimarães, G. Landi, J. Brenes, J. Mangues-Bafalluy, J. Baranda, D. Corujo, V. Cunha, J. Fonseca, J. Alegria, A. Z. Orive, J. Ordonez-Lucena, P. Iovanna, C. J. Bernardos, A. Mourad, and X. Costa-Pérez, "Multi-domain solutions for the deployment of private 5G networks," *IEEE Access*, vol. 9, pp. 106865–106884, 2021.

[9] ORAN Alliance. (2021). *O-RAN Minimum Viable Plan and Acceleration Towards Commercialization*. Accessed: Feb. 6, 2023. [Online]. Available: https://www.o-ran.org

[10] ORAN Alliance. (2020). *O-RAN Use Cases and Deployment Scenarios: Towards Open and Smart RAN*. Accessed: Feb. 6, 2023. [Online]. Available: https://www.o-ran.org/resources

[11] M. Varela, L. Skorin-Kapov, and T. Ebrahimi, "Quality of service versus quality of experience," in *Quality of Experience* (T-Labs Series in Telecommunication Services), S. Möller and A. Raake, Eds. Cham, Switzerland: Springer, 2014, pp. 85–96.

[12] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: Modeling techniques and their applications," *J. Internet Services Appl.*, vol. 5, no. 1, pp. 1–17, Dec. 2014.

[13] G. N. Nedeltcheva, A. De La Fuente Ruiz, L. O. E. Arrieta, N. Bat, and L. Blasi, "Towards supporting the generation of infrastructure as code through modelling approaches—Systematic literature review," in *Proc. IEEE 19th Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2022, pp. 210–217.

[14] E. Chirivella-Perez, J. M. A. Calero, Q. Wang, and J. Gutiérrez-Aguado, "Orchestration architecture for automatic deployment of 5G services from bare metal in mobile edge computing infrastructure," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–18, Nov. 2018.

[15] *ETSI MEC: An Introduction*, ETSI, Sophia Antipolis, France, 2022.

[16] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "ETSI white paper #11 mobile edge computing—A key technology towards 5G," Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Tech. Rep., 11, 2015.

[17] J. Wiersma, "Cloud and edge computing," in *Data Center Handbook*. Hoboken, NJ, USA: Wiley, 2021.

[18] J. Guerreiro, L. Rodrigues, and N. Correia, "Allocation of resources in SAaaS clouds managing thing mashups," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 3, pp. 1597–1609, Sep. 2020.

[19] J. Guerreiro, L. Rodrigues, and N. Correia, "Resource allocation model for sensor clouds under the sensing as a service paradigm," *Computers*, vol. 8, no. 1, p. 18, Feb. 2019.

[20] K. M. Giannoutakis, M. Spanopoulos-Karalexidis, C. K. F. Papadopoulos, and D. Tzovaras, "Next generation cloud architectures," in *The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing*, T. Lynn, J. G. Mooney, B. Lee, and P. T. Endo, Eds. Cham, Switzerland: Springer, 2020.

[21] M. A. Calles, *Serverless Security: Understand, Assess, and Implement Secure and Reliable Applications in AWS, Microsoft Azure, and Google Cloud*. New York, NY, USA: Apress, 2020.

[22] R. Zhu, L. Liu, H. Song, and M. Ma, "Multi-access edge computing enabled and novel applications," *Neural Comput. Appl.*, vol. 32, no. 19, pp. 15313–15316, Oct. 2020.

[23] B. Attanasio, A. Mazayev, S. D. Plessis, and N. Correia, "Cognitive load balancing approach for 6G MEC serving IoT mashups," *Mathematics*, vol. 10, no. 1, p. 101, Dec. 2021.

[24] Z. Wen, K. Yang, X. Liu, S. Li, and J. Zou, "Joint offloading and computing design in wireless powered mobile-edge computing systems with full-duplex relaying," *IEEE Access*, vol. 6, pp. 72786–72795, 2018.

[25] Á. Santos, N. Correia, and J. Bernardino, "On the suitability of cloud models for MEC deployment purposes," in *Proc. 6th Exp. Int. Conf.*, 2023, pp. 1–12.

[26] F. Spinelli and V. Mancuso, "Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 1, pp. 596–630, 1st Quart., 2021.

[27] S. K. Sharma, I. Woungang, A. Anpalagan, and S. Chatzinotas, "Toward tactile internet in beyond 5G era: Recent advances, current issues, and future directions," *IEEE Access*, vol. 8, pp. 56948–56991, 2020.

[28] *Multi-Access Edge Computing (MEC): Study on Inter-MEC Systems and MEC-Cloud Systems Coordination*, Standard MEC 035, ETSI, 2021.

[29] ETSI. *Application Mobility Service API*. Accessed: Feb. 26, 2023. [Online]. Available: https://forge.etsi.org/rep/mec/gs021-amsi-api

[30] *Network Functions Virtualisation (NFV). Architectural Framework*, Standard GS NFV 002, ETSI, 2005.

[31] G. Lee, "Software-defined networking," in *Cloud Networking*, G. Lee, Ed. Boston, MA, USA: Morgan Kaufmann, 2014.

[32] Red Hat. *RedHat Ansible—Automation for Everyone*. Accessed: Feb. 26, 2023. [Online]. Available: https://ansible.com

[33] PaaSage Consortium. *Cloud Application Modelling and Execution Language (CAMEL)*. Accessed: Feb. 26, 2023. [Online]. Available: https://camel-dsl.org/

[34] A. P. Achilleos, K. Kritikos, A. Rossini, G. M. Kapitsaki, J. Domaschka, M. Orzechowski, D. Seybold, F. Griesinger, N. Nikolov, D. Romero, and G. A. Papadopoulos, "The cloud application modelling and execution language," *J. Cloud Comput.*, vol. 8, no. 1, p. 20, Dec. 2019.

[35] PaaSage Consortium. *PaaSage*. [Online]. Available: https://paasage.ercim.eu/

[36] K. Kritikos, J. Domaschka, and A. Rossini, "SRL: A scalability rule language for multi-cloud environments," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 1–9.

[37] A. Rossini, K. Kritikos, N. Nikolov, J. Domaschka, F. Griesinger, and D. Seybold. (Dec. 2016). *CAMEL Documentation*. PaaSage Consortium. [Online]. Available: https://gitlab.ow2.org/paasage/camel/raw/master/documents/CAMELDocumentation.pdf

[38] Progress. *Chef Software DevOps Automation Solutions*. Accessed: Feb. 26, 2023. [Online]. Available: https://www.chef.io/

[39] Progress. *Chef Documentation*. Accessed: Feb. 26, 2023. [Online]. Available: https://docs.chef.io/

[40] OpenStack. *Heat Documentation*. Accessed: Feb. 26, 2023. [Online]. Available: https://docs.openstack.org/heat/latest/

[41] OpenStack. *Heat Orchestration Template (HOT) Guide*. Accessed: Feb. 26, 2023. [Online]. Available: https://docs.openstack.org/heat/latest/template_guide/hot_guide.html

[42] Perforce. *Puppet—Infrastructure & IT Automation at Scale*. Accessed: Feb. 26, 2023. [Online]. Available: http://www.puppet.com

[43] Perforce. *Writing Plans in YAML*. Accessed: Feb. 26, 2023. [Online]. Available: https://www.puppet.com/docs/bolt/latest/writing_yaml_plans.html

[44] Perforce. *Puppet Forge*. Accessed: Feb. 26, 2023. [Online]. Available: https://forge.puppet.com/

[45] Perforce. *Puppet Overview*. Accessed: Feb. 26, 2023. [Online]. Available: https://www.puppet.com/docs/puppet/7/puppet_overview.html

[46] HashiCorp. *Automate Infrastructure on Any Cloud With Terraform*. Accessed: Feb. 27, 2023. [Online]. Available: https://www.terraform.io/

[47] HashiCorp. *What is Terraform?* Accessed: Feb. 27, 2023. [Online]. Available: https://developer.hashicorp.com/terraform/intro

[48] HashiCorp. *Terraform Registry*. Accessed: Feb. 27, 2023. [Online]. Available: https://registry.terraform.io/

[49] HashiCorp. *HCL Native Syntax Specification*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md

[50] OASIS. *OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC*. Accessed: Feb. 27, 2023. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[51] *TOSCA Simple Profile in YAML Version 1.3*, Org. Adv. Struct. Inf. Standards, Woburn, MA, USA, 2020.
[52] OASIS. (Jan. 2023). *TOSCA Version 2.0 Committee Specification Draft 05. Organization for the Advancement of Structured Information Standards*. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.html
[53] *TOSCA—Topology and Orchestration Specification for Cloud Applications Version 1.0*, Org. Adv. Struct. Inf. Standards, Woburn, MA, USA, 2013.
[54] XLAB. *xOpera*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/xlab-si/xopera-opera
[55] XLAB. *xOpera Documentation*. Accessed: Feb. 27, 2023. [Online]. Available: https://xlab-si.github.io/xopera-docs/
[56] Cloudify. *Cloudify DevOps Automation & Orchestration Platform*. Accessed: Feb. 27, 2023. [Online]. Available: https://cloudify.co/
[57] Ubicity Co. *Ubicity Orchestrator*. Accessed: Feb. 27, 2023. [Online]. Available: http://www.ubicity.com/products.html
[58] T. Liron and P. Jordan. *Turandot*. Accessed: Feb. 27, 2023. [Online]. Available: https://github.com/tliron/turandot
[59] D. Fraunholz, R. Schörghofer-Vrinssen, H. König, W. Mühlbauer, and R. Zahoranksy, "Mobility-enabling edge cloud infrastructure: Testbed and experimental evaluation," in *Proc. IEEE Cloud Summit*, Oct. 2021, pp. 19–24.
[60] F. Barbarulo, C. Puliafito, A. Virdis, and E. Mingozzi, "Enabling application relocation in ETSI MEC: A container-migration approach," in *Proc. IEEE 33rd Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2022, pp. 1–6.
[61] F. Barbarulo, C. Puliafito, A. Virdis, and E. Mingozzi, "Extending ETSI MEC towards stateful application relocation based on container migration," in *Proc. IEEE 23rd Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2022, pp. 367–376.
[62] N. Slamnik-Kriještorac, H. C. C. de Resende, C. Donato, S. Latré, R. Riggio, and J. Marquez-Barja, "Leveraging mobile edge computing to improve vehicular communications," in *Proc. IEEE 17th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2020, pp. 1–4.
[63] J. Santa, J. Ortiz, P. J. Fernandez, M. Luis, C. Gomes, J. Oliveira, D. Gomes, R. Sanchez-Iborra, S. Sargento, and A. F. Skarmeta, "MIGRATE: Mobile device virtualisation through state transfer," *IEEE Access*, vol. 8, pp. 25848–25862, 2020.
[64] M. M. A. Muthanna, V. Nikolayevich, A. Volkov, and K. Abdukodir, "Approaches for multi-tier cloud structure management," in *Proc. 11th Int. Congr. Ultra Modern Telecommun. Control Syst. Workshops (ICUMT)*, Oct. 2019, pp. 1–7.
[65] R. Riggio, S. N. Khan, T. Subramanya, I. G. B. Yahia, and D. Lopez, "LightMANO: Converging NFV and SDN at the edges of the network," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2018, pp. 1–9.
[66] H.-T. Chien, Y.-D. Lin, C.-L. Lai, and C.-T. Wang, "End-to-end slicing with optimized communication and computing resource allocation in multi-tenant 5G systems," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2079–2091, Feb. 2020.
[67] C. Parada, F. Fontes, C. Marques, V. Cunha, and C. Leitão, "Multi-access edge computing: A 5G technology," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2018, pp. 277–279.
[68] T. Doan-Van, A. Kropp, G. T. Nguyen, H. Salah, and F. H. P. Fitzek, "Programmable first: Automated orchestration between MEC and NFV platforms," in *Proc. 16th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2019, pp. 1–2.
[69] T. V. Doan, A. Kropp, G. T. Nguyen, H. Salah, and F. Frank, "Reusing sub-chains of network functions to support MEC services," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2019, pp. 1–8.
[70] L. Nkenyereye, J. Hwang, Q.-V. Pham, and J. Song, "Virtual IoT service slice functions for multiaccess edge computing platform," *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11233–11248, Jul. 2021.
[71] S. Bolettieri, D. T. Bui, and R. Bruno, "Towards end-to-end application slicing in multi-access edge computing systems: Architecture discussion and proof-of-concept," *Future Gener. Comput. Syst.*, vol. 136, pp. 110–127, Nov. 2022.
[72] L. T. Bolivar, C. Tselios, D. Mellado Area, and G. Tsolis, "On the deployment of an open-source, 5G-aware evaluation testbed," in *Proc. 6th IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Mar. 2018, pp. 51–58.
[73] M.-Y. Wu, J.-C. Huang, Y.-M. Hung, C.-Y. Chien, J. S. Luo, and S.-P. Liang, "The edge cloud implementation and application of transnational smart factory of 5G private network," in *Proc. 23rd Asia–Pacific Netw. Operations Manage. Symp. (APNOMS)*, Sep. 2022, pp. 1–6.
[74] T. Dreibholz, S. Mazumdar, F. Zahid, A. Taherkordi, and E. G. Gran, "Mobile edge as part of the multi-cloud ecosystem: A performance study," in *Proc. 27th Euromicro Int. Conf. Parallel, Distrib. Network-Based Process. (PDP)*, Feb. 2019, pp. 59–66.
[75] Á. Leiter and L. Bokor, "A study on use cases and business aspects of cloud stock exchange," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2020, pp. 732–737.
[76] J. Hammer and H. Hellwagner, "Efficient transparent access to 5G edge services," in *Proc. IEEE 8th Int. Conf. Netw. Softwarization (NetSoft)*, Jun. 2022, pp. 91–96.
[77] D. Santos, R. Silva, D. Corujo, R. L. Aguiar, and B. Parreira, "Follow the user: A framework for dynamically placing content using 5G-enablers," *IEEE Access*, vol. 9, pp. 14688–14709, 2021.
[78] T. Taleb, P. A. Frangoudis, I. Benkacem, and A. Ksentini, "CDN slicing over a multi-domain edge cloud," *IEEE Trans. Mobile Comput.*, vol. 19, no. 9, pp. 2010–2027, Sep. 2020.
[79] *Study on MEC Support for Alternative Virtualization Technologies*, Standard GR MEC 027, ETSI, 2019.
[80] L. Roda-Sanchez, C. Garrido-Hidalgo, F. Royo, J. L. Maté-Gómez, T. Olivares, and A. Fernández-Caballero, "Cloud–edge microservices architecture and service orchestration: An integral solution for a real-world deployment experience," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100777.

ÁLVARO SANTOS received the B.Sc. degree in informatics engineering and the M.Sc. degree in systems and information technologies from the University of Coimbra, Portugal, in 1994 and 2000, respectively. He is currently pursuing the Ph.D. degree in informatics engineering with the University of Algarve, Portugal. From 1995 to 1999, he was a Teaching Assistant with the Polytechnic Institute of Coimbra, where he has been an Adjunct Professor, since 2000. He has taught several curricular units, such as artificial intelligence, operating systems, network services (he has a CCNA certification), advanced programming, mobile architectures, and mobile programming. He has more than 25 publications in refereed conferences, journals, and book chapters. His research interests include the IoT, cloud computing, edge computing, mobile computing, cross-platform solutions, and software design patterns.

JORGE BERNARDINO (Member, IEEE) received the Ph.D. degree from the University of Coimbra, in 2002. From 2005 to 2010, he was the President of the Coimbra Engineering Institute (ISEC). From 2017 to 2019, he was also the President of ISEC Scientific Council. In 2014, he was a Visiting Professor with CMU. He was the Director of the Applied Research Institute (i2A), Polytechnic of Coimbra, from 2019 to 2021. He is currently a Coordinator Professor with the Polytechnic of Coimbra-ISEC, Portugal. He has authored more than 200 publications in refereed conferences and journals and participated in several national and international projects. His research interests include big data, NoSQL, data warehousing, dependability, the Internet of Things, and software engineering.

NOÉLIA CORREIA received the B.Sc. and M.Sc. degrees in computer science from the University of Algarve, Faro, Portugal, in 1995 and 1998, respectively, and the Ph.D. degree in optical networks (computer science) from the University of Algarve, in 2005, in collaboration with University College London, U.K. She is currently a Lecturer with the Science and Technology Faculty, University of Algarve. Her research interests include the application of optimization techniques to several network design problems, in the optical, wireless, sensor networks, and the IoT fields, and development of algorithms. She is a Founding Member of the Center for Electronics, Optoelectronics and Telecommunications, University of Algarve, a research center supported by the Portuguese Foundation for Science and Technology. She is also the Networks and Systems Group Coordinator.