Journal of Cloud Computing:
Advances, Systems and Applications

REVIEW

Open Access

Orchestration in the Cloud-to-Things compute continuum: taxonomy, survey and future directions

Amjad Ullah[1,2*], Tamas Kiss[2], József Kovács[2,3], Francesco Tusa[2,4], James Deslauriers[2], Huseyin Dagdeviren[2], Resmi Arjun[2] and Hamed Hamzeh[2]

Abstract

IoT systems are becoming an essential part of our environment. Smart cities, smart manufacturing, augmented reality, and self-driving cars are just some examples of the wide range of domains, where the applicability of such systems have been increasing rapidly. These IoT use cases often require simultaneous access to geographically distributed arrays of sensors, heterogeneous remote, local as well as multi-cloud computational resources. This gives birth to the extended Cloud-to-Things computing paradigm. The emergence of this new paradigm raised the quintessential need to extend the orchestration requirements (i.e., the automated deployment and run-time management) of applications from the centralised cloud-only environment to the entire spectrum of resources in the Cloud-to-Things continuum. In order to cope with this requirement, in the last few years, there has been a lot of attention to the development of orchestration systems in both industry and academic environments. This paper is an attempt to gather the research conducted in the orchestration for the Cloud-to-Things continuum landscape and to propose a detailed taxonomy, which is then used to critically review the landscape of existing research work. We finally discuss the key challenges that require further attention and also present a conceptual framework based on the conducted analysis.

Keywords Cloud-to-Edge continuum, Cloud-to-Things continuum, Fog computing, Edge computing, IoT application, Microservices, Application orchestration, Orchestration, Resource management

Introduction

The advent of cloud computing has reshaped the way in which software is developed, deployed and used. Since its inception, the adoption of cloud services has continually increased. This is evident from the worldwide public cloud service revenue growth of 33 %, from 266.4 billion dollars in 2020 to 354.6 billion dollars in 2022 [1]. This increased shift towards cloud computing is due to its inherent characteristics, such as on-demand provisioning, pay-as-you-go utility model and elasticity, which offer economic benefits as well as operational efficiencies to enterprises [2].

In order to fully exploit the strength of cloud computing, effective and optimised usage of the associated computing resources is important. This is the responsibility of an orchestration system. More formally, an orchestration system automates the seamless delivery of applications over clouds, and guarantees various Quality of Service (QoS) goals, by handling the required complex tasks of

*Correspondence:
Amjad Ullah
a.ullah@napier.ac.uk
[1] School of Computing, Engineering & the Built Environment, Edinburgh Napier University, Edinburgh, UK
[2] School of Computer Science & Engineering, University of Westminster, London, UK
[3] Institute for Computer Science and Control (SZTAKI), Eötvös Loránd Research Network (ELKH), Budapest, Hungary
[4] Department of Electronic and Electrical Engineering, University College London, London, UK

resource selection, deployment, monitoring, and run-time control of the resources and applications [3].

In the last decade, cloud orchestration has become a mature research area and there emerged a large number of orchestration solutions. These include vendor-specific solutions, such as Amazon's AWS Cloud-Formation [4], OpenStack HEAT [5], Microsoft Azure's Resource Manager (ARM) templates [6], and Google's Deployment Manager [7]; some Open-source cloud agnostic initiatives, such as Kubernetes [8], Docker Swarm [9], Apache Brooklyn [10], Cloudify [11], Cloudiator [12], Alien-4Cloud [13], MODAClouds [14] and MiCADO [15]. The key purpose of all such tools is to improve resource utilisation and introduce a great deal of agility by making application development, deployment, execution and maintenance easier for cloud applications.

In recent years, the introduction of IoT has fuelled a new breed of applications, which in addition to cloud resources, also require IoT devices to capture and possibly process data from local environments. Such systems have a wide range of requirements in terms of low-latency analytics, data privacy and sensitivity, context awareness, time- and location- awareness, and simultaneous access to geographically distributed arrays of sensors, remote localised heterogeneous computational resources and to large-scale on-the-fly multi-cloud computational resources. A traditional cloud computing architecture is impractical, if not inadequate, to handle the aforementioned requirements. This gives rise to new computational paradigms such as fog computing, edge computing, and compute continuum.

The terms fog computing and edge computing are often used interchangeably to loosely refer to moving processing or computation away from the central cloud to nodes that are closer to endpoints at the network edge. Though they both aim to reduce the amount of data sent to the cloud in data-dense applications, there are subtle differences between the two. Fog computing is an intermediate layer between the cloud and edge that represents the nodes between the cloud to the IoT sensors and actuators, possibly spanning across multiple layers of the network topology. In contrast, in edge computing, the nodes where the computation takes place are normally very close to the IoT devices in terms of network proximity, often only one or a few hops away from the IoT devices, or even embedded within the connected device [16].

The compute continuum—also known by other names such as cloud continuum, cloud-edge continuum, cloud-to-edge continuum, or cloud-to-things continuum—on the other hand, refers to the extension of cloud with energy-efficient and low-latency devices closer to the data sources located at the network edge [17]. More specifically, it extends the traditional Cloud towards multiple entities such as Fog, Edge, and IoT to provide different capabilities including analysis, processing, storage, and data generation [18]. Our adoption of the term Cloud-to-Things is to indicate the notion that the continuum connects cloud(s) and the IoT-connected devices (i.e., things) [18], where we consider the 'things' mainly as a source of data that need to be processed in real-time using various layers of resources scattered across the continuum.

The emergence of these new paradigms raised the quintessential need to extend the orchestration requirements of applications from the centralised cloud-only environment to the entire spectrum of resources in the Cloud-to-Things continuum, as the existing cloud orchestration solutions are unable to address them. This mainly includes the application deployment and management to be performed in a more complex, heterogeneous and geographically distributed infrastructure, where resources are located across different layers of the continuum. More specifically, the following are some of the key challenges of orchestration in the Cloud-to-Things compute continuum [19–22]:

1 The Cloud-to-Things compute continuum is highly diverse, where resources are not only distributed across different layers of the spectrum but also heterogeneous having different architecture, operating systems, and computational capabilities. An orchestration system needs to provide seamless and simultaneous access to such a heterogeneous and decentralised resource landscape.

2 The federated coordination across different administrative domains to facilitate end-to-end services across different cloud, fog and edge providers is challenging. An orchestration system needs standardised APIs and interfaces to achieve such coordination.

3 A specific challenge to address in the case of edge nodes is to deal with volatility and mobility i.e., the nodes may shut down or lose connectivity or their locations may change. In such scenarios, the orchestration system needs to deal with resource fluctuation and changing environmental conditions.

4 Efficient monitoring mechanisms are required to collect the statuses of the workload and resource usage statistics across the entire spectrum of the continuum.

5 The implementation of efficient run-time mechanisms that enforce policy-based deployment and run-time reconfiguration of target applications to ensure that the system meets the SLA goals specified in the form of contextual configurations in terms of resource discovery, optimal placement, optimise

resource usage, efficient processing of data, and security aspects.

6 The scale of the compute continuum can be massive, where resources can be gathered from different cloud and edge providers to fulfil the needs of target applications. An orchestration system needs to deal with the required level of scalability across the different administrative domains.
7 Lastly, an orchestration system is required to guarantee the security of the overall system against different attack scenarios while minimising the need for user-supplied configurations. This is particularly challenging in the Cloud-to-Things continuum due to the heterogeneity of the resources involved and the possibility of their belonging to different administrative domains.

To deal with the above-mentioned challenges, there has recently been a lot of attention, both in industry and academia, to the development of Cloud-to-Things Orchestration Solutions (CoTOS). This paper is an attempt to gather, analyse and synthesis the research work conducted in the field of orchestration systems for the Cloud-to-Things continuum. The key contributions of this paper are as follows:

1 We identified a wide range of key characteristics in relation to the orchestration of IoT applications in the Cloud-to-Things computing continuum. These characteristics are the essential ingredients of, and therefore, important for the evaluation of CoTOS. Using these characteristics, a novel taxonomy of CoTOS is proposed, which is vital for the understanding and analysis of existing solutions.
2 We performed a thorough review covering a wide range of existing orchestration solutions from industry and academia that target the Cloud-to-Things continuum. The entire landscape of existing CoTOS is classified into different logical groups, and a detailed consolidated review and analysis of each group is performed in light of the proposed taxonomy.
3 Based on the results obtained from the review, we identified and discussed the key issues and gaps in the existing landscape of orchestration solutions to highlight future research directions.
4 Lastly, we proposed a conceptual architecture of a novel and comprehensive orchestration framework as a reference to alleviate the identified gaps.

The rest of this paper is structured as follows. Section "Related work" discusses the existing related review papers to highlight the gaps and motivations in order to justify the need for conducting yet another review. Section "Taxonomy" presents our proposed taxonomy and explains each of the included characteristics. A thorough review of existing orchestration solutions, using the proposed taxonomy, is carried out in Section "Review of existing CoTOS". Section "Discussion, Issues, and Future directions" further reflects on the summarised results to identify key issues and gaps from the review and to highlight research directions for the future. Section "A conceptual framework of orchestration in the Cloud-to-Things compute continuum", presents and discusses a conceptual framework that can be used as a reference for future implementations of orchestration solutions. Lastly, Section "Conclusion" concludes this paper.

Related work

This section discusses the most relevant review papers from the Cloud-to-Things orchestration domain, with a view to analyse their strengths and weaknesses and highlight how they differ from the review carried out in this work.

The most relevant studies related to the Cloud-to-Things orchestration include [19, 23, 24]. The authors in these papers have identified and discussed the target application scenarios and key challenges, in order to derive requirements that can be used for the design of a Cloud-to-Things orchestration solution. Based on these requirements, a detailed evaluation and analysis of some of the existing reference architectures and fog orchestration solutions have been provided. However, the list is not exhaustive and the authors, except in [19], have only covered a very small number of solutions. Furthermore, all these studies lack a detailed taxonomy.

Similarly, papers [25–28] also identified and discussed the core issues and challenges related to the orchestration of IoT applications. The focus though in Karima, et al. [28] is in the context of 5G (and beyond) networks. However, none of these papers provided a detailed taxonomy nor carried out a detailed review of existing orchestration solutions.

The authors in [29] produced a systematic review of the deployment and orchestration approaches for the IoT. Their proposed taxonomy consists of the following three categories: (1) deployment and orchestration support, (2) specification, and (3) advance prospects (as defined by authors), i.e., monitoring, parameter adaptation, and trustworthiness features. As such, the authors used these very high-level characteristics only to classify the available solutions. In contrast, in this paper, we consider deployment and run-time management of IoT applications as the two essential key ingredients of an orchestration solution. Based on this notion, we used them as fundamental categories in our taxonomy. We further identified a large number of detailed, lower-level characteristics associated with these key ingredients to be part of the taxonomy. As a result, we review the existing approaches in light of these essential characteristics rather than the aforementioned high-level categories. Lastly, we also used additional aspects to classify the available approaches into different categories to provide a detailed comparative analysis and review of the overall spectrum of existing orchestration solutions.

The review by Wu in [30] is mainly focused on the aspects related to architecture and AI-powered data processing techniques. The architecture was discussed in the context of an underlying communication infrastructure, such as the industrial network, mobile and vehicular networks; whereas, the data processing techniques are categorised and discussed based on the various functions from the orchestration viewpoint, such as Offloading, Placement, and Resource management. The scope of [30] is on the different possible underlying architectures for the IoT ecosystem and the data processing techniques used by the applications. This paper, in contrast to [30], aims to perform a critical review of existing orchestration solutions.

Vaquero et al. [31] carried out an interesting and comprehensive review related to the challenges of next-generation service orchestration, where they focused on how the emergence of new technology trends such as Network Function Virtualisation (NFV), Software Defined Networking (SDN), Fog/Edge computing, and Serverless computing have changed requirements for the orchestration of microservices. Using the identified requirements, the authors further reviewed and discussed the state-of-the-art techniques by classifying them based on the implementation aspects, such as Machine learning techniques, P2P/Agent-based, Hierarchical and no orchestration. In contrast, our paper focused on the review of existing solutions with respect to the key functions of orchestration rather than their underlying implementation techniques.

The review in [32, 33] mainly focused on Kubernetes-based orchestration architectures that have been used within the context of the smart-city domain. Their key focus is on identifying the fundamental requirements for edge orchestration, analysis of existing kubernetes-based architectures and in general the evaluation of Kuberentes as the suitable candidate for cloud-edge orchestration. The authors further reviewed and discussed the state-of-the-art Kubernetes architectures by classifying them mainly into three categories including frameworks that realise edge orchestration, solutions that implement custom modifications and extensions to Kubernetes, and solutions that only deal with edge layer using customised Kubernetes. All these categories in our paper are captured through only one category, titled Lower level (Please see Section "Review of existing CoTOS" for further details). Furthermore, we also include a range of other categories to cover the entire spectrum of cloud-to-edge orchestration solutions. Lastly, our paper provides a detailed taxonomy for cloud-to-edge orchestration, where the scope is also not limited to the smart city domain.

Fakude et al. [34] and Šatkauskas et al. [35] have discussed fog/edge orchestration from the viewpoint of security in fog-enabled IoT-based computing environments. However, neither of these studies are detailed and only discusses a small number of existing works from the perspective of various security challenges. The focus in both studies is on the identification of security-oriented challenges related to fog orchestration. In the same realm, Al-Doghman [36] focused on highlighting the challenges of IoT management and secure decision-making at the edge for AI-based Microservices. All these studies, in comparison to our paper, do not provide a detailed taxonomy, formal classification and detailed analysis of existing orchestration solutions. Lastly, there is no discussion on identifying research gaps and future research directions.

Besides the above-mentioned studies, there are a number of considerably extensive review papers such as [37–42] that have discussed the resource management related research works. The focus of these papers is on the classification of approaches that relate to resource management using different viewpoints. For example, the taxonomy proposed in [40] relies on the classification based on the core functions such as application placement, resource scheduling, offloading. Similarly, Luo et al. [41] focused on the core issues of computation offloading, resource allocation, and resource provisioning. On the other hand, the authors in [39] use a set of criteria consisting of four points (type of resource, objective, resource location, and usage) to classify the available research work, where Duc et al. [42] reviewed machine learning techniques for resource provisioning in Edge-Cloud environment. All these papers provide a consolidated view of the available literature in the fog/edge computing area from a resource management point of view. In all these papers, there are either no or very limited attention provided to the automated orchestration of applications and resources.

In contrast to the above-mentioned related works, the research works in [3, 43–46] focus specifically on orchestration. However, their scope is only limited to the cloud environment and does not cover the Cloud-to-Things ecosystem, as it is done in this paper. In terms of structure, our work extends and complements the taxonomies proposed in the aforementioned cloud orchestration review papers.

To summarise, in contrast to the related works, our scope is on the overall key functions of orchestration rather than the underlying implementation techniques. As a result, this paper presents a detailed taxonomy of relevant characteristics, features, and dimensions related to the Cloud-to-Things orchestration. This taxonomy is further used as a unified framework to evaluate and perform a thorough analysis of existing orchestration solutions.

Taxonomy

We identified a widerange of characteristics in relation to the orchestration of IoT applications in the Cloud-to-Things computing continuum using the various studies discussed in Section "Related work", literature review of target orchestration solutions, as well as our own experience of implementing an application-level cloud orchestration solution called MiCADO [15] and a CoTOS called MiCADO-Edge [22]. The identified characteristics of the taxonomy represent the essential ingredients of a CoTOS and therefore are important to be considered from an implementation viewpoint. Figure 1 presents the proposed taxonomy, where the identified characteristics are structured and summarised under five main categories.

The categorisation of attributes enabled us to demystify the concept and scope of orchestration for the continuum. However, all attributes are relevant to the entire orchestration solution and are not part of a specific type of resource environment, i.e., cloud or edge. The overall purpose of this taxonomy is to provide a unified framework whereby all candidate solutions can be objectively compared and evaluated. A brief description of each category and of their associated characteristics is provided in the following subsections.

Cloud resource handling

This category groups together the characteristics related to the cloud infrastructure part of the orchestration system and includes the three aspects described below.

Environment This attribute refers to the underlying support of a CoTOS for cloud environment(s) in terms of the ability to use or combine resources from different cloud providers. The possibilities include *Single cloud*, where a CoTOS only supports a single specific cloud environment; *Multi-cloud*, where a CoTOS facilitates the selection of suitable resources from multiple cloud environments, however, only one is utilised at a time; and *Cross-cloud*, where multiple cloud environments are exploited simultaneously to allow the distribution of components belonging to the same application across

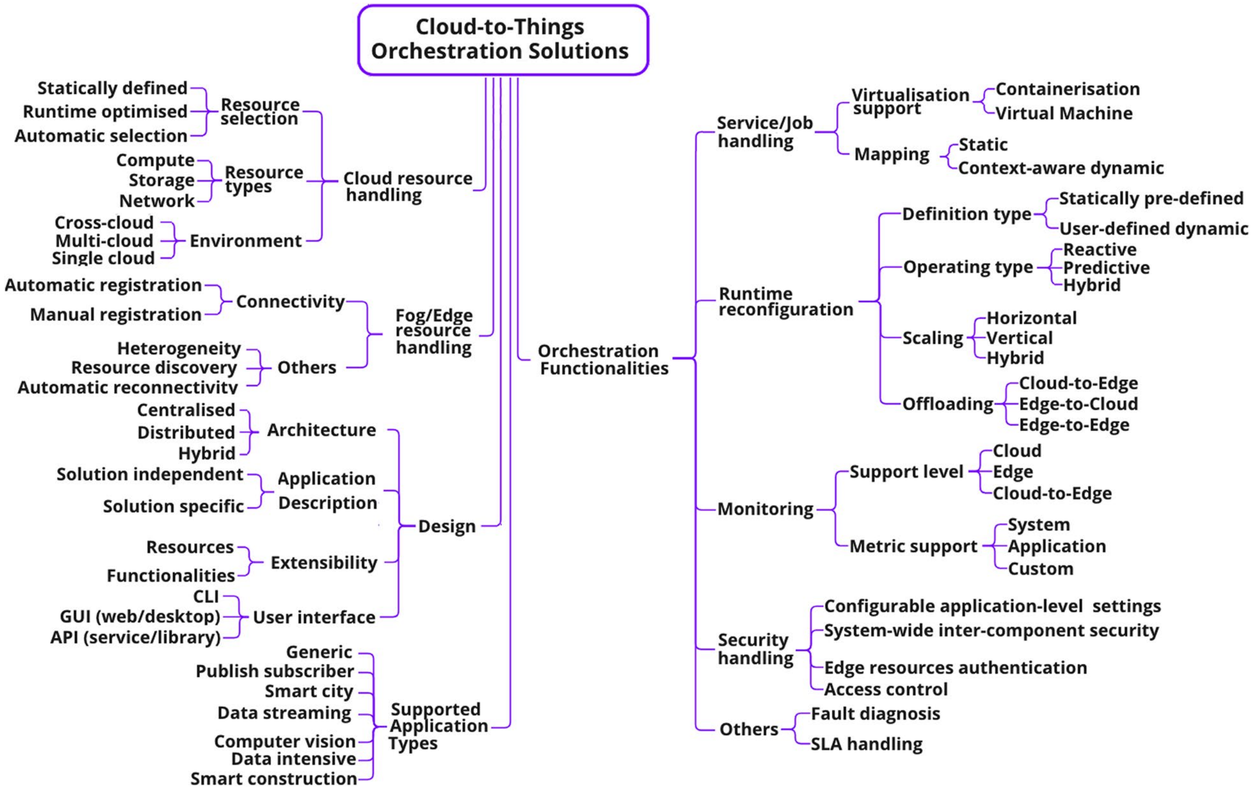

Fig. 1 Taxonomy of Cloud-to-Things orchestration

different cloud providers. The use of multi and cross-cloud features is of particular importance as they help in optimising cost and performance by allowing the selection of suitable offers. This is also important to avoid vendor lock-in. Furthermore, it also helps in addressing privacy issues by allowing the use of specific cloud providers or private clouds for certain application components [3].

Resource types The support of a CoTOS in relation to the different types of resources that can be dynamically controlled by the orchestration solution. The choices include the three common utility services provided by cloud providers, i.e., compute, storage, and network.

Resource selection The support of a CoTOS that determines how the resources are selected during the deployment process. The available choices include *Statically defined*, where specific instances of resources are statically assigned by the application owner at the time of deployment; *Automatic selection*, where the application owner specifies the general resource characteristics and the CoTOS automatically selects the suitable instances at run-time, however, the selection does not change at run-time; *Runtime optimised*, where the CoTOS automatically chooses the suitable resources from a diverse range of cloud vendors based on certain specific optimisation criteria, e.g., cost, locality.

Fog/Edge resource handling

This category groups together the characteristics related to the handling of resources from fog and edge (referred to as "non-cloud" collectively hereafter). More particularly, this covers the key aspects described in the following.

Heterogeneity The Cloud-to-Things compute continuum is highly heterogeneous, namely computational devices of different natures are usually required to support the requirements of an IoT application. This attribute will measure the support of CoTOS for device heterogeneity.

Connectivity In the Cloud-to-Things scenario, both dynamically created cloud resources (e.g., VMs) and non-cloud physical ones are available. Therefore, the CoTOS has to provide a mechanism that enables the connectivity/registration of these non-cloud resources to a resource pool, such that they can be utilised for the deployment as per the requirements of the IoT application. In this regard, the Connectivity attribute analyses the underlying support of a CoTOS that enables the connection of non-cloud resource elements to the pool of resources. This support can be further classified into two categories: i.e., *Manual registration*, where the CoTOS facilitates users through some manual pre-defined procedure that allows the registration of non-cloud resources with the CoTOS prior to the deployment process; or *Automatic registration*, where the CoTOS provides automatic procedure that allows the registration of non-cloud resources at run-time, even after the deployment process.

Automatic re-connectivity Non-cloud resources can be volatile in nature due to a number of reasons (e.g., low-powered computational devices, mobility, network connection), where they may lose connectivity to the rest of the system at different points in time. In this regard, the attribute refers to the ability (or not) of a CoTOS to support automatic re-connectivity of a non-cloud resource.

Resource discovery As resources in the Cloud-to-Things continuum are geographically distributed, it can be important for a CoTOS to support discovering all the available resources. Resource discovery refers to the ability of a CoTOS to support optimal re-configuration decisions by finding the most suitable resources based on certain contextual requirements.

Orchestration functionalities

This category groups together the essential functions of a CoTOS.

Service/Job handling This attribute determines the mechanism related to the deployment and management of services (or jobs in the case of batch-based applications). This can be further subdivided into the two aspects reported next.

1 Virtualisation support: this attribute can either refer to *Virtual Machines (VMs)* to indicate that the CoTOS features the dynamic provision of VMs and the ability of direct deployment and management of application components on VMs without the use of containers; and *Containerisation* to indicate that the CoTOS provides support for the deployment and management of application components through containers.
2 Mapping: The mapping mechanism of application components to Cloud-to-Things resources can either be *Static*, where the application owner statically configures application components to the avail-

able resources (or resource types); or *Context-aware dynamic*, where the mapping is dynamically determined based on various user-defined contextual conditions associated with the application components and/or resources.

Run-time reconfiguration One of the key functions of any orchestration solution is its adaptability at run-time using reconfiguration of application components and associated resources according to the changing working conditions. We classify the *Run-time reconfiguration* based on the list of attributes reported below.

1 Definition type: This represents the nature of the available reconfiguration functions. It can be one of the following two types: *Statically pre-defined*, where a set of reconfiguration policies already exists and the application owners are restricted to provide threshold values based on some already established criteria; or *User-defined dynamic*, where the application owners have the freedom to write their own policies based on available system and/or application-level metrics.
2 Operating type: This represents the triggering behaviour of the reconfiguration operation. There are three possible types: *Reactive*, where the reconfiguration is performed as a response to some changes; *Proactive*, where changes are anticipated and reconfiguration decisions are performed in advance; or *Hybrid*, meaning that the same solution consist of both reactive and proactive reconfiguration mechanisms.
3 Scaling: This attribute represents the automated scaling ability of a CoTOS. It can be of the following three types: *Horizontal*, where the number of additional resources is increased or decreased depending on the needs; *Vertical*, where the capacity of existing computational resources is increased or decreased; or *Hybrid*, where the system supports both Horizontal and Vertical scaling.
4 Offloading: This refers to the transfer of computational tasks from one execution device to another. For example, within the context of Cloud-to-Things compute continuum, services are offloaded from cloud to edge devices due to latency sensitivity and/or geo-distributed requirements. Similarly, a service request can be offloaded from edge to cloud or another edge device if the existing device can not fulfil the required computational capacity. Consequently, three types of offloading generally occur. These are Cloud-to-Edge, Edge-to-Cloud, and Edge-to-Edge.

Monitoring An orchestration system hugely relies on run-time monitoring through which information on the status of the system and of the application is gathered. The collected information is used to trigger run-time reconfiguration decisions in order to comply with system-stated objectives. Using this attribute, the monitoring support of a CoTOS can be evaluated in the following two aspects.

1 Level of support: To identify whether monitoring of application components and resources is possible at each layer of the continuum or not.
2 Metrics support: The provided support of a CoTOS in gathering different types of metrics. The different types include *system-level* (e.g., CPU/memory utilisation), *application-level* (e.g., number of active HTTP requests), as well as the ability to define *custom metrics* for collection (e.g., number of running jobs in a batch processing application).

Security handling A CoTOS is required to guarantee the security of the overall system against different attack scenarios while minimising the need for user-supplied configurations. Security handling in a CoTOS is a challenging task because an application in a Cloud-to-Things ecosystem typically runs on heterogeneous resources. Furthermore, these resources can contain low-powered devices that also operate in different administrative domains. Using this attribute, the support of CoTOS security handling will be evaluated according to the aspects reported below.

1 Configurable application level security settings: The support of a CoTOS that gives application owners the ability to define application level security settings (e.g., firewall setting, TLS/SSL certificate, ports configuration) in a configurable way.
2 System-wide inter-component communication: The internal function of a CoTOS that enables secure communication amongst the different parts of a system, i.e., between system components that may operate in different VMs (and/or different layers).
3 Edge resource authentication: An important function of a CoTOS is to facilitate the registration of non-cloud resources to the pool of resources that are then used for the deployment of application components. Such a registration process should be secure, where only authenticated non-cloud resources will be allowed to become part of the resources pool.

4 Access control: This attribute will evaluate the support of CoTOS functionality in relation to the access control that includes aspects like secure access to the system resources.

Fault diagnosis The support for the detection of system and/or application level faults at run-time, e.g., a fault in the cloud provider's system causing an unexpected termination of a VM, an unhandled run-time exception at application level forcing to stop a container, or a volatile edge node losing connection with the cluster.

Service Level Agreement (SLA) handling This attribute will evaluate the support of a CoTOS in relation to the handling of SLA-related functionalities, such as specification, enforcement, and negotiation.

Design

This category grouped together the design aspects of the orchestration solution. The following attributes are identified in what follows.

Architecture The employed architecture of a CoTOS influences how the overall system operates to perform the key orchestration functions, such as resource handling, application management, deployment and runtime reconfiguration decisions. This can be one of the following three types: *Centralised*, where a central entity, usually operating at the cloud layer, is responsible for all functions; *Decentralised*, where multiple system entities are running at different Cloud-to-Things continuum layers and handle various orchestration functions accordingly. *Hybrid*, where a combination of the centralised and decentralised approaches are employed by the CoTOS.

Extensibility The support provided by the design of a CoTOS for facilitating extension in terms of the addition of new resource providers, and the implementation of additional orchestration functions.

User interface The ways users can interact with the CoTOS. The possible types include *Graphical User Interface (GUI), Command Line Interface (CLI),* and *API-based Interfaces.*

Application description The orchestration solution usually provides application owners with a mechanism to provide the description of an application by expressing the specification of resources and components, the application topology, and any associated scaling and security policies. A number of well-known high-level description standards are available for this purpose, e.g., TOSCA [47]—an OASIS [48] standard for describing complex application topologies in the cloud. A standard TOSCA template in YAML defines the various components of a cloud application (software, storage, networks, virtual machines) as *nodes*, which may have *requirements* for, or share *relationships* with, other nodes in the template. TOSCA also supports *policies* for defining rules for scalability, monitoring, placement or security that will govern application behaviour at run-time.

The possible values are labelled as *Solution independent* to represent that the provided mechanism is based on some standard and is independent of the underlying solution; or *Solution specific* to represent that the provided mechanism is specifically designed for a given solution.

Supported application types

A CoTOS can be developed to target a specific application area. This attribute will evaluate a CoTOS in relation to its suitability with respect to particular application area/s. We treat this as an open-ended attribute, where specific application areas, such as *Data streaming, Computer vision, or Generic*, will be listed.

Review of existing CoTOS

This section presents a comprehensive review of existing CoTOS in light of the proposed taxonomy. The available landscape of orchestration solutions is very diverse, as subsets of solutions are quite different in nature from each other. Therefore, we first classified the landscape of existing solutions into different categories in order to reduce the overall complexity. The classification, on the one hand, allowed us to cover representative solutions from each category. On the other hand, this enabled us to perform a detailed comparative analysis of solutions that are closely related to each other and to cluster the results of each category. The overall hierarchy of these categories can be seen in Figure 2 and their brief description is reported below.

1 Lower level: This category represents those solutions that act as middleware, lacking a high-level abstraction layer, and often requiring the knowledge and configuration of underlying low-level technical details in relation to setting up the infrastructure resources to be used for the application deployment. Furthermore, these solutions, also do not provide core orchestration functions such as deployment and reconfiguration based on user-provided dynamic criteria. Hence, these solutions cannot be directly considered as cloud-to-things orchestrators. However, they are essential for higher-level application orches-

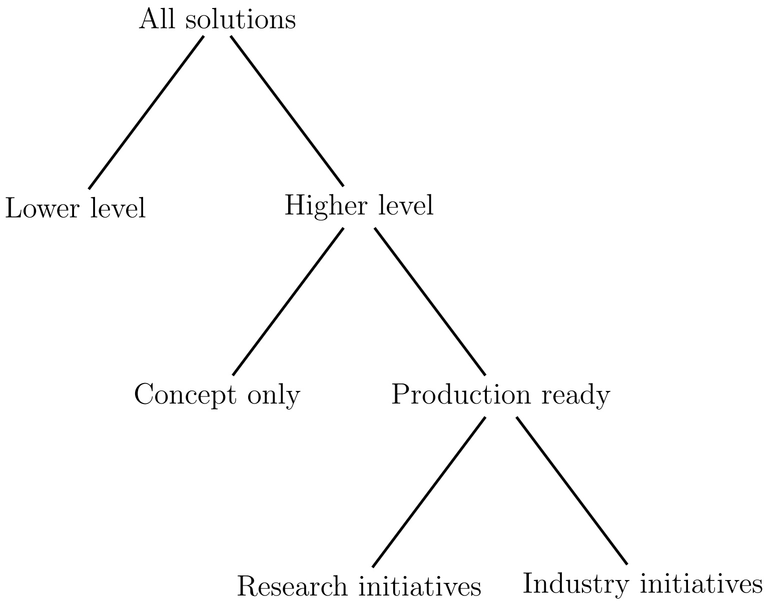

Fig. 2 Classification of existing orchestration solutions

trators to rely on as a middleware for the extension of orchestration capabilities to the edge.

2 Higher level: This category represents a subset of solutions that hides the underlying complexity of resource settings and management using a high-level abstraction layer. Such an abstraction layer can be provided using a GUI and/or some standardised specification language, e.g., TOSCA. Such solutions may rely on the use of some lower-level solution (further discussed in Section "Lower level solutions"). From the application owners' viewpoint, the higher-level solutions are of particular interest. However, from the viewpoint of orchestration solution developers, the lower-level solutions are also relevant when analysing and selecting technologies that can support their higher-level orchestrator. Therefore, both are included in our analysis for completeness, and as a way to equally support application owners and orchestration solution developers. The higher-level solutions are further classified based on their existing status, i.e., *Concept only*, which consists of those academic research proposals that only provide a conceptual framework and/or prototypical implementation only, and *Production ready*, which consists solutions that provide a fully working implementation. The *Production ready* solutions are further grouped into, Research initiatives that are developed as a result of some research projects, or Industry initiatives, where they are industry products and are commercially available.

In the following subsections, we respectively review and analyse a subset of relevant orchestration solutions from each of the above categories.

Lower level solutions

All major public cloud providers such as Amazon, Microsoft, Google and Alibaba, provide middleware solutions that enable application developers to combine their edge resources and to use them simultaneously with the respective cloud resources. Some examples of such solutions include AWS Greengrass [49], Azure IoT Edge [50], Google Distributed Cloud Edge [51], Alibaba Link IoT Edge [52], IBM Edge Application Manager [53], and Akamai EdgeWorkers [54]. These (and other similar) solutions have been developed with their respective cloud platforms in mind. Hence, they are not cross-platform solutions and cause a degree of vendor lock-in and therefore are not of particular interest for this review paper.

There also exist a number of vendor-agnostic middleware solutions that fall into this category. A subset of such solutions are discussed in this section. The key factors that led to the inclusion of these solutions include 1) the availability of their implementation, 2) their implementations' being regularly maintained, and 3) the presence of available technical documentation and/or associated research papers. It is important to note that such middleware solutions can be used by higher-level application orchestrators to extend their capabilities to the edge. However, these solutions cannot be directly considered as cloud-to-edge orchestrators, as they lack some core essential features such as a high-level abstraction layer and dynamic deployment or reconfiguration based on user-provided criteria. Therefore, in this section, we only review these solutions and will not present their results in accordance with the taxonomy. The rest of this section discusses these solutions.

Project EVE [55], a Linux Foundation (LF) project, provides a flexible foundation for IoT edge deployments with a choice of any hardware, application, and cloud. EVE enables centralized scalable management of large volumes of edge compute nodes, where the orchestration of the underlying hardware and installed software is achieved through the open EVE API, which ensures consistency across diverse platforms. EVE is complementary to other LF Edge application frameworks including Open Horizon [56], EdgeX Foundry [57], and Fledge [58]. Open Horizon facilitates the management and deployment of workloads on edge devices from a management hub cluster. EdgeX Foundry provides an Edge IoT plug-and-play ecosystem with an aim to simplify and standardize edge computing architectures in the Industrial IoT market. Fledge is a Kubernetes-compatible container orchestrator for edge devices. Lastly, Fledge, in collaboration with the EVE system provides orchestration services and container run-time for Fledge-based applications.

KubeEdge [59] is another open-source initiative with a significant community behind it. KubeEdge extends native containerised application orchestration capabilities to non-cloud nodes at the edge of the network. It facilitates seamless and automatic configuration of edge nodes to make them part of a central Kubernetes cluster. KubeEdge empowers application developers to orchestrate apps, manage devices, and monitor application and system status at edge nodes, just like a normal Kubernetes cluster in the cloud. The components of KubeEdge facilitate the underlying infrastructure support for network, application deployment and synchronisation of metadata between cloud and edge. KubeEdge follows a centralised model. Hence, there is the risk of isolation for edge sites and therefore it is impossible to provision or reconfigure workloads hosted on non-cloud workers if the Kubernetes master node cannot be reached.

In contrast to such a centralised approach, Kubefed [60] and Submariner [61] follow a federated approach, where each edge site can continue to operate in case of network partitions. Such a federated approach offers the advantage of independent control over each edge site, in comparison of a single point of control as in the case of a centralised approach.

Kubefed, despite following a federated approach, still provides a unified way to manage the life cycle of a multi-cluster workload environment. Therefore, Kubefed can be considered as a centralized server that distributes and propagates Kubernetes API objects to multiple clusters. It extends the Kubernetes API by leveraging the use of CustomResourceDefinitions, which is a mechanism to provide user-defined data types in Kubernetes. Overall, although Kubefed is able to provide cluster autonomy to a degree, it also presents several limitations to the edge use cases. For example, it does not implement any sort of knowledge or cooperation between the clusters themselves. Furthermore, the federation control plane, which is designed in a centralized manner, requires the re-creation of a lot of existing features at the federation level.

Submariner, on the other hand, aims to solve the network connectivity between multiple Kubernetes instances. Unlike Kubefed, Submariner can expose Pods and Services from one cluster to another without requiring a new API. Submariner relies on a few internal CustomResourceDefinitions (CRD) to make the intercluster communication possible. All the involved clusters synchronize their state to a shared cluster called Broker, which is responsible for the storage of all the CRD objects. Using this approach, Submariner succeeded in establishing interactions across the Services and Pods of independent clusters. However, the scalability and the robustness of sharing information are limited as most of the locally created objects, such as Deployments and Namespaces, remain local only.

StarlingX [62], similarly to KubeEdge, also extends native containerised application orchestration capabilities to the edges of the network, however, with two key distinctions: (1) StarlingX is specific to the use of OpenStack cloud, and (2) it forms independent edge clouds in contrast to just connecting an edge node with a centralised cluster. The StarlingX solution has a central Kubernetes-based control centre called central cloud, with as many as required sub-clouds deployed on the edge nodes. Using this model, StarlingX forms a federated architecture in a way, similar to that of KubeFed and Submariner. However, it still does not facilitate support for cross-cloud orchestration operations. Hence, all sub-clouds are independently controlled by their own controller. Unique features of StarlingX are its support of cluster management for services running on the HA (High Availability) master/control nodes cluster and recovery of services running on all nodes within the cluster.

OpenIoTFog [63] specifically focuses on the Industrial Internet of Things (IIoT) with two objectives: 1) to extend orchestration functions to the edge devices, and 2) to support the vision of Industry 4.0 by facilitating various related functions, such as real-time data aggregation, asset supervision, predictive maintenance, asset safety and the enablement of Digital Twins. The main aim of the OpenIoTFog is to provide software-based programmable logic controllers that can be dynamically updated and re-configured without production downtime. From a functional viewpoint, OpenIoTFog follows a similar model to that of KubeEdge, i.e., an agent component is required to be installed on an edge device, which makes that device part of a centralised cluster where dynamic policies, concerning the deployment of services on specific edge devices, can be applied. In addition, the agent component can also gather data from various sensors via industrial field bus systems and various (industrial) wireless technologies. It can also standardise, communicate and aggregate them through secured standard-compliant interfaces.

Fornax [64], developed within the scope of an umbrella project called Centaurus [65], is an open-source edge computing framework for managing compute resources on edge environments. The key novel aspect of this project, amongst all the other ones described in this category, is its hierarchical topology that allows edge clusters to be formed and organised in a multi-layer tree-like structure. Hence, the infrastructure can be managed in N layers in comparison with the two-layered approach of KubeEdge and OpenIoTFog, or the federated approach adopted by other works including KubeFed, Submariner, StarlingX.

Higher level solutions

Concept-only solutions

A large number of academic research papers are focused on the Cloud-to-Things orchestration aspects. It is not possible to cover all such papers individually and therefore, we shortlisted 10 papers from this category for review in detail, where other papers are briefly introduced. Amongst the 10 papers, half of them are the most highly cited papers of all time so far and the rest of them are all papers published in 2019 and onwards. It is important to note that these solutions are theoretical with no or just proof-of-concept implementation. The rest of this section discusses these solutions, where Table 1 further presents a complete summary of the reviewed solutions in light of the attributes from the taxonomy.

ENORM [66], a framework for edge node resource management, aimed to address the following three problems: (1) Edge node provisioning, (2) Workload deployment on edge with a focus on how to deploy and what services to deploy, and (3) resource management at the edge. ENORM follows a decentralised architecture, where edge nodes are responsible for their own resource management decisions. However, the overall architecture is static in nature, as all edge nodes are known in advance to the cloud servers' managers running in the cloud. The focus of ENORM is mostly on the operations of edge nodes, where it supports provisioning, monitoring, vertical scaling, and offloading applications. However, the details related to the cloud layer are not known, e.g., how the cloud server managers that are responsible for different applications are provisioned and maintained.

Fernandez et al. [67] introduced slice orchestrator, which facilitates the automated orchestration of IoT services based on certain specific operational (and/or business) requirements over a set of shared infrastructures. Their idea is based on the 5G concept called network slice, which is an end-to-end logical network, capable of providing an agreed quality of service for a defined customer's purpose [68]. Based on this notion, an IoT slice would be a partition of the entire end-to-end IoT solution created to serve a specific (or a group of) customer(s). The job of the slice orchestrator is to establish network slices, set up edge and cloud tenants, and the deployment of IoT functions as per the specific requirements related to resources in terms of computing, storage, network, and target locations (e.g., edge and/or, cloud, and transport network). This solution followed a hybrid architecture, where a centralised slice orchestrator creates and manages slices but also relies on other domain-specific resource orchestrators (e.g., a different cloud orchestrator is responsible for a specific cloud environment) to perform key resource management functions such as resource selection and deployment. However, no details are provided regarding resource provisioning by domain orchestration, run-time reconfiguration aspects and the requirement specification that will be given as input to the system.

Alam et al. [69] introduced a 3-layered reference architecture that makes use of Docker as the underlying orchestration tool for the automated deployment of microservices as containers. Their system follows a centralised model, where key functions like monitoring, adaptation, and orchestration take place at the cloud layer. Their Fog layer is mainly used as a gateway to mediate between the cloud and edge layers for system-specific operations (e.g., to update the status of connected edge devices) or application-specific operations (e.g., data transformation). This system is mostly suitable for publish-subscribe based IoT applications. Similarly to Fernandez et al., [67], no details of various important functions, such as device connectivity at the edge level, resource provisioning at different layers, and run-time reconfiguration, are provided. However, different to others, they include a data mining component, which is responsible for erroneous behaviour detection such as responsiveness of deployed components, and edge devices' statuses. Hence, their system is adaptable in case of any failures.

Table 1 Comparative summary of the concept-only orchestration solutions

Attributes				**ENORM [66]**	**Fernandez et al. [67]**	**Alam et al. [69]**	**Santos et al. [70]**	**Foggy [73]**	**Castellano et al. [74]**	**HYDRA [75]**	**Caravela [76]**	**Mathias et al. [77]**	**Hetero-Edge [80]**
Cloud resource handling		Environment	Single cloud	✓		✓		✓	✓				
			Multi-cloud		✓		✓			✓			
			Cross-cloud										
		Resource types	Compute	✓	✓	✓	✓	✓	✓	✓	✓		
			Storage		✓				✓		✓		
			Network		✓				✓		✓		
		Resource selection	Statically defined	✓	✓	✓		✓					
			Automatic selection					✓	✓	✓	✓		
			Run-time optimised					✓	✓				
Fog/Edge resource handling		Connectivity	Manual registration	✓	✓	✓		✓	✓	✓			✓
			Automatic registration				✓				✓	✓	
		Others	Heterogeneity	✓	✓	✓	✓	✓		✓	✓	✓	✓
			Auto reconnectivity								✓		
			Resource discovery				✓	✓		✓	✓	✓	
Orchestration functionalities	Service/Job Handling	Virt support	VM	✓	✓		✓	✓	✓				
			Containerisation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Mapping	Static	✓	✓	✓	✓	✓			✓	✓	
			Context aware					✓	✓	✓			✓
	Run-time reconfiguration	Definition type	Statically pre-defined	✓								✓	
			User-defined dynamic						✓				
		Operating type	Reactive	✓				✓	✓	✓			
			Proactive				✓						
			Hybrid										
		Scaling	Horizontal							✓			
			Vertical	✓									
			Hybrid						✓			✓	

Table 1 (continued)

Attributes				ENORM [66]	Fernandez et al. [67]	Alam et al. [69]	Santos et al. [70]	Foggy [73]	Castellano et al. [74]	HYDRA [75]	Caravela [76]	Mathias et al. [77]	Hetero-Edge [80]
		Offloading	Cloud-to-Edge	✓		✓	✓						
			Edge-to-Cloud	✓		✓							
			Edge-to-Edge				✓						
	Monitoring	Support level	Cloud			✓							
			Edge	✓							✓	✓	✓
			Cloud-to-Edge			✓		✓	✓	✓			
		Metrics support	System				✓	✓	✓	✓	✓	✓	✓
			Application	✓		✓		✓	✓				✓
			Custom										
	Security handling		Configurable app level	✓				✓					
			Sys wide inter-comp				✓						
			Edge authentication									✓	
			Access control							✓			
	Others		Fault diagnosis			✓	✓						
			SLA Handling	~					~				
Design		Architecture	Centralised			✓		✓				✓	✓
			Decentralised	✓					✓	✓	✓		
			Hybrid		✓		✓						
		App description	Solution independent									✓	✓
			Solution specific					✓	✓	✓	✓		
		Extensibility	Resources										
			Functionalities					✓		✓	✓		
		User interface	GUI (Web/Desktop)				✓						
			CLI	✓	✓			✓	✓		✓	✓	✓
			API (Service/Library)				✓				✓		
Supported App types				G	G	PS	SC	G	DS	G	G	G	CV

[Supported = ✓, partially supported = ~] Supported App types ⇒ Publish Subscriber **(PS)**, Smart City **(SC)**, Data Streaming **(DS)**, Computer Vision **CV**, Generic **(G)**

Santos et al. [70] focused on the optimal application placement problem in smart city applications while considering the reduction in network bandwidth usage and improved latency. Their proposal extends the ETSI NFV MANO architecture [71] with additional functions of monitoring and data analysis. Their system follows a hybrid approach where management and decision-making related to the various functions happen at the cloud layer by cloud node (CN) and at the local layer by fog nodes (FNs). CN is responsible for the global view of the system including operations like coordination and control of FNs, global level data analysis and monitoring of the overall SLA. Each FN on the other hand has its own orchestrator and is responsible for autonomously managing its own local infrastructure, associated devices, and the life-cycle of microservices, as well as interfacing with the modules for resource discovery, system monitoring, data analysis, security, machine to machine communication, and decision making related to application life cycle and related policies. However, no details are provided in relation to these policies, their structure, or how they will be passed on to the system. This solution provides both GUI and API access to facilitate application owners managing and controlling FNs (and CN) independently and to perform manual updates if required. Lastly, a fog protocol based on the existing Open Shortest Path First (OSPF) routing protocol [72] has been proposed to enable and exchange communication between fog and edge layers. Details on edge device management, application description and run-time reconfiguration are missing.

Foggy [73] framework, similarly to Santos et al. [70], aimed to minimise latency and perform optimal application placement. Foggy follows a centralised model. It consists of an orchestration server (OS)—a central entity responsible for deployment and resource management decisions—and an orchestration client (OC)—running on each computational resource and is responsible for enforcing deployment decisions. Overall, Foggy offers the following unique characteristics in contrast to other solutions discussed in this category: 1) To facilitate an automated build, a direct integration of a version control system (such as Github) and continuous integration process as part of their system architecture; 2) A pluggable policy-driven deployment planner that dynamically identifies suitable resources based on user provided requirements; 3) A JSON based container specification to facilitate application owners to provide service requirements using qualitative constructs such as Low, Medium, and High. However, it is not clear how these qualitative specifications for different aspects, such as computation and latency, are mapped within the system. Similarly to others, Foggy also does not cover details related to edge device registration, standardised application description and run-time reconfiguration.

Castellano et al. [74] solution follows a distributed approach where a dedicated instance of a service-defined orchestrator (SDO) is initiated every time a new application is deployed. The input to the system is an application deployment request that mainly consists of a list of components, their topology and a set of declarative statements to form the Orchestration Behaviour Model (OBM) that drives the orchestration functions. The OBM features aspects, such as infrastructure and/or application state, required objectives to be optimised, the events and the corresponding actions to be performed. Using the OBM, every SDO instance aims to make optimal decisions with respect to the managed application. However, this also raises the resource allocation issue for different instances of SDOs at the shared infrastructure level when resources are limited. To cope with this, Castellano et al. [74] introduced Dragon—an additional component responsible for the optimal partitioning of the underlying shared resources across different SDOs. Using Dragon, the SDO can decide to terminate an application component if it cannot allocate the required resources to that particular component. Their proposed declarative statementsbased application description approach, however, is specific to this solution only and does not follow a standardised approach.

HYDRA [75], similar to Castellano et al. [74] also follows a decentralised architecture, where a set of distributed nodes without the presence of a centralised entity are responsible for performing the orchestration functions of one application. HYDRA actually builds a peer-to-peer (P2P) overlay network of computational nodes, where every node serves both as an orchestrator as well as a computational resource—responsible for running the application micro-services. HYDRA supports both location-agnostic as well as location-aware application deployment with a primary focus on the overall scalability and resilience aspects of the underlying resource infrastructure through its decentralised architecture. This has been achieved through the adoption of a dynamic partitioning scheme, where orchestrator nodes operate independently to control the needs on a per-application basis.

Caravela [76] follows a similar decentralised model, where all key aspects such as the overall architecture, resource discovery and scheduling are also based on the concept of a P2P overlay network. However, different for HYDRA, it follows a market-oriented approach, where volunteer resources can join the ecosystem and get rewarded for their services. Caravela dynamically builds edge cloud from the volunteer resources that are further used to deploy applications using Docker containers.

Caravela's scope, however, is only limited to non-cloud layers and does not include resource provisioning from cloud.

Mathias et al. [77] solution consists of a Fog Orchestrator (FO)—a central entity responsible for maintaining a resource catalogue of fog nodes, overall service management, global level monitoring, and orchestration—and an agent component called Fog Orchestrator Agent (FOA) that runs on every fog node and is responsible for activities such as management of connected edge devices, security and monitoring. The working mechanism of this solution suggests that FO composes a TOSCA-based orchestration template using information obtained from a resource catalogue and monitoring components. This template is further used for deployment and run-time management. Such usage of TOSCA for expressing orchestration strategies is common and has been used by many solutions such as [11, 22, 78, 79]. However, in this case, the TOSCA template is dynamically generated by the system and, therefore, it is not clear what the initial input to the system is. A unique prospect of this solution, in contrast to others discussed in this category, is that the FOA can also act as FO if the connection is lost between them. However, this behaviour is static and the specific fog node has to specify this at the time of joining. Furthermore, the scope of the overall solution only includes the non-cloud layers.

Hetero-Edge [80] follows a similar concept to that of Mathias et al. [77], where a central entity has been used to handle the orchestration functions at the non-cloud (edge) layer only. The solution, however, is specific to computer vision applications and relies on the use of Apache Storm (or something similar, such as Apache Flink). Hetero-Edge breaks down an application into smaller Apache Storm tasks and then efficiently maps them onto the connected edge nodes with the objective of minimising the overall end-to-end latency. The specification of tasks is provided through a directed acyclic graph, where the mapping is performed using their custom-developed task scheduler that takes into account the estimated performance and resource demands of tasks. The solutions proposed by Donassolo et al. [81] also follow a similar model, which supports orchestration at the non-cloud layers, however, with a particular focus on optimising the provisioning cost of IoT applications.

Some other more recent notable contributions include GeneSIS [82], which proposed a model-driven approach to automate the deployment of different kinds of deployable artefacts including binary, ThingML-based [83], and container; ECCO [84], which proposed an orchestration framework for enabling the collective use of edge-cloud resources for road context assessment; KubeHICE [85], which took on the challenge of addressing hardware heterogeneity by automatically matching the right computational device that is compatible with the instruction set architecture (ISA) supported by the containerized application; and Gand et al. [86] and Sonmez et al. [87], which focused on the presence and importance of uncertainty in the cloud-to edge environment and therefore adopted a fuzzy logic-based approach for workload deployment.

In addition to the above-mentioned solutions, there are also some research works that did not directly cover the core orchestration functions, however, they emphasised the importance of other related aspects. For example, the authors in [88, 89] introduced the notion of trusted orchestration, where the proposed approach aimed at identifying and tracking orchestration activities to improve trust across the involved actors of the system. Similarly, Kochovski et al. [90] proposed a smart contract (SC) based architecture for SLA management and verification amongst relevant entities and actors of a decentralised environment. More recent works on the DRL-based advanced techniques for dynamic load balancing [91] and network dynamic clustering [92] in edge computing focused on the overall optimisation of cloud-to-edge system. Such solutions can be integrated into distributed orchestration solutions to support self-organisation and optimisation behaviours. Lastly, with the growing popularity of Deep Learning (DL) applications, there is also an increasing interest in proposing resource management solutions that are specifically tailored to DL applications. For example, FlowCon [93] monitors the execution of DL jobs at run-time to make informed resource allocation and placement decisions. Similarly, SpeCon [94] is a container scheduler that aims to optimise resource usage and improves the performance of DL training jobs, whereas DQoES [95] aims at dynamically adjusting cloud resources to meet the target Quality of Experience (QoE) specified by the clients. The scope of all the aforementioned DL-tailored solutions, however, only includes the cloud layer. The details of these papers do not directly fall within the scope of the proposed taxonomy and therefore have not been included here in larger details.

Production ready solutions

Research initiatives In the last few years, a number of EU-funded research projects focused on developing cloud-to-edge solutions. The selection of this subset was made considering three key aspects: 1) whether their core functionality was related to the cloud-to-edge orchestration, 2) whether their implementation was available, and 3) whether there were any publications associated with the solution. The rest of this section discusses these solutions, and Table 2 further presents a complete summary of the reviewed solutions in light of the attributes from the taxonomy.

SODALITE@RT [79] supports the deployment and management of applications across a cloud-to-edge infrastructure in a portable manner. The term “portable” is based on their use of TOSCA as the deployment model to represent application components and resources; and the use of the Infrastructure as Code (IaC) concept [96] to implement the life-cycle operations of components, for which they utilised Ansible [97]. SODALITE@RT follows a centralised model, where a central component called a meta-orchestrator receives TOSCA-based deployment models and Ansible implementation scripts to set up the resources and to perform the deployment. The Ansible scripts are cloud provider specific that the orchestrator pulls from an IaC repository. Such an approach enables custom implementation, however, also burdens application developers with the production of Ansible scripts in comparison with other TOSCA-based solutions such as [22, 78], where the TOSCA model is the only input. The Ansible scripts take care of the cloud resources handling, where the edge resources are handled as part of a Kubernetes cluster. However, details on edge cluster formation are not provided and therefore it is not clear whether a meta-orchestrator creates the edge cluster or it must exist prior to the deployment process. SODALITE@RT also provides an event-condition-action-based policy language to support custom redeployment policies. Furthermore, it also supports access control and mechanisms for secure storage of application secrets. However, no mechanism for application-level security configurations is provided.

Capillary [98] focused on the use of a custom-built monitoring system to measure QoS parameters and Offloading across different resource layers based on various user-defined characteristics, including geographic positioning. The offloading decisions follow an “offload to next immediate layer” model (e.g., edge to fog or fog to cloud) that resembles the capillary fluid movement, hence the name Capillary. It follows a centralised approach, where a central entity called Capillary container orchestrator performs the deployment and offloading operations. The input to the system is a TOSCA deployment model that includes various details, such as resource capacity requirements for services, zone details, and constraints on QoS thresholds that are used for reconfiguration purposes. At run-time, the monitoring system raises alarms based on the developer-provided thresholds. As a result, a sub-component of the orchestrator, similar to SODALITE@RT [79], takes an offloading decision, changes the TOSCA model and triggers re-deployment. For resource handling, the cloud resources are dynamically provisioned by the orchestrator based on the user-provided minimum requirements for the service. However, no details on the provisioning of the fog and edge infrastructure are provided.

MiCADO-Edge [22] is also a centralised solution, where a central entity called MiCADO-Master is responsible for the automated deployment and management of a microservices-based application across the cloud-to-edge continuum using a single TOSCA based deployment model. This model consists of details related to computational resources, component specification, application topology, service placement mapping, user-defined scaling policies and any application-specific security settings. The key focus in MiCADO-Edge is on generalising the resources across the different layers of the continuum by facilitating a mechanism to allow the resources from fog and edge layers (referred to as non-cloud resources) to join a centralised cluster prior to the application deployment process. Once they become part of the MiCADO cluster, developers can reference them in the TOSCA-based deployment model to define placement and reconfiguration policies. Furthermore, MiCADO-Edge empowers application developers to write custom dynamic scaling policies based on a wide range of application and system metrics. MiCADO-Edge, however, currently lacks support for context-based placement of application services and developers are required to provide static mapping between services and resources.

PrEstoCloud [99, 100] follows a similar model of a TOSCA-based orchestration solution. However, it provides an optimisation step before deployment. This step consists of receiving TOSCA in a high-level form (type level TOSCA model as they referred to it), which also contains optimisation criteria independent from the underlying infrastructure resources. Based on the provided criteria, the solution automatically produces a more specific instance-level TOSCA deployment model containing the specific resources across the infrastructure that are to be used for application deployment. Hence, it provides an optimised placement mechanism. Furthermore, PrEstoCloud also focused on facilitating predictive reconfiguration based on the changing data stream conditions considering data-intensive applications.

mF2C [101] adopted an N-layered approach to utilise the available resources in the continuum from edge (Layer-N) to the cloud (Layer-0), in contrast to the two-layered (i.e., cloud and non-cloud) approach followed by MiCADO-Edge [22] and the typical three-layered approach as followed in [98]. Their proposed solution is decentralised, where deployed mF2C agents, coordinate with each other to find suitable resources, closer to the

Table 2 Comparative summary of research projects based orchestration solutions

Attributes				**SODALITE@RE [79]**	**Capillary [98]**	**mF2C [101]**	**MiCADO-Edge [22]**	**PrEstoCloud [99, 100]**	**DECENTER [102]**	**Rainbow [104]**	**Pledger [103]**	**Slack4things [105, 106]**
Cloud resource handling		Environment	Single cloud		✓							✓
			Multi-cloud			✓	✓				✓	
			Cross-cloud	✓				✓	✓			
		Resource types	Compute	✓	✓	✓	✓	✓	✓	✓	✓	✓
			Storage									✓
			Network									
		Resource selection	Statically defined	✓			✓				✓	✓
			Automatic selection		✓	✓		✓	✓	✓		
			Run-time optimised					✓			✓	
Fog/Edge resource handling		Connectivity	Manual registration	✓	✓	✓	✓				✓	
			Automatic registration					✓	✓	✓	✓	✓
		Others	Heterogeneity	✓	✓	✓	✓	✓	✓	✓	✓	✓
			Auto reconnectivity	✓		✓	✓	✓		✓		✓
			Resource discovery			✓		✓	✓	✓		✓
Orchestration functionalities	Service/Job Handling	Virt support	VM				✓					
			Containerisation	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Mapping	Static	✓	✓		✓				✓	
			Context aware	~	✓			✓	✓	✓		✓
	Run-time reconfiguration	Definition type	Statically predefined		✓	✓	✓		✓	✓		
			User-defined dynamic	✓			✓	✓			✓	✓
		Operating type	Reactive	✓	✓	✓	✓		✓	✓	✓	✓
			Proactive					✓				
			Hybrid									
		Scaling	Horizontal				✓	✓				
			Vertical					✓				
			Hybrid							✓	✓	

Table 2 (continued)

Attributes			SODALITE@RE [79]	Capillary [98]	mF2C [101]	MiCADO-Edge [22]	PrEstoCloud [99, 100]	DECENTER [102]	Rainbow [104]	Pledger [103]	Slack4things [105, 106]
	Offloading	Cloud-to-Edge		✓						✓	✓
		Edge-to-Cloud		✓						✓	✓
		Edge-to-Edge	✓				✓		✓		✓
Monitoring	Support level	Cloud							✓		
		Edge							✓		
		Cloud-to-Edge	✓	✓	✓	✓	✓	✓		✓	✓
	Metrics support	System	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Application	✓	✓	✓	✓	✓		✓	✓	✓
		Custom	✓	✓		✓	✓			✓	
Security handling		Configurable app level				✓					
		Sys wide inter-comp			✓	✓	✓			✓	
		Edge authentication						✓	✓	✓	
		Access control	✓				✓		✓	✓	
Others		Fault diagnosis				✓	✓				
		SLA Handling			∼		∼	∼	∼	∼	

Table 2 (continued)

Attributes			**SODALITE@RE** [79]	**Capillary** [98]	**mF2C** [101]	**MiCADO-Edge** [22]	**PrEstoCloud** [99, 100]	**DECENTER** [102]	**Rainbow** [104]	**Pledger** [103]	**Slack4things** [105, 106]
Design	Architecture	Centralised	✓	✓		✓	✓	✓		✓	
		Decentralised			✓				✓		✓
		Hybrid									
	App description	Solution independent	✓	✓		✓	✓	✓		✓	
		Solution specific			✓				✓		
	Extensibility	Resources	✓			✓	✓			✓	
		Functionalities				✓	✓				
	User interface	GUI (Web/Desktop)	✓				✓	✓	✓	✓	
		CLI							✓		
		API (Service/Library)		✓	✓	✓			✓	✓	✓
Supported App types			G	DI	G	G	DI	SC	G	G	G

[Supported = ✓, partially supported = ~] Supported App types ⇒ Generic **(G)**, Data Intensive **(DI)**, Smart construction **(SC)** edge, for the execution of application services. The input to the system, e.g., a service execution request is received by the mF2C agent at the lower layer. The receiving agent, in coordination with other agents at the same layer, aims to execute the service request if the required resource specification can be fulfilled. Otherwise, the request is further passed on to the mF2C agents at the upper layer. The service execution request is in JSON format that includes the required resource specification used by the mF2C agents to make deployment decisions. In terms of resource handling, the mF2C architecture supports the automatic discovery of other mF2C agents, the dynamic formation of clusters, and also reconfiguration in case of device mobility prospects. However, it does not address aspects like scaling, offloading, dynamic provisioning of resources, and configurable policies.

DECENTER [102] is specifically developed for transforming construction sites into smart and safe environments. Hence, this solution facilitates methods that are specifically tailored to the problems related to construction processes. The unique feature of DECENTER, amongst other solutions in this category, is the Blockchain-based resource brokerage mechanism, which facilitates the trusted brokerage and negotiation of computational resources that can be used for deployment. Furthermore, all transactions of the system are traceable and can be formally verified. Hence, improving the trust and transparency of the overall system. DECENTER follows a centralised architecture, where four key components of the system including Application composer, QoS-aware decision maker, Monitoring system, and Orchestrator are responsible for performing the key orchestration functions. It also facilitates users with a GUI interface to select the services they want to use and define their QoS objectives. These details, along with the monitoring data, are used by the QoS-aware decision maker to perform deployment decisions that are forwarded to the orchestrator. DECENTER also supports automatic redeployment, when the system encounters violation of QoS specifications. However, it lacks functions like dynamic auto-scaling and offloading.

Pledger [103] also makes use of Blockchain to improve trust, secure communication and enable ad-hoc networks between the resources of non-cloud layers to collaborate with each other for the execution of a specific application. Although Pledger's overall architecture follows a centralised model, its implementation does not comply with a traditional adapter-based interaction model between different parts of the system. Pledger provides different toolkits for resource providers to integrate their resources into the Pledger ecosystem and for the application owners to perform mapping of their applications on specific resources, which is further assessed and reconfigured by the core Pledger service to ensure optimised use.

Rainbow [104] particularly focused on the issue of lack of handling concerning the fog-specific constraints related to the deployed services. For this purposes, their proposed solution consists of a high-level abstraction mechanism, where application topology and the related constraints on services are described through a graph. The Rainbow orchestration system accepts the graph as input and deals with the optimised placement of the services and the execution thereafter. The orchestration system follows a decentralised model, where different components of the system may run on the different computational nodes that are part of the Rainbow ecosystem. To address the various challenges of the fog environment (such as low-powered devices, intermittent connectivity, and the interactions of sub-components), the system follows a publish-subscribe mechanism where a component called Orchestrator Repository maintains the states of the system and its sub-components. The Rainbow platform facilitates the dynamic registration of edge devices and their reconfiguration as per defined service level objectives (SLO) violations. However, its scope is only limited to fog/edge resources and lacks the dynamic provisioning of cloud resources.

Slack4things [105] is an open-source initiative developed by the Mobile and Distributed Systems Lab (MDSLab) at the University of Messina, Italy. This project aimed to provide an OpenStack-based IoT framework for managing IoT devices seamlessly, i.e., without considering their physical location, network configuration, and underlying technology. The tools from this project are further extended by Merlino et al. [106] to build a distributed orchestration solution based on a three-layer architecture that covers cloud-fog-edge, and supporting both horizontal and vertical task offloading. With the former, tasks can be migrated within the same layer, e.g., from one edge device to another; with vertical offloading, tasks can move across different layers, e.g., from edge to fog, or from fog to cloud. Unlike other systems, this solution is based on independent managers deployed at each layer of the architecture; hence, applications can be directly deployed, partly or as a whole, to any layer through the provided managers.

Beyond the projects introduced above, there are some relevant EU research initiatives that have just recently started; however, at the time of review, we were not able to find any reported results from these initiatives therefore, we only briefly review them below for the purpose of completeness. The **European Cloud, Edge and IoT (CEI) Continuum** [107] is an umbrella initiative that provides the strategic guidance and next stage of tech development to achieve the goals of an active and dynamic European CEI ecosystem, with an emphasis on promoting the establishment of a global and open ecosystem for the Cloud-Edge-IoT technologies. The initiative coordinates across clusters of Research and Innovation Actions to support industries and researchers in creating impact, promoting the link between open source and open standards, and engaging relevant industrial alliances in actions directed toward open approaches. Among such clusters, the Meta-Operating Systems for the Next Generation IoT and Edge Computing (MetaOS) [108] are relevant for this review paper, and include projects such as AerOS, FluiDOS, ICOS, NebulOus, NEMO and NEPHELE. Likewise, the cluster AI-enabled computing continuum from Cloud to Edge (CognitiveCloud) [109] is also related to our work, and includes projects such as AC3, ACES, CloudSkin, CODECO, COGNIFOG, DECICE, EDGELESS, MLSysOps and SovereignEdge.Cognit. More details of these projects are available at the CEI website.

Industry initiatives This section presents an overview of some of the existing industry platforms that support the combined orchestration of cloud and edge resources. The key factors that led to the inclusion of these solutions are both the availability of their implementation and the presence of technical documentation and/or associated white papers. It is important to note that, even though such solutions often rely on underlying open source components, such as Docker and Kubernetes, they are in fact vendor specific, with their scope mostly focused on the orchestration of (5G) network services. Moreover, these being industry-oriented solutions, we found that they often lacked documentation providing in-depth descriptions of the related technical details. Instead, the available documentation focused more on the presentation of features for targeting customers. Therefore, the evaluation of the characteristics of these solutions against the taxonomy was not obvious due to the lack of information. Nonetheless, we include these solutions in the paper for the purposes of completeness, even though a detailed comparative summary table will not be presented in this section.

HPE GreenLake [110] cloud-to-edge is an infrastructure-as-a-service solution that brings the public cloud model to multiple IT environments, such as private clouds, multi-cloud and on-premises, in order to deliver an agile cloud everywhere modality to the users. HPE GreenLake allows for the integration, management and monitoring of all the above resources through a centralised interface. Users can access different types of deployable resources and services, e.g., bare-metal, compute, storage, containers and data protection services, as well as HPC, AI/ML and virtual desktop infrastructures.

Intel Smart Edge Open [111] is an edge computing software toolkit for building platforms optimized for the edge. This is done by providing a toolkit of functionality selected from across the cloud native landscape, which has been extended and optimised to be used at the edge. This solution is able to work with heterogeneous hardware resources from the on-premise edge to regional data centres. These are managed by using a set of "experience kits", provided by Intel and built on top of Kubernetes, that combine 5G capabilities and cloud-native components to simplify the deployment of complex network architectures, significantly reducing development time and cost. For instance, the Developer Experience Kit provides the base capabilities to run containerised edge services, including networking, security, and telemetry. An experience kit consists of building blocks that can be chosen according to the customer's needs. Specifically, Resource management provides identification, configuration, allocation, and continuous monitoring of the hardware and software resources on the edge cluster; the Telemetry and Monitoring combine application telemetry, hardware telemetry, and events to create a heat-map across the edge cluster and enable the orchestrator to make scheduling decisions.

AMCOP [112]—Aarna Networks Multi Cluster Orchestration Platform—is an open-source platform for orchestration, life-cycle management, and closed-loop automation of cloud-native network services and edge computing applications. AMCOP aims to solve the problem of managing the growing number of edge applications and edge sites by offering intent-based orchestration of network services and composite edge computing applications, which comprise cloud-native network functions and cloud native applications; it also supports service assurance for edge and 5G services through real-time, policy-driven closed-loop automation. AMCOP works by interfacing (northbound) with the collection of systems/applications that a network service provider already uses to operate its business (OSS/BSS), and by orchestrating infrastructure and network services/applications across multiple heterogeneous Kubernetes clusters (southbound).

Ormuco [113] is a solution that aims to lead the deployment and usage of edge computing as an effective approach to deliver data processing. The platform was developed to respond to the needs of modern businesses that require the setup of an infrastructure-as-a-service via a decentralised approach in order to increase their revenue, reduce both the operations and maintenance costs, and automate the deployment of systems and applications on demand. The platform's Cerebro virtual sysadmin is able to collect logs from heterogeneous computing nodes and applications; these are used to learn the expected behaviour of the deployed software and to notify application owners of any detected potential anomalies.

Azion [114] is an end-to-end encrypted edge orchestration service with cloud management and zero-touch provisioning, created for large-scale edge networks. Users can manage and control resources across the edge in real-time and orchestrate services more easily, according to specific service requirements. The orchestration relies on an agent, installed on the edge nodes, that provides encrypted remote node management to the Azion Control panel, within the Real-Time Manager, deployed in the cloud. The Edge Node module enables devices to be created, managed and implements the integration with the orchestrator. The Edge Services module enables the customers to create their own services and allows them to be managed and orchestrated by the Real-Time Manager.

ONAP [115] platform provides a unified operating framework for vendor-agnostic, policy-driven service design and implementation, as well as analytics and lifecycle management for large-scale workloads and services. Network operators can use ONAP to orchestrate both physical and virtual network functions; hence, they can capitalise on their existing network infrastructure while being part of a vibrant VNF ecosystem that includes providers around the globe. The ONAP Operations Manager (OOM) module, based on Kubernetes is responsible for orchestrating the end-to-end lifecycle management and monitoring of ONAP components, as well as enforcing scalability and resiliency mechanisms.

ZEDEDA [116] is a cloud-based orchestration solution for the secure control of distributed edge computing deployments, which provides users with full-stack remote management of edge computing hardware and applications deployed both on clouds or on-premises systems. ZEDEDA leverages EVE-OS [55], a secure, open universal operating system, developed with vendor-neutral and open-source governance as part of the Linux Foundation's LF Edge organization. EVE-OS simplifies the deployment, orchestration and security of cloud-native and legacy applications on distributed edge compute nodes. EVE-OS encrypts data, maintains device and software integrity and supports VMs, containers and clusters (Docker and Kubernetes).

Discussion, issues, and future directions

Section "Review of existing CoTOS" thoroughly reviewed the existing solutions by classifying them into different groups. Table 1 and 2 further summarised the solutions from the concept-only and research initiatives categories by outlining their characteristics based on the proposed taxonomy. This section discusses and reflects on the results from tables with the aim of highlighting directions for future work in relation to the advancement of cloud-to-edge orchestration. More particularly, Section "Open issues" discusses the open issues that require further consideration, whereas Section "A conceptual framework of orchestration in the Cloud-to-Things compute continuum", based on the analysis of the CoTOS landscape and the identification of notable gaps, presents a generic high-level conceptual framework for the development of the next generation CoTOS.

Open issues

Standardised support for application description

A key distinction that we made in this paper is the differentiation between lower-level and higher-level solutions using the presence (or lack) of a high-level abstraction layer. Such an abstraction aims to increase portability and interoperability by empowering users to specify their applications' requirements using a high-level standardised method to avoid any kind of vendor and/or technology lock-in. It is evident from results by focusing on the *"App description"* attribute that a number of solutions such as ENORM [66], Fernandez et al. [67], Alam et al. [69], Santos et al. [70] do not provide such an abstraction layer. A number of other solutions, including Foggy [73], Gabriele et al. [74], mF2C [101], and Rainbow [104] used a YAML or custom DSL based abstraction mechanism. However, all these approaches are specific to respective solutions, hence solution dependent. On the other hand, most of the solutions in the research initiative category including SODALITE@RE [79], Capillary [98], MiCADO-Edge [22], PrEstoCloud [99, 100], DECENTER [102] and Pledger [103] follow a solution independent approach, where they mainly apply the TOSCA standard format for application description. TOSCA is a well-known, popular and standardised cloud orchestration modelling language that has been extensively used by many cloud orchestration tools. However, the latest TOSCA standard (i.e., Version 1.3 [117] at the time of writing) still lacks support for edge-related aspects. Due to the lack of native support, all the aforementioned solutions provide ad-hoc extension and implementation for the edge-related aspects. Such ad-hoc adoption of TOSCA loses the inherent portability aspects of utilising TOSCA as an abstraction layer. Hence, further efforts are required to develop (or extend) existing standardised modelling languages to enable native support for edge-related aspects.

SLA management

The complex nature of the cloud-to-edge continuum, which consists of heterogeneous computational and communication infrastructures belonging to multi-domains, and often relies on ephemeral mobile, low-power computational devices with volatile connectivity, brings in possibly varying run-time conditions that can ultimately influence the delivery of the expected QoS. Therefore, from a system viewpoint, the management of SLAs of IoT applications is a very complex task. It is evident from the results by referring to the *"SLA handling"* attribute that, some of the existing works, e.g., ENORM [66], Fernandez et al. [67] from the concept-only category, and almost all solutions from the research initiative category have considered the SLA aspect. However, the majority of these solutions have focused only on SLA enforcement, where they perform reconfiguration as a result of the violation of certain conditions. We consider that the scope of SLA management is larger than just SLA enforcement, and orchestration systems are also needed to focus on other related aspects, e.g., a standardised way of describing SLA, reporting of SLA violation, formal assurance of SLA, SLA negotiation considering different administrative domains, and SLA monitoring across different administrative domains. However, amongst the reviewed solutions, very limited efforts are provided with regard to these aspects. Some limited notable examples include the reporting of SLA violations by mF2C [101] and the formal assurance of SLA in DECENTER [102]. For the SLA specification, which is an essential task for QoS-aware orchestration, a number of solutions provide their custom method to define a reconfiguration policy, e.g., MiCADO-Edge asks application owners to define their SLA by writing a Python-based scaling policy using a set of exposed variables, while mF2C allows resource based conditions. However, all such methods assume specific knowledge of the underlying systems. On the other hand, commercial solutions, such as AMCOP and ONAP also have their SLA specification mechanisms based on existing ETSI proposals [118]. However, the scope of these is mainly focused on the network domain only and hence may not be applicable to the whole cloud-to-edge continuum. Further efforts are required to provide a standardised format that facilitates application owners to specify the SLA requirements of their applications. Some limited individual works in this direction include [119, 120]. These works advocate the use of model-driven SLA specifications that extend TOSCA with SLA-defining constructs. For other aspects, there are also some initial works, e.g., for QoS negotiation [121, 122], and for formal assurance of SLA [90, 123]. From an overall orchestration viewpoint, these aspects are essential and there should be more focus on incorporating these functions.

Context-aware resource discovery

As already pointed out for SLA Management, the multi-domain, heterogeneous nature of the cloud-to-edge continuum poses considerable challenges to the unified management of the available resources. These should ideally be made available as a pool that can grow and shrink as resource elements are dynamically discovered across the different layers of the involved administrative domains. The results (relevant attributes include: *"Resource discovery"* and *"Mapping"*) inform us that most of the solutions lack a resource discovery mechanism and therefore provide a static mapping, where users manually define the association of components to specific resources, rather than a context-aware one. However, the cloud-to-thing continuum model is highly dynamic, where IoT applications have different requirements relating to resource, e.g., locality, type of resources, operating system, architecture, remaining battery power. Hence, from a uniform resource management viewpoint, an orchestration solution should provide a standardised dynamic way to empower resource owners to securely register their resources using the various associated contextual attributes, and where these resources can be dynamically discoverable across different administrative domains at the time of deployment and reconfiguration decisions. Hence, we foresee the usage of solutions similar to the one described in our previous work [124], where a *Resource Marketplace* was considered as a dynamic approach for sharing resource availability information among various domains. This also allows the execution of mapping algorithms aimed at identifying resources across the cloud-to-edge continuum that can be used to satisfy the performance requirements of the IoT services requested by the users.

Proactive run-time reconfiguration

It is evident from the results by focusing on the *"Operating type"* attribute that almost all of the reviewed solutions are reactive in nature, rather than proactive. Such solutions execute reconfiguration actions (i.e., either scaling and/or offloading) in response to the changes in the behaviour of the system that can be identified through the fulfilment (or violation) of some conditions. The key issue with the reactive approach is the delay elapsed between the time of the reaction to a change and the actual completion time of the reconfiguration process [125]. Reactive orchestration is also more prone to oscillation. You may react too fast and then react again, causing too many reconfigurations. The proactive approach on the other hand anticipates future behaviour of the system and performs the necessary reconfiguration in advance. Within the cloud-only domain, there is already a lot of attention provided on the use of proactive approaches to perform auto-scaling (e.g., please check these review papers for further details [126, 127]). However, as evident from the results, there is very limited attention on the use of proactive approaches by orchestration systems in the cloud-to-edge continuum. Therefore, a new breed of machine learning-based contextual models need to be developed to take reconfiguration decisions by anticipating future system behaviour considering a large range of relevant contextual information, such as: System resources (e.g., utilisation, capabilities, types, availability), Networking aspects (e.g., congestion level, available bandwidth, communication overhead), Energy requirements (e.g., battery utilisation, battery life), Environmental context (e.g., locality, time of the day), Application service context (low latency, QoS specification), and Social behaviour aspects, (e.g., roaming habits of users).

Decentralised architecture

Most of the reviewed solutions, especially the ones with implementations, follow a centralised approach. This approach offers a number of key advantages, such as significantly reducing the complexity of the implementation and offering consistent decision-making, thanks to the information relevant to the decision process being located in a single place. However, the centralised approach also comes with significant drawbacks [37]. For example, it suffers from the lack of scalability, it offers a single point of failure and a centralised target for cyber-attacks. Moreover, a single central component can easily become a bottleneck from an efficiency point of view. Such a model fits more naturally in the single-cloud domain. However, considering the cross-cloud and distributed nature of the cloud-to-edge ecosystem, a centralised approach also raises additional concerns, e.g., the transfer of recurrent monitoring data from distributed resources to a central point, sporadic connectivity, privacy and locality constraints. A decentralised approach can address the aforementioned limitations. In such an approach, multiple orchestrators (decision makers) work independently, and each orchestrator manages its dedicated applications or its own domain of the infrastructure and also collaborates with each other to reach QoS standards or fulfil SLAs or policy requirements [128, 129].

Security handling

The core focus of this paper was not on the security aspects of orchestration systems. However, we still evaluated existing solutions in four basic but essential aspects related to security. It is evident from the results by focusing on the sub-attributes of *"Security handling"* that only a few solutions have very limited and partial attention on some of the four aspects. Hence, more attention is required in this regard. One of the key challenges for orchestration systems related to security is the resource-constrained computational devices at the edge, which in some cases are unable to support the traditional security methods. Therefore, new methods need to be designed considering the distributed, multi-administrative domain and resource-constrained nature of the cloud-to-edge environment. Further specific details in relation to security challenges and some solutions can be found in [35, 130, 131].

A conceptual framework of orchestration in the Cloud-to-Things compute continuum

Based on the analysis of the CoTOS landscape, and the identification of notable gaps described in the previous section, we present a generic high-level conceptual framework for the development of the next generation of CoTOS. This section describes the high-level abstract details of the framework to help researchers and developers in identifying potential components and building blocks, based on the functionalities and notable missing features of the current solutions.

The ideas that underpin our framework proposal are also aligned with the objectives targeted by some of the newly started research initiatives in the context of orchestration in the cloud-to-edge continuum, such as those mentioned earlier in the paper [107]. Similar to [108], we also foresee the development of a unified framework for smart IoT applications, acting as a Meta-Operating System that enables seamless cloud and edge computing orchestration by bringing computation, data and intelligence closer to where the data is produced. As in [109], AI techniques will also be used to build a cognitive framework that will automatically adapt to the growing complexity and data deluge by integrating seamlessly and securely diverse computing and data environments, spanning from core cloud to edge.

The implementation aspects of the suggested framework's components and overall execution model are left out for developers to choose based on their key requirements. Figure 3 illustrates the high-level architecture of our envisioned system, where the overall structure has been categorised into three layers, namely the *Application Description layer*, the *Orchestration layer*, and the *Infrastructure layer*. The following sections explain each of these layers.

Application description layer

We expect enabling application owners to produce vendor-agnostic and interoperable deployment models for their applications that will be deployed, managed and executed within the Cloud-to-Things continuum. We aim to have a single uniform standardised deployment

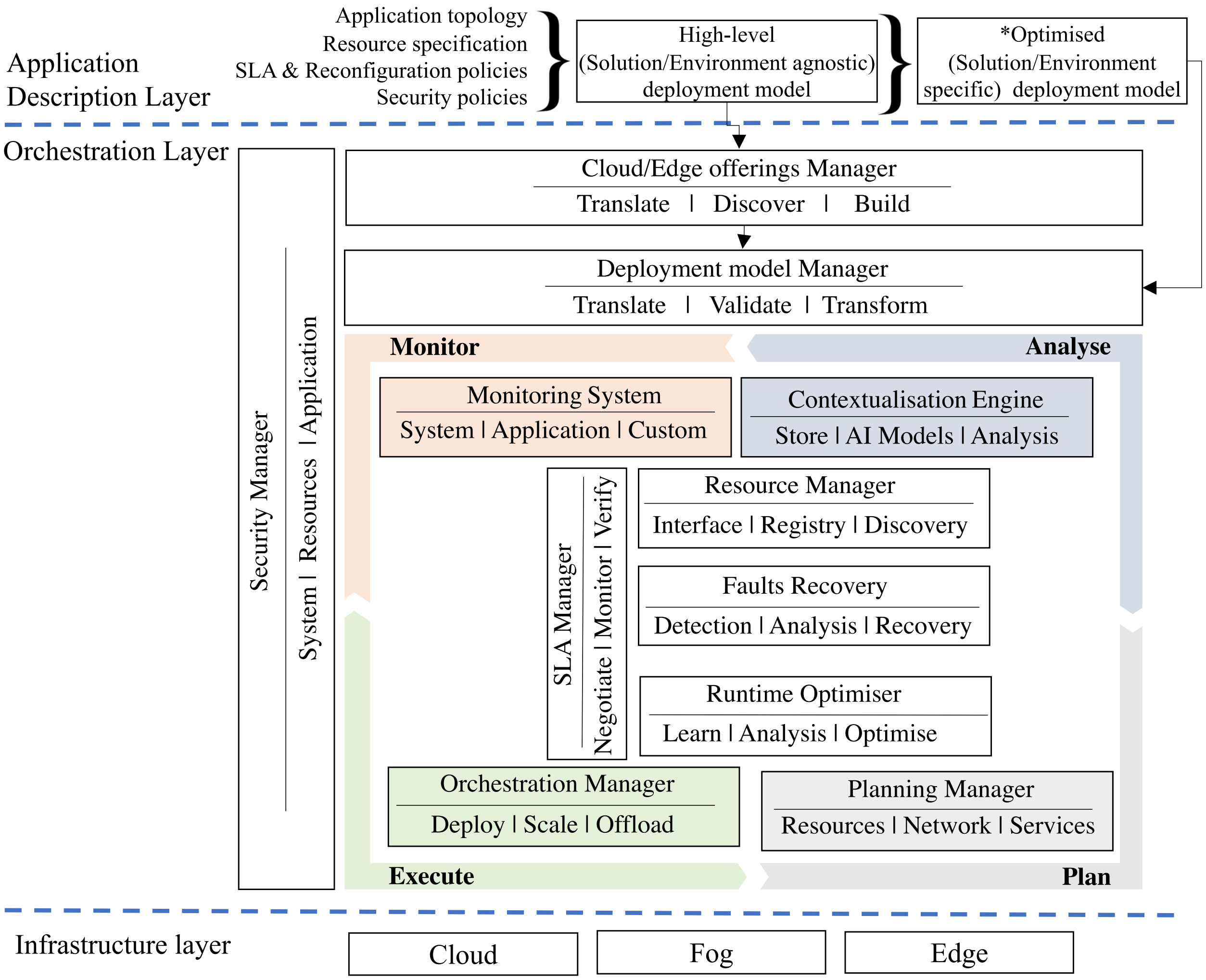
Application Description Layer
Application topology
Resource specification
SLA & Reconfiguration policies
Security policies
High-level (Solution/Environment agnostic) deployment model
*Optimised (Solution/Environment specific) deployment model
Orchestration Layer
Security Manager
System | Resources | Application
Cloud/Edge offerings Manager
Translate | Discover | Build
Deployment model Manager
Translate | Validate | Transform
Monitor
Analyse
Monitoring System
System | Application | Custom
Contextualisation Engine
Store | AI Models | Analysis
SLA Manager
Negotiate | Monitor | Verify
Resource Manager
Interface | Registry | Discovery
Faults Recovery
Detection | Analysis | Recovery
Runtime Optimiser
Learn | Analysis | Optimise
Orchestration Manager
Deploy | Scale | Offload
Planning Manager
Resources | Network | Services
Execute
Plan
Infrastructure layer
Cloud
Fog
Edge

Fig. 3 Conceptual framework of orchestration in the Cloud-to-Things computing continuum

descriptor (a deployment model for the application) that incorporates all the relevant details of the target application, including the topological structure of the involved microservices, the specification of the cloud-to-edge resource requirements and SLA specification, reconfiguration, and application-level security policies. Inspired from [78, 124], we consider a multi-level deployment model. A *high-level*, where an application owner provides an abstract description of the required resources and some optimisation criteria, rather than specific details in relation to resource providers, and/or instances. Based on user-provided optimisation criteria, the *Cloud/Edge Offerings Manager* component is responsible to produce an optimised deployment model consisting of specific details of the resources that will be used. The optimised deployment model is then passed on to the *Deployment Model Manager* component. Alternatively, application owners could also follow a static approach, whereby they can directly produce a deployment model, containing all the required resource-specific details, which can be directly passed on to the *Deployment Model Manager* component.

Orchestration layer

At this layer, the *Deployment Model Manager* receives either the *high-level* or *intermediate-level* deployment model mentioned above, and it is responsible for the translation, validation and transformation of the model into the corresponding low-level details related to the platform (and resources), such as specific orchestration manifests, policies enforcement. For the key operations, i.e., the deployment, reconfiguration and run-time operational management of the target application, we envision the MAPE-K [132] loop architectural concept of self-adaptive systems. As Figure 3 depicts, there are specific components that are responsible for each stage of the

MAPE-K loop. More particularly, the role of the *Monitoring System* is to collect the system and application-related metrics, as well as the related contextual information, across the entire Cloud-to-Things ecosystem. The collected information will be received by the *Contextualisation Engine* and utilised at different points in time; It will be fed into a wide range of relevant Artificial Intelligence models to make system-level proactive decisions related to the applications being executed. These decisions will be further planned and scheduled in terms of specific actions, considering the available resources by the *Planning Manager*. Lastly, the *Orchestration Manager* will execute the planned actions.

The key MAPE-K components of the envisioned system are supported by the *Resource Manager*—responsible for the overall management of both cloud and non-cloud resources. For the non-cloud resources, it will also manage resource registration and dynamic discovery at run-time; *Run-time Optimiser*—responsible to optimise the overall deployment setup in terms of resource usage and application performance; *SLA Manager*—responsible to manage the SLA related aspects, including providing an interface for negotiation, monitoring and verification at run-time; and the *Faults Recovery*—responsible for detecting run-time errors in relation to application services and resources, and taking steps for automatic recovery. Lastly, the *Security Manager* consists of various enablers that are responsible for dealing with the overall security of the system across the multi-level cloud-to-edge layers, such that the entire system can securely operate on heterogeneous resources geographically dispersed across multiple domains. It will also be responsible for enforcing the application-level security policies defined by the application owners at the *Deployment layer*. Moreover, the *Security Manager* should also provide methods for authentication of resources (and IoT devices), such that unauthorised use of resources can be eliminated at the resource level and not only at the system component level.

Infrastructure layer

Various types of geographically dispersed resources belonging to the Cloud-to-Things continuum, and potentially spanning multiple administrative domains, are the constituting elements of this layer. These resources will be used by the above-mentioned components of the *Orchestration layer* in order to simultaneously deploy, reconfigure and manage IoT applications.

Conclusion

The increasing adoption of the Cloud-to-Things computing model emphasises the importance of intelligent and robust orchestration solutions to address the quintessential needs of modern IoT applications, which require simultaneous access and management of geographically distributed arrays of sensors, heterogeneous remote, local and multi-cloud computational resources, as well as dynamic handling of the application execution. In this paper, we thoroughly reviewed a diverse range of existing orchestration solutions; we then proposed a novel taxonomy that consists of a wide set of characteristics that we deemed essential for the automated deployment and run-time management of IoT applications within the Cloud-to-Things continuum.

Based on the obtained results from this review, we identified six key areas, where current solutions are lacking focus. These areas include standardisation support for application description, SLA management, context-aware resource discovery, proactive run-time reconfiguration, decentralised architectures, and security management. These areas highlight directions for future work. Moreover, based on these identified areas, we also presented a proposal for a conceptual framework that can provide a foundation for the implementation of future orchestration solutions.

Authors' contributions

All authors equally contributed to the design of taxonomy and classification and the general conceptualisation and concept of the paper. **Amjad:** Initial conceptualisation, Design of the conceptual framework, Contributed to the write-up of Sections including Introduction, Related work, Taxonomy, Concept-only, Research initiatives, "Discussion, issues, and future directions" and "A proposal of conceptual framework". **Tamas:** Design of the conceptual framework, and review of the entire manuscript. **J ozsef:** Contributed to the design of taxonomy, the write-up of Section "Industry initiatives", and review of the entire manuscript. **Francesco:** Contributed to the write-up of sections including "Industry initiatives", "Discussion, issues, and future directions", "A proposal for a conceptual framework", and review of the entire manuscript. **James:** Contributed to the write-up of Section "Lower-level solutions". **Huseyin:** Contributed to the write-up of Section "Lower-level solutions", and review of the entire manuscript. **Resmi:** Contributed to the write-up of Section "Lower-level solutions". **Hamed:** Contributed to the write-up of Section "Research initiatives".

Funding

This work was supported by the following projects funded by the European Commission's H2020 Programme: DIGITbrain (project number 952071), PITHIA-NRF (project number 101007599) and Harpocrates (project number 101069535).

Availability of data and materials

Not applicable.

Declarations

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 13 December 2022 Accepted: 4 September 2023
Published online: 27 September 2023

References

1. Gartner (2019) Gartner forecasts worldwide public cloud revenue to grow 17 in 2020. https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020. Accessed 5 Oct 2020
2. Marston S, Li Z, Bandyopadhyay S, Ghalsasi A, Zhang J, Ghalsasi A (2011) Cloud computing - the business perspective. Decis Support Syst 52(1)176–189
3. Tomarchio O, Calcaterra D, Modica GD (2020) Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. J Cloud Comput. https://doi.org/10.1186/s13677-020-00194-7
4. Amazon (2020) Aws cloudformation: Speed up cloud provisioning with infrastructure as code. https://aws.amazon.com/cloudformation/. Accessed 18 Oct 2020
5. OpenStack (2020) Openstack orchestration. https://wiki.openstack.org/wiki/Heat. Accessed 18 Oct 2020
6. Azure (2020) Azure resource manager (arm) templates. https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview. Accessed 19 Oct 2020
7. Google (2020) Google cloud depyment manager. https://cloud.google.com/deployment-manager. Accessed 19 Oct 2020
8. Kubernetes (2020) Kubernetes : Production-grade container orchestration. https://kubernetes.io/. Accessed 4 Oct 2020
9. Docker (2020) Docker swarm. https://docs.docker.com/engine/swarm/. Accessed 4 Oct 2020
10. Apache Brooklyn (2020) Apache brooklyn: software for managing cloud applications. http://brooklyn.apache.org/. Accessed 4 Oct 2020
11. Cloudify (2020) Cloudify orchestration platform - multi cloud, cloud native & edge. https://cloudify.co/. Accessed 6 Sep 2022
12. Cloudiator (2020) Cloudiator: A multi-tenant, cross-cloud orchestration framework. https://github.com/cloudiator. Accessed 4 Oct 2020
13. Alien4Cloud (2020) Alien 4 cloud. https://alien4cloud.github.io/. Accessed 4 Oct 2020
14. MODAClouds (2020) Modaclouds multi-cloud devops alliance: Modaclouds releases multi-cloud devops toolbox. http://multiclouddevops.com/. Accessed 4 Oct 2020
15. Kiss T, Kacsuk P, Kovács J, Rakoczi B, Hajnal Á, Farkas A, Gesmier G, Terstyanszky G (2019) Micado–microservice-based cloud application-level dynamic orchestrator. Futur Gener Comput Syst 94:937–946
16. (1934) IEEE standard for adoption of openfog reference architecture for fog computing. IEEE Std 2018:1–176
17. Kimovski D, Mathá R, Hammer J, Mehran N, Hellwagner H, Prodan R (2021) Cloud, fog, or edge: Where to compute? IEEE Internet Comput 25(4):30–36
18. Moreschini S, Pecorelli F, Li X, Naz S, Hästbacka D, Taibi D (2022) Cloud continuum: the definition. IEEE. Access 10:131876–131886
19. Svorobej S, Bendechache M, Griesinger F, Domaschka J (2020) Orchestration from the Cloud to the Edge. In: Lynn T, Mooney JG, Lee B, Endo PT (eds) The Cloud-to-Thing Continuum: Opportunities and Challenges in Cloud, Fog and Edge Computing. Springer International Publishing, Cham, p 61–77. https://doi.org/10.1007/978-3-030-41110-7_4
20. Bittencourt L, Immich R, Sakellariou R, Fonseca N, Madeira E, Curado M, Villas L, DaSilva L, Lee C, Rana O (2018) The internet of things, fog and cloud continuum: Integration and challenges. Internet Things 3:134–155
21. DesLauriers J, Kiss T, Ariyattu RC, Dang H-V, Ullah A, Bowden J, Krefting D, Pierantoni G, Terstyanszky G (2021) Cloud apps to-go: Cloud portability with TOSCA and MiCADO. Concurr Comput: Practice and Experience 33(19):e6093
22. Ullah A, Dagdeviren H, Ariyattu RC, DesLauriers J, Kiss T, Bowden J (2021) MiCADO-Edge: Towards an Application-level Orchestrator for the Cloud-to-Edge Computing Continuum. J Grid Comput 19(4):1–28
23. Velasquez K, Abreu DP, Assis MR, Senna C, Aranha DF, Bittencourt LF, Laranjeiro N, Curado M, Vieira M, Monteiro E, Madeira E (2018) Fog orchestration for the Internet of Everything: state-of-the-art and research challenges. J Internet Serv Appl. https://doi.org/10.1186/s13174-018-0086-3
24. Lynn T, Mooney JG, Lee B, Endo PT (2020) The cloud-to-thing continuum: opportunities and challenges in cloud, fog and edge computing. https://doi.org/10.1007/978-3-030-41110-7
25. Wen Z, Yang R, Garraghan P, Lin T, Xu J, Rovatsos M (2017) Fog orchestration for internet of things services. IEEE Internet Comput. https://doi.org/10.1109/MIC.2017.36
26. Jiang Y, Huang Z, Tsang DH (2018) Challenges and Solutions in Fog Computing Orchestration. IEEE Netw. https://doi.org/10.1109/MNET.2017.1700271
27. Comma-Di L, Abdullaziz OI, Antevski K, Chundrigar SB, Gdowski R, Kuo PH, Mourad A, Yen LH, Zabala A, (2018) Opportunities and challenges of joint edge and Fog orchestration. In 2018 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2018. https://doi.org/10.1109/WCNCW.2018.8369006
28. Velasquez K, Abreu DP, Curado M, Monteiro E (2022) Resource orchestration in 5G and beyond: Challenges and opportunities. Comp Commun 192:311–315
29. Nguyen PH, Ferry N, Erdogan G, Song H, Lavirotte S, Tigli JY, Solberg A (2019) Advances in deployment and orchestration approaches for IoT - A systematic review. In: Proceedings - 2019 IEEE International Congress on Internet of Things, ICIOT 2019 - Part of the 2019 IEEE World Congress on Services. https://doi.org/10.1109/ICIOT.2019.00021
30. Wu Y (2020) Cloud-edge orchestration for the Internet of Things: Architecture and AI-powered data processing. IEEE Internet Things J 8(16):12792–12805
31. Vaquero LM, Cuadrado F, Elkhatib Y, Bernal-Bernabe J, Srirama SN, Zhani MF (2019) Research challenges in nextgen service orchestration. Futur Gener Comput Syst. https://doi.org/10.1016/j.future.2018.07.039.1806.00764
32. Böhm S, Wirtz G (2022a) Towards orchestration of cloud-edge architectures with kubernetes. In: Science and Technologies for Smart Cities: 7th EAI International Conference, SmartCity360°, Virtual Event, December 2-4, 2021, Proceedings, Springer, pp 207–230
33. Böhm S, Wirtz G (2022) Cloud-edge orchestration for smart cities: A review of kubernetes-based orchestration architectures. EAI Endorsed Trans Smart Cities 6(18):e2–e2
34. Fakude NC, Tarwireyi P, Adigun MO, Abu-Mahfouz AM (2019) Fog Orchestrator as an Enabler for Security in Fog Computing: A Review. In: Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019. https://doi.org/10.1109/IMITEC45504.2019.9015896
35. Šatkauskas N, Venčkauskas A, Morkevičius N, Liutkevičius A (2020) Orchestration Security Challenges in the Fog Computing. In: International Conference on Information and Software Technologies, Springer, pp 196–207
36. Al-Doghman F, Moustafa N, Khalil I, Sohrabi N, Tari Z, Zomaya AY (2023) AI-Enabled Secure Microservices in Edge Computing: Opportunities and Challenges. IEEE Trans Serv Comp. 16(2):1485-1504. https://doi.org/10.1109/TSC.2022.3155447
37. Hong CH, Varghese B (2019) Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. ACM Comput Surv (CSUR) 52(5):1–37
38. Hong CH, Varghese B (2019) Resource Management in Fog/Edge Computing. ACM Comput Surv. https://doi.org/10.1145/3326066
39. Toczé K, Nadjm-Tehrani S (2018). A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing. https://doi.org/10.1155/2018/7476201
40. Ghobaei-Arani M, Souri A, Rahmanian AA (2020). Resource Management Approaches in Fog Computing: a Comprehensive Review. https://doi.org/10.1007/s10723-019-09491-1
41. Luo Q, Hu S, Li C, Li G, Shi W (2021) Resource scheduling in edge computing: A survey. IEEE Commun Surv Tutor 23(4):2131–2165
42. Duc TL, Leiva RG, Casari P, Östberg PO (2019) Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. ACM Comput Surv (CSUR) 52(5):1–39
43. Raj P, Raman A (2018). Automated Multi-cloud Operations and Container Orchestration. https://doi.org/10.1007/978-3-319-78637-7_9
44. Bellendorf J, Mann ZÁ (2018) Cloud topology and orchestration using TOSCA: A systematic literature review. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). https://doi.org/10.1007/978-3-319-99819-0_16
45. Ranjan R, Benatallah B, Dustdar S, Papazoglou MP (2015) Cloud Resource Orchestration Programming: Overview. Issues, and Directions. https://doi.org/10.1109/MIC.2015.20

46. Weerasiri D, Barukh MC, Benatallah B, Sheng QZ, Ranjan R (2017). A taxonomy and survey of cloud resource orchestration techniques. https://doi.org/10.1145/3054177
47. Lauwers C, Tamburri D OASIS Topology and Orchestration Specification for Cloud Applications. www.oasis-open.org/committees/tosca. Accessed 6 Dec 2021
48. (2020) Oasis. https://www.oasis-open.org/. Accessed 6 Aug 2023
49. AWS (2022) AWS Greengrass. https://aws.amazon.com/greengrass/. Accessed 28 Oct 2022
50. Azure (2022) What is Azure IoT Edge. https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge?view=iotedge-1.4. Accessed 28 Oct 2022
51. Google (2022) Google Distributed Cloud Edge. https://cloud.google.com/distributed-cloud. Accessed 28 Oct 2022
52. Alibaba (2022) Link IoT Edge. https://www.alibabacloud.com/product/linkiotedge. Accessed 28 Oct 2022
53. IBM (2022) IBM Edge Application Manager. https://www.ibm.com/cloud/edge-application-manager. Accessed 28 Oct 2022
54. Akimi (2022) Akami EdgeWorkers. https://developer.akamai.com/akamai-edgeworkers-overview. Accessed 28 Oct 2022
55. Linux foundation (2022) Project eve. https://www.lfedge.org/projects/eve/. Accessed 4 Sep 2022
56. Open Horizon (2022) Open horizon. https://open-horizon.github.io/docs/. Accessed 4 Sep 2022
57. EdgeX Foundry (2022) Edgex foundry. https://www.edgexfoundry.org. Accessed 4 Sep 2022
58. Goethals T, De Turck F, Volckaert B (2020) Fledge: Kubernetes compatible container orchestration on low-resource edge devices. In: Internet of vehicles : technologies and services toward smart cities, 6th International Conference, IOV 2019, Proceedings, Springer, pp 174–189
59. KubeEdge (2022) Kubeedge: A kubernetes native edge computing framework. https://kubeedge.io/en/. Accessed 4 Sep 2022
60. KubeFed (2022) Kubernetes federation project. https://github.com/kubernetes-sigs/kubefed. Accessed 4 Sep 2022
61. Submariner (2022) Submariner, connected kubernetes overlay networks. https://github.com/submariner-io/submariner. Accessed 4 Sep 2022
62. Starlingx (2022) Starlingx: Distributed edge cloud native platform. https://www.starlingx.io/. Accessed 4 Sep 2022
63. Openiotfog (2022) Openiotfog: Edge computing for industry 4.0 applications. https://openiotfog.org. Accessed 4 Sep 2022
64. Fornax-project (2022) Fornax - and edge computing framework. https://github.com/centaurusinfra/fornax. Accessed 31 Oct 2022
65. Centaurus-project (2022) Centaurus - An infrastructure platform for distributed cloud. https://www.centauruscloud.io/. Accessed 31 Oct 2022
66. Wang N, Varghese B, Matthaiou M, Nikolopoulos DS (2017) Enorm: A framework for edge node resource management. IEEE Trans Serv Comput 13(6):1086–1099
67. Fernandez JM, Vidal I, Valera F (2019) Enabling the orchestration of iot slices through edge and cloud microservice platforms. Sensors 19(13):2980
68. GSMA (2022) Gsma network slicing: Use case requirements. https://www.gsma.com/futurenetworks/wp-content/uploads/2018/04/NS-Final.pdf. Accessed 2 Sep 2022
69. Alam M, Rufino J, Ferreira J, Ahmed SH, Shah N, Chen Y (2018) Orchestration of microservices for iot using docker and edge computing. IEEE Commun Mag 56(9):118–123
70. Santos J, Wauters T, Volckaert B, De Turck F (2017) Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. Entropy 20(1):4
71. ETSI (2022) NFV in ETSI. https://www.etsi.org/technologies/nfv. Accessed 4 Sep 2022
72. Moy J (1997) OSPF Version 2. https://www.rfc-editor.org/rfc/rfc2178. Accessed 13 Oct 2022
73. Yigitoglu E, Mohamed M, Liu L, Ludwig H (2017) Foggy: A framework for continuous automated iot application deployment in fog computing. In: 2017 IEEE International Conference on AI Mobile Services (AIMS), pp 38–45. https://doi.org/10.1109/AIMS.2017.14
74. Castellano G, Esposito F, Risso F (2019) A service-defined approach for orchestration of heterogeneous applications in cloud/edge platforms. IEEE Trans Netw Serv Manag 16(4):1404–1418
75. Jimenez LL, Schelen O (2020) Hydra: Decentralized location-aware orchestration of containerized applications. IEEE Trans Cloud Comput 10(4):2664–2678
76. Pires A, Simão J, Veiga L (2021) Distributed and decentralized orchestration of containers on edge clouds. J Grid Comput 19:1–20
77. De Brito MS, Hoque S, Magedanz T, Steinke R, Willner A, Nehls D, Keils O, Schreiner F (2017) A service orchestration architecture for fog-enabled infrastructures. In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), IEEE, pp 127–132
78. Tsagkaropoulos A, Verginadis Y, Compastié M, Apostolou D, Mentzas G (2021) Extending tosca for edge and fog deployment support. Electronics 10(6):737
79. Kumara I, Mundt P, Tokmakov K, Radolović D, Maslennikov A, González RS, Fabeiro JF, Quattrocchi G, Meth K, Nitto ED, et al (2021) Sodalite@rt: Orchestrating applications on cloud-edge infrastructures. J Grid Comput 19(3). https://doi.org/10.1007/s10723-021-09572-0
80. Zhang W, Li S, Liu L, Jia Z, Zhang Y, Raychaudhuri D (2019) Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, pp 1270–1278
81. Donassolo B, Fajjari I, Legrand A, Mertikopoulos P (2019) Fog based framework for IoT service provisioning. In: 2019 16th IEEE annual consumer communications & networking conference (CCNC), IEEE, pp 1–6
82. Ferry N, Nguyen P, Song H, Novac PE, Lavirotte S, Tigli JY, Solberg A (2019) Genesis: Continuous orchestration and deployment of smart iot systems. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol 1. IEEE, pp 870–875
83. Morin B, Fleurey F, Husa KE, Barais O (2016) A generative middleware for heterogeneous and distributed services. In: 2016 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE), IEEE, pp 107–116
84. Cozzolino V, Ott J, Ding AY, Mortier R (2020) Ecco: Edge-cloud chaining and orchestration framework for road context assessment. In: 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI), IEEE, pp 223–230
85. Yang S, Ren Y, Zhang J, Guan J, Li B (2021) Kubehice: Performance-aware container orchestration on heterogeneous-isa architectures in cloud-edge platforms. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), IEEE, pp 81–91
86. Gand F, Fronza I, El Ioini N, Barzegar HR, Azimi S, Pahl C (2020) A fuzzy controller for self-adaptive lightweight edge container orchestration. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science-CLOSER, SciTePress, pp 79–90
87. Sonmez C, Ozgovde A, Ersoy C (2019) Fuzzy workload orchestration for edge computing. IEEE Trans Netw Serv Manag 16(2):769–782
88. Pahl C, El Ioini N, Helmer S, Lee B (2018) An architecture pattern for trusted orchestration in IoT edge clouds. In: 2018 third international conference on fog and mobile edge computing (FMEC), IEEE, pp 63–70
89. El Ioini N, Pahl C (2018) Trustworthy orchestration of container based edge computing using permissioned blockchain. 2018 Fifth International Conference on Internet of Things: Systems. Management and Security, IEEE, pp 147–154
90. Kochovski P, Stankovski V, Gec S, Faticanti F, Savi M, Siracusa D, Kum S (2020) Smart contracts for service-level agreements in edge-to-cloud computing. J Grid Comput 18(4):673–690
91. Liu Q, Xia T, Cheng L, Van Eijk M, Ozcelebi T, Mao Y (2021) Deep reinforcement learning for load-balancing aware network control in IoT edge systems. IEEE Trans Parallel Distrib Syst 33(6):1491–1502
92. Liu Q, Cheng L, Ozcelebi T, Murphy J, Lukkien J (2019) Deep reinforcement learning for IoT network dynamic clustering in edge computing. 2019 19th IEEE/ACM international symposium on cluster. Cloud and Grid Computing (CCGRID), IEEE, pp 600–603
93. Mao Y, Sharma V, Zheng W, Cheng L, Guan Q, Li A (2023) Elastic Resource Management for Deep Learning Applications in a Container Cluster. IEEE Trans Cloud Comp 11(2):2204–2216. https://doi.org/10.1109/TCC.2022.3194128
94. Mao Y, Fu Y, Zheng W, Cheng L, Liu Q, Tao D (2021) Speculative container scheduling for deep learning applications in a kubernetes cluster. IEEE Syst J 16(3):3770–3781

95. Mao Y, Yan W, Song Y, Zeng Y, Chen M, Cheng L, Liu Q (2023) Differentiate Quality of Experience Scheduling for Deep Learning Inferences With Docker Containers in the Cloud. IEEE Transactions on Cloud Computing 11(2):1667-1677. https://doi.org/10.1109/TCC.2022.3154117
96. Morris K (2016) Infrastructure as code: managing servers in the cloud. O'Reilly Media, Inc
97. Ansible (2022) Ansible documentation. https://docs.ansible.com/ansible/latest/index.html. Accessed 30 Sep 2022
98. Taherizadeh S, Stankovski V, Grobelnik M (2018) A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. Sensors 18(9). https://doi.org/10.3390/s18092938. https://www.mdpi.com/1424-8220/18/9/2938
99. Verginadis Y, Apostolou D, Taherizadeh S, Ledakis I, Mentzas G, Tsagkaropoulos A, Papageorgiou N, Paraskevopoulos F (2021) Prestocloud: a novel framework for data-intensive multi-cloud, fog, and edge function-as-a-service applications. Inf Resour Manag J 34(1):66–85
100. Verginadis Y, Alshabani I, Mentzas G, Stojanovic N (2017) Prestocloud: Proactive cloud resources management at the edge for efficient real-time big data processing. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science - CLOSER. p. 611–617. SciTePress, NSTICC, https://doi.org/10.5220/0006359106110617
101. Masip-Bruin X, Marín-Tordera E, Sánchez-López S, Garcia J, Jukan A, Juan Ferrer A, Queralt A, Salis A, Bartoli A, Cankar M et al (2021) Managing the cloud continuum: Lessons learnt from a real fog-to-cloud deployment. Sensors 21(9):2974
102. Kochovski P, Stankovski V (2021) Building applications for smart and safe construction with the decenter fog computing and brokerage platform. Autom Constr 124(103):562
103. Pledger-project (2022) Pledger project. http://www.pledger-project.eu/. Accessed 27 Oct 2022
104. Rainbow-project (2022) Rainbow Horizon2020 Project. https://rainbow-h2020.eu/rainbow-platform/. Accessed 22 Oct 2022
105. Slack4things (2023) Slack4things: An openstack-based internet of things framework. http://stack4things.unime.it/. Accessed 21 Apr 2023
106. Merlino G, Dautov R, Distefano S, Bruneo D (2019) Enabling workload engineering in edge, fog, and cloud computing through Openstack-based middleware. ACM Trans Internet Technol 19(2):1–22
107. EUCloudEdgeIoT (2006) Building the European Cloud, Edge & IoT Continuum for business and research. https://eucloudedgeiot.eu. Accessed 20 Apr 2023
108. European Commission (2021) Future European platforms for the Edge: Meta Operating Systems (RIA). https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl4-2021-data-01-05. Accessed 20 Apr 2023
109. European Commission (2022) Cognitive Cloud: AI-enabled computing continuum from Cloud to Edge (RIA). https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl4-2022-data-01-02. Accessed 20 Apr 2023
110. HP (2022) HPE GreenLake. https://www.hpe.com/us/en/greenlake.html. Accessed 12 Sept 2022
111. Intel (2022) Intel Smart Edge Open. https://smart-edge-open.github.io/docs/product-overview/. Accessed 16 Sept 2022
112. Aarna (2022) Aarna Networks Multi Cluster Orchestration Platform (AMCOP). https://www.aarnanetworks.com/products/amcop. Accessed 12 Sept 2022
113. Ormuco (2022) Ormuco IaaS. https://ormuco.com/iaas/. Accessed 12 Sept 2022
114. Azion (2022) Azion Edge Orchestrator. https://www.azion.com/en/documentation/products/edge-orchestrator. Accessed 12 Sept 2022
115. ONAP (2022) Open Network Architecture Platform (ONAP). https://www.onap.org/architecture. Accessed 13 Sept 2022
116. Zededa (2022) ZEDEDA Technologies. https://zededa.com/technologies/. Accessed 13 Sept 2022
117. Lauwers C, Tamburri D OASIS Topology and Orchestration Specification for Cloud Applications. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf. Accessed 27 Nov 2022
118. ETSI (2015) Quality of ict services: Template for service level agreements (SLA). https://tinyurl.com/etsi-SLA. Accessed 9 Dec 2022
119. Alqahtani A, Li Y, Patel P, Solaiman E, Ranjan R (2018) End-to-end service level agreement specification for iot applications. In: 2018 International Conference on High Performance Computing & Simulation (HPCS), IEEE, pp 926–935
120. Antonescu AF, Braun T (2015) Service level agreements-driven management of distributed applications in cloud computing environments. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, pp 1122–1128
121. Antonacci M, Brigandì A, Caballer M, Cetinić E, Davidovic D, Donvito G, Moltó G, Salomoni D (2019) Digital repository as a service: automatic deployment of an invenio-based repository using tosca orchestration and apache mesos. In: EPJ Web of Conferences, vol 214. EDP Sciences, p 07023
122. Costantini A, Duma DC, Martelli B, Antonacci M, Galletti M, Tisbeni SR, Bellavista P, Modica GD, Nehls D, Ahouangonou JC, et al (2021) A cloud-edge orchestration platform for the innovative industrial scenarios of the iotwins project. In: International Conference on Computational Science and Its Applications, Springer, pp 533–543
123. Alzubaidi A, Solaiman E, Patel P, Mitra K (2019) Blockchain-based SLA management in the context of IoT. IT Prof 21(4):33–40
124. Tusa F, Clayman S (2022) End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum. Futur Gener Comput Syst. https://doi.org/10.1016/j.future.2022.11.026. https://www.sciencedirect.com/science/article/pii/S0167739X22003971
125. Ullah A (2017) Towards a novel biologically-inspired cloud elasticity framework. PhD thesis, University of Stirling
126. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Grid Comput 12(4):559–592
127. Ullah A, Li J, Shen Y, Hussain A (2018) A control theoretical view of cloud elasticity: taxonomy, survey and challenges. Clust Comput 21(4):1735–1764
128. Sgambelluri A, Tusa F, Gharbaoui M, Maini E, Toka L, Perez JM, Paolucci F, Martini B, Poe WY, Melian Hernandes J, Muhammed A, Ramos A, de Dios OG, Sonkoly B, Monti P, Vaishnavi I, Bernardos CJ, Szabo R (2017) Orchestration of network services across multiple operators: The 5g exchange prototype. In: 2017 European Conference on Networks and Communications (EuCNC), pp 1–5. https://doi.org/10.1109/EuCNC.2017.7980666
129. Tusa F, Clayman S, Valocchi D, Galis A (2018) Multi-domain orchestration for the deployment and management of services on a slice enabled nfvi. In: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp 1–5. https://doi.org/10.1109/NFV-SDN.2018.8725769
130. Ren J, Zhang D, He S, Zhang Y, Li T (2019) A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Comput Surv (CSUR) 52(6):1–36
131. Fakude NC, Tarwireyi P, Adigun MO, Abu-Mahfouz AM (2019) Fog orchestrator as an enabler for security in fog computing: A review. In: 2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC), IEEE, pp 1–6
132. Computing A et al (2006) An architectural blueprint for autonomic computing. IBM White Paper 31(2006):1–6

Publisher's Note