Tomarchio *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2020) 9:49
https://doi.org/10.1186/s13677-020-00194-7

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks

Orazio Tomarchio[†*], Domenico Calcaterra[†] and Giuseppe Di Modica[†]

Abstract

The number of both service providers operating in the cloud market and customers consuming cloud-based services is constantly increasing, proving that the cloud computing paradigm has successfully delivered its potential. Nevertheless, the unceasing growth of the cloud market is posing hard challenges on its participants. On the provider side, the capability of orchestrating resources in order to maximise profits without failing customers' expectations is a matter of concern. On the customer side, the efficient resource selection from a plethora of similar services advertised by a multitude of providers is an open question. In such a multi-cloud landscape, several research initiatives advocate the employment of software frameworks (namely, cloud resource orchestration frameworks - CROFs) capable of orchestrating the heterogeneous resources offered by a multitude of cloud providers in a way that best suits the customer's need. The objective of this paper is to provide the reader with a systematic review and comparison of the most relevant CROFs found in the literature, as well as to highlight the multi-cloud computing open issues that need to be addressed by the research community in the near future.

Keywords: Cloud computing, Cloud resource orchestration, Multi-cloud, Cloud interoperability, Interconnected clouds, Cloud brokerage

Introduction

Over the last few years, cloud computing has established itself as a new model of distributed computing by offering complex hardware and software services in very different fields. As reported in the RightScale 2019 State of the Cloud Report [1], many companies and organisations have successfully adopted the cloud computing paradigm worldwide, while more and more are approaching it as they see a real opportunity to grow their business. According to that report, 94 percent of IT professionals surveyed said their companies are using cloud computing services, and 91 percent are using the public cloud. Organisations leverage almost 5 clouds on average, and companies are running about 40 percent of their workloads in the cloud. The enterprise cloud spend is growing quickly as companies plan to spend 24 percent more on public cloud in 2019 vs. 2018.

The competition between cloud providers is getting stronger in order to acquire increasing market shares: a key point to optimise resource usage and fully exploit the potential of cloud computing is the issue of *resource orchestration* [2]. Cloud resource orchestration regards complex operations such as selection, deployment, monitoring, and run-time control of resources. The overall goal of orchestration is to guarantee full and seamless delivery of applications by meeting Quality of Service (QoS) goals of both cloud application owners and cloud resource providers. Resource orchestration is considered to be a challenging activity because of the scale dimension that resources have reached, and the proliferation

*Correspondence: orazio.tomarchio@unict.it
[†]Orazio Tomarchio, Domenico Calcaterra and Giuseppe Di Modica contributed equally to this work.
Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy

of heterogeneous cloud providers offering resources at different levels of the cloud stack.

Cloud Resource Orchestration Frameworks (CROFs) have emerged as systems to manage the resource lifecycle, from the selection phase to the monitoring one [2–4]. Today most of commercial cloud providers offer a cloud orchestration platform to end-users [5]: however, these products are proprietary and, for obvious business reasons, are not portable. In addition, although modern configuration management solutions exist (e.g., Amazon OpsWorks, Ansible, Puppet, Chef) that provide support for handling resource configuration over cloud services, all potential users (ranging from professional programmers and system administrators to non-expert end-users) are often required to understand various low-level cloud service APIs and procedural programming constructs in order to create and maintain complex resource configurations.

The advent of the multi-cloud computing further exacerbates the already challenging orchestration issues. The multi-cloud paradigm is a very recent technological trend within the cloud computing landscape, which revolves around the opportunity of taking advantage of services and resources provided by multiple clouds [6, 7]. Multi-cloud presumes there is no a priori agreement between cloud providers, and a third party is responsible for the services. That is the case for *Cloud brokerage* scenarios, where a broker intermediates between cloud providers and cloud consumers [8]. In order to enable an effective multi-cloud paradigm, it is essential to guarantee an easy *portability* of applications among cloud providers [9, 10]. This new requirement calls for more powerful resource orchestration mechanisms cross-cutting multiple cloud administrative domains, i.e., capable of dealing with the heterogeneity of the underlying cloud resources and services.

This work explores the many issues of resource orchestration in the cloud landscape. A review of existing works in the addressed field is conducted in order to identify the challenges that have mostly attracted researchers in recent years, and highlight the aspects that have not been fully covered yet. The main contribution of our work is twofold. Firstly, by deeply analysing recently appeared literature, we build a comprehensive taxonomy of desirable features and dimensions useful to characterise CROFs. Then, in accordance with the identified features, we compare several CROFs from both industry and academia. This will help the reader not only to understand the strengths of each framework, but also to identify the unsolved challenges that have to be addressed in the near future.

The remainder of the paper is organised as follows. In "Research methodology" section the methodology followed in our study is described. "Related surveys" section presents a survey of existing works related to our study. In "Analysis framework" section we identify the CROF capabilities which have been used to carry out the review presented in "Review of cROFs" section. In "Critical discussion" section we summarise the results of the review, emphasising current limitations and open challenges. Finally, "Conclusion" section concludes our work.

Research methodology

The primary motivation of this study is to shed light on the recent advances that both industry and academia have made in facing the cloud resource orchestration's issues in the multi-cloud landscape.

With this aim in mind, we identified the fields relevant to our study in order to clearly frame the research scope. Beyond the quite expected *cloud resource orchestration* topic, the following macro topics were also investigated: *cloud interoperability*, *cloud brokerage*, *interconnected clouds*. As outlined in "Introduction" section, cloud resource orchestration deals with the discovery, selection, allocation, and management of cloud resources. When multiple clouds are in place, cloud brokering and interoperability issues due to the simultaneous access to heterogeneous services of interconnected providers cannot be neglected in the analysis of cloud resource orchestration.

We surveyed the literature recently produced in the mentioned fields. Specifically, we sought for proposals, frameworks, prototypes, commercial products somehow addressing the above discussed issues. The databases taken into consideration in this survey are the following: Scopus[1], ACM Digital Library[2], IEEE Xplore Digital Library[3], Elsevier ScienceDirect[4], and SpringerLink[5]. We also took care of filtering out research items that are dated earlier than the last decade.

We found out that many researchers have already published surveys that are relevant to our object of study. Each of these surveys lists and classifies, under different perspectives, numerous initiatives taken under the big umbrella of the cloud resource orchestration field, be them fully-fledged CROFs or minor proposals focusing just on a restricted set of orchestration features. The primary objective of the study proposed in this work is to provide a new, unified analysis of the existing initiatives, which embraces all the analysis perspectives proposed by the past surveys and eventually identifies the missing ones.

Therefore, as shown in Fig. 1, the first step of our study consisted in reviewing the literature surveys with the aim of a) consolidating the list of CROFs and, in general, proposals on which to run a qualitative comparative analysis,

[1] http://www.scopus.com/
[2] https://dl.acm.org/
[3] http://ieeexplore.ieee.org/
[4] https://www.sciencedirect.com/
[5] https://link.springer.com/

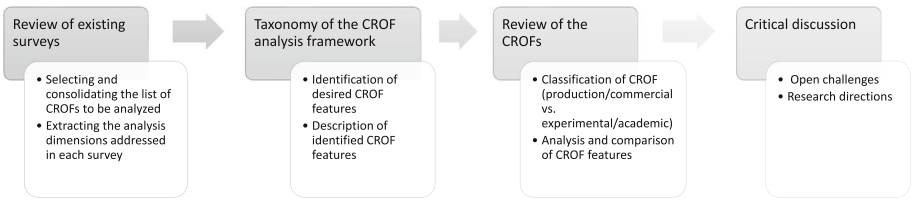

Fig. 1 Research methodology steps

b) extracting the analysis dimensions addressed in each survey. Then, the second step was to elaborate an analysis framework in order to provide a more comprehensive set of features on which a new comparison step would be run. Next, following the references found in the surveys, each CROF on the list was further revised according to the above-mentioned comparative guidelines, and the output of the analysis was eventually gathered in a synoptic table, helping the reader to compare the different features. Finally, the comparison results were the basis for a critical discussion on the state of the art, open challenges and future expectations on CROFs.

Related surveys

This section presents the results of a literature survey we conducted in order to identify published studies that relate to our work to varying degrees. Specifically, we investigated the vast area of cloud computing searching for proposals and initiatives falling under the theme *cloud resource orchestration* in the multi-cloud landscape.

Of particular importance in the context of the discussion were the following works: Inter-cloud Challenges, Expectations and Issues Cluster position paper [11], and the Manifesto for Future Generation Cloud Computing [12]. Both works acknowledged *resource provisioning and orchestration* as an open challenge. In [11], Ferrer et al. recognised it as a research area with a high business impact in the medium term. Besides, in light of more and more heterogeneous cloud resources distributed across diverse cloud typologies and models, both studies stressed the importance of investigating related research areas, such as *cloud interoperability and portability*, service discovery and composition (i.e., *cloud brokerage*), and *interconnected clouds*. The relationship of these related research areas with the main topic of this survey are schematically depicted in Fig. 2. We depicted the multi-cloud resource orchestration research scope as a big umbrella fully covering the cloud resource orchestration research area, and partially sharing themes covered by the cloud brokerage, inter-clouds and cloud interoperability/portability research fields.

We remark that the study conducted in this first investigative step did not intend to seek for actual proposals and initiatives in the focused fields. Instead, it targeted the literature works proposing themselves surveys of the most relevant proposals (step 1 in Fig. 1). Here, the aim is to highlight the limits of existing literature surveys and, thus, to provide a motivation to our work. Also, by "surveying existing literature surveys" we were able to collect the pointers to the actual research proposals, which were the object of investigation in the next steps of our study.

Below, we discuss some of the most representative literature surveys broken down into the four above-mentioned cloud sub-topics. In each of the following sections the subtopic is briefly introduced, and the aspects relevant to the multi-cloud orchestration topic are pointed out.

Cloud interoperability

The cloud computing community typically uses the term *interoperability* to refer to the ability of easily *moving workloads and data from one cloud provider to another* or between private and public clouds [13]. Ten years ago, the standardisation bodies NIST [14], OMG [15] and DMTF [16] developed, among others, several use cases related to cloud interoperability. All the bodies, independently of each other, defined a common umbrella of interoperability use cases covering topics such as user authentication, workload migration, data migration and workload management.

In [17], the authors performed a comprehensive survey on cloud interoperability, with a focus on interoperability among different IaaS cloud platforms. They investigated the existing efforts on taxonomies and standardisation of cloud interoperability, and identified some open issues to advance the research topic as well. Nevertheless, the presented solutions and concepts are mainly focused on IaaS interoperability.

In [18], the authors did their survey on service interoperability and portability on cloud systems with respect to cloud computing service discovery. Still, other interoperability approaches such as the Model Driven Engineering (MDE) and open solutions were not extensively explored.

In [19], the authors described the main challenges regarding cloud federation and interoperability, as well as showcased and reviewed the potential standards to tackle these issues. Similar to [17], their work is restricted to IaaS interoperability, with no other service or deployment models being covered.

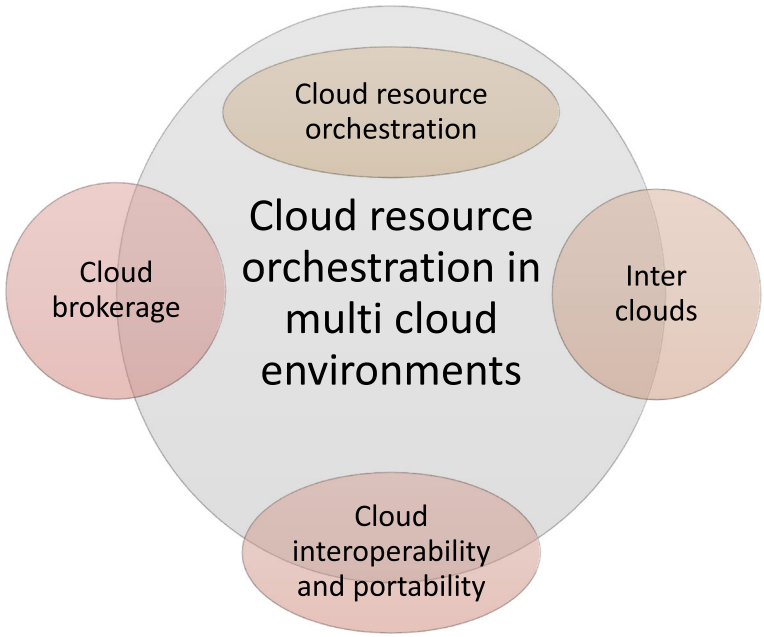

Fig. 2 Related research areas

Cloud brokerage

According to the Gartner definition [20], "Cloud services brokerage is an IT role and business model in which a company or other entity adds value to one or more (public or private) cloud services on behalf of one or more consumers of that service via three primary roles including aggregation, integration and customization brokerage". As defined by NIST [21], a cloud service broker "... is an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers." From these definitions, it is clear that any business player which intends to act as a broker between the cloud consumers and the cloud providers must cope with the diversity of providers and the heterogeneity of the multitude of services the latter offer.

In [6], the authors proposed taxonomies for inter-cloud architectures and application brokering. They presented a detailed survey of both academic and industry developments for inter-cloud, cataloguing many projects and fitting them onto the introduced taxonomies. They also analysed the existing works and identified open challenges in the area of inter-cloud application brokering. Their efforts are nonetheless limited to broker-based strategies.

In [22], a systematic literature survey was conducted to compile studies related to cloud brokerage. The authors presented an understanding of the state of the art and a novel taxonomy to characterise cloud brokers, identifying the main limitations of current solutions and highlighting areas for future research. However, just like [6], their whole analysis only covers broker-based approaches.

Interconnected clouds

Interconnected clouds, also called *Inter-cloud*, can be viewed as a natural evolution of cloud computing. Intercloud has been introduced by Cisco [23] as an interconnected global "cloud of clouds" that mimics the term Inter-net, "network of networks". Basically, the Inter-cloud refers to a mesh of clouds that are unified based on open standard protocols to provide a cloud interoperability.

A more sophisticated definition of Inter-cloud is given by the Global Inter-cloud Technology Forum (GICTF) [24]: "Inter-cloud is a cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through an interworking of cloud systems of different cloud providers based on coordination of each consumer's requirements for service quality with each provider's SLA and use of standard interfaces".

In [8, 9, 25], the author investigated the consumption of resources and services from multiple clouds, as well as proposed a list of requirements for interoperability solutions, highlighting the technological barriers and some well-known solutions for multi-cloud environments. The author did not present the origin of these requirements, nor did she identify the degree of fulfillment of the requirements by theoretical approaches and technical solutions.

In [26], the authors discussed all the relevant aspects motivating cloud interoperability, categorising and identifying cloud interoperability scenarios and architectures. They provided a taxonomy of the main challenges for the

Inter-cloud realisation. A comprehensive review of the state of the art, including standardisation initiatives, ongoing projects and studies in the area, was also conducted.

In [27], the authors analysed the existing literature to identify how interoperability in cloud computing has been addressed. They investigated requirements and usage scenarios for interoperable applications as well as cloud interoperability solutions, presenting a limited list of open issues and directions for future research.

In [28], the authors surveyed the literature to analyse and categorise various solutions for solving the interoperability and portability issues of Interconnected clouds, referring to both user-side (Multi-clouds or Aggregated service by Broker) and provider-side (Federated clouds or Hybrid clouds) scenarios, as specified in [8, 25]. They also performed a comparative analysis of the literature works falling into the same category, and discussed the challenges of Interconnected clouds along the same lines as [17] and [26].

Despite delving into Interconnected clouds, starting with motivation, scenarios, possible solutions for interoperability, and ending with open issues and future directions, all these works ([26–28]) gave limited attention to cloud resource orchestration. In addition, none of them covered aspects pertaining to the application development, deployment, and lifecycle management.

Cloud resource orchestration

In a panorama where organisations get to use many types of cloud computing systems simultaneously, the complexity of the workloads devoted to the management of the life-cycle of resources (data and applications) across the systems dramatically increases. *Cloud orchestration* is the process of managing these multiple workloads, in an automated fashion, across several cloud solutions. Typical activities underlying such a complex process are the resource description, selection, configuration, deployment, monitoring and control. Let us not forget that the orchestration problem is exacerbated by the diversity of the cloud systems, for what concerns both technical and administrative features.

In [2], the authors characterised the cloud resource orchestration in a multi-layered stack, and highlighted the main research challenges involved in programming orchestration operations for different cloud resource types across all layers of a cloud resource stack. The scope of their analysis is nevertheless restricted to the area of cloud resource orchestration.

In [3], the authors proposed a multidimensional taxonomy for classifying and comparing cloud resource orchestration techniques from both industry and academia, identifying open research issues and offering directions for future study. Similar to [2], their work only covers the topic of cloud resource orchestration.

In [29], the authors performed a systematic literature survey to build up a taxonomy of the main research interests regarding TOSCA. Different topics were addressed, such as devising cloud orchestration methods using TOSCA, extending the language of TOSCA, and presenting tools for manipulating TOSCA models. Despite being envisioned as a topic which is expected to play an increasingly important role, *interoperability* received very limited attention.

Analysis framework

In this section we introduce the desired capabilities for CROFs, focusing on deployment and management aspects. From the consumers' standpoint, CROFs implement a service-oriented model which ensures successful hosting and delivery of applications by using cloud resources in order to meet their QoS requirements. Our reference architecture for CROFs is depicted in Fig. 3. Processes and services involved in cloud resource orchestration are categorised depending on their functionalities in relation to this reference model.

The *Access Layer* regulates interaction with the framework. Users can access services from the lower layers by means of CLIs, Web APIs, and Dashboards. The *Application Management Layer* concerns the handling of applications throughout their entire lifecycle, from the Development to the Execution passing through the Deployment. The *Development* refers to languages and models to typically represent applications, workflows, QoS requirements, and policies. Application descriptions define application components as well as their relationships. Workflow descriptions specify the behavioural aspects of applications by means of declarative or imperative approaches. Policy descriptions provide applications with dynamic control behaviours (e.g. defining load-based policies to scale up and down applications) in order to meet QoS requirements. The *Deployment* refers to the actual application deployment on cloud resources, which might go through a preliminary resource discovery process. The *Execution* entails effective automation of complex management tasks, such as scaling and failure handling, which typically require a monitoring engine collecting system and application metrics. Based on the captured metrics, a recovery engine and a policy enforcement engine can determine the decisions to make in order to recover from failures and enforce policies, respectively.

The *Resource Management Layer* includes services (e.g. discovery services, provisioning services, monitoring services) handling resources throughout their whole lifecycle. These services coordinate the required actions from the upper layer by leveraging operations at the Resource Provisioning Layer. The *Resource Provisioning Layer* encompasses services offering the most basic operations regarding cloud resources. A range of provisioning services (e.g.

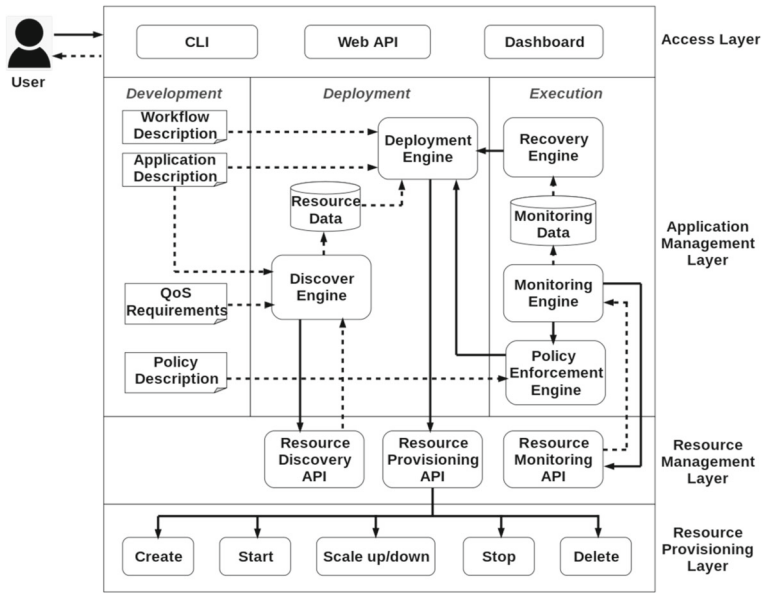

Fig. 3 Reference architecture for CROFs

create, start, scale, stop, and delete) are usually furnished for every supported resource.

Significant research has been done in academic and industry landscapes toward characterisation of cloud orchestration tools. In [4, 30], Baur et al. investigated the required features for such tools, and gave a definition of them. In [31], Ranjan et al. introduced technical dimensions for CROF analysis, thus providing insights into existing frameworks. In [3], Weerasiri et al. identified the main dimensions and common building blocks which characterise cloud resource orchestration solutions.

In [32–34], the authors presented their vision for cloud computing, including views on future research areas, one of them being resource provisioning and orchestration. A thorough analysis of these research areas and related challenges from different perspectives was carried out.

In [35], GigaSpaces Research investigated prevalent approaches for managing applications in cloud environments, namely, orchestration, PaaS (Platform as a Service) and CMP (Cloud Management Platform). A number of categories serving as a common ground for comparison between the different approaches were proposed.

Based on the study of Baur et al.[4], we enriched the list of desirable capabilities pertaining to CROFs by reviewing the literature and integrating the aforementioned works. Such capabilities, summarised in Fig. 4, can be classified into two main categories as either *Cloud Features* or *Application Features*. Details about each set of features are provided in the following subsections.

Cloud features

Cloud features address cloud infrastructure aspects with special focus on supported deployment across multiple cloud providers. Whilst some works [4, 35] investigated features such as multi/cross-cloud support and integration of external services and systems, others [31] focused on capabilities such as interoperability and access modes to CROFs. We propose a comprehensive approach which takes into account all the said aspects that we discuss next.

Multi-cloud support

Supporting *multiple cloud* providers is one of the most crucial features for CROFs, as it allows to select the best matching offer for an application from a diverse cloud landscape. Cloud providers often differ from each other regarding their APIs. For that reason CROFs should offer a cloud abstraction layer (see "Interoperability approach" section), which hides differences and avoids the need for provider-specific customisation causing the vendor lock-in issue.

Cross-cloud support

Cross-cloud support enhances the multi-cloud feature by allowing to distribute component instances of a single application over multiple cloud providers. The advantages of cross-cloud deployment are threefold: a) it allows a sophisticated selection of the best-fitting cloud providers on a per component instance basis, optimising costs or improving quality of services; b) it leverages the application availability as it introduces resilience against the

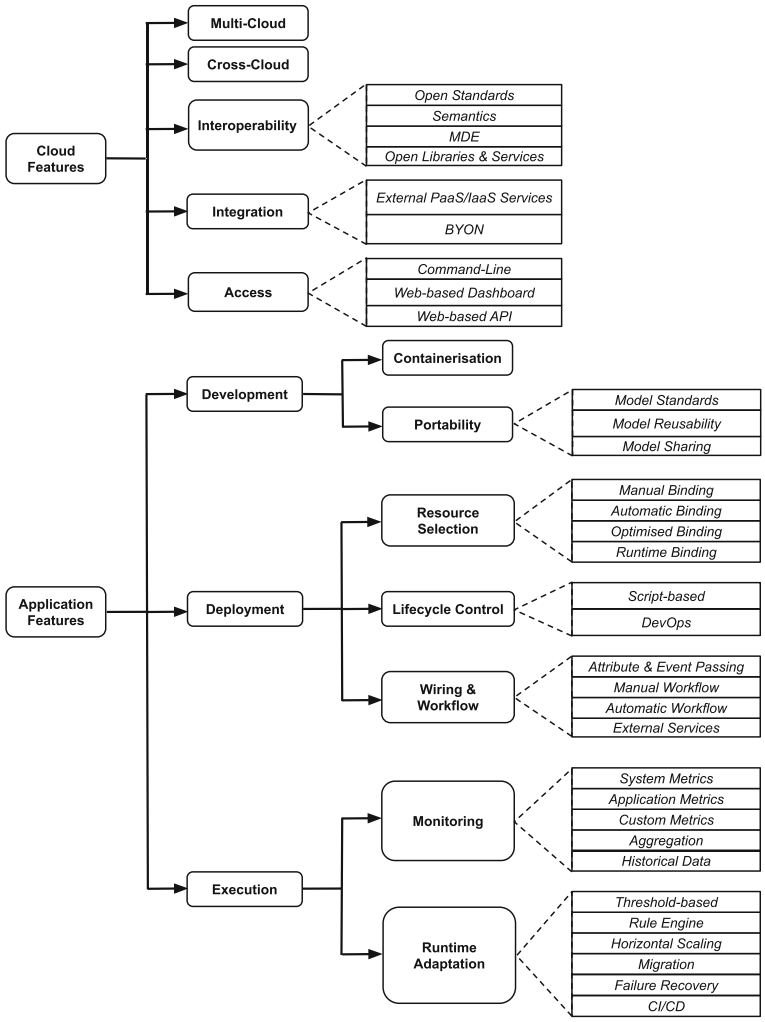

Fig. 4 Taxonomy of the CROF characterisation framework

failure of individual cloud providers; and c) it helps coping with privacy issues.

Interoperability approach

In the context of cloud computing, *interoperability* can be defined as the ability to develop applications that combine resources that can interoperate, or work together from multiple cloud providers, hence taking advantage of specific features provided by each provider [27]. A few research papers [9, 27, 28] comprehensively reviewed the literature in order to dissect the state of the art in cloud interoperability, resulting in a diverse range of approaches falling into the following categories: *open standards*, *semantics*, *model-driven engineering (MDE)*, and *open libraries & services*.

Formulating standards for cloud computing is the most obvious solution for interoperability. Even though a plethora of standards have been proposed so far (e.g., OCCI[6], CIMI[7], OVF[8], CDMI[9], TOSCA [36]), lack of widespread accepted standards necessitates investigating other solutions for interoperability. When cloud providers use different APIs and data models in order to exhibit the same features, semantic interoperability becomes involved. Semantic technologies (e.g, OWL[10], SPARQL[11], SWRL[12]) can prove useful to provide semantic interoperability among different cloud providers. Broker-based approaches can also alleviate semantic interoperability by means of ontology-based interfaces concealing the differences among cloud vendors. Cloud interoperability can also be addressed by exploiting MDE techniques [10].

[6] https://occi-wg.org/about/specification/
[7] https://www.dmtf.org/standards/cmwg
[8] https://www.dmtf.org/standards/ovf
[9] https://www.snia.org/cdmi
[10] https://www.w3.org/TR/owl-syntax/
[11] https://www.w3.org/TR/sparql11-query/
[12] https://www.w3.org/Submission/SWRL/

Another viable solution for cloud interoperability includes open libraries (e.g., Apache jclouds[13], Apache Libcloud[14]) and services, which rely on *abstraction layers* in order to decouple application development from proprietary technologies of cloud providers.

Integration

Support for advanced *IaaS/PaaS services* (e.g., DBaaS, LBaaS, FWaaS) is desirable. It reduces complexity and management efforts for the end user. On a negative note, it comes at the expense of flexibility.

BYON (Bring Your Own Node) captures the ability to use already running servers for application deployment. In particular, it enables the use of servers not managed by a cloud platform or virtual machines on unsupported cloud providers.

Access

This feature captures what interfaces CROFs use to interact with cloud resources. Three types of interfaces are usually supported: command-line, web-based dashboard, and web-based API.

Command-line interfaces wrap cloud-specific API actions as commands or scripts executable through shell environments. Despite command-line interfaces being easier to implement, their usage requires a deep understanding about cloud resources and related orchestration operations.

Web-based dashboards present cloud resources as user-friendly artifacts and resource catalogues. Visual artifacts and catalogues aim at simplifying resource selection, assembly, and deployment. These features make Web-based dashboards simpler and more flexible than command-line interfaces.

Web-based APIs allow other tools and systems (e.g. monitoring tools) to integrate cloud resource management operations into their functionalities. They provide the highest abstraction out of the three interface types.

Application features

Application features address development, deployment, and execution aspects of applications. To this end, unlike all previous works, we collect features according to the application phase they pertain to. For instance, with reference to the development phase, we have identified Portability and Containerisation as relevant features. Furthermore, we also propose a classification of application domains of interest for CROFs.

Application domain

Application domain refers to the types of applications that CROFs have been targeted and customised for. Academic research has been done toward the characterisation of application domains over the past few years [31][37][38]. Grounding on the study of Buyya et al. [37], we classified application domains into two categories: *Scientific applications*, and *Business applications* (see Fig. 5).

Cloud computing systems meet the needs of different types of applications in the scientific domain: *high-performance computing (HPC)* applications, *high-throughput computing (HTC)* applications, and *Large-scale data analytics/Internet of Things (IoT)*, which is a matter of common interest for both scientific and business sectors. In regard to the business domain, cloud computing is the preferred technology for a wide range of applications, from *multi-tier web applications* (e.g., web, mobile, online gaming applications) to *media and content delivery network (CDN)* applications (e.g, video encoding & transcoding, video rendering, video streaming, web/mobile content acceleration).

Portability

Portability has been defined as the capability of a program to be executed on various types of data processing systems without converting the program to a different language and with little or no modification [39]. In the context of cloud computing, portability can be classified into three categories: data portability, function or application portability, and service or platform portability [40]. In particular, *application portability* refers to the ability to define application functionalities in a vendor-agnostic way.

Supporting *open standards* such as CAMP [41] and TOSCA [36] for modelling the application topology and the component lifecycles facilitates the usage of CROFs and further increases the reusability of the topology definition, as it restricts the vendor lock-in issue to cloud provider level. *Reusability* can also be improved via a modularised approach regarding the application description. Methods to achieve modularity include templating, parameterisation, and inheritance. Furthermore, since the initial effort for describing applications and application components is high, *model sharing* by means of existing libraries or marketplaces would be beneficial.

Containerisation

Container-based virtualisation [42] is a key approach for sharing the host operating system kernel across multiple guest instances (i.e., containers), while keeping them isolated. Environment-level containers provide a resource isolation mechanism with little overhead compared to OS-level hypervisors [43]. Moreover, the increased isolation offered by containers allows resource consumption to be configured, controlled, and limited at the instance level.

[13] https://jclouds.apache.org/
[14] http://libcloud.apache.org/

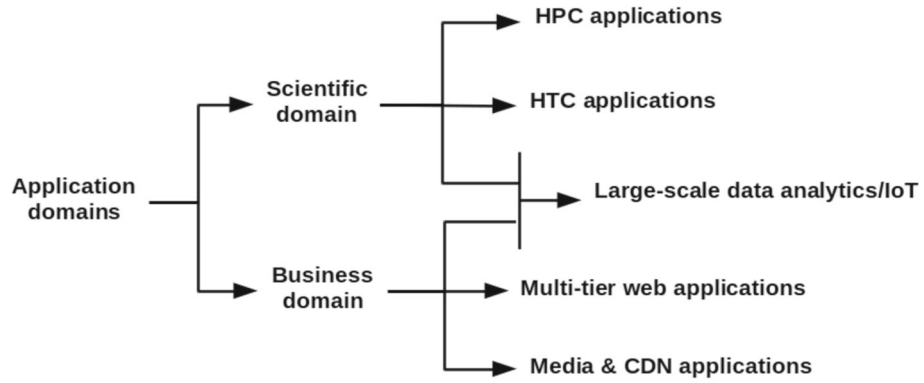

Fig. 5 Application domain classification

Docker is the leading Linux-based platform for developing, shipping, and running applications through container based virtualisation.

Since managing a large amount of containers inside a Docker cluster can be difficult, container-centric orchestrators such as Docker Swarm[15], Google Kubernetes[16], and Apache Mesos[17] have appeared. They perform orchestration at container level by automating the provisioning and management of complex containerised deployments across multiple hosts and locations.

Resource selection

Resource selection refers to the level of automation supported by CROFs with respect to the selection of hardware and software resources. It usually involves identifying and analysing alternative cloud resources based on selection criteria. Resource selection approaches can be classified into four categories.

In a *manual binding* users provide the concrete unique identifiers of the cloud entities. In an *automatic binding* they specify abstract requirements (e.g. number of cores), which CROFs are responsible for binding to a concrete offer at runtime. Automatic binding can be enhanced by offering an *optimised binding*, which leverages optimisation criteria based on attributes of the cloud provider (e.g., price, location) to select the best fitting offer. A *dynamic binding* offers a solving system that enables changes to the binding based on runtime information (e.g., metric data from the monitoring system).

Lifecycle control

Lifecycle control defines the actions that need to be executed in order to fully manage cloud applications. Existing CROFs provide varying levels of automation, typically categorised as *script-based*, and *DevOps* approaches.

A script-based approach consists of a set of shell scripts, which are executed in a specific order. It has limited ability to express dependencies, react to changes, and verify configurations. Script-based approaches can be extended to support DevOps tools (e.g., Chef[18], Puppet[19], Ansible[20]) that offer a more sophisticated approach to deployment management and ready-to-use deployment descriptions.

Wiring & workflow

Most cloud applications are distributed with components residing on different virtual machines. When application deployment takes place, an application instance consisting of one or more component instances gets created. Since dependency relationships may exist between components, the deployment functionality also has the task of *wiring* component instances together.

A straight-forward approach to resolve those dependencies is *attribute and event passing*, in which case lifecycle scripts lock/wait for attributes to become available or register listeners on topology change events. An improvement is a *manual workflow* defined by users in order to take care of the deployment order. Nevertheless, the easiest way for users to deploy applications is an *automatic workflow* deduction from the lifecycle actions defined on components and their relationships. Additionally, CROFs may offer extensions for external services like IaaS/PaaS services (see "Integration" section) to ensure that the deployment engine is aware of this dependency.

Monitoring

Tracking the behaviour of applications is the key to assessing the quality of the deployment and an important building block for adaptation. As a first step this comprises the collection of metrics. CROFs should offer a way to measure *system metrics* (e.g., CPU usage) and *application metrics* (e.g., number of requests). If predefined metrics

[15] https://docs.docker.com/engine/swarm/
[16] https://kubernetes.io/
[17] http://mesos.apache.org/
[18] https://www.chef.io/
[19] https://puppet.com/
[20] https://www.ansible.com/

are not sufficient, a well defined way to add *custom metrics* should be provided. *Aggregation* mechanisms enable to compute higher-level metrics and combine multiple metrics as well. Access to *historical data* is also desirable in order to support a higher-level evaluation of monitoring data.

Runtime adaptation

CROFs should automatically adapt applications in order to deal with dynamic deviations (e.g., increased load). Operations to face such changes are mainly *scaling*, and *migration*. However, the adaptation support of many CROFs is limited to horizontal scaling with *threshold-based* triggers. *Rule engines* leveraging complex metrics and QoS goals would be an improvement.

Since cross-cloud deployments may experience failures, CROFs should also support *recovery* from undesired, erroneous states. Another feature related to adaptation is *continuous integration/continuous delivery (CI/CD)*, which allows to modify the topology model of deployed applications reducing changes to as few as possible.

Review of cROFs

This section presents a selection of CROFs from different landscapes. Notwithstanding that the current state of the art embraces a large number of frameworks, this work contemplates a subset of them which we deem to be representative of the characteristics of the majority of existing solutions. We classify the frameworks in two categories: production/commercial CROFs, and experimental/academic ones.

Production/commercial CROFs are used in a production environment by private and public cloud providers. Whereas some of them are closed-source, others are open-source and supported by a thriving community of developers and users. *Experimental/academic CROFs* usually originate from the research scenery and advance the state of the art, even though their implementation is mostly prototypal.

We discuss next each class of CROFs, and analyse their main capabilities from both cloud and application perspectives, as extensively covered in "Analysis framework" section. Table 1 provides a bird's-eye view of the frameworks taken under consideration. Specifically, each row represents a CROF (*Name*) and specifies the original authors (*Organisation*), basic dates for the initial and latest releases (*Active*), a brief introduction (Description), and the sources consulted (*References*).

Production/commercial cROFs

Nowadays, there is a great variety of production/commercial CROFs around [44], such as *infrastructure-centric* services (e.g., Heat, CloudFormation) provided by cloud providers which are also IaaS providers, *platform-centric* (e.g., Cloud Foundry, OpenShift) and *platform-agnostic* (e.g., Cloudify, Terraform) tools provisioning resources from IaaS providers. In this section we first debate some of the most relevant solutions introduced in Table 1, and subsequently summarise their cloud and application features in Tables 2 and 3 respectively.

Heat

OpenStack Heat [45] is a service for managing the entire life-cycle of infrastructure and applications within OpenStack clouds. It implements an orchestration engine to launch multiple composite cloud applications based on either a CloudFormation compatible template format (CFN) or the native OpenStack Heat Orchestration Template format (HOT). HOT templates are defined in YAML.

A Heat template describes the infrastructure of a cloud application in a declarative fashion, enabling creation of most OpenStack resource types as well as more advanced functions (such as instance high availability, instance autoscaling, and nested stacks) through OpenStack-native REST API calls. The resources, once created, are referred to as *stacks*. Heat templates are consumed by the *OpenStackClient*, which provides a command-line interface (CLI) to OpenStack APIs for launching stacks, viewing details of running stacks, and updating and deleting stacks.

Heat only allows a single-cloud deployment on an OpenStack environment. With reference to interoperability, Heat provides neither semantics nor MDE solutions, but it provides support for TOSCA via the independent Heat-Translator project [21] which translates TOSCA templates to HOT.

Regarding portability, Heat partially supports model standards (TOSCA) and reusability via input parameters, and template composition. It also supports containerisation by means of OpenStack Zun service [22].

Cloud resources can only be selected through manual binding, whereas both manual and automatic workflows can leverage script-based or DevOps tools (such as Chef and Puppet) in order to handle the whole application life-cycle. Heat provides horizontal scaling with threshold triggers based on infrastructure metrics. It partially supports continuous delivery by updating existing stacks, resulting in some resources being updated in-place and others being replaced with brand new resources. Failure recovery capabilities are also supported by means of manual workflows and stacks update.

[21] https://wiki.openstack.org/wiki/Heat-Translator
[22] https://wiki.openstack.org/wiki/Zun

Table 1 List of CROFs under consideration for review purposes

	Name	Organisation	Active	Description	References
	Heat	OpenStack	2010-present	Heat orchestrates composite cloud applications via templates, through both an OpenStack-native API and a CloudFormation-compatible Query API.	[45]
	Cloudify	GigaSpaces	2012-present	Cloudify is a TOSCA-based cloud orchestration framework which enables to model applications and services and automate their entire lifecycle.	[46]
	Brooklyn	Apache	2012-present	Brooklyn is a cloud orchestration framework implementing OASIS CAMP that allows to deploy and manage applications via declarative blueprints.	[47]
	Stratos	Apache	2013-2017	Stratos is a polyglot PaaS framework that helps model and run composite and scalable applications on all major cloud infrastructures.	[48]
	Alien4Cloud	FastConnect	2014-present	Alien4Cloud is a web-based platform providing means to model, deploy and manage TOSCA-based applications via a TOSCA runtime engine.	[49]
	Terraform	HashiCorp	2014-present	Terraform is an infrastructure-as-code tool that enables to provision, and manage infrastructures using a high-level configuration language.	[50]
Commercial	CloudFormation	AWS	2011-present	CloudFormation is an infrastructure-as-code tool that helps model and set up AWS infrastructure resources by means of a JSON encoded template.	[51]
	Cloudiator	University of Ulm	2015-2017	Cloudiator is a cross-cloud orchestration tool that allows to describe an application once and deploy it on different public and private cloud providers.	[52, 53]
	Roboconf	University of Grenoble Alpes	2014-2017	Roboconf is both a platform and a framework tool to deploy and manage elastic cloud applications using automatic reactions and reconfigurations.	[54]
	INDIGO	INDIGO consortium	2015-2017	INDIGO is a data and computing platform targeted at scientific communities, which optimises application execution on cloud and grid infrastructures.	[55, 56]
	MiCADO	COLA consortium	2017-2019	MiCADO is a highly customisable multi-cloud orchestration and auto-scaling framework for Docker containers, orchestrated by Kubernetes.	[57]
	MODAClouds	MODAClouds consortium	2012-2015	MODAClouds is a toolbox and a runtime platform for the design and automatic deployment of applications on multiple clouds with guaranteed QoS.	[58, 59]
Academic	SeaClouds	SeaClouds consortium	2013-2016	SeaClouds is a framework that enables seamless adaptive multi-cloud management of service-based applications over multiple heterogeneous clouds.	[60, 61]

Table 2 Cloud-based comparison of production/commercial CROFs

	CROFs						
Cloud Features	**Heat**	**Cloudify**	**Brooklyn**	**Stratos**	**Alien4Cloud**	**Terraform**	**CloudFormation**
Multi-Cloud	**x**	✓	✓	✓	✓	✓	**x**
Cross-Cloud	**x**	✓	✓	✓	✓	✓	**x**
Interoperability							
- Open Standards	**0**	**0**	**x**	**x**	**0**	**x**	**x**
- Semantics	**x**	**x**	**x**	**x**	**x**	**x**	**x**
- MDE	**x**	**x**	**x**	**x**	**x**	**x**	**x**
- Open Libraries & Services	**x**	**x**	✓	✓	**x**	**x**	**x**
Integration							
- External IaaS/PaaS services	**x**	**x**	**x**	**x**	**x**	✓	**x**
- BYON	**x**	✓	✓	**x**	✓	**x**	**x**
Access							
- Command-Line	✓	✓	✓	✓	✓	✓	✓
- Web-based Dashboard	✓	✓	✓	✓	✓	✓	✓
- Web-based API	✓	✓	✓	✓	✓	✓	✓

x = not fulfilled, **0** = partially fulfilled, ✓ = fully fulfilled

Cloudify

Cloudify [46] is an open-source orchestration framework based on TOSCA. It provides services in order to model applications and automate their entire life-cycle through a set of built-in workflows. Application templates are referred to as *blueprints*, which are YAML documents written in *Cloudify's DSL (Domain Specific Language)*. Blueprints are normally consumed by the Cloudify CLI, which includes all of the commands necessary to run any actions on *Cloudify Manager*.

Typical blueprints contain declarations for various resource types, including cloud resources. Cloudify allows multi-cloud and cross-cloud deployments by means of built-in plugins. It also supports BYON, and leverages TOSCA for interoperability and portability. However, despite being aligned with the modelling standard, Cloudify's DSL does not directly reference the standard types.

Cloudify supports containerisation using Docker. Container orchestration is also available through Kubernetes. Cloud resources can only be selected through manual binding, whereas both manual and automatic workflows can leverage script-based or DevOps tools (such as Ansible, Chef, and Puppet) in order to handle the application life-cycle. Cloudify provides infrastructure, application, and custom metrics. It also enables the definition of custom aggregations and policies using Clojure[23] and Riemann[24].

Cloudify offers built-in workflows for application healing (by applying the uninstall and install workflows' logic, respectively) and horizontal scaling. Complex scenarios (e.g., vertical scaling, cloud bursting) are not supported out of the box. Live migration is partially-fulfilled in the context of containerised applications, though. Multiple pods with containerised applications can be moved between nodes in the same Kubernetes cluster, without service disruption. Continuous delivery is supported through deployment updates, which allow to modify a running topology by adding/removing/modifying nodes. Modifying existing nodes will cause their automatic reinstallation, though.

Brooklyn

Apache Brooklyn [47] is an open-source framework for modelling, deploying, and managing distributed applications defined using declarative YAML blueprints written in *Brooklyn's DSL*. Brooklyn's YAML format follows the CAMP specification [41], but uses some custom extensions. Support for TOSCA is planned for the near future. Blueprints are usually consumed by the Brooklyn client CLI in order to access a running Brooklyn Server. A web console and powerful REST-APIs are available as well.

Brooklyn allows multi-cloud and cross-cloud deployments on many public and private clouds. It also supports private infrastructures (BYON), and leverages Apache jclouds as cloud abstraction layer for interoperability. Portability is achieved via model reusability mechanisms (e.g, type inheritance) and model sharing (e.g, types

[23] https://clojure.org/
[24] http://riemann.io/

Table 3 Application-based comparison of production/commercial CROFs

	CROFs						
	Application Features	**Heat**	**Cloudify**	**Brooklyn**	**Stratos**	**Alien4Cloud**	**Terraform**	**CloudFormation**
	Portability							
	- Model Standards	**0**	**0**	**0**	**x**	**0**	**x**	**x**
	- Model Reusability	**0**	✓	**0**	**0**	✓	✓	**0**
	- Model Sharing	**x**	✓	✓	**0**	✓	✓	**x**
Development	**Containerisation**	✓	✓	**x**	✓	✓	✓	✓
	Resource Selection							
	- Manual Binding	✓	✓	✓	✓	✓	✓	✓
	- Automatic Binding	**x**	**x**	**0**	**x**	**x**	**x**	**x**
	- Optimised Binding	**x**	**x**	**x**	**x**	**x**	**x**	**x**
	- Dynamic Binding	**x**	**x**	**x**	**x**	**x**	**x**	**x**
	Lifecycle Control							
	- Script-based	✓	✓	✓	**x**	✓	✓	✓
	- DevOps	✓	✓	✓	✓	✓	✓	✓
	Wiring & Workflow							
	- Attribute & Event Passing	✓	✓	**0**	✓	✓	✓	✓
	- Manual Workflow	✓	✓	**x**	✓	✓	**x**	**x**
	- Automatic Workflow	✓	✓	**x**	**x**	✓	✓	✓
Deployment	- External Services	**x**	**x**	**x**	**x**	**x**	✓	**x**
	Monitoring							
	- System Metrics	✓	✓	**x**	✓	✓	✓	✓
	- Application Metrics	**x**	✓	**x**	✓	✓	**x**	✓
	- Custom Metrics	**x**	✓	✓	**x**	✓	**x**	✓
	- Aggregation	✓	✓	**0**	✓	✓	✓	✓
	- Historical Data	**x**	✓	**x**	**x**	✓	**x**	✓
	Runtime Adaptation							
	- Threshold-based	✓	✓	✓	✓	✓	✓	✓
	- Rule Engine	**x**	✓	**x**	✓	✓	**x**	**x**
	- Horizontal Scaling	✓	✓	✓	✓	✓	✓	✓
	- Migration	**x**	**0**	**x**	**0**	**0**	**x**	**0**
	- Failure Recovery	✓	✓	**0**	✓	✓	**x**	✓
Execution	- CI/CD	**0**	✓	**0**	**x**	✓	✓	✓

x = not fulfilled, **0** = partially fulfilled, ✓ = fully fulfilled

shared either locally or in a Git repository). Brooklyn does not support containers out of the box. However, containerisation can be integrated by means of separate projects (e.g, Cloudsoft Clocker[25]).

Brooklyn supports manual as well as basic automatic binding for resource selection, whereas it does not support workflow scenarios. Life-cycle actions (i.e. *effectors*) for entities can be configured through either shell scripts or Chef recipes. Brooklyn pulls metrics by either executing remote actions or accessing an external monitoring tool. Nevertheless, it is the user's responsibility to implement those actions, or to provide an interface to an external monitoring tool.

Metrics/QoS can be fed into policies, which automatically take actions such as restarting failed nodes, or scaling out. By default, a threshold-based policy is available. Continuous delivery is exclusively possible on component level, namely

[25] http://www.clocker.io/

by redeploying single components with updated software.

Stratos

Apache Stratos [48] is an open-source PaaS framework which allows developers to build distributed applications and services. Applications are typically composed of sets of *cartridges* representing descriptions of abstract VMs hosting both business and infrastructure services, combined with deployment and scaling policies. Stratos defines configurations and applications in a specific JSON format, therefore they can be shared. Reusability is limited, since cartridges contain references to IDs of IaaS snapshots and hardware configuration. Applications can be managed by means of Stratos CLI. A web console and powerful REST-APIs are available as well.

Stratos support multiple providers and utilises *Apache jclouds* as cloud abstraction layer for interoperability. Despite using jclouds, BYON is not supported. No external services are supported either. Stratos leverages *Kubernetes* as a cluster orchestration framework in order to provide containerisation. Cloud resources are manually selected when configuring cartridges. In addition, while the life cycle description for managing VMs is done by Stratos itself, the software setup is delegated to Puppet. Only manual workflows are supported.

Stratos uses a cartridge agent residing within each VM in order to access system and application metrics. It is not possible to define custom metrics. Using *in-flight requests*, *load average*, and *free memory* metrics combined with a complex event processor and the Drools rule engine[26], Stratos enacts a multi-factored horizontal auto-scaling. It also includes cloud bursting, allowing to seamlessly migrate applications between clouds. Recovery actions are supported in case some tasks within VMs of an application topology fail, by automatically destroying and recreating the affected cartridge instance. Continuous delivery is not supported, since users need to undeploy applications before changing their definitions.

Alien4Cloud

Alien4Cloud (Application LIfecycle ENabler for cloud) [49] is an open-source platform that makes application management on the cloud easy for enterprises. It leverages other existing open-source projects that help orchestrating cloud applications and focus on run-time aspects (e.g., Cloudify). In Alien4Cloud, applications templates (*blueprints*) are modelled in TOSCA in order to allow interoperability and portability. Blueprints can also be shared across platform users via a maintained TOSCA catalog. However, Alien4Cloud supports a slightly modified version of TOSCA Simple Profile.

Application deployment is done through an *orchestrator* on a *location* configured for and managed by an orchestrator. Alien4Cloud supports a number of orchestrators (Cloudify, Puccini[27], and Marathon[28]) via plugins. Locations describe a logical deployment target ranging from private/public clouds to a set of physical machines (BYON), or even Docker containers (Kubernetes and Mesos). Multi-cloud and cross-cloud deployments are supported.

Cloud resources can only be selected through manual binding (*node substitution*), whereas both manual and automatic workflows can leverage script-based or DevOps tools (such as Ansible, Chef, and Puppet) in order to handle the application life-cycle. Regarding monitoring and run-time adaptation, since Cloudify can be used as Alien4Cloud's backend orchestration solution, the same considerations apply. In particular, Alien4Cloud supports horizontal scaling as well as continuous delivery.

Terraform

Terraform [50] is an open-source *infrastructure as code* tool for building, changing, and versioning infrastructures in a platform-agnostic way. It uses its own high-level configuration language known as *Hashicorp Configuration Language (HCL)*, or optionally JSON, in order to detail the infrastructure setup. Despite being non-compliant with any model standards, HCL supports reusability via *modules* and *module composition*. Reusable modules can also be shared by means of the *Terraform Registry* as well as other sources (e.g., GitHub, Bitbucket). Configurations are usually consumed by the *Terraform CLI*, but *Terraform Enterprise* also provides both a web-based dashboard and REST APIs.

Terraform can manage multiple cloud providers and even cross-cloud dependencies by means of special plugins called *providers*. Providers are available for Docker containers and container orchestration as well as external cloud services (e.g. Amazon RDS[29]). However, no support is provided for BYON. Cloud resources are manually selected during configuration, while life-cycle actions can be configured through *provisioners* executing scripts or running configuration management (Chef, Puppet, Salt). Only automatic workflows are supported.

Terraform leverages *providers* in order to provide autoscaling capabilities with threshold triggers on system metrics gathered by monitoring services (e.g., Azure Monitor[30], Amazon CloudWatch[31]). Continuous delivery is supported by *applying* configuration updates, which allow to add/remove/modify resources. When resource arguments cannot be updated in-place, the existing resource

[26] https://www.drools.org/

[27] https://github.com/tliron/puccini
[28] http://mesos.apache.org/
[29] https://docs.aws.amazon.com/rds/index.html
[30] https://docs.microsoft.com/en-us/azure/azure-monitor/
[31] https://docs.aws.amazon.com/cloudwatch/index.html

will be replaced by a new one instead. No recovery actions are supported out of the box, since any errors need to be addressed manually.

CloudFormation

AWS CloudFormation [51] is a template-based *infrastructure-as-code* tool for managing AWS infrastructure deployments. All resources and dependencies are declared in a JSON or YAML template, which CloudFormation uses as a blueprint for building AWS resources. A collection of managed resources is called *stack*. Although CloudFormation templates do not comply with any model standards, reusability is partially supported via input parameters, and nested stacks. Templates are usually consumed by the CloudFormation console, or REST APIs, or CLI.

CloudFormation can only model and manage AWS resources. No support is provided for multiple cloud providers or BYON. Containerisation is natively supported via *Elastic Container Service (ECS)*[32] resources. Container orchestration is also supported by means of *Elastic Kubernetes Service (EKS)*[33] resources as well. Cloud resources are selected through manual binding, whereas lifecycle actions can be configured through user-data scripts or DevOps tools (Chef, Puppet). Only automatic workflows are supported.

CloudFormation provides automatic scaling capabilities by means of *AWS Auto Scaling*[34], which uses *dynamic scaling* and *predictive scaling* to automatically scale resources based on *Amazon CloudWatch* metrics. Customised metrics for *Application Auto Scaling* can also be defined. Live migration is partially-fulfilled in the context of containerised applications. For instance, it's possible to gracefully migrate existing applications from a worker node group to another. Continuous delivery is supported by stack updates. Depending on the resource and properties being updated, an update might interrupt or even replace an existing resource. Recovery actions are supported by automatically rolling back the existing stack on failure.

Experimental/academic cROFs

In this section, we initially review an ensemble of significant experimental/academic CROFs outlined in Table 1, and then summarise them according to their cloud and application features in Tables 4 and 5 respectively. Additionally, we briefly run through other research initiatives focusing only on specific aspects of CROFs.

[32] https://docs.aws.amazon.com/ecs/index.html
[33] https://docs.aws.amazon.com/eks/
[34] https://aws.amazon.com/autoscaling/

Cloudiator

Cloudiator [52, 53, 62] is an open-source cross-cloud orchestration framework, which relies on Apache jclouds in order to support many public and private cloud platforms. The main orchestration component, namely *Colosseum*, can be accessed via a Java client, or a web-based user interface, or a REST-API.

The application description consists of individual *components*, which are assembled to form a full application. Each component provides interface operations (e.g., bash scripts) for managing the component life-cycle. Dependencies between application components are described through communication entities linking *provided ports* and *required ports*. Despite being non-compliant to any modelling standards, application components are reusable across different applications.

The *resource broker* is responsible for automatically selecting the correct cloud offer (previously discovered by the *discovery engine*), depending on the desired requirements/constraints on virtual machine configuration. The *deployment engine* acquires the virtual machine and forwards the component installation request to the remote life-cycle agent, namely *Lance*. Lance runs component instances within Docker containers by default. In addition, only automatic workflows are supported.

Automatic scaling capabilities are provided by means of *AXE*, a monitoring and adaptation engine embedded in Cloudiator, which implements scalability rules consisting of threshold-based conditions linked to raw or composed metrics. Migration features are partially fulfilled by supporting access to OpenStack's live migration functionality. Recovery actions are supported by the *recovery engine*, which detects abnormal states of system entities marking them as failed, and applies solutions based on failure categories. The same mechanism is used in order to represent changes in the models (continuous delivery).

Roboconf

Roboconf [54, 63] is an open-source scalable orchestration framework for multi-cloud platforms. Many IaaS providers (e.g., OpenStack, AWS, Azure, vSphere), as well as Docker containers and local deployments for on-premise hosts, are supported by using special plugins. Roboconf partially supports interoperability by means of OCCI extensions and a generic target implementation based on Apache jclouds. In addition, it can be accessed by means of a shell-based console, or a web-based user interface, or a REST API.

Roboconf provides a CSS-inspired DSL, which allows to describe applications and their execution environments in a hierarchical way. A distributed application is seen as a set of *components*, building an acyclic graph describing both containment and run-time relationships between

Table 4 Cloud-based comparison of experimental/research CROFs

	CROFs					
Cloud Features	**Cloudiator**	**Roboconf**	**INDIGO-DataCloud**	**MiCADO**	**MODAClouds**	**SeaClouds**
Multi-Cloud	✓	✓	✓	✓	✓	✓
Cross-Cloud	✓	✓	✓	✓	✓	✓
Interoperability						
- Open Standards	**x**	**0**	✓	**0**	**x**	**0**
- Semantics	**x**	**x**	**x**	**x**	**x**	**x**
- MDE	**x**	**x**	**x**	**x**	✓	**x**
- Open Libraries & Services	✓	**0**	**x**	**x**	✓	✓
Integration						
- External IaaS/PaaS services	**x**	✓	**x**	**x**	✓	✓
- BYON	**x**	✓	✓	**x**	**x**	✓
Access						
- Command-Line	**x**	✓	✓	**x**	✓	**x**
- Web-based Dashboard	✓	✓	✓	✓	**x**	✓
- Web-based API	✓	✓	✓	✓	✓	✓

x = not fulfilled, **0** = partially fulfilled, ✓ = fully fulfilled

components, and a group of *instances* of these components. Component definitions can be reused via abstract types (*facets*), imports, and inheritance.

Roboconf consists of several modules. The *Deployment Manager (DM)* is in charge of instantiating and managing VMs and remote agents. *Agents* use plugins (such as Bash or Puppet) in order to handle the life-cycle of software instances. The DM and the agents communicate with each other through an asynchronous messaging server. The *SoftwareInstanceManager* is responsible for automatically generating software life-cycle management and monitor software instances themselves.

Automatic scaling capabilities are provided by means of *autonomic management* implemented by the DM and the remote agents. Agents send notifications to the DM whenever certain threshold-based conditions linked to system metrics are met. The DM's decision engine responds to those notifications using corresponding imperative rules. Monitoring application metrics still needs to be addressed. Both application migrations and global/per-component rollbacks (continuous deployment) are part of Roboconf's roadmap, but they are not supported out of the box yet.

INDIGO-DataCloud

INDIGO-DataCloud (INtegrating Distributed data Infrastructures for Global ExplOitation) [55, 56, 64] is an open-source data and computing platform targeted at scientific communities, and provisioned over Cloud and Grid-based infrastructures as well as over HTC and HPC clusters. The INDIGO-DataCloud framework has been developed within the homonymous project funded under the EU's Horizon 2020 Framework Programme [65].

The INDIGO-DataCloud project extended existing PaaS solutions in order to provide automatic distribution of applications and/or services over a hybrid and heterogeneous set of IaaS infrastructures. Some of the key INDIGO PaaS components include: *Orchestrator*, *Infrastructure Manager (IM)*, *CloudProviderRanker*, *Monitoring*, *SLA Manager (SLAM)*, *Managed Services/Application (MSA) Deployment*, and *Data Management Services*. The Orchestrator coordinates the process of deploying services and applications on both on-premise and public IaaS platforms. It can be accessed via a command-line interface (*Orchent*), or a GUI-based portlet, or a REST API.

The Orchestrator delegates the deployment to the IM, to OpenStack Heat or to the Mesos frameworks, based on TOSCA templates and a list of providers ranked by the CloudProviderRanker. The Monitoring component collects monitoring data from both PaaS core services and client infrastructure/services by means of specific probes. The SLAM establishes an agreement between customer and provider about capacity and quality targets. The Data Management Services provide an abstraction layer for accessing the data storage in a unified and federated way.

INDIGO-DataCloud supports multi-cloud and cross-cloud deployments, as well as interoperability by leveraging open standards (OCCI, CDMI). It also promotes portability by adopting an extension of TOSCA for describing applications and services. Cloud resources are automatically selected and optimised by the CloudProviderRanker, depending on SLAs and

Table 5 Application-based comparison of experimental/research CROFs

	CROFs						
	Application Features	**Cloudiator**	**Roboconf**	**INDIGO-DataCloud**	**MiCADO**	**MODAClouds**	**SeaClouds**
	Portability						
	- Model Standards	**x**	**x**	**0**	**0**	**x**	**0**
	- Model Reusability	✓	✓	✓	✓	**0**	✓
	- Model Sharing	✓	✓	✓	**0**	✓	✓
Development	**Containerisation**	✓	✓	✓	✓	**x**	**x**
	Resource Selection						
	- Manual Binding	**x**	✓	**x**	✓	**x**	**x**
	- Automatic Binding	✓	**x**	✓	**x**	✓	✓
	- Optimised Binding	✓	**x**	✓	**x**	✓	✓
	- Dynamic Binding	**x**	**x**	**x**	**x**	**x**	✓
	Lifecycle Control						
	- Script-based	✓	✓	**x**	**x**	✓	**x**
	- DevOps	**x**	✓	✓	**x**	✓	**x**
	Wiring & Workflow						
	- Attribute & Event Passing	✓	✓	✓	✓	✓	✓
	- Manual Workflow	**x**	**x**	**x**	**x**	**x**	**x**
	- Automatic Workflow	✓	✓	✓	✓	✓	✓
Deployment	- External Services	**x**	✓	**x**	**x**	✓	✓
	Monitoring						
	- System Metrics	✓	✓	✓	✓	✓	✓
	- Application Metrics	✓	**x**	**x**	**x**	✓	✓
	- Custom Metrics	✓	**x**	**x**	**x**	**x**	**x**
	- Aggregation	✓	**x**	✓	✓	✓	✓
	- Historical Data	**x**	**x**	✓	✓	✓	✓
	Runtime Adaptation						
	- Threshold-based	✓	✓	✓	✓	✓	✓
	- Rule Engine	✓	✓	✓	✓	✓	✓
	- Horizontal Scaling	✓	✓	✓	✓	✓	✓
	- Migration	**0**	**x**	**0**	**x**	✓	**0**
	- Failure Recovery	✓	**x**	✓	✓	**0**	✓
Execution	- CI/CD	✓	**x**	**x**	✓	✓	**x**

x = not fulfilled, **0** = partially fulfilled, ✓ = fully fulfilled

monitoring data. A configuration management solution based on Ansible roles is adopted to carry out both the deployment of the application and the creation of the pre-configured Docker images. Only automatic workflows are supported.

Runtime actions, such as horizontal scaling and failure handling, are automatically supported by the MSA Deployment (based on Apache Mesos), which uses the *Automatic Scaling Service* (based on EC3/CLUES[35]) to ensure the elasticity of the cluster, Marathon[36] and Chronos[37] frameworks in order to handle Long-Running Services (LRS) and application jobs, respectively. Marathon can also migrate services if problems occur. Despite different DevOps practises being adopted for both the core services and user applications (e.g., automated builds of each application image are triggered once a new change is committed to its repository), hot changes in application deployments are not supported out of the box.

[35] https://www.grycap.upv.es/clues/eng/index.php

[36] https://mesosphere.github.io/marathon/

[37] https://mesos.github.io/chronos/

MiCADO

MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) [57] is an open-source multi-cloud orchestration and auto-scaling framework for Docker containers, orchestrated by Kubernetes (or alternatively by Docker Swarm). The full MiCADO framework has been investigated and implemented in the COLA (Cloud Orchestration at the Level of Application) project funded by the European Commission [66].

MiCADO core services are deployed on *MiCADO Master*, which is configured as the Kubernetes Master Node and provides the *Docker Engine*, *Occopus* [67] (to scale VMs), *Prometheus*[38] (for monitoring), *Policy Keeper* (to perform decision on scaling), and *Submitter* (to provide submission endpoint) microservices. During operation, *MiCADO workers* are instantiated on demand and join the cluster managed by the MiCADO Master.

MiCADO supports multi-cloud and cross-cloud deployments on various public and private cloud infrastructures. It also provides interoperability and portability by means of a TOSCA-based *Application Description Template (ADT)*, which comprises three sections: a) the definition of the individual applications making up a *Kubernetes Deployment*, b) the specification of the VM and c) the implementation of scaling policies for both VM and Kubernetes scaling levels. ADTs can be consumed by means of a web-based dashboard or a REST API.

Cloud resources are manually selected when configuring VMs. The application life-cycle is handled by MiCADO itself, which leverages Occopus and Kubernetes for managing VMs and containers, respectively. Only automatic workflows are supported. MiCADO allows automated scaling depending on VM and container metrics gathered by two built-in exporters on each MiCADO worker: *Prometheus Node Exporter*[39] and *CAdvisor*[40]. Scaling policies can be defined specifically for the applications. Lastly, continuous delivery capabilities are supported via "rolling updates" on Kubernetes Deployments.

MODAClouds

MODAClouds (MOdel-Driven Approach for the design and execution of applications on multiple Clouds) [58, 59] is an open-source design-time and run-time platform for developing and operating multi-cloud applications with guaranteed QoS. The MODAClouds framework has been developed within the homonymous project funded by the European Commission [68].

The MODAClouds Toolbox consists of three main components: *Creator4Clouds*, *Venues4Clouds*, and *Energizer4Clouds*. Creator4Clouds is a design-time platform which allows to design multi-cloud applications, carry out performance and cost evaluation, and plan the deployment strategy by choosing the service providers that best suit all business and QoS requirements. Venue4Clouds is a decision support system (DSS) to choose the most suitable cloud providers depending on different aspects such as application architecture, business risk, quality and cost. Energizer4Clouds is a run-time platform to deploy, manage, monitor and assure operations of multi-cloud services. Specifically, *Tower4Clouds* sub-component is responsible for collecting, analysing, and storing monitoring information, whereas *SpaceOps4Clouds* sub-component enacts application self-adaptation in order to meet predefined objectives and/or constraints whenever changes happen.

MODAClouds supports multi-cloud and cross-cloud deployments on both IaaS and PaaS providers. It leverages an MDE approach in order to support interoperability between cloud providers. In particular, *MODACloudML* is a set of UML extensions enabling developers to model multi-cloud applications through three level of abstractions: Cloud-enabled Computation Independent Models (CCIM), Cloud-Provider Independent Models (CPIM), and Cloud-Provider Specific Models (CPSM). These models facilitate portability, since they are mostly reusable. Cloud resources can be automatically selected and optimised via *Venues4Clouds* and *SpaceDev4Clouds*, and managed through either shell scripts or Puppet. Only automatic workflows are supported.

Within the MODAClouds runtime environment, the *Models@Runtime engine* is responsible for enacting adaptation actions such as application scaling and bursting, data and application migration, and continuous delivery on both infrastructure and component levels. Failure recovery is partially supported for data migration and scaling/bursting scenarios.

SeaClouds

SeaClouds (SEamless Adaptive multi-Cloud management of service-based applicationS) [60, 61] is an open-source platform for deploying and managing multi-component applications over heterogeneous clouds. The SeaClouds framework has been investigated and implemented within the homonymous project funded by the European Commission [69].

The SeaClouds architecture comprises six main components: *Dashboard*, *Discoverer*, *Planner*, *Deployer*, *Monitor*, and *SLA Service*. The Dashboard allows to model applications (topology and requirements). The Discoverer identifies the available capabilities offered by cloud providers. The Planner receives the *AAM (Abstract Application Model)* from the Dashboard and creates a set of *ADP (Abstract Deployment Plan)* meeting the application requirements. From the selected plan a *Deployable Application Model (DAM)* is to be generated, containing the

[38] https://prometheus.io/
[39] https://github.com/prometheus/node_exporter
[40] https://github.com/google/cadvisor

information needed by the Deployer (based on Apache Brooklyn [47]) to deploy, configure and run the application. The Monitor collects infrastructure and application level metrics from the targeted cloud providers in order to verify that QoS requirements are met. And if not, reconfiguration actions can be triggered. The SLA Service enforces business-oriented policies and business actions to apply in case of violation.

SeaClouds supports multi-cloud and cross-cloud deployments on both IaaS and PaaS providers. It also promotes interoperability and portability by adopting a TOSCA-based representation for AAMs and ADPs, as well as a CAMP-based description for DAMs. Cloud resources are automatically selected and optimised by the Planner. Changes to the binding can also occur in case of reconfiguration actions. Only automatic workflows are supported.

SeaClouds allows repairing actions, such as scaling horizontally and vertically cloud resources, or restarting and replacing failed components. It also supports replanning in order to handle the cases that cannot be solved by repairing. A migration of application modules may happen in this process. Continuous delivery is not supported out of the box.

Other initiatives

In this section, we briefly review a number of other research approaches derived from related EU projects which address, to varying degrees, multi-cloud orchestration, interoperability and portability. Specifically, a few works target semantic interoperability (i.e., moSAIC, cloud4SOA), some explore the benefits of federated cloud networks (BEACON, ATMOSPHERE), whereas others focus on application portability via non-standard (i.e, Claudia, OPTIMIS, ASCETiC, HARNESS), partially-standard (i.e, soCloud) and fully-standard (i.e., CELAR, CloudLightning) cloud modelling.

mOSAIC [70, 71] is an open-source API and platform for multiple clouds designed and developed within the homonymous project [72]. Application deployment and portability across multiple clouds are facilitated by means of a common API and a high-level abstraction of cloud resources. mOSAIC also enables application developers to specify resource requirements in terms of a cloud ontology, whereas the platform, using a brokering mechanism, performs a matchmaking process in order to find the best-fitting cloud services. In so doing, developers can postpone their decision on the procurement of cloud services until runtime. However, even though a platform-independent component-based programming model is used, applications need to be implemented by leveraging one of the supported language-dependent APIs (Java, Python).

Cloud4SOA [73, 74] is a multi-cloud broker-based solution developed under the homonymous project [75], which addresses semantic interoperability and portability challenges at the PaaS layer. It supports multi-platform matchmaking, management, monitoring and migration of applications by semantically interconnecting heterogeneous PaaS offerings. Similar to mOSAIC, Cloud4SOA introduces a cloud ontology establishing a set of abstractions among different PaaS offerings while exposing a multi-PaaS standardised API for the seamless application deployment and management across different cloud platforms. Despite being independent of specific APIs offered by the underlying PaaS offerings, adapters acting as a middleware between the Cloud4SOA API and native PaaS APIs are still needed.

The main goal of the BEACON project [76] is to develop techniques to federate cloud network resources, and to enable an efficient and secure deployment of federated cloud applications. Specifically, the proposed approach is to build a homogeneous virtualisation layer on top of heterogeneous underlying physical networks, computing and storage infrastructures. By leveraging the combination of Cloud federation, Software Defined Networking (SDN), and Network Function Virtualization (NFV) technologies, the project has delivered an innovative design of a Federation Management system acting as an external service provider dealing with federated networking services among multiple federated OpenStack Clouds [77].

ATMOSPHERE [78] aims to design and implement a framework and platform relying on lightweight virtualisation, hybrid resources and Europe and Brazil federated infrastructures to develop, build, deploy, measure and evolve trustworthy, cloud-enabled applications. Orchestration and deployment of complex application topologies is achieved through the TOSCA standard. In the context of the project, partners developed a federated network architecture [79] by creating multi-tenant overlay networks across different sites. The developed framework offers services such as distribution and inter-site migration of VMs, resource management, and network management.

Claudia [80] is a service management system implementing an abstraction layer that allows for the automatic service deployment and scaling depending on both infrastructure and service status. Conversely to mOSAIC and Cloud4SOA, each service in Claudia is defined by its corresponding Service Description File (SDF) whose syntax is based on the OVF standard, thereby providing vendor and platform portability. However, special OVF extensions must be defined in order to support automatic scalability, deployment-time customisation and external connectivity specification.

OPTIMIS [81] is a toolkit which addresses and optimises the whole service lifecycle on the basis of aspects

such as trust, risk, eco-efficiency and cost, taking into consideration a number of cloud scenarios, namely, cloud federation, multi-cloud, hybrid cloud. However, regarding multi-cloud, interoperability with non-OPTIMIS providers can only be achieved by using APIs and adapters externally to the OPTIMIS components. According to OPTIMIS programming model, each service is defined as a collection of core elements being packed along with any external software components into VM images. Similar to Claudia, these VM images are configured by means of a service manifest based on the OVF standard, but a set of OVF extensions are required in order to specify the functional and non-functional requirements of the service.

ASCETiC [82] is an open architecture and approach to multi-cloud optimising energy efficiency, designed within the homonymous EU project [83]. Analogous to OPTIMIS, the OVF specification is employed to define a complete set of VMs to be deployed at an IaaS provider. Nevertheless, OVF extensions are necessary in order to support SLA negotiation and self-adaptation rules.

The HARNESS project [84] develops a cloud computing platform incorporating non-traditional and heterogeneous computational, networking and storage resources into the data centre stack to provide high performance at low cost. HARNESS envisions an enhanced cloud PaaS software stack that not only supports existing commodity technologies, but also incorporates heterogeneous technologies such as Dataflow Engines (DFEs), programmable routers and different types of storage devices [85]. The project demonstrated its results via extensions to OpenStack.

soCloud [86] is a service-oriented component-based PaaS for managing portability, elasticity, provisioning, and high availability across multiple clouds. Application descriptors are based on the OASIS Service Component Architecture (SCA) standard [87]. However, since the SCA model doesn't allow to define non-functional requirements, special SCA extensions are required. A custom DSL is also used in order to describe elasticity. Additionaly, not only does soCloud support only SCA-based applications, but maintaining the mappings to various cloud providers and keeping up with recent features of supported clouds are a concern.

CELAR [88, 89] is a resource management platform able to automatically deploy, monitor and scale applications over a cloud infrastructure. Applications are described using TOSCA, which ensures the portability of application descriptions across different IaaS platforms. However, every time a new application is to be deployed, users need to issue the request to the appropriate *CELAR Server* instance inside the cloud they want to deploy their application to. In contrast to mOSAIC, cloud4SOA and ASCETiC, no brokering mechanism is defined in order to best fit cloud resource requirements. Furthermore, cross-cloud is not supported.

CloudLightning [90] is a heterogeneous cloud service management and delivery model developed within the homonymous EU project [91]. Based on the principles of self-organisation and self-management, CloudLightning allows users to design and deploy their applications without the need for selecting the most suitable resources. This separation of concerns is made possible using a CloudLightning-specific service description language (CL-SDL), which extends TOSCA in order to capture specific attributes . The declarative approach is enriched with resource discovery mechanisms allowing easier identification and consumption of a variety of heterogeneous resources. CloudLightning proposes a solution based on a *Gateway Service*, which relies on two open-source tools: *Alien4Cloud* acting as the Gateway Service UI and *Brooklyn-TOSCA*[41] acting as the deployment orchestrator. In view of the above, the same remarks made in "Brooklyn" section are applicable to CloudLightning.

Critical discussion

Tables 2, 3, 4 and 5 summarise the CROFs presented in "Review of cROFs" section by outlining the features debated in "Analysis framework" section. We discuss the main characteristics of these frameworks next.

Most of the reviewed CROFs provide different access modes, including web-based dashboards and APIs, and allow both multi-cloud and cross-cloud deployments, except for Heat and CloudFormation which, as infrastructure-centric services, only support their own IaaS providers (i.e., OpenStack and Amazon, respectively). Besides, some of them natively support deployments on BYON (e.g., Cloudify, Brooklyn, Roboconf, SeaClouds). Interoperability between cloud providers is mainly achieved by means of open standards and open libraries/abstraction layers (e.g. jclouds). Open standards appear to be gaining ground, especially in academic scenarios. As such, a number of academic CROFs provide interoperability via OCCI (e.g., Roboconf) or CDMI (e.g., INDIGO-DataCloud) support, while others do via TOSCA (MiCADO, SeaClouds). Despite being the focus of previous research efforts (e.g., mOSAIC, Cloud4SOA), semantic approaches seem to be no longer a priority compared to the adoption of open standards. Of all the initiatives, MODAClouds is the only one to employ model-driven methodologies.

With regard to application portability, CROFs from both industry and academia are placing ever-increasing importance on modelling standards. However, while taking TOSCA (e.g., Cloudify, MiCADO) and CAMP (e.g.,

[41] https://github.com/cloudsoft/brooklyn-tosca

Brooklyn, SeaClouds) as reference models, they happen to customise and extend standard types. Thus, a further effort would be appropriate in order to ensure greater compliance with the aforementioned specifications. Aside from the adoption of standard models, model reusability is encouraged by means of modules shared either locally or remotely. Besides, since containers provide improved application encapsulation and abstraction from resources, most of the CROFs support containers as well as container orchestration.

As regards resource provisioning, there are different aspects of the matter that need to be considered, such as: selection, configuration and deployment of resources. In multi-cloud scenarios, selection is far from being a trivial task due to the diversity of cloud services' characteristics and QoS. While manual selection is supported in the majority of CROFs, automatic and optimised selections are almost exclusively supported by academic CROFs. The optimised selection leverages QoS and technical requirements, and is carried out either based on static information on the service quality provided by cloud providers or through dynamic negotiation of SLAs. A few multi-cloud projects (e.g., INDIGO-DataCloud, MODAClouds, SeaClouds) provide support for SLA management, even though multi-cloud SLAs are not covered. Limited support is currently available for dynamic selection (i.e., SeaClouds).

Resource deployment can be manual or automatic. While most commercial CROFs support both manual and automatic workflows, academic CROFs exclusively support automatic ones. Using standard models such as TOSCA, where applicable, proves useful both for defining a custom workflow and for automatically generating one. However, since current standards lack support for modelling the semantics related to the instantiation of relationships between component instances, the actual wiring of component instances depends on the capabilities offered by the CROF enacting the deployment. On that note, standard extensions in support of sophisticated wiring on instance level would be desirable. As for resource configuration, on the one hand scripts are extensively supported, but on the other hand configuration management tools are mostly supported by commercial CROFs. Nonetheless, a few academic projects (e.g., INDIGO-DataCloud and MODAClouds) exploit these tools in order to enact DevOps practices as well.

Monitoring plays a key role in keeping track of the status of applications as well as physical and virtual resources. Monitoring metrics at different abstraction levels (e.g., infrastructure and application ones) and capturing dependencies between these levels allow to perform root cause analysis, such that any issues at infrastructure level can automatically lead to run-time infrastructure adaptation which best fits run-time application requirements. While infrastructure metrics are widely supported by both commercial and academic CROFs, application and custom metrics necessitate further investigation. Metric aggregation mechanisms are available for a large majority of CROFs. Nevertheless, in light of multi-cloud scenarios, where applications and resources may be largely distributed, metric collection and aggregation from heterogeneous cloud environments are necessary. As a result, standardised interfaces and formats should be inspected.

Monitoring data allows for different purposes such as enforcing SLAs, enabling elasticity, ensuring QoS. SLAs can be used as a basis for cloud services and respective applications to be managed during their lifecycle. Multi-cloud management requires specific mechanisms for run-time adaptation across a diversity of cloud set-ups, including scalability, migration, fault-tolerance, continuous delivery. While reactive approaches to run-time adaptation are fairly consolidated among all CROFs, predictive approaches (based on workload prediction models and machine learning optimisation) are only supported in some commercial CROFs (e.g., AWS CloudFormation) and should be more explored.

Both academic and commercial CROFs largely provide support for threshold-based horizontal scaling. Policy-based approaches, especially in the academic landscape, are gaining in importance as well. Migration support is still limited in both industry and academia, as it is closely linked to portability in all its facets, i.e., VM portability, application portability, data portability. Although platform-independent standards (TOSCA) and virtualisation techniques (containers) have improved application encapsulation and abstraction from resources, platform-independent data representation and standardisation of data import and export across diverse and heterogeneous clouds need to be inspected. In this regard, MODAClouds provides a solution to the data migration issue, albeit in the context of scalable NoSQL databases.

Both academic and commercial CROFs support failure recovery mechanisms based on restarting/replacing failed components or, in a worst-case scenario, rolling back entire application stacks. Of all academic CROFs, Cloudiator, MODAClouds and SeaClouds allow to identify abnormal and undesirable states of the system and apply a limited set of autonomic actions. However, the emergence of decentralised multi-cloud setups connecting a wider variety of entities and resources requires autonomic management systems that consider self-organisation, self-management and self-healing across a diversity of cloud deployments. Continuous delivery is well supported in the commercial landscape, and it is also gaining ground in the academic one because of the ever-growing use of DevOps methodologies.

Conclusion

Cloud computing technology has greatly evolved over the past few years, transforming the traditional infrastructure, platform and software resources into elastic and on-demand virtual components. However, heterogeneous and multi-layer resources have to be orchestrated in an effective way in order to ensure that end-users are provided with acceptable quality levels.

In this work we thoroughly analysed the cloud orchestration landscape: after presenting a taxonomy of relevant features and dimensions, we mapped and evaluated several cloud resource orchestration frameworks against it, especially focusing on multi-cloud capabilities. This systematic analysis has allowed to identify key open research issues, also proposing a set of future research directions in the cloud orchestration scenario.

Abbreviations

CROF: Cloud Resource Orchestration Framework; MDE: Model Driven Engineering; IaaS: Infrastructure as a Service; PaaS: Platform as a Service; CMP: Cloud Management Platform; SLA: Service Level Agreement; TOSCA: Topology and Orchestration Specification for Cloud Applications; VM: Virtual Machine; BYON: Bring Your Own Node; CI/CD: Continuous Integration/Continuous Delivery

Acknowledgements

Not applicable.

Authors' contributions

All authors contributed equally to the article. The author(s) read and approved the final manuscript.

Funding

Not applicable.

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 14 April 2020 Accepted: 4 August 2020
Published online: 10 September 2020

References

1. RightScale (2019) RightScale 2019 State of the Cloud Report. https://info.flexera.com/SLO-CM-WP-State-of-the-Cloud-2019. Accessed 12 Oct 2019
2. Ranjan R, Benatallah B, Dustdar S, Papazoglou MP (2015) Cloud resource orchestration programming: Overview, issues, and directions. IEEE Internet Comput 19(5):46–56. https://doi.org/10.1109/MIC.2015.20
3. Weerasiri D, Barukh MC, Benatallah B, Sheng QZ, Ranjan R (2017) A Taxonomy and Survey of Cloud Resource Orchestration Techniques. ACM Comput Surv 50(2):26–12641. https://doi.org/10.1145/3054177
4. Baur D, Seybold D, Griesinger F, Tsitsipas A, Hauser CB, Domaschka J (2015) Cloud Orchestration Features: Are Tools Fit for Purpose?,. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC). pp 95–101. https://doi.org/10.1109/UCC.2015.25
5. Bousselmi K, Brahmi Z, Gammoudi MM (2014) Cloud services orchestration: A comparative study of existing approaches. In: 28th International Conference on Advanced Information Networking and Applications Workshops. pp 410–416. https://doi.org/10.1109/WAINA.2014.72
6. Grozev N, Buyya R (2014) Inter-cloud architectures and application brokering: taxonomy and survey. Softw Pract Experience 44(3):369–390. https://doi.org/10.1002/spe.2168
7. Ferry N, Rossini A, Chauvel F, Morin B, Solberg A (2013) Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In: IEEE Sixth International Conference on Cloud Computing. pp 887–894. https://doi.org/10.1109/CLOUD.2013.133
8. Petcu D (2014) Consuming resources and services from multiple clouds. J Grid Comput 12(2):321–345. https://doi.org/10.1007/s10723-013-9290-3
9. Petcu D, Vasilakos A (2014) Portability in Clouds: Approaches and Research Opportunities. Scalable Comput Pract Experience 15(3):251–270. https://doi.org/10.12694/scpe.v15i3.1019
10. Ferry N, Rossini A (2018) CloudMF: Model-Driven Management of Multi-Cloud Applications. ACM Trans Internet Technol 18(2):16–24. https://doi.org/10.1145/3125621
11. Ferrer AJ (2016) Inter-cloud research: Vision for 2020. Procedia Comput Sci 97:140–143. https://doi.org/10.1016/j.procs.2016.08.292
12. Buyya R, Srirama SN, Casale G, Calheiros R, Simmhan Y, Varghese B, Gelenbe E, Javadi B, Vaquero LM, Netto MAS, Toosi AN, Rodriguez MA, Llorente IM, Vimercati SDCD, Samarati P, Milojicic D, Varela C, Bahsoon R, Assuncao MDD, Rana O, Zhou W, Jin H, Gentzsch W, Zomaya AY, Shen H (2018) A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. ACM Comput Surv 51(5):105–110538. https://doi.org/10.1145/3241737
13. Lewis GA (2013) Role of standards in cloud-computing interoperability. In: 46th Hawaii International Conference on System Sciences. pp 1652–1661. https://doi.org/10.1109/HICSS.2013.470
14. Badger L, Bohn R, Chandramouli R, Grance T, Karygiannis T, Patt-Corner R, Voas E (2010) Cloud Computing Use Cases. https://www.nist.gov/itl/use-cases. Accessed 12 Oct 2019
15. Ahronovitz M, et al. (2010) Cloud Computing Use Cases White Paper Version 4.0. http://www.cloud-council.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf. Accessed 12 Oct 2019
16. Distributed Management Task Force (2010) Use Cases and Interactions for Managing Clouds. https://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf. Accessed 12 Oct 2019
17. Zhang Z, Wu C, Cheung DWL (2013) A survey on cloud interoperability: Taxonomies, standards, and practice. SIGMETRICS Perform Eval Rev 40(4):13–22. https://doi.org/10.1145/2479942.2479945
18. Stravoskoufos K, Preventis A, Sotiriadis S, Petrakis EGM (2014) A Survey on Approaches for Interoperability and Portability of Cloud Computing Services. In: Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER2014). pp 112–117. https://doi.org/10.5220/0004856401120117
19. García ÁL, del Castillo EF, Fernández PO (2016) Standards for enabling heterogeneous IaaS cloud federations. Comput Stand Interfaces 47:19–23. https://doi.org/10.1016/j.csi.2016.02.002
20. Gartner (2018) Competitive Landscape: Cloud Service Brokerage. https://www.gartner.com/en/documents/3889023/competitive-landscape-cloud-service-brokerage. Accessed 12 Oct 2019
21. Liu F, Tong J, Mao J, Bohn RB, Messina JV, Badger ML, Leaf DM (2011) NIST Cloud Computing Reference Architecture. https://www.nist.gov/publications/nist-cloud-computing-reference-architecture. Accessed 12 Oct 2019
22. Elhabbash A, Samreen F, Hadley J, Elkhatib Y (2019) Cloud brokerage: A systematic survey. ACM Comput Surv 51(6):119–111928. https://doi.org/10.1145/3274657
23. Bernstein D, Ludvigson E, Sankar K, Diamond S, Morrow M (2009) Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. In: Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services. ICIW '09. pp 328–336. https://doi.org/10.1109/ICIW.2009.55
24. Global Inter-cloud Technology Forum (2010) Use Cases and Functional Requirements for Inter-Cloud Computing: A white paper. http://www.gictf.jp/doc/GICTF_Whitepaper_20100809.pdf. Accessed 12 Oct 2019
25. Petcu D (2013) Multi-cloud: Expectations and current approaches. In: Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds. MultiCloud '13. ACM, New York. pp 1–6. https://doi.org/10.1145/2462326.2462328
26. Toosi AN, Calheiros RN, Buyya R (2014) Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Comput Surv 47(1):7–1747. https://doi.org/10.1145/2593512
27. Nogueira E, Moreira A, Lucrédio D, Garcia V, Fortes R (2016) Issues on developing interoperable cloud applications: definitions, concepts,

approaches, requirements, characteristics and evaluation models. J Softw Eng Res Dev 4(1):7. https://doi.org/10.1186/s40411-016-0033-6
28. Kaur K, Sharma DS, Kahlon DKS (2017) Interoperability and portability approaches in inter-connected clouds: A review. ACM Comput Surv 50(4):49–14940. https://doi.org/10.1145/3092698
29. Bellendorf J, Mann ZÁ (2018) Cloud Topology and Orchestration Using TOSCA: A Systematic Literature Review. In: Kritikos K, Plebani P, de Paoli F (eds). Service-Oriented and Cloud Computing. pp 207–215. https://doi.org/10.1007/978-3-319-99819-0_16
30. Domaschka J, Griesinger F, Baur D, Rossini A (2015) Beyond Mere Application Structure Thoughts on the Future of Cloud Orchestration Tools. Procedia Comput Sci 68:151–162. https://doi.org/10.1016/j.procs.2015.09.231
31. Khoshkbarforoushha A, Wang M, Ranjan R, Wang L, Alem L, Khan SU, Benatallah B (2016) Dimensions for evaluating cloud resource orchestration frameworks. Computer 49(2):24–33. https://doi.org/10.1109/MC.2016.56
32. Clusters of European Projects on Cloud (2015) Inter-cloud Challenges, Expectations and Issues Cluster Position Paper: Initial Research Roadmap and Project's Classification. https://eucloudclusters.wordpress.com/future-cloud. Accessed 12 Oct 2019
33. Clusters of European Projects on Cloud (2016) Inter-cloud Challenges, Expectations and Issues Cluster Position Paper: Research Roadmap Update. https://eucloudclusters.wordpress.com/future-cloud. Accessed 12 Oct 2019
34. Clusters of European Projects on Cloud (2017) Future Cloud Cluster Vision for 2030. https://eucloudclusters.wordpress.com/future-cloud. Accessed 12 Oct 2019
35. GigaSpaces Research CloudifyTeam (2016) Cloud Management in the Enterprise - An Overview of Orchestration vs. PaaS vs. CMP. https://cloudify.co/blog/cloud-management-roundup-orchestration-paas-cmp/. Accessed 12 Oct 2019
36. OASIS (2013) Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Accessed 6 July 2020
37. Buyya R, Vecchiola C, Selvi ST (2013) Chapter 10 - Cloud Applications. In: Buyya R, Vecchiola C, Selvi ST (eds). Mastering Cloud Computing. Morgan Kaufmann, Boston. pp 353–371. https://doi.org/10.1016/B978-0-12-411454-8.00010-3
38. Costache S, Dib D, Parlavantzas N, Morin C (2017) Resource management in cloud platform as a service systems: Analysis and opportunities. J Syst Softw 132:98–118. https://doi.org/10.1016/j.jss.2017.05.035
39. Kolb S, Wirtz G (2014) Towards Application Portability in Platform as a Service. In: IEEE 8th International Symposium on Service Oriented System Engineering. pp 218–229. https://doi.org/10.1109/SOSE.2014.26
40. Oberle K, Fisher M (2010) ETSI CLOUD - Initial Standardization Requirements for Cloud Services. In: Proceedings of the 7th International Conference on Economics of Grids, Clouds, Systems, and Services. GECON'10. Springer, Berlin, Heidelberg. pp 105–115. https://doi.org/10.1007/978-3-642-15681-6_8
41. OASIS (2014) Cloud Application Management for Platforms Version 1.1. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html. Accessed 12 Oct 2019
42. Soltesz S, Pötzl H, Fiuczynski ME, Bavier A, Peterson L (2007) Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In: Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. EuroSys '07. ACM, New York. pp 275–287. https://doi.org/10.1145/1272996.1273025
43. Singh S, Singh N (2016) Containers & Docker: Emerging roles & future of Cloud technology. In: 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). pp 804–807. https://doi.org/10.1109/ICATCCT.2016.7912109
44. Komarek A, Pavlik J, Sobeslav V (2017) Hybrid System Orchestration with TOSCA and Salt. J Eng Appl Sci 12(9):2396–2401. https://doi.org/10.36478/jeasci.2017.2396.2401
45. OpenStack (2016) OpenStack Heat. https://wiki.openstack.org/wiki/Heat. Accessed 12 Oct 2019
46. Cloudify (2019) Cloudify. http://cloudify.co/. Accessed 12 Oct 2019
47. The Apache Software Foundation (2016) The Apache Brooklyn project. https://brooklyn.apache.org/. Accessed 12 Oct 2019
48. Apache (2015) Apache Stratos. https://stratos.apache.org/. Accessed 12 Oct 2019
49. FastConnect (2018) Alien4Cloud. https://alien4cloud.github.io. Accessed 12 Oct 2019
50. HashiCorp (2019) HashiCorp Terraform. https://www.terraform.io/. Accessed 12 Oct 2019
51. Amazon (2016) Amazon CloudFormation. https://aws.amazon.com/cloud\discretionary-for\discretionary-ma\discretionary-tion/. Accessed 12 Oct 2019
52. Baur D, Domaschka J (2016) Experiences from building a cross-cloud orchestration tool. In: Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms. CrossCloud '16. pp 4–146. https://doi.org/10.1145/2904111.2904116
53. Baur D, Seybold D, Griesinger F, Masata H, Domaschka J (2018) A Provider-Agnostic Approach to Multi-cloud Orchestration Using a Constraint Language. In: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp 173–182. https://doi.org/10.1109/CCGRID.2018.00032
54. Pham LM, Tchana A, Donsez D, de Palma N, Zurczak V, Gibello P (2015) Roboconf: A Hybrid Cloud Orchestrator to Deploy Complex Applications. In: IEEE 8th International Conference on Cloud Computing. pp 365–372. https://doi.org/10.1109/CLOUD.2015.56
55. Salomoni D, Campos I, Gaido L, et al. (2016) Indigo-datacloud: foundations and architectural description of a platform as a service oriented to scientific computing. CoRR abs/1603.09536. 1603.09536. Accessed 6 July 2020
56. Salomoni D, Campos I, Gaido L, et al. (2018) INDIGO-DataCloud: a Platform to Facilitate Seamless Access to E-Infrastructures. J Grid Comput 16(3):381–408. https://doi.org/10.1007/s10723-018-9453-3
57. Kiss T, Kacsuk P, Kovacs J, Rakoczi B, Hajnal A, Farkas A, Gesmier G, Terstyanszky G (2019) MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator. Futur Gener Comput Syst 94:937–946. https://doi.org/10.1016/j.future.2017.09.050
58. Ardagna D, Di Nitto E, Mohagheghi P, Mosser S, Ballagny C, D'Andria F, Casale G, Matthews P, Nechifor C, Petcu D, Gericke A, Sheridan C (2012) MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds. In: 4th International Workshop on Modeling in Software Engineering (MISE). pp 50–56. https://doi.org/10.1109/MISE.2012.6226014
59. Nitto ED, Matthews P, Petcu D, Solberg A (2017) Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach. https://doi.org/10.1007/978-3-319-46031-4
60. Brogi A, Carrasco J, Cubo J, D'Andria F, Ibrahim A, Pimentel E, Soldani J (2014) SeaClouds: Seamless adaptive multi-cloud management of service-based applications. In: 17th Conferencia Iberoamericana en Software Engineering (CIbSE 2014). Curran Associates, Inc., Pucon. pp 95–108
61. Brogi A, Fazzolari M, Ibrahim A, Soldani J, Wang P, Carrasco J, Cubo J, Durán F, Pimentel E, Di Nitto E, D'Andria F (2015) Adaptive management of applications across multiple clouds: The SeaClouds Approach. CLEI Electron J 18:2–2. https://doi.org/10.19153/cleiej.18.1.1
62. University of Ulm (2015) Cloudiator. http://cloudiator.org/. Accessed 12 Oct 2019
63. Linagora (2013) Roboconf. http://roboconf.net. Accessed 12 Oct 2019
64. Caballer M, Zala S, García ÁL, Moltó G, Fernández PO, Velten M (2018) Orchestrating Complex Application Architectures in Heterogeneous Clouds. J Grid Comput 16(1):3–18. https://doi.org/10.1007/s10723-017-9418-y
65. INDIGO consortium (2017) The INDIGO-DataCloud project. https://www.indigo-datacloud.eu/. Accessed 12 Oct 2019
66. COLA consortium (2017) The COLA Project. https://project-cola.eu/. Accessed 12 Oct 2019
67. Kovács J, Kacsuk P (2018) Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. J Grid Comput 16(1):19–37. https://doi.org/10.1007/s10723-017-9421-3
68. MODAClouds consortium (2012) The MODAClouds project. http://multiclouddevops.com/. Accessed 12 Oct 2019
69. SeaClouds consortium (2013) The SeaClouds project. http://www.seaclouds-project.eu/. Accessed 12 Oct 2019
70. Petcu D, Macariu G, Panica S, Crăciun C (2013) Portable Cloud applications—From theory to practice. Futur Gener Comput Syst 29(6):1417–1430. https://doi.org/10.1016/j.future.2012.01.009

71. Petcu D, Martino BD, Venticinque S, Rak M, Máhr T, Lopez GE, Brito F, Cossu R, Stopar M, Šperka S, Stankovski V (2013) Experiences in building a mOSAIC of clouds. J Cloud Comput Adv Syst Appl 2(1):12. https://doi.org/10.1186/2192-113X-2-12
72. mOSAIC consortium (2010) The mOSAIC project. http://www.mosaic-cloud.eu/. Accessed 12 Oct 2019
73. DAndria F, Bocconi S, Cruz JG, Ahtes J, Zeginis D (2012) Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings. In: 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp 407–414. https://doi.org/10.1109/SYNASC.2012.65
74. Kamateri E, Loutas N, Zeginis D, Ahtes J, D'Andria F, Bocconi S, Gouvas P, Ledakis G, Ravagli F, Lobunets O, Tarabanis KA (2013) Cloud4SOA: A Semantic-Interoperability PaaS Solution for Multi-cloud Platform Management and Portability. In: Lau K-K, Lamersdorf W, Pimentel E (eds). Service-Oriented and Cloud Computing, ESOCC 2013. pp 64–78. https://doi.org/10.1007/978-3-642-40651-5_6
75. Cloud4SOA consortium (2010) The Cloud4SOA project. http://www.cloud4soa.eu/. Accessed 12 Oct 2019
76. BEACON consortium (2015) The BEACON project: Enabling Federated Cloud Networking. http://www.beacon-project.eu/. Accessed 12 Oct 2019
77. Celesti A, Levin A, Massonet P, Schour L, Villari M (2016) Federated networking services in multiple openstack clouds. pp 338–352. https://doi.org/10.1007/978-3-319-33313-7_26
78. ATMOSPHERE consortium (2017) The ATMOSPHERE project: Adaptive, Trustworthy, Manageable, Orchestrated, Secure, Privacy-assuring Hybrid, Ecosystem for Resilient Cloud Computing. https://www.atmosphere-eubrazil.eu/. Accessed 12 Oct 2019
79. Castañeda IA, Blanquer I, de Alfonso C (2019) Easing the deployment and management of cloud federated networks across virtualised clusters. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science, CLOSER 2019, Heraklion, Crete, Greece, May 2-4, 2019. pp 601–608. https://doi.org/10.5220/0007877406010608
80. Rodero-Merino L, Vaquero LM, Gil V, Galán F, Fontán J, Montero RS, Llorente IM (2010) From infrastructure delivery to service management in clouds. Futur Gener Comput Syst 26(8):1226–1240. https://doi.org/10.1016/j.future.2010.02.013
81. Ferrer AJ, Hernández F, Tordsson J, Elmroth E, Ali-Eldin A, Zsigri C, Sirvent R, Guitart J, Badia RM, Djemame K, Ziegler W, Dimitrakos T, Nair SK, Kousiouris G, Konstanteli K, Varvarigou T, Hudzia B, Kipp A, Wesner S, Corrales M, Forgó N, Sharif T, Sheridan C (2012) OPTIMIS: A holistic approach to cloud service provisioning. Futur Gener Comput Syst 28(1):66–77. https://doi.org/10.1016/j.future.2011.05.022
82. Ferrer AJ, Lordan F, Ortiz D, Guitart J, Macias M, Panuccio P, M. Badia R, Ponsard C, Temporale C, García D, Sirvent R, Deprez J, Sommacampagna D, Djemame K, Armstrong D, Agiatzidou E, Ejarque J, Blasi L, Kammer M (2014) Ascetic - adapting service lifecycle towards efficient clouds. In: European Project Space on Information and Communication Systems - EPS Barcelona. SciTePress, Barcelona. pp 89–106. https://doi.org/10.5220/0006183400890106
83. ASCETiC consortium (2013) The ASCETiC project. http://ascetic-project.eu/. Accessed 12 Oct 2019
84. HARNESS consortium (2012) The HARNESS project: Hardware and Network-Enhanced Software Systems for Cloud Computing. http://www.harness-project.eu/. Accessed 12 Oct 2019
85. Coutinho JGF, Pell O, O'Neill E, Sanders P, McGlone J, Grigoras P, Luk W, Ragusa C (2014) Harness project: Managing heterogeneous computing resources for a cloud platform. In: Goehringer D, Santambrogio MD, Cardoso JMP, Bertels K (eds). Reconfigurable Computing: Architectures, Tools, and Applications. Springer. pp 324–329. https://doi.org/10.1007/978-3-319-05960-0_36
86. Paraiso F, Merle P, Seinturier L (2016) soCloud: a service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds. Computing 98(5):539–565. https://doi.org/10.1007/s00607-014-0421-x
87. OASIS (2011) Service Component Architecture (SCA). http://www.oasis-opencsa.org/sca. Accessed 12 Oct 2019
88. Giannakopoulos I, Papailiou N, Mantas C, Konstantinou I, Tsoumakos D, Koziris N (2014) CELAR: Automated application elasticity platform. In: IEEE International Conference on Big Data (Big Data). pp 23–25. https://doi.org/10.1109/BigData.2014.7004481
89. CELAR consortium (2012) The CELAR project. http://www.celarcloudproject.eu/. Accessed 12 Oct 2019
90. Selea T, Drăgan I, Fortiş T-F (2017) The CloudLightning Approach to Cloud-user Interaction. In: Proceedings of the 1st International Workshop on Next Generation of Cloud Architectures. CloudNG:17. pp 4–145. https://doi.org/10.1145/3068126.3068130
91. CloudLightning consortium (2018) The CloudLightning project. https://cloudlightning.eu/. Accessed 12 Oct 2019

Publisher's Note