International Journal on Software Tools for Technology Transfer (2023) 26:21–32
https://doi.org/10.1007/s10009-023-00705-2

GENERAL

Regular

Knowledge representation of the state of a cloud-native application

Joanna Kosińska[1] · Grzegorz Broton[1] · Maciej Tobiasz[1]

Accepted: 9 March 2023 / Published online: 22 June 2023

Abstract
Cloud Computing has revolutionized the way applications are developed, deployed, and maintained. Over the past decade, we have observed dynamically growing interest in Cloud Computing. The benefits of the cloud approach caused the increasing popularity of Cloud-native applications. Cloud-native is an approach to developing and deploying applications according to the concepts of DevOps, Continuous Integration/Continuous Delivery (CI/CD), containers and microservices. The knowledge about Cloud Computing has become extensive and complex. Fortunately, before Cloud-native applications development, there was a great deal of effort to develop tools for effective knowledge representation. Ontologies are a convenient way to show the relations between domain-specific concepts. In this paper, we propose an ontology named CNOnt that describes the state-of-the-art of Cloud-native applications. CNOnt covers aspects from the clusterization perspective. First, this paper presents the engineering perspective of building the CNOnt ontology. Second, we demonstrate a use case of our ontology that proves the correctness of CNOnt development. This ontology is exhausted in CNOnt Broker. It is a system that applies the information in the OWL file into the Kubernetes cluster and in reverse. The knowledge representation makes Cloud-native applications understandable to third-party systems and increases interoperability between different microservices.

Keywords Cloud-native · Cluster computing · Knowledge representation · Microservice · Semantic computing · Ontology

1 Introduction

Cloud-native [1–3] stands for an approach to building and running applications that fully exploit the advantages of the Cloud Computing (CC) delivery model [4]. Cloud native offers new standards for the development and deployment of applications [5]. Twelve-factor app [6] patterns describe building a software-as-a-service application. The patterns focus on speed, safety, and scale. The general objective of being Cloud-native is to improve the speed of application delivery, scalability, and resilience and reduce the technical risk of its deployment [2]. To be Cloud-native also means to be agnostic. Agnostic of the underlying infrastructure, OS, programming language, etc. This attitude decreases vendor lock-in. The developer treats all resources simply as cloud-based. An open community formed for standardization of Cloud-native Computing, namely Cloud-native Computing Foundation (CNCF) [7], provided a trail map that is an overview of the process of moving towards Cloud-native architecture [8]. One of its main and obligatory steps regards the orchestration of certain IT tasks run in containers. The proliferation of containers increased the need for their orchestration. Container orchestrators can also control containers remotely. Orchestrators help users build, scale, and manage complex applications and, at runtime, influence the lifecycle of a Cloud-native application.

The Cloud-native is a still-growing trend. It has one of the fastest adoption rates of any new approach [9]. R&D environments reported publications concerning that technology [10–12]. However, the papers show that Cloud-native is still in the incubation phase. There is no accepted and widely used Cloud-native definition. The rapid development of Cloud-native has made the domain a complex field of study. It uses many technologies and various terms to describe the landscape of Cloud-native. The diverse understanding of the terminology used to describe Cloud-native

✉ J. Kosińska
kosinska@agh.edu.pl

G. Broton
gbroton@student.agh.edu.pl

M. Tobiasz
mtobiasz@student.agh.edu.pl

[1] Faculty of Computer Science, Electronics and Telecommunications, Institute of Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

makes it hard to communicate between researchers, cloud engineers, and customers. Therefore, there is a great need to represent the accumulated knowledge about the cloud and present it in a way that is understandable to both people and machines.

The contributions of this paper are (i) the presentation of main vocabulary terms in the domain of Cloud-native, (ii) the knowledge representation of the Cloud-native application state together with the presentation of CNOnt ontology, and (iii) the presentation of a broker between Cloud-native applications and their semantic representation. We underline that this paper concerns only the aspects related to clusterization.

Ontologies [13–15] are a way to create a representation of domain knowledge. In this paper, we present an ontology about the state-of-the-art of Cloud-native applications within the clusterization aspect. Our goal was to create a comprehensive representation of knowledge in this field of study that contains the latest achievements and technologies. To achieve the goal, we have used OWL [16] with functional RDF [17, 18] representation.

The structure of this paper is as follows. First, we present the purpose and contribution of the research. The following section outlines current standards and technologies related to the knowledge representation of Cloud-native applications. This section also compares cluster description files (on the example of Kubernetes) with .owl ontology files. Based on this knowledge, the following section proposes a list of vocabulary terms important in the domain of Cloud-native limited to clusterization aspects. The vocabulary determines the scope and boundaries of our research. Section 5 introduces our CNOnt ontology. We provide an abstraction of the execution environment that aligns with the concepts of CNOnt. The next section presents a use case of our ontology. It introduces the layered architecture of the implemented system. The architecture is so general that modeling most cases with this architecture in mind is possible. The next section assesses the usability of the presented use case by assessing the application in overhead and performance tests. Finally, the last section concludes the research and shows its future directions.

2 Related work

Semantic Computing (SC) [19] acts as a bridge between humans and computers. It is a mixture of techniques that help in semantic analysis [20], natural language processing [21], data mining [22] and knowledge graphs [23]. In every system, data and its governance [24] play a crucial role. Knowledge Representation, a part of SC, is a solution to the problem of data management. The knowledge representation refers to expressing knowledge in the symbolic form, for instance, as ontologies. Ontology creates a link between machines and humans. Ontologies have gained popularity and, in many domains, have been successfully applied. Every field of study has taken advantage of domain ontologies [25–28].

CC history is known from archive magazines such as Weekly,[1] Forbes,[2] various blogs or books [29–31]. There are many definitions of CC. Most of them have common parts. However, NIST proposed the most complete and compact definition[3] [32]. It is still valid. The boundaries of distinguished service models (Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS)) are blurred. For example, SaaS platforms cannot exist without solid PaaS, and PaaS platforms require advanced infrastructure concepts offered in the IaaS model. Being Cloud-native means fully exploiting the CC model, its infrastructure, middleware, authentication and authorization, logging, network management, and many more, depending on the platform offerings. If we take advantage of all CC services, we optimize resource utilization [33]. Among the CC models, the mentioned services are offered by the IaaS and PaaS models [34]. From PaaS platforms Cloud-native platforms were derived. They significantly enhance the deployment of Cloud-native applications.

Cloud-native is not a new approach. However, it is still considered in software development [35]. Unfortunately, it does not touch on the aspects of the knowledge representation paradigm, although its usage in this context is a natural step. It seems that after hardening the philosophy of Cloud-native, other technologies would be applied to it, particularly SC.

Table 1 compares the cluster description files, made using the example of Kubernetes .yaml files, with the ontology files. The comparison reveals the pros and cons of both solutions. It also justifies our research on the representation of knowledge of Cloud-native applications.

The following paragraph describes the consecutive features of the comparison (see Table 1). Kubernetes uses .yaml files to represent the state of a cluster. They describe (some are optional and not always specified):

- containerized deployments that are running (and on which nodes),
- available and requested resources to the deployments,
- configuration of policies (e.g., restarts, upgrades, fault-tolerance).

The purpose of both types of files is the same. Both represent the state, in the case of this paper, of a cluster. The content of the files describes the user's intent of the cluster view. They inform the current workload configuration. The configuration, as well as the OWL files, can be easily copied or transferred between different projects. For developers, this

[1] http://www.computerweekly.com.

[2] http://www.forbes.com.

[3] https://www.nist.gov.

Table 1 Comparison of cluster description files (on the example of Kubernetes [36] .yaml files) with .owl ontology files

Feature	Cluster description files	Ontology files
Content	Desired/current view of the system	Desired/current view of the system
Reusability	Intended to be reusable.	Intended to be reusable.
Recipient	Is intended to be used when the information available in .yaml files need to be processed by the cluster.	Is intended to be used when the information available in documents needs to be processed by foreign applications.
The simplicity of usage	Intended for suitably experienced engineers.	Feasible for ordinary users.
Extensibility	Limited by the system's API conventions.	Freely extensible.
Management style	.yaml files are stored in multiple places (each cluster has its own configurations). This can lead to duplicates or conflicts.	Actions are taken based on an .owl file that can contain configurations regarding different clusters.

is a significant virtue. Another application can reuse a larger part of the files. Both approaches are intended to be reusable. From then on, there are some subtle differences between both approaches. They start already from the recipient. The .yaml files intend to be processed by the Kubernetes orchestrator, while different applications process the ontology files. The .owl files introduce an additional intermediary layer that supports communication in universal terms. The destiny of the compared files implies their internal complexity. The Kubernetes .yaml files are challenging. Appropriate skills of Kubernetes mastering are required. .owl files are readable by humans and machines. When it comes to extensibility, the two approaches presented differ. The first one is more restrictive due to the API conventions of the underlying system. The resource cannot be characterized by concepts unknown from the system's perspective (unless the new resource is added according to the operator pattern [37]). On the other hand, the OWL approach does not have this limitation and can describe any property of each resource. There is also a slight difference in the management style. Each cluster stores its configuration locally, while the .owl approach enables having all configurations in one place. The pros and cons of both modes are the same as when comparing distributed versus centralized management. Which style is chosen depends on the particular case.

The distinguished features and their values allow us to state that both solutions apply to different aspects and, rather than being in competition with each other, are complementary. Exhausting both can address more interesting scenarios in Cloud-native.

To the best of our knowledge, research papers describe ontologies generally related to CC, not particularly to Cloud-native applications [38, 39]. The second mentioned paper proposes a novel approach to the description via StratusML and lifecycle management of Cloud applications. This concept can be narrowed down to Cloud-native applications. The same author, in his PhD thesis [40] erroneously named one of its chapters: Architecture Framework for Cloud Native Applications, as its content regards ordinary CC applications. In turn, research [41] uses a hybrid of FMs (Feature Models) and ontology that helps to manage multi-cloud configurations. The paper [42] proposes an ontology model that addresses the problem of recommending services based on their specific requirements. The book [43] describes useful examples of semantic SOA approaches and perspectives of semantic Web Services with an overview of its standards. Complementary of these descriptions are the papers [44, 45] that present the state of the art in the application of ontologies in CC. As can be noticed, these ontologies cover only certain aspects of a Cloud-native application, usually those that already exist in CC environments.

Also, to our best knowledge, research papers related to clusterization and orchestration aspects did not suit Cloud-native applications requirements. The paper [46] identifies key capabilities of Fog orchestrators. However, it omits its semantic features. While describing technological directions, the future work of the research does not mention the importance of knowledge representation. Research [47] proposes a new overlay approach to address Cloud-native needs. Moreover, it presents the ontology-based reasoning framework for service composition but does not include this ontology in the proposed OverCloud system. In summary, it is not possible to use the concepts presented in these papers. They are too specific and focus on the given context.

It should also be noted that the above papers are not fresh, meaning that the proposed knowledge representation is not up-to-date. Cloud-native has risen on top of CC. It is working at the microservices level and among Cloud-native apps. Most of the concepts introduced are not yet present in well-known modeling languages (CloudML [48], CAMEL [38], etc.). However, it is possible to map the concepts or even use an existing CC ontology and extend it. The mentioned research [45] lists the Universal Cloud Interface (UCI) initiative, with the aim of solving portability and interoperability issues. Our CNOnt ontology is based on the UCI ontology.

We also have to emphasize that research in the area of Cloud-native applications ontologies, knowledge representation, its management, and governance in most systems is in its infancy [24]. The direct inspiration for the present research comes from research [49] in which the lack of an additional layer related to the representation of knowledge is apparent. The paper designs the AMoCNA (Autonomic Management of Cloud-native Applications) framework with

the MDA[4] meta-modelling approach. In the present paper, we propose a layer on top of that framework, another meta-modeling approach that enables the representation of the knowledge.

3 Requirements of knowledge representation of the state of a Cloud-native application

The Challenge of Sustainable Smart Societies, emerging as Society 5.0 [50] is based on incl. large-scale computing of big data. Society 4.0 has already produced a large amount of information that should be managed autonomously. The management of increasing amounts of data was one of the main reasons why Autonomic Computing (AC) [51] has emerged. AC is rooted in functions of the human body, namely *Autonomous Nervous System*.[5] In the same way, as the human brain is free from dealing with vital functions, the computing system uses the same unconscious way to perform its tasks.

The tight integration of enterprise systems with a Cloud-native approach is undoubtedly attractive from a financial perspective. However, it cannot complicate resource management and utilization. This requirement is particularly critical if the management cost increases significantly. A remedy seems to be the AC paradigm in Cloud-native environments. AC paradigm was introduced in response to the need for increasing complexity in the installation, configuration, optimization, and maintenance of heterogeneous computing systems. However, it is hard to combine both technologies without solid information processing [52] including phases as data representation, thus formalizing the environment and simplifying the complexity of the system.

The General Knowledge Management process [53] consists of five steps: creation, storage, retrieval, transfer, and application of knowledge. Although the cited paper refers to the software documentation problem, proposed Fig. 1 can be treated more generally as a high-level view of the use of knowledge. Our solution concerns the second step, that is, Knowledge Storage or, more precisely, Knowledge capture and representation (KCR). We propose to acquire and extract knowledge of the state of a Cloud-native application from diverse sources, and then express it by an ontology so the knowledge can be used by different applications or humans.

We distinguished several features of the representation of knowledge of the state of a Cloud-native application. They are as follows:

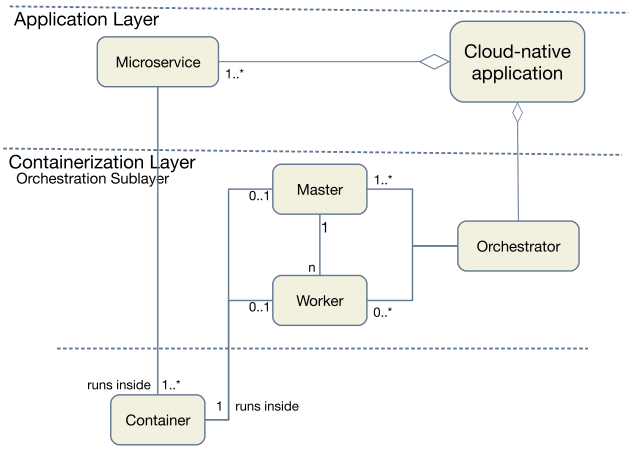

Fig. 1 High-level view of clusterization aspects of a Cloud-native application

- Support for the Cloud-native paradigm – The proposed solution should take into account the dynamic nature of a Cloud-native environment and particularly their rapid changes in resource requirements and structure of loosely coupled microservices.
- Extensibility – The proposed model should be extensible; thus, it is still abstract and vendor-neutral. Along these lines, it can capture many aspects such as authentication, authorization, scalability, or geographic distribution. Holistic knowledge will give a true view of a distributed execution environment. Consequently, it will provide a wide array of capabilities (e.g., health management, optimization).
- Formalized – The blueprint for representing knowledge should be readable by humans and machines. They are comfortable for humans because they create a common understanding between people who have different points of view on the domain.
- Reason over data – The proposed solution should enable one to determine facts that are not explicitly defined. For example, if an artifact is a Node, we can deduce that it is a part of a cluster.
- Low overhead – The overload generated while operating systems that support knowledge representation should be as low as possible. At the same time, the overhead generated by components that realize the transformation of knowledge representing the state of a Cloud-native application, should also remain low.

The itemized requirements determine the selection of knowledge representation techniques.

[4] Model Driven Architecture.

[5] unconsciously regulates such functions as heart rate, digestion, respiratory rate, vasomotor activity, and so on.

Table 2 Identified vocabulary terms of a Cloud-native application related to the clusterization aspect

Concept	Definition
Cloud-native application	Is an application developed and deployed according to concepts of DevOps [55], CI/CD [56, 57], containers [4] and microservices [11].
Microservice	Microservices architecture is a style derived from SOA [58] where the application is a collection of small services, independent of each other. A microservice can be deployed, scaled, and restarted without disturbing the work of other microservices. They communicate with the environment through the HTTP API. Microservices decompose monolithic applications into small, independent components.
Orchestrator	Is a workflow management solution. The orchestrator enables the automation of the creation, monitoring, and deployment of resources in the environment [59]. They refer to the act of container scheduling, cluster management, and also provisioning of additional hosts.
Container	A container image is a lightweight, standalone, executable package of software that includes everything needed to run it: code, runtime, system tools, and system libraries settings [60]. It is just a set of serialized file systems with configuration and metadata. Deploying it (running as a container) means mounting the filesystem in a namespace.
Cluster	Is the environment in which the containers run with their orchestrator. A cluster consists of two or more nodes that collaborate to accomplish a particular task. A node represents a single computing resource.
Master	Is a cluster node that controls and manages a set of nodes that carry workloads. In the context of Cloud-native, clusters and orchestrators are synonyms.
Worker	Is a cluster node that carries a workload.

4 Presentation of vocabulary terms important in the domain of Cloud-native

Cloud-native generally has a positive overtone. However, it has some negative aspects. The following part of the paper points them out.

The current trend in IT is to move the workloads from bare metal to virtualized environments and then to containers towards serverless. In response to this challenge, microservices have emerged. Enterprise microservice applications contain thousands of instances of containerized services. Such distributed, often geographically dispersed, applications enabled an additional cost of management that businesses have to bear. New offers and services overwhelm the end-user. They are frequently updated, improved, and deleted. Additionally, each provider tends to use its service representation. Due to interoperability [54] and the mentioned problems, we noticed a strong need for research in this area.

Paper [49] summarizes the notion of a Cloud-native application in a figure of its execution environment. The environment is divided into layers. The current scope of research is limited to clusterization aspects. Therefore, it allows us to restrict the distinguished components to the components of the presented Clusterization Layer (Fig. 1). This figure depicts the fundamental concepts of Cloud-native from a clusterization perspective. It emphasizes the importance of an orchestrator (its responsibility is to create a cluster), which provides automated management of the containers and coordinates their creation, monitoring, and deployment. Its role is crucial for Cloud-native applications since their execution environments consist of multiple, loosely coupled microservices running as containers.

Analysis of Cloud-native applications' requirements (pointed out in Sect. 1) and their knowledge representation (Sect. 3) made it possible for us to propose a unified vocabulary of terms in the domain of Cloud-native (Table 2). We define only new terms compared with CC ontologies that are only in the Cloud-native. The identified terms align with the components shown in Fig. 1. The proposed vocabulary is fundamental to the CNOnt ontology that we introduce in the next section.

5 CNOnt: Cloud-native ontology

Knowledge representation of the Cloud-native application state concerns many aspects depending on the context. The low-level knowledge representation of infrastructure includes, among others, provisioning, orchestration, migration, tagging, discovery, etc. The containerization enforces tasks and capabilities such as state management and scheduling, high availability and fault tolerance, security, networking, service discovery, continuous deployment, governance, etc. The prior tasks are similar as in CC and are widely discussed with those technologies [61–63]. The Unified Cloud Interface Project (UCI) ontology [64] proposes representation of the knowledge of CC resources. This representation is generic and includes most CC features [61] such as cloud service models (IaaS, PaaS, SaaS) and fundamental infrastructure elements such as servers, virtual machines, and network elements. Extra features specific to certain cloud providers build a separate `Vendor_Extensions` branch. The ontology proposed by UCI is part of a large project involving Semantic Web technologies.

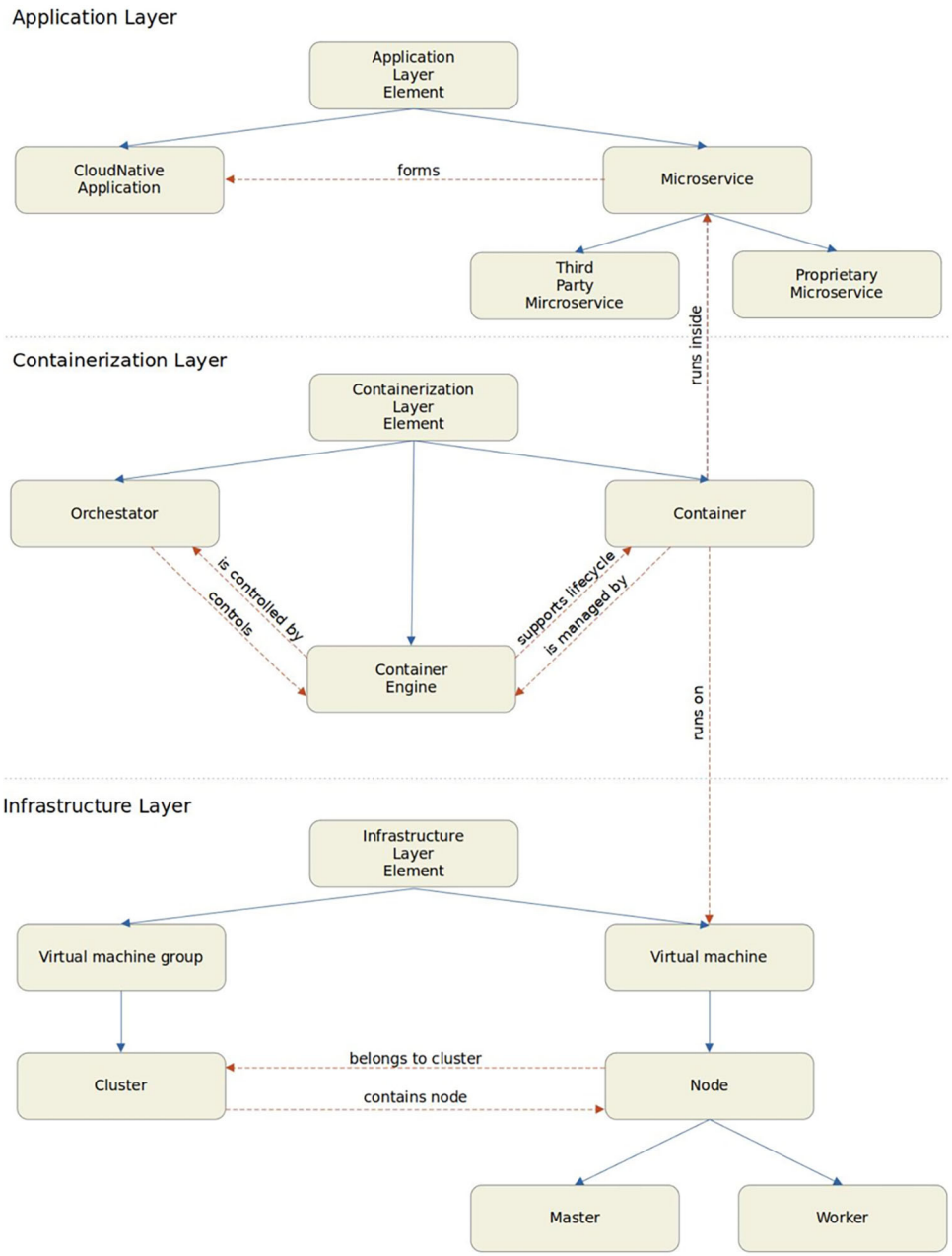

Fig. 2 Cloud-native concepts from the ontology perspective

The second group of tasks, the containerization tasks, is mainly the responsibility of orchestrators' strategies. Supporting these management tasks is still a field of many research works [65–67]. However, as noted in Sect. 2, the knowledge representation aspects have not been touched and unified yet. To extend interoperability among clusterization aspects, we propose a domain Cloud-native ontology named CNOnt. CNOnt, based on knowledge acquisition and processing capabilities, represents the state of the Cloud-native application. It extends widely known CC ontologies (e.g., TOSCA [68] project) as it defines Cloud-native concepts. We postulate that the actual configuration of the cluster on which the Cloud-native application is deployed and run in the main reflects the state of the Cloud-native environment.

We propose to extend the UCI ontology with the vocabulary terms proposed in Sect. 4. We named this extension CNOnt. The recommended concepts, depicted in Fig. 2, align with the principles of the Cloud-native. The characteristics of the UCI ontology are superficial and not sufficient for presenting the Cloud-native concepts. The InfrastructureElement proposed by the UCI ontology includes terms covering multiple areas of Cloud-native landscape, not solely related to the infrastructure. According to UCI taxonomy, some of the terms (e.g., ApplicationContainer) belong to

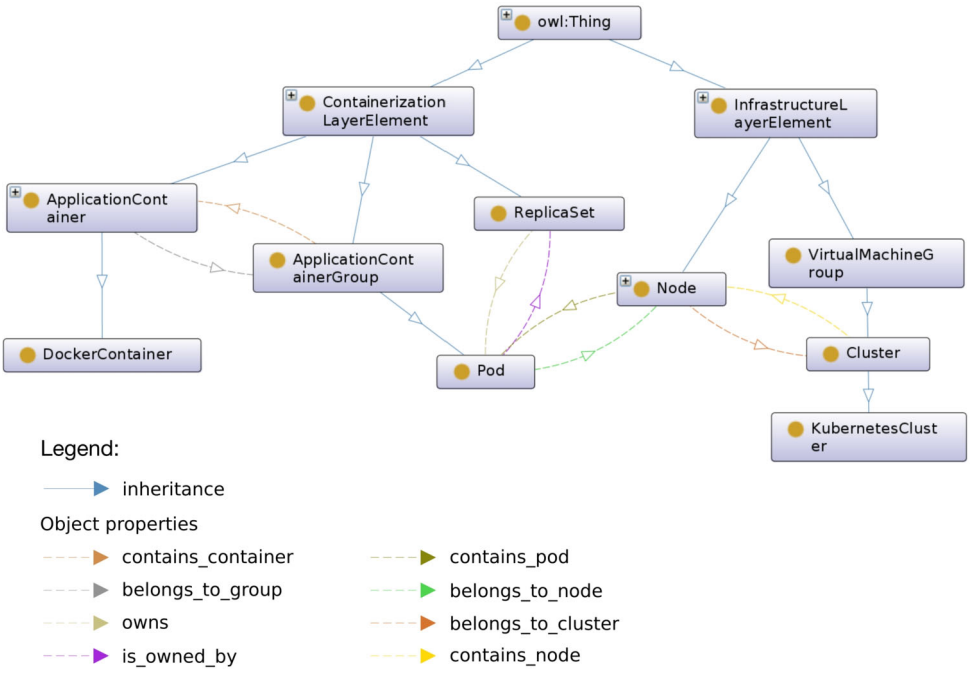

Fig. 3 Main CNOnt classes (Color figure online)

Contenerization Layer. Consequently, to overcome UCI limitations, we have split the InfrastructureElement class from the UCI ontology into three separate entities: ApplicationLayerElement, ContainerizationLayerElement, and InfrastructureLayerElement. Each of them groups distinct concepts of Cloud-native landscape by their abstraction level. The split allowed for the extension of the ontology with new entities, assigned as subclasses to the corresponding layer classes. Figure 2 presents all important classes in the context of the Cloud-native execution environment. All are modeled in the CNOnt ontology. The names of the depicted relations reflect the parameters of the properties of the objects.

According to [69] CNOnt can be classified as an Enterprise Ontology. Adding a layer of concepts related to a particular clusterization technology - Kubernetes - allows us for such classification. The key concepts of Kubernetes, such as Pod and ReplicaSet, are represented by OWL classes, as shown in Fig. 3. This figure comes from the OntoGraph view from the Protege [70] tool. Among others, we defined classes of Node, Pod, and DockerContainer. They inherit from the appropriate parent classes. The blue arrows mark the inheritance relationship. For clarity, we have decided to show only some of the relationships. Other arrows represent relationships modeled by object properties, for example `contains_pod` (arrow from Node to Pod), `belongs_to_node` (arrow from Pod to Node). Some basic concepts that form the lowest layer of abstraction are modeled by data properties, e.g. `cpu_limits`, `cpu_requests`. These data properties are visible in CNOnt as shown in the Listing 1. The above Listing presents fragments from the CNOnt that define some of the data properties of the Docker

```
# Data Property: :cpu_limits (:cpu_limits)
DataPropertyDomain(:cpu_limits :DockerContainer)
DataPropertyRange(:cpu_limits xsd:string)

# Data Property: :cpu_requests (:cpu_requests)
DataPropertyDomain(:cpu_requests :DockerContainer)
DataPropertyRange(:cpu_requests xsd:string)
```

Listing 1 CNOnt declaration of data properties

Container class. The example shows properties of containers related to CPU resource utilization as used by Kubernetes. Unlike object properties that define relations between individuals, data properties define relations between individuals and datatypes [71]. In this example, xsd:string is a primitive datatype used as a range of the data property. Although the CPU value is mostly specified by a number, in the presented work, xsd:string was the most suitable type to represent it.

Our CNOnt ontology is publicly available.[6] The use of ontologies has expanded in recent years. The following section presents one possibility of CNOnt usage.

6 Use case of CNOnt

The following description shows the practical usage of the CNOnt ontology. For this reason, we developed a CNOnt Broker. We present its capability of saving the current environment's state based on concepts specified by the CNOnt

[6] https://github.com/greg9702/CNOnt-Broker/blob/master/core/ontology/assets/CNOnt_template.owl.

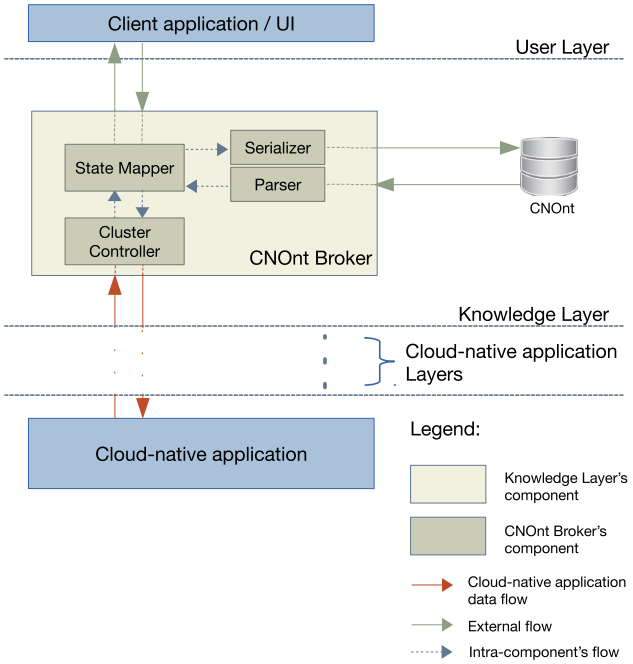

Fig. 4 Example of CNOnt usage (Color figure online)

ontology. However, as stated in the Abstract, the reverse direction is also possible.

The CNOnt Broker presented can help automate the gathering and preparation of data used for further analysis. Data can be collected from heterogeneous sources and then unified. As a consequence, the CNOnt Broker offers consistent data that describe the domain.

The system consists of several components, divided into three logical layers that provide its capabilities. As shown in Fig. 4, these layers are as follows:

- User Layer - a top layer of the system architecture, identified with a user interface, usually realized as a web application. This layer creates a link between the system and a human. Its role is to provide user-friendly interaction with the system.
- Knowledge Layer - a central layer of the system that stores and interacts with CNOnt. It consists of two components, namely CNOnt and CNOnt Broker. The CNOnt ontology is a source of knowledge for the CNOnt Broker. Furthermore, the Knowledge Layer links all the other layers and provides data flow between those layers.
- Cloud-native application Layers - the bottom layer of the system, usually identified as an application execution environment. In the presented work, clusterization aspects realize the Kubernetes orchestrator.

The components of each layer are loosely coupled services running in its docker container. The components of the User Layer and the Cloud-native application Layer can be easily changed or replaced. Replacement is possible due to

```
# Object Property: :belongs_to_cluster
  (:belongs_to_cluster)

SubObjectPropertyOf(:belongs_to_cluster
    owl:topObjectProperty)
InverseObjectProperties(:belongs_to_cluster
    :contains_node)
ObjectPropertyDomain(:belongs_to_cluster :Node)
ObjectPropertyRange(:belongs_to_cluster
    :KubernetesCluster)
```

Listing 2 Fragment of Kubernetes deployment returned by CNOnt Broker

the generic characteristics of the Knowledge Layer. The ontology, by definition, is flexible and generic. However, there is a need for a broker that will make the CNOnt benefits available to the whole system. CNOnt Broker accomplishes this. It is the main component of the system.

CNOnt Broker offers an environment state mapping functionality. It acquires knowledge of ontology and then constructs individuals [72] representing the current state of the environment. The output is generated in an OWL functional format. It is worth noting that the CNOnt Broker functionality is not limited to a particular environment. It is environmental-agnostic. Figure 4 presents its architecture. On the basis of the roles of all components, they are divided into three main groups. The first is related to the interaction with ontology. The components `Serializer` and `Parser` provide domain concepts defined in the ontology. The next group constitutes components related to the data sources (the presented work has only one data source - a Kubernetes cluster). These components communicate with the bottom layer of the system, that is, the Cloud-native application layers. They acquire data on the current state of the environment. In CNOnt Broker, this functionality provides the `Cluster Controller` component that shares unstructured data. `State Mapper` processes the unstructured data and integrates it with the domain knowledge obtained from the CNOnt ontology. Such structure, and thus CNOnt Broker capabilities, enable deducing the actual relationships inside a Kubernetes cluster. A fragment of a Kubernetes deployment is shown in Listing 2.

It also coordinates the flow of data within the system. It exposes HTTP endpoints, enabling interaction with the system.

7 Evaluation

The purpose of the presented evaluation is to convince the Readers to use CNOnt and measure the performance of CNOnt Broker associated with the growing number of entities (Kubernetes abstractions) in the environment. CNOnt Broker under the hood invokes the functions of the Kubernetes API Server to manage the cluster. Therefore, the

imposed overhead results only from the knowledge representation of the cluster.

Based on Openstack Nova instances and particularly set resources quotas, we set up a 20-node Kubernetes cluster consisting of one master and 19 worker nodes. The nodes were set in the .owl file and then pulled by CNOnt Broker and created accordingly. Each machine was assigned 2 vCPU and 2 GiB of memory. To demonstrate the practical aspect of the CNOnt ontology, we added individuals to the .owl file that reflect our evaluation infrastructure. The relationship between the cluster and the nodes is modeled by the properties of the inverse objects `belongs_to_cluster` and `contains_node` (these properties are declared in CNOnt). Similarly, appropriate object properties define individuals representing pods and docker containers. This setup makes it possible for CNOnt Broker to deduce the actual relationships inside a Kubernetes cluster.

The evaluation measures the response time of the CNOnt Broker relative to the growing number of entities in the cluster. Adding more nodes while the experiment is running will result in peak changes in total cluster resources. The resize would have a noticeable impact on the results. Therefore, the size of the cluster was constant during the whole process. By manipulating the number of running Pod replicas, CNOnt Broker was processing a different number of entities. However, running more replicas than the cluster can handle influenced the results because communication and cluster consistency were disturbed due to the overload of the nodes. Therefore, we used a reasonable range of the total number of replicas running in the cluster. The next aspect was to select the desired deployment, which besides its opportunity to scale out, generates some load on the machines. We decided to use an open-source demo Cloud-native application - sock-shop deployment [73]. The selected version of this deployment consists of 14 Pods related to the shop application and two replicas of Pods that simulate the application load. The chosen deployment provides quite varied Pods in the term of their roles and properties. More interesting for the presented scenario is that CNOnt Broker can handle optional properties of the objects. As the result, different objects of the same type can have different sets of properties. The last considered aspect is to decide which Pods should be scaled out. We have decided to simultaneously scale all Pods out due to their previously mentioned diversity. Scaling at the same time out all Pods fulfills the requirements for the evaluation, achieving as objective results as possible when adding new running replicas.

The evaluation process started with the number of 48 Pods running in the cluster. Every step of the experiment added an additional replica of every Pod from the sock-shop Cloud-native application. For every new setup, the deployment launched, also producing ten measurements. Each measurement counted the time needed to obtain a response that contains the cluster state mapping. The experiment stopped when it exceeded 500 total Pods running in the cluster.

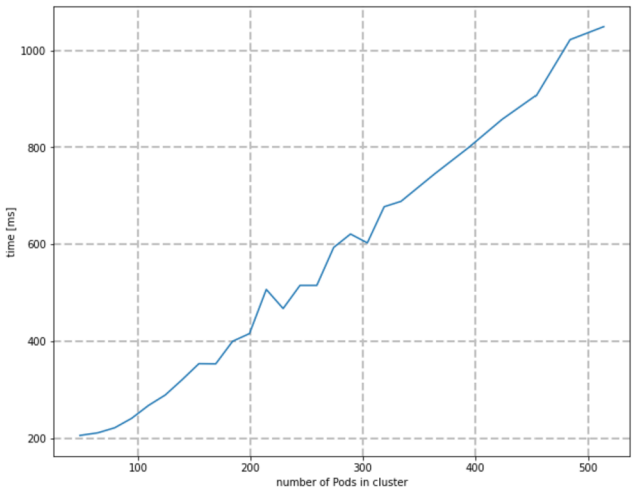

Fig. 5 CNOnt Broker performance

It is worth noting that CNOnt Broker was running on the master node. This node was not available for scheduling to other Pods used in this experiment. Launching it on some other node could affect the results. During the experiment, the load on the worker nodes increased, which could worsen the response time of the profiled system. Running a CNOnt Broker on the master node with a fixed number of running Pods throughout the evaluation process provides an immutable runtime environment for all steps.

Figure 5 shows the response time of the CNOnt Broker in milliseconds versus the number of Pods running in the cluster. The response time is calculated as a mean value acquired from 10 measurements in each step.

In the current implementation, CNOnt Broker does not use any cache mechanisms. It fetches the data from the sources every time. A growing number of Pods running in the cluster affects the Kubernetes API server response time, but this aspect is beyond the scope of this work. In the presented results, CNOnt Broker's total response time includes the process of fetching the data. The system in this scenario uses only one source of data. The source of data is the API server accessed via the Kubernetes client. Using multiple sources would not significantly affect the performance of the system. It is possible to handle these processes in parallel.

We observe a linear relationship between the increase in the number of replicas and the response time. The results show that the response time is tightly dependent on the number of entities processed by the presented system. In particular, it increases linearly with the number of running Pod replicas in the cluster. Thus, CNOnt Broker introduces a very low overhead. Such results make the response time of the system predictable. The results met our expectations. They show that the costs associated with CNOnt are negligi-

ble compared to the functionalities and benefits pointed out in previous sections that it brings.

8 Summary

This paper systematizes vocabulary terms that are present in Cloud-native execution environments. Additionally, the proposed CNOnt ontology utilizes these terms. Implemented and installed on a properly prepared testbed, the CNOnt Broker verifies all propositions. It was necessary to establish a testbed due to the need to analyze and assess the proposed approach. The performance of the proposed concepts is satisfactory and does not overload the execution environment. With success, the semantic extensions enrich the Cloud-native execution environments, making them more understandable between different microservices and humans. The introduction of knowledge representation also facilitates reasoning [74] on the cluster capabilities and mitigates lock-in risks [54].

The development of the CNOnt Broker ended successfully. To accomplish its tasks, it communicates with a widely known and established Kubernetes orchestrator. Although it is just a prototype, it provides the most important features and enables further development. The main goals are the following:

- easily extensible ontology architecture,
- retrieval of Kubernetes deployment based on ontology,
- integration of OWL and Kubernetes.

Also, we implemented a React-based [75] client application that allows one to create, delete, and preview the deployment. Nevertheless, further extensions of the CNOnt Broker and the CNOnt ontology are needed.

The presented research only tackles the clusterization aspect of Cloud-native environments. Equally important to explore is the programming philosophy perspective. The missing investigation regards DevOps practices, particularly CI/CD concepts. Such extension of CNOnt would make it a highly complete solution among Cloud-native applications.

Acknowledgements The research presented in this paper was supported by funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education. This research was supported in part by PLGrid infrastructure.

Data Availability Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare that they have no conflict of interest.

References

1. Stein, M.: Migrating to Cloud-Native Application Architectures. O'Reilly Media, USA (2015)
2. Currie, A.: The Cloud Native Attitude. Container Solutions Publishing, Netherlands (2017)
3. Davis, C.: Cloud Native Patterns: Designing Change-Tolerant Software. Manning Publications, USA (2019)
4. Cloud-native apps. https://pivotal.io/cloud-native. Last visited August, 2020
5. Brunner, S., Blöchlinger, M., Toffetti, G., Spillner, J., Bohnert, T.M.: Experimental evaluation of the cloud-native application design. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 488–493 (2015)
6. The twelve factor app. https://12factor.net. Last seen on June, 2020
7. Cloud Native Computing Foundation. https://www.cncf.io. Last seen on November 2020
8. Cloud Native LandScape. https://github.com/cncf/landscape. Last seen on November, 2020
9. 2019 CNCF Survey results are here: Deployments are growing in size and speed as cloud native adoption becomes mainstream. https://www.cncf.io/blog/2020/03/04/2019-cncf-survey-results-are-here-deployments-are-growing-in-size-and-speed-as-cloud-native-adoption-becomes-mainstream/?utm_source=thenewstack&utm_medium=website&utm_campaign=platform. Last seen on November, 2020
10. Kratzke, N., Peinl, R.: Clouns - A Cloud-native Application Reference Model for Enterprise Architects (2017). CoRR arXiv:1709.04883
11. Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., Bohnert, T.M.: Self-managing cloud-native applications: design, implementation, and experience. Future Gener. Comput. Syst. **72**, 165–179 (2017)
12. Gannon, D., Barga, R.a.: Cloud-native applications. IEEE Cloud Comput. **4**, 16–21 (2017). https://doi.org/10.1109/MCC.2017.4250939
13. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993). https://doi.org/10.1006/knac.1993.1008
14. Noy, N.F., McGuinness, D.L.: Ontology development 101: a guide to creating your first ontology. Tech. Rep., Stanford University (2001). http://www-ksl.Stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html
15. Dou, D., Wang, H., Liu, H.: Semantic data mining: a survey of ontology-based approaches. In: Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), pp. 244–251 (2015)
16. Owl web ontology language guide. https://www.w3.org/TR/2004/REC-owl-guide-20040210/. Last visited November, 2020
17. Structural specification and functional-style syntax. https://www.w3.org/2007/OWL/draft/ED-owl2-syntax-20081128/. Last visited September, 2020

18. Martins, J., Nunes, R., Karjalainen, M., Kemp, G.J.L.: A functional data model approach to querying RDF/RDFS data. In: Gray, W.A., Jeffery, K.G., Shao, J. (eds.) Sharing Data, Information and Knowledge, 25th British National Conference on Databases, BNCOD 25, Cardiff, UK, July 7-10, 2008. Proceedings, Lecture Notes in Computer Science, vol. 5071, pp. 153–164. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-70504-8_14
19. Hitzler, P., Krtzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies, 1st edn. Chapman & Hall, London (2009)
20. Sundaram, S.S., Abraham, S.S.: Semantic representation for age word problems with schemas. New Gener. Comput. **37**(4), 429–452 (2019). https://doi.org/10.1007/s00354-019-00069-9
21. Sundaram, S.S., Khemani, D.: Natural language processing for solving simple word problems. In: Proceedings of the 12th International Conference on Natural Language Processing, pp. 394–402. NLP Association of India, Trivandrum, India (2015). https://www.aclweb.org/anthology/W15-5955
22. Rettinger, A., Lösch, U., Tresp, V., d'Amato, C., Fanizzi, N.: Mining the semantic web. Data Min. Knowl. Discov. **24**(3), 613–662 (2012). https://doi.org/10.1007/s10618-012-0253-2
23. Edelstein, E., Pan, J.Z., Soares, R., Wyner, A.: Knowledge-driven intelligent survey systems towards open science. New Gener. Comput. **38**(3), 397–421 (2020). https://doi.org/10.1007/s00354-020-00087-y
24. Al-Ruithe, M., Benkhelifa, E., Hameed, K.: A systematic literature review of data governance and cloud data governance. Pers. Ubiquitous Comput. (2018). https://doi.org/10.1007/s00779-017-1104-3
25. Hernandez, N., Mothe, J., Chrisment, C., Egret, D.: Modeling context through domain ontologies. Inf. Retr. **10**(2), 143–172 (2007)
26. Bashar, M.A., Li, Y.: Interpretation of text patterns. Data Min. Knowl. Discov. **32**(4), 849–884 (2018)
27. Elçi, A., Çelik Ertuğrul, D.: Ontology-based smart medical solutions. Expert Syst. **37**(1), e12518 (2020). https://doi.org/10.1111/exsy.12518
28. Lu, W., Cai, Y., Che, X., Lu, Y.: Joint semantic similarity assessment with raw corpus and structured ontology for semantic-oriented service discovery. Pers. Ubiquitous Comput. **20**(3), 311–323 (2016). https://doi.org/10.1007/s00779-016-0921-0
29. Hwang, K., Dongarra, J., Fox, G.C.: Distributed and Cloud Computing: From Parallel to the Internet of Things, 1st edn. Morgan Kaufmann, San Mateo (2011)
30. Koulopoulos, T.M.: Cloud Surfing: A New Way to Think About Risk, Innovation, Scale and Success (Social Century), 1st edn. Routledge, London (2012)
31. Jamsa, K.: Cloud Computing, 1st edn. Jones & Bartlett, Burlington (2012)
32. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Tech. Rep., National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8930 (2011)
33. Nandhini, J.M., Gnanasekaran, T.: Enhanced fault identification and optimal task prediction (efiotp) algorithm during multi-resource utilization in cloud-based knowledge and personal computing. Pers. Ubiquitous Comput. (2019)
34. Rani, D., Ranjan, R.K.: A comparative study of saas, paas and iaas in cloud computing. International Journal of Advanced Research in Computer Science and Software Engineering **4**(6) (2014)
35. Kratzke, N., Quint, P.C.: Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. J. Syst. Softw. **126**(C), 1–16 (2017). https://doi.org/10.1016/j.jss.2017.01.001
36. Production-grade container orchestration. https://kubernetes.io. Last seen on May, 2021
37. Ibryam, B., Huß, R.: Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications. O'Reilly Media, USA (2019). https://books.google.pl/books?id=8WmRDwAAQBAJ
38. Achilleos, A.P., Kritikos, K., Rossini, A., Kapitsaki, G.M., Domaschka, J., Orzechowski, M., Seybold, D., Griesinger, F., Nikolov, N., Romero, D., Papadopoulos, G.A.: The cloud application modelling and execution language (camel). J. Cloud Comput. **8**(1), 20 (2019)
39. Hamdaqa, M., Tahvildari, L.: Stratuspm: an analytical performance model for cloud applications. In: 10th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, MESOCA 2016, Raleigh, NC, USA, October 3, 2016, pp. 24–31. IEEE Comput. Soc., Los Alamitos (2016). https://doi.org/10.1109/MESOCA.2016.13
40. Hamdaqa, M.: An integrated modeling framework for managing the deployment and operation of cloud applications. Ph.D. thesis, University of Waterloo, Ontario, Canada (2016) http://hdl.handle.net/10012/10672
41. Quinton, C., Haderer, N., Rouvoy, R., Duchien, L.: Towards multi-cloud configurations using feature models and ontologies. In: Proceedings of the 2013 International Workshop on Multi-Cloud Applications and Federated Clouds, pp. 21–26 (2013). https://doi.org/10.1145/2462326.2462332
42. Rajendran, V., Swamynathan, S.: A novel approach for semantic service discovery in cloud using broker agents. In: Proceedings of International Conference on Advances in Computing, Communication and Information Science (ACCIS'14) (2014). https://doi.org/10.13140/2.1.4022.6085
43. Davies, J., Studer, R., Warren, P.: Semantic Web Technologies: Trends and Research in Ontology-Based Systems. Wiley, Hoboken (2006)
44. Bergmayr, A., Wimmer, M., Kappel, G., Grossniklaus, M.: Cloud modeling languages by example. In: Proceedings of the 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, SOCA'14, pp. 137–146. IEEE Comput. Soc., USA (2014). https://doi.org/10.1109/SOCA.2014.56
45. Imam, F.T.: Application of ontologies in cloud computing: the state-of-the-art (2016). CoRR arXiv:1610.02333
46. Velasquez, K., Abreu, D.P., Assis, M.R.M., Senna, C., Aranha, D.F., Bittencourt, L.F., Laranjeiro, N., Curado, M., Vieira, M., Monteiro, E., Madeira, E.: Fog orchestration for the Internet of everything: state-of-the-art and research challenges. J. Internet Serv. Appl. **9**(1), 14 (2018)
47. Han, J., Park, S., Kim, J.: Dynamic overcloud: realizing microservices-based iot-cloud service composition over multiple clouds. Electronics **9**(6), 969 (2020). https://doi.org/10.3390/electronics9060969
48. Gonçalves, G., Endo, P.T., Santos, M., Sadok, D., Kelner, J., Melander, B., Mångs, J.E.: Cloudml: an integrated language for resource, service and request description for d-clouds. In: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, pp. 399–406 (2011)
49. Kosińska, J., Zieliński, K.: Autonomic management framework for cloud-native applications. J. Grid Comput. (2020)
50. Deguchi, A., Hirai, C., Matsuoka, H., Nakano, T., Oshima, K., Tai, M., Tani, S.: What is Society 5.0? In: Society 5.0, pp. 1–23. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-2989-4_1
51. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/MC.2003.1160055
52. Cardoso, J., Sheth, A.: The semantic web and its applications. In: Semantic Web Services, Processes and Applications, pp. 3–33. Springer, Berlin (2006)
53. Ding, W., Liang, P., Tang, A., van Vliet, H.: Knowledge-based approaches in software documentation: a systematic literature re-

view. Inf. Softw. Technol. **56**(6), 545–567 (2014). https://doi.org/10.1016/j.infsof.2014.01.008
54. Opara-Martins, J., Sahandi, R., Tian, F.: Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. J. Cloud Comput. **5**(1), 4 (2016)
55. Where the world meets devops. https://devops.com. Last visited August, 2020
56. Fowler, M.: Continuous Integration. https://martinfowler.com/articles/continuousIntegration.html. Last seen on November, 2020
57. Fowler, M.: Continuous Delivery. https://martinfowler.com/bliki/ContinuousDelivery.html. Last seen on November, 2020
58. Marks, E.A., Bell, M.: Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology. Wiley, New York (2006)
59. Orchestrator - technet - Microsoft. https://technet.microsoft.com/en-us/library/hh237242(v=sc.12).aspx. Last seen on July, 2020
60. What is a container. https://www.docker.com/what-container. Last seen on March, 2019
61. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: a Berkeley View of Cloud Computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (2009). http://www.eecs.Berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
62. Galante, G., Bona, L.C.E.d.: A survey on cloud computing elasticity. In: Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing, UCC'12, pp. 263–270. IEEE Comput. Soc., Washington (2012). https://doi.org/10.1109/UCC.2012.30
63. Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R.: Cost of virtual machine live migration in clouds: a performance evaluation. In: Proceedings of the 1st International Conference on Cloud Computing, CloudCom'09, pp. 254–265. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-10665-1_23
64. Unified cloud interface project. https://code.google.com/archive/p/unifiedcloud/. Last visited September, 2020
65. Thinakaran, P., Raj, J., Sharma, B., Kandemir, M.T., Das, C.R.: The curious case of container orchestration and scheduling in gpu-based datacenters. In: Proceedings of the ACM Symposium on Cloud Computing, SoCC'18, p. 524. Assoc. Comput. Mach., New York (2018). https://doi.org/10.1145/3267809.3275466
66. Chung, A., Park, J.W., Ganger, G.R.: Stratus: cost-aware container scheduling in the public cloud. In: Proceedings of the ACM Symposium on Cloud Computing, SoCC'18, pp. 121–134. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3267809.3267819
67. Alreshidi, A., Ahmad, A.: Architecting software for the Internet of thing based systems. Future Internet **11**(7) (2019). https://doi.org/10.3390/fi11070153
68. Oasis topology and orchestration specification for cloud applications (tosca). https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. Last seen on August, 2021
69. Rockwell, J., Witherell, P., Femandes, R., Grosse, I., Krishnamurty, S., Wileden, J.: A web-based environment for documentation and sharing of engineering design knowledge. ASME Des. Eng. Tech. Conf. **3**, 671–683 (1970). https://doi.org/10.1115/DETC2008-50086
70. A free, open-source ontology editor and framework for building intelligent systems. https://protege.stanford.edu. Last seen on June 2021
71. Owl - semantic web standards - world wide web consortium. https://www.w3.org/OWL/. Last visited September, 2020
72. Uschold, M.: Demystifying OWL for the Enterprise. Synthesis Lectures on the Semantic Web: Theory and Technology. Springer, Cham (2018). https://doi.org/10.2200/S00824ED1V01Y201801WBE017
73. Sock Shop - a Microservices Demo Application. https://microservices-demo.github.io. Last seen on October 2020
74. Eiter, T., Ianni, G., Polleres, A., Schindlauer, R., Tompits, H.: Reasoning with rules and ontologies. In: Reasoning Web, pp. 93–127. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11837787_4
75. A JavaScript library for building user interfaces. https://reactjs.org. Last seen on August 2021