HOCC:An ontology for holistic description of cluster settings *

Yannis Poulakis[1,2][0000−0001−9336−5700], Georgios Fatouros[1,3][0000−0001−6843−089X], and George Kousiouris[4] and Dimosthenis Kyriazis[1][0000−0001−7019−7214]

[1] University of Piraeus, Piraeus, Greece
{gpoul, dimos}@unipi.gr
[2] Byte Computer, Athens, Greece
[3] Innov-Acts Ltd,Nicosia, Cyprus
gfatouros@innov-acts.com
[4] Harokopio University, Athens, Greece
gkousiou@hua.gr

Abstract. Ontologies have become the de-facto information representation method in the semantic web domain, but recently gained popularity in other domains such as cloud computing. In this context, ontologies enable service discovery, effective comparison and selection of IaaS, PaaS and SaaS offerings and ease the application deployment process by tackling what is known as the vendor lock-in problem. In this paper we propose a novel ontology named HOCC: holistic ontology for effective cluster comparison. The ontology design process is based on four different information categories, namely Performance, SLA, cost and environmental impact. In addition we present our approach for populating, managing and taking advantage of the proposed ontology as developed in a real world Kubernetes cluster setting, as well as instantiating the ontology with example services and data (namely performance aspects of a serverless function).

Keywords: ontology · cloud-computing · semantic-web · Kubernetes · OWL

1 Introduction

The amount of processing power that modern applications require, calls for infrastructures that can handle the workload that is generated, efficiently, in a fault tolerant manner and with respect to their environmental footprint [14]. To that end the cloud and edge computing paradigms have emerged, providing the necessary solutions for resource and service provisioning. Edge specifically, additionally provides the necessary tools to bring processing and storage close to the

* The research leading to the results presented in this paper has received funding from the European Union's funded Project PHYSICS under grant agreement no 101017047.

data sources while also adding resource mobility and low energy consumption[8]. Usually such clusters are formed via the use of Kubernetes [15], an open-source cluster management software that also provides alternative distributions for edge clusters[16].

The plethora of available cluster composition schemata, calls for distinction of the accompanying advantages and disadvantages of each one. This holds especially true in multi-cluster scenarios, where several clusters are managed by a central hub and the options for application or service deployment vary. To tackle this problem, we propose a formal cluster description schema that is based on Web Ontology Language (OWL)[4] and a system that enables automated retrieval of information from Kubernetes-based clusters, transformation according to the ontology definitions into named individuals and storage into a graph database. In essence the main contributions of this paper can be summarized as follows:

- Design of a novel ontology that is based on four different pillars of information (Performance and Technical Properties, SLA Description, Cost Evaluation, Environmental Impact), necessary for a holistic approach in cluster comparison.
- An architecture that enables population, management and information retrieval of the proposed ontology tested against a real world Kubernetes cluster.

In section 2 we present related works in ontologies for cloud computing. Section 3 analyses our ontology design rationale and section 4 demonstrates the ontology population method, evaluation and an example of instantiation. Finally sections 5 concludes our work while also presenting our future work directions.

2 Related Work

One of the earlier works in ontologies that focuses on depicting information on cloud resources was that of moSAIC [12] in which authors utilize ontologies to describe functional and non functional aspects of cloud computing to tackle service negotiation between provider and consumer. In a similar fashion authors of [17] use an OWL-based ontology to address several features and concepts that are directly connected to the cloud computing paradigm while also displaying how this ontology is populated on an Azure Kubernetes cluster through SPARQL queries. CL-Ontology[6] extends mOSAIC by incorporating additional semantics to support interoperability of HPC environments in the cloud.

Service Level Agreement (SLA) document description through a common vocabulary, has also gained attention in the semantics domain. The EU horizon project SLALOM contributed to a formal description of SLA characteristics extending from the 19086-2 standard of the International Organization for Standardization (ISO). In [3] the authors propose an ontology schema that is appropriate for depicting SLA terms and their various properties and relationships and a method of automatic information extraction through SLA public documents via pattern matching.

In [13] and [2] the authors utilize the ontology paradigm to assess the service discovery in the cloud via semantic cloud service representation. In the latter authors also display experimental results when compared to google searches of three different specified queries. In a similar fashion in [10] authors propose an ontology based system that allows for effective queries when searching the necessary cloud resources for the deployment of a user specified job.

The work of Borisova et al [5] displays several examples on how the TOSCA ontology modelling schema can be adapted to describe a variety of Kubernetes specifics. Focusing on the PaaS offerings specifically, authors of [3] define a PaaS marketplace that is based on the ontology information representation schema.

Authors of [9] present challenges that are presented when modelling cloud services with ontologies as emerged from the practical application of the UFO-S ontology. Finally the recent work [1] presents some of current works in the field and addresses new opportunities that arise with the use of ontologies in cloud computing

3 Ontology Design

Ontologies in computer science, provide a structural format in order to represent information about certain entities and their respective properties and relationships mainly through classes, subclasses, object and data properties. In the presented ontology we consider the class *:Cluster* and *:ClusterNode* direct subclasses of *owl:thing* to be cornerstone entities of this ontology. Throughout this section, we use the blank ":" to state terms that are defined in the designed ontology. For a more thorough examination of all entities we suggest visualization of the ontology as uploaded in our github repository [5].

We expect the produced ontology to be populated and utilized in order to compare and therefore select clusters for application deployment in a holistic manner. This realization is the base of our design process and leads us to the distinction of four different pylons that encapsulate necessary aspects for cluster description. Namely these aspects are the following: *(a) Performance and Technical Properties (b) SLA Description, (c) Cost Evaluation, (d) Environmental Impact*. All of the ontology formulation process has been realized with the use of the graphical tool Protégé, a staple in ontology modelling.

3.1 Performance and Technical Properties

First we consider in the ontology design different resource characteristics that will eventually match certain requirements that are derived from the application modelling process. These characteristics display hardware information (GPU enabled nodes, Ram and CPU capabilities, etc.), software information (OS image, Hypervisor, etc.) and generic properties (Location, IP, etc.)

[5] https://github.com/yannispoulakis/HOCC/blob/main/HOCC-Ontology.owl

Typically a cluster's computational capabilities derive from node aspects, thus we couple the *:ClusterNode* class with the appropriate properties. Specifically, nodes are connected with *:VirtualUnit* or *:PhysicalUnit* via the *:isHostedOn* property. The physical unit class aims to cover bare metal nodes in the form of edge nodes or on premises bare-metal computers, while the virtual unit can refer to both private or provided by cloud vendors virtual machines. In addition, the "unit" couple of classes are connected to the *:RawComputationalResource* class which consists the *:Ram, :GPU* and *:CPU* subclasses that include the necessary properties for defining the allocatable respective values. We create a series of object and data properties that link the classes among them and with real values describing their capabilities. In addition we incorporate the *:benchmark* class that aims to describe certain workloads that tested against a cluster to provide a performance indication.

3.2 SLA

One additional aspect we consider in the design process is the depiction of information that is found in service level agreements (SLA), such as the terms and conditions to be met cloud providers as stated in the respective public documents of the services they refer to. Differences in how the cloud providers describe similar SLA terms leads to inconsistencies on how services are compared, monitored and in turn selected for a given task. In the presented ontology we select to describe SLA terms as different classes with a group of properties that depict their respective information. This includes the definition of metrics, their target value and the respective rebate in case of agreement breach.

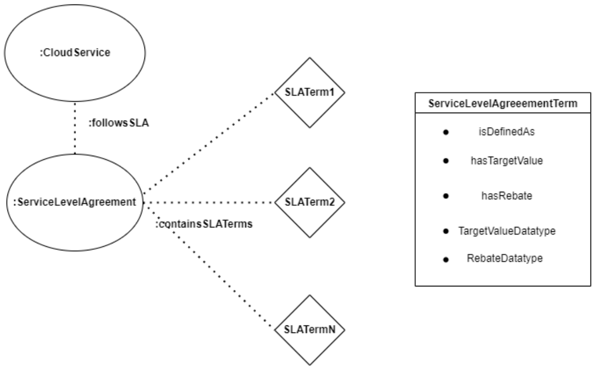

Fig. 1. Ontology formation of the SlA classes and their accompanying data properties.

More specifically we define the parent class *:ServiceLevelAgreement*, bind to *:CloudService*, that is related with terms via the *:hasSLATerm* to the presented terms. Each *:ServiceLevelAgreementTerm* comes with a set of data properties that define target values, type of measurement and definition as presented in Figure 1.

3.3 Cost

A clear overview of cluster expenses that come with setup, maintenance and scaling is a key factor to enable cost optimization. For private and hybrid clouds that utilize bare metal machines as nodes and certain software units, such as specific OS or hypervisors, we think as appropriate to capture the cost per acquisition of both software and hardware necessary for the cluster realization. For clusters that are based fully or partially on cloud provider offerings we declare a set of classes that capture the various pricing models. To that end we define the *physics:CostSchema* class which consists of four different subclasses that represent the basis of different cost schemata. Table 1 gives a formal overview of the defined specialized classes. The parent class *physics:CostSchema* is also assigned certain properties that define the currency used to measure pricing model, namely *physics:withCurrencyValue* and *physics:withCurrencyType* that are inherited to all subclasses and are of type *rdf:string* and *rdfs:decimal* respectively.

Table 1. Subclasses of the specialized CostSchema parent class

Subclasses	Usage
:CostPerAcquisition	*Acquisition cost of hardware and software units.*
:CostPerRequest	*Cost per request. Used in cases such as storage and serverless.*
:CostPerTimePeriod	*Pricing per units of time . Most commonly measured by hour or seconds.*
:CostPerSpecialUnit	*Special case for defining custom units of pricing measurements.*

We consider the case of serverless function offering pricing model, as described in the Lambda service of AWS, as a use case to validate our modelling approach. This type of service, combines charges per request and execution time, thus we opt to allow classes of table 1 to exist jointly. Furthermore the execution time rates vary depending on the amount of memory and ephemeral storage configured for the function to be executed. To that end we assign four different data properties to the *:CostSchema* class, *:withFactor*, *:withFactorValue*, *:CostAlterationOperation* and *:CostAlterationValue*. Additionally both of these charging factors take place after a certain threshold has been exceeded thus we define the data properties *:overThreshold* and *:underThreshold* to define certain lower and upper limits of the respective measurement unit that start the charging process.

3.4 Environmental Impact

Finally we also believe that it is essential to provide a basis for annotating clusters with the appropriate indicators of their environmental footprint. To that end we also include two unique classes, *:EnergySchema* and *:ResourceEfficiency*, in the ontology. The former comes with a set of subclasses, namely *:EnergyUsage*, which is paired with data properties to depict the min, max and average values, *:EnergySource*, *:RenewableEnergy* to declare the percentage of energy that comes from sustainable sources and finally *:PowerUsageEffectiveness* which declare the actual percentage of power that is used for cluster operations. The latter indicates resource efficiency by capturing the time that a cluster is utilized against idle time.

4 Ontology Population and Evaluation

In this section we present our proposed architecture for ontology population and storage as tested in a real world Kubernetes Cluster.

4.1 Ontology Population Pipeline

Figure 2 presents an overview of our proposed architecture, for ontology population from kubernetes clusters.We opt to base our development for compatibility with the Kubernetes ecosystem, due to its widespread adoption, community support and efficiency.A developed pod, which is the Kubernetes unit of deployment, is deployed in each of the available clusters that we aim to extract information from. Afterwards each pod loads the respective cluster configuration through the Kubernetes python client and makes calls to the Kubernetes API to extract the available raw information. Deployment of this pod also includes a set of coupled permissions that are declared according to the Kubernetes Role-based access control (RBAC) method.

Afterwards, with the use of OWLReady2 [6] we process the output of the Kubernetes API calls and define the produced individuals along with their properties, while following the guidelines of the proposed ontology. In certain cases this procedure is straight forward such as checking the computational aspects of a node via the list-nodes command. In other cases we perform a keyword search to implicitly identify information. Finally we serialize this kind of information in the Turtle syntax so it can be stored and further used by semantic reasoners. To accommodate for information that cannot be accessed via the Kubernetes API, we also include in our code a graphical interface for manual creation of individuals.

All of our code is available in our GitHub project repository [7]. It contains both the ontology in a raw XML format and the individuals creation methods wrapped in a python flask service. Additionally the project repository contains the necessary files for docker image creation and cluster deployment.

4.2 Ontology Evaluation

In this section we present two different ontology evaluation measures that have been previously used in the literature [11, 7], namely Relationships Richness

[6] https://readthedocs.org/projects/owlready2/downloads/pdf/latest/

[7] https://github.com/yannispoulakis/HOCC

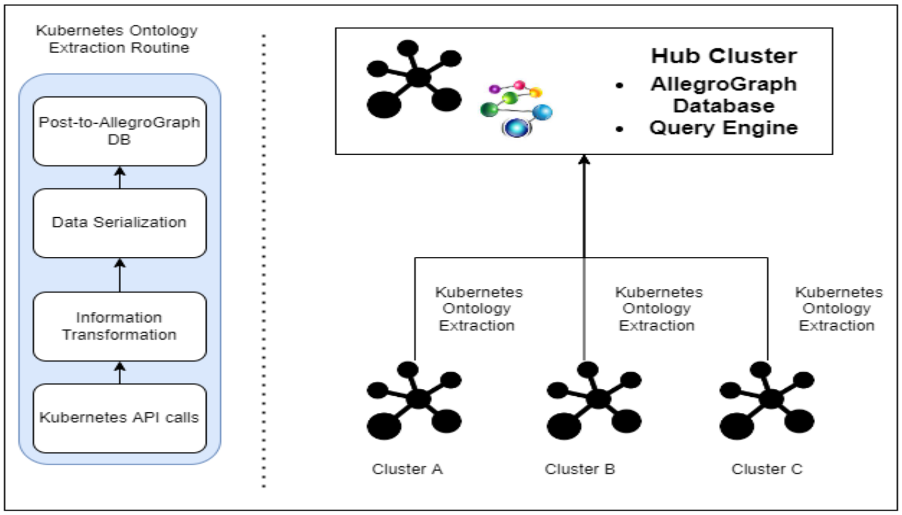
Kubernetes Ontology Extraction Routine
Post-to-AllegroGraph DB
Data Serialization
Information Transformation
Kubernetes API calls
Hub Cluster
AllegroGraph Database
Query Engine
Kubernetes Ontology Extraction
Kubernetes Ontology Extraction
Kubernetes Ontology Extraction
Cluster A
Cluster B
Cluster C

Fig. 2. Proposed pipeline for extracting and managing Kubernetes cluster specific information according to the ontology design.

(RR) and Attributes Richness (AR). Formally, these metrics are calculated by equations 1 and 2. The Relationships Richness metric reflects the diversity of relationships and is the ratio of existing relationships in the ontology divided by the number of subclasses and relationships. We report this value to be 0.4 for our ontology. Additionally the Attributes richness defines how many data properties on average classes have tied to. It is the ratio of all data properties divided by the number of classes and the respective value for our ontology is 0.6578, which is slightly higher than other IT related ontologies (0.65 in [7]). The comparison of the RR indicates a lower value in our case (0.4 compared to 0.55), owed to the fact that the four different categories of consideration (energy, cost, performance and SLA) need to be mapped primarily as subclasses. Unfortunately metrics for the most similar ontologies defined in Section 2 were not available from the respective publications.

$$PR = \frac{|P|}{|SC| + |P|} \tag{1}$$

$$AR = \frac{|ATT|}{|C|} \tag{2}$$

4.3 Ontology Mapping to A Benchmark Scenario

Finally in figure 3 we display the interconnections that are formed by defining a benchmark scenario that results to a certain score. Essentially we define a concept where a serverless function is used for benchmarking reasons and runs on the OpenWhisk platform deployed on ”Cluster1” , a named individual that corresponds to the class *:cluster*.

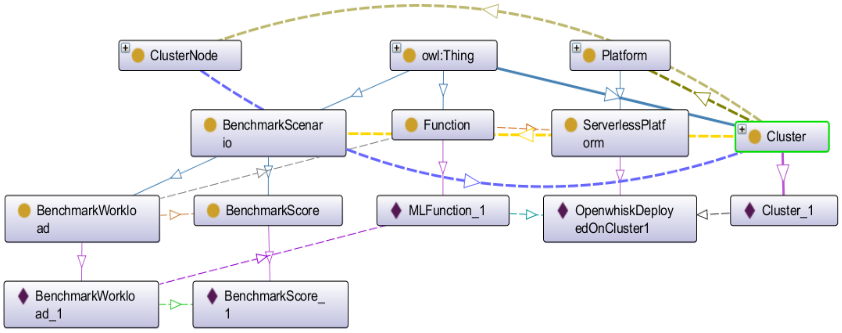

Fig. 3. Example benchmark scenario representation by individuals forming the respective relationships asserted by the ontology design.

5 Conclusions and Future Work

Throughout this work we have presented a novel ontology design for the characterization of cloud computing services and clusters based on four different information categories. We have also showcased our approach for populating the ontology in a real-world scenario that considers Kubernetes formed clusters. We believe the usage of ontologies in cloud and edge computing for multi dimensional characterization of clusters will play a critical role in selecting services and cloud resources in a holistic manner. Future work may include optimizations and extraction methods that may provide further information on cluster hardware, as well as specific software or configuration abilities of a given cluster. These may include for example specific policies followed by the cluster owner, like scheduling or priority capabilities.

References

1. Agbaegbu, J., Arogundade, O.T., Misra, S., Damaševičius, R.: Ontologies in cloud computing—review and future directions. Future Internet **13**(12), 302 (2021)
2. Ali, A., Shamsuddin, S.M., Eassa, F.E.: Ontology-based cloud services representation. Research Journal of Applied Sciences, Engineering and Technology **8**(1), 83–94 (2014)
3. Bassiliades, N., Symeonidis, M., Gouvas, P., Kontopoulos, E., Meditskos, G., Vlahavas, I.P.: Paasport semantic model: An ontology for a platform-as-a-service semantically interoperable marketplace. Data Knowl. Eng. **113**, 81–115 (2018)

4. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A., et al.: Owl web ontology language reference. W3C recommendation **10**(2), 1–53 (2004)
5. Borisova, A., Shvetcova, V., Borisenko, O.: Adaptation of the tosca standard model for the kubernetes container environment. In: 2020 Ivannikov Memorial Workshop (IVMEM). pp. 9–14. IEEE (2020)
6. Castañé, G.G., Xiong, H., Dong, D., Morrison, J.P.: An ontology for heterogeneous resources management interoperability and hpc in the cloud. Future Generation Computer Systems **88**, 373–384 (2018)
7. Hasan, M.M., Kousiouris, G., Anagnostopoulos, D., Stamati, T., Loucopoulos, P., Nikolaidou, M.: CISMET: A Semantic Ontology Framework for Regulatory-Requirements-Compliant Information Systems Development and Its Application in the GDPR Case. International Journal on Semantic Web and Information Systems (IJSWIS) **17**(1), 1–24 (January 2021)
8. Khan, W.Z., Ahmed, E., Hakak, S., Yaqoob, I., Ahmed, A.: Edge computing: A survey. Future Generation Computer Systems **97**, 219–235 (2019)
9. Livieri, B., Guarino, N., Zapattore, M., Guizzardi, G., Bochicchio, M., Longo, A., Nardi, J., Quirino, G., Barcelos, M., Falbo, R.: Ontology-based modeling of cloud services: Challenges and perspectives. In: Proceedings of Short and Doctoral Consortium Papers Presented at the 8th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling (PoEM 2015), Valencia, Spain, November 10-12, 2015. vol. 1497, pp. 61–70. RWTH (2015)
10. Ma, Y.B., Jang, S.H., Lee, J.S.: Ontology-based resource management for cloud computing. In: Nguyen, N.T., Kim, C.G., Janiak, A. (eds.) Intelligent Information and Database Systems. pp. 343–352. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
11. Mazzola, L., Kapahnke, P., Vujic, M., Klusch, M.: Cdm-core: A manufacturing domain ontology in owl2 for production and maintenance. In: KEOD. pp. 136–143 (2016)
12. Moscato, F., Aversa, R., Di Martino, B., Fortiş, T.F., Munteanu, V.: An analysis of mosaic ontology for cloud resources annotation. In: 2011 federated conference on computer science and information systems (FedCSIS). pp. 973–980. IEEE (2011)
13. Rekik, M., Boukadi, K., Ben-Abdallah, H.: Cloud description ontology for service discovery and selection. In: 2015 10th International Joint Conference on Software Technologies (ICSOFT). vol. 1, pp. 1–11. IEEE (2015)
14. Siddik, M.A.B., Shehabi, A., Marston, L.: The environmental footprint of data centers in the united states. Environmental Research Letters **16**(6), 064017 (2021)
15. Thurgood, B., Lennon, R.G.: Cloud computing with kubernetes cluster elastic scaling. In: Proceedings of the 3rd International Conference on Future Networks and Distributed Systems. pp. 1–7 (2019)
16. Xiong, Y., Sun, Y., Xing, L., Huang, Y.: Extend cloud to edge with kubeedge. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC). pp. 373–377. IEEE (2018)
17. Zhang, Q., Haller, A., Wang, Q.: Cocoon: cloud computing ontology for iaas price and performance comparison. In: International semantic web conference. pp. 325–341. Springer (2019)