Deployment Orchestration of Microservices with Geographical Constraints for Edge Computing

Massimo Villari, *Member, IEEE* †, Antonio Celesti, *Member, IEEE* †, Giuseppe Tricomi †, Antonino Galletta, *Member, IEEE* †, Maria Fazio †

† Department of Engineering, University of Messina, Italy - {mvillari, acelesti, gtricomi, angalletta, mfazio}@unime.it

***Abstract*—Nowadays, Edge computing allows to push the application intelligence at the boundaries of a network in order to get high-performance processing closer to both data sources and end-users. In this scenario, the Horizon 2020 BEACON project - enabling federated Cloud-networking - can be used to setup Fog computing environments were applications can be deployed in order to instantiate Edge computing applications. In this paper, we focus on the deployment orchestration of Edge computing distributed services on such fog computing environments. We assume that a distributed service is composed of many microservices. Users, by means of geolocation deployment constrains can select regions in which microservices will be deployed. Specifically, we present an Orchestration Broker that starting from an ad-hoc OpenStack-based Heat Orchestraton Template (HOT) service manifest of an Edge computing distributed service produces several HOT microservice manifests including the the deployment instruction for each involved Fog computing node. Experiments prove the goodness of our approach.**

***Index Terms*—Cloud computing, edge Computing, microservices, deployment, orchestration.**

I. Introduction

Nowadays, Edge computing allows to push the application intelligence at the boundaries of a network in order to get high-performance processing closer to both data sources and end-users. However, the deployment of distributed services and microservices in Edge computing environments raises several issues [1]. In this context, federating virtual Cloud networks [2] represents an interesting approach to arrange environments in which to deploy distributed Edge computing services. In this paper, we present the results of the Horizon 2020 Project BEACON (Enabling Federated Cloud Networking) in terms of service orchestration brokering for deployment of distributed Edge computing services in federated OpenStack-based Cloud networking environments [3]. In such a scenario, we assume that a distributed service is composed of many microservices that can be deployed by different federated Cloud providers in different sites. Users by means of specific manifest files can specify geolocation deployment constrains. In particular, in this scientific work, we present the OpenStack Federation Flow Manager (OSFFM), an Orchestration Broker able to analyze special files that contain geolocation constrains and configurations about of microservice defined by users. OSFFM by means of RESTful API is able to interact with federated OpenStack Clouds orchestrated by means of their own HEAT modules.

We remark that with this paper we intend to extend the Heat Orchestrator Template (HOT) [4], not to define a new standard in distributed microservice deployment. In the proposed system we define two different HOT-based manifests: the distributed service manifest and the microservice manifest. OSFFM analyses the distributed service manifest, extracts the elements that compose the microservice manifest and pass them to the HEAT module of the federated OpenStack Clouds that automatically deploy the requested microservice.

An important feature of this approach is that the Orchestration Broker is able to select target federated Clouds according to their geographic location in order to avoid the data export control problem.

The rest of the paper is organized as follows: Section II describes related works. Section IV presents the Orchestration Broker design. The HOT service manifest is discussed in Section III. Experiments performed on a real OpenStack-based testbed are presented in Section V specifically focusing on HOT service manifest parsing and deployment processing times. Section VI concludes the paper also providing lights to the future.

II. Related Work

Several works are available in literature regarding service orchestration in Cloud computing considering different aspects. In [5], it is proposed a component-oriented method for automated provisioning of Cloud business applications able to manage the whole application's lifecycle. Such a method is based on Cloud orchestration tools that manage the deployment and dependencies of supplied components. In particular, a Linked Unified Service Description Language to describe services for matching user's requirements is proposed. Designing an edge Cloud network implies first determining where to install facilities among the available sites, then assigning sets of access points, while supporting VM orchestration and considering partial user mobility information, as well as the satisfaction of service-level agreements. With this regards, in [6] it is proposed a link-path formulations supported by heuristics to compute solutions in reasonable time. In particular, authors qualify the advantage in considering mobility for both users and VMs as up to 20% less users not satisfied in their SLA with a little increase of opened facilities. In [7] Cloud4IoT is proposed. It is a platform offering automatic deployment, orchestration and dynamic configuration of IoT

978-1-5386-1629-1/17/$31.00 ©2017 IEEE

support software components and data-intensive applications for data processing and analytics, thus enabling plug-andplay integration of new sensor objects and dynamic workload scalability. Cloud4IoT enables the concept of Infrastructure as Code in the IoT context: It empowers IoT operations with the flexibility and elasticity of Cloud services. Furthermore it shifts traditionally centralized Cloud architectures towards a more distributed and decentralized computation paradigm, as required by IoT technologies, bridging the gap between Cloud Computing and IoT ecosystems. In [8] a set of ontologies to semantically represent Cloud resources is proposed. Specifically, three Cloud resource description standards are considered, that are: Topology and Orchestration Specification for Cloud Applications (TOSCA), Open Cloud Computing Interface (OCCI), and Cloud Infrastructure Management Interface (CIMI). The proposed piece of framework promotes the creation of a common semantic knowledge base of Cloud resources described using these different standards. In [9], it is proposed an approach that facilitates the evaluation of different application topologies by enabling Cloud users to easily provision and evaluate different Topology and Orchestration Specification for Cloud Applications (TOSCA) options based on performance and cost metrics. In particular, it is shown the technical feasibility of the approach based on a case study with the WordPress blogging application where various topologies were automatically provisioned and evaluated.

Regarding, the orchestration of networking services, in [10], a control orchestration protocol is proposed. It provides a transport application programming interface solution for joint Cloud/network orchestration, allowing interworking of heterogeneous control planes to provide provisioning and recovery of quality of service (QoS)-Aware end-To-end services. In the field of future 5G network applications, in [11], it is proposed a SDN/NFV orchestrator that automatically serves mobile network operators (MNOs) capacity requests by computing and allocating virtual backhaul tenants. Such backhaul tenants are built over a common physical aggregation network, formed by heterogeneous technologies (e.g., packet and optical) that may be owned by different infrastructure providers.

In the context of orchestration of energy-aware services, in [12], a novel energy-aware resource orchestration framework for distributed Cloud infrastructures is introduced, in order to manage both network and IT resources in a typical optical backbone. A high-level overview of the system architecture is provided by focusing on the definition of the different layers of the whole infrastructure, and introducing the Path Computation Element, which is the key component of the proposed architecture.

III. HOT Service Manifest for the Deployment of Edge Computing Services

In this Section, we discuss about of data exchange files adopted in the proposed system. In particular we created two different manifest files that extend the OpenStack Heat Orchestration Template (HOT): distributed service manifest and microservice manifest. HOT as well as our manifests is based on a markup language widely adopt for data serialization: YAML. It is interoperable and open, contains both structural information and data. YAML is human-friendly and simplifies the interaction among modern programming languages. The main goal of the language is to minimize the amount of structural characters and to represent in a natural and meaningful way data. All flavors of data structures realized by YAML can be adequately represented with three basic primitives: mappings (hashes/dictionaries), sequences (arrays/lists) and scalars (strings/numbers). The specification provides a complete language to serialize any native data structure through the exploitation of such primitives also providing simple typing and aliasing mechanisms.

A. Heat Orchestration Template

HOT introduced in OpenStack "Grizzly" release, was designed to replace the Heat CloudFormation-compatible format (CFN). It could be divided into five sections:

1) preliminary information section;
2) parameter groups section (introduced after the OpenStack Juno release);
3) parameter section;
4) resource section;
5) output section.

HOT is continuously improved by the Heat core team. All releases are backward compatible. HOT is composed of several resources that can be grouped in logical element named stacks.

B. Service Manifests

In order to enable OSFFM to deploy distributed Edge computing services, in the proposed system, we created two different service manifest:

- the distributed service manifest;
- the microservice manifest.

Microservice manifest is fully compliant with HOT, instead in distributed service manifest we introduced three entities:

1) service placement policies according to location constraints;
2) location-aware elasticity rules;
3) network reachability rules.

We mapped these entities as resources, in this way it is possible to compose a complex manifest simply nesting them. Specifically, for activities related to location-aware elasticity rules and service placement policies, we created two resources:

- **OS::Beacon::Georeferenced_deploy**. Defined as array of GeoJSON MultiPolygon. It used to store geographic references. An example is shown in the Listings 1;
- **OS::Beacon::ServiceGroupManagement**. Used for activities related to location-aware elasticity rules and service placement policies. In particular main fields are *resource* that contains its resources and *geo_deploy* that stores OS::Beacon::Georeferenced_deploy elements. Listing 2 shows an example.

IV. Architectural Design

In order to instantiate Edge computing distributed services, we designed and implemented a federation management component called OpenStack Federation Flow Manager (OSFFM).

```
geoshape_exportControl:
type: OS::Beacon::Georeferenced_deploy
properties:
label: Shape label
description: description
shapes: [{"type":"Feature","id":"ITA","properties":{"name":"Italy"}, "geometry":
{"type":"MultiPolygon", "coordinates":[[[[15.520376,38.231155], [9.214818,39.240473],...]]]}}]
```

Listing 1: Example of OS::Beacon::Georeferenced_deploy resource.

```
federation:
type: OS::Beacon::ServiceGroupManagement
properties:
name: GroupName
geo_deploy: { get_resource: geoshape_exportControl}
resource:
groups: {get_resource: microservice1}
```

Listing 2: Example of OS::Beacon::ServiceGroupManagement resource.

OSFFM is the Orchestrator Broker of the H2020 project "BEACON". In the BEACON project we consider to have several OpenStack Clouds that make each others a-priori agreements in order to share resources. The purpose of the OSFFM is to interact with OpenStack federated Clouds in order to manage the deployment of distributed applications.

In the design phase we made the following assumptions:

- The infrastructure is composed of twin Clouds. This means that Virtual Machine (VM) images, users, networks, security groups, key pairs, as well as other configurations are replicated in each Cloud, in this way the system guarantees uniformity in the management phase.
- OpenStack Clouds interact with an external element the Federation Agent that is able to create OVN tunnel among them as described in [3].

In Fig. 1 is depicted the logical architecture of the proposed system.

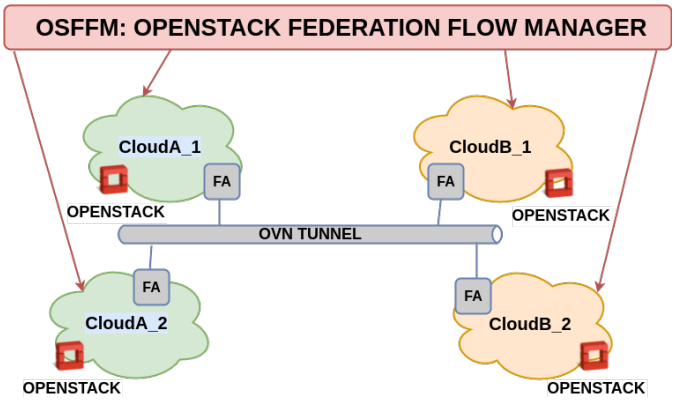

Fig. 1: Overall high-level federation architecture.

As showed in the picture, the OSFFM, by means of ad-hoc APIs, is able to interact with federated Clouds in order to manage virtual machines and to create virtual networks.OSFFM produces a huge amount of data both for monitoring and management stuff. These data must stored and supervised. For this purposes, in our system we included MongoDB, a document-based database Big-Data oriented. Each federated OpenStack Cloud runs a VM that contains a software component named Federation Agent (FA). The FA by means of the OpenvSwitch functionalities is able to create virtual networks and virtual paths among federated Clouds. The internal stucture of the OSFFM is showed in Fig 2.

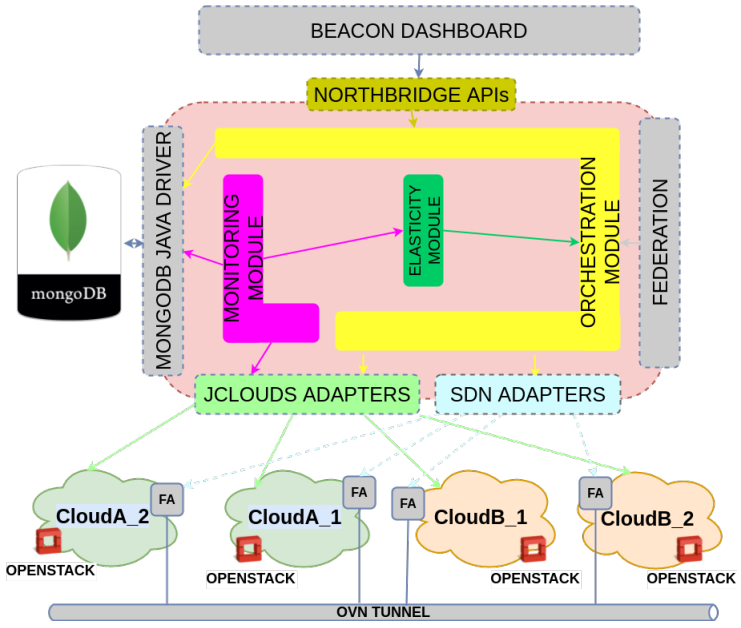

Fig. 2: OSFFM system Architecture.

The OSFFM exposes four APIs in order to interact with other components:

1) **Southbridge**. It is used to send commands to federated OpenStack Clouds. It includes two different kind of adapters: jClouds adapters and SDN adapters. jClouds adapters are used in order to interact with OpenStack modules. They allow to create, start, stop, resume and retrieve information about the virtual instances. The SDN adapters implement client RESTful APIs able to

interact with the FA machine in order to create and destroy virtual tunnel.
2) By means of **Eastbridge** APIs, OSFFM sends and receives instructions for the virtual network management. In particular by means of these APIs OSFFM is able to communicate with others OSFFM instances.
3) External users interact with the OSFFM through the **Northbridge** APIs. In particular they can make RESTful requests directly or by means of an ad-hoc deployed dashboard.
4) Management and monitoring of virtual resources produces a lot of data that OSFFM have to manage. In order to accomplish that we introduced the **Westbridge** API. It, by means of MongoDB Java driver is able to interact with the system database.

The OSFFM is composed by three specific modules:

1) OSFFM_ORC: Orchestrator module.
2) OSFFM_MON: Monitoring module.
3) OSFFM_ELA: Elasticity Location Aware module.

These modules are able to interact each others directly but, in order to in order to communicate with external component use the above exposed APIs.

1) OSFFM_ORC: This module is able to execute all steps required for deploying a distributed microservice.It receives from the Northbridge API the distributed service manifest, split it into microservice manifests and store them, by means of Westbridge API, inside specific collections into MongoDB. At the same time, OSFFM by means of Southbridge API, in particular using jClouds adapters, sends microservice manifests to the Heat module of selected federated Clouds.

Specifically, the Orchestrator module execute different steps.

- The first one consists to processes geographical constrains along with virtual resource informations retrieved from OpenStack components;
- The second one is to enable the communication between NorthBridge and SouthBridge;
- The third one is to coordinate the activities of all OSFFM Orchestrator modules.

The OSFFM Orchestrator module, in order to speed up the elaboration time, stores into in RAM data-structures such as hash-tables the distributed microservice manifests.

OSFFM Orchestrator module execute a lot of methods for the distributed microservice deployment. The main are:

- *ManifestInstantiation*. It is executed in order to create an instance of the ManifestManager thread.
- *DeployManifest*. It is used in order to deploy the same microservice manifest in the Clouds involved in the federation. We remark that in order to provide fault tolerance features the same microservice will be deployed in all federated Clouds.
- *SendShutSignalStack4DeployAction*. This method is invoked in order to shoutdown all replicas of deployed machine by the DeployManifest method.
- *ManagementGeoPolygon & ManagementRetrieveCredential*. These methods are used to retrieve informations from MongoDB. In particular the first one it is used in order to retrieve informations about of datacenter present in a provided Geoshape. Instead, the second one it is executed to retrieve user credentials.

2) OSFFM_ELA: This module is designed to control and to maintain the performance of the application deployed in the federation. This goal is achieved providing the functions required to horizontally scale of a resource in the federation. This module interacts with the target Cloud when a particular condition occurs so as to trigger a specific action. By interfacing this component with the monitoring one, it is possible to have the parameters needed to verify both the Cloud infrastructure and VMs internal state. When the previous information is correlated with other Clouds information the module is able to make the required scalability decisions. The OSFFM_ELA interacts directly with OSFFM_ORC to accomplish the decision made and interacts with OSFFM_MON to receive status notifications about monitored condition.

3) OSFFM_MON: This module is designed as a collector of the various monitoring flows coming from several monitoring components inside the federated Clouds. The OSFFM_MON is interconnected with the Clouds via the SouthBridge interface and makes requests in order to discover information needed to monitor state instantiated via stacks. For OSFFM architecture, the monitoring solution is realized by adapting the monitoring module used by OpenStack, Ceilometer, at a federation level and interconnected with the federated Ceilometer creating a hierarchical structure.

V. Experiments

In this Section, considering a prototype of the OSFFM Orchestrator module developed in JAVA. We conducted four different analysis

1) Distributed Service Manifest Split;
2) Parsing;
3) Distributed Service Manifest Recomposition;
4) Deployment.

For our tests we consider three different scenarios. For each of them we create a different distributed service manifests with an increasing degree of complexity. In particular, we focused on resources belonging in following categories:

- OS::Beacon::Georeferenced_deploy
- OS::Beacon::ServiceGroupManagement
- OS::Nova::Server

The first distributed service manifest (Scenario A) describes 1 service group composed by 1 instance of a Cirros VM that represents the microservice. The manifest also specifies 1 georeferenced area about where the the service group has to be instantiated. The second distributed service manifest (Scenario B) describes 2 service groups composed by 3 instances of a Cirros VM that represent the microservices. The manifest also specifies 2 georeferenced areas where the the service group has to be instantiated. The third distributed service manifest (Scenario C) describes 3 service groups composed by 5 instances of a Cirros VM that represent the microservices. The manifest also specifies 2 georeferenced areas where the the service group has to be instantiated.

Our prototype was deployed on a Dell PowerEdge R530 server rack with an Intel Xeon E5-2650 v.3 processor(2,30GHz, 20 cores), 64GB of RAM, 2 HDDs of 20GB each, a QNAP NAS with 7 HDDs (3 TB each).Experiments were repeated 30 times in order to consider mean values, standard deviation and 95% confidence interval.

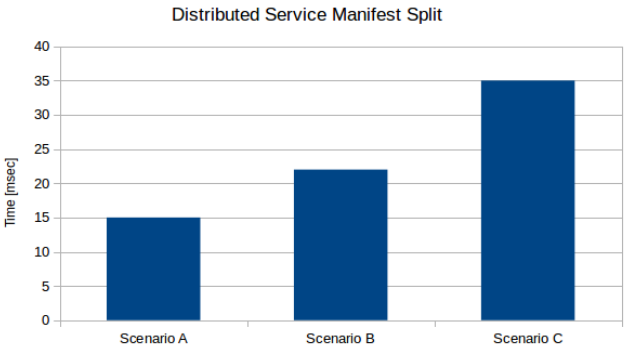

Fig. 3: Deployment time.

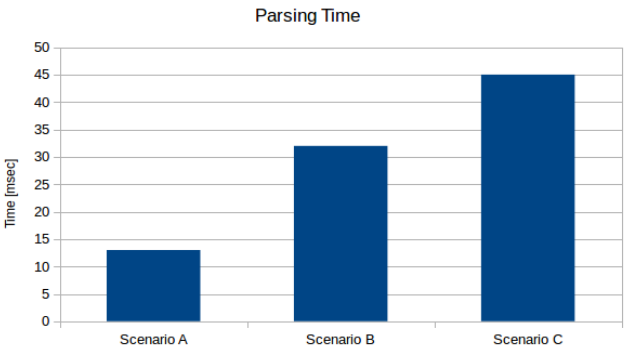

Fig. 4: Parsing time.

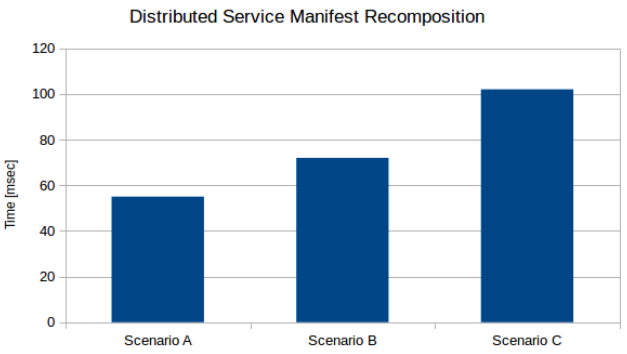

Fig. 5: Deployment time.

How we can observe in the histogram depicted in Fig 3, shows the execution time to split a distributed service manifest in the different scenarios. OSFFM to accomplish this task present very low processing times. How we can observe in the histogram depicted in Figure 4, shows the parsing tasks performed by the OSFFM Orchestrator on the three distributed service manifests present low processing times. In fact the processing time gap between the first configuration and the third one is only 32ms. How we can observe in the histogram depicted in Figure 5 the recomposition tasks performed by the OSFFM Orchestrator on the three distributed service manifests very interesting process time. In fact the time gap between Scenario A and C is about 40 msecs.

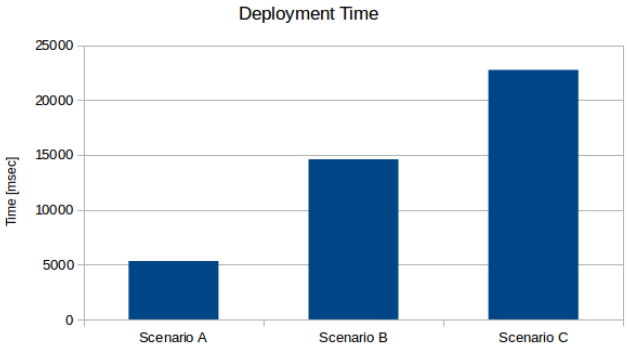

Fig. 6: Deployment time.

How we can observe in the histogram depicted in Figure 6 the deployment tasks performed by the OSFFM Orchestrator on the three distributes service manifests present considerable processing times. In fact the processing time of the third configuration is roughly 22s with a gap of roughly 17s compared to the first configuration. However, even in this case the processing times are acceptable.

VI. Conclusion and Future Work

In this paper, considering the European H2020 BEACON project, we focused on orchestration brokering for the deployment of Edge computing distribute services. The approach described drives the Edge computing service deployment in a federated Cloud networking environment via a HOT service manifest that is analysed by the the Orchestration Broker that automatically extracts all elements describing how related microservices have to be deployed in federated Clouds via their HEAT modules. An important feature of this approach is the ability to select target federated Cloud as function of the geographic position of federated Clouds that allows deploy microservices on the edges of a networks. In fact, a borrower, using the described approach, can select exactly the geographical area where a microservice have to be deployed by the Orchestration Broker.

In future works, we will focus on the maintenance tasks performed by the OSFFM_ELA and OSFFM_MON modules on the virtual resources in which a distributed service is deployed.

Acknowledgment

This research was supported by the European Union's Horizon 2020 Research and Innovation Programme Project BEACON under Grant Agreement No 644048.

REFERENCES

[1] M. Fazio, A. Celesti, R. Ranjan, C. Liu, L. Chen, and M. Villari, "Open issues in scheduling microservices in the cloud," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 81–88, 2016.

[2] R. Moreno-Vozmediano, E. Huedo, I. M. Llorente, R. S. Montero, P. Massonet, M. Villari, G. Merlino, A. Celesti, A. Levin, L. Schour, C. Vázquez, J. Melis, S. Spahr, and D. Whigham, *BEACON: A Cloud Network Federation Framework*. Cham: Springer International Publishing, 2016, pp. 325–337.

[3] A. Celesti, A. Levin, P. Massonet, L. Schour, and M. Villari, *Federated Networking Services in Multiple OpenStack Clouds*. Cham: Springer International Publishing, 2016, pp. 338–352. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-33313-7_26

[4] Heat Orchestration Template (HOT) specification, http://docs.openstack.org/developer/heat/template_guide/hot_spec.html.

[5] H. Benfenatki, C. Ferreira Da Silva, G. Kemp, A.-N. Benharkat, P. Ghodous, and Z. Maamar, "Madona: a method for automated provisioning of cloud-based component-oriented business applications," *Service Oriented Computing and Applications*, vol. 11, no. 1, pp. 87–100, 2017.

[6] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, 2017.

[7] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Dupont, D. Charalampos, F. De Pellegrini, F. Antonelli, and S. Cretti, "Cloud4iot: A heterogeneous, distributed and autonomic cloud platform for the iot," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2017, pp. 476–479.

[8] K. Yongsiriwit, M. Sellami, and W. Gaaloul, "A semantic framework supporting cloud resource descriptions interoperability," in *IEEE International Conference on Cloud Computing, CLOUD*, 2017, pp. 585–592.

[9] A. Sampaio, T. Rolim, N. Mendonça, and M. Cunha, "An approach for evaluating cloud application topologies based on tosca," in *IEEE International Conference on Cloud Computing, CLOUD*, 2017, pp. 407–414.

[10] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, R. Martinez, M. Moreolo, J. Fabrega, A. Aguado, S. Yan, D. Simeonidou, J. Gran, V. Lopez, P. Kaczmarek, R. Szwedowski, T. Szyrkowiec, A. Autenrieth, N. Yoshikane, T. Tsuritani, I. Morita, M. Shiraiwa, N. Wada, M. Nishihara, T. Tanaka, T. Takahara, J. Rasmussen, Y. Yoshida, and K.-I. Kitayama, "Control orchestration protocol: Unified transport api for distributed cloud and network orchestration," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A216–A222, 2017.

[11] R. Martinez, A. Mayoral, R. Vilalta, R. Casellas, R. Munoz, S. Pachnicke, T. Szyrkowiec, and A. Autenrieth, "Integrated sdn/nfv orchestration for the dynamic deployment of mobile virtual backhaul networks over a multilayer (packet/optical) aggregation infrastructure," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A135–A142, 2017.

[12] G. Fioccola, P. Donadio, R. Canonico, and G. Ventre, "Dynamic routing and virtual machine consolidation in green clouds," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, 2017, pp. 590–595.