applied sciences

MDPI

Article

TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform

Dongmin Kim [1], Hanif Muhammad [1], Eunsam Kim [2], Sumi Helal [3] and Choonhwa Lee [1,*]

[1] Division of Computer Science and Engineering, Hanyang University, Seoul 133-791, Korea; alkga@hanyang.ac.kr (D.K.); sad143007@yahoo.com (H.M.)
[2] Department of Computer Engineering, Hongik University, Seoul 121-791, Korea; eskim@hongik.ac.kr
[3] School of Computing and Communications, Lancaster University, Lancaster LA1 4WA, UK; s.helal@lancaster.ac.uk
* Correspondence: lee@hanyang.ac.kr; Tel.: +82-2-2220-1268

Received: 28 November 2018; Accepted: 2 January 2019; Published: 7 January 2019

Abstract: Kubernetes, a container orchestration tool for automatically installing and managing Docker containers, has recently begun to support a federation function of multiple Docker container clusters. This technology, called Kubernetes Federation, allows developers to increase the responsiveness and reliability of their applications by distributing and federating container clusters to multiple service areas of cloud service providers. However, it is still a daunting task to manually manage federated container clusters across all the service areas or to maintain the entire topology of cloud applications at a glance. This research work proposes a method to automatically form and monitor Kubernetes Federation, given application topology descriptions in TOSCA (Topology and Orchestration Specification for Cloud Applications), by extending the orchestration tool that automatizes the modeling and instantiation of cloud applications. It also demonstrates the successful federation of the clusters according to the TOSCA specifications and verifies the auto-scaling capability of the configured system through a scenario in which the servers of a sample application are deployed and federated.

Keywords: auto-scaling; cloud computing; Docker; cloud orchestration; cloud federation; TOSCA (Topology and Orchestration Specification for Cloud Applications)

1. Introduction

In recent years, organizations that have made the transition from building and managing their own computing facility to cloud computing have been benefiting from maximized capacity and cost-efficiency [1]. It is further known that using a "container" approach, which separates component tasks into small individual processes rather than installing an entire application on each virtual machine, has various advantages, such as using computational resources efficiently and enabling finer-grained deployments [2,3]. The cloud computing service providers (hereafter "cloud providers") such as Google, Microsoft, and Amazon, support container-based virtualization [4], and container orchestration tools for automating the distribution and management of containerized applications are being offered as well. Technologies like containerization and other IT-enabled dynamics capabilities tend to provision the evolutionary fitness of these organizations through agility boost regarding market capitalization and operational adjustment, improving competitive performance [5]. The cloud orchestration market is estimated to be projected to cross US$ 20 billion by 2025 with a sales revenue expected to register a compound annual growth rate of 14.6% according to online market research [6].

One advantage of using a container orchestration tool to form container clusters is the ability to resize the cluster system at runtime automatically. This function automatically adjusts the number

of containers in response to real-time changes in workloads, thereby reducing the incurred cost by allowing an efficient allocation of computational resources on an on-demand basis [7,8]. Furthermore, deploying a container cluster federation across multiple service areas allows cloud providers to improve reliability and responsiveness [9]. However, this is still considered to be a complicated process; As container clusters are increasingly being deployed and federated over multiple services from different cloud providers [10], managing and monitoring cloud applications across the entire service areas of an organization is a big challenge. When it comes to a means of modeling cloud applications, there exist several prominent alternatives. We have chosen Topology and Orchestration Specification for Cloud Applications (TOSCA) as our modeling language for cloud applications over other options which include CAMEL (Cloud Application Modelling and Execution Language), CAML (Cloud Application Modelling Lanuage), CloudML (Could Modelling Language), and container platform-native DSLs(Domain-Specific Languages) [11]. The reason for this is that we wanted to demonstrate a multi-cloud orchestration solution first for mainstream cloud technology and platforms. Hence, we use the Organization for the Advancement of Structured Information Standards (OASIS) standard, TOSCA, for specifying cloud application topology in a declarative way [12].

This research work proposes a solution to the orchestration problem by introducing new add-on features to the tool that automates the modeling and orchestration of the applications to be deployed on cloud services. Our solution provides the federation and monitoring functions across different cloud service areas. By demonstrating the feasibility of multi-cloud orchestration of cloud services for popular cloud platforms, our work is expected to touch off further research developments in the cloud research community.

The remainder of this paper is organized as follows. Section 2 surveys various technologies and tools that pertain to application containerization and examines the orchestration automation tool to be used for cloud computing in this study. Section 3 examines the TOSCA-based cloud orchestration system for the container cluster federation. Section 4 demonstrates the operation of the proposed system and verifies its auto-scaling ability through a scenario in which web game servers are federated. Section 5 discusses prominent studies related to our approach, and Section 6 summarizes the result of this research work and proposes possible themes for further research.

2. Containerized Service Orchestration Technology

2.1. Docker and Kubernetes

Configuring a cloud service requires the installation of various components and programs. For instance, configuring a web server requires an operating system, web daemon, and a database server. The conventional method of installing constituting parts on actual machines limits the computing resources to a particular service execution and does not allow the resources to be shared by different services.

Docker [13] containerizes individual processes and allows them to be run within a separated lightweight execution environment called a container. As the adoption of Docker has increased, the need to automatize the deployment and management of containerized applications has recently arisen [14]. Developed in this context, Kubernetes [15] is an open-source container orchestration tool that automatically installs and manages a cluster of Docker containers. The service developer can create Docker images containing desired service elements and Kubernetes can deploy and manage the components and their relationships. Kubernetes includes the following elements:

- **Kubernetes pod**: this is an essential building block of Kubernetes, usually containing multiple Docker containers.
- **Kubernetes node**: this represents a VM (Virtual Machine) or physical machine where the Kubernetes pods are run.
- **Kubernetes cluster**: this consists of a set of worker nodes that cooperate to run applications as a single unit. Its master node coordinates all activities within the cluster.

- **Kubernetes Federation**: this is a cluster of clusters, i.e., viewed as a backbone cluster that combines multiple Kubernetes clusters. For example, when one Kubernetes cluster is running on Google Cloud Platform in Tokyo, Japan, and another is running in Oregon, U.S., one Kubernetes federation might be configured such that if there is a problem in the Oregon platform, the Tokyo cluster would be able to take over the share of the faulty platform, thereby increasing the resiliency of the service. Figure 1 shows an example of the Kubernetes Federation architecture. Kubernetes provides a flexible, loosely coupled mechanism for service delivery. The federation application program interface (API) server interacts with the cluster through the federation controller manager. The master is responsible for exposing the API, scheduling the deployments, and overall cluster management. The interaction with the Kubernetes cluster is done through the federation controller manager using the federation API server.

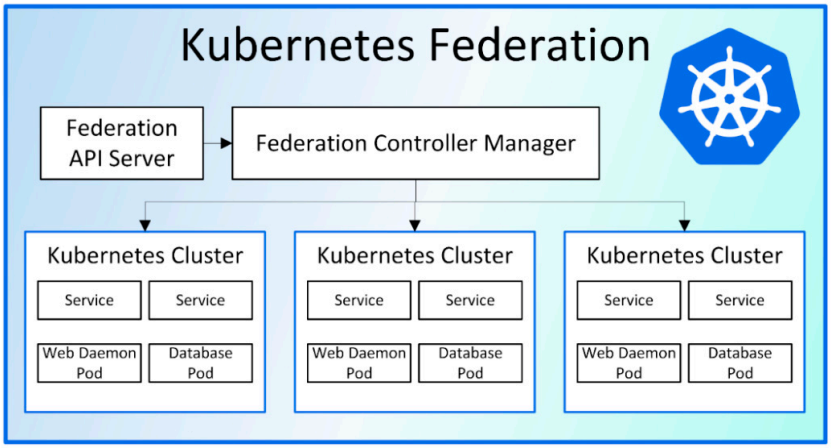

Figure 1. Example of the Kubernetes federation architecture. The federation application program interface (API) server automatically manages the subordinate Kubernetes clusters by providing the API that is equivalent to the Kubernetes API.

2.2. Orchestration Tools Supporting Topology and Orchestration Specification for Cloud Applications (TOSCA)

Organizations seek cloud providers that support the functions required for their services at a lower price. However, once a service is initiated with a particularcloud provider, it is difficult for the organization to switch to a different provider [16,17]. As a solution to this problem, the OASIS, the international nonprofit standard organization, has recently introduced a standard to enable the portability of cloud-based applications, called TOSCA [12,18]. The TOSCA template defines the topology of an entire application by formulating a directional graph with the node and relationship templates.

- The node template defines the components of the cloud-based applications. The node type is specified to express the characteristics and the available functions of the service component.
- The relationship template defines the relations between the components. The relationship type is specified to express the relationship characteristics between the components.

These TOSCA templates are used to describe the components and inter-component relationships of cloud applications in a declarative way so that the topology descriptions can be instantiated and deployed on a particular cloud platform later on. Hence, the application portability.

Several cloud orchestration tools are capable of supporting the TOSCA standard. Brogi et al. [19] introduced the SeaClouds platform that can manage the service-based applications across different cloud providers. Alexander et al. [20] proposed TOSCAMP (TOSCA + CAMP (Cloud Application Management for Platforms)) that supports the entire orchestration process, from modeling a cloud-based application to its deployment, by an integration of TOSCA that conducts topology

modeling and CAMP that performs management and deployment of applications. Studies pertaining to the performance assessment of TOSCA-based cloud applications [21] and the cost-aware deployment and management of entire cloud services are also being conducted [22].

For the validation of the proposed system, this study uses Cloudify [23], which has a feature called Cloudify plug-in that is relatively development-friendly for the extension of functions. Also, Cloudify has recently released Kubernetes plug-in that allows Kubernetes clusters to be included as cloud service components [24]. It should be noted that even though this research work uses Cloudify to develop the content further, the system proposed in this research article can use any orchestration solutions as long as it supports the TOSCA standards.

3. Federation Frameworks of Containerized Services

3.1. Overall Architecture

Figure 2 shows a sample cloud service scenario devised to highlight the problem that our research work in this article seeks to resolve. First, the cloud service is established using container clusters using the configuration and system specification provided by the administrator and user. These system specifications are by standards, default configuration, and user-defined customization. Then, the clusters are deployed across the cloud provider's different service areas (such as Northeast Asia, North America, and West Europe in our use case scenario) and federated to increase the reliability and responsiveness of the cloud service. The objective of the TOSCA-based cloud orchestration system proposed in this research article is to automate this entire orchestration process using a cloud orchestration tool and to support its operation monitoring in the future as well. Therefore, the primary focus of our system design was to support the following functions:

1. Automation of the distribution and federation of container clusters by defining the Kubernetes cluster federation in the TOSCA description of the application, receiving the information of the Kubernetes clusters, and executing "join Kubernetes federation".
2. Automation of the service status management by defining Kubernetes horizontal pod auto-scaler (HPA) information with regard to the TOSCA description of the cloud application and enabling its operation in the cloud orchestration tool.
3. Enabling the identification and monitoring of the entire service topology of the application by allowing the cloud orchestration tool to access the information of the Kubernetes components.

Figure 3 outlines the proposed system architecture that realizes these three functions. The orchestration system receives relevant component services and pod information through YAML scripts. The orchestration system is based on TOSCA standard descriptions, which makes it easy to receive the Kubernetes clusters information, and eventually, join them to the federation. The HPA (horizontal pod autoscaler) scales up and scales down the number of pods in the entire Kubernetes federation in an automatic manner. The monitoring agent plug-in allows the orchestration system to monitor the status of each component at all times.

Key component interactions, especially between the Orchestration System and Kubernetes Federation, are marked as (i), (ii), and (iii) in the figure. Actions and interactions performed by each of the interfaces are discussed below.

(i) Defining a Kubernetes federation according to the TOSCA standard makes it easy to communicate Kubernetes clusters information and join them to the federation. Input TOSCA descriptions contain new federation and cluster components that are backed up by the corresponding federation and cluster types we introduced in Kubernetes plugins. For an association between a cluster and its intended federation, Kubernetes plugin module first makes a connection to the federation and then executes "kuberfed join" command to make the cluster join the federation.

(ii) The system also supports Kubernetes HPA, which auto-scales the number of pods across the entire Kubernetes federation. The HPA component type is defined to be associated with the

K8s API Mapper so that AutoscalingV1Api requests can be sent to the Kubernetes Federation. V1HorizontalPodAutoscaler and V1DeleteOptionsinputs payloads are transported for HPA creation and deletion, respectively.

(iii) Furthermore, the monitoring agent allows the system to monitor the status of the Kubernetes components. A monitoring agent installed on each pod allows direct monitoring access to individual pods. Shell Runner module is introduced to our system architecture to support the installation of the Diamond monitoring program on the pod via the kubectl shell command execution.

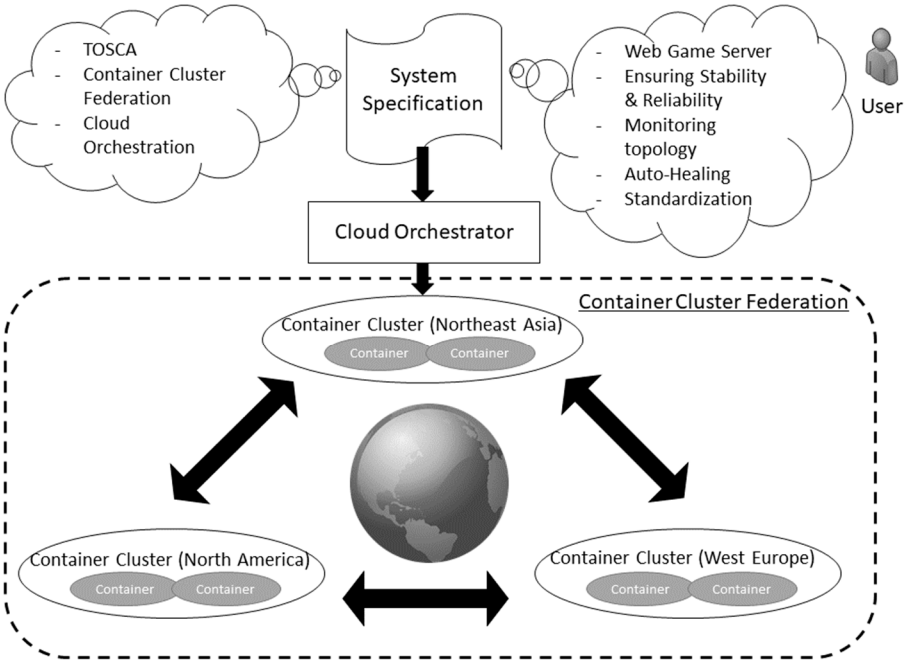

Figure 2. Use of cloud orchestration tool for the federation of container clusters in terms of web game server configuration. Using the tool increases the resiliency and reliability of the service, and allows the monitoring of the entire topology. TOSCA: Topology and Orchestration Specification for Cloud Applications.

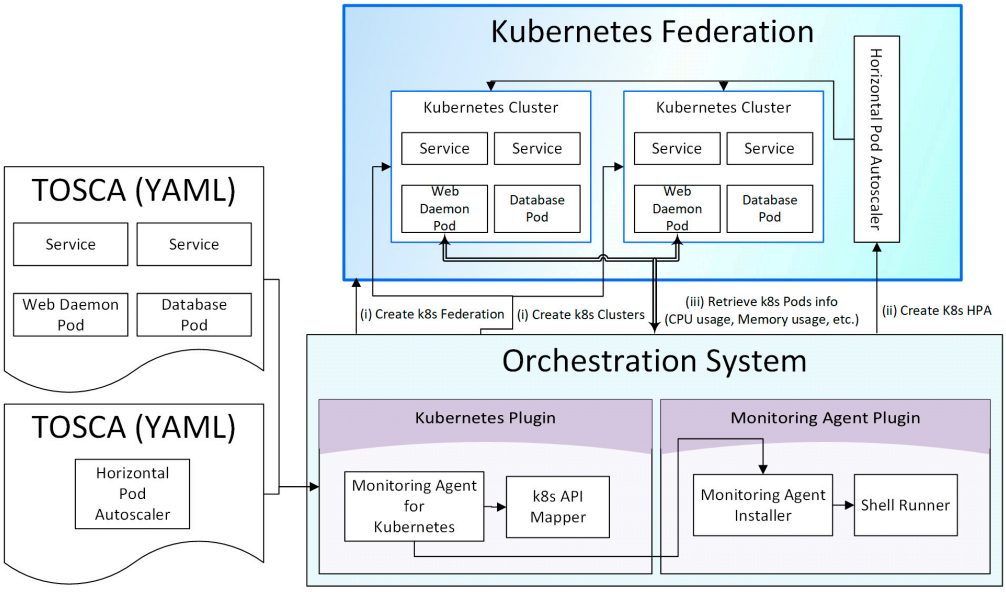

Figure 3. Proposed cloud orchestration system architecture for container cluster federation.

3.2. Implementation of Kubernetes Federation Cluster Configuration

To express the relationship between individual Kubernetes clusters and Kubernetes Federation components in the TOSCA topology, the proposed system defined the TOSCA node and relationship templates as YAML-based Cloudify plug-ins in the following manner (as Cloudify Kubernetes plugin: plugin.yaml):

- The Kubernetes cluster template is implemented as a component containing the Kubernetes cluster information that exists in the cloud provider's service when creating the topology of applications.
- The Kubernetes Federation template is implemented as a component using the information of the Kubernetes cluster with the Kubernetes Federation control plane installed.
- Kubernetes cluster template and Kubernetes Federation template are defined to permit master/slave configuration ("managed_by_master").

The Kubernetes cluster components connected to the Kubernetes federation components are automatically generated and deleted during the orchestration lifecycle. This setup is implemented automatically by executing a series of commands that enable the Kubernetes cluster template to be affiliated to the Kubernetes Federation when the template is implemented as a component (as Cloudify Kubernetes plugin: cluster_create FUNCTION of cloudify_kubernetes/tasks.py). The deletion of the Kubernetes cluster components is accomplished similarlyy (as the Cloudify Kubernetes plugin: cluster_delete FUNCTION of cloudify_kubernetes/tasks.py Module). As a result of this setup, the developer can automate the distribution and federation of the Kubernetes clusters by defining the clusters and federation in TOSCA. Furthermore, if the topology of the application written for a single Kubernetes cluster is changed to be deployed over a Kubernetes Federation, the application components are automatically redistributed and federated to individual Kubernetes clusters, thereby providing a useful means to increase the reliability and responsiveness of an organizations' cloud service.

3.3. Definition of Horizontal Pod Autoscaler (HPA) Components

Using Kubernetes HPA enables an automatic adjustment of the number of pods according to the workload of the service. Kubernetes administrators can set the minimum and maximum of the number of pods by providing an HPA option.

HPA can be extended to generate the pods for Kubernetes Federation as well in the same way. The number of pods to increase or decrease is communicated to each Kubernetes clusters. To use an HPA, the proposed system used a YAML-based definition for the TOSCA node template and included it with the Cloudify plug-in (as Cloudify Kubernetes plugin: plugin.yaml).

3.4. Monitoring the Information of Kubernetes Components

Some information of the Kubernetes components cannot be determined when their creation is requested and can only be obtained after some time has elapsed. For example, a Kubernetes service component that provides Kubernetes components with an IP address has the IP value of null at the time of its creation. About 30 seconds to a minute later, they may be given an IP address that can be read from the service component. A new and efficient method for receiving the information has been devised for the situations that require an active retrieval of information from Kubernetes components (as Cloudify Kubernetes plugin: resource_read Function of cloudify_kubernetes/tasks.py Module).

Moreover, Cloudify, the cloud orchestration tool used in this work, continually monitors the application components by remotely installing Cloudify Agent on the smallest computing unit that constitutes the application topology. However, Kubernetes does not recommend a SSH connection to individual pods in virtual computing machines environment [25]. Kubernetes instead allows connecting to the pods via "kubectl exec" command (direct access to the container) that can send commands to the individual pods. The system proposed in this study has defined a new method that automatically completes the *kubectl* commands using the name of the pod in question and the

Kubernetes config to install Cloudify Agent (as Cloudify Kubernetes plugin: make_conn_cmd Function of the directory cloudify_kubernetes/cloudify_agent/installer/operations.py Module).

4. Evaluation

4.1. Environment Setup for Development and Performance Verification

A simple web game server scenario has been devised to prove the operability of the proposed system and to verify its auto-scaling ability. It is required to develop a game server in this scenario that offers the game "Pacman" worldwide online. It assumes that the developer has defined a game server for a single Kubernetes cluster in a TOSCA template.

Figure 4 illustrates such an execution environment setup across multiple clusters. User interactions with the game are being handled by the load balancer to automatically add or remove web pods in both the Kubernetes clusters in Tokyo, Japan, and Oregon, US according to the workload and number of online users in the system. Figure 5 presents the skeleton code of the corresponding TOSCA descriptions of the game server federation. In the figure, Kubernetes Federation and cluster components are defined at element 1 through element 3, and HorizontalPodAutoscaler is defined as a TOSCA Node at element 6.

Cloudify Manager 4.2 (Cloudify, New York, NY, USA, 2017) was used to run in a virtual machine configured with CentOS 7 x64 as its operating system hosted by VirtualBox in the Ubuntu 16.04 LTS x64 environment. Kubernetes 1.8 was used run on Google Cloud Platform, and Kubernetes clusters were built in the Tokyo, Japan and Oregon, US areas. Each cluster contains two nodes; the node type is n1-standard-2 (vCPU 2, RAM 7.5 GB), and Nginx is used as Web daemon, and MongoDB as the database server by default. Each node has a web pod, database pod, and a persistent data volume. Each web pod and DB pod have assigned a unique IP address through which it can be accessed accordingly. The number of nodes is later scaled up or down according to the incoming traffic.

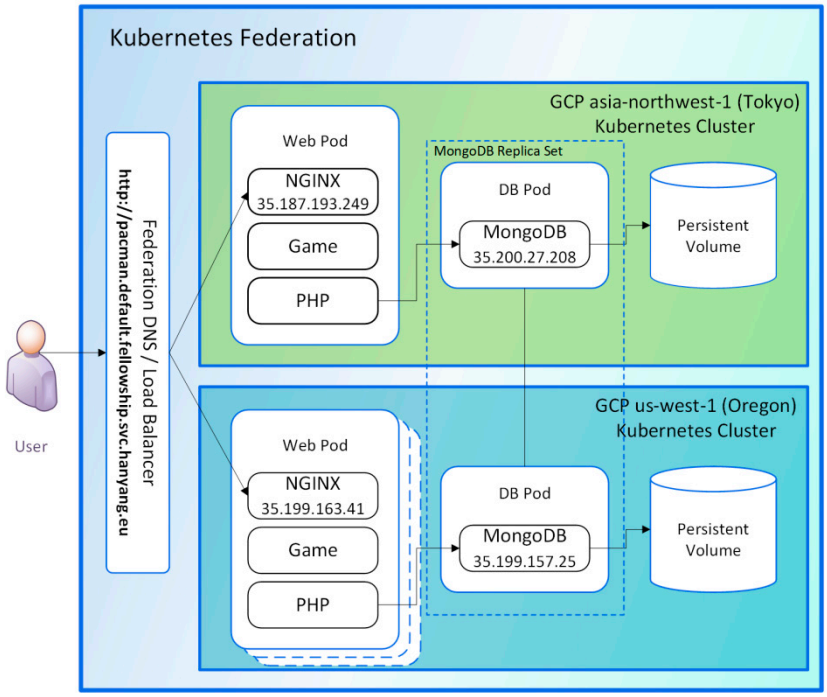

Figure 4. Web game server scenario to demonstrate the operation of the proposed system architecture and verifying its auto-scaling ability. GCP: Google Cloud Platform.

```
# <Web game server scenario example>

inputs:
  k8s_federation_conf: # (1-a)
        - cluster:
            server: https://3.3.3.3
          name: fellowship
  # (skipped)
  k8s_cluster_conf_1: # (2-a)
        - cluster:
            server: https://1.1.1.1
          name: gke_cloudify-kubefed_asia-northeast1-a_cluster1
  # (skipped)
  k8s_cluster_conf_2: # (3-a)
        - cluster:
            server: https://2.2.2.2
          name: gke_cloudify-kubefed_us-west1-a_cluster2
  # (skipped)

node_templates:
  mongo_storage_class_1: # (skipped)  # (7-a)
  mongo_storage_class_2: # (skipped)  # (7-b)
  mongo_pv_claim_1: # (skipped)  # (8-a)
  mongo_pv_claim_2: # (skipped)  # (8-b)
  mongo_svc: # (skipped)  # (9)
  mongo_rs: # (skipped)  # (10)
```

```
k8s_federation:  # (1-b)
  properties:
    configuration:
      file_content: { get_input: k8s_federation_conf }
k8s_cluster_1:  # (2-b)
  properties:
    configuration:
      file_content: { get_input: k8s_cluster_conf_1 }
k8s_cluster_2:  # (3-b)
  properties:
    configuration:
      file_content: { get_input: k8s_cluster_conf_2 }
pacman_svc:  # (4)
  # (skipped)
  relationships:
    - type: cloudify.kubernetes.relationships.managed_by_master
      target: k8s_federation
pacman_rs:  # (5)
  # (skipped)
  relationships:
    - type: cloudify.kubernetes.relationships.managed_by_master
      target: k8s_federation
pacman_hpa:  # (6)
  type: cloudify.kubernetes.resources.HorizontalPodAutoscaler
  # (skipped)
  relationships:
    - type: cloudify.kubernetes.relationships.managed_by_master
      target: k8s_federation
```

Figure 5. Skeleton code of the Topology and Orchestration Specification for Cloud Applications (TOSCA) definitions corresponding to the Web game server scenario.

4.2. Kubernetes Federation by TOSCA

Cloudify Manager automatically creates a federation and installs the Pacman server, once the TOSCA template describing the Kubernetes federation is loaded to distribute. The topology map of the entire application can be found in the Cloudify Manager's web UI upon the completion of the installation. The Cloudify Agent being automatically installed during the distribution process of each pod periodically reports the status of the pods. This enables real-time monitoring of any problems of specific components in the topology map as shown in Figure 6. The figure shows the web UI of the Cloudify manager after the web game server configuration and cluster federation are completed. The Kubernetes clusters, Mongo DB, and Pacman modules are all connected to the Mongo storage accordingly.

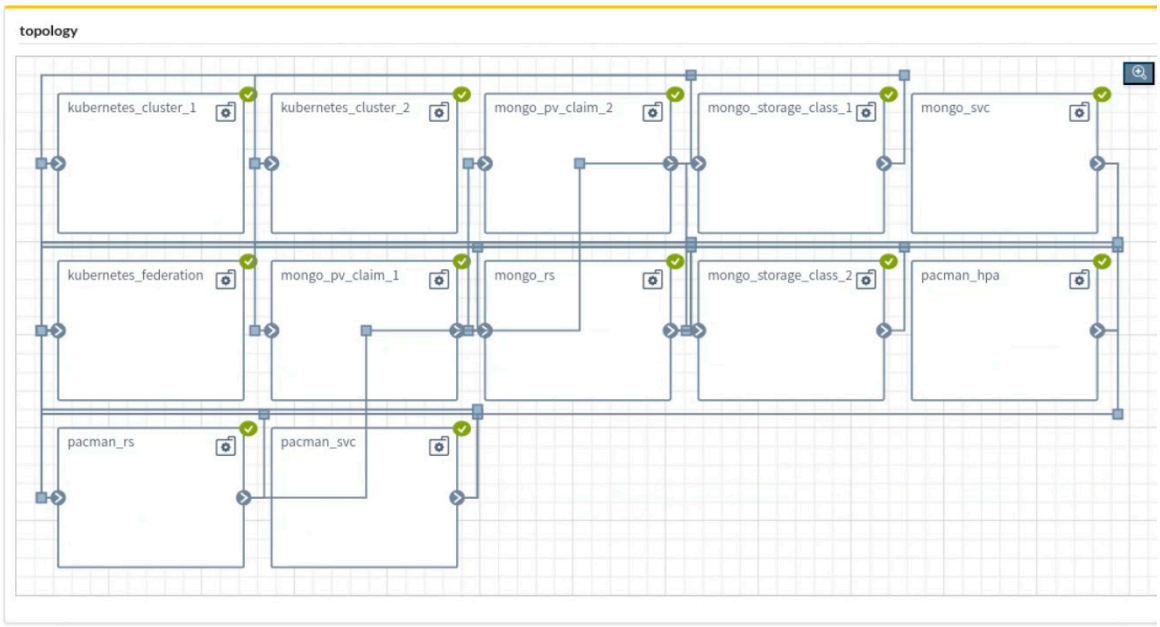

Figure 6. Web UI of the Cloudify Manager after the Web game server configuration and cluster federation is complete.

More detailed monitoring of the status of each pod can be displayed by a graph, as shown in Figure 7. The system information sent from the pods and the transmission cycle can be modified by defining the settings for Diamond daemon on the TOSCA template. As the workload of the web daemon pod increases, the load is distributed in the cluster consequently. Figure 7 shows the pod status change, when input traffic to the Oregon cluster was manually generated by sending 15 requests 10,000 times to the server's web page using the Apache HTTP server benchmarking tool (using ab –n 10,000 –c 15 <URL> command). The "cpu_total_user" shows the percentage of the processes executed in user mode over the entire CPU core. The "loadavg_01" and "loadavg_05" show the number of average processes that are on standby for execution for one and five minutes, respectively. As the pod has two virtual cores, the values exceeding 2.00 are an indication of the processes in the queue.

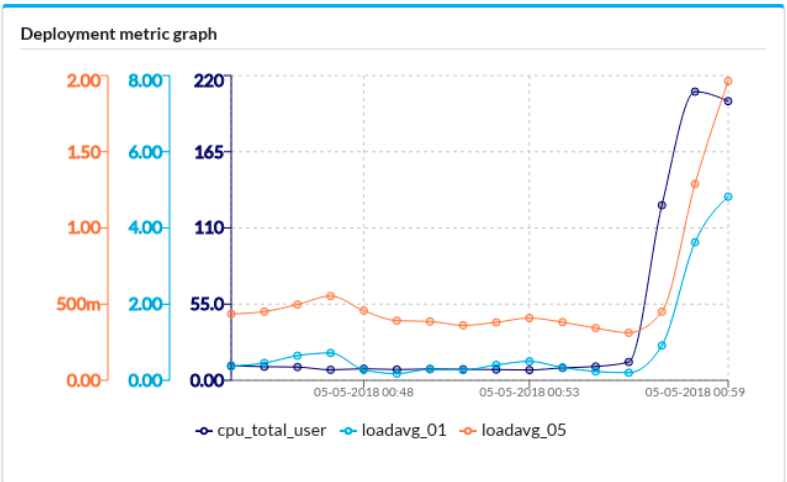

Figure 7. Example of the pod status monitoring using the web UI of Cloudify Manager: diagram shows the situation in which the workload of a Web daemon pod in a specific area increases dramatically. "cpu_total_user" shows the percentage of the processes executed in user mode for the entire central processing unit (CPU) core. As the pod node has two CPU cores, CPU usage can rise to a maximum of 200%. Moreover, "loadavg_01" and "loadavg_05" show the number of average processes that are on standby for execution for one and five minutes, respectively.

4.3. Federated Auto-Scaling by TOSCA

As the Kubernetes clusters are federated, the game service is automatically provided by a cluster in the other service area in the event of a sudden spike in the workload. The computing power must be increased by adding more pods to the available node pool, if necessary. Figure 8 shows the effect of pod auto-scaling under the forced load increase situation, as described in Section 4.2. Firstly, in the normal operation status of the application, the number of nodes in the Tokyo cluster is one. There is also a single pod in the Oregon cluster as shown in the figure. After a sudden increase in incoming client requests to the cluster, the system automatically adjusts the number of Pacman pods in the Oregon cluster to handle the workload surge smoothly, while maintaining the system performance. The graph compares CPU usage in the federated clusters under normal and heavy load cases. As the input grows beyond the capacity, the federated HPA kicks in to add more pods to the Oregon cluster. In the experimental run, up to four pods are allocated to distribute the load increase among them, which is indicated in the case of "Heavy Load with HPA". The "Heavy Load" case represents a single pod case for the same load. It is noted that the target CPU usage for HPA is set to 500 millicores in the experiment.

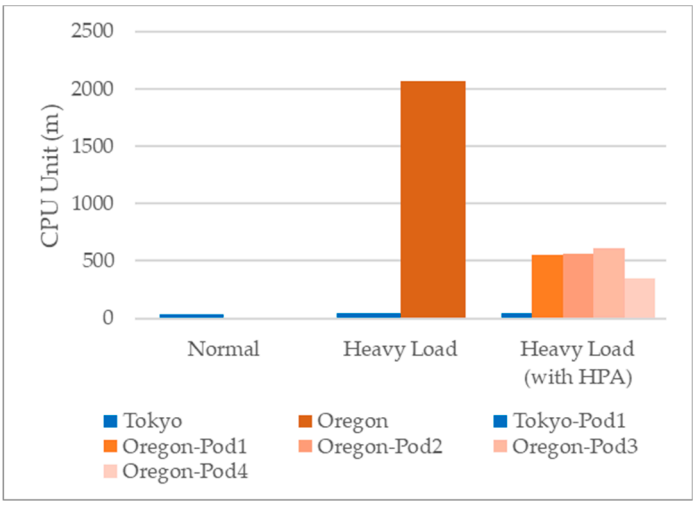

Figure 8. Auto-scaling and load balancing among Kubernetes clusters.

5. Related Research

The necessity to federate cloud services has been discussed over the past several years [9,26,27]. However, existing studies are limited to the scope that roughly indicates the direction for future research. In contrast, this work has explored and demonstrated the feasibility of federated cloud services by building a working prototype system based on relevant standards and mainstream technologies; this paper proposes an architectural design of container-based cloud orchestration system that can scale out to multiple clusters, as the system load increases. It also presents our validation efforts to conduct a performance verification study by deploying and running a sample cloud service in an actual cloud environment.

The problem of multi-cloud orchestration support has been investigated for a while [9,20,26–28]. As argued in recent research, there might be a need for a look from a different angle [11,29]; the problem might be more effectively tackled when considering cloud application portability, multi-cloud interoperability, and elastic runtime adaptation altogether at the same time. According to the proposed approach, the multi-cloud problem can be divided into two sub-problems of elastic platform definition and cloud application definition [11]. Support for infrastructure-awareness of elastic container platforms provides an execution foundation for cloud-native applications translated from their universal definition to a particular format targeting a specific container platform. Therefore, the end result of their research is to enable cloud applications to migrate to a different cloud service provider at runtime. It is noted that we also aim for a similar research goal, that is, application topology-based multi-cloud orchestration support on top of container platforms. However, the difference is that our primary goal is to corroborate the integration of the trio targeting for a specific container platform, i.e., Kubernetes, whereas their work provides a more generic multi-cloud migration solution that can accommodate heterogeneous container platforms including Kubernetes, Docker Swarm, and Apache Mesos. There are a few other differences worth mentioning. First, runtime migration to different container platforms is not our primary focus. Also, when it comes to the way to define application blueprints, they rejected TOSCA to define a new, lightweight DSL. It contrasts our design decision which was made in favor of TOSCA, because the language is considered as a mature standard for cloud application definitions.

The BEACON framework proposed by Moreno-Vozmediano et al. [30] pertains to a networking solution for the federation of the cloud network; thus, it is different from this work, because the emphasis in our case is placed on enabling federation in the service level. The research work of Tricomi et al. [28] introduced a multi-component application development structure that used TOSCA expressions to orchestrate the deployment of various cloud applications in OpenStack across multiple federated cloud providers. This differs from ours in that we present an approach and corresponding

architectural design that can reap the benefits of application portability from TOSCA-based declarative topology descriptions and performance gains from container-based fine-grained compositions. Villegas et al. [27] thought that the problem could be solved by collecting and stacking the cloud service federation vertically that are classified into the software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS) cloud services. Our work proposed a more concrete solution to the same problem, using a container-based method for federating the clusters, to keep abreast of recent advances in the relevant technologies.

A model-driven framework can be used to connect a platform-independent cloud model of services with cloud-specific operations [31]. Cloud management tools were used to deliver auto-scaling deployment across multiple clouds using automated model-to-configuration transformation. This is different from our work because we do not use the model transformation approach, with our proposed orchestration architecture centered on the idea of federating container clusters using a TOSCA-based cloud orchestration tool. We showed that the container clusters can be automatically distributed and federated to the service areas of a cloud provider. Our approach enables the federation at the service level and has the competence to be portable and declarative.

Our evaluation study of the proposed scheme is designed to use a web game server case, which adequately shows the efficacy of it to deal with varying loads. However, it should also be noted that a follow-up validation of our orchestration system, which involves some benchmark applications being widely used within the microservice research community [32], should provide further assessment of our approach. Such an effort should reveal the strengths and weaknesses of our proposal compared to other approaches to multi-cloud service orchestration.

6. Conclusions

This paper proposed an architectural design and its prototype implementation that federates Kubernetes clusters using a TOSCA-based cloud orchestration tool. By using the prototype, it was verified that container clusters could be automatically distributed and federated to the service areas of a cloud service provider.

The primary contribution of this work lies in its TOSCA-based orchestration architecture that allows the federation of the container clusters within the service areas of an individual cloud provider as well as across that of different cloud providers. It can also achieve efficient utilization of the cluster computing resources, as the federation of the container-based clusters enables them to be deployed dynamically at the granularity of micro-services. Our validation efforts considered a single cloud provider case only for federated cloud services, which is currently the predominant form of container cluster federation. However, it is noted that our approach does not constrain us to the case of multiple cloud providers; Deploying and auto-scaling the federation of clusters across different cloud providers, using the cloud orchestration tool, will realize a federation with better reliability and availability.

This study helps us realize that containers cannot only be seen as an alternative to VMs at the infrastructure level, but they are also an application packaging mechanism relevant to platform- and software-as-a-service offerings. Providing cloud application management based on lightweight virtualization, the container technology positively impacts on both the development and deployment aspects such as testing and monitoring of industrial containerized applications. The system proposed in this research article allows active auto-scaling using Kubernetes federation HPA. As a follow-up to this research in this article, we propose research on a scheme that actively changes the auto-scaling policy based on the monitoring data of application components obtained from the cloud orchestration system.

Author Contributions: C.L. and S.H. conceived the proposed orchestration system architecture for container cluster federation; D.K., H.M., and E.K. implemented the proof-of-concept system and performed the validation tests; D.K. and E.K. analyzed the performance data and wrote the paper.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2017R1A2B4010395).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.H.; Konwinski, A.; Lee, G.; Patterson, D.A.; Rabkin, A.; Stoica, I.; et al. *Above the Clouds: A Berkeley View of Cloud Computing*; EECS Department, University of California: Berkeley, CA, USA, 2009.
2. Hoenisch, P.; Weber, I.; Schulte, S.; Zhu, L.; Fekete, A. Four-fold auto-scaling on a contemporary deployment platform using docker containers. In Proceedings of the Lecture Notes in Computer Science, Goa, India, 16–19 November 2015; pp. 316–323. [CrossRef]
3. Li, W.; Kanso, A. Comparing Containers versus Virtual Machines for Achieving High Availability. In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E), Tempe, AZ, USA, 9–13 March 2015; pp. 353–358. [CrossRef]
4. Gerber, A. The State of Containers and the Docker Ecosystem 2015. Available online: https://www.oreilly.com/webops-perf/free/state-of-docker-2015.csp (accessed on 6 January 2019).
5. Mikalef, P.; Pateli, A. Information technology-enabled dynamic capabilities and their indirect effect on competitive performance: Findings from PLS-SEM and fsQCA. *J. Bus. Res.* **2017**, *70*, 1–16. [CrossRef]
6. Persistence Market Research: Cloud Orchestration Market. Available online: https://www.persistencemarketresearch.com/market-research/cloud-orchestration-market.asp (accessed on 6 January 2019).
7. Wu, Q. *Making Facebook's Software Infrastructure More Energy Efficient with Auto-Scale*; Technical Report; Facebook Inc.: Cambridge, MA, USA, 2014.
8. Kouki, Y.; Ledoux, T. SCAling: SLA-driven Cloud Auto-scaling. In Proceedings of the 28th ACM Symposium on Applied Computing, Coimbra, Portugal, 18–22 March 2013; pp. 411–414. [CrossRef]
9. Grozev, N.; Buyya, R. Inter-Cloud architectures and application brokering: Taxonomy and survey. *Softw. Pract. Exp.* **2014**, *44*, 369–390. [CrossRef]
10. Liu, C.; Loo, B.T.; Mao, Y. Declarative automated cloud resource orchestration. In Proceedings of the Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11), Cascais, Portugal, 26–28 October 2011; p. 26. [CrossRef]
11. Quint, P.-C.; Kratzke, N. Towards a Lightweight Multi-Cloud DSL for Elastic and Transferable Cloud-native Applications. In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER), Madeira, Portugal, 19–21 March 2018; pp. 400–408. [CrossRef]
12. OASIS, T. Topology and Orchestration Specification for Cloud Applications Version 1.0. *Organ. Advacement Struct. Inf. Stand.* **2013**. Available online: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (accessed on 6 January 2019).
13. Docker—Build, Ship, and Run any App, Anywhere. Available online: https://www.docker.com (accessed on 6 January 2019).
14. Paraiso, F.; Challita, S.; Al-Dhuraibi, Y.; Merle, P. Model-driven management of docker containers. In Proceedings of the IEEE International Conference on Cloud Computing, San Francisco, CA, USA, 27 June–2 July 2017. [CrossRef]
15. Kubernetes. Available online: https://kubernetes.io (accessed on 6 January 2019).
16. 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.* **2016**, *5*, 4. [CrossRef]
17. Nikolaos, L. D1.1 Requirements Analysis Report. Cloud4SOA Project Deliverable. Available online: https://pdfs.semanticscholar.org/20fb/57b26982a404138a32ff756e73d26c29a6f2.pdf (accessed on 6 January 2019).
18. TOSCA Simple Profile in YAML Version 1.0. Available online: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/os/TOSCA-Simple-Profile-YAML-v1.0-os.html (accessed on 6 January 2019).
19. Brogi, A.; Ibrahim, A.; Soldani, J.; Carrasco, J.; Cubo, J.; Pimentel, E.; D'Andria, F. SeaClouds: A European Project on Seamless Management of Multi-cloud Applications. *SIGSOFT Softw. Eng. Notes* **2014**, *39*, 1–4. [CrossRef]
20. Alexander, K.; Lee, C.; Kim, E.; Helal, S. Enabling end-to-end orchestration of multi-cloud applications. *IEEE Access* **2017**, *5*, 18862–18875. [CrossRef]
21. Sampaio, A.; Rolim, T.; Mendonça, N.C.; Cunha, M. An approach for evaluating cloud application topologies based on TOSCA. In Proceedings of the IEEE International Conference on Cloud Computing, San Francisco, CA, USA, 27 June–2 July 2016; pp. 407–414. [CrossRef]

22. Alexander, K.; Lee, C. Policy-based, cost-aware cloud application orchestration. In Proceedings of the International Conference on Software & Smart Convergence, Vladivostok, Russia, 27 June–1 July 2017.
23. Cloudify Cloud & NFV Orchestration Based on TOSCA. Available online: https://cloudify.co (accessed on 7 January 2019).
24. Kubernetes Plugin. Available online: http://docs.getcloudify.org/4.1.0/plugins/kubernetes (accessed on 6 January 2019).
25. Jérôme, P. If You Run SSHD in Your Docker Containers, You're Doing It Wrong! Available online: https://jpetazzo.github.io/2014/06/23/docker-ssh-considered-evil (accessed on 6 January 2019).
26. Murudi, V.; Kumar, K.M.; Kumar, D.S. Multi Data Center Cloud Cluster Federation—Major Challenges & Emerging Solutions. In Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets, Karnataka, India, 19–20 October 2017. [CrossRef]
27. Villegas, D.; Bobroff, N.; Rodero, I.; Delgado, J.; Liu, Y.; Devarakonda, A.; Fong, L.; Masoud Sadjadi, S.; Parashar, M. Cloud federation in a layered service model. *J. Comput. Syst. Sci.* **2012**. [CrossRef]
28. Tricomi, G.; Panarello, A.; Merlino, G.; Longo, F.; Bruneo, D.; Puliafito, A. Orchestrated Multi-Cloud Application Deployment in OpenStack with TOSCA. In Proceedings of the IEEE International Conference on Smart Computing, Hong Kong, China, 29–31 May 2017; pp. 1–6. [CrossRef]
29. Kratzke, N. About the Complexity to Transfer Cloud Applications at Runtime and How Container Platforms Can Contribute? In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER), Porto, Portugal, 24–26 April 2017; pp. 19–45. [CrossRef]
30. Moreno-Vozmediano, R.; Huedo, E.; Llorente, I.M.; Montero, R.S.; Massonet, P.; Villari, M.; Merlino, G.; Celesti, A.; Levin, A.; Schour, L.; et al. BEACON: A cloud network federation framework. *J. Comput. Syst. Sci.* **2016**, 325–337. [CrossRef]
31. Alipour, H.; Liu, Y. Model Driven Deployment of Auto-Scaling Services on Multiple Clouds. In Proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C), Seattle, WA, USA, 30 April–4 May 2018; pp. 93–96. [CrossRef]
32. Aderaldo, C.M.; Mendonça, N.C.; Pahl, C.; Jamshidi, P. Benchmark Requirements for Microservices Architecture Research. In Proceedings of the IEEE/ACM International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), Buenos Aires, Argentina, 20–28 May 2017; pp. 8–13. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).