Architecture for Orchestrating Containers in Cloud Federations

Yodit Gebrealif[1](✉), Mohammed Mubarkoot[1], Jörn Altmann[1](✉), and Bernhard Egger[2]

[1] Technology Management, Economics and Policy Program, Seoul, South Korea
mubarkoot@snu.ac.kr, jorn.altmann@acm.org
[2] Department of Computer Science and Engineering, Seoul National University, Seoul, South Korea
bernhard@csap.snu.ac.kr

Abstract. Containerization technology helps achieving not only better portability and interoperability but also better performance and efficiency on various cloud computing arrangements. Such technology is expected to empower cloud federations by enhancing portability and scalability across the federation. In this paper, we propose an architecture by adding two subcomponents to the NIST reference architecture for identifying resources and managing container orchestration in cloud federation environments. The architecture adds two subcomponents to the NIST reference architecture. The proposed two new sub-components enable resource identification and container orchestration across cloud federation members. These names of the two subcomponents are the Resource Identifier and the Container Orchestrator, respectively. The Resource Identifier component identifies the appropriate federated member for allocating tasks based on previous experience and current status. The Container Orchestrator facilitates the management and orchestration of containers at the federation level. We also identified several techniques, which can be used for resource identification. Among those, linear regression technique is selected for resource provisioning and identification of federation members. Further, these techniques are also expected to learn from log files from previous executions and prioritize resources based on the current resource status and previous experience.

Keywords: Container orchestration · Distributed cloud federation · Artificial Intelligence (AI) · Resource identification · Linear regression technique

1 Introduction

Cloud federations extend the use of cloud resources beyond a provider's resources and enhance collaboration between cloud service providers (CSP) based on a service level agreement (SLA) [1, 2]. This helps addressing issues related to scalability, interoperability, and maximization of resource utilization [3]. Implementation arrangements of cloud federations can include centralized and peer-to-peer models. In all arrangements, the

K. Tserpes et al. (Eds.): GECON 2021, LNCS 13072, pp. 66–75, 2021.
https://doi.org/10.1007/978-3-030-92916-9_6

federation manager (FM) is the key component that handles such collaboration between federation members [1, 4].

One of the key challenges that cloud federation aims at addressing is the scalability of resources. A certain CSP might experience a lack of resources, while, at the same time, other CSPs possess extra resources that are not fully utilized [5, 6]. We believe that orchestrating resources between CSPs can be a game-changer in cloud federations. Among resource orchestration frameworks that exist in the market (Table 1), only the MiCADO framework [7] deals with container orchestration. However, MiCADO only targets multi-cloud environments and not cloud federations. The main difference between a multi-cloud and a federated cloud is that the latter is collaborative in nature, while, in a multi-cloud, the relationships between the CSPs are independent [1]. Other differences also include service composition and user-provider relationship [8].

Containerization technology can efficiently enhance scalability, security, and governance of orchestrated resources [7]. Containers are resource-efficient units, since they require minimum resources to run. A container image encapsulates all requirements and dependencies of an application into a packaged unit [4, 7, 8]. This makes them highly scalable and portable [7]. While managing containers is usually the task of the container orchestrator (e.g., Kubernetes and Docker Swarm [7]), these orchestrators work within the resources of a CSP and cannot extend their functionality to other CSPs. As a result, this restricts the full utilization of containerization technology at the federation level. Furthermore, implementing container orchestration at the federation level has an impact on improving the portability and scalability of applications. Moreover, enhancing that with machine learning capabilities can efficiently optimize resource identification and prioritization. In this paper, we propose an architecture that utilizes containerization technology for cloud federations and allows resource identification and container orchestrating across a federation.

The remainder of this paper is structured as follows: Sect. 2 introduces the existing state of art cloud resource orchestration frameworks. Section 3 presents the overall proposed architecture, and Sect. 4 discusses the new components and how they work in detail. Finally, Sect. 5 presents the conclusion and future work.

2 State-of-the-Art

There are many resource orchestration frameworks in the market (Table 1). While most of these frameworks target multi-cloud environments, only MiCADO [7] deals with container orchestration, though only within the context of multi-cloud environments.

Table 1. Cloud resource orchestrators available in the market, based on [7].

Name	Description
Heat	Heat launches multiple composite cloud applications in the form of text files (via templates) that are treated like a code
Cloudify	Cloudify provides integrated infrastructure with Ansible, Kubernetes, AWS Cloud Formation, Azure ARM, and Terraform
Brooklyn	This framework for cloud orchestration allows deployment and management of applications via declarative blueprints
Startos	Startos of polyglot PaaS provides a platform for developing, testing, and running applications on all major cloud infrastructures
Alien4-Cloud	This web-based platform accelerates the design of application infrastructures and enables reusability by providing a blueprint catalog and components
CloudFormation	CloudFormation is part of the AWS infrastructure and provides a blueprint of an application that helps to design, model, and set up infrastructure using JSON encoded templates
Cloudiator	This multi-user-capable web-based service allows the description of application and deployment on different public and private clouds
Roboconf	Roboconf is a platform and a tool to deploy and manage elastic cloud applications using deployment, probes, automatic reactions, and reconfigurations
INDIGO	This data and computing platform targets scientific communities and provides an e-infrastructure with cloud frameworks, applications, and tools for clouds and grids
MiCADO	MiCADO is a multi-cloud orchestration and auto-scaling framework for application clusters of Docker containers run on Kubernetes

3 Proposed Architecture

3.1 Requirements for a Cloud Federation Architecture

To utilize containerization technology in cloud federations, the management of containers requires to:

- Have control over container clusters deployed on different CSPs. It is required, in order to support the orchestration lifecycle of containers (i.e., create, deploy, update, scale and terminate) and enhance monitoring.
- Regularly monitor container instances running across different CSPs. It is needed to check their health status and take necessary actions (e.g., replace failed instances).
- Have elastic resource provisioning so that the load and usage of resources can efficiently be managed and calculated.
- Have a component that allows secure management of customers' instances running across the federation.

Fulfilling these requirements enables a provider-agnostic and successful orchestration of containers across a federation and a clear picture to customers on where their containers are running. Moreover, it facilitates managing, monitoring, and billing of container clusters and instances running across the federation. Based on the reviewed literature, the NIST reference architecture is well suited to apply container orchestration for achieving system portability [4]. Given this, applying container technology enhances portability, scalability, efficiency, and service resiliency in cloud federations [9].

3.2 Overall Architecture

The National Institute of Standards and Technology (NIST) defines four components in a cloud federation reference architecture (Fig. 1). These are the Federation Broker, the Federation Operator (FO), the Federation Audit, and the Federation Carrier component [4]. The FO includes six subcomponents namely the Membership Manager, the Policy Manager, the Resource Manager, the Monitoring & Reporting component, the Accounting & Billing component, and the Portability & Interoperability component. These subcomponents are in charge of specific tasks in the federation; the interested reader is referred to NIST Cloud Federation Reference Architecture [4] for a detailed explanation of each subcomponent. In this paper, we adapted the NIST architecture for a centralized (i.e., central broker) arrangement. We further assume that each CSP has a Resource Manager (RM) interface, through which the current status of the CSP's resources can be sent to the central broker.

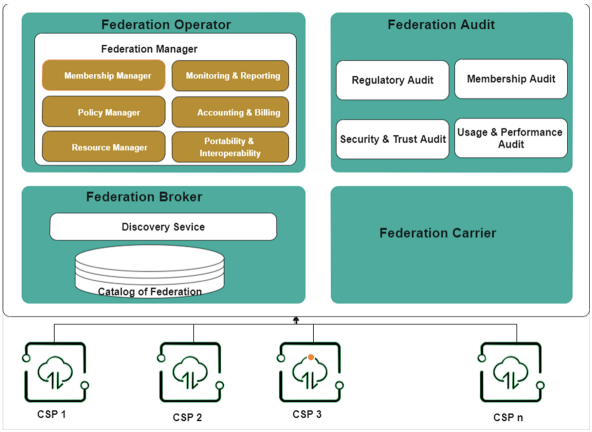

Fig. 1. NIST cloud federation reference architecture [4]

To meet our objective and requirements, we propose two new components to the NIST architecture: the Resource Identifier and the Container Orchestrator.

3.3 New Sub-components

Resource Identifier (RI). The purpose of this subcomponent is to identify resources and enhance the RM's activity (Fig. 2) through the application of various techniques.

The most widely used techniques for resource provisioning are neural networks, linear regression, and support vector machines [10]. Zhang et al. [11] use machine learning techniques to analyze a multi-dimensional cloud resource allocation problem and conclude that the linear allocation algorithm achieves the best performance in their experiment. Therefore and as a starting point, we also propose the linear regression technique for the Resource Identifier component.

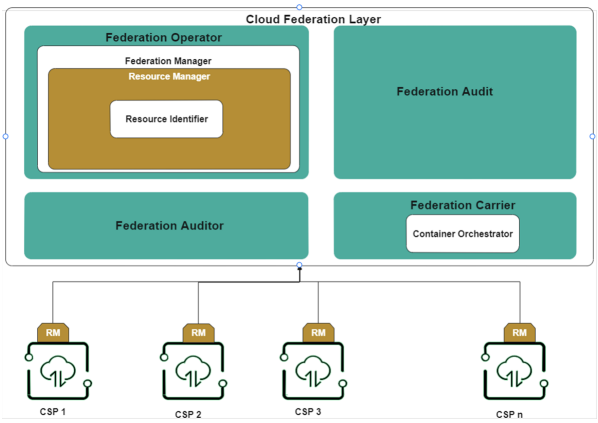

Fig. 2. The proposed cloud federation architecture, which is based on the NIST architecture, includes two new components: the resource identifier and the container orchestrator.

We assume the following resource and user information are provided:

Resource Information. We assume that CSPs offer n types of resources such as compute resources, storage resources, and network resources. The capacity is represented as a vector C = (c1, c2, …, cn). Given these, the unit cost of each resource is represented in the vector P = (p1, p2, …, pn).

User Information. A user *i* can submit requirements denoted by the vector $r^{(i)} = (r_1{}^{(i)}, r_2{}^{(i)}, r_3{}^{(i)}, \ldots, r_n{}^{(i)})$, where $r_s{}^{(i)}$ represents the resources of type *s* (e.g., compute, storage, network) requested by user *i*. User *i* sets a price threshold $t^{(i)}$ that represents the user's willingness to pay for requirements $r^{(i)}$. The overall submission information of user *i* is represented by vector $R^{(i)} = (r^{(i)}, t^{(i)})$. Over time, a user submits various resource requirements $r^{(i)}$ and the corresponding price thresholds $t^{(i)}$, which can be considered in a hypothesis function. The hypothesis function is given by:

$$h_\beta(t^{(i)}) = \beta_0 + \beta_1 r_1^{(i)} + \beta_2 r_2^{(i)} + \ldots + \beta_n r_n^{(i)} \tag{1}$$

Where β1, β2, …, βn stand for the predicted unit price of n types of resources requested by the user i, and β0 is the prediction noise. The variable hβ(t(i)) can be understood as a price expression of the resources required by user i. This hypothesis function predicts the unit willingness-to-pay βi for each requested resource by user i according to the price t(i) that the user was willing to pay.

Based on the resource information and the user information, the aim is to determine the appropriate CSPs for handling the required resources requested. The RI component is initialized when the client's request for extra resources is received along with the price that the client is willing to pay. Then, the RI component contacts all CSPs for their current resource status. Once the status of CSPs is identified, the list of available CSPs will be filtered. If the forwarded list is empty, which means, there is no available CSP currently, then the message about the unavailability of all CSPs will be sent to the requester. But if the list is not empty, then the log file, which contains historical information of requested resources, proposed price by a customer, and information on allocated CSP for that request will be extracted from the database. Then, the log files will be checked, in order to check whether the same request has been fulfilled before. Once identified, the CSP information, which was chosen previously for a similar request, will be forwarded to filter the list based on the geographically nearest location. This step is conducted to save computational power. But if a similar request could not be found in the log file database, then the component uses Eq. 1 to compute and predict whether the user of the requesting CSP would accept the available resources of the CSPs. This is the case, if the price of the available resources of the CSPs is lower than the estimated price. After the unit price is predicted and a list of CSPs, which can deliver their service based on the price, are identified. Then, the geographically nearest CSP from the list is selected. Finally, the resource identifier returns the selected CSP information and updates its log file. Figure 3 presents the detailed step-by-step activity of the RI component.

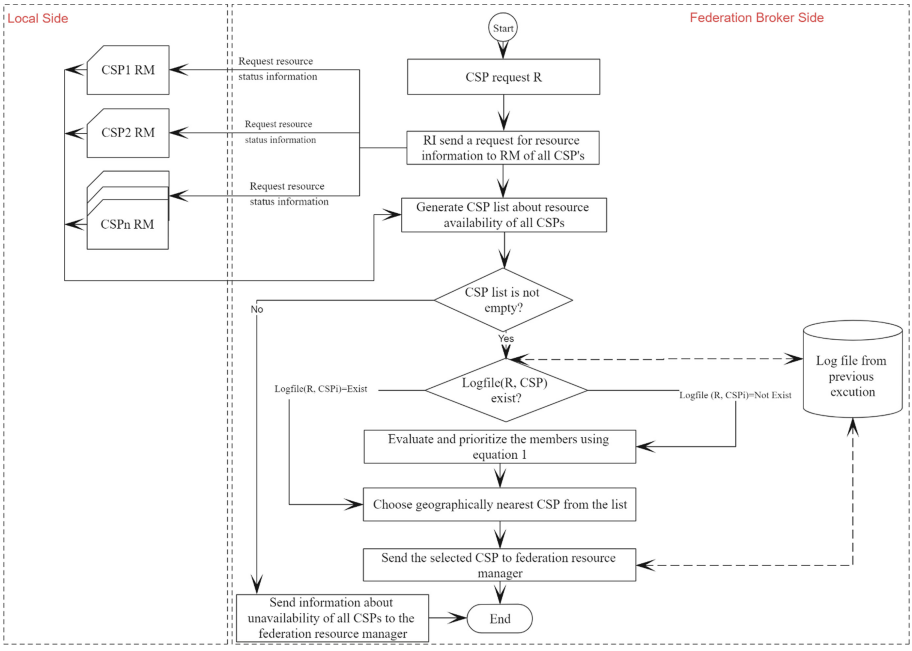

Fig. 3. Resource identifier algorithm.

Container Orchestrator. This subcomponent is proposed as part of the Federation Carrier component. It facilitates the orchestration of containers across the federation. Whenever the federation orchestrator receives a request, for example, for regular cloud computing, containers are managed and controlled by the orchestrator (e.g., Docker Swarm, Kubernetes). However, since cloud federation deals with completely independent entities, managing container clusters and updating master nodes that typically lead to rebuilding the container image, can be challenging from many aspects. To overcome such problems, the federation container orchestrator takes full control of container clusters deployed across federation members and manages all their lifecycle. In other words, creating, deploying, scaling, and terminating containers across the federation is the responsibility of the container orchestrator.

The container orchestrator is initialized by a request from a CSP along with a container's token. The token is used to uniquely identify each container image. Once the request is received, this component checks whether or not the image exists in the repository. If so, the container orchestrator pulls a copy of the container image from the container repository. If not, it accesses the container image from the requester CSP. Once the container image has been received either from the CSP or from the repository, communication can be established with the CSP, which is selected by the resource identifier for deploying the container. Figure 4 visualizes the operation of the container orchestrator component.

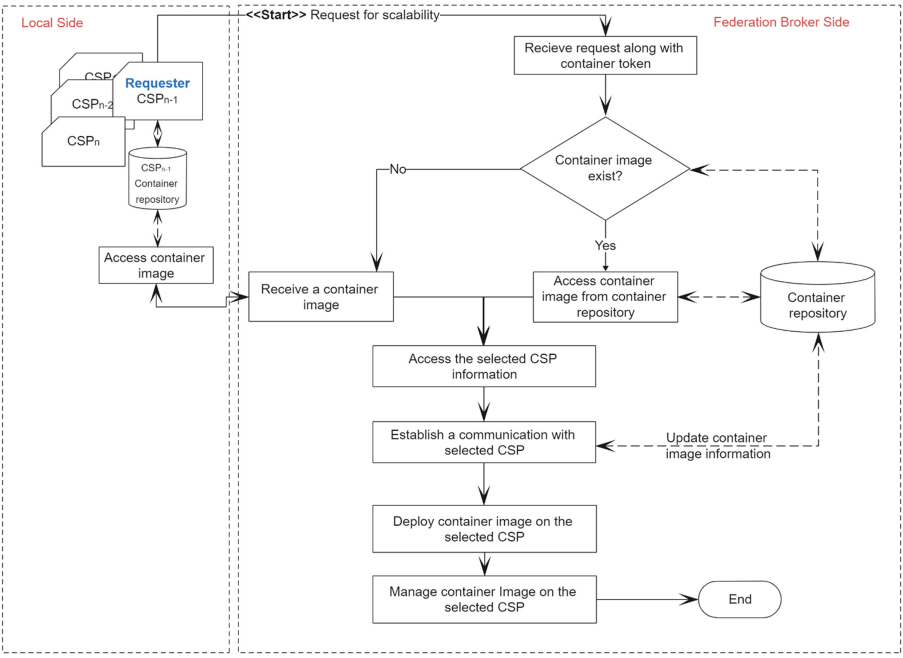

Fig. 4. Container orchestrator.

4 Discussion

Cloud federation provides an inexpensive way of maximizing resource utilization through increased resource flexibility, scalability, and efficient service delivery. Optimizing resource identification and taking advantage of containerization technology in the resource management of a federated environment can add more benefits to cloud clients and CSPs. On top of these benefits are: (i) the resource identification technique matches the customers' needed resources within the budget they can afford. As a result, customers save time and effort as they do not have to hassle looking up and comparing prices for the resources they need; (ii) The containerization technology maximizes the efficiency of resources, as containers require less system resources compared to virtual machines and, more importantly, they are highly portable regardless of where they are deployed; (iii) The combination of resource identification and container orchestration across federation accelerates deployment and production cycles. This work aims at achieving that by proposing a component for resource identification and a component for container orchestration in cloud federations.

Regarding resource identification, Tsakalidou et al. [10] found that the most widely used techniques for resource provisioning are neural networks, linear regression, and support vector machines. Linear regression, in particular, is found to achieve better performance compared to other techniques. Therefore, the linear regression technique is chosen over other techniques for our purpose. The Linear regression technique enhances resource identification of the cloud federation based on previously generated provisioning logs, tracing and monitoring information, and the currently available resources of the federation members [7]. In this regard, the continuous provisioning of resources by federation members can be leveraged to create a self-optimizing resource identification mechanism. This can help optimizing resource provisioning by federation members and improve their accuracy for future requests. As a result, this continuous optimization of resource identification and provisioning enhances the overall quality of service on one hand and helps spot potential improvements of federation members on the other hand. Moreover, it can help drive better resource allocation decisions and orchestrate resources in the federation with minimum cost and more efficiency. Therefore, the resource identifier plays an important role not only in choosing where to orchestrate resources efficiently but also in addressing possible improvements of the whole federation.

Regarding the second component, container orchestration solves the problem of compatibility and portability of applications that are already implemented in other cloud arrangements [7]. The proposed architecture includes the container orchestrator, which is a sub-component of the distribution manager, is responsible for orchestrating containers across federation members. While existing container orchestrator's work at the provider level, the proposed one manages container clusters at the federation level. At a high level, this gives the federation an advantage of maximizing the use of resources in an efficient way. At a low level, this adds more flexibility and portability support for an application. An additional benefit is that it allows automated management of applications, and therefore, reduces the overhead needed to manage the entire lifecycle of containers deployed across a federation. As container orchestration in the federation has not been proposed in cloud federation, this sub-component could give additional benefits for federated clouds with respect to scalability, flexibility, and portability.

5 Conclusion and Future Work

In this article, we presented a container orchestration architecture in a centralized (i.e., cloud broker) federation environment. The proposed architecture allows the resource manager to prioritize the selection among federated CSPs based on logs and tracing information generated from continuous provisioning of resources, enabling self-optimizing decisions.

By enhancing the NIST reference architecture, we added two subcomponents called Resource Identifier and Container Orchestrator. These two subcomponents track the available resources in the federated cloud and use this information to prioritize and decide on which federation members shall run the container. By taking advantage of containerization technology, these sub-components allow managing and orchestrating containers across federation members and, as a result, enhance portability and scalability of applications in federated environments. For future work, a proper simulation tool will be selected to analyze and evaluate the proposed architecture by implementing the linear regression technique for resource identification.

Acknowledgments. This research was supported by the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea) and National Research Foundation of Korea (NRF). This work was also supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2019R1F1A1058487) funded by the Ministry of Science and ICT (MSIT) of Korea.

References

1. Aryal, R.G., Marshall, J., Altmann, J.: Architecture and business logic specification for dynamic cloud federations. In: Djemame, K., Altmann, J., Bañares, J.Á., Agmon Ben-Yehuda, O., Naldi, M. (eds.) GECON 2019. LNCS, vol. 11819, pp. 83–96. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36027-6_8
2. Cho, Y., Jo, C., Kim, H., Egger, B.: Towards economical live migration in data centers. In: Djemame, K., Altmann, J., Bañares, J.Á., Agmon Ben-Yehuda, O., Stankovski, V., Tuffin, B. (eds.) GECON 2020. LNCS, vol. 12441, pp. 173–188. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63058-4_15
3. Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., Kunze, M.: Cloud federation. In: Computing, p. 7 (2011). https://www.researchgate.net/publication/312280049_Cloud_federation. Accessed 23 Mar 2021
4. Lee, C.A., Bohn, R.B., Michel, M.: The NIST Cloud Federation Reference Architecture (2020). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-332.pdf
5. Aryal, R.G., Altmann, J.: Fairness in revenue sharing for stable cloud federations. In: Pham, C., Altmann, J., Bañares, J.Á. (eds.) GECON 2017. LNCS, vol. 10537, pp. 219–232. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68066-8_17
6. Aryal, R.G., Altmann, J.: Dynamic application deployment in federations of clouds and edge resources using a multiobjective optimization AI algorithm. In: 2018 3rd International Conference Fog Mobile Edge Computing. FMEC 2018, pp. 147–154 (2018). https://doi.org/10.1109/FMEC.2018.8364057

7. Tomarchio, O., Calcaterra, D., Modica, G.D.: Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. J. Cloud Comput. **9**(1), 1–24 (2020). https://doi.org/10.1186/s13677-020-00194-7
8. Ahmed, U., Raza, I., Hussain, S.A.: Trust evaluation in cross-cloud federation: survey and requirement analysis. ACM Comput. Surv. **52**(1), 1–37 (2019). https://doi.org/10.1145/3292499
9. Kim, D., Muhammad, H., Kim, E., Helal, S., Lee, C.: TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform. Appl. Sci. **9**(1), 1–14 (2019). https://doi.org/10.3390/app9010191
10. Tsakalidou, V.N., Mitsou, P., Papakostas, G.A.: Machine learning for cloud resources management -- An overview, January 2021. http://arxiv.org/abs/2101.11984. Accessed 25 May 2021
11. Zhang, J., Xie, N., Zhang, X., Yue, K., Li, W., Kumar, D.: Machine learning based resource allocation of cloud computing in auction. Comput. Mater. Contin. **56**(1), 123–135 (2018). https://doi.org/10.3970/cmc.2018.03728