Service Management in Virtual Machine and Container Mixed Environment using Service Mesh

Hokeun Lim
School of Eletronic Engineering
Soongsil University
Seoul, Republic of Korea
limhk@dcn.ssu.ac.kr

Younghan Kim
School of Eletronic Engineering
Soongsil University
Seoul, Republic of Korea
younghak@ssu.ac.kr

Kyoungjae Sun
School of Eletronic Engineering
Soongsil University
Seoul, Republic of Korea
gomjae@dcn.ssu.ac.kr

2021 International Conference on Information Networking (ICOIN) | 978-1-7281-9101-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICOIN50884.2021.9333888

***Abstract*— The microservice architecture is an architectural style in which a service focused on performing one function is divided into independent and deployable small units of service, linked with other services through API, and independently developed and operated. However, in the communication between APIs, additional functions are required for communication between microservices such as load balancing and logging. This can be solved through service mesh, but currently service mesh only considers the container environment, so additional design is required for a structure that can manage virtual machines in the service mesh structure. In this paper, we propose a structure for deploying and managing microservices in a container and virtual machine mixed environment using service mesh.**

Keywords—MSA, Istio, Tacker, OpenStack, Kubernetes

I. INTRODUCTION

A microservice architecture is that structure in which a service model that separates a service focused on performing one function into the smallest independent and deployable service, connects with other services through API, and can independently develop and operate each.[1] In monolithic architecture, when an error occurs or when performing an update, the entire service has to be stopped, but in microservices, components are independent, so errors can be resolved and updated with only the corresponding service stopped. However, as the structure of microservices was changed, the monolithic to inter-process communication was changed to API communication, and an additional function for API communication was required. Communication functions are written in the code of each services and had to be written according to the development language, and the service code needed to be rewritten for modify communication codes. This is very inefficient, and the structure that came out to solve it is the service mesh structure. Through the service mesh architecture, communication between microservices can be controlled uniformly. However, there is some case that using container and virtual machine together, but the current service mesh architecture only consider container environment, and if a virtual machine is to be used, the user must proceed manually.[2][3] An additional structure is required to automate proxy deployment in virtual machines. In this paper, we propose a structure to deploy services and manage communication between services in an integrated container and virtual machine environment using a service mesh and service orchestrator.

II. RELATED WORKS

A. Microservice Architecture and Service Mesh

The microservice is consists of several service models that perform one function. Therefore, it is possible to develop each service by using a suitable programming language or tool, and since the services are organized in an independent structure, it is possible to expand the service unit without affecting other services. In the case of a monolithic structure, a service consisting of several modules is operated in a single workload. As the microservice structure was changed, each module was operated for each workload, and the communication between processes was changed to API communication between the workloads. With the change to API communication between workloads, additional functions such as logging, load balancing, and service discovery were required for API communication. These communication functions are written in the service code, and have the disadvantage of touching the code directly to modify it. In order to manage communication functions uniformly, a service mesh structure has emerged for solve this problem.[4]

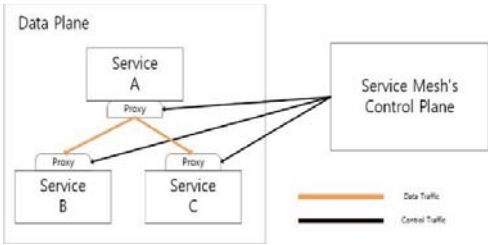

Figure 1. Service Mesh Architecture

The service mesh structure takes the form of performing communication functions in the proxy by using it as the data plane, and the control plane takes the role of controlling the proxy by calling API.

B. Virtual Machine and Container

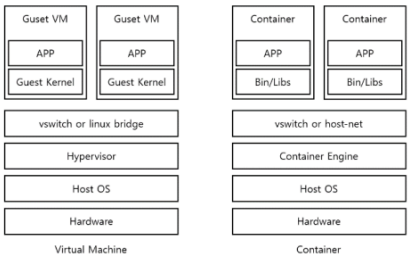

Figure 2. Virtual Machine and Container

It is a structure in which an independent operating system can be executed through hardware virtualization, which is a VM, which is a core virtualization configuration of a cloud system, and requires the OS, kernel, binaries and related libraries required when executing application programs for services, and the host is highly dependent on hardware resources. On the other hand, the container shares the kernel module required to run the application program with the host and uses only the binaries and libraries that the program needs, so it is less dependent on hardware than the VM. In the case of a virtual machine, a separate OS and library are required, so more resources are required than those required to run containers, and the time to run them is also longer. Therefore, the virtualization of a cloud system recently follows a container method a lot.[5] However, in terms of services, services that have not yet been containerized or virtual machines are still more efficient than container. Therefore, there are cases where virtual machines and containers are mixed, and in this case, a new structure is required that can manage traffic passing through both workloads.[6]

III. Service Management Architecture in a Virtual Machine and Container Mixed Environment

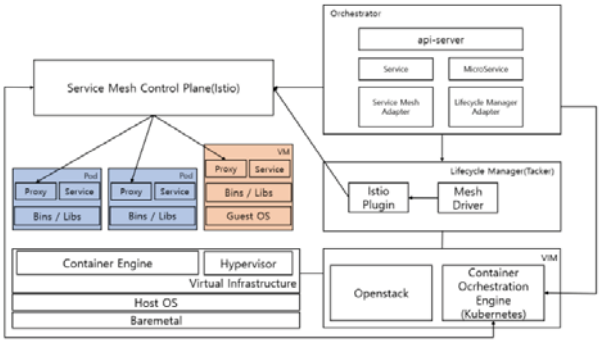

Figure 3. Proposed Architecture

In this paper, Figure 3 is proposed for service management in Virtual Machine and Container mixed Environment. The orchestrator is a component that receives service creation, and interacts with Tacker and Istio to perform control requests for communication between microservices that constitute a service. VNFM is used for virtual machine and container life cycle management, and OpenStack and Kubernetes are used as VIM. OpenStack and Kubernetes play a role of creating virtual machine and container workloads, respectively, and when creating container workloads, Kubernetes plays a role of deploying proxies to automatically include containers in the service mesh structure.[7]

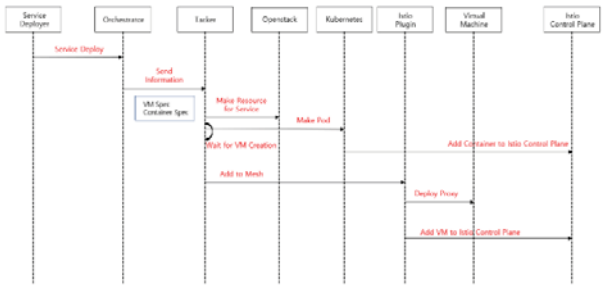

Figure 4. Sidecar Deploy Procedure

However, when creating a workload through OpenStack, the proxy is not automatically deployed, and therefore, if you want to control the traffic that passing through the virtual machine in the service mesh structure, an additional proxy deployment procedure is required. OpenStack is a component that only provides a virtual environment, and deploying a proxy to a virtual machine is a special case of adding a virtual machine to the service mesh structure. As Tacker performs the overall lifecycle management of the service, Tacker has to take the role of deploying proxy, and it only occurs when the service is specified to be added to the mesh. Therefore, it is developed as a plugin in Tacker, and when you want to use it, install a proxy. This function is handled by the Istio Plugin, and the it is operated when tacker check the field of service deployment file. Tacker sends a request to add virtual machine into the mesh to the Istio Plugin, and Istio Plugin works with Istio Control Plane. Istio Plugin send the information of virtual machine to add it to the Istio Control Plane.

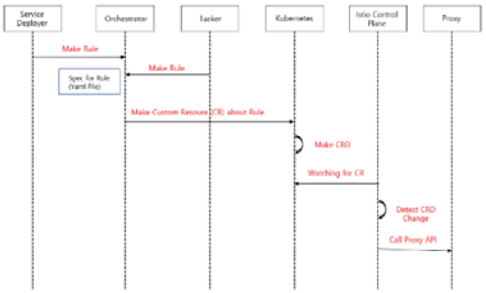

Figure 5. Configuring Communication Procedure

The service is deployed through an orchestrator, and when the service is deployed, the specifications of the microservices that make up the service and the configuration contents of the connection between the microservices are included. The orchestrator delivers the virtual machine or container specification information required for the service to Tacker to create microservices. Tacker forwards requests for container creation to Kubernetes, and processes requests for virtual machine creation by interacting with OpenStack. In the case of containers, the proxy is deployed through the Kubernetes controller function. Tacker specifies the namespace to use the Kubernetes controller function, and the proxy is automatically deployed when the service is created. The proxy deployment of the virtual machine is performed by the Istio Plugin, which is the Tacker Plugin function, and it notifies Tacker after proxy deployment. Tacker notifies the orchestrator after creating virtual machine and container workloads and deploying proxies.

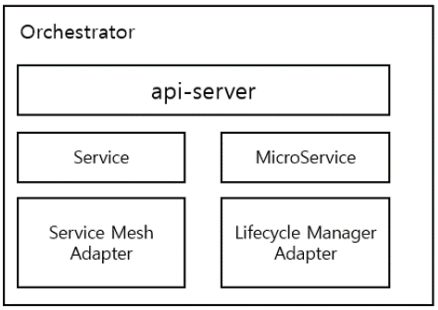

Figure 6. Orchestrator Components

The orchestrator structure is shown as Figure 6. There is an API server for receiving user requests, and there are four as components for deploying services. Service Mesh Adapter is an element that manages the API server address of the Service Mesh Control Plane, and is a component in case of using multiple service mesh implementations. Lifecycle Manager Adapter is also a component to manage the IP address of LCM(Lifecycle Manager) because there can be multiple LCMs used when deploying services. Microservice component is a component for creating microservices, interacts with LCM Adapter and Tacker, and delivers microservice creation request to LCM. The Service component receives notification that the service deployment is over through the microservice component, interacts with the Service Mesh Adapter to establish a connection between services, and sends a request to the service mesh control plane.

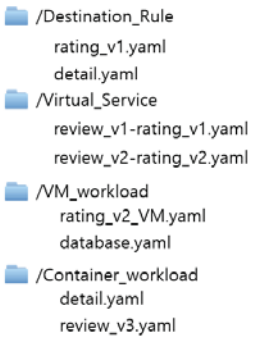

Figure 7. Service Package Example

In order to deploy a service through an orchestrator, the package structure is shown as Figure 7. The package consists of 4 directories and is for connectivity between virtual machines, containers, and microservices. The files for the workload should be the same as the definition of the virtual machine and container defined by Tacker, and the connectivity is created according to the template defined by Istio. When deploying a virtual machine workload through Tacker, the command for proxy deployment has to be written in specific field to deploy the proxy. In addition, by writing the flag to be added to the service mesh as true, the service mesh plugin is operated and add virtual machine to the service mesh control plane.

IV. IMPLEMENTATION RESULT OF ARCHITECTURE

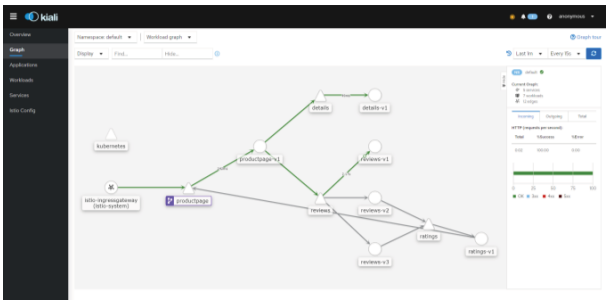

Figure 8. Deployed Services

The results of deploying service through the orchestrator proposed in this paper are shown as Figure 8. I use the book info example which is from Istio Document.[8] Each service runs in a container or virtual machine, and you can see that all services are registered in the service mesh control plane check by kiali. The definition of connectivity between services is set through Istio, and the set result can be checked through the Istio command. After the service was deployed, the connectivity of services can be checked by Istio Command and we can modify it by using it. Deployed services can be checked through Tacker commands, and container services can be checked using Kubernetes commands.

V. CONCLUSTION

In this paper, we proposed an orchestrator and structure for service management in a virtual machine and container mixed environment using a service mesh. Based on the structure, a service mesh orchestrator was used to automate proxy deployment for virtual machines on the LCM side, and the service was configured by defining service deployment and connectivity consisting of a microservice structure in a mixed virtual machine and container environment. This was verified through an experiment by developing code with OpenStack, Kubernetes, and Istio, which are opensource for the implementations of Components.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2020-0-00946,Development of Fast and Automatic Service recovery and Transition software in Hybrid Cloud Environment)

REFERENCES

[1] O.Al-Debagy and P.Martinek, "A Comparative Review of Microservices and Monolithic Architectures," 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 2018, pp. 000149-000154

[2] Z. Benomar, F. Longo, G. Merlino and A. Puliafito, "Enabling Container-Based Fog Computing with OpenStack," 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Atlanta, GA, USA, 2019, pp. 1049-1056.

[3] Docker blog, Container and VMs Together. [Online]. Avaliable : https://www.docker.com/blog/containers-and-vms-together/

[4] W. Li, Y. Lemieux, J. Gao, Z. Zhao and Y. Han, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco East Bay, CA, USA, 2019, pp. 122-1225.

[5] C. Pahl, "Containerization and the PaaS Cloud," in IEEE Cloud Computing, vol. 2, no. 3, pp. 24-31, May-June 2015.

[6] L. T. Bolivar, C. Tselios, D. Mellado Area and G. Tsolis, "On the Deployment of an Open-Source, 5G-Aware Evaluation Testbed," 2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), Bamberg, 2018, pp. 51-58

[7] H. Yang, C. Hoang and Y. Kim, "Architecture for Virtual Network Function's High Availability in Hybrid Cloud Infrastructure," 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 2018, pp. 1-5.

[8] Istio Document, Bookinfo Application. [Online]. Avaliable : https://istio.io/latest/docs/examples/bookinfo/.