논문 18-43-10-18 The Journal of Korean Institute of Communications and Information Sciences '18-10 Vol.43 No.10
https://doi.org/10.7840/kics.2018.43.10.1682

컨테이너 오픈스택을 위한 단대단 모니터링 시스템 설계

응웬쫑당찐◆, 유 명 식°

End-to-End Monitoring System Design for Containerized OpenStack

Trinh Nguyen◆, Myungsik Yoo°

요 약

클라우드 공급자는 컨테이너 기술을 사용하는 새로운 방식으로 발전하고 있다. 목표는 유연성이 없고 자원 집약적인 클라우드 인프라 구축 프로세스를 유연하고 경제적인 프로세스로 대체하는 것이다. 새로운 소프트웨어 아키텍처는 서비스 연결 방식을 변경한다. 이로 인해 클라우드 플랫폼은 자원 설정, 성능, 오류 관리 등을 포함한 여러 가지 새로운 문제에 직면하게 된다. 따라서 컨테이너 시스템의 가용성과 안정성을 보장하려면 모니터링 기능을 심층적으로 고려해야 한다. 그러나 이러한 문제는 아직 연구되지 않았으며, 이러한 컨테이너 기반 클라우드 환경을 위한 단대단 모니터링 시스템을 설계에 대한 연구가 수행되어야 한다. 본 논문은 컨테이너화된 OpenStack 시스템을 위한 모니터링 아키텍처와 메커니즘을 제안한다.

Key Words : Monitoring System, Container-based Cloud, OpenStack, Docker, Kolla

ABSTRACT

Cloud providers are moving toward a new deployment method that is using the container technology. The objective is to replace the inflexible, painful, resource-intensive deployment process of the cloud infrastructure with a flexible, painless, inexpensive deployment process. For instance, OpenStack uses the Kolla project [11] to foster the effort and has already reached a mature level which can be used in production. The new software architecture changes how services are connected together. That exposes the cloud platform to many new challenges including resource provision, performance, failure management etc. Therefore, to guarantee the availability and stability of the containerized system, monitoring features have to be considered thoroughly. However, that subject is left unexplored and raises a need to design an end-to-end monitoring system for such container-based cloud environment. This paper proposes a monitoring architecture and mechanism for the containerized OpenStack system.

※ This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion).

◆ First Author : (ORCID:0000-0001-6885-5935)Department of ICMC Convergence Technology, Soongsil University, Seoul, Republic of Korea, dangtrinhnt@soongsil.ac.kr, 학생회원

° Corresponding Author : (ORCID:0000-0002-5578-6931)School of Electronic Engineering, Soongsil University, Seoul, Republic of Korea, myoo@ssu.ac.kr, 종신회원

논문번호 : 201810-300-D-RN, Received October 1, 2018; Revised October 4, 2018; Accepted October 4, 2018

Ⅰ. Introduction

OpenStack is an open source cloud platform that has a modular architecture. Deploying OpenStack is difficult because it comprises many different components (e.g. Nova, Neutron, Keystone etc.) that need to be connected together. A big problem with most of the existing deployment methods is that all OpenStack services are deployed as static packages on top of a shared operating system. This makes the ongoing operations, troubleshooting and upgrades really problematic. To resolve those issues, cloud operators deploy all OpenStack services using containers. Kolla was born out of that idea and become one of the main projects that are housed within OpenStack's umbrella. The deployment process is quite complicated, however, the end result is a highly flexible OpenStack cloud deployed using Docker containers, managed and orchestrated by Kolla Ansible, a set of Ansible playbooks that automate the tasks.

Kolla provides production-ready containers and deployment tools for operating OpenStack clouds that are scalable, fast, reliable, and upgradeable. Kolla uses Docker to store the file-system contents of images and provides an execution environment for those containers. One key advantage of Docker is that a registry system provides a central repository of images which can be used to instantiate a deployment. By choosing Docker, cloud providers obtain the superpower of immutability with the Docker registry. Kolla has approximately ninety containers. Sixty would likely be running; the remainders are base containers used as intermediate build steps shared by children containers. Figure 1 illustrates the OpenStack deployed with Kolla's architecture at a higher level where all of the services are containers.

Monitoring a cloud platform like OpenStack has been a mature subject in both literature and reality. It is a critical process for checking the vitality of a cloud infrastructure's software, physical as well as virtual assets. Moreover, monitoring is the first step in minimizing the cloud failures's impact [10]. Cloud providers often use agent-based monitoring solutions that are installing client software inside OpenStack services to collect all the needed information. Because most for the services are built natively in the operating system in the traditional deployment model, monitoring agents are obviously another kind of application that can operate at the system level to read most of the desired data. However, when all of the services are packed into containers, that monitoring model will have to change. Specifically, the agent software needs to be upgraded with the ability to harvest information from the running containers. Also, OpenStack services's interconnections require a different monitoring approach. Because, unlike conventional services, the containers communicate with each other not using ports but running in a host networking mode, effectively disabling any network

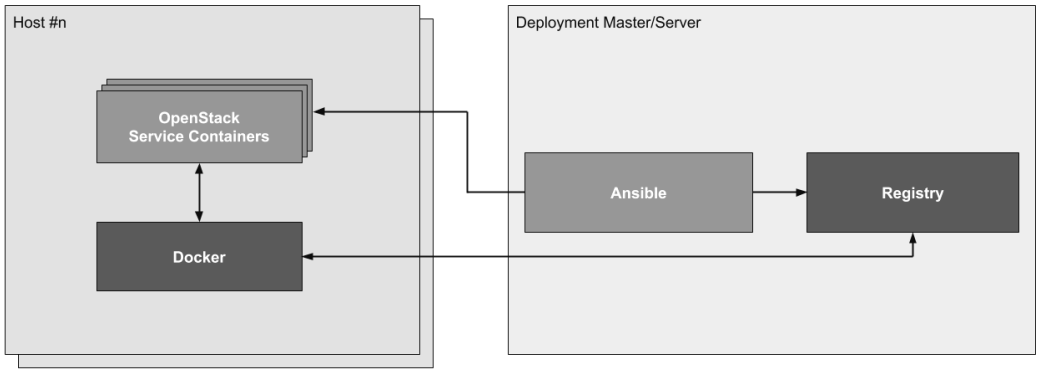

Fig. 1. OpenStack Kolla's Hight Level Architecture

isolation and giving all containers access to TCP/IP stacks of their Docker hosts. Those considerations are critical because without the data sent from the agents, monitoring server is useless and thus, breaks OpenStack's reliability.

By studying the Docker container's architecture and how it is orchestrated in the container-based cloud platform this paper presents a novel end-to-end monitoring solution for OpenStack. The proposed framework not only tackles the challenges of the dynamic and complex nature of the containerized system but also tries to provide a practical and extensible platform that can evolve with the cloud. The rest of this paper is organized as follows. Section II describes the architecture and building blocks of the proposed monitoring system for container-based OpenStack. Section III shows the experimental results to demonstrate the feasibility and effectiveness of the designed solution. Finally, section IV concludes the paper and discusses future works.

Ⅱ. An End-To-End Monitoring System For Containerized Openstack

Having a good understanding of the containerized cloud platform is crucial in designing a comprehensive monitoring solution for it. Thus, a brief analysis of the containerized OpenStack needs to be introduced first. The containerized OpenStack as shown in Fig. 1 consists of two main functional groups. The first group is Deployment Master or server that is in charge of orchestrating the container deployment for OpenStack services using Ansible [12] automation tool. In addition, a Docker Registry is optionally placed within the deployment master to provide the container images for Ansible tasks. Otherwise, Kolla has to use an external container repository. The second functional group of OpenStack Kolla is the Deployment Hosts where it could be only one host or multiple hosts depending on the size of the desired cloud system. On each host, there is a Docker engine that is used to manipulate the dockerized services. Hence, two different monitoring targets can be identified which are the Deployment Master or Hosts and the containerized OpenStack services inside each host.

Based on the architecture of Kolla, an end-to-end monitoring solution is designed. The proposed system comprises two components. The first and foremost is the server side which is also split into elements. Those are the monitoring server where monitored data is scraped from the agents and stored in the time-based format, the alert manager that manages all the alerting tasks when a predefined event occurs (e.g. send alarm email to administrator when a target is down), and finally, monitored data stored in the server that can be queried and visualized by the front-end. The next component is the client side or the monitoring agents which plays a very important role in the system as discussed in the previous section of this paper.

Additionally, there are two types of agents deployed in this proposed system because two different monitoring targets are presented. They are the Deployment Master or Hosts which are operating systems, and the containerized OpenStack services which are special packages that contain the library dependencies, the binaries, and a basic configuration. The agents will expose a specific set of metrics from the environment in which they are installed. The agents deployed in the Deployment Master or Hosts are normal applications that can scrape information such as CPU, Memory usage, Disk I/O etc. directly from the underlying OS. On the other hand, the client software that monitors containers, i.e. the cAgent, needs to have the ability to communicate with the cgroups virtual file system [13] where Docker containers performance statistics are stored via the Docker engine [8]. The later agent type can be realized using cAdvisor [14], a special tool designed to monitor containers. It can be installed natively in the host or inside a container.

By being flexible on the agent software, the proposed monitoring system opens an opportunity for many different useful metrics of the targets to be collected and processed. However, that method creates a problem with complex communication between the client and monitoring server which traditionally requires lots of development effort.

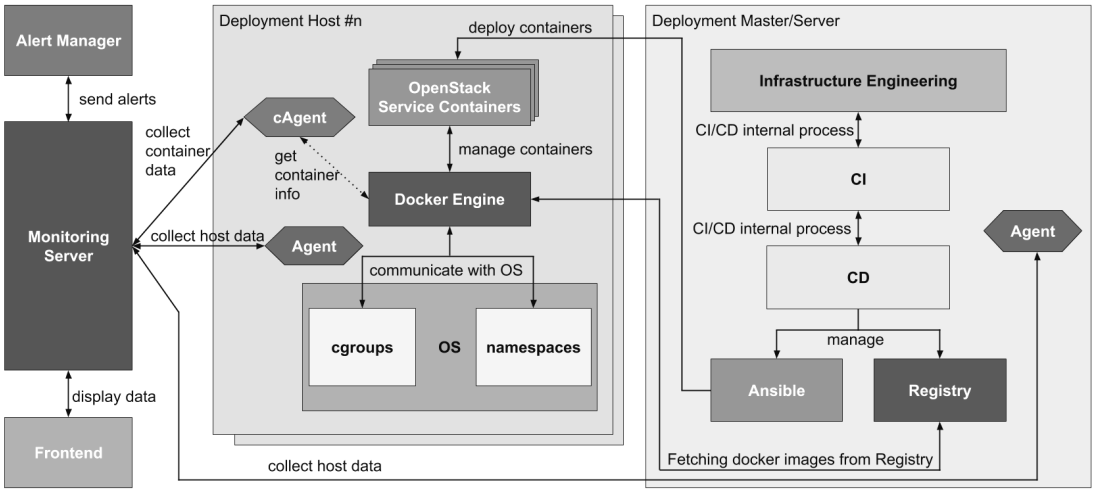

Fig. 2. The Proposed Monitoring System Architecture.

Therefore, a unified communication protocol of monitoring server and its agents is required. In this proposed system, monitoring clients expose metric data of the targets to the server via a TCP/IP channel that is a port and the address of those targets. Hence, Prometheus [15], a monitoring software, which supports that kind of client-server communication is leveraged.

As shown in Figure 2, the proposed architecture can monitor the Deployment Master and the Deployment Hosts in which the containerized OpenStack services are deployed as well as the containers. Additionally, all the autonomous deployment tasks are controlled in harmony with the Continuous Integration and Continuous Development (CI/CD). Those capabilities of the proposed system create an end-to-end monitoring solution for the container-based cloud platform.

Ⅲ. Experiment

An experiment was conducted to evaluate the proposed monitoring system. Firstly, a containerized OpenStack environment is setup using Kolla with one Deployment Master and one Deployment Host containing all the services including Heat, Nova, Neutron, Cinder, Glance, Swift, Horizon, and Keystone. Next, a Prometheus server instance along with the Alert Manager and the Grafana dashboard [16] as front-end are installed in another machine. Finally, for each monitoring targets defined in Section II, we deploy the suitable agents and configure them to connect with the Prometheus server. Specifications of the three servers are presented in Table 1.

In this experiment, two scenarios are examined to recognize the feasibility and practicality of the proposed mechanism. The first scenario is that when a high CPU usage event occurs on the Deployment Host, the agent directly harvests information from the servers and send it to Prometheus. The abnormal event will be displayed on the Grafana dashboard. As shown in Figure 3, the CPU usage (CPU in user mode area) of the Deployment Host is more than

Table 1. Specifications of the experiment

Server	Specifications
Deployment Master	CPU Intel Xeon E3 1240 3.4GHz 4 x 4GB DDR3 1333MHz 1TB HDD Ubuntu 16.04 LTS x64
Deployment Host	CPU Intel Xeon E3 1240 v2 3.4GHz 4 x 4GB DDR3 1333MHz 1TB HDD Ubuntu 16.04 LTS x64
Prometheus Server	CPU Intel Core i5 7600 3.5GHz 8GB HDDR4 2400MT/s 1TB HDD Ubuntu 16.04 LTS x64

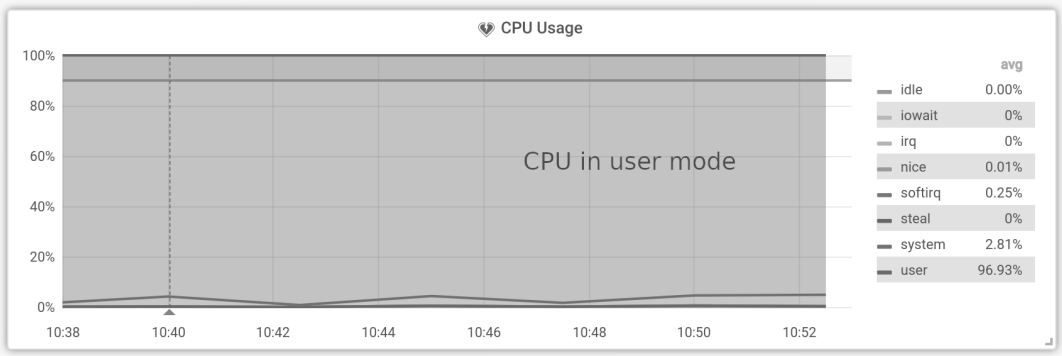

Fig. 3. High CPU usage (CPU in usage mode) is shown in the Grafana dashboard.

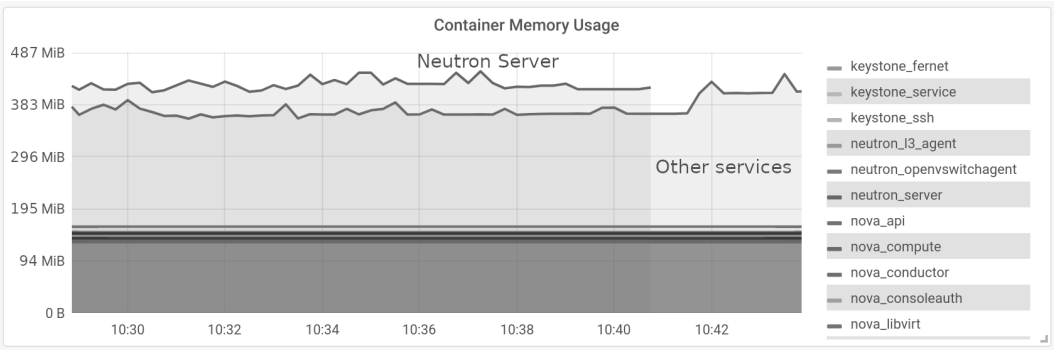

Fig. 4. The Neutron Server container is down. Grafana stops displaying Neutron Server container's memory usage.

90%. The second scenario is when a containerized OpenStack component (in this case, we chose the Neutron Server service) is killed, cAdvisor which is the monitoring agent installed inside another container exposes the disconnection of Neutron Server metric to Prometheus server. The Grafana dashboard stops displaying the monitoring data of the Neutron Server container. For example, the memory usage of the Neutron Server container is stopped showing as presented in Figure 4.

Ⅳ. Conclusion

With the advanced features of the container technology, OpenStack deployment and operation become increasingly flexible and painless. However, the containerized architecture of the cloud while eliminates some existing problems such as dependencies, resource consumption etc. creates new challenges in failure management and reliability assurance. Tackling those issues, the proposed monitoring mechanism through careful consideration and experiment has proved to be feasible and practical to provide an end-to-end monitoring solution for the container-based OpenStack. Future work will consider a deployment model for the agents that are automatic and scalable to adapt to the increasingly complex structure of the cloud.

References

[1] M. Großmann and C. Klug, "Monitoring container services at the network edge," *IEEE 2017 29th Int. Teletraffic Congress* (ITC 29), vol. 1, 2017.

[2] C.-C. Chang, et al., "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," *IEEE GLOBECOM*, Dec. 2017.

[3] F. Moradi, et al., "ConMon: an automated

container based network performance monitoring system," *IEEE 2017 IFIP/IEEE Symp. Integrated Netw. and Serv. Management (IM)*, pp. 54-62, May 2017.

[4] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 42-48, 2017.

[5] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure devops," *2016 IEEE Int. Conf. Cloud Eng. (IC2E)*, pp. 202-211, 2016.

[6] W. Lloyd, et al., "Serverless computing: An investigation of factors influencing microservice performance," *2018 IEEE Int. Conf. Cloud Eng. (IC2E)*, pp. 159-169, 2018.

[7] C. Pahl, et al., "Cloud container technologies: a state-of-the-art review," *IEEE Trans. Cloud Computing*, p. 1, May 2017.

[8] E. Casalicchio and V. Perciballi, "Measuring docker performance: What a mess!!!." in *Proc. 8th ACM/SPEC on Int. Conf. Performance Eng. Companion, ACM*, pp. 11-16, Apr. 2017.

[9] V. Agrawal, et al., "Log-based cloud monitoring system for OpenStack," *2018 IEEE BigDataService*, pp. 276-281, Mar. 2018.

[10] P. T. Endo, et al., "Minimizing and managing cloud failures," *Computer*, vol. 50, no. 11, pp. 86-90, 2017.

[11] The OpenStack Foundation, *OpenStack Kolla project*, https://wiki.openstack.org/wiki/Kolla..

[12] Ansible, [ONLINE]. Available: https://www.ansible.com.

[13] cGroups, [ONLINE]. Available: http://man7.org/linux/man-pages/man7/cgroups.7.html.

[14] cAdvisor, [ONLINE]. Available: https://github.com/google/cadvisor.

[15] Prometheus, [ONLINE]. Available: https://prometheus.io.

[16] Grafana, [ONLINE]. Available: *https://grafana.com*.

응웬쫑당찐 (Trinh Nguyen)

Trinh Nguyen received his B.Eng. degree in Computer Networking from University of Information Technology, VNU-HCM, Ho Chi Minh City, Vietnam, in 2012. He has been pursuing the Master's degree in ICT at Soongsil University since Fall 2017. His research interests include Software-Defined Networking, Network Function Virtualization and Cloud Computing.

유 명 식 (Myungsik Yoo)

Myungsik Yoo received his B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, Republic of Korea, in 1989 and 1991, and his Ph.D. in electrical engineering from State University of New York at Buffalo, New York, USA in 2000. He was a senior research engineer at Nokia Research Center, Burlington, Massachusetts. He is currently a professor in the school of electronic engineering, Soongsil University, Seoul, Republic of Korea. His research interests include visible light communications, sensor networks, Internet protocols, control, and management issues.