Proposal for Namespace Design and Distributed Name Resolution Mechanisms in Container-Based Environments

2010044 Takumi Kataoka

With the recent growth in a wide variety of IT services, including Web services, there is a high demand for flexible and agile systems for efficient service construction, operation, and management efficiency. In order to meet such needs, the use of container-based virtualization technology is increasing in cloud computing and distributed systems.

Container-based virtualization is an OS-level virtualization technology that enables the creation of containers, which are isolated spaces on a per-process basis, in which services and applications can be run. By using containers, it is possible to make it look as if the services inside the container are occupying OS resources. Containers are implemented using the functions provided by the kernel of the host OS. Containers do not virtualize the guest OS compared to VM, Virtual Machine, a hardware-level virtualization technology. So, containers have the advantages of fast boot-up, portability, and the ability to easily reduce environmental differences. Existing container implementations have not been able to use containers generically, so deploying services using physical servers or Virtual Machines for service provision was time-consuming and inflexible. However, with the advent of Docker, the current popular implementation of container-based virtualization, it is now possible to rapidly launch and deploy services and applications, enabling a quick response to constantly changing requirements and scalable service delivery.

The importance of service discovery that can handle dynamic changes in the endpoint that uses services in containers and fast status changes such as start and stop containers is increasing. The use cases of service provided by containers are increasing, and the number of containers is expected to increase in the future. The service providers must manage information to use many containers and provide services that allow service users to use this information. In addition, container networking builds its network inside the container host, and its namespace is hidden from the existing namespace. However, there are many access requests from outside the host to services running inside the container. We also considered that it would be important to ensure the reachability of services from outside the host by running a lot number of containers to use lightweight containers inside the computer effectively. Therefore, transparent name resolution is required to find the service running inside the container from outside the host.

The purpose of this study is to provide a service discovery system that enables flexible external access to services running inside the container from

outside the host. Additionally, we will discuss methods that can be used in general and that do not depend on managed services provided by cloud service providers. As a derivative effect, it will be possible to access services running inside containers transparently without being aware of the location of the services.

In this study, we investigated networking technologies in container environments, represented by the bridge Network, and service discovery topics in various environments. As a result, we found that the container network inside the OS hosting the container uses a technology that translates source and destination Internet and transport identifiers, making it possible to access from inside the container to outside the host. However, it was difficult to discover the services running inside the container from outside the host. To solve this problem, we design a system that satisfies the following requirements.

(1) Discoverable the services provided inside the container from outside the host

(2) Design of service name space

(3) Can be used in a general-purpose container environment

To design a system that meets the above-mentioned requirements, this study examined an optimal design method for a service discovery system that can access services running inside containers. Then, we propose a service discovery system that combines a dynamic name resolution mechanism using a distributed database on the Internet and a mechanism that exposed to the outside the host information on endpoints for reaching services running inside containers from inside the host that provides the container. The service provider stores the Internet identifier and transport identifier in the database at the beginning of service delivery using the service in the container, and the service user accesses the service by referring to this information. To take the benefit of the name resolution mechanism on the Internet, the design of the namespace and the name service were adopted to be included inside the existing namespace to ensure the transparency of the name resolution. Regarding the methodology for the method of exposing the service endpoints, we adopted a method that does not add any additional implementation to the existing container implementation from the viewpoint of dissemination and transparency.

So as to confirm the operation of the proposed method, we conducted experiments assuming that the service is provided and that the service is used. In the experiments for service delivery, the proposed method was used to start a container and update information in a database. In the experiment for using the service, name resolution was performed using a SOCKS proxy to imitate the referring to information about the service, and

Web access was performed. As a result, we confirmed that it is possible to update information for using web services provided in the container and to use the services provided inside the container. From the above, we confirmed that transparent service discovery is possible by using the method proposed in this study.

As a result of this research, we showed the possibility of transparent service discovery without being aware of the network or service location. We believe that by upgrading the methodology proposed in this study, it will be possible to achieve service discovery that does not depend on the environment of the application managing multiple groups of containers and thus managing more containers. As items for future study, the need for extended implementation of name resolver APIs that perform name resolution and the quantitative evaluation of scalability was clear.