A Container-level Application Deployment Method for Micro-service Architecture

Lin Ma [a], Yong Liu and Ximei Du [b]

Henan University of Science and Technology, Luoyang 471000, China.

[a]machenglin2015@126.com, [b]dxmduximei@163.com

Abstract

With the rapid development of the Internet, the scale and the architecture of a monolithic application become so large and complicated gradually. Hence the application deployment with virtual machine (VM) is beconming more and more complicated and time-consuming and its energy consumption is relatively large, which reduces the system's utilization of resources. Aiming at solving energy consumption and time-consuming issues for application deployment, a container-level application deployment method for micro-service architecture is proposed in this paper, and the management mechanism of the overall architecture design and container deployment for micro-service system is also introduced to complete the deployment of container-level application. We use the current container technology to deploy applications by introducting the concept of micro-service in the Linux operating system. Then, the experimental results based on the OpenStack cloud platform prove that our proposed container-level deployment approach is superior to the traditional VM deployment approach in lowing energy consumption and low time-consuming, which not only addresses the issues of application deployment on high energy consumption and time-consuming to a certain extent, but also eases the monolithic architecture some problems, such as, poor maintenance and scalability utilization.

Keywords

OpenStack platform; Micro-Service architecture; Application deployment;Virtual machine(VM); Container technology.

1. Introduction

In recent years, with the rapid development of cloud computing [1] and mobile Internet [2], a large number of applications have been migrated to the cloud platform. As an important link in the application migration process, application deployment is becoming more and more energy consuming and time-consuming seriously [3]. In addition, the application migration process requires a large quantity of time and servers for support, and the longer time of opening servers also increases the energy consumption and time-consuming to a certain extent, such as, Barroso et al. using 5000 multiple services conducted a six-month test, and the statistics obtained the average server utilization of 11% to 50% [4]. On the one hand, in order to minimize the consumptions of energy and time of the application migration and improve the efficiency of the application migration, an application deployment has aroused concerns of many enterprises and professionals gradually. On the other hand, in order to minimize costs, the application virtualization technology is widely used in application deployment. Virtualization technology can not only meet the general performance requirements of the enterprise, but also can better protect the security of the system [5]. The traditional deployment methods mainly focus on virtual machine VMs. For example, Beloglazov et al. proposed a resource management system to continuously integrate virtual machines to save energy [6]. In order to optimize resources and reduce energy consumption, Beloglazov et al. proposed a self-adaptive threshold algorithm for dynamically integrating virtual machines is pro-posed[7]. Buyya et al.

proposed a virtual machine-based application model to save energy and refine resource allocation strategies [8]. Zhou et al. proposed a three-threshold energy-saving virtual machine deployment algorithm for Energy-Efficient Management [9]. Some methods, based on the VMs virtual machine to complete the deployment work, are also proposed [6-9]. And the virtualization technology can meet the general requirements of the enterprise, but the energy saving effect of the algorithm [6-7] needs to be improved, and the literature [8-9] considered less system performance. In addition, there is a common problem existing in the existing virtualization technologies, that is, there is a strong dependence on the underlying layer and a lack of independent use functions [10]. Therefore, the traditional deployment method has not been able to solve the problem of energy consumption and time-consuming.

With the rapid development of the Internet, the size of monolithic applications has become larger and larger. The large and complex applications increase the energy consumption of application deployment to a certain extent, which not only greatly reduces the resource utilization of application systems, but also brings users poor experience [11]. As for this challenge for architecting monolithic applications, micro-services architecture [12] decomposes a huge monolithic application into small, interconnected and service-specific services. And micro-services provides a solution for the low consumption of energy and time. The emergence of Docker [13] in virtualization provides us with another direction to solve the problem. Docker is an advanced LXC-based container engine that consists of an engine that manages lightweight containers, a client, and the AUFS file system. Docker's main application scenario[14] provide users with a simple "container" organization that can deploy a standard application environment quickly and automatically. Besides, Docker own less resource occupation and can start up application quickly [15], which can well meet the demand of deploying applications on low energy consumption and time-consuming. Docker creates a lightweight and portable container for every application [16] and is a good proponent of micro-services. Many micro-services use Docker container technology for their technical solutions.

In view of the above situation, this paper proposes a container-level deployment approach for micro-services architecture. The approach in this paper uses micro-services architecture as a framework and Docker is seen as a deployment container to implement effective deployment of applications. The main flow of the method is as follows: firstly, the application is deployed in a set of Docker containers of the micro-services architecture. Each Docker provides a specific service, and the services interact with each other in a service-oriented manner. Secondly, using container-level architecture design , the management mechanism of running container deployment, the running container application deployment algorithm and effective container deployment algorithm to improve the efficiency of the container; Finally, experiments conducted on the OpenStack cloud platform demonstrate that container-level deployment of applications is superior to traditional VM deployment applications in terms of low power consumption and low time-consuming.

This paper has two contribution points. Firstly, this paper uses the current popular container technology as a service carrier to complete the application deployment, which shorten the deployment time to some extent and solve the problem of the traditional virtual machine on slow response and large overhead [17]; secondly, this paper adopts the micro-services architecture. Under the concept of micro-services, we propose that give priority to running container, using the "write Dockerfile file, generate the destination image, pull the image deployment application" approach to complete the application deployment specific work, which makes deployment easier, faster and less energy intensive. In addition, the comprehensive experimental results also demonstrate that the deployment method we proposed is superior to the traditional method.

The rest of this paper is organized as follows. Section 2 discusses the work in this paper. Section 3 introduces our proposed the method of container-level application deployment by writing Dockerfile file. Section 4 tests and analyzes the start-up time and physical memory consumption of Docker and virtual machine VM under the same conditions on the OpenStack cloud platform. Then, the experiment results based on the OpenStack cloud platform prove that the container-level deployment approach is superior to the traditional VM deployment approach in low power consumption and low time-consuming.

2. Related Work

2.1 Docker Container Technology

Docker is an open source container engine [18]. Based on the Linux container technology, it further encapsulates some operating interfaces of the container to mamage and use containers for developers without special attention to the underlying complexities of Linux containers operating mechanism. Fig.1 is Docker's architecture.

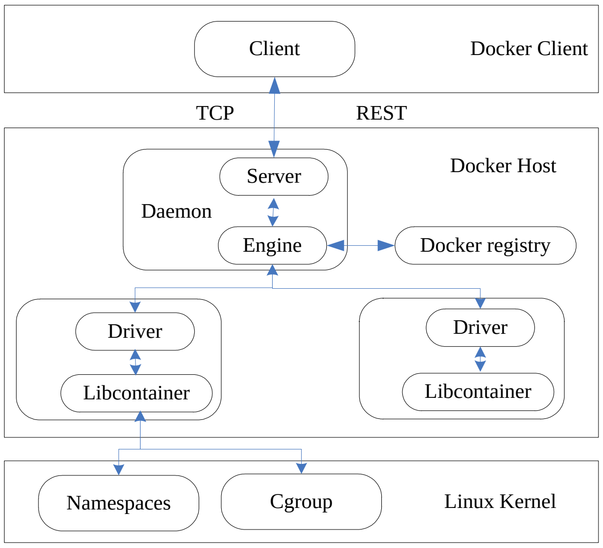

Fig.1 Docker's architecture

Docker's core elements are: image, container, daemon and registry. Docker image is the basis for building a container that can contain the complete operating system and the corresponding application software. Users can download or create the required image according to demand [19]; Docker container is based on the mirror to create a running instance, which provides services by runing the container. Docker containers can only use the resources within the specified range and the container can be seen as an isolated system platform that owns its own operating system [20]; Docker daemon is mainly composed of Docker Server, Engine and Job three parts, which is a resident process in the background of the daemon [21]; Docker registry is used to store the container image which is divided into public warehouse and private warehouse. Docker Hub is the largest shared repository that any user can download from a public repository. Private warehouses are generally used inside the LAN for personal and group and are available locally.

Docker shares portions of the host kernel and operating system instances. The core philosophy of Docker is to "build once and run everywhere" compared to traditional virtual machine VM-based virtualization. Fig.2 shows the comparison between virtual machine and Docker virtualization.

Docker is a revolution in virtualization technology. Docker can realize the efficient deployment and simplify the relocation of the operating environment caused by the migration of the server, which effectively improves the working efficiency while reducing the errors of redeployment. At the same time, the post-maintenance of Docker container is also simple and quick, reducing the entire project development cycle. Docker containers occupy less physical space, which can improve the utilization of the entire physical space resources [22].

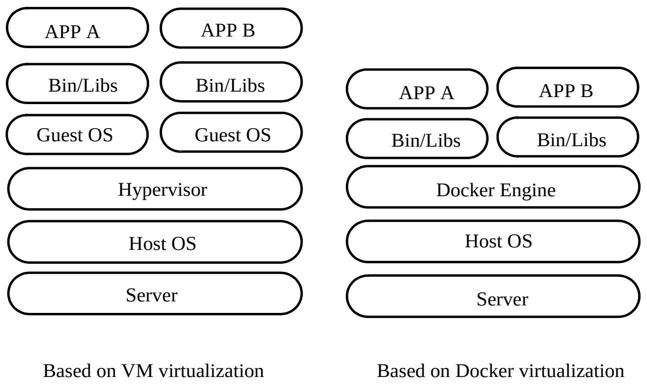

Fig.2 The Comparison Between Virtual Machine and Docker Virtualization

2.2 Micro-services Architecture

Micro-services architecture is a new technology for deploying applications and services in the cloud. In the micro-services architecture, each micro-service [23] can be deployed, scaled and tested independently and each micro-service performs only one specific function. When you need to add some of the features, you just need to add the features that you need for a particular service. In addition, in the micro-services architecture, different services can even be implemented in different programming languages and developed by independent teams [24]. Micro-services architecture [25] reflects the idea of Internet applications, having the following advantages: 1) Micro-services architecture is loosely coupling, which can provide more flexibility; 2) Micro-services can be targeted to solve specific problems through different programming languages and tools for development. 3) Each micro-service can be independently developed by different teams, without affecting each other and speeding up the market launch; 4) Micro-services architecture is a huge enabler for continuous delivery, which allows frequently release of different services while maintaining the availability and stability of the rest of the system.

3. The Method of Micro-service Application Deployment Locality

3.1 The Overall Design of a Micro-service System Deployment

The design of micro-service-oriented system is based on container in this paper. Its main idea is to deploy different micro-services in different Docker containers, and provides specific services to the system through containers. We should put different micro-services in different containers to develop using different languages. The system supports application deployment in different languages such as Java, PHP, Python, Node.js and so on. Fig.3 is the container-level overall design architecture of micro-service system which is proposed in this paper.

i) The entrance container is the total entrance facing the micro-service system, responsible for managing users, projects and so on.

ii) The master control container is responsible for scheduling operation of the container. When the user wants to deploy the application, the portal container notifies the master control container. The master control container allocates the operation container to the application to be deployed by the portal container.

iii) The runtime container is configured with the necessary environment which is used for compilation, operation and debugging of the application. The runtime container is created by different images. You can install different operating environments such as Java, PHP, Python, and Node.js in the runtime container as needed. Different runtime containers have different runtime environments at the same time. The master control container is responsible for unified scheduling and runtime environment Switch. Java application debugging environment here is seen as a special operating environment to configure, so as to achieve the unified management of the operation of the container.

iv) The flexible container has the same function as the container for operation, which is mainly used for debugging, ensuring the service quality and saving resources when the operation changes. The strategy of different containers flexibility is different.

v) Database records various types of system information: programmer's login, exit time, the current development environment, the current development of the project, the use of container resources.

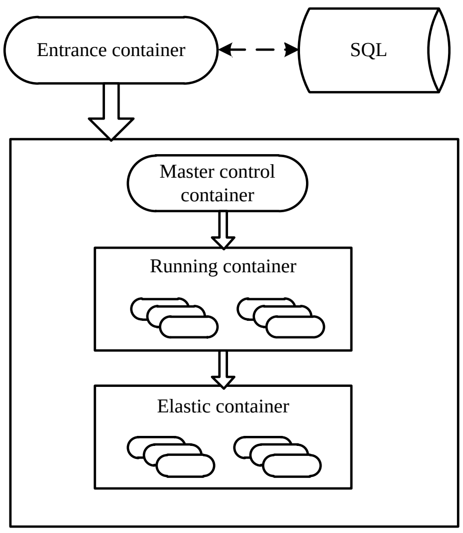

Figure 3. The container-level overall design architecture of micro-service system

3.2 Container Deployment Management Mechanism and Algorithm

Container deployment management is mainly for the management of running containers in this paper. Each running container can have different running environments required by different applications. Therefore, we need to manage the containers for different applications.

a) Run container initialization

We must open the appropriate server when deploy the application. And we will take some time in turning on the server each time. In order to save system deployment time, we divide applications into the more frequent use of the applications and the use of the average number of applications. For the more frequently used applications, reserved for the server, that is, the initial state of our application server is turned on. Set it to off for applications with normal usage frequency. Specific parameters are defined as follows:

Table 1. Run the container initialization parameter list

Runner_type	Container status	Specific settings
Use of higher frequency applications	Ready	Server is turned on
Use the frequency of the general applications Slave3	Ready 2	Server is turned off

b) Scheduling Management Mechanism and Algorithm for Operating Container Deployment Application

Based on the different states of container initialization, the strategy used by the application is also different when we allocate containers. According to the different types of container deployment applications, this paper proposes the following scheduling management strategies:

1) When deploy applications that is used frequently, we should first find the corresponding server opened but not deployed application container to deploy, and then finding the server whose status is turned off. If the container status is ready container, we should open such applications Server for the next deployment.

2) If 1) the search fails, we should find the server whose status is turned off. If the container status is ready container, we should open the appropriate server and deploy the application.

3) If 2) failed to find, we should find the server whose status is on. If the container status is ready container, we should first shut down the server that has been opened, and then open the server of the application type to deploy the application.

4) If 3) failed to find, it indicates that there is no free container and we should returns the message waiting for the resource.

5) When deploying a usage-neutral application, we should first turn on the appropriate server and confirm that the container status is ready, then, we should deploy the application.

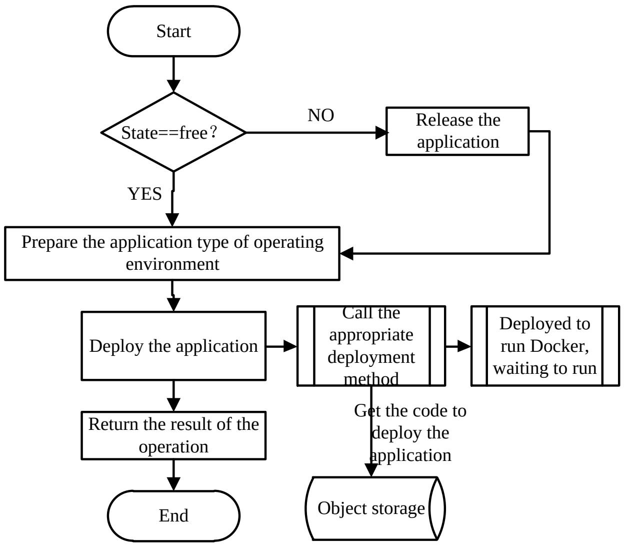
Start
State==free?
NO
Release the application
YES
Prepare the application type of operating environment
Deploy the application
Call the appropriate deployment method
Deployed to run Docker, waiting to run
Get the code to deploy the application
Return the result of the operation
End
Object storage

Fig.4 The algorithm flowchart of application deployment of a running container

Before applications are deployed, firstly, we should determine the container that deploy the current application using scheduling management policies. Secondly, we should start a series of deployments to the ready applications. In order to avoid applying the repeated deployment, we should determine the status of the container before the deployment. If the container is idle, we should begin to prepare the operating environment required for the application. And when deploying a runtime environment-ready application, the application code is obtained from the object storage and deployed in a container. Thirdly, the application runtime returns the operation result to the user when the application deployment is completed. Fig.4 shows the algorithm flow of running container when we deploy applications.

3.3 Micro-service Deployment of Specific steps

We devide the container-level deployment for the microservices architecture into three parts: creating a Docker container, preparation for system mirroring and application deployment. The preparation for system mirroring refers to the process of creating mirrors and saving the created mirrors to a local private mirror repository. The steps of deploying application are as follows: firstly, we use the docker image command displaying the image that stored in the public repository (DockerHub). Then we select the desired Docker image, and use the command “docker run” to start the container and enter the command line; Secondly, we should prepare Dockerfile file and the specific approach is: first specifying the mirror image based on the creation of the base image (ie: the parent mirror) and author information; second, installing necessary software based on the basic image; Then, using mirror operation instructions configure the relevant information; Thirdly, we use Docker to run dockerfile and generate the purpose of the image. Finally, we should save the target mirror in a local mirror repository and we should only pull it from the local repository when the target mirror needs to be deployed. Fig.5 is a flowchart of the application deployment process.

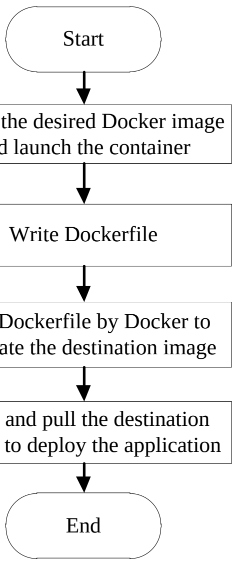
Start
Select the desired Docker image and launch the container
Write Dockerfile
Run Dockerfile by Docker to generate the destination image
Save and pull the destination image to deploy the application
End

Fig.5 Flowchart of the application deployment process

Dockerfile [26] is a script composed of several instructions. Docker generates a customized image by reading the Dockerfile. When you need to customize additional requirements, you should just add or modify instructions on the Dockerfile and regenerate the image [27].

4. Micro-Service Deployment of Specific Steps

4.1 Experimental Results and Analysis

To verify the rationality of the proposed deployment method, OpenStack [28] is used as a cloud platform to build a cloud computing environment consisting of 14 virtual machines. Docker is used as a virtual machine to create seven VMs and seven Docker virtual machine.

4.2 Environmental Design and Results

(1) Docker and VM boot time comparison

In order to test the average start-up time of Docker and VM, we tested the OpenStack platform by deploying seven VMs and seven Docker VMs at the same time. Using the method of the control variables [29], except for the Hypervisor [30], the other hardware conditions are the same. The specific conditions set in Table 2, Fig.6 for the comparison results.

Table 2. Experiment conditions setting table

Hypervisor	The purpose of the mirror	platform	Virtual machine configuration
Docker	Ubuntu14.4 purpose mirror	OpenStack	Single processor, 2G memory, 20G hard drive, Ubuntu14.4 operating system
VM	Ubuntu14.4 qcow2 mirror	OpenStack	Single processor, 2G memory, 20G hard drive, Ubuntu14.4 operating system

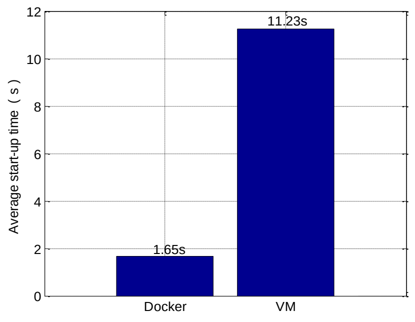

Fig.6 Average startup time for Docker and VM

The test found that the average startup time based on the Docker cloud platform is 1.65s and the average startup time based on the VM cloud platform is 11.23s. The data in figure 6 shows that the average launch speed of micro-services architecture based on Docker-based OpenStack platform is much faster than the average startup speed of micro-services architecture application under VM-based OpenStack platform. As a result, container-level deployment approaches for micro-services outperform traditional VM-based deployment methods in terms of average launch time for application deployments.

(2) Docker and VM physical memory comparison

In the container-level deployment approach to micro-services, Docker containers are used as a type of virtual machine. To test the physical memory consumption of two virtual machines, Docker and VM, this paper test the physical cost of opening the same number (up to seven) virtual machines and deploying the same micro-service application under the condition of table 2. VM are set to 2G memory in test. The test results are shown in Fig.7.

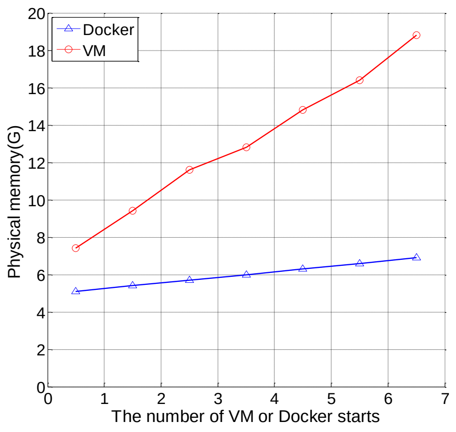

Fig.7 VM virtual machine and Docker container physical machine memory comparison

As can be seen from figure 7, when the same number of virtual machines are is powered on, the physical memory consumed by opening VM consumes more than the same amount of physical memory consumed by opening Docker by the same number of nodes. Therefore, this experiment proves that based on this method, Docker container has less resource consumption than VM virtual machine in terms of physical memory resources when deploying the same micro-service application.

The experimental results show that the proposed container-level deployment method for micro-services is better than the traditional VM-based deployment method in energy consumption and time-consuming.

5. Conclusion

In this paper, we propose a container-level deployment approach for micro-services, which improves the deployment efficiency compared with the traditional VM-based deployment. The method uses Docker containerization technology to achieve a low-power system deployment. Experimental results demonstrate that our proposed method is superior to the traditional method of virtual machine deployment.

References

[1] Lee, J., “A view of cloud computing”, Communications of the Acm, Vol. 53, No. 1, (2013), 50-58.

[2] Mehdizadeh, A., Mohammad poor, M. and Soltanian, Z., “Secured route optimization and micro-mobility with enhanced handover scheme in mobile IPv6 networks”, Internaional Journal of Engineering, Vol. 29, No. 11, (2016).

[3] Rezai, H. and Speily, O.R.B., “Energy aware resource management of cloud data centers”, International Journal of Engineering”, Vol. 31, No. 1, (2017), 1730-1739.

[4] Barroso, L.A. and Hölzle, U., "The case for energy proportional computing", Computer, Vol. 40, No. 12, (2007), 33-37.

[5] Asadi, F. and Hamidi, H., "An architecture for security and protection of big data", International Journal of Engineering, Vol. 30, No. 10, (2017), 1479-1486.

[6] Beloglazov, A. and Buyya, R., "Energy efficient resource management in virtualized cloud data centers", in Cluser, Cloud and Grid Computing, International Conference on , IEEE/ACM., (2010), 826-831.

[7] Beloglazov, A. and Buyya, R., "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers", in Middleware for Grids , Clouds and e-Science, Proceedings of the 8th International Workshop on, ACM., (2010), 1-6.

[8] Buyya, R., Ranjan, R. and Calheiros R.N., "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities" in High PERFORMANCE Computing & Simulation, International Conference on, IEEE., (2012), 1-11.

[9] Zhou, Z. and Hu Z., "Virtual machine deployment algorithm for reducing energy consumption in cloud environment", Journal of South University of South (natural Science Edition), Vol. 42, No. 5, (2014), 109-114.

[10] Ma, Y. and Huang, G., "Research on the virtualization of application software based on Docker ", Software, Vol. 36, No. 3, (2015), 10-14.

[11] Wang, L., "Parsing the micro-service architecture (a) monolithic architecture system and its challenges", (2015).

[12] Thönes, J., "Micro-services", IEEE Software, Vol. 32, No. 1, (2015), 116-116.

[13] Mao, M., "Auto-scaling to minimize cost and meet application deadlines in cloud workflows", in High Performance Computing, Networking, Storage and Analysis, International Conference on, IEEE., (2011), 1-12.

[14] Chung, M.H., Quang-Hung, N., Nguyen, M.T. and Thoai, N., "Using Docker in high performance computing applications", in Communications and Electronics, International Conference on, IEEE/CCE.,(2016), 52-57.

[15] Soltesz, S., Fiuczynski, M.E., Bavier, A. and Peterson, L., "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors", in Computer Systems, Sigops/eurosys European Conference on, ACM., (2007), 275-287.

[16] Matthias, K. and Kane, S., "What is Docker ?",(2016).

[17] Boettiger, C., "An introduction to Docker for reproducible research", Acm Sigops Operating Systems Review, Vol. 49, No.1, (2015), 71-79.

[18] Zhang, J. and Zhu, X., "Research on the Docker-oriented overlay network", Telecommunication Engineering Technology and Standardization, No. 9, (2015), 74-77.

[19] Chen, Q., Chen, C. and Zhang, Y., "Docker Technology Implementation analysis", Information and communication technology, No. 2,(2015), 37-40.

[20] Liu, S., Li, Q. and Li. B., "Study on the isolation of containers based on Docker technology", Software, Vol. 36, No. 4,(2015), 110-113.

[21] Zhang, Z. and Huang, B., "Docker application based on OpenStack cloud Platform", Software, No. 11, (2014), 73-76.

[22] Zhao, W., Yang, L. and Liu, H. et al., "The optimization of resource allocation based on process mining", Lecture Notes in Computer Science., (2015), 341-353.

[23] Richardson, C., "Micro-services architecture", International Journal of Open Information Technologies, Vol. 2, No.9, (2014), 24-27.

[24] Fowler, M. and Lewis, J., "Micro-services", (2014).

[25] Hao, T., Wu, H., and Wu, Q. et al., "Container-Level flexible resource supply method for micro-service architecture", Computer research and development, Vol. 54, No. 3, (2017), 597-608.

[26] Geng, M., Chen, M. and Wei, J., "The build tool of container mirroring for Dockerfile", Computer system applications, Vol. 25, No. 11,(2016), 14-21.

[27] Li, J. and Liu, G., "Research on automated container deployment of Hadoop distributed cluster", Computer application research, Vol. 33, No. 11, (2016), 3404-3407.

[28] Corradi, A., Fanelli, M. and Foschini, L., “VM consolidation: A real case based on OpenStack Cloud”, Future Generation Computer Systems, Vol. 32, No. 1, (2014), 118-127.

[29] Tepeš, B., Deur, J. and Kasać, J., “Optimization of control variables for automated transmission engagement”, in Control, Ukacc International Conference on, IET., (2013), 1-6.

[30] Szefer, J. and Lee, R.B., ”Architectural support hypervisor-secure virtu-lization”, in Architectural Support for Programming Languages and Operating Systems.