2023 7th International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS) | 978-1-6654-5606-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICMSS56787.2023.10117715

Architecture Design of Enterprise Information System Based on Docker and DevOps Technology

Tao Chen
KPMG Information Technology Services (Nanjing) Co., LTD
Nanjing, Jiangsu, China
1393899065@qq.comu

Haiyan Suo
Jiangsu Provincial People's Hospital
Nanjing, Jiangsu, China
Suohaiyan2014@163.com

***Abstract*—Under the background of large and medium-sized platform strategy, many excellent platform as a service architecture designs have emerged. The focus of the technical platform is to build the PaaS layer. The PaaS layer is designed to form a cluster environment with relatively distributed virtual machines provided by the IaaS layer, the main technologies to be used include Docker container and Kubernetes orchestration engine. In addition, software project development is a process of comprehensive practice, and it is far from enough to rely on Docker technology alone. In view of the lack of standardized project management, code management, testing and quality assurance means in software project development, the introduction of DevOps software development model and the study of enterprise information system architecture design are of great significance for guiding the design of large-scale distributed architecture.**

Keywords-Docker, Virtual machine, Container technology, Enterprise information system, DevOps, Distributed architecture

I. INTRODUCTION

Docker is one of the most popular Linux container solutions[1]. It is an open source application container technology developed based on Go language[2]. Its source code is hosted on GitHub. Docker is the cornerstone of PaaS platform, which integrates functions such as R&D, packaging, and application release[3]. At present, the mainstream solution for business development is microservice architecture, and container technology and container choreography technology are the technical basis for packaging and running microservices. Docker images run on the container engine[4]. It is lightweight and independent, including everything necessary to run container applications, such as system tools, running environment, code and settings. The container separates the application from its running environment[5]. No matter what the infrastructure of the container software is, it can run in Linux and Windows based operating systems and achieve the same running effect[6]. This study analyzes the principle of Docker technology, and gives the architecture design of enterprise information system based on Docker and DevOps technology.

II. DOCKER CONTAINER TECHNOLOGY

Docker is one of the most popular Linux container solutions[7]. It is an open source application container technology developed based on Go language, and its source code is hosted on GitHub. Docker integrates functions such as development, packaging, and application release, and is the cornerstone of the PaaS platform. The mainstream solution for business development is microservice architecture, and container technology and container choreography technology are the technical basis for packaging and running microservices.

Traditional business applications run directly on the physical server, and the operating system of the physical machine has no corresponding technical means to stably run multiple business applications[8]. In this scenario, when a new business application needs to be added, the purchasing department has to purchase a new server. The essential reason for this problem is that there is no effective way to judge the server performance required by new business applications[9]. If deployed on an old server, it may cause the old server to be paralyzed. If it is deployed on a new server, the model and specification of the purchased server can only be inferred based on experience[10]. The performance of new servers is often excessive, which increases the cost of the business department, and the utilization of server resources is often maintained at a very low level, wasting the company's server resources[11].

In order to solve this problem, virtual machine technology and container technology have emerged successively. Before the rise of container technology, services could only be installed in virtual machines. A physical machine can run multiple virtual machine systems, and each application is deployed in an independent virtual machine system to isolate applications through virtual machines. Because each virtual machine has an independent operating system and disk file system[12], which will monopolize some memory and hard disk space, an application will occupy dozens of MB of memory, but the virtual machine operating system will occupy dozens of GB, resulting in the virtual machine occupying too much physical machine performance resources. At the same time, if the virtual machine operating system has security vulnerabilities, patching will be extremely troublesome. At present, the mainstream virtual machine systems include VMWare, VirtualBox, KVM, Xen, etc[13].

In order to reduce the overhead of repeated deployment of the underlying operating system, all containers created by Docker share the same operating system, and the volume of containers is much smaller than that of virtual machines[14]. Docker deploys smaller image packages[15], faster startup, and less overhead on the physical machine operating system[16]. In addition, many commonly used middleware components have Docker image packages out of the box, which makes it easier to manage dependent files than virtual machines[17]. The running scenario of the virtualized and containerized applications is shown in Figure 1

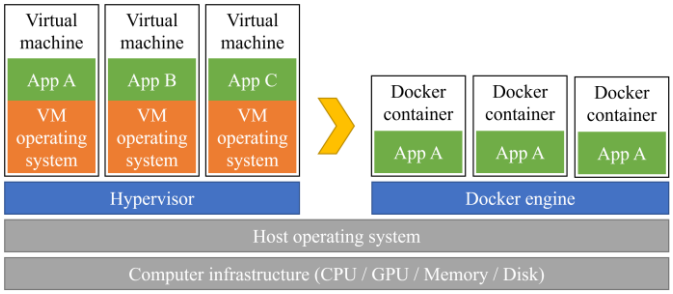

Fig. 1. This section describes the operating scenarios of VM applications and containerized applications

In Figure 1, the virtual machine transforms one physical server into three logical servers. Each virtual machine contains an operating system and applications. Virtual machines usually occupy tens of gigabytes[18]. The hypervisor allows multiple virtual machines to run on a single computer[19]. The running framework of the container application is shown in Figure 1. The container packages the code and dependencies into a binary file. Multiple containers can run concurrently on the same computer[20]. Each container runs as an isolated process in user space, sharing the operating system kernel. The space occupied by the container is usually dozens of MB, which is far less than the resources occupied by the virtual machine. In addition, the container requests fewer resources from the operating system during operation.

Docker containers do not need to consume additional dedicated operating system resources, and start up quickly. Docker packages all business applications into container images, which can be easily migrated between different operating systems. Docker container image is a lightweight, independent and executable software package, which includes everything needed to run applications: system tools, running environment, code and settings.

Due to the distributed characteristics of microservice architecture, each independent service is designed to tolerate the failure of other services, and the dependence between services is resolved through remote communication. Each service can communicate with processes on any physical space machine, and dependencies between services may fail. This requires developers to consider the elasticity of services when designing, rather than taking this requirement as a follow-up consideration. The use of container technology can largely solve the above problems caused by microservice architecture[21]. Deploying microservice applications to containers brings flexible deployment and portability to enterprise services. In the development, testing, online, maintenance, upgrade and other stages of the software life cycle, the mechanism of "write once, run everywhere" has been implemented.

III. DevOps Practice

DevOps, as a new paradigm, can achieve a high degree of coordination between development and IT operation and maintenance, so as to improve the reliability, stability, elasticity and security of the production environment while completing high-frequency deployment.

DevOps is a combination of development and operation and maintenance[22]. Software developers and IT operation and maintenance personnel work closely together and use automation tools to complete the construction, testing and release of software systems, so as to improve the frequency and quality of software release. The relationship between developers, testers, operation and maintenance personnel and Docker is shown in Figure 2.

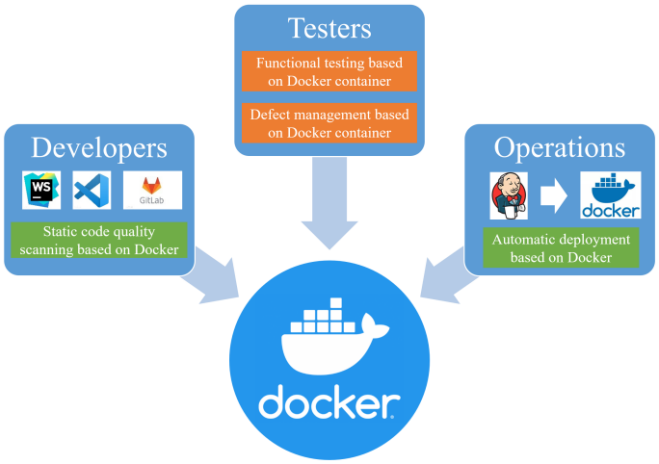

Fig. 2. The relationship between developers, testers, operations and docker

The emergence of DevOps makes it possible for multiple teams to work together. In the past, the development team was only responsible for writing code, the operation and maintenance team was only responsible for deployment and operation, and the operation and maintenance team did not participate in any development process. In this case, the operation and maintenance team has no say in the product launch plan, and does not need to pay attention to the function of product release, but continues to feed back problems in operation and maintenance to the development department, and constantly corrects problems in subsequent versions, which increases a lot of communication costs for both teams and greatly reduces the work efficiency. DevOps enables the operation and maintenance team to have an impact on the system design to improve the overall functional characteristics. At the same time, the development team assists the operation and maintenance department to successfully deliver the system and solve problems, with twice the result with half the effort.

The container technology makes DevOps architecture easier to implement. DevOps often requires multiple software implementations. Because the container is inherently isolated, different versions of software can run on the same host without affecting each other. Compared with virtual machines, it starts faster and takes less resources. In addition, the packaging method of the container can achieve a high degree of unity between the software version dependencies of the development environment and the production environment, and the packaged images are easy to distribute. The relationship between DevOps and Docker is shown in Figure 3.

As shown in Figure 3, DevOps can be divided into 10 phases, including 6 closely related to Docker: Development, Build, Deploy, Test, Security test, and then the Development phase. Continuous integration is the core engineering practice

of DevOps, which runs through the whole life cycle of software development. The adoption of appropriate DevOps tools can improve the success rate of the implementation of DevOps, and play a multiplier role in the successful implementation of the transformation of DevOps. The appearance of containers makes it relatively easy to implement DevOps.

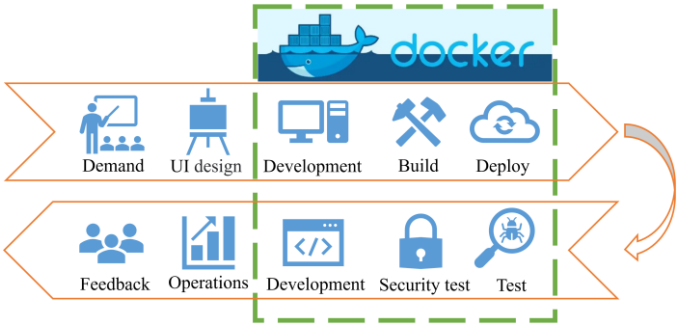

Fig. 3. The relationship between DevOps and docker

DevOps software development method advocates better communication, collaboration and integration among development, operation and maintenance and quality assurance teams, so that high-quality software products can be delivered more quickly. The process includes continuous development, continuous integration, continuous deployment and continuous monitoring. Among them, continuous integration and continuous deployment of automation are the core, and a complete automation tool chain is the essential foundation. The impact of Docker and DevOps on the organization is shown in Figure 4.

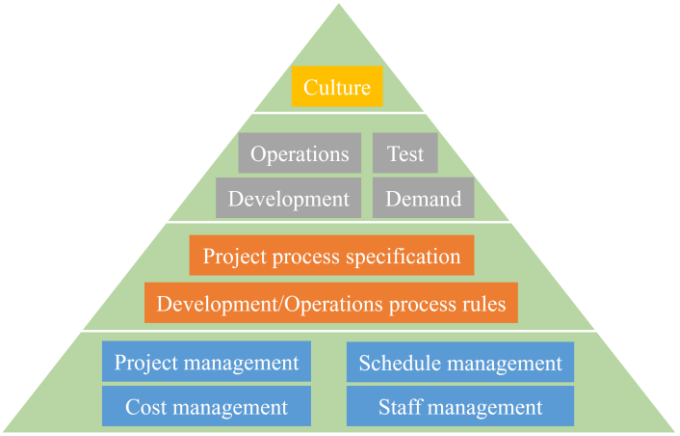

Fig. 4. The impact of Docker and DevOps on organizations

As shown in Figure 4, the initial impact of Docker and DevOps on the organization is reflected in the management methods, including the management of Project, Schedule, Cost and Staff. It then affects the project process specification and the development/deployment process specification. Finally, it affects team members and organizational culture. Through the construction of DevOps project, it emphasizes the communication, cooperation and respect between teams. In terms of organization and culture, everyone can reach an agreement on the goal. Everything aims to deliver valuable services faster and better.

IV. Enterprise Information System Architecture Design

After practicing DevOps platform, this research designs enterprise information system architecture based on Docker and DevOps technology. First of all, the business system needs to be transformed into a container so that business applications can flexibly adapt to changes in the external environment and achieve loose coupling between services. The design of containerized microservice architecture becomes the focus. The concept of microservice is aimed at ordinary services. It divides ordinary services of a single application. These divided small services coordinate and cooperate with each other to provide users with complete service value. For different business scenarios, each service is deployed and built independently. For the microservice architecture, it is necessary to select the appropriate language and tools according to the context of the business process, so as to build a specific service. Compared with the single overall model, the containerized microservice architecture has better scalability, is easy to deploy and manage, and saves costs.

The microservice runs in the virtual running environment provided by the container. DevOps simplifies the process of container creation, integration, deployment, operation and maintenance. With the help of DevOps technology, containerized microservices can easily implement distributed deployment. This research takes enterprise information system as an example to explore the practice of distributed container application. Based on Docker technology, containerized application architecture is shown in Figure 5

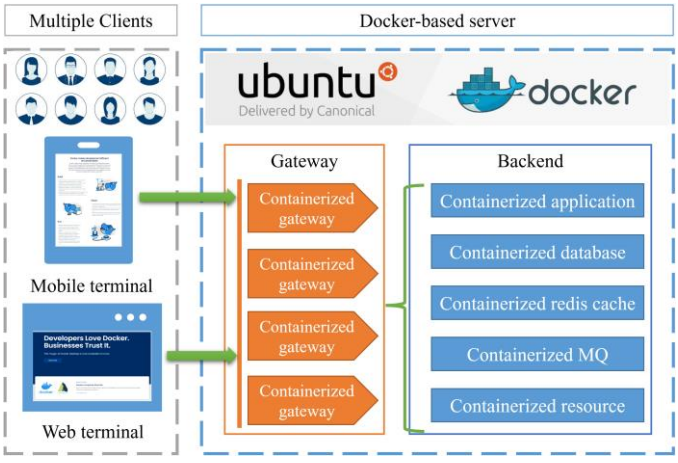

Fig. 5. Containerized application architecture

As shown in Figure 5, taking the enterprise information system as an example, the left side indicates that multiple users use different terminal devices to send requests to the server. The right side represents the server. The server has installed the ubuntu operating system and deployed a container gateway based on Docker technology. The gateway implements distributed clustering and load balancing. The container gateway is mainly used as the entrance of all mobile terminals and Web terminals in the distributed architecture to receive all Web requests. Gateway implementation includes routing and forwarding, gateway security protection, identity authority authentication, protocol adaptation, flow control and other functions.

The back-end consists of multiple container applications, including application, database, Redis, MQ, resource and other business containers. The core of the back-end architecture is to divide large applications into multiple non overlapping small independent services. These mutually independent services rely on lightweight communication mechanisms to achieve communication, and achieve rich application functions through the coordination between microservices. Containerized microservices have high cohesion, low coupling, easy scalability and high autonomy, and are easy to upgrade and expand horizontally. Each individual microservice can be developed, managed and deployed independently as an independent module. Back-end containers can also easily realize distributed deployment through DevOps technology.

In the distributed container architecture, the back-end microservices are deployed independently. There is no strong correlation on deployment between microservices, and they usually rely on microservice communication. Microservice communication refers to how to efficiently and accurately realize information interaction and message transmission between microservices in the process of network transmission. It is generally divided into synchronous and asynchronous communication. Synchronous communication generally uses Thrift and REST protocols, without message middleware, and is suitable for simple application scenarios. Asynchronous communication usually adopts message middleware and MQ protocol to realize flexible interaction of messages, which is suitable for relatively complex application scenarios. Generally, the distributed architecture will use these two communication methods to achieve system scalability and availability.

High availability is another important challenge in the design of distributed system architecture. Its goal is to reduce the time when the system cannot provide services through some designs. The biggest bottleneck and challenge of high availability of enterprise information system lies in the single point, that is, the computing and storage resources in the system are concentrated on one node. The way to avoid a single point of high availability is clustering. Even if a node is down and unavailable, the service can be migrated to other backups.

Therefore, in order to ensure high availability of the system, the core criterion of distributed architecture design is redundancy. The goal of redundancy is to overcome the single point bottleneck at various levels shown in the figure above. Of course, redundancy means that the cost increases. Reasonable redundancy design needs to constantly explore the balance between cost and stability. In addition, although redundancy ensures backup during fault handling, it is not expected that manual intervention is required to restore the system process every time a fault occurs. Therefore, some mechanism is needed to ensure the automation of failover to achieve high availability of the system.

After the enterprise information system is deployed online, the last stage is operation and maintenance. The system is maintained and monitored during user use. The data visualization generated in this process is a very important basis and guarantee for the operation and maintenance team to respond in a timely manner. The system uses the popular web-based distributed operation and maintenance monitoring tool Zabbix to monitor the system status, logs, container status, and container application logs, flexibly notify the operation and maintenance personnel, and quickly locate and solve existing problems.

V. Conclusion

In the traditional enterprise information system application mode, the application service is closely bound to the server, and the operation efficiency is limited by the hardware performance of a single server. The overall performance of the information system needs to be further improved. Container technology represented by Docker and container deployment technology represented by DevOps are the main thrust of the new round of cloud technology revolution that has arisen in recent years. By deploying enterprise information systems, distributed container applications are realized, and multiple application servers are formed into a cluster for external services. Docker and DevOps technologies are fully used. Based on the DevOps mode, the enterprise information system is combined with the transformation of containerized microservices. The DevOps platform provides scheduling strategies to schedule various application services in the form of containers on demand, distributed and run on multiple servers to maximize the use of server resources, thus improving the performance of the information system.

The distributed gateway deployed in this study can process user requests concurrently, improving the overall processing capacity. The deployed distributed background microservices can process data with high concurrency, and middleware such as database, cache and message queue can be deployed in a distributed manner.

The enterprise information system built by the distributed architecture technology proposed in this study runs stably after being put into operation. The distributed container architecture design reduces the difficulty of enterprise information system development and the risk of later operation and maintenance. The application of Docker virtualization technology makes the deployment of microservices more flexible, fast and efficient. Use DevOps technology to encapsulate business service components into container images to achieve containerization of each service component. Deploy and manage container clusters through DevOps system to provide feasible technical solutions for enterprises to build large distributed system.

References

[1] Lin Cai et al. “Improving Resource Usages of Containers Through Auto-Tuning Container Resource Parameters” IEEE Access (2019): n. pag.

[2] Lele Ma et al. “Efficient Live Migration of Edge Services Leveraging Container Layered Storage” IEEE Transactions on Mobile Computing (2019): n. pag.

[3] Wajdi Hajji and Fung Po Tso. “Understanding the Performance of Low Power Raspberry Pi Cloud for Big Data” Electronics (2016): n. pag.

[4] Shai Barlev et al. “Secure yet usable: Protecting servers and Linux containers” Ibm Journal of Research and Development (2016): n. pag.

[5] Christoph Jansen et al. “Curious Containers: A framework for computational reproducibility in life sciences with support for Deep Learning applications” Future Generation Computer Systems (2020): n. pag.

[6] Pasquale Salza and Filomena Ferrucci. “Speed up genetic algorithms in the cloud using software containers” Future Generation Computer Systems (2019): n. pag.

[7] Qichen Chen et al. “Design of an adaptive GPU sharing and scheduling scheme in container-based cluster” Cluster Computing (2019): n. pag.

[8] B. K. Tripathy, Kumaran K., M. Sumaithri, and T. Swathi, "Enhanced Rule Induction Using Incremental Approach fora Dynamic Information System," International Journal of Computer Theory and Engineering vol. 3, no. 4, pp. 509-515, 2011.

[9] P. Tanuska and T. Skripcak, "Data-Driven Scenario Test Generation for Information Systems," International Journal of Computer Theory and Engineering vol. 3, no. 4, pp. 565-572, 2011.

[10] Geetha Sivaraman, V. Lakshmana Gomathi Nayagam, and R. Ponalagusamy, "Intuitionistic Fuzzy Interval Information System," International Journal of Computer Theory and Engineering vol. 4, no. 3, pp. 459-461, 2012.

[11] Mohd Nazri Ismail, "Development of WAP Based Students Information System in Campus Environment," International Journal of Computer Theory and Engineering vol. 1, no. 3, pp. 266-271, 2009.

[12] Edouard Bugnion et al. “Hardware and Software Support for Virtualization” (2017).

[13] Lei Chen et al. “Research on Virtualization Security in Cloud Computing” IOP Conference Series: Materials Science and Engineering (2020): n. pag.

[14] Yudha Alif Auliya et al. “Performance Comparison of Docker and LXD with ApacheBench” Journal of Physics: Conference Series (2019): n. pag.

[15] Lin Cai et al. “Improving Resource Usages of Containers Through Auto-Tuning Container Resource Parameters” IEEE Access (2019): n. pag.

[16] John Paul Martin et al. “Exploring the support for high performance applications in the container runtime environment” Human-centric Computing and Information Sciences (2018): n. pag.

[17] Weibei Fan et al. “A Live Migration Algorithm for Containers Based on Resource Locality” Journal of Signal Processing Systems (2019): n. pag.

[18] Josip Balen et al. “Performance evaluation of windows virtual machines on a Linux host” Automatika (2020): n. pag.

[19] Belen Bermejo and Carlos Juiz. “Virtual machine consolidation: a systematic review of its overhead influencing factors” The Journal of Supercomputing (2020): n. pag.

[20] Roger Baig et al. “Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons” Future Generation Computer Systems (2018): n. pag.

[21] Hao Zhang et al. “Application and Practice of Microservice Architecture in Multidimensional Electronic Channel Construction” Journal of Physics: Conference Series (2019): n. pag.

[22] V. Arulkumar and R. Lathamanju. “Start to Finish Automation Achieve on Cloud with Build Channel: By DevOps Method” Procedia Computer Science (2019): n. pag.