

Assignment

by Xueyang Hu

Submission date: 10-Apr-2023 07:43PM (UTC+0100)

Submission ID: 201820309

File name: 41181_Xueyang_Hu_Assignment_488474_1215300514.docx (1.46M)

Word count: 5978

Character count: 34129

**ACCA7027 – CLOUD ENGINEERING
PRINCIPLES AND PRACTICE**

ASSIGNMENT



<200549> - <Xueyang Hu>

UNIVERSITY OF WALES TRINITY SAINT DAVID - Wales Institute of Science and Arts - School
of Applied Computing

Table of Contents

1.0	Introduction	2
2.0	<i>Docker</i>	2
2.1	<i>Introduction</i>	2
2.2	<i>The benefits of containerisation</i>	3
2.3	<i>Protability</i>	4
2.4	<i>Image and publishing image</i>	5
2.5	<i>Container volumes</i>	5
2.6	<i>Docker network</i>	7
3.0	<i>Main conception in cloud computing</i>	8
3.1	<i>Application gateway</i>	8
3.2	<i>Load balancing and scalability</i>	9
3.3	<i>Example: Cloud Game</i>	11
4.0	<i>Main driving forces of cloud computing's expansion</i>	12
4.1	<i>Main driving force</i>	12
4.2	<i>Serverless computing</i>	12
5.0	Conclusion	13
	Bibliography	14

1.0 Introduction

With the development of network transmission technology and the increasing demand for large-scale computing, cloud computing is widely applied as a cheap large-scale computing solution. Qian L et al. [1] believe that cloud computing is a computing technology that provides IT services through IP network connections and massive low-cost computing units. It typically integrates a large amount of computing resources by service providers and allocates them to different users dynamically. Armtrust M et al. [2] emphasize the components of cloud computing in their concept: it is a collection of applications for services, integrated hardware and system software that provide these services. Dillon T et al. [3] extracted five characteristics of the U.S. NIST 's definition of cloud computing architecture: on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. These features enable cloud computing to have advantages such as low cost, high scalability, and high generality. However, they also bring some problems. For example, it's difficult to ensure information security, and hard to determine charging standards, etc. Early cloud computing platforms include Amazon's AWS, Google's GAE, and Microsoft's Azure. With the development of cloud computing technology and the expansion of demand, there are more and more cloud platform service providers, such as Ali Cloud and Oracle Cloud. The application range of cloud computing is also increasing, such as online training of large-scale neural network models, cloud games, and so on.

This article is divided into three parts: The first part will introduce Docker and explain how to use it. Docker is an open source application container engine that allows user to deploy applications to the cloud easily. The second part will explain three concepts in cloud computing: application gateway, load balancing and scalability, and use an example to illustrate how they can their benefits in cloud service. Finally, the article will introduce the concept of serverless function and analyze the advantages and expansion reasons of cloud computing.

Demonstration video link: https://uwotdsd-my.sharepoint.com/:v:/g/personal/2200549_student_uwotdsd_ac_uk/ESZxxsfBAX5GivnpH2bpiMtBX_2gU3s33gjdOr6Pd3qlcg?e=QQtgQY

2.0 Docker

2.1 Introduction

During the process of program development and running, the discontinuity of interfaces between programming language, framework, architecture in the runtime environment creates great difficulties for the entire process. Docker is an advanced solution to this problem. Docker is an application running platform that simplifies the environment building process. It isolates applications and the environment through containerisation, making applications independent of each other and be able to be executed on the same operating system kernel at the same time[4]. It integrates many methods to simplify the process of building environment, such as virtualization, component reuse, version control, etc. Docker has the advantages like easy to use, move fast, and allow users to collaborate with teams through DockerHub. This gives it a wide range of applications.

The convenience of Docker on environment construction makes it applicable in many scenarios. The two common applications are the construction of cloud computing platforms

and code reproduction. Compared to other virtualization products, Docker has advantages such as fast deployment of applications, lightweight, and high portability, which make it costs less and often be used in environment and application deployment in cloud[5]. At the same time, it also provides great convenience for the reproduction and use of code. Carl believes that the reproduction of [6] code faces four technical challenges: difficulty of reproducing dependent platforms and libraries, inaccuracy of environment building documents, change of dependent library versions, and researchers' fatigue in learning various dependent libraries. The Docker platform can solve most of these problems. In terms of document description, Dockerfile provides a standardized and compileable description method, which makes code installation and operation easier. What's more, image version is a good way of library version control.

2.2 The benefits of containerisation

Before introducing containerization, the concept of virtual machine need to be known. This concept was proposed by IBM in the 1960s. It is designed for concurrent access of physical machine. A virtual machine is a copy of a computer's underlying system, which can exist multiple instance on a single physical machine and each one is completely isolated from each other. Users can execute, develop, and test applications on a virtual machine just like using a real physical machine, without worrying about system crashes caused by other users on the same physical machine. It presents multiple operating environments simultaneously on a physical machine through methods such as partitioning computing resources, simulation, and time sharing[7]. This was one of the main ways for computers to reduce the cost of hardware in the early days, and it is still being applied today. However, the operating system and running environment in each virtual machine are independent of each other, which makes this technology still not lightweight enough. In order to lower hardware costs, the concept of containerization has been proposed.

A container is a standard software unit that packages code and all its dependencies, enabling applications to quickly and reliably be migrated to different computing environments[8]. It has characteristics like resource isolation and allocation mechanism which is similar to the virtual machine, but it virtualizes the operating system instead of hardware, making it more lightweight. Each virtual machine needs to have their own operating system, applications, libraries and other content, while containers can share the same operating system kernel. Therefore, they occupy less space than virtual machine. According to the research of AM Potdar et al. [9], the container's performance is better than virtual machine in CPU performance, memory throughput, disk IO, load testing, running speed and other aspects. This also makes it used more widely today.

Docker, as one of the most well-known containerization providers, its technology has become an industry standard. The container in Docker is created from image by Docker Engine as a process, which not only has high isolation security and lightweight characteristics, but also has strong platform universality [8,10]. It can be deployed on servers with Linux or Windows operating systems as the kernel and is adopted by all major data centers and cloud service providers.

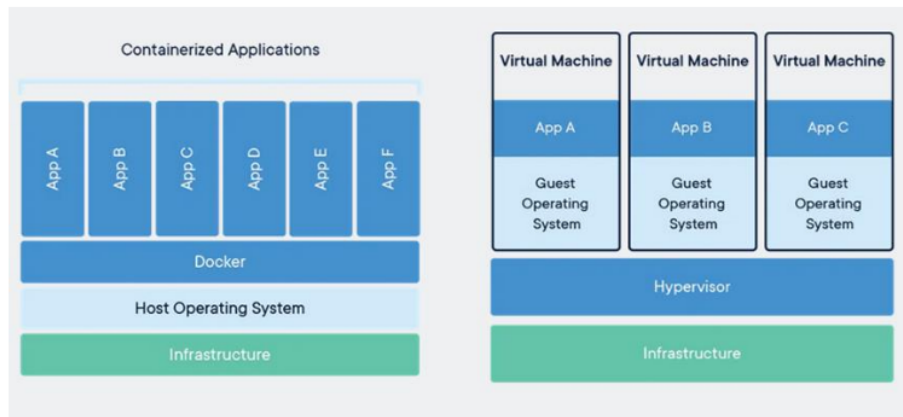


Figure 1 - The architecture's comparison between Docker's container(left) and Virtual Machine(right)[8]

2.3 Protability

Docker's Portability refers to the ease of porting applications built within Docker's container. Docker can easily deploy the same program image on servers on different platforms and supports dynamic modification. This high portability is built on the simplicity of building the environment. Docker use Dockerfile to build images, which can provide same container for programs on different platforms. The relationship between Dockerfile, Image, and container is like the relationship between code, executable programs, and processes, and the cost of only transferring code to different platforms and compiling it is very low. This high portability provides convenience for developers to develop, test, and publish multi-platform programs.

Dockerfile is a text file with a certain structure, usually seen as a standardized description of an image. The most basic Dockerfile usually consists of four parts: FROM, RUN, EXPOSE, and ENTRYPOINT. FROM is used to declare the basic images required in new image. These basic images are usually from DockerHub, an online code and image data hub. RUN is used to declare the commands that run when the container is built. EXPOSE is used to declare the container's open ports for applications running in it. ENTRYPOINT is used to declare the commands that the container needs to execute each time it starts. In addition to these four basic parts, there are also many other command blocks with different usage, like CMD can be used to declare additional commands to be executed during startup, ADD and COPY can be used to add files to the image file system, and WORKDIR can be used to declare the default command execution directory in container. These blocks together form the Dockerfile. Dockerfile is usually placed in the creation directory of the image and compiled by Docker Server through the command '**Docker image build -[tag] name**' to generate the final image.

The underlying structure of the Dockerfile compilation result, which is also called image, is designed based on the theory of the Advanced Multi-Layered Unification Filesystem(AUFS) in Linux. It can be seen as an hierarchy structure. The bottom layer is the base image, and each commands in Dockerfile corresponds to a temporary image layer, where each layer is built on top of the previous layer. all layers are read-only. When a container is started from

the image, the Docker engine will generate a container layer from the top layer of image. This structure brings Docker advantages such as fast deployment and saving memory space.

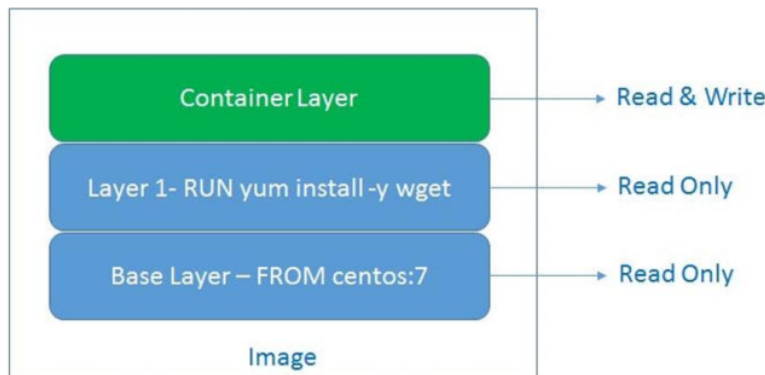


Figure 2 - Docker Hierarchy Diagram [11]

2.4 Image and publishing image

Image is a template used to create a container, which describes the content of the container. Images usually come from three sources: copying images from existing containers, local image templates, and compiling Dockerfiles created by user. Due to the fact that it does not include the operating system kernel (the containers share kernels in the Docker server), its size is very small and it runs quickly. Usually, '**Docker container run [- option] imagename**' is used to run the image as a container process, and the container can use option parameter to execute operations like running in the foreground or background, using virtual terminal of the container or deleting the image after running, etc. Multiple different containers can be created by the same image, but they must have different interface numbers. The state of the container can be changed through commands such as stop, restart, pause, etc.

DockerHub is the world's largest container image repository, with content from various developers, open source projects, and software vendors. Users can use it freely to download public images, store and publish images, set access authority, and subscribe to private image libraries through DockerHub [12]. It has advantages such as high automation, high security, and rich community content. Sharing images through DockerHub can significantly shorten code deployment time, thereby improving the efficiency of the development team. Images can be pulled from online registry through the '**Docker image pull [- options] imagename**' command, or pushed to the registry through the '**Docker image push [- options] imagename**' command. Users can also meet the needs of version control by specifying the image version number to be pulled or pushed.

2.5 Container volumes

As mentioned before, the image in Docker is composed of multiple read-only layers stacked together. When modifying a file in the container, the modification will only exist in the top-level read/write layer which is generated temporarily, and will not be propagated to the read-only layer below, meaning that this modification will not be reflected in the image. Therefore, when the container is deleted and restarted with the same image, all previous modifications will be lost. In order to persistence the data in the container, Docker proposes the concept of volume. It realizes the persistence of data in the container by dividing a space on the host

machine for storing files and mount it on containers, so that users can back up data in the container, and data can be shared between containers. By using the command 'Docker volume create voluename', an area can be partitioned in the host space and can be managed by docker. By using volume name, data from this area can be loaded into different containers.

There are two mounting methods for volume, one is direct volume mounting, which maps files and directories in the host machine directly to the container. The lifecycle of volumes and containers is independent of each other, and when a container is deleted, its mounted data volumes will not be deleted or recycled. The '**Docker container run -name containername -v voluename/targetdirpath: accessrights**' command allows the container to be mounted with volume. The powerful functionality of DockerCLI allows users to specify the location where the volume is mounted in the container and the access authority to the volume.

Another method is mounting volume containers. Containers can mount a volume-mounted container through the parameter '**--volumes -from [containername]**' when creating a container, and this volume-mounted container is called volume container. This is a simple way to achieve data sharing between containers. When the volume container stops working, it will not be immediately recycled, but will only be recycled after all the container which mounted this container stop working.

If the container use the real directory of the host machine instead of volume name during the mounting process, it is called a bind mount. The host directory for bind mount is not managed by the docker because it has not been registered in the docker. This instruction is usually used for development rather than software release, as the actual directory of different hosts varies.

The following is a practical presentation to demonstrate the process and function of using volume: first, create a document named volume_for_assignment in the Docker.

```
C:\Users\86139\Desktop\docker test>docker volume create volume_for_assignment
volume_for_assignment
```

Figure 3 – command line window—create volume

Through the command ls, it can be seen that the volume has been successfully created.

```
C:\Users\86139\Desktop\docker test>docker volume ls
DRIVER      VOLUME NAME
local       0afd9bebbba75ea3bdd8c7aeelbbbaa013f92d4d979e3606924840eb527e33936d
local       06c7e41e4409e8ac1d0d2b9bf007464132b2b7147ddadd2945c26494e3b19650
local       my_volumeA
local       volumel
local       volume_for_assignment
```

Figure 4 – command line window—show volume list

Next, mount the created volume to a new container with ubuntu as the image. Launch the virtual console of the container, enter the location where volume is mounted in the container, create a new file hello.txt, and exit.

```

C:\Users\86139\Desktop\docker test>docker run -itd -v volume_for_assignment:/usr/dir --name mycontainer ubuntu:latest
aa70174fd04be9ad69d822d98503f138dbf50da03fc825e42f0526a7e026b375

C:\Users\86139\Desktop\docker test>docker exec -it mycontainer bash
root@aa70174fd04b:/# cd /usr/dir
root@aa70174fd04b:/usr/dir# touch hello.txt
root@aa70174fd04b:/usr/dir# ls
hello.txt
root@aa70174fd04b:/usr/dir# exit
exit

```

Figure 5 – command line window—connect volume to container

Stop the container and restart it, then enter the directory again. Through the `ls` command, it can be seen that the file `hello.txt` is still in the directory and does not disappear with the container stopping. This shows that volume has the ability of persistence the data in the container.

```

C:\Users\86139\Desktop\docker test>docker container stop mycontainer
mycontainer

C:\Users\86139\Desktop\docker test>
C:\Users\86139\Desktop\docker test>docker start mycontainer
mycontainer

C:\Users\86139\Desktop\docker test>docker exec -it mycontainer bash
root@aa70174fd04b:/# cd /usr/dir
root@aa70174fd04b:/usr/dir# ls
hello.txt
root@aa70174fd04b:/usr/dir# exit
exit

C:\Users\86139\Desktop\docker test>

```

Figure 6 – command line window—restart container

2.6 Docker network

The IP of the Docker container will change after each restart, so the containers cannot communicate continuously using IP addresses. The Docker network is proposed to solve this problem. It provides a network for communication between containers and a way for containers to communicate with external networks. Even if the container IP changes, the container can be accessed directly through its name. User can create a user-defined Docker network by using the command '**Docker network create networkname**'. The subnet IP of the network can be specified through the parameter '**--subnet subnetIP/mask**'. When creating a container, the '**-- network networkname**' parameter can be used to connect the container to the specified Docker network.

Docker network has five drivers: host, container, none, bridge, and overlay. Before explaining the characteristics of each driver, it is necessary to first explain Docker's resource isolation mode. Docker uses Linux's namespace technology to achieve resource isolation, and the same applies to the network. Each network namespace provides an independent network environment, including network card and routing, etc. In host driver, the container and its host share a network namespace and communicate with the external through the host's IP and port. In container driver, the container will share a network namespace with existing containers, rather than sharing it with the host. The containers in this net can communicate through IO network card devices. In none driver, each container will have its own

independent network namespace, but there will be no settings for its network. In fact, the container in this driver's network is equivalent to turning off the network function.

Bridge driver is Docker's default network creation driver and is also the most commonly used driver. In this driver, each container has a separate network namespace and is connected to a virtual bridge. The bridge works like a physical switch. The container is equivalent to be placed in a secondary network. It communicates with the external network through the switch and host. Docker will assign a different IP subnet from the host to each bridge driver network from the private IP network segment, and the IP of the container is generated from network's subnet segment. The bridge driver allows containers connected to the same network to communicate with each other, which can be set by configuring the parameter '--icc'. At the same time, user can also use parameters '--ip_forward' and '--iptables' control communication between containers and between containers and the external network.

The overlay driver allows docker containers on different hosts to communicate. Multiple hosts can be connected through a distributed network called swarm. When a host joins a swarm, two new networks are created internally: an overlay network responsible for communicating with the external network and a bridge network connecting the internal container.

Docker will have three default networks after installation. Their driver are host, none, and bridge. User can set the driver by using '-d drivename' when creating a network.

3.0 Main conception in cloud computing

3.1 Application gateway

Application gateway is an application running on a firewall system between two networks. It is equivalent to a proxy, and when a client requests the service, its request is redirected to the application gateway, which then creates a connection with the service provider behind the firewall on behalf of the client and executes the operation[13]. This process typically establishes two links, between the client and proxy server where the application gateway exists, and between the proxy server and the service provider. In this way, the application gateway can monitor the data packets sent by the client, thus protecting the computer behind the firewall from common web attacks, such as SQL injection and session hijacking. In addition, it can also provide functions such as load balancing and session retention.

Taking the application gateway of cloud service provider Azure as an example, the definition of the application gateway on the Microsoft Azure official website is a web traffic load balancer that enable user to manage traffic to user's web application[14]. Unlike traditional load balancers that work at the transport layer, Application Gateway provides load balancing and routing based on the seventh layer, that is, the application layer. It can make routing decisions based on the attributes of HTTP requests, such as passing data streams to different specific server pools based on the attributes of URLs. In addition, it also provides many other useful functions. For example, the encryption and decryption function of SSL/TLS. The traditional method requires encryption and decryption on the web server side, which brings huge overhead to the web server. The Azure Application Gateway can decrypt the data transmitted from the client on the proxy server side and send the decrypted data to the web server. Conversely, it can also encrypt data transmitted from the web server and send it to the client [15]. This greatly reduces the burden on the web server side. In addition,

Azure's application gateway can also provide functions such as autoscaling and multiple-site hosting.

Despite the various advantages mentioned above, compared to other firewall technologies, this technology requires additional server resources, which is undoubtedly more expensive[13].

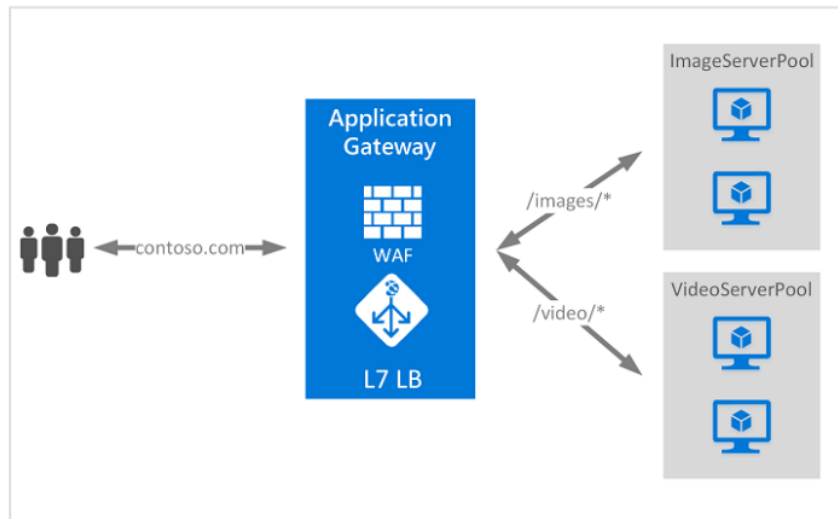


Figure 7 – How application gateway works in Azure[14]

3.2 Load balancing and scalability

Software vendors often deploy their software on multiple servers to provide instant services for the software. However, when the traffic volume is high, the load imbalance between these servers may lead to issues such as long service response time. And this is why load balancers appear. Load balancing is a technique of allocating incoming traffic among available servers in order to respond to user requests faster [16]. It is usually deployed in front of the resource server. After receiving user requests, it will construct different request queues based on the burden of the request to the server, and use different algorithms to select the appropriate server for the request based on the monitored server status. The benefits of load balancing include improving resource utilization and availability, reducing excessive supply of infrastructure, and so on. The selection strategy includes Least-Connection Load Balancing which select the least connected server, and Round-Robin Load Balancing which choose the server in a cyclic order, etc. Its working modes include Static Load Balancing which uses fixed rules for selecting servers, and Dynamic Load Balancing based on current traffic and load conditions to determine the allocation order, etc.

Nowadays, most software suppliers deploy their software on the cloud on order to save costs, so load balancing is particularly important in cloud services. For the convenience of users, cloud providers have developed a business model similar to SaaS, PaaS, and other service models, called Load Balancing as a Service (LBaaS). This service should have the characteristics such as low resource demand, real-time monitoring of the latest server load information, ensuring the reliability of network communication, etc. In addition, it also needs to consider its pricing model, service level agreements (SLAs), and service architecture for multiple users, etc[16].

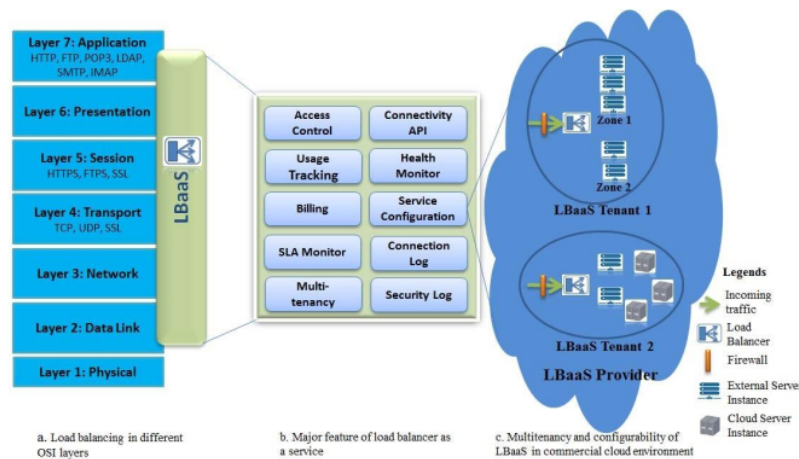


Figure 8 – How LBaaS works[14]

Taking Azure's load balancer as an example, Azure provides a load balancer working in the fourth layer of the OSI model, which is responsible for allocating front-end traffic to the back-end server resource pool in a reasonable way[17]. It has two modes: public load balancer and internal load balancer. The former's front-end uses a public IP, mainly used for load balancing when external users interact with internal networks; The latter front-end uses private IP, mainly for load balancing when interacting among internal networks. In addition, Azure's load balancer can also provide information and advice on traffic and other aspects through Azure Monitor, and can also provide visual results of these information through Insights for Azure Load Balancer.

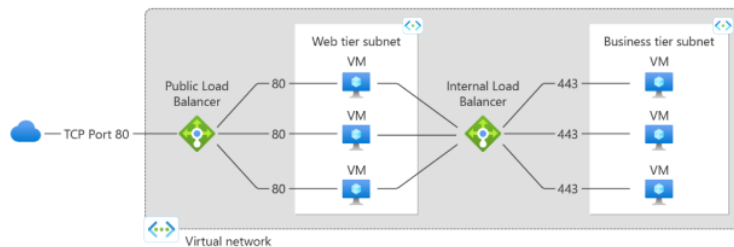


Figure 9 – How public load balancer and internal load balancer work[17]

The resources of cloud computing are usually priced based on the duration and scale of use, and the scale of required resources can dynamically change for some applications, such as streaming media platforms with different traffic at different time periods. In order to reduce costs while ensuring user experience, cloud computing platforms provides automatic provisioning mechanisms to expand or reduce the amount of computing resources occupied based on the resource utilization rate. This is the scalability of cloud computing. Scalability can be defined as the ability of a specific system to adapt to problems as their size changes [18]. Scalability is reflected in various aspects of the system, such as space scalability, which requires the system to be able to change its resource size when the workload changes, so that it can remain within an acceptable range. What's more, structural scalability requires the system framework to allow for dynamic changes in the number of received service requests. Sometimes, the enhancement of scalability of one aspect may decrease the scalability of another aspect, such as the need to monitor resource usage for space scalability, which

means an increase in system load and can affect the system's load scalability. How to balance the scalability of various aspects is also a problem that needs to be considered when designing cloud computing frameworks.

Cloud scalability typically has the following requirements for cloud platforms [18]. Virtualization, which means the ability to manage hardware resources in a software manner, thereby reducing management costs. This requires unified hardware standards; Resource sharing, which means different applications and organizations can share a portion of computing resources to achieve the goal of maximizing resource utilization; Dynamic configuration, which means resources can be provided on demand, requires monitoring of traffic and automated decision-making. There are many cloud platforms implement cloud scalability through Warehouse-Scale Computers (WSCs). WSCs are equivalent to a data center, which uses a unified hardware and software platform and uses a collection of computing units to replace a group of independent servers. These resources are deployed with different application through the public management layer.

Taking Azure's VM Scale sets as an example [19], it provides management capabilities for applications running across multiple VMs. It allows users to create and manage a set of VMs, and the number of VM instances can be increased or decreased according to user needs. This allows users to perform large-scale computing work and can improve resource availability and application resilience by distributing VMs across regions.

Load balancing and scaling together ensure the maximum utilization of resources. Scaling is responsible for planning the scale of resource, while load balancing is responsible for planning the allocation of traffic to ensure the full utilization of resources. What's more, both require monitoring the size of traffic and the usage condition of computing resources. Therefore, in many cloud computing service providers, the two services are bundled together. This is also why the two are complimentary concepts.

3.3 Example: Cloud Game

Cloud games are applications that remotely render interactive games in the cloud. Players interact remotely with the application through the client, which sends players' input to the cloud. Then the cloud application calculates the logic based on the input and renders the corresponding image, and returns it to the client in the form of a video and displays it to the player [20]. It provides a good solution for players who want to play high-quality games but have insufficient computing resource on their devices, but there are still many problems, such as high bandwidth requirements and delayed interactions.

Cloud game applications have the characteristic of significant changes in the number of users between different time periods, which requires that the amount of resources they occupy can dynamically change according to request number. Scaling and Load Balancer can ensure dynamic changes in the quantity of resources while maximizing utilization. At the same time, the fact that resources can be distributed across different regions in Scaling also shortens the latency of interaction, as players also come from different region. What's more, its resource intensive and unified format makes large-scale computing for game rendering possible. The Application Gateway can also play a role in balancing load. In addition, cloud games require a large amount of data transmission, and there are various user identities, which makes their servers more vulnerable to attacks. The Application Gateway can isolate computing resources and clients and control the access according to the user's identity, thereby protecting computing resources from damage.

4.0 Main driving forces of cloud computing's expansion

4.1 Main driving force

There are three main driving force of cloud computing's expansion: cost, flexibility, and security.

Cost is the main driving force behind the expansion of cloud computing. In traditional IT enterprises, they need to build their own data centers, which involve the cost of purchasing infrastructure equipment, planning sites, and equipping specialized IT personnel to manage the data centers. The cost of such construction is often very high. Moreover, some enterprises' IT services have low latency requirements, which require the enterprise to build their data centers in a distributed manner. At the same time, these servers are connected through long-distance links and have high communication costs. Using cloud computing services can significantly reduce these costs. Compared to enterprise data centers, cloud service centers have the advantage of economies of scale, that is, they can deploy a large number of servers in the same region and rent them to different users. Meanwhile, due to its large-scale data center architecture, cloud service centers can replace traditional servers with unified computing units, which reduces the cost of infrastructure. Finally, due to automation requirements, cloud service centers require fewer IT personnel than typical data centers. The ratio of IT personnel to servers in the data center of general enterprises is 1:100, while cloud service centers can reach 1:1000, which reduces the cost of maintaining cloud service centers. Therefore, although the upfront costs may be high, these factors can allow cloud service providers to rent servers to users at a lower price [21].

Flexibility is a major feature of cloud computing services and one of the main driving forces for their expansion. As mentioned earlier, users can dynamically change the number of cloud VM they use based on their software needs. Meanwhile, cloud services such as Load Balancer also ensure that the utilization of all VM can be maximized. This flexibility is very useful for software providers such as streaming platforms with significant changes in traffic over different time periods, as it can reduce costs while preventing service crashes caused by insufficient computing power. The growth in the number of such software has also led to the expansion of cloud computing services.

Security is one of the main issues faced by cloud computing. On the one hand, due to the limited participation of users in cloud service management, some commercial institutions find it difficult to trust third-party cloud service providers and are therefore unwilling to completely transfer digital assets to cloud servers. On the other hand, due to the multi-user leasing architecture, data uploaded by one institution may be accessed by another institution. This leads to the suspicion among cloud service users. These security issues hinder the expansion of cloud computing [22]. However, with the development of cloud computing technology, some protocols and systems for cloud security have been proposed. For example, the Advanced Cloud Protection System (ACPS) has an interceptor module that can detect suspicious activity on the host and store it in the recorder module. At the same time, it also has functions such as using IP table detection for network security, preventing malicious virtual machines and data attacks [23], protecting the security of cloud platforms from all aspects. The improvement of cloud computing security is also one of the main driving forces for cloud computing's expansion.

4.2 Serverless computing

Serverless computing refers to a programming model that executes code fragments directly in the cloud to run application without considering how to manage or control the infrastructure. In this model. Most of the operational issues of the server, such as scalability and maintenance, are left to the cloud provider, and users only need to use the API provided by the cloud provider to run the code and build the application [24]. Under this approach, cloud providers charge based on the time period and resources used by users to execute code, rather than based on the number of servers rented, etc. From the user's perspective, this approach can avoid the cost of server environment setup and resource management, just considering how to design the application; From the perspective of cloud providers, this approach provides an opportunity to control the entire development foundation, thereby they can reduce operational costs through efficient optimization.

Taking Azure's Azure Function as an example [25], it can provide end-to-end development without worrying about infrastructure management. By using the Function extension on Visual Studio, program construction and debugging can be carried out locally, and can be directly deployed to the cloud on a large scale. In addition, reports on current program performance and suggestions for improvement can also be obtained through Azure monitor. Azure Functions define the way functions are called through triggers and connect resources by declaring bindings. AWS Lambda is also a computing service that implements the Serverless function [26]. This service only needs to write and upload the code as a zip file or container image to execute, and can optimize execution time and performance based on the characteristics of the function itself.

Compared to containers, Serverless Function has a higher degree of decoupling in programming. The container requires the entire application to be uploaded to the cloud and then executed in a deployed specific environment, while the Serverless Function decomposes the application into multiple functions and runs them periodically in a event driven way without the need for deploying environment. From this perspective, the development logic of Serverless is undoubtedly simpler and has stronger scalability in the cloud. However, due to the higher level of programming, it is also more difficult to observe and debug. In most conditions, containers are more suitable for building long-time running programs, while serverless is suitable for responsive applications that do not need to run all the time.

5.0 Conclusion

This article studies the platform and related principles of cloud computing. Firstly, research was conducted on the Docker platform, exploring its containerization, portability, image, volume, etc. Then, three major concepts in cloud computing were elaborated, including application gateway, load balancing, and scalability. Finally, the main driving forces of cloud computing were discussed, and the serverless function was introduced in detail.

Bibliography

- [1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," *Lecture Notes in Computer Science*, pp. 626–631, 2009.
- [2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "[PDF] A view of cloud computing: Semantic scholar," *Commun. ACM*, 01-Jan-1970. [Online]. Available: <https://www.semanticscholar.org/paper/A-view-of-cloud-computing-Armbrust-Fox/dba46ff18ed8e96497ebe197611536a8128e7386>. [Accessed: 03-Apr-2023].
- [3] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010.
- [4] "Why Docker," Docker, 21-Feb-2023. [Online]. Available: <https://www.docker.com/why-docker/>. [Accessed: 03-Apr-2023].
- [5] D. Liu and L. Zhao, "The research and implementation of cloud computing platform based on Docker," 2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014.
- [6] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.
- [7] Chiueh S N T and Brook S, "A survey on virtualization technologies," *Rpe Report*, vol. 142, 2005.
- [8] "What is a container?," Docker, 21-Feb-2023. [Online]. Available: <https://www.docker.com/resources/what-container/>. [Accessed: 04-Apr-2023].
- [9] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, "Performance evaluation of docker container and virtual machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.
- [10] "The industry-leading container runtime," Docker, 21-Feb-2023. [Online]. Available: <https://www.docker.com/products/container-runtime/>. [Accessed: 04-Apr-2023].
- [11] A. Prakash, "Understanding docker layers and caching," *GeekyAnts Tech Blog*, 16-Jun-2022. [Online]. Available: <https://techblog.geekyants.com/understanding-docker-layers-and-caching-1>. [Accessed: 04-Apr-2023].
- [12] "What is Docker Hub?," Docker, 21-Feb-2023. [Online]. Available: <https://www.docker.com/products/docker-hub/>. [Accessed: 04-Apr-2023].
- [13] W. Staff, "What is application gateway?," *Webopedia*, 24-May-2021. [Online]. Available: <https://www.webopedia.com/definitions/application-gateway/>. [Accessed: 06-Apr-2023]
- [14] Greg-Lindsay, "What is Azure application gateway," *What is Azure Application Gateway | Microsoft Learn*. [Online]. Available: <https://learn.microsoft.com/en-us/azure/application-gateway/overview>. [Accessed: 06-Apr-2023].

- [15] Greg-Lindsay, "Azure application gateway features," Microsoft Learn. [Online]. Available: <https://learn.microsoft.com/en-us/azure/application-gateway/features>. [Accessed: 06-Apr-2023].
- [16] M. Rahman, S. Iqbal, and J. Gao, "Load balancer as a service in cloud computing," 2014 IEEE 8th International Symposium on Service Oriented System Engineering, 2014.
- [17] Mbender-Ms, "What is azure load balancer? - azure load balancer," Azure Load Balancer | Microsoft Learn. [Online]. Available: <https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview>. [Accessed: 07-Apr-2023].
- [18] J. Cáceres, L. M. Vaquero, L. Roderio-Merino, Á. Polo, and J. J. Hierro, "Service scalability over the cloud," Handbook of Cloud Computing, pp. 357–377, 2010.
- [19] Ju-Shim, "Azure Virtual Machine Scale sets overview - azure virtual machine scale sets," Azure Virtual Machine Scale Sets overview - Azure Virtual Machine Scale Sets | Microsoft Learn. [Online]. Available: <https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview>. [Accessed: 07-Apr-2023].
- [20] R. Shea, Jiangchuan Liu, E. C.-H. Ngai, and Yong Cui, "Cloud gaming: Architecture and performance," IEEE Network, vol. 27, no. 4, pp. 16–21, 2013.
- [21] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud," ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 68–73, 2008.
- [22] M. Ali, S. U. Khan, and A. V. Vasilakos, "Security in cloud computing: Opportunities and challenges," Information Sciences, vol. 305, pp. 357–383, 2015.
- [23] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," Journal of Network and Computer Applications, vol. 34, no. 4, pp. 1113–1122, 2011.
- [24] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, "Serverless Computing: Current Trends and open problems," Research Advances in Cloud Computing, pp. 1–20, 2017.
- [25] "Azure functions – serverless functions in computing: Microsoft Azure," – Serverless Functions in Computing | Microsoft Azure. [Online]. Available: <https://azure.microsoft.com/en-au/products/functions/#content-card-list-oc803c>. [Accessed: 10-Apr-2023].
- [26] 2 Y. 37, "AWS Lambda," Amazon. [Online]. Available: <https://aws.amazon.com/cn/lambda/>. [Accessed: 10-Apr-2023].

Assignment

GRADEMARK REPORT

FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

Docker Overview + Benefits (20): 14

Docker Examples (30): 19.5

Video (5): 5

Cloud Comp tech (25): 21.5

Cloud Comp functions (20): 17

Feedback:

=====

Good description of docker and benefits. The examples demonstrated the fundamentals, could have provided a little more interesting example.

Discussion of load balancing, etc., is quite good but a bit more technical discussion could have been provided.

Nice discussion of the factors driving the expansion of the cloud.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16
