

Zhiqiang Geng-2200520-Cloud Engineering Principles and Practice Assignment

by Zhiqiang Geng

Submission date: 28-Apr-2023 09:03AM (UTC+0100)

Submission ID: 203814596

File name: 40743_Zhiqiang_Geng_Zhiqiang_Geng-2200520-
Cloud_Engineering_Principles_and_Practice_Assignment_488474_1049346395.docx (3.95M)

Word count: 8759

Character count: 48984

**ACCA7027 –CLOUD ENGINEERING
PRINCIPLES AND PRACTICE**

**THEORY AND APPLICATION OF
CLOUD TECHNOLOGY**



Student Number : 2200520 Name : Zhiqiang Geng

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

Table of Contents

1.0 Introduction.....	3
1.1 virtualization technology.....	3
1.1.1 Classification of virtualization technology	3
1.2 Containerization technology and docker.....	4
1.2.1 container technology	4
1.2.2 Docker	5
1.2.3 Advantages of using containerization technology	6
2.0 Docker	7
2.1 Common commands for using image	7
2.2 Using a Dockerfile to customize the image.....	8
2.2.1 Build a Redis image using Dockerfile	9
2.3 Docker Repository and image push	10
2.3.1 Push the local image to dockerhub.....	10
2.4 data management.....	10
2.4.1 Volume	11
2.4.2 Create data volume.....	12
2.4.3 mount data volume.....	12
2.4.4 Delete data volume	13
2.5 Use the network.....	13
2.5.1 external access container.....	14
2.5.2 connection between containers	14
2.5.3 Create a Docker network to complete container interconnection	15
2.5.4 Docker Compose	16
3 cloud computing	18
3.1 load balancing technology	18
3.1.1 http redirection load balancing	18
3.1.2 IP Load Balancing	18
3.1.3 Reverse proxy load balancing	18
3.1.4 Kubernetes load balancing technology.....	18
3.2 scalability.....	19
3.2.1 How Kubernetes achieves scalability	20
3.3 Application Gateway	20
3.3.1 Azure Application Gateway.....	21

3.4 Why load balancing and scaling are complimentary concepts.....	21
3.5 The benefits of combining the three technologies.....	21
3.6 serverless functions and a containerized approach	22
3.6.1 case comparison	23
3.6.2 summary	25
4.0 Conclusion	25
Bibliography	27

1 Introduction

Here we mainly introduce the related concepts of virtualization technology, containerization technology and Docker, and the benefits of using containerization technology or using Docker can bring us convenience.

<Youtube Link: <https://youtu.be/DfZ3UqyRV0Y>>

1.1 virtualization technology

Before explaining Docker, let's talk about virtualization technology. Virtualization technology is the theoretical basis of Docker and the purpose of Docker's function.

Virtualization technology is designed to solve the problem of seamlessly migrating written programs to other machines for normal execution. In the actual production environment, different types of CPUs produced by different manufacturers, different instruction sets, different hardware, running different operating systems, and different shared link libraries may cause the program to run abnormally, making it difficult to migrate the program directly to other machines. Virtualization technology, which enables programs to run seamlessly on different machines, is the best way to solve this problem. Virtualization technology abstracts the CPU, storage, network and other entity resources in a computer and presents them to applications in a unified form [1]. Programs running in virtual environments use the resources provided by virtualization programs and do not touch the real environment. By virtualizing programs to create exactly the same virtual environment, applications can run exactly the same as before when migrating to other machines. Figure 1-1 shows the distributed storage implemented through virtualization [1].

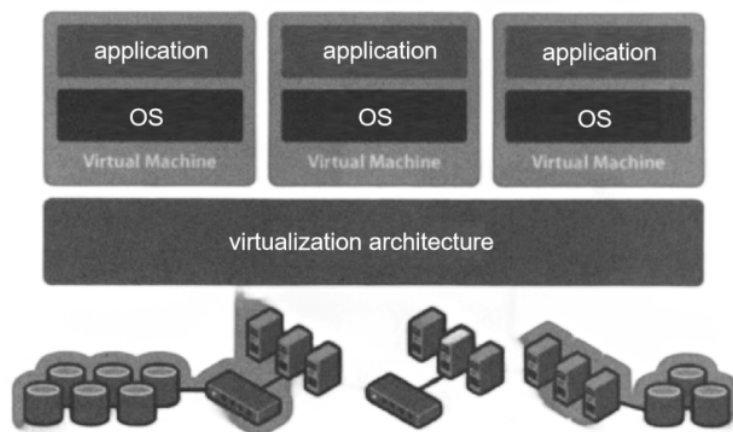


Figure 1-1 Virtualization Realization of Distributed Storage

1.1.1 Classification of virtualization technology

Virtualization technologies are mainly divided into the following categories:

1.CPU virtualization: Use a specially optimized instruction set to control the virtual process and improve performance.

2.Server virtualization: Simplifies management and improves efficiency by prioritizing resources and allocating server resources to workloads that need them most, whenever and wherever possible, reducing the amount of resources reserved for individual workload peaks.

3.Storage virtualization: Using large-scale RAID subsystems and multiple I/O channels to connect to servers, intelligent controllers provide LUN access control, caching, and other management functions.

4.Network virtualization: The integrated devices form a logical unit, which is represented as a network element node in the network. Management and configuration are simplified, and cross-device links can be aggregated, which greatly simplifies the network architecture and enhances redundancy and reliability.

5.Application virtualization: including application software virtualization and desktop virtualization.

1.2 Containerization technology and docker

Container is the core technology that makes up Docker, and it plays a pivotal role in Docker.

1.2.1 container technology

In the previous section we talked about a variety of virtualization classifications, the most representative of which is the virtual machine implemented through the Hypervisor. However, although virtual machines have achieved great results in cross-platform deployment, the huge organizational structure And low efficiency has been hindering its development, therefore, another branch of virtualization - container technology has emerged. Container virtualization technology is an important part of virtualization and the basis for the survival of Docker. We must focus on it [2].

Container technology is to package applications into individual containers. Through this packaging process, container technology isolates each program and breaks the dependencies and connections between programs. In other words, with the support of container technology, a huge service system can be composed of containers in which many different applications reside. This process of dismantling and recombining minimizes the coupling between applications. The external interface of the container is the same as the connection slot between the containers, which makes the process of assembling a system using the container and the programs in it extremely simple and convenient [3]. In Figure 1-2, we can see the main difference between platform virtualization and container virtualization.

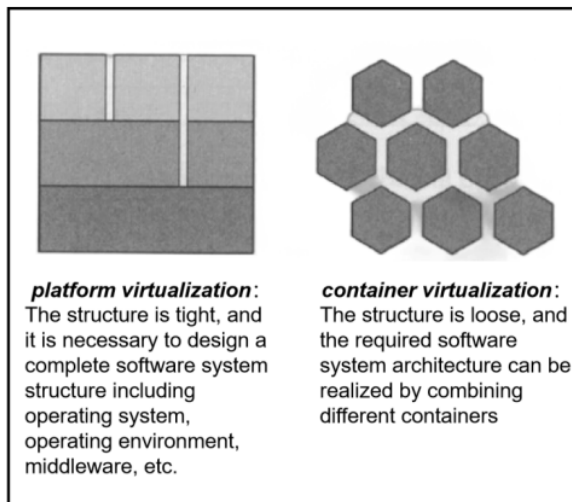


Figure 1-2 Comparison between platform virtualization and container virtualization

1.2.2 Docker

Since Docker was open-sourced, it has attracted a lot of attention and discussion. So far, its GitHub project has more than 46,000 stars and more than 10,000 forks. The popularity of the Docker project even prompted dotCloud to change its name to Docker at the end of 2013. Docker was originally developed and implemented on Ubuntu 12.04; Red Hat has supported Docker since RHEL 6.5; Google also widely uses Docker in its PaaS products [4].

Docker uses the Go language launched by Google for development and implementation. Based on technologies such as cgroup, namespace, and AUFS-like Union FS of the Linux kernel, Docker encapsulates and isolates processes, which belongs to the virtualization technology at the operating system level. Because the isolated process is independent of the host and other isolated processes, it is also called a container.

Docker has further encapsulated on the basis of containers, from file system, network interconnection to process isolation, etc., which greatly simplifies the creation and maintenance of containers. This makes Docker technology lighter and faster than virtual machine technology. Figures 1-4 and 1-5 below compare the differences between Docker and traditional virtualization methods. The traditional virtual machine technology is to virtualize a set of hardware, run a complete operating system on it, and then run the required application process on the system; while the application process in the container runs directly on the kernel of the host, and the container does not have its own kernel, and there is no hardware virtualization. Therefore, containers are more portable than traditional virtual machines [5].

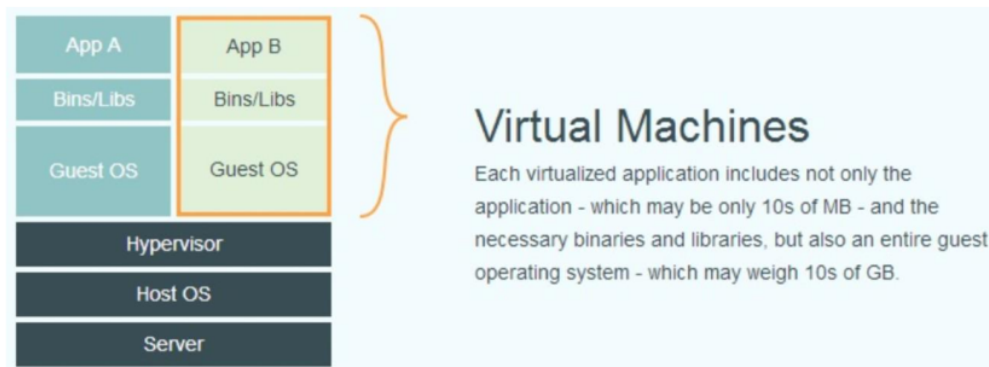


Figure 1-4 Traditional virtualization technology



Figure 1-5 Docker

1.2.3 Advantages of using containerization technology

Containers fundamentally change the way people develop, distribute, and run software. Software developers can build software locally, knowing that it will run on any host environment, whether it's an IT department's computer room, a user's laptop, or a cluster in the cloud, with no difference in runtime [5]. Operation and maintenance engineers only need to focus on maintaining the network and resources to ensure uptime, reducing the time spent on configuring the environment and resolving system dependencies.

A container is an encapsulation of an application and its dependencies. At first glance, a container is just a lightweight virtual machine. Like a virtual machine, it has an isolated operating system instance to run applications.

But containers have advantages that enable them to enable use cases that are difficult or even impossible to achieve with traditional virtual machines.

- ① The container can share resources with the host operating system, so its efficiency is an order of magnitude higher. Starting and stopping containers takes just a split second. Compared with running programs directly on the host, the performance loss of containers is very low, or even zero.
- ② Containers are portable, which is very likely to completely solve problems caused by slight changes in the operating environment, and may even completely stop developers' complaints: "But the program works normally on my computer!"

③The container is lightweight, which means that developers can run dozens of containers at the same time, and can simulate the situation of a distributed system in a real operating environment. The number of containers that an operation and maintenance engineer can run on a host far exceeds when only using virtual machines.

④For end users and developers, the advantages of containers are not only reflected in cloud deployment. Users can download and execute complex applications without spending a lot of time on configuration and installation issues, and without worrying about changes to the system itself. On the other hand, application developers no longer have to worry about differences in user environments and whether dependencies are satisfied.

What's more, virtual machines and containers have different fundamental goals. The purpose of a virtual machine is to completely emulate another environment, while the purpose of a container is to make an application portable and include all dependencies.

The following table 1-1 is a comparison chart of container technology and virtual machine technology

Table 1-1 Summary of comparison between docker and traditional virtual machines

Characteristic	container	virtual machine
Start Time	second level	minute level
Hard disk usage	Typically MB	Usually GB
performance	close to native	weaker than
System Support	A single machine supports thousands of containers	Generally dozens

2 Docker

2.1 Common commands for using image

Here are a few commonly used commands.

① The command to obtain the image from the Docker image warehouse: **\$ docker pull**. Its command format is:

\$ docker pull [option] [Docker Registry addr[:port]]/Repository-name[:tag]

②Run container: **\$ docker run**. Related parameters.

-i: interactive operation;

-t Use an interactive terminal.

--rm: Remove the container after it exits (usually also requires dockerrm).

ubuntu: This refers to using the ubuntu image as the basis to start the container, and other images can also be used as the basis.

bash: After the image name is the command, use bash, an interactive shell.

③ List the top-level image: **\$ docker image ls**, if you need to see the middle layer image, use **docker image ls -a**;

You can also list only the specified layer images: **\$ docker image ls xxx**.

④ Check the space occupied by images, containers, and data volumes: **\$ docker system df**

⑤ If you want to delete the local image, you can use the **\$ docker image rm** command, the format of which is:

\$ docker image rm [option] <image1> [<image2> ...]

Among them, <image> can be the image short ID, image long ID, image name or image summary.

2.2 Using a Dockerfile to customize the image

It can be understood that the customization of the image is actually to customize the configuration and files added by each layer. If we can write each layer of modification, installation, construction, and operation commands into a script, and use this script to build and customize the image, then the aforementioned problems that cannot be repeated, the problem of image construction transparency, and the size of the Problems will be solved [7]. This script is the Dockerfile. Dockerfile is a text file, which contains a series of instructions (Instruction), each instruction builds a layer, so the content of each instruction is to describe how the layer should be built.

There are many important instructions in Dockerfile, such as RUN, FROM, COPY, ENV, ARG, VOLUME, etc. Here we mainly introduce the most commonly used instructions: FROM and RUN.

FROM: All customized images are based on FROM, where nginx is the basic image required for customization. Subsequent operations are based on nginx[8].

RUN: Used to execute the following command line commands. There are two formats:

shell format and exec format:

```
RUN <command line command>

# <command line command> is equivalent to the shell command operated on the terminal.
```

```
RUN ["executable file", "parameter 1", "parameter 2"]

# For example:

# RUN ["/test.php", "dev", "offline"] is equivalent to RUN ./test.php dev offline
```

2.2.1 Build a Redis image using Dockerfile

```
FROM debian:jessie

RUN buildDeps='gcc libc6-dev make' \
    && apt-get update \
    && apt-get install -y $buildDeps \
    && wget -O redis.tar.gz "http://download.redis.io/releases/redis-3.2.5.tar.gz" \
    && mkdir -p /usr/src/redis \
    && tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1 \
    && make -C /usr/src/redis \
    && make -C /usr/src/redis install \
    && rm -rf /var/lib/apt/lists/* \
    && rm redis.tar.gz \
    && rm -r /usr/src/redis \
    && apt-get purge -y --auto-remove $buildDeps
```

The main job of this dockerfile is to compile and install the redis executable. The author uses a RUN instruction and uses && to concatenate the required commands instead of using different RUN instructions for each command, which can avoid building a very bloated image. Each Dockerfile instruction builds a layer, so redundant instructions increase build and deployment times, and pack unnecessary things like build environments and updated software into the image[9]. The build command is: docker build: and its format is:

```
docker build [options] <context path/URL/->
```

The construction process is shown in Figure 2-1

```
root@hecs-140405:/home/CodeTest# docker build -t redis:v1 .
[+] Building 0.3s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 667B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [1/2] FROM docker.io/library/nginx@sha256:0d17b565c37bcd895e9d92315a05c1c3c9a29f762b011a10c54a66cd53c9b31
=> CACHED [2/2] RUN echo '<h1> hello, docker! </h1>' > /usr/share/nginx/html/index.html
=> exporting to image
=> => exporting layers
=> => writing image sha256:0ea9b58c4ee79f88742497c097e13d17ac92e5e13634b73bcff1142d4bfc44ac
=> => naming to docker.io/library/redis:v1
```

Figure 2-1 The construction process of redis image

Note that you will see a '.' at the end of the dockerbuild command. '.' indicates the current directory, but it is not the current path of the Dockerfile, but the specified context. Because Docker is a typical C/S architecture application, it is divided into Docker client (that is, the docker command that is usually typed) and Docker server (dockerd daemon process). The Docker build context is the decompressed content of the tar file uploaded by the Docker client to the server, that is, the file under the specified path after the dockerbuild command line.

Then use the docker images list command to view existing images. From Figure 2-2, you can find that the redis image with tag v1 already exists.

```

root@hecs-140405:/home# docker images
REPOSITORY      TAG         IMAGE ID      CREATED        SIZE
nginx            v1          0ea9b58c4ee7  24 hours ago  141MB
redis            v1          0ea9b58c4ee7  24 hours ago  141MB

```

Figure 2-2 Redis image build results

2.3 Docker Repository and image push

The warehouse is a centralized location for storing images, and the registration server is a specific server that manages the warehouse. A registration server can have multiple warehouses, and each warehouse can contain multiple images. In this sense, a repository can be viewed as a concrete project or directory. For example, for the warehouse address `dl.dockerpool.com/ubuntu`, `dl.dockerpool.com` is the registered server address, and `ubuntu` is the warehouse name. Usually, these two concepts do not need to be strictly distinguished.

Docker officially maintains a public repository called Docker Hub, which already contains more than 15,000 images. For most requirements, we can directly download the required images in Docker Hub to meet the requirements.

Before using Docker Hub, we need to log in through the login command `docker login`. The login process can refer to Figure 2-3.

```

root@hecs-140405:/home/CodeTest# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID
Username: gilesggg
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

```

Figure 2-3 docker login process

2.3.1 Push the local image to dockerhub

The basic process is as follows:

- ① Execute `docker login` to log in to your docker hub
- ② Tag the image you want to push (use the `tag` command), and add user-name to the tag
- ③ Use `docker push user-name/image-name:version` to push. Note that the user-name here must be the user name of your own docker hub, otherwise the push may fail.

2.4 data management

This chapter introduces how to manage data within Docker and between containers. There are two main ways to manage data in containers: Volumes and Bindmounts. The working modes of the two are shown in Figure 2-4.

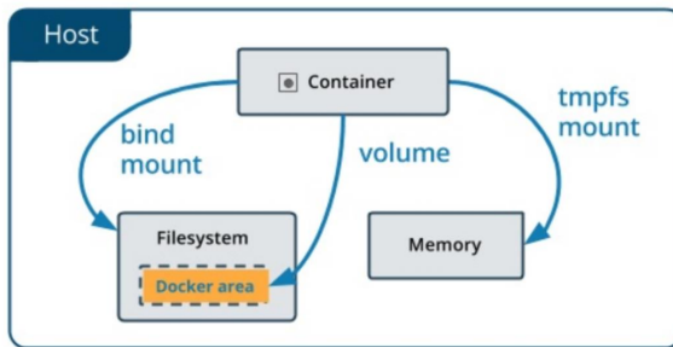


Figure 2-4 docker data management mode

2.4.1 Volume

The file environment inside the container is a temporary layer provided by the union file system. While programs are free to manipulate these files, all reading and writing takes place in a sandboxed environment. This layer, which temporarily records file modifications, is discarded when the container is stopped and deleted. Even if we can save the modification of the file in the container by submitting the image, using this method to achieve file persistence is not only cumbersome but also inconvenient for automatic management, so it cannot achieve the desired effect. In addition, this method only solves the problem of file output, but not the problem of file input.

In order to achieve the purpose of obtaining files from the outside and persistently storing files, Docker proposes the concept of data volumes. A data volume is a file or directory mounted on the filesystem inside a container. In the container, the data volume looks exactly the same as other files or directories, but because the data volume is mounted into the container from the outside, it can exist independently outside the life cycle of the container. The life cycle of the data volume is different from that of the container, so when the container exits or is deleted, the data volume still exists in Docker and will not be affected. Therefore, if a data volume is used to hold files, these files will not disappear when the container ends [10].

As a special file or directory in the container, the data volume is very different from other files or directories in the container and has unique characteristics. Figure 2-5 shows the relationship between data volumes and containers.

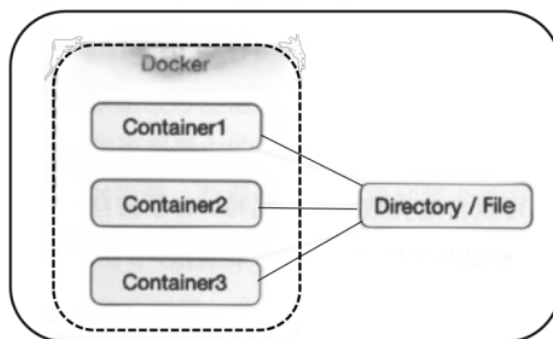


Figure 2-5 Relationship between data volume and container

To sum up: a data volume is a special directory that can be used by one or more containers, and can provide many useful features: ① data volumes can be shared and reused between containers ② changes to data volumes will take effect immediately ③ data volumes The update will not affect the image ④The data volume will always exist by default, even if the container is deleted [11].

2.4.2 Create data volume

We can use the `$ docker volume create my-vol` name to create a data volume, and we can use the `$ docker volume ls` command to view all data volumes.

At the same time, use the following command on the host to view the information of the specified data volume. The command and result are shown in Figure 2-6 below:

```
root@hecs-140405:/home/CodeTest# docker volume inspect my-volume
[
  {
    "CreatedAt": "2023-04-27T15:44:00+08:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/my-volume/_data",
    "Name": "my-volume",
    "Options": null,
    "Scope": "local"
  }
]
```

Figure 2-6 docker volume information

2.4.3 mount data volume

The data volume created by creating a container will be automatically mounted into the container and become part of the directory tree in the container. For the data volumes specified by the `-v` parameter and already existing in the host machine, Docker will also find them and mount them.

For those data volumes created through the `dockervolume create` command, we can use the form of `-v<name>:<dest>` to mount the data volume when using the `docker create` or `docker run` command.

```
$ sudo docker create --name web -v myvalume:/myvalume nginx
```

In the parameter form `-v<name>:<dest>`, `name` represents the name of the data volume. Because we use the defined data volume name, it is relatively convenient to configure the data volume, otherwise you need to enter a lengthy random name for the data volume. And `dest` indicates the path where the data volume is mounted to the container. If the corresponding directory in the container already exists, the mount operation will temporarily hide the original directory, and the data volume will be replaced to that place. The `-v` here can also be replaced with `--volume` and `--mount`.

For example, we can start a web container hanging on the data volume with the following code:

```
$ docker run -d -P \
    --name web \
    # -v my-vol:/webapp \
    --mount source=my-vol,target=/webapp \
    training/webapp \
    python app.py
```

Using the method of mounting the host directory or file as a data volume, it is very convenient to share files between the inside and outside of the container. In particular, when the configuration and code files that may need to be temporarily modified are placed in the container by mounting the host directory, the modification results will be reflected in the container immediately, making it easier to modify the file.

Since Docker can mount special directories in these hosts to containers as data volumes, it can naturally also mount other existing directories in the host to containers as data volumes. When creating a container, you can use `-v<src>:<dest>`. This parameter form mounts the directory in the host to the container.

```
$ sudo docker create --name web /var/html: /html -v /var/log/nginx: /var/log/nginx nginx
```

If we need to protect and limit the data in the volume, we can set the data volume to be readable, only we can add the three characters `:"ro"` after the parameter, such as in a Web service container, we want the HTML code directory and its contents to be read-only, while the log directory should be read-write. We can have the following settings.

```
$ sudo docker create --name web /var/html: /html: ro -v /var/log/nginx: /var/log/nginx nginx
```

2.4.4 Delete data volume

Because data volumes exist out of the container, although we create data volumes through the command to create a container, these data volumes will still not be destroyed due to the stop and deletion of the container. Therefore, we can provide data volumes to different containers at the same time, and achieve the reuse of data volumes.

You can use the docker volume list command with the docker volume remove `<volume-name>` command to delete the data volume, but this method is dangerous because you are likely to delete the data volume being used, so use `docker rm -v <container-name>` to delete, because this command is to delete the data volume used by the container when deleting the container.

2.5 Use the network

A large number of network application services based on distributed cloud computing are inseparable from providing services to users externally through the network, and internally also need to carry out information exchange between different service components, or between microservices. Docker as a cloud computing field The leader in application deployment, it also provides strong support for the network used by containers[12].

As shown in Figure 2-7, when the network inside the container is connected to docker0 through VethPair, it belongs to the same subnet as other containers connected to docker0, and the host is also connected to docker0 through a virtual network card. In other words, the host and the container achieve mutual access through docker0.

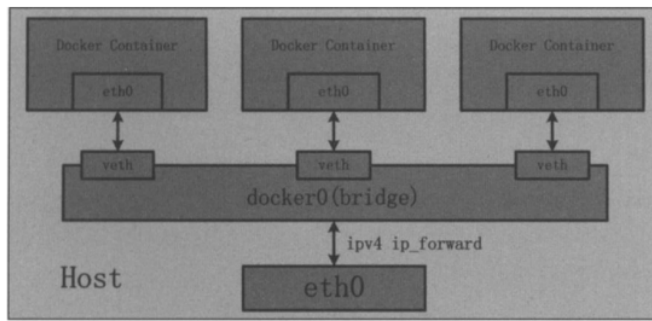


Figure 2-7 Network topology diagram of the container

We can use the `docker inspect <container-name>` command to obtain container-related information, and find the `NetworkSetting` field in the information result, which stores the network information of the container. In the network information, we can also see that the container is allocated by `NetworkNamespace` Hardware information, such as Mac address; you can also find the IP address, gateway address, bridge configuration, etc. allocated by the container in the `docker0` subnet.

2.5.1 external access container

Although containers can access external networks directly, external networks cannot access containers directly. By default, external networks must pass through the `docker0` gateway to connect to container networks, but because there may be multiple containers at the same time, this network relay cannot forward all containers at the same time. There are many solutions to this problem, such as port mapping in simple Docker network configuration commands and parameters. For containers that we want external network access to, we can provide Docker with access ports to implement port binding with the host.

So when some network applications are running in the container, if you want to make them accessible to the outside world, you can specify the port mapping through the `-P` or `-p` parameters.

2.5.2 connection between containers

Port mapping means that by default, the container breaks through the `docker0` subnet to transfer the access of the external network to a specific port to the port in the container, so that the application running in the container can process the program from the outside of the host, and Send processing requests over the network.

Sometimes, an application running in a container also needs to exchange data with an application running in another container over the network. At this point, it is necessary to complete the network connection between the container and the container through the container connection.

Generally speaking, there are two ways to connect containers. One is to use the `--link` parameter to interconnect containers, and the other is to add containers to a custom Docker network to connect multiple containers. Generally, you choose to use or .

Here we first introduce the first one - using the `--link` parameter to interconnect containers:

To set up inter-container communication, we can implement it by carrying the `-link` parameter when creating the container, and the usage form of the parameter is `-link<container>`. Containers created with this parameter will open network access to connected containers. In the new

container, we can connect and request the applications in the connected container through the port exposed by the connected container, and obtain the data provided by them.

Here we create a MySQL container and have a web service container connect to it.

```
$ sudo docker run -d --name mysql mysql
$ sudo docker run -d -p 80: 80 -p 443: 443 --name web --link mysql nginx
```

To set the communication parameters between containers, you only need to specify the connected container, not the port of the connected container. When creating and starting the connected container, you don't need to use the -P or -p parameter to map the port. This ensures that the port of the connected container is only used in the communication between the containers, and will not be exposed to the external network or accessed by other containers.

2.5.3 Create a Docker network to complete container interconnection

First create a new Docker network using the command below.

```
$ docker network create -d bridge my-net
```

The -d parameter specifies the Docker network type.

Then you can connect the container to the network. Run a container with the following command and connect to the newly created my-net network

```
$ docker run -it --rm --name busybox1 --network my-net busybox sh
```

Open a new terminal, run another container and join the my-net network

```
$ docker run -it --rm --name busybox2 --network my-net busybox sh
```

Then let the two busyboxes ping each other:

```
/ # ping busybox1
/ # ping busybox2
```

It can be seen from Figure 2-8 that the two can be pinged successfully, indicating that the network is established successfully, and the two containers can communicate with each other, that is, the busybox1 container and the busybox2 container have established an interconnection relationship.

```

/ # ping busybox2
PING busybox2 (172.19.0.3): 56 data bytes
64 bytes from 172.19.0.3: seq=0 ttl=64 time=0.068 ms
64 bytes from 172.19.0.3: seq=1 ttl=64 time=0.066 ms
64 bytes from 172.19.0.3: seq=2 ttl=64 time=0.049 ms
64 bytes from 172.19.0.3: seq=3 ttl=64 time=0.063 ms
64 bytes from 172.19.0.3: seq=4 ttl=64 time=0.062 ms
64 bytes from 172.19.0.3: seq=5 ttl=64 time=0.056 ms
64 bytes from 172.19.0.3: seq=6 ttl=64 time=0.053 ms
64 bytes from 172.19.0.3: seq=7 ttl=64 time=0.063 ms
64 bytes from 172.19.0.3: seq=8 ttl=64 time=0.081 ms
64 bytes from 172.19.0.3: seq=9 ttl=64 time=0.075 ms
64 bytes from 172.19.0.3: seq=10 ttl=64 time=0.074 ms
64 bytes from 172.19.0.3: seq=11 ttl=64 time=0.068 ms
64 bytes from 172.19.0.3: seq=12 ttl=64 time=0.074 ms
64 bytes from 172.19.0.3: seq=13 ttl=64 time=0.073 ms
64 bytes from 172.19.0.3: seq=14 ttl=64 time=0.077 ms
64 bytes from 172.19.0.3: seq=15 ttl=64 time=0.072 ms
64 bytes from 172.19.0.3: seq=16 ttl=64 time=0.063 ms
64 bytes from 172.19.0.3: seq=17 ttl=64 time=0.077 ms
64 bytes from 172.19.0.3: seq=18 ttl=64 time=0.089 ms
64 bytes from 172.19.0.3: seq=19 ttl=64 time=0.076 ms
64 bytes from 172.19.0.3: seq=20 ttl=64 time=0.076 ms

PING busybox1 (172.19.0.2): 56 data bytes
64 bytes from 172.19.0.2: seq=0 ttl=64 time=0.051 ms
64 bytes from 172.19.0.2: seq=1 ttl=64 time=0.056 ms
64 bytes from 172.19.0.2: seq=2 ttl=64 time=0.060 ms
64 bytes from 172.19.0.2: seq=3 ttl=64 time=0.070 ms
64 bytes from 172.19.0.2: seq=4 ttl=64 time=0.077 ms
64 bytes from 172.19.0.2: seq=5 ttl=64 time=0.069 ms
64 bytes from 172.19.0.2: seq=6 ttl=64 time=0.073 ms
64 bytes from 172.19.0.2: seq=7 ttl=64 time=0.059 ms
64 bytes from 172.19.0.2: seq=8 ttl=64 time=0.073 ms
64 bytes from 172.19.0.2: seq=9 ttl=64 time=0.086 ms
64 bytes from 172.19.0.2: seq=10 ttl=64 time=0.074 ms
64 bytes from 172.19.0.2: seq=11 ttl=64 time=0.069 ms
64 bytes from 172.19.0.2: seq=12 ttl=64 time=0.060 ms
64 bytes from 172.19.0.2: seq=13 ttl=64 time=0.074 ms
64 bytes from 172.19.0.2: seq=14 ttl=64 time=0.079 ms
64 bytes from 172.19.0.2: seq=15 ttl=64 time=0.070 ms
64 bytes from 172.19.0.2: seq=16 ttl=64 time=0.066 ms
64 bytes from 172.19.0.2: seq=17 ttl=64 time=0.070 ms
64 bytes from 172.19.0.2: seq=18 ttl=64 time=0.072 ms
64 bytes from 172.19.0.2: seq=19 ttl=64 time=0.073 ms
64 bytes from 172.19.0.2: seq=20 ttl=64 time=0.069 ms
64 bytes from 172.19.0.2: seq=21 ttl=64 time=0.073 ms

```

Figure 2-8 Two containers in the network perform ping operations on each other

2.5.4 Docker Compose

If you have multiple containers that need to be connected to each other, a good way is to use Docker Compose.

Docker Compose is one of Docker's official orchestration (Orchestration) projects, responsible for quickly deploying distributed applications in clusters. Compose is positioned as "Defining and running multicontainer Docker applications", and its predecessor is the open source project Fig. Using a Dockerfile template file allows users to easily define a separate application container. However, in daily work, it is often encountered that multiple containers need to cooperate with each other to complete a certain task. For example, to implement a web project, in addition to the web service container itself, it is often necessary to add a back-end database service container, and even a load balancing container. Compose just meets such needs. It allows users to define a set of associated application containers as a project through a single docker-compose.yml template file (YAML format)[13].

The code in Figure 2-9 below shows how to use Compose to easily make Wordpress run in an independent environment.

```
1 version: '3.1'
2
3 services:
4
5   wordpress:
6     image: wordpress
7     # restart: always
8     ports:
9       - 8080:80
10    environment:
11      WORDPRESS_DB_HOST: db
12      WORDPRESS_DB_USER: exampleuser
13      WORDPRESS_DB_PASSWORD: examplepass
14      WORDPRESS_DB_NAME: exampledb
15    volumes:
16      - wordpress:/var/www/html
17
18   db:
19     image: mysql:5.7
20     # restart: always
21     environment:
22       MYSQL_DATABASE: exampledb
23       MYSQL_USER: exampleuser
24       MYSQL_PASSWORD: examplepass
25       MYSQL_RANDOM_ROOT_PASSWORD: '1'
26     volumes:
27       - db:/var/lib/mysql
28
29 volumes:
30   wordpress:
31   db:
```

Figure 2-9 code screenshot

Then we can access wordpress in the form of ip:port, where the port mapping is 8080:80, that is, port 8080 of the host is mapped to port 80 of the container, and the ip address is the address of the server I use. Then you can enter the WordPress installation page, as shown in Figure 2-10 below.

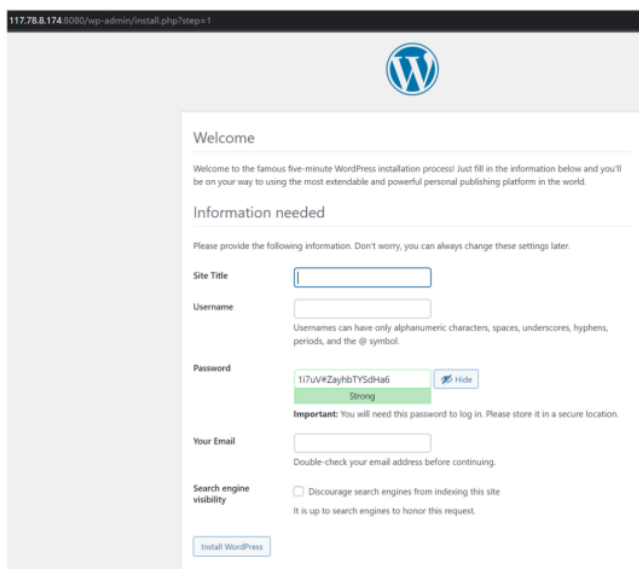


Figure 2-10 screenshot of wordpress interface

3 cloud computing

Cloud computing is a kind of distributed computing, which decomposes large-scale data calculation processing programs into many small programs, and processes and analyzes these small programs through multiple server systems to obtain results and return them to users [14]. The current cloud service is no longer limited to distributed computing, but combines computer technologies such as utility computing, load balancing, parallel computing, network storage, hot backup redundancy, and virtualization to achieve a comprehensive hybrid evolution and leap.

3.1 load balancing technology

The realization of load balancing technology is mainly divided into the following types: ① HTTP redirection load ② DNS domain name resolution load ③ reverse proxy load ④ IP load (NAT load and IP tunnel load) ⑤ direct routing (LVS-DR) ⑥ IP tunnel (LVS-TUN)

3.1.1http redirection load balancing

Redirection is ubiquitous in modern networks because HTTP applications always do the following three things: ① Reliable execution of HTTP transactions ② Minimize latency ③ Save network bandwidth.

3.1.2IP Load Balancing

DNS load balancing is a distributed computing technology that distributes client requests to multiple servers through domain name resolution services, thereby achieving balanced distribution of network traffic. Specifically, DNS load balancing generally adopts the method of mapping multiple IP addresses to the same domain name. Each IP address corresponds to a different server. When a client sends a request to the domain name, the DNS server will return one of the IP addresses. The client sends a request to the corresponding server based on the IP address [15].

3.1.3 Reverse proxy load balancing

Using a proxy server can forward the request to the internal web server. Using this acceleration mode can obviously improve the access speed of static web pages. Therefore, you can also consider using this technology to allow the proxy server to evenly forward the request to one of the multiple internal Web servers, so as to achieve the purpose of load balancing.

The remaining three types of load balancing will not be described in detail.

3.1.4 Kubernetes load balancing technology

Kubernetes is a system for running and coordinating containerized applications on a group of hosts, designed to provide predictability, scalability, and high availability to fully manage the life cycle of containerized applications and services . Users can define how applications run and how they interact with other applications or the outside world. Kubernetes is highly flexible and definable. In a brief summary, it has the following important features: (1) Automatic boxing (2) Self-healing (self-healing) (3) Horizontal expansion (4) Service discovery and load balancing (5) Automatic release and rollback (6)) key and configuration management (7) storage orchestration (8) batch processing execution.

We mainly look at the fourth "service discovery and load balancing": Kubernetes has a built-in service discovery function for the system through one of its add-ons, KubeDNS (or CoreDNS), which configures DNS names for each Service and allows clusters [16]. The client directly uses this name to send an access request, and the Service has a built-in load balancing mechanism through iptables or ipvs.

Cluster mode of Kubernetes: Kubernetes uses a shared network to process multiple physical or virtual machine clusters. One server in the cluster (or a group of servers in a high-availability deployment) is used as the Master, which is responsible for managing the entire cluster, and the remaining machines are used as Worker Nodes. As shown in Figure 3-1. These hosts in a cluster can be physical servers or virtual machines.

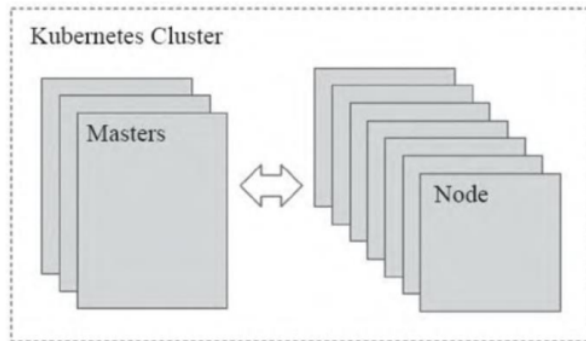


Figure 3-1 Kubernetes cluster host

The work of load balancing is mainly done by the Master, because the Master is the gateway and hub of the cluster, responsible for things such as exposing APIs for users and clients, tracking the health status of other servers, scheduling workloads in an optimal way, and orchestrating other components. It is the core contact point between users or clients and the cluster, and is responsible for most of the centralized management and control logic of the Kubernetes system. The Master node is similar to the queen bee in the bee colony.

The most commonly used core objects of Kubernetes: Pod, Deployment, Service, etc. Among them, Pod is the smallest/simplest basic unit created or deployed by Kubernetes. The controller itself is also a resource type, and it has multiple implementations, among which the implementation related to the workload includes Replication Controller and so on. Service is an implementation of "microservice". In fact, it is an abstraction: a logical collection composed of multiple Pod objects is defined through rules, and a policy for accessing this group of Pod objects is attached.

If the entire cluster can be compared to a traditional server, then Kubernetes (Master) is like the operating system kernel, whose main responsibility is to abstract resources and schedule tasks, and Pod resource objects are those processes running in user space. To put it simply, the LoadBalancer type of Service distributes the load of external requests to the NodePorts of the host IPs of multiple Nodes, preventing a single Node from bearing too much load, thereby completing the load balancing work.

3.2 scalability

In cloud computing, scalability refers to the ability of a system or application to quickly adjust the size of its resources to meet demand when it encounters changing load or user demand, without having to redesign or refactor the entire system. This scalability is an important feature of cloud computing, which can automatically increase or decrease computing, storage, network and other resources to adapt to load changes [17]. Scalability can help enterprises quickly expand or reduce resources according to demand, thereby saving costs and avoiding waste. At the same time, this ability can also ensure that applications can maintain stable and efficient operation during peak

periods. Scalability is a key advantage of cloud computing and one of the important reasons why enterprises choose cloud computing.

3.2.1 How Kubernetes achieves scalability

In Kubernetes' ecosystem, different components are provided in multiple dimensions and levels to satisfy different scaling scenarios.

- CA (Cluster Autoscaler): Node-level cluster-autoscaler components
- HPA (Horizontal Pod Autoscaler): Pod number auto expansion/shrinking
- VPA (Vertical Pod Autoscaler): Pod configuration auto-expands/shrinks, mainly CPU, memory, add-on-resizer components

Horizontal Pod Autoscaler automatically expands the number of Pods in ReplicationController, Deployment, ReplicaSet, and StatefulSet based on CPU utilization. In addition to CPU utilization, automatic scaling can also be performed based on custom metrics provided by other applications. Pod auto-scaling does not apply to objects that cannot be scaled, such as DaemonSet.

Vertical Pod Auto Scaling VPA automatically calculates the flow of containers in these Pods and their current CPU and memory usage, and uses this data to determine optimized resource constraints and requests to ensure that these Pods are always in a high resource utilization state. For example, you have a Pod that uses 50% of the CPU but only requests 10%. VPA will determine that the Pod consumes more CPUs than the requested one, and it will enable workloads such as Deployment to restart the Pod to take effect the new recommended values, thereby achieving vertical resource scaling.

3.3 Application Gateway

There are some of my own understandings here, so there is vernacular rhetoric. A gateway is the connection point between one area of the network and another area. The reason why so many areas are divided is mainly because of easy management. Changes in one area can affect the other as little as possible. An area; for the network, the area is nested according to the protocol stack, and the scope of the top domain of the stack is usually larger. For example, an IP domain can nest multiple link layer domains, and a link layer domain can nest multiple Physical layer domain, but if the same layer nests or the lower layer nests the upper layer area, it is generally called overlay technology (the upper layer nested lower layer is too common, so it is not specifically called overlay), which is also a commonly used technology in VPN; the port/interface of the gateway It can terminate one area, and then convert the message to the format of another area. For example, the vlan sub-interface of a router can first terminate the physical layer, then terminate the link layer including vlan, and then terminate the three-layer subnet, and then transfer to another A subnet, or tunnel, or non-IP network, etc.;

Overlay technologies commonly used in cloud data centers and corresponding gateways:

① SAN network used for centralized storage; ② VxLAN; ③ VPN gateway: IPsec technology is commonly used in cloud data centers. One is to securely connect clients to the private network of the data center, and the other is to connect the private ④ Application gateway/API gateway: This gateway will terminate the 4-layer connection outside the data center, and then do application layer security, interface adaptation, metering and billing, etc. Encapsulation and forwarding in the form of the internal network of the center ⑤ Bastion machine: This is also an application layer gateway. Users first log in to the bastion machine, and then use it as a springboard to log in to the private

network inside the data center. The bastion machine records login logs, desktop video and other information security content; ⑥router, NAT, ELB: Cloud data centers have physical and NFV-based gateways, all of which are three-tier gateways [18].

3.3.1 Azure Application Gateway

Azure Application Gateway is a dedicated virtual appliance that provides Layer 7 load balancing and other Layer 7 routing capabilities, such as a round-robin mechanism for incoming traffic, cookie-based session affinity, URL path-based routing, and managed Multiple site capabilities. As part of an application gateway, a web application firewall (WAF) can help protect web applications from common web exploits. Application gateways can be configured as Internet-facing only gateways, internal-only gateways, or a combination of both. This dedicated virtual appliance provides a flexible way to control application delivery and quickly scale resources up or down as needed [20].

3.4 Why load balancing and scaling are complimentary concepts

Combining previous load balancing and scalability related content, we can draw the following conclusions:

Load balancing is a technique for distributing workload so that requests are distributed evenly among multiple servers or processing units to ensure that each server or processing unit can handle their share. This helps reduce the burden on individual servers or processing units, thereby increasing system availability and performance.

However, if the system is faced with higher load demands, even with the use of load balancing techniques, it may cause a single server or processing unit to reach the limit of its processing capacity. To solve this problem, the system needs to increase the number of processing units to meet higher load requirements, which is called scaling.

Scaling is the process of increasing or decreasing the number of processing units to accommodate load requirements. Scaling can be horizontal, increasing or decreasing the number of processing units, or vertical, increasing or decreasing the processing power of a single processing unit. By scaling, the system can adapt to different load requirements and increase its availability and performance.

Thus, load balancing and scaling are complementary concepts that together provide a way to address system scalability. Load balancing techniques can ensure that requests are distributed across multiple processing units, while scaling can increase the number of processing units to meet higher load demands. In practical applications, it is usually necessary to use load balancing and scaling technologies at the same time to ensure high availability and performance of the system.

3.5 The benefits of combining the three technologies

Containerization, scalability, and application gateways are three distinct but complementary technologies that, combined, can provide customers with many benefits [19].

First, containerization technologies can improve application portability and reliability because they package applications into lightweight, self-contained containers that can be easily deployed and run in different environments. This means that applications can be moved between different cloud service providers, which increases application portability.

Second, scalability techniques can enable applications to automatically adapt to varying load requirements. By using the automatic scaling function, the system can automatically adjust the

number of resources according to the current load situation, thereby improving the availability and performance of the system.

Finally, an application gateway is a centralized point of entry that helps manage application traffic and security. It centralizes multiple applications at a single point of entry and provides granular control over traffic management and security. The application gateway can perform authentication, authorization, and security audit on incoming requests, thereby improving the security of the system.

When these technologies are combined, customers can achieve better application portability, high availability, scalability and security. For example, containerization technology can make applications easy to deploy and manage, autoscaling technology can automatically adjust resources according to load, and application gateway can ensure application security and performance. The combination of these technologies can improve application reliability and performance, and help customers better manage their applications.

Case ---- e-commerce website

Suppose you have an e-commerce website that needs to handle many concurrent requests from thousands of users. To ensure that the website can run stably, technologies such as containerization, scalability, and application gateways need to be employed.

First, the website's application is packaged into a container so that it can be deployed to different cloud service providers. For example, Docker can be used to package an application into a container image and upload it to a cloud service provider's container registry.

Second, use the auto-scaling feature to ensure that the website can automatically adjust resources under different load demands. When the load is low, the number of resources can be reduced, and when the load is high, the number of resources can be increased. For example, you can use an Amazon EC2 Auto Scaling group to automatically add or remove EC2 instances based on load.

Finally, use an application gateway to ensure website security and performance. For example, an AWS Application Load Balancer (ALB) can be used as an application gateway to funnel all traffic to a single entry point and authenticate and authorize incoming requests. In addition, you can use the advanced routing function of ALB to route requests to different backend services to better handle complex requests.

In this example, containerization technology makes applications easy to deploy and manage, autoscaling technology can automatically adjust resources according to load, and application gateway can ensure application security and performance. Through the combination of these technologies, e-commerce websites can achieve better reliability and performance, and be able to better manage their applications.

3.6 serverless functions and a containerized approach

Serverless functions, also known as Function-as-a-Service (FaaS), are a new approach to cloud-based web services. In traditional web services, you usually need to maintain the server and infrastructure yourself. However, in the case of serverless functions, instead of maintaining the servers yourself, you can upload your code to the cloud and use the provider's serverless platform to manage them. When a request needs to be processed, the platform will automatically start the code and run it on the virtual machine. Serverless functions not only simplify the development process, but also reduce costs and increase scalability.

Serverless functions have the following advantages and disadvantages compared to containerization:

advantage:

- ① **Faster development and deployment:** Serverless functions only need to write and upload code, without considering the configuration and maintenance of infrastructure and servers. This makes development and deployment much faster and easier.
- ② **Cost saving:** Serverless functions are billed on demand, and you will only pay when you need it. This means you can adjust resources as needed and only pay for what you actually use. In contrast, containerization requires maintaining servers and infrastructure, so the cost is usually higher.
- ③ **High scalability:** Serverless functions can be automatically scaled as needed, so they can easily handle changing workloads. In contrast, containerization needs to manage containers and servers by itself, so it requires more work to ensure scalability.

insufficient:

- ① **Running time limit:** Serverless functions are usually limited by running time, so they may not be suitable for long-running tasks. In contrast, containerization has no such limitations and is thus better suited for long-running tasks.
- ② **Portability:** Serverless functions are usually vendor-specific and thus may not be easily portable to other vendor's platforms. Containerization, by contrast, is portable because they run inside containers.

To sum up, both serverless functions and containerization have their own advantages and disadvantages. Serverless functions are a good choice if you need to develop and deploy web services quickly and handle changing workloads. If you need long-running tasks or portability, containerization may be a better fit. Whichever method you choose, you need to consider your needs and resource budget, as well as your technical capabilities and experience.

3.6.1 case comparison

When considering serverless functions or containerization, knowing the real-world comparisons can help you better understand the differences and pros and cons of the two approaches. The following is a comparison of two actual cases:

Case 1: Serverless Functions vs. Containerization - Web Applications

Suppose you have a cloud-based web application that needs to handle different numbers of requests at different times. Here's a comparison of implementing this application using serverless functions and containerization:

Table 3-1 Comparison of serverless functions and containerization

Field	Serverless Function	Containerization
Platform	Azure Functions, AWS Lambda, Google Cloud Functions, etc.	Any platform that supports containerization technology
Deployment	Just upload code, don't need to care about server management and deployment	Can run anywhere, but requires personal management and deployment
Cost	Only pay for resources actually used	Requires more administration and configuration, but provides greater flexibility and control
Scalability	Can be scaled automatically so it can handle a large number of concurrent requests	Can scale horizontally to handle higher loads
Run time	limit Usually runs in minutes, not suitable for long-running tasks	Can handle long-running tasks
Portability	General, often vendor specific	Very good

In this case, using serverless functions is a better fit than containerization because the application needs to handle varying numbers of requests and needs to scale automatically. Serverless functions are better suited for this scenario because they scale automatically and are billed on demand.

Case 2: Serverless Functions vs. Containerization - Order Processing for an Online Retail Store

An online retailer handles a high volume of online orders during the holiday season and needs to ensure that the system is highly available and scalable to meet consumer demand.

Serverless functions solution: Online retailers can use serverless functions to process these orders. They can use serverless computing platforms like AWS Lambda or Azure Functions. Serverless functions allow online retailers to simply upload their code without having to worry about server management and availability issues. Also, thanks to autoscaling, the serverless function can handle a large number of concurrent requests, thus meeting the peak traffic demands of the online retailer during the holiday season.

Containerized solutions: Online retailers can use containerization technologies such as Docker to process these orders. They can package the order processing application and its dependencies into Docker containers and run those containers on a container orchestration platform such as Kubernetes. Containerized solutions require online retailers to manage and deploy containers themselves. Containerization can manage and adjust resources very flexibly, so the online retailer can scale horizontally to handle higher loads as needed.

According to this case, both serverless functions and containerization technologies can provide a solution to meet the needs of online retailers. Serverless functions can offer higher scalability, lower cost, and simpler deployment. Containerization can provide greater flexibility and control, as

well as the ability to handle longer-running tasks. Online retailers can choose the solution that best suits their business based on their specific needs.

3.6.2 summary

Both serverless functions and containerization are modern technology solutions when it comes to implementing cloud-based web services, but there are some differences between them. Serverless functions are an event-driven computing model that only runs code when a request is processed and can scale automatically. In contrast, containerization is a lightweight virtualization technology that packages together an application and its dependencies to run anywhere. The following table 3-2 is a comparison table of their advantages and disadvantages.

Table 3-2 Comparison of the advantages and disadvantages of the two

Solution	Advantages	Disadvantages	Applicable Scenarios
Serverless functions	Fast deployment, auto-scaling, lower cost	Certain function limitations, possibly higher need for persistent storage	Handles short-lived tasks, event-driven scenarios
Containerization	Greater portability, flexibility and control, broader functionality	Greater complexity, higher cost, more management and configuration required	Long-running tasks, greater flexibility and Portability Scenarios

In real-world cases, no server functions are more appropriate for short-term task or event-driven scenarios, while containerization is more appropriate for long-running tasks or when greater flexibility and portability are required. The choice of using serverless functions and containerization depends on actual requirements and resource budgets.

Overall, Server-Free functions and containerization are modern technical solutions that improve the performance and scalability of Web services. Serverless functions are a faster, more automated solution for working with short-term tasks or event-driven scenarios. Containerization, on the other hand, provides greater flexibility and portability and is suitable for scenarios where long-running tasks or scenarios require greater control and customization.

4.0 Conclusion

What I've covered in this article covers virtualization, containerization, and cloud computing. It introduces different virtualization technologies and explains the advantages of containerization technologies including Docker. This document provides general instructions for using images and explains how to customize images using Dockerfiles. It also discusses data management and networking in containers. In the cloud computing section, different load balancing techniques are covered, including how Kubernetes achieves scalability. The document concludes with a case study discussion of the benefits of combining the three technologies, serverless functions and containerization approaches.

<Youtube Link: <https://youtu.be/DfZ3UqyRV0Y>>

Bibliography

- [1] Yoshimura T, Nakazawa R, Chiba T. ImageJockey: A framework for container performance engineering. In: Proc. of the 13th IEEE Int'l Conf. on Cloud Computing (CLOUD). Beijing: IEEE, 2020. 238 – 247. [doi: 10.1109/CLOUD49709.2020.00043]
- [2] Ruan BW, Huang H, Wu S, Jin H. A performance study of containers in cloud environment. In: Proc. of the 10th Asia-Pacific Services Computing Conf. on Advances in Services Computing. Zhangjiajie: Springer, 2016. 343 – 356. [doi: 10.1007/978-3-319-49178-3_27]
- [3] Zerouali A, Mens T, Decan A, Gonzalez-Barahona J, Robles G. A multi-dimensional analysis of technical lag in debian-based Docker images. *Empirical Software Engineering*, 2021, 26(2): 19. [doi: 10.1007/s10664-020-09908-6]
- [4] Gong Zheng. The Definitive Guide to Kubernetes: Full Contact from Docker to Kubernetes Practice (Commemorative Edition). BEIJING BOOK CO. INC., 2017.
- [5] Wang Qiang, Wang Ruigang, and Zhou Deyong. "Docker-Based Dynamic Load Balancing Elastic Scaling System." *Computer and Digital Engineering* 46.6 (2018): 1140-1144.
- [6] Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. "An introduction to docker and analysis of its performance." *International Journal of Computer Science and Network Security (IJCSNS)* 17.3 (2017): 228.
- [7] Boettiger, Carl. "An introduction to Docker for reproducible research." *ACM SIGOPS Operating Systems Review* 49.1 (2015): 71-79.
- [8] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." *Linux j* 239.2 (2014): 2.
- [9] Mouat, Adrian. *Using docker: developing and deploying software with containers.* "O'Reilly Media, Inc.", 2015.
- [10] Kane, Sean P., and Karl Matthias. *Docker: Up & Running.* "O'Reilly Media, Inc.", 2023.
- [11] Bernstein D. Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 2014, 1(3): 81 – 84. [doi: 10.1109/MCC.2014.51]
- [12] Zhang Y, Yin G, Wang T, Yu Y, Wang HM. An insight into the impact of Dockerfile evolutionary trajectories on quality and latency. In: Proc. of the 42nd Annual Computer Software and Applications Conf. (COMPSAC). Tokyo: IEEE, 2018. 138 – 143. [doi: 10.1109/COMPSAC.2018.00026]
- [13] Fokaefs M, Barna C, Veleda R, Litoiu M, Wigglesworth J, Mateescu R. Enabling DevOps for containerized data-intensive applications: An exploratory study. In: Proc. of the 26th Annual Int'l Conf. on Computer Science and Software Engineering. Toronto: ACM, 2016. 138 – 148.

- [14] Singh C, Gaba NS, Kaur M, Kaur B. Comparison of different CI/CD tools integrated with cloud platform. In: Proc. of the 9th Int’ l Conf.on Cloud Computing, Data Science & Engineering. Noida: IEEE, 2019. 7 – 12. [doi: 10.1109/CONFLUENCE.2019.8776985]
- [15] Chen LP, Babar MA, Zhang H. Towards an evidence-based understanding of electronic data sources. In: Proc. of the 14th Int’ l Conf. on Evaluation and Assessment in Software Engineering. Keele: ACM, 2010. 135 – 138.
- [16] Al-Dhuraibi Y, Paraiso F, Djarallah N, Merle P. Autonomic vertical elasticity of Docker containers with ElasticDocker. In: Proc. of the 10th IEEE Int’ l Conf. on Cloud Computing (CLOUD). Honolulu: IEEE, 2017. 472 – 479. [doi: 10.1109/CLOUD.2017.67]
- [17] Liu Yue. "Overview of Cloud Computing and Application Research of Mobile Cloud Computing." Information and Communication Technology 4.2 (2010): 14-20.
- [18] Sun Xianghua. "Research Status and Development Trend of Cloud Computing." Computer Measurement and Control 19.5 (2011): 998-1001.
- [19] Furht, Borivoje, and Armando Escalante. Handbook of cloud computing. Vol. 3. New York: springer, 2010.
- [20] Wang, Lizhe, et al. "Cloud computing: a perspective study." New generation computing 28 (2010): 137-146.
- [21] Rittinghouse, John W., and James F. Ransome. Cloud computing: implementation, management, and security. CRC press, 2016.
- [22] Sosinsky, Barrie. Cloud computing bible. John Wiley & Sons, 2010.
- [23] Dikaiakos, Marios D., et al. "Cloud computing: Distributed internet computing for IT and scientific research." IEEE Internet computing 13.5 (2009): 10-13.

Zhiqiang Geng-2200520-Cloud Engineering Principles and Practice Assignment

GRADEMARK REPORT

FINAL GRADE

91 / 100

GENERAL COMMENTS

Instructor

Docker Overview + Benefits (20): 18
Docker Examples (30): 26

Video (5): 5

Cloud Comp tech (25): 24

Cloud Comp functions (20): 18
Feedback:
=====

Docker benefits were described, overview could have been a clearer. Very good examples.

Excellent discussion of cloud tech, liked the Kubernetes evaluation you included.

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

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29
