

CEXP

by Peng Xu

FILE	25655_PENG_XU_CEXP_343871_931657820.DOCX (3.93M)		
TIME SUBMITTED	09-MAY-2021 08:54AM (UTC+0100)	WORD COUNT	6873
SUBMISSION ID	152313509	CHARACTER COUNT	35799



Cloud Engineering Principles and Practice

MSc Computing Portfolio

Computer Networks and Cyber Security

ACCA7027

Dr Mike Dacey

Peng Xu

2006614

contents

1. docker.....	3
1.1 Benefits of containerization	4
1.1.1 The importance of containers	4
1.1.2 Benefits	5
1.2 Image.....	7
1.2.1 Build a image	9
1.3 Docker volume	12
1.3.1 Related concepts.....	12
1.3.2 Create Docker volume.....	15
1.4 Docker network.....	17
1.5 developersb Application.....	18
1.5.1 Deploy a static developersbsite	18
2. Kubernetes	21
2.1 Related concepts	21
2.1.1 Node0:.....	22
2.1.2 Pod:	23
2.1.3 Automatic service discovery:	24
2.1.4 Three network models of Kubernetes	26
2.1.5 Kubernetes network namespace	28
2.2 Application	29
Reference	33

1. docker

Docker is a container used to hold applications. Just like a cup can hold water, a pen holder can hold a pen, a school bag can hold a book, developer can put hello word in docker, developer can put a developersite in docker, and developer can put any program developer want. Put it in docker.

docker container

- Standardization

1) Mode of Transportation

There is a super terminal for docker transportation, and wherever goods are needed, the docker whale will first send the goods to the super terminal, and then the docker whale will send the goods to the destination from the super terminal. For the corresponding technology, for example, the developer needs to put the desktop The application of the computer is deployed to the laptop. The developer may choose to send it with QQ or copy it with a USB flash drive. Docker standardizes this process. Developers only need to execute a docker command on the desktop, send the whale over, and send the program. Go to the Super Dock, execute a docker command on the notebook, and then the whale will send the program from the Super Dock to the notebook.[1]

Transport standardization docker standardized transmission process Execute the docker command to complete the sending and transmission.

2)Storage method

When copying the program to the notebook, specify a directory, and the developer must remember this directory, because next time the developer may have to modify the changes and continue to pass it, after having docker, there is no need, and the developer does not need to remember the program. Where is it? The storage is standardized. Developers only need to execute one command when they want to use the runtime.[2]

Storage standardization, developer only need to execute one command when developer want to use the runtime

3)API interface

Docker provides a series of rest api interfaces, including a control to start, stop, view, delete, etc. for docker, which is the developer's application. For example, when the developer wants to start tomcat, the developer needs to execute the startup command. The shutdown command must be

executed when it is stopped. If it is not tomcat, the developer may need some other commands to control it. With interface standardization, all applications can be controlled by only executing the same command. With docker, the developer remembers docker The command can operate it.

Interface standardization. Execute the same command to control all applications. Just remember the docker command execution.[3]

· 4)Isolation

Developers have their own cpu, hard disk, and memory when using virtual machines, and they do not feel the presence of external hosts at all. Docker is similar, but it is lighter and can be created and destroyed quickly. It may take a few minutes to create a virtual machine. , It only takes one second to create docker. The bottom-level technology is a Linux kernel restriction mechanism called LXC. LXC is a light developersight container virtualization technology. Processes and resources are isolated with maximum efficiency. Through cgroup, namespace Other restrictions, the physical resources used by the isolation process group, such as CPU, MEMORY, etc., this mechanism has been added to the Linux kernel 7 or 8 years ago, and it did not become popular until the birth of docker in 2013. Docker success is like a hero Create the times, and the times create heroes.If there is no cloud computing, agile development, and high-frequency elastic scaling requirements, without the rapid development of the IT industry for so many years, there will be no docker.

1.1 Benefits of containerization

"Docker" literally means people who work on the wharf. This name should be the result of the official deliberation. It has a strong metaphorical nature: it borrows a system that has matured in the real world-the global logistics system. Mapping the role of docker in the software field.[4]

In the global logistics system, a very important invention is the container.

1.1.1 The importance of containers

In order to understand this matter, developers can first look at the logistics situation before the appearance of the container: the goods are packed in boxes after they are produced in the factory, and then boxes are moved to the truck, and then boxes are unloaded, and boxes are sent to the train. , Transported to the railway station near the dock, unloaded another box, loaded it on the truck, pulled it to the freighter, and then loaded it box by box.

It can be seen that in the entire process, a lot of time, manpower, and material resources are wasted on the loading and unloading in the middle. In the logistics system, due to the limitation of the distance and the speed of the means of transportation, the time for the goods to be on the road is a certain amount. It is difficult to shorten this time until the transportation technology is improved, so the "cargo loading and unloading time" becomes the logistics system.

The bottleneck in this bottleneck has been greatly improved after the emergence of containers. The important thing about the container is that it provides a universal standard specification for packaged goods (size and shape conform to uniform standards), which has some huge advantages: only need to be packaged once before transportation, the container can be put on trains, trucks, Pull it to the dock and put it directly on the cargo ship; after unloading the ship, put it on the train or truck and transport it to the destination. And because the container conforms to the unified standard, the entire process is very easy to mechanize, which triggers the standardization process of the entire global logistics centered on the container, which saves a lot of resources, the logistics cost drops rapidly, and promotes the flow and reconfiguration of global resources.[5]

Docker borrows the metaphor of the container

Docker is like a docker who loads cargo in a container. It packages applications into a container with a certain standard specification. In the language of the computer field, this container packaged according to a certain specification is called "image". In fact, it is to add some extra content, format and the like to the original code to produce something that meets a certain standard.

1.1.2 Benefits

1) Easy to deploy

For people who are new to programming languages, setting up the environment often takes several hours, and one of the small problems may take a long time to solve. In many cases, it is necessary to get help from other members of the team regarding environment construction. With the container, this becomes very easy. [6]The developer's development environment is just the address of one or several container images, and at most an execution script that controls the deployment process is needed. Or further put the environment imaging and imaging script into a git project, publish it to the cloud, and pull it locally when needed.

2) Deployment Security

When a developer receives a bug feedback, many times there is no problem with the local resources. This happens because of the inconsistency of the environment. The debugging during the development process often cannot guarantee the problems of other environments, but the developer has to pay for this is really a distressing thing. With containers, this will rarely happen. Developers can use container technology to keep the development environment, test environment, and production environment unified in version and dependencies, ensuring that the code is executed in a highly unified environment. The unification of the test environment can also solve the environmental requirements of the CI process.

Today with increasing demands for distributed technology and capacity expansion, if operation and maintenance can use container technology to deploy the environment, it will not only save a lot of deployment time, but also minimize many errors caused by manual configuration of the environment.

3) Good isolation

Whether it is development or production, often multiple services may need to be run on one machine, and the dependency configurations required by each service are not the same. If two applications need to use the same dependency, or the dependencies between two applications need to be different. There are some conflicts, and problems are prone to occur at this time. Therefore, it is better to isolate different services provided by different applications on the same server. Containers have a natural advantage in this regard. Each container is an isolated environment. Developers require all services provided inside the container, and the container can provide all of it on its own. This kind of high cohesion can realize fast separation of problematic services, and can realize fast troubleshooting and timely processing in some complex systems. (Of course, it should be noted that this isolation is only compared with the server, and the virtual machine technology must have better isolation)

4) Fast rollback

The previous rollback mechanism of the container generally needs to be redeployed based on the previous version of the application and replace the current problem version. In the initial era, it may be a complete development to deployment process, and it often takes a long time to execute this process. In a git-based environment, it may be to roll back a certain historical commit and then redeploy. These are not fast enough compared with the container technology, and may cause new problems (because it is based on the modification of the new version). The container technology is born with a rollback attribute, because each historical container or image will be saved, and it is very fast and simple to replace a container or a certain historical image.

5) Low cost

This may be one of the most obvious and useful advantages. Before the emergence of containers, developers often needed a new server or a virtual machine to build an application. The purchase cost and operation and maintenance cost of the server are very high, and the virtual machine needs

to occupy a lot of unnecessary resources. In contrast, container technology is much smaller and lighter, and only needs to build the dependencies required by the application inside a container. This is also the main reason for the rapid development of container technology.

6) Low developers' management costs

With the continuous popularization and development of container technology, the subsequent container management and orchestration technology has also been developed. Orchestration tools such as Docker Swarm, Kubernetes, Mesos, etc. are constantly iteratively updated, which allows container technology to have more possibilities and more room for development in the production environment. With the development of the big environment, the cost of using and learning docker and other containers is also decreasing, becoming the choice of more developers and enterprises.

1.2 Image

If a developer has ever been a VM administrator, he can understand the Docker image as a VM template. The VM template is like a stopped VM, and the Docker image is like a stopped container. As a developer, a developer can understand the image as Class.[7]

First, a developer needs to pull the image from the image warehouse service. A common image warehouse service is Docker Hub, but there are other image warehouse services.

The pull operation will download the image to the local Docker host, which can be used to start one or more containers.

The image is composed of multiple layers. After each layer is superimposed, it looks like an independent object from the outside. Inside the image is a streamlined operating system (OS), and it also contains files and dependencies necessary for the application to run.

Because the container is designed to be fast and small, the image is usually relatively small.

It has been mentioned many times before that imageing is like a stopped container (class). In fact, a developer can stop the operation of a container and create a new image from it.

Under this premise, the image can be understood as a build-time structure, and the container can be understood as a run-time structure.

(1) Images and containers

The above figure shows the relationship between the image and the container from the top-level design level. Usually use the docker container run and docker service create commands

to start one or more containers from a certain image.

Once the container is started from the image, the relationship between the two becomes mutually dependent, and the image cannot be deleted until all the containers started on the image are stopped. Attempting to delete an image without stopping or destroying the container that uses it will cause an error.[8]

(2) The image is usually small

The purpose of the container is to run an application or service, which means that the image of the container must contain the operating system and application files necessary for the application/service to run.

However, the container pursues speed and compactness, which means that when building an image, it is usually necessary to cut out unnecessary parts and keep the volume small.

For example, a Docker image usually does not contain 6 different shells for users to choose—usually there is only a streamlined shell in a Docker image, or even no shell.

The image does not contain the kernel yet—the containers are the kernel of the shared Docker host. So it is sometimes said that the container only contains the necessary operating system (usually only operating system files and file system objects).

The Hyper-V container runs on a dedicated lightweight VM and uses the operating system kernel inside the VM.

The official Docker image, Alpine Linux, is only about 4MB, which can be said to be a typical example of the small size of the Docker image.[9]

However, the more common state of the image is like the official Ubuntu Docker image, which is about 110MB. Most of the useless content has been cut off in these images.

The Windows image is larger than the Linux image, which is related to the working principle of the Windows OS. For example, the uncompressed latest Microsoft .NET image (microsoft/dotnet:latest) exceeds 1.7GB. After the Windows Server 2016 Nano Server image

(microsoft/nanoserver:latest) is pulled and decompressed, its volume is slightly larger than 1GB.

(3) Pull image

After the Docker host is installed, there is no image locally.

docker image pull is a command to download images. The image is downloaded from the warehouse of the remote image warehouse service.[10]

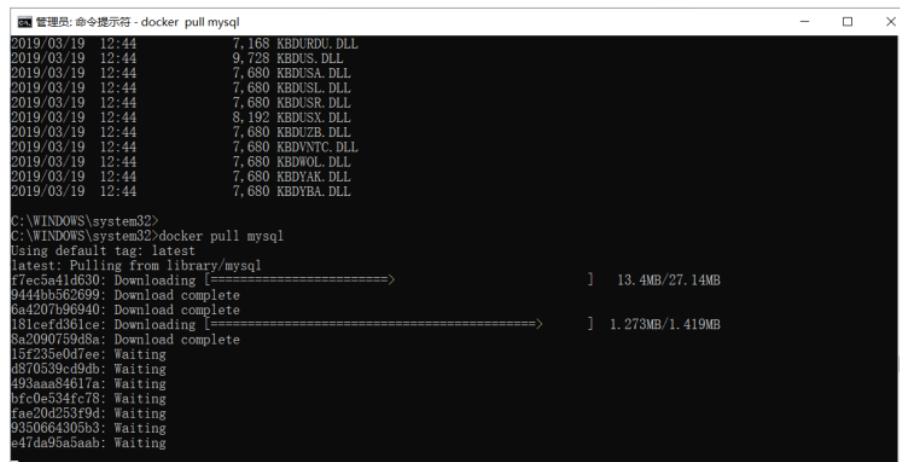
Image warehouse service

Docker images are stored in the Image Registry service.

The image warehouse service of the Docker client is configurable, and Docker Hub is used by default.

The image repository service includes multiple image repositories (Image Repository). Similarly, a image warehouse can contain multiple images.

1.2.1 Build a image



```
管理員: 命令提示符 - docker pull mysql
2019/03/19 12:44      7,168 KBDURDL.DLL
2019/03/19 12:44      9,728 KBDUSL.DLL
2019/03/19 12:44      7,680 KBDUSA.DLL
2019/03/19 12:44      7,680 KBDUSL.DLL
2019/03/19 12:44      7,680 KBDUSR.DLL
2019/03/19 12:44      8,192 KBDUSX.DLL
2019/03/19 12:44      7,680 KBDUZY.DLL
2019/03/19 12:44      7,680 KBDVNTC.DLL
2019/03/19 12:44      7,680 KBDWOL.DLL
2019/03/19 12:44      7,680 KBDYAK.DLL
2019/03/19 12:44      7,680 KBDYBA.DLL

C:\WINDOWS\system32>
C:\WINDOWS\system32>docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
f7ec5a41d630: Downloading [=====>]      13.4MB/27.14MB
9444bb562699: Download complete
6a4207b96940: Download complete
181cefd361ce: Downloading [=====>]      1.273MB/1.419MB
8a2090759d8a: Download complete
15f235e0d7ee: Waiting
d870539cd9db: Waiting
493aaa84617a: Waiting
bfc0e334fc78: Waiting
fac20d253f9d: Waiting
9350664305b3: Waiting
e47da95a5aab: Waiting
```

Figure.1 download image mysql

	TAG	IMAGE ID	CREATED	SIZE	
mysql	IN USE	latest			

mysql	IN USE	latest	0627ec6901db	19 days ago	555.7 MB	ⓘ RUN ▶
docker/getting-started	IN USE	latest	3ba8f2ff0727	about 2 months ago	27.94 MB	Inspect
hello-world	IN USE	latest	d1165f221234	2 months ago	13.34 KB	Pull
						Push to Hub
						Remove

Figure.2 Upload to dockerhub

```

Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  btrfs-tools debootstrap lxc rinse
The following packages will be REMOVED:
  lxc-docker lxc-docker-1.4.1
The following NEW packages will be installed:
  docker.io
0 upgraded, 1 newly installed, 2 to remove and 277 not upgraded.
Need to get 4,111 kB of archives.
After this operation, 9,960 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty-updates/universe docker.io amd64 1.0.1-dfsg1-0ubuntu1-ubuntu0.14.04.1_amd64.deb ...
Fetched 4,111 kB in 42s (95.9 kB/s)
(Reading database ... 164745 files and directories currently installed.)
Removing lxc-docker (1.4.1) ...
Removing lxc-docker-1.4.1 (1.4.1) ...
docker stop/waiting
Processing triggers for man-db (2.6.7.1-1) ...
Selecting previously unselected package docker.io.
(Reading database ... 164693 files and directories currently installed.)
Preparing to unpack .../docker.io_1.0.1-dfsg1-0ubuntu1-ubuntu0.14.04.1_amd64.deb ...
Unpacking docker.io (1.0.1-dfsg1-0ubuntu1-ubuntu0.14.04.1) ...
Processing triggers for man-db (2.6.7.1-1) ...
Processing triggers for ureadahead (0.100.0-16) ...
ureadahead will be reprofiled on next reboot
Setting up docker.io (1.0.1-dfsg1-0ubuntu1-ubuntu0.14.04.1) ...
addgroup: The group `docker' already exists as a system group. Exiting.
docker.io start/running, process 30975
Processing triggers for ureadahead (0.100.0-16) ...

```

Figure.3 Install the Docker in the ubuntu

(1) Methodology

Image can be built by docker commit instructions, or by instruction docker build in Dockerfile.

(2) structure dockerfile

command :

mkdir -p

cd dockerfile/df_testi

vin Dockerfile

Result :

```
# First dockerfile for test
FROM ubuntu:14.04
MAINTAINER dormancypress "dormancypress@outlook.com"
RUN apt-get update
RUN apt-get install -y nginx
EXPOSE 80
~
```

Figure.4 Docker file content

(3) structure image

docker build-t="dorrancypress/df_test1"

Result :

```
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.04
--> d0955f21bf24
Step 1 : MAINTAINER dormancypress "dormancypress@outlook.com"
--> Running in 4dee1e7433d9
--> a63e4da99ab4
Removing intermediate container 4dee1e7433d9
Step 2 : RUN apt-get update
--> Running in c13d9a85ca16
Ign http://archive.ubuntu.com trusty InRelease
Ign http://archive.ubuntu.com trusty-updates InRelease
Ign http://archive.ubuntu.com trusty-security InRelease
Hit http://archive.ubuntu.com trusty Release.gpg
Get:1 http://archive.ubuntu.com trusty-updates Release.gpg [933 B]
Get:2 http://archive.ubuntu.com trusty-security Release.gpg [933 B]
Hit http://archive.ubuntu.com trusty Release
Get:3 http://archive.ubuntu.com trusty-updates Release [63.5 kB]
Get:4 http://archive.ubuntu.com trusty-security Release [63.5 kB]
Get:5 http://archive.ubuntu.com trusty/main Sources [1335 kB]
```

Figure.5 Build dockerfile

Docker image

Result :

```
ky@ubuntu:~/dockerfile/df_test1$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
mancypress/df_test1 latest             4489535c6641       16 seconds ago    227.2 MB
mancypress/commit_test1 latest            aa6bdbc8178d       50 minutes ago    227.2 MB
mancypress/nginx     latest            299833dfa9ad       19 hours ago      206.4 MB
ubuntu               14.04.2           d0955f21bf24       2 weeks ago       188.3 MB
ubuntu               latest            d0955f21bf24       2 weeks ago       188.3 MB
ubuntu               trusty            d0955f21bf24       2 weeks ago       188.3 MB
ubuntu               trusty-20150320   d0955f21bf24       2 weeks ago       188.3 MB
ubuntu               14.04            d0955f21bf24       2 weeks ago       188.3 MB
ubuntu               quantal           c5881f11ded9       9 months ago      172.2 MB
ubuntu               12.10            c5881f11ded9       9 months ago      172.2 MB
ky@ubuntu:~/dockerfile/df_test1$
```

Figure.6 Docker image

(4) Verification access

curl http:127.0.0.1:49156

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Figure.7 Image Build - Up Success

1.3 Docker volume

1.3.1 Related concepts

There are two main ways to persist data in Docker:

- bind mount
- volume

Docker's data persistence Even if the data does not end with the end of the container, the data exists on the host machine-either in a specified directory of the host (using bind mount), or using the volume managed by docker (/var/ lib/docker/volumes).

(1)bind mount

Bind mount has been used by people since the early days of docker to mount the directory of the host machine to the container. But bind mount is not portable in different host systems. For example, the directory structure of Windows and Linux is different, and the host directory pointed to by bind mount cannot be the same. This is why the bind mount cannot appear in the Dockerfile, because the Dockerfile is not portable.

Mount the host-data directory in the current directory on the host machine to the /container-data directory in the container:

```
docker run -it -v $(pwd)/host-dava:/container-data alpine sh
```

There are a few things to note:

- The directory path of the host machine must be a full path (to be precise, it needs to start with / or ~/), otherwise docker will treat it as a volume instead of a volume.
- If the directory on the host machine does not exist, docker will automatically create the directory
- If the directory in the container does not exist, docker will automatically create the directory
- If the directory in the container already has content, docker will overwrite it with the directory on the host.

(2)volume

Volume also bypasses the file system of the container and writes data directly to the host machine, but the volume is managed by docker, and all volumes under docker are in the specified directory /var/lib/docker/volumes on the host machine.

Mount my-volume to the /mydata directory in the container:

```
docker run -it -v my-volume:/mydata alpine sh
```

Then developer can view the volume for my-volume.

The directory of volume on the host machine is /var/lib/docker/volumes/my-volume/_data. At this point, if my-volume does not exist, docker will automatically create my-volume and then mount it.

developer can also not specify the volume on the host:

```
docker run -it -v /mydata alpine sh
```

At this point, docker will automatically create an anonymous volume and mount it to the /mydata directory in the container. The directory path of the anonymous volume on the host machine is similar to:
/var/lib/docker/volumes/300c2264cd0acfe862507eedf156eb61c197720f69e7e9a053c87c2182b2e7d8/_data.

In addition to letting docker help developers automatically create a volume, developer can also create it developerrself:

```
docker volume create my-volume-2
```

Then mount this existing my-volume-2 to the container:

```
docker run -it -v my-volume-2:/mydata alpine sh
```

It should be noted that, unlike bind mount, if the volume is empty and the directory in the container contains content, then docker will copy the content in the container directory to the volume, but if there is already content in the volume, it will The directory in the container is overwritten.

VOLUME in Dockerfile

In the Dockerfile, the operator can also use the VOLUME instruction to declare that a directory in containr needs to be mapped to a volume:

```
#Dockerfile
```

```
VOLUME /foo
```

This means that when docker is running, docker will create an anonymous volume and bind this volume to the /foo directory of the container. If there is already content in the /foo directory of the container, it will copy the content to the volume. That is, VOLUME /foo in the Dockerfile has the same effect as docker run -v /foo alpine.

The VOLUME in the Dockerfile makes it automatically create an anonymous volume every time a new container is run. If data needs to be shared bet developersen different containers, the developer still needs to run docker run -it -v my-volume: The method of /foo stores the data in /foo in the specified my-volume.

Therefore, VOLUME /foo will be ambiguous at some point, and it will cause problems if developer don't understand it.

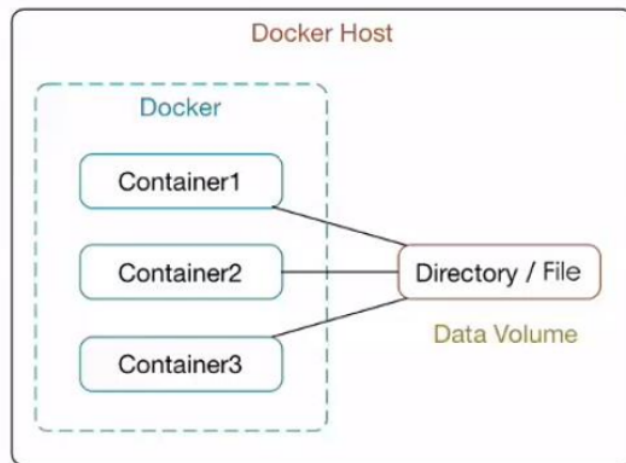


Figure.8 Data Volume and docker independence

1.3.2 Create Docker volume

(1)Enter a container

`docker run -it -v ~/datavolume:/data ubuntu /bin/bash`

```

drwxr-xr-x  2 root root 4096 Mar 20 05:22 bin
drwxr-xr-x  2 root root 4096 Apr 10 2014 boot
drwxr-xr-x  2 root root 4096 Apr 22 10:20 data[
drwxr-xr-x  5 root root 380 Apr 22 10:20 dev
drwxr-xr-x 64 root root 4096 Apr 22 10:20 etc
drwxr-xr-x  2 root root 4096 Apr 10 2014 home
drwxr-xr-x 12 root root 4096 Mar 20 05:21 lib
drwxr-xr-x  2 root root 4096 Mar 20 05:20 lib64
drwxr-xr-x  2 root root 4096 Mar 20 05:19 media
drwxr-xr-x  2 root root 4096 Apr 10 2014 mnt
drwxr-xr-x  2 root root 4096 Mar 20 05:19 opt
dr-xr-xr-x 246 root root   0 Apr 22 10:20 proc
drwx----- 2 root root 4096 Mar 20 05:22 root
drwxr-xr-x  7 root root 4096 Mar 20 05:21 run
drwxr-xr-x  2 root root 4096 Mar 20 06:16/sbin
drwxr-xr-x  2 root root 4096 Mar 20 05:19 srv
dr-xr-xr-x 13 root root   0 Apr 22 10:20 sys
drwxrwxrwt  2 root root 4096 Mar 20 05:22 tmp
drwxr-xr-x 11 root root 4096 Mar 20 06:16 usr
drwxr-xr-x 12 root root 4096 Mar 20 06:16 var

```

Figure.9 Enter a container

(2)Create a file

```
touch /data/c1
```

```
echo "I'm in container"> /data/c1
```

```
drwxr-xr-x  2 root root 4096 Apr 22 18:21 datavolume
drwxr-xr-x  2 jkxy jkxy 4096 Mar 22 11:43 Desktop
drwxrwxr-x 11 jkxy jkxy 4096 Apr 19 17:19 dockerfile
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Documents
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Downloads
-rw-r--r--  1 jkxy jkxy 8980 Jan 26 15:09 examples.desktop
drwxrwxr-x  2 jkxy jkxy 4096 Apr  8 20:56 fontconfig
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Music
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Pictures
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Public
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Templates
drwxr-xr-x  2 jkxy jkxy 4096 Jan 26 15:43 Videos
```

Figure.10 Create Successfully

(3)Looking for the ID

```
docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
MES				
c2ceb5e88164	ubuntu:14.04	"/bin/bash"	2 minutes ago	Exited (0) About a minute ago

Figure.11 Looking for the ID

(4)Follow id to view objects

```
docker inspect c2ceb5e88164
```

```

"NetworkSettings": {
  "Bridge": "",
  "Gateway": "",
  "GlobalIPv6Address": "",
  "GlobalIPv6PrefixLen": 0,
  "IPAddress": "",
  "IPPrefixLen": 0,
  "IPv6Gateway": "",
  "LinkLocalIPv6Address": "",
  "LinkLocalIPv6PrefixLen": 0,
  "MacAddress": "",
  "PortMapping": null,
  "Ports": null
},
"Path": "/bin/bash",
"ProcessLabel": "",
"ResolvConfPath": "/var/lib/docker/containers/c2ceb5e88164",
"RestartCount": 0,
"State": {
  "Error": "",
  "ExitCode": 0,
  "FinishedAt": "2015-04-22T10:21:49.001190068Z",
  "OOMKilled": false,
  "Paused": false,
  "Pid": 0,
  "Restarting": false,
  "Running": false,
  "StartedAt": "2015-04-22T10:20:34.321968271Z"
},
"Volumes": {
  "data": "/home/jkxy/datavolume"
},
"VolumesRW": {
  "data": true
}
}

```

Figure.12 Data volume was created successfully

1.4 Docker network

(1)Install bridge management tools

sudo apt-get install bridge-utils

```

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  bridge-utils
0 upgraded, 1 newly installed, 0 to remove and 63 not upgraded.
Need to get 29.2 kB of archives.
After this operation, 146 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty/main bridge-utils amd64 1.5-6ubuntu2 [29.2 kB]
Fetched 29.2 kB in 2s (11.2 kB/s)
Selecting previously unselected package bridge-utils.
(Reading database ... 196417 files and directories currently installed.)
Preparing to unpack .../bridge-utils_1.5-6ubuntu2_amd64.deb ...
Unpacking bridge-utils (1.5-6ubuntu2) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up bridge-utils (1.5-6ubuntu2) ...

```

(2)View bridge devices

sudo brctl show

bridge name	bridge id	STP enabled	interfaces
docker0	8000.56847afe9799	no	
***	8000.000000000000	no	

(3)Assign network Card

```
root@bdd49f7f5513:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:3/64 Scope:Link
          UP BROADCAST RUNNING MTU:1500 Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:648 (648.0 B)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

()Establish network interface:

bridge name	bridge id	STP enabled	interfaces
docker0	8000.56847afe9799	no	vethda1c8d9
***	8000.000000000000	no	

```
vethda1c8d9 Link encap:Ethernet  HWaddr 82:f7:c0:8d:53:9f
            inet6 addr: fe80::80f7:c0ff:fe8d:539f/64 Scope:Link
            UP BROADCAST RUNNING MTU:1500 Metric:1
            RX packets:8 errors:0 dropped:0 overruns:0 frame:0
            TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:648 (648.0 B)  TX bytes:648 (648.0 B)
```

Figure.13 Interface information

1.5 developersb Application

1.5.1 Deploy a static developersbsite

(1) Establish containers with port mapping and install nginx

```
docker run -p 80 --name developersb -i -t ubuntu
```

```
apt-get install -y nginx
```

```

Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  fontconfig-config fonts-dejavu-core geoip-database libfontconfig1
  libfreetype6 libgd3 libgeoip1 libjpeg-turbo8 libjpeg8 libtiff5
  libvpx1 libx11-6 libx11-data libxau6 libxcb1 libxdmcp6 libxml2 libxpm4
  libxslt1.1 nginx-common nginx-core sgml-base xml-core
Suggested packages:
  libgd-tools geoip-bin fcgiwrap nginx-doc sgml-base-doc debhelper
The following NEW packages will be installed:
  fontconfig-config fonts-dejavu-core geoip-database libfontconfig1
  libfreetype6 libgd3 libgeoip1 libjpeg-turbo8 libjpeg8 libtiff5
  libvpx1 libx11-6 libx11-data libxau6 libxcb1 libxdmcp6 libxml2 libxpm4
  libxslt1.1 nginx nginx-common nginx-core sgml-base xml-core
0 upgraded, 25 newly installed, 0 to remove and 0 not upgraded.
Need to get 5612 kB of archives.
After this operation, 19.8 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ trusty/main libgeoip1 amd64 1.6.0-1 [71.0 kB]
Get:2 http://archive.ubuntu.com/ubuntu/ trusty/main libxau6 amd64 1:1.0.8-1 [8376 B]
Get:3 http://archive.ubuntu.com/ubuntu/ trusty/main libxdmcp6 amd64 1:1.1.1-1 [12.8 kB]
Get:4 http://archive.ubuntu.com/ubuntu/ trusty/main libxcb1 amd64 1.10-2ubuntu1 [38.0 kB]
Get:5 http://archive.ubuntu.com/ubuntu/ trusty/main libx11-data all 2:1.6.2-1ubuntu2 [111 kB]
Get:6 http://archive.ubuntu.com/ubuntu/ trusty/main libx11-6 amd64 2:1.6.2-1ubuntu2 [560 kB]
9% [6 libx11-6 259 kB/560 kB 46%] 20.1 kB/s 4min 13s

```

Figure.14 stalling the nginx

(2) Use vim to create static pages

vim index.html

(3) Edit static developersbsite:

```

<html>

<head>

<title>Nginx in Docker-</title>

</ head>

<body>

<h1>Peng Xu 2006614</h1>

</ body>

</html>

```

(4) Configure nginx file

```

server {
    listen 80 default_server;
    listen [::]:80 default_server ipv6only=on;

    root /var/www/html;
    index index.html index.htm;

    # Make site accessible from http://localhost/
    server_name localhost;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;
        # Uncomment to enable naxsi on this location
        # include /etc/nginx/naxsi.rules
    }
}

```

Figure.15 nginx file

(5) Port Mapping

```

ubuntu:~$ docker port web
80 -> 0.0.0.0:49167
ubuntu:~$ docker top web
PID

```

Figure.16 Port Mapping

(6) Verify that the developersb site is running in the ubuntu browser

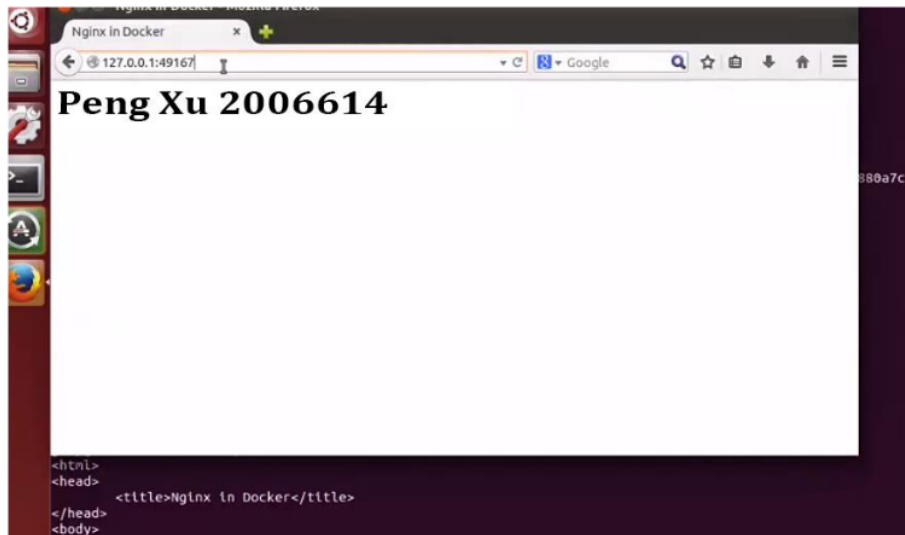


Figure.17 developersb page configuration is successful

The developersb page is successfully accessed.

2. Kubernetes

2.1 Related concepts

Kubernetes is Google's open source container orchestration tool. It is a product of the Borg system that has been running for N years in Google. Therefore, its maturity has been widely concerned by the industry since its birth, and it has quickly become an orchestration tool market. The mainstream of Docker, its community activity is very high, and the version iteration speed is also amazing. Its main function is to orchestrate Docker containers. Of course, Docker is just an engine for container tools. Kubernetes can support a variety of container engines, but from At present, the Docker container engine has absolute advantages. The container needs to be orchestrated and is easy to understand, because our core business is usually different forms of expansion of LNMT/P, but their operation of NMT/P is The order is that MySQL must be started first, then Tomcat or PHP, and finally Nginx. To control this order, developers need a container orchestration tool to help us achieve it. In addition, our business hopes to be online 7x24 hours, how to ensure ? It is difficult to achieve real-time by people, but the orchestration tool can. Kubernetes has helped us implement a lot of controllers. This controller can help us monitor the running status of the container and automatically rebuild the container (restart in the container era is to rebuild) the container , And can also automatically create a new Pod (the smallest unit in Kubernetes, there can be one or more containers in each Pod) according to the expansion rules defined by us when the container processing capacity is insufficient, and automatically delete it after the pressure drops. Pod and other functions.

Kubernetes:

It needs at least 3 Master nodes.

The main components of the Master node:

API Server: Mainly used to receive requests from other components and deal with them accordingly.

Scheduler: It is mainly responsible for selecting the best node to run the pod. It selects the appropriate node, usually in two stages. The first stage is to pass the preliminary screening first, according to the maximum requirement of running the pod, such as: 4G memory, 6 CPUs and other requirements are used for preliminary screening. If 3 of the 20 are selected, it will enter the second stage. When the most suitable node is selected from these 3, various factors need to be considered comprehensively. [11]Finally, the best running node will be selected and told to the API Server, and the Master node is responsible for scheduling the Pod to run on the node node.

Controller: It is responsible for monitoring whether the Pod running on the Node is working properly. If it finds that a Pod it monitors has failed, it will notify the API Server and ask the

Master to restart the same Pod on other nodes. Then kill the failed Pod.

Controller-Manager: It is responsible for monitoring whether all Controllers are working properly. In a Kubernetes cluster, there are many Controllers with different functions. The Controller that monitors the health of the Pod is just one of them. Controller-Manager is used to monitor the work of these Controllers. State, once the Controller fails, it will restart a new Controller to replace it.

But what if the Controller-Manager fails? In fact, when the Master is created, it requires at least 3 nodes. Each node will start a Controller-Manager, start one as the main Controller-Manager, and the other two or more as backup Controller-Managers. Once the main Controller-Manager If it fails, the standby Controller-Manager will continue to work instead of it.

2. 1. 1 Node0

Pod: It is the most basic unit in Kubernetes. A Pod is actually an abstraction of a container, or the shell of a container. It is actually one or more containers inside. Only one application can run in a container, so sometimes an application needs to rely on another application to work together. For example, if ELK wants to collect the log of each application, it needs to deploy its own log on each application node. Collectors, such as Logstash, for Nginx, if its logs are put into ELK, it needs to be put together with Logstash, but only one application can be run in a container, so it is necessary to run two containers in a Pod , To complete this requirement, so in the Kubernetes system, a Pod is taken as a whole and a basic atom is managed.

Label Selector: In a Kubernetes cluster, there are many Pods that need to be managed. They are scheduled to run on different Node nodes. So many Pods run on different Node nodes. How to manage them? Pass the string name? Then a Pod fails during operation and is restarted on other Nodes, so its string name must be changed, so how can this uniquely identify a Pod? So this requires the use of Label Selector (label selector), in fact, Pod in the Kubernetes system, they all have a type of label, such as: all Pods running Nginx will be labeled with App=Nginx, and all Pods running HAProxy will be labeled App=HAProxy. Type App=HAProxy, etc... Then when looking for all Pods running Nginx, developer can use the tag selector, pass the condition App, and the value of Nginx to query, developer can set all Pods that are App=Nginx to Mark it out and perform management operations.[12]

Kubelet: It runs on Node, accepts the start, stop or delete Pod request sent by APIServer, and according to the request, the daemon that actually performs operations on Node.

Docker: It is the operating environment provider of the actual Pod.

2. 1. 2 Pod

Autonomous Pod: It also sends a request to the API Server, and the API Server selects the best Node node through the Scheduler. Finally, kubelet creates a Pod in the docker environment, but once the Node node goes down, the Pod will disappear. Therefore, this method is not recommended.

Pod managed by the controller:

controller is divided into the following:

(1).Replication Controller (Replica Controller): It is an early Pod manager. It is responsible for receiving user requests to create a precisely specified number of Pods. If the user requests to create 2 NginxPods, it will automatically send a request to APIServer, and APIServer is letting The Scheduler selects the best Node node to create two Pods. If the currently created Pod is on Node3, and Node3 goes down after running for a period of time, ReplicationController will find that the number of replicas is not enough, and it will request APIServer again. It controls the Scheduler to re-select an optimal Node node to recreate a Pod, so that it can ensure that the number of Pods can always reach the number expected by the user. It also supports rolling updates: Simply put, if our image is upgraded now, developers need to start the Pod with a new version of the image, then the replica controller will request the APIServer to use the new image to create the Pod, the API is still the scheduling scheduler, choose the best Node, kubelet creates a new Pod based on the latest image. [13]At this time, there are three Pods. The replica controller will delete the old Pod at this time, and then request the APIServer to create a new Pod. Finally, delete the old Pod, which completes the rolling update. , Of course, it also supports rollback operations.

The following is the new version of the controller:

(2)ReplicaSet: Replica Set Controller

Deployment: ReplicaSet is not used directly. Deployment is generally used to manage stateless applications.

HPA (HorizontalPodAutoscaler: Horizontal Pod Autoscaler):

In actual work, when developers define a set of Pods, if it is two Pods, they can usually cope with users' smaller-scale visits, but

At a certain moment, the user visits suddenly increase, and 2 Pods are not enough to cope with it. At this time, HPA control can also be defined by itself, such as the CPU usage of two Pods.

The rate should be controlled at about 60%. HPA is calculated. If the CPU utilization rate is higher than 60%, 2 more Pods need to be added, then it will be automatically recreated

Two Pods are used to carry user access traffic. When the traffic drops and the CPU usage of the 4

Pods is very low, HPA will automatically destroy the extra 2 Pods.

(3) StatefulSet: Stateful Replica Set Controller

(4) DaemonSet: If developer need to run a Pod on a specific Node node, developer need to use it.

(5) Job: It is used to define a Pod that only needs to run a job once. For example, if developers need to start a Pod to clean up the garbage, developers can define a job. If the job is executed and the Pod exits, this is a normal expectation. If the Pod is running If the failure exits during the process, but the cleanup work is not completed, the Job will request to create a new Pod again to complete the remaining cleanup work.

Cronjob: Run periodic jobs.

These controllers are produced in order to complete the operation mode expected by the user.

2.1.3 Automatic service discovery:

Problem: Pod as a container provides some kind of service for clients, such as Nginx, etc. However, developers, Pod will inevitably have problems during operation. Therefore, it is necessary to destroy the problematic Pod and create a new Pod. The hostname and IP of the new Pod will change. How can the front-end client access these Pods that provide services normally?

The solution in Kubernetes:

provides an auto-discovery service. For the front-end client, it is a proxy. For the back-end Pod, it is the scheduler. How does the auto-discovery service work? It actually uses Label Selector. The automatic discovery service will first hand over the Pod label it manages to Label Selector. It will find all Pods that contain this label, then automatically discover the service, and then detect the IP of each Pod it finds. And ports, and use them as service Pods that developer can schedule.

What is the nature of the automatic discovery service? It is actually an ipvs rule. The iptables target port mapping rule was used in the early days, but its load balancing ability is too poor. In the new version of Kubernetes, the ipvs NAT rule is used instead. But developer need to pay attention to whether it is iptables DNAT rules or ipvs NAT rules, they all have an address for the client, that is, when the client accesses this address, iptables will do DNAT for the traffic to this address and convert it to its backend. The Pod's IP and port are the same as ipvs. It also converts

the traffic accessing the address into the back-end Pod's IP and port. Then this address is clearly known to the front-end client, and it is fixed; but these addresses are No ping! But they can be parsed because these addresses are only used as rules and are not configured on any interface,

which means that they are not supported by the TCP/IP protocol stack.

In addition, the client access to these Services (that is, the automatic discovery service) is accessed by name, and the DNS that resolves these names is the basic service in Kubernetes! These DNSs are called AddOns (attachment) components.[14]

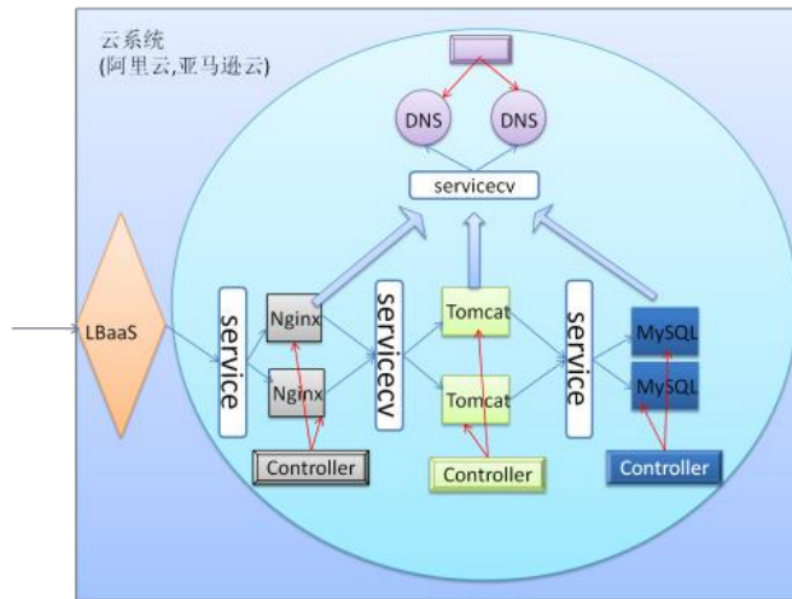


Figure.18 Schematic diagram of the main components of the Kubernetes system

Service: is the automatic discovery service

Controller: It is used to control the life cycle of the Pod. Each Controller is only responsible for controlling a part of the Pod. Once the Pod fails, it will automatically request the APIServer to create a new Pod to replace the failed Pod.

Pod: Pod such as Nginx, Tomcat, etc. If it wants to access other Pod, it needs to know who the front-end proxy (Service) of the Pod is. Therefore, developer need to tell it clearly during configuration. For example, Nginx needs to proxy user requests to Tomcat. developer need to know the domain name of Tomcat's front-end Service, and this domain name is notified to Nginx in advance. This is fixed. Nginx will first visit the Service of DNS, obtain the resolution address of the domain name, and then visit the Tomcat's front-end Service. The same goes for Tomcat to access the backend MySQL. Therefore, DNS service is a basic service in Kubernetes and must be provided.

If the domain name and IP of the service are changed, DNS will automatically trigger the modification of the corresponding A record.

In the Kubernetes system, the components that need to be created manually are: service, controller

If the Kubernetes system is deployed in a computer room, then it must have a set of edge physical hosts configured with public network IP so that it can access external traffic.

Of course, a scheduler can also be deployed separately outside Kubernetes, and the scheduler will send traffic to the Kubernetes intranet. But at this time the scheduler will not be within the manageable scope of Kubernetes, so the figure shows the situation of Kubernetes running on the cloud system. Use the LBaaS in these cloud systems to create a virtual scheduler so that Kubernetes can call The underlying cloud API is used to manage these virtual schedulers.

2.1.4 Three network models of Kubernetes

(1)When there are multiple containers in a Pod, the communication between them depends on the local loopback (lo)

(2)The communication between Pods must be completed through service.

In Docker, the communication between containers requires two NAT translations. That is, the container on NodeA needs to access the container on NodeB. It first sends the request to the gateway (usually the docker0 bridge), and then docker0 is forwarding the traffic. Before going out, developer must do a SNAT first, and then after reaching the NodeB, developer also need to do a DNAT to enter the container in the NodeB and realize the communication between the containers. But in Kubernetes, this is too cumbersome, and it will also make communication changes more confusing, because the container on NodeA does not know who the container node on the visited NodeB is, so it becomes like this: Pod sends the request to the gateway (docker0), docker0 and the physical network card are bridged. When docker0 wants to send a request to the target, it learns the address of the target by querying the local iptables rules or ipvs rules, so it forwards the traffic to the target. After the target receives the request, it does The response is sent back to its front-end service, and the service is proxying the request to the local Pod. Simple diagram:

NodeA 【Pod--->docker0->Service---】 ----->NodeB 【--->docker0--->Pod】

(3)Service has become the key to access, but service is only a rule.

Kube-proxy: It is used to manage the Service. After the user manually creates the service, the

subsequent management is done by kube-proxy. It will automatically monitor whether the Pod at the back end of the service has changed. If there is a change, it will immediately notify the API Server. After the API Server receives the notification, it will notify all associated subscribers of the message, such as kube-proxy. After the kube-proxy on a Node receives this notification message, it will automatically reflect this message in the local iptables or ipvs rules.

To API Server: It will receive all dynamic changes in the Kubernetes cluster. All changes must be notified to API Server first. Where is the data of API Server stored? In order to avoid data loss caused by API Server failure, these data need to be stored in a shared storage. This shared storage is etcd.

Etcd: It is a Key-Value storage database, similar to Redis, but it is more powerful and it supports more coordination mechanisms.

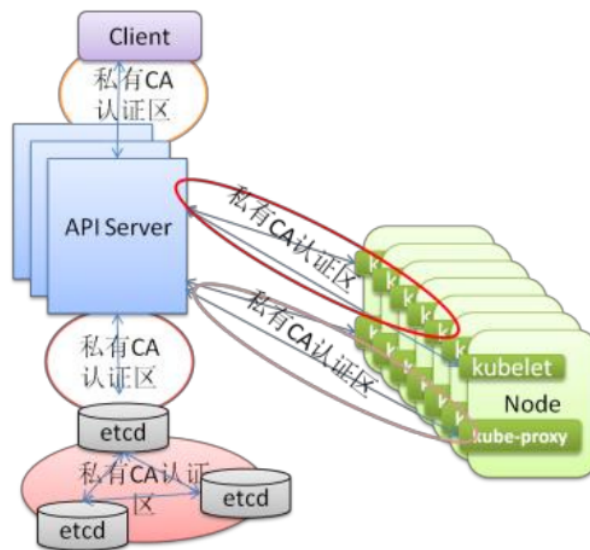


Figure.19 API Server and etcd directly use HTTP protocol to communicate

The communication between API Server and kubelet and kube-proxy is also HTTP protocol.

In order to avoid data theft, the CA authentication mechanism must be enabled between them, that is, the HTTPS protocol is used.

Based on all the above foundations, it is now simplified as shown below:

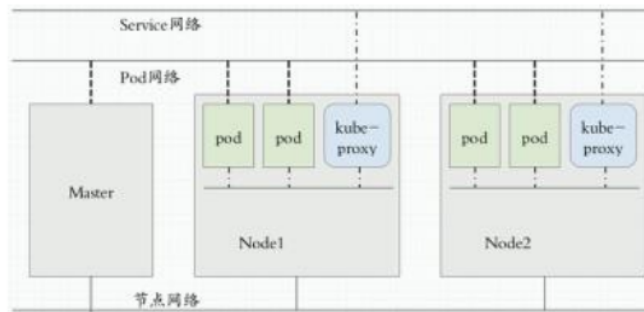


Figure.20 three network models of Kubernetes

The three types of networks shown in this figure:

Node network: It is a physical network that developers need to build when building a Kubernetes cluster.

Service network and Pod network need to introduce third-party network plug-ins through CNI (container network interface).

Because Kubernetes does not provide network support functions.

In the CNI specification of Kubernetes, it requires that the network plug-in needs to be implemented: network management and network strategy.

Network strategy: It is used to realize whether the Kubernetes network name space can communicate with each other, in a Kubernetes network name space, whether Pod and Pod are also developersd to communicate with each other, etc. These control strategies are implemented by iptables rules, Actually.

2.1.5 Kubernetes network namespace

This network name space is the management boundary provided by K8s, that is, developers can define multiple network name spaces, and each network name space can be used as a different project or different tenant. For example: A company applied for two network name spaces, One is used as the development environment, and the other is used as the production environment. When developers don't need the current production environment, and to use the upgraded production environment, developers only need to delete the network namespace of the existing production environment, so the k8s network The namespace actually provides a management boundary. But the Pods in the two network namespaces are on the same network, and they can communicate with

each other, so the network strategy is used to determine whether they can communicate.

(1)CNI:There are many products that implement the CNI specification. The following are the more developersll-known products:

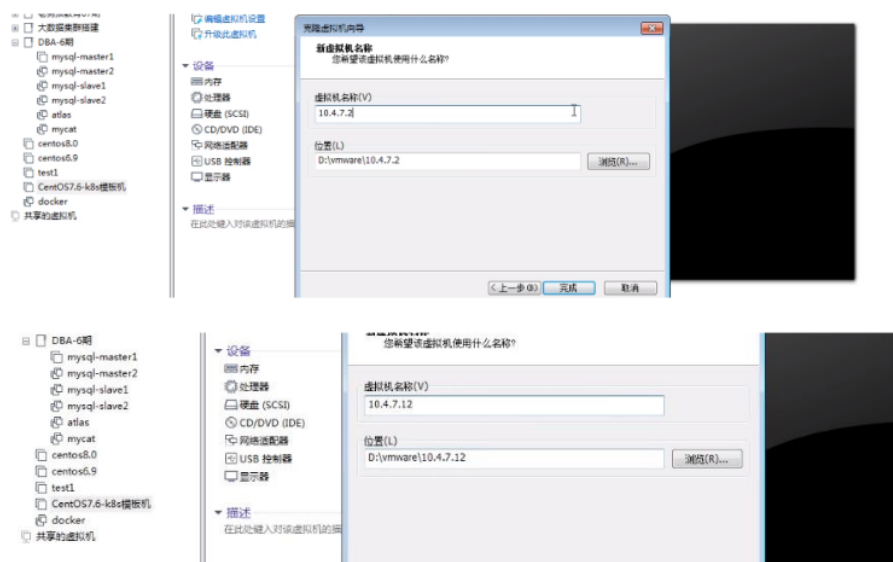
(2)Flannel: It is a relatively simple implementation of CNI, which only implements network configuration.

(3)Calico: It is a very complex implementation of CNI, which implements network configuration and network strategy, and internally implements a three-layer network tunnel for routing communication, and its routing protocol is BGP.

(4)Canel: It is a combination project of flannel and calico. It uses the simplicity of flannel network configuration and integrates calico's network strategy.

2.2 Application

(1)Prepare Virtual Machine



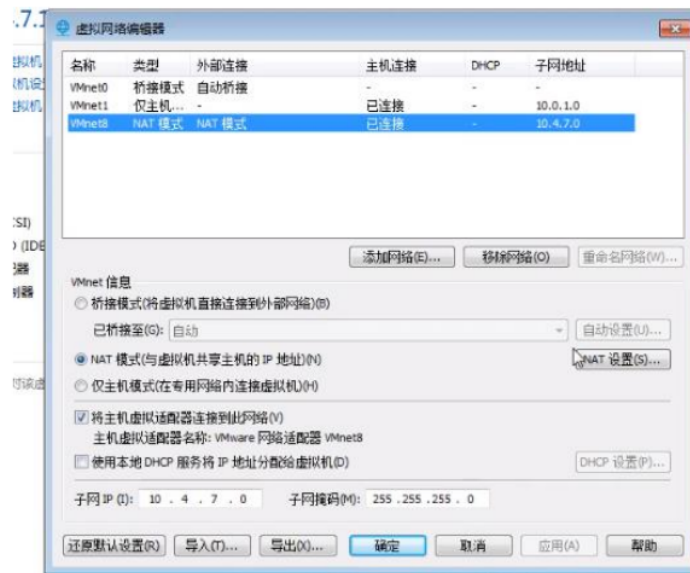


Figure.21 Build virtual machines

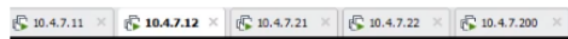


Figure.22 Virtual machine port

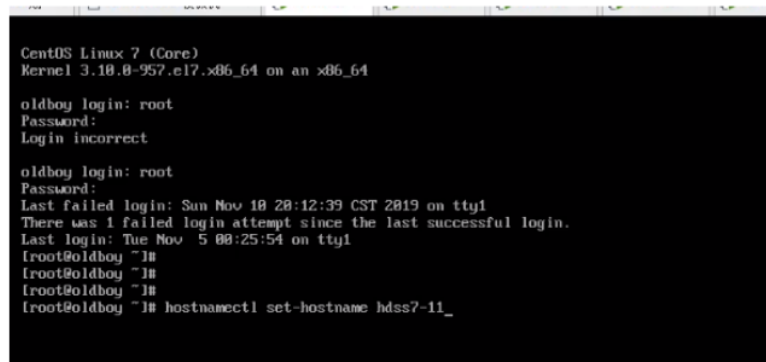


Figure.23 Configure virtual machines

(2)Log in to the virtual machines

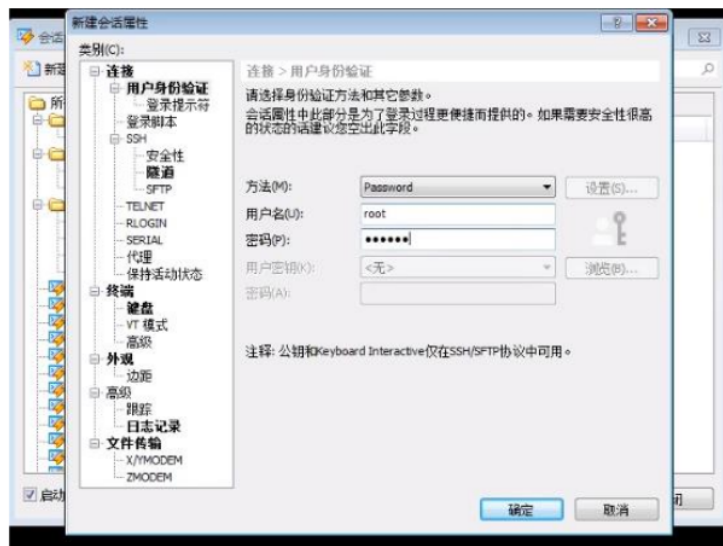


Figure.24 Login window

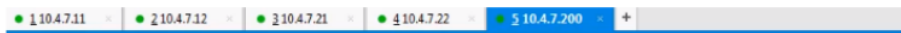


Figure.25 Virtual Machine Window

(3)system initialization command

Turn off the firewall

```
systemctl stop firewalld.service
systemctl status firewalld.service
systemctl disable firewalld
```

Close the Swap

```
swapoff -a
sed -ri 's/.*swap.*/&/' /etc/fstab
echo "vm.swappiness = 0" >> /etc/sysctl.conf
sysctl -p
```

Set the startup parameters

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

kubeadm init

```

[init] Using Kubernetes version: v1.21.0
[preflight] Running pre-flight checks
[WARNING Firewall] firewall is active, please ensure ports [6443 10250] are open or your cluster may not function correctly
[WARNING IsDockerSystemCheck] detected "cgroups" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/
r1/ [WARNING Swap] running with swap on is not supported. Please disable swap
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

^C
[root@localhost ~]# docker images
REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
registry.aliyuncs.com/google_containers/kube-apiserver    v1.21.0           4d217480042e      3 weeks ago      120MB
registry.aliyuncs.com/google_containers/kube-controller-manager v1.21.0           09780903cc37      3 weeks ago      120MB
registry.aliyuncs.com/google_containers/kube-scheduler    v1.21.0           62ad3129eca8      3 weeks ago      50.6MB

```

Figure.26 Initialization completed

```

REPOSITORY                                TAG                IMAGE ID           CREATED           SIZE
registry.aliyuncs.com/google_containers/kube-apiserver    v1.21.0           4d217480042e      3 weeks ago      120MB
registry.aliyuncs.com/google_containers/kube-proxy        v1.21.0           38dddb5fe90e      3 weeks ago      122MB
registry.aliyuncs.com/google_containers/kube-controller-manager v1.21.0           09780903cc37      3 weeks ago      120MB
registry.aliyuncs.com/google_containers/kube-scheduler    v1.21.0           62ad3129eca8      3 weeks ago      50.6MB
registry.aliyuncs.com/google_containers/pause             3.4.1             0f8457a4c2ec      3 months ago     683kB
registry.aliyuncs.com/google_containers/etcd              3.4.13-0          0369cf4383ff      8 months ago     253MB

```

Figure.27 Docker image

(4)Prepare docker

```

yum install -y yum-utils device-mapper-persistent-data lvm2
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

```

```

Installing:
yum-utils.noarch                                1.1.31-54.el7_8                base                                122 k
Updating:
device-mapper-persistent-data.x86_64            0:8.5-3.el7_9.2                updates                            423 k
lvm2.x86_64                                     7:2.02.187-6.el7_9.5           updates                            1.3 M
Installing for dependencies:
libnl2-python.x86_64                           2.9.1-6.el7.5                  base                                247 k
python-chardet.noarch                           2.2.1-3.el7                    base                                227 k
python-kitchen.noarch                           1.1.1-5.el7                    base                                267 k
Updating for dependencies:
device-mapper.x86_64                            7:1.02.170-6.el7_9.5           updates                            297 k
device-mapper-event.x86_64                      7:1.02.170-6.el7_9.5           updates                            192 k
device-mapper-event-libs.x86_64                 7:1.02.170-6.el7_9.5           updates                            192 k
device-mapper-libs.x86_64                       7:1.02.170-6.el7_9.5           updates                            375 k
lvm2-libs.x86_64                                7:2.02.187-6.el7_9.5           updates                            1.1 M

Installed:
yum-utils.noarch 0:1.1.31-54.el7_8

Dependency Installed:
libnl2-python.x86_64 0:2.9.1-6.el7.5                python-chardet.noarch 0:2.2.1-3.el7                    python-kitchen.noarch 0:1.1.1-5.el7

Updated:
device-mapper-persistent-data.x86_64 0:8.5-3.el7_9.2                lvm2.x86_64 7:2.02.187-6.el7_9.5

Dependency Updated:
device-mapper.x86_64 7:1.02.170-6.el7_9.5                device-mapper-event.x86_64 7:1.02.170-6.el7_9.5                device-mapper-event-libs.x86_64 7:1.02.170-6.el7_9.5
device-mapper-libs.x86_64 7:1.02.170-6.el7_9.5                lvm2-libs.x86_64 7:2.02.187-6.el7_9.5

Complete!

```

Figure.28 Install successfully

(5)Install the k8s Cluster

Docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

```
sh initMaster.sh
```

Figure.29 Initialization completed

The screenshot shows the 'Workload Status' section of the Kubernetes Dashboard. On the left sidebar, the 'Workloads' menu is selected. The main area displays four large green circles representing the status of different workload types: Daemon Sets, Deployments, Pods, and Replica Sets. The 'Daemon Sets' circle is highlighted with a red circle and a tooltip indicating '100%'.

Workload Type	Status
Daemon Sets	100%
Deployments	0%
Pods	0%
Replica Sets	0%

Figure.30 Installation of offline cluster

- [1] Wang min. construction of data science virtualization experimental platform based on docker [J]. Laboratory science, 2019,22 (3): 104-106
- [2] Ding Haibin, Cui Jun, Lu Kai. Design and implementation of Devops system based on docker [J]. Command information system and technology, 2017,8 (3): 87-92
- [3] Chen jianjuan, Liu Xingxing. Distributed elk log analysis system based on kubernetes [J]. Electronic technology and software engineering, 2016 (15): 211-212
- [4] TAHERIZADEH S,GROBELNIK M.Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications[J].Advances in Engineering

Software,2020,140(C):102734.

[5] BURNS B,GRANT B,OPPENHEIMER D,et al.Borg,Omega,and Kubernetes[J].Communications of the ACM,2016,59(5):50-57.

[6] developersng Fuyuan, Dan Xinghua, Yan Zhiyuan, et al. Design and practice of container cloud platform based on kubernetes [J]. Railway computer application, 2019,28 (12): 49-53

[7] Jinjin developersi. Design and implementation of docker distributed container automatic operation and maintenance system based on k8s [D]. Wuhan: Central South University for nationalities, 2018

[8] Wang developersijun. Construction of container cloud platform based on kubernetes [J]. Computer knowledge and technology, 2019,15 (36): 47-48

[9] PAN Y,CHEN I,BRASILEIRO F,et al.A performance comparison of cloud-based container orchestration tools[C]//2019 IEEE international conference on big knowledge (ICBK).Beijing,China:IEEE ,2019:191-198.

[10] Yu Changfa, Cheng Xuelin, Yang Xiaohu. Design and implementation of distributed tensorflow platform based on kubernetes [J]. Computer science, 2018,45 (11a): 527-531

[11] Zhou Jia developersi. Design and implementation of kubernetes cross cluster management [D]. Hangzhou: Zhejiang University, 2017

[12] Zhai Yarong, Yu Jingang. Analysis system for automatically collecting kubernetes logs based on filebeat [J]. Computer system applications, 2018,27 (9): 81-86

[13] MERCL L,PAVLIK J.Public cloud kubernetes storage performance analysis[C]//Computational collective intelligence.C ham:Springer ,2019.

[14] MEDEL V,TOLOSANA-CALASANZ R,BA Ñ ARES J Á, et al.Characterising resource management performance in Kubernetes[J].Computers & Electrical Engineering,2018,68:286-297.

FINAL GRADE

65/100

GENERAL COMMENTS

Instructor

A good attempt overall, a good attempt at the implementation with a reasonable report. Use of sources is fairly good too.

As a suggestion, endeavour to work on your academic writing a little - difficult in a second language, but this is where you can pick up the most marks. The report submitted is good, but the structure and written style are a bit limiting in terms of marks.

Still, a good attempt, well done.

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

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34