

assignment 2006702

by Yachao Li

FILE	25503_YACHAO_LI_ASSIGNMENT_2006702_343871_40873870.DOCX (4.45M)		
TIME SUBMITTED	09-MAY-2021 04:32PM (UTC+0100)	WORD COUNT	3923
SUBMISSION ID	152344556	CHARACTER COUNT	21765

ASSIGNMENT

ON

Cloud Engineering Principles and Practice

ACCA7027

BY

Yacha Li

2006702

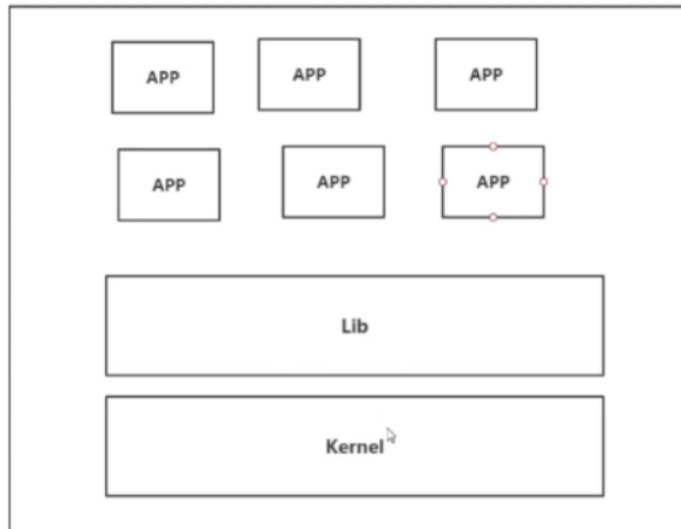
UNIVERSITY OF WALES TRINITY SAINT DAVID

UNIVERSITY OF WALES TRINITY SAINT DAVID

April, 2021

1. Benefits of containment

1.1 Virtual machine technology before

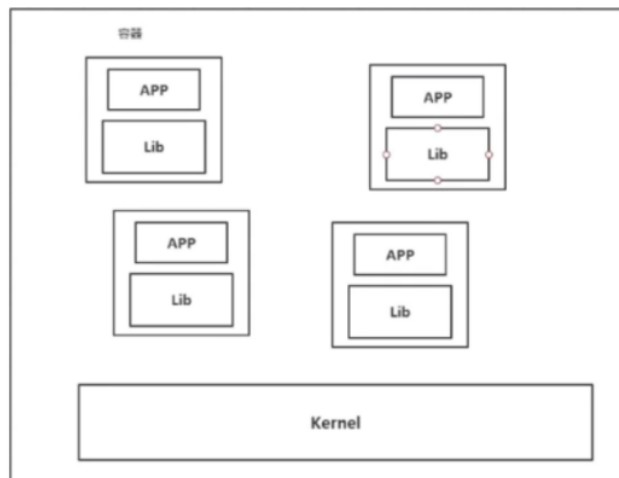


1.2 Disadvantages of virtual machine technology:

1. It takes up a lot of resources
2. Redundant steps
3. Slow start!

1.3 Containerization Technology

Containerization technology is not to simulate a complete operating system



1.4 Compare the differences between docker and virtual machine technologies

Traditional virtual machine, virtual out a hardware, run a complete operating system, and then install and run software on this system.

The application in the container runs directly in the host's content. The container does not have its own kernel and virtual hardware, so it is light.

Each container is isolated from each other, and each container has its own file system, which does not affect each other.

Devops (development, operation and maintenance)

Faster delivery and deployment of applications

Traditional: a pair of help documents, setup.

Docker: package the image, release the test and run it with one click.

More convenient upgrade and expansion

After using docker, we deploy applications just like building blocks.

The project is packaged as a mirror to expand server a! Server B.

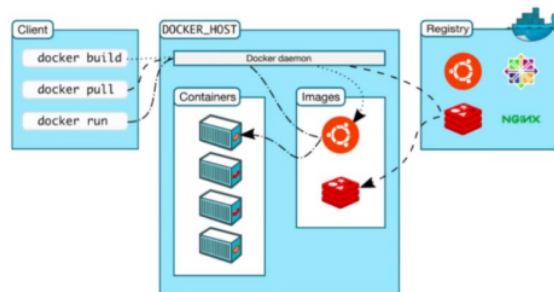
Simpler system operation and maintenance

After containerization, our development and test environments are highly consistent.

More efficient utilization of computing resources

Docker is a kernel level virtualization, which can run many container instances on a physical machine! Server performance can be squeezed to the extreme.

Basic components of docker



1.5 Eight scenarios of docker

1. Simplify configuration

The biggest advantage of virtual machine is that it can run all kinds of platforms (software, systems) with different configurations on your hardware facilities. Docker provides the same function while reducing the overhead. It allows you to summarize the running environment and configuration in the code and then deploy them. The configuration of the

same docker can be used in different environments, This reduces the coupling between hardware requirements and application environment

2. Code pipeline management

Code from the developer's machine to the final deployment in the production environment, need to go through a lot of backbone environment. And each intermediate environment has its own small difference. Docker provides a consistent environment for applications from development to online, making the code pipeline much simpler

3. Improve development efficiency

The common goals of developers in different environments are as follows:

- 1、 Want to make the development environment as close as possible to the production environment
- 2、 Want to build a development environment quickly

The memory of the machine in the development environment is usually small. When we used to use virtual environment, we often need to add memory for the machine in the development environment. Now docker can easily make dozens of services run in docker

4. Isolation application

Development will run different applications on one machine

- 1、 In order to reduce the cost, the server is integrated
- 2、 Split an integrated application into a single low coupling service (micro Service Architecture)

5. Integrate servers

The ability of docker to isolate applications enables docker to integrate multiple servers to reduce costs. Since there is no memory occupation of multiple operating systems and unused memory can be shared among multiple instances, docker can provide a better server integration solution than virtual machine

6. Debugging ability

Docker provides a lot of tools. These tools are not necessarily for containers, but they are suitable for containers. They provide many functions, including setting checkpoints for containers, setting versions, and checking the differences between the two containers. These features can help debug bugs

7. Multi tenant environment

In the application of multi tenant environment, it can avoid the rewriting of key applications. A special example of this scenario is to develop a fast and easy-to-use multi tenant environment for the application of lot (Internet of things). The basic code of this kind of multi tenant environment is very complex and difficult to handle. Replanning the application not only consumes time, but also wastes money

Using docker, you can create an isolated environment for multiple instances of each tenant's application layer, which is not only simple but also low-cost, because the docker environment starts quickly and diff command is very efficient

8. Rapid deployment

Docker creates a container for the process without starting an operating system

Generally, the resource utilization rate of data center is only 30%. The resource utilization rate can be improved by using docker and effective resource allocation

Second, the shortcomings of docker

The installation of docker is very easy. At present, docker supports all Linux systems (Ubuntu, RHEL, Debian, etc.). Through boot2docker virtual tool, docker can run normally under OS X and windows

Note: limitations of docker operating environment:

It must run on a 64 bit machine and only supports x86 at present_ AMD64 and amd64,32 systems are not supported

The Linux kernel of the system must be 3.8 or higher. The kernel supports device mapper, aufs, VFS, Btrfs and other storage formats

The kernel must support cgroups and namespace

Three related knowledge points of docker

Docker component (core): image, container and library.

Adopt C / S architecture: client (executive program) → communicate with waiting program (provide docker service) through command line and API.

Image: a read-only static template (framework system); Including environment and application execution code (framework language); Hierarchical mechanism is adopted. The new data is attached to the original data through the federated file system.

Container: a runtime environment, which is the running state of the image and a dynamic representation of the image execution.

Library: use the registration server (common or private) to store and share users' images. It is the directory where a specific user stores images.

*Docker features:

*Isolation: libcontainer (default container) => kernel namespace.

*Security: internal (cgroups) | container and host (kernel capability mechanism).

*Measurability: cgroups → resource measurement and allocation (users).

*Portability: aufs → layer concept.

9. The advantages of using aufs as the file system of docker container are as follows:

Save storage space: multiple containers can share a basic image storage

Fast deployment: avoid multiple replication operations when deploying multiple containers from the same underlying image

Upgrade convenience: upgrade a basic image to all containers based on it

Incremental modification: other directory files can be modified without changing the underlying image. All operations take place at the top write operation layer, which increases the shareable content of the underlying image

Advantages of docker (compared with LxC)

*Cross platform portability: packaged according to a unified standard (in the image), and can be propagated.

*Application oriented: Optimize application deployment (design philosophy) → API, interface and documentation (embodiment).

*Version control: track, query, record version information (application change history) and roll back version

*Component reuse: component building (basic image) → python (running environment) | PostgreSQL (basic image).

2. Portability

2.1 Tomcat image

1. Prepare the image file Tomcat compression package, JDK compression package.

```
root@localhost tomcat# ls -l
总用量 194548
-rw-r--r--. 1 root root 9455895 6月 13 14:07 apache-tomcat-8.0.53.tar.gz
-rw-r--r--. 1 root root 189756259 6月 13 14:50 jdk-8u161-linux-x64.tar.gz
root@localhost tomcat#
```

2. Write dockerfile file, official name: dockerfile, build will automatically find this file.

FROM centos

MAINTAINER lyc<123456@qq.com>

COPY readme.txt /usr/local/readme.txt

ADD jdk-8u161-linux-x64.tar.gz /usr/local/

ADD apache-tomcat-8.0.53.tar.gz /usr/local

RUN yum -y install vim

ENV MYPATH /usr/local

WORKDIR \$MYPATH

ENV JAVA_HOME /usr/local/jdk1.8.0_161

ENV CLASSPATH \$JAVA_HOME/lib/dt.jar:\$JAVA_HOME/lib/tools.jar

ENV CATALINA_HOME /usr/local/apache-tomcat-8.0.53

ENV CATALINA_BASH /usr/local/apache-tomcat-8.0.53

ENV PATH \$PATH:\$JAVA_HOME/bin:\$CATALINA_HOME/lib:\$CATALINA_HOME/bin

EXPOSE 8080

CMD /usr/local/apache-tomcat-8.0.53/bin/startupsh && tail -F /usr/local/apache
- tomcat-8.0.53/bin/logs/catalina.out

Building a mirror image

docker build -t diytomcat .

3. Start the image and create the container

docker run -d -p 9090:8080 --name kuangshentomcat02 -v

```
/home/kuangshen/build/tomcat/test:/usr/local/apache-tomcat-8.0.53/webapps/test
```

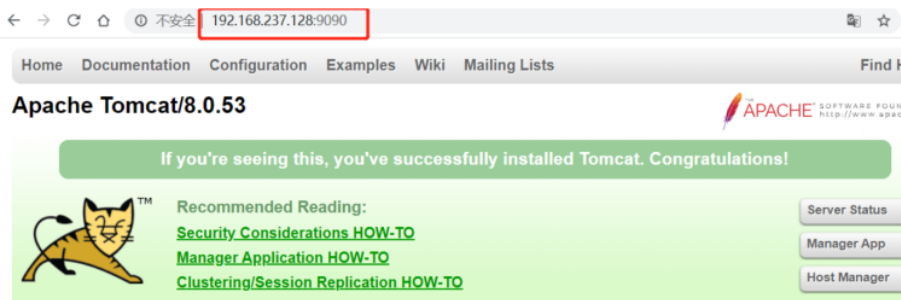
```
-
```

```
v
```

```
/home/kuangshen/build/tomcat/tomcatlogs:/usr/local/apache-tomcat-8.0.53/logs
```

Diytomcat

4. Access test



3. Publishing images

1. Published on <https://hub.docker.com/>

2. Login account

3. Submit your own image on our server

```
[root@localhost tomcat]# docker login --help
```

```
Usage: docker login [OPTIONS] [SERVER]
```

Log **in** to a Docker registry.

If no server is specified, the default is defined by the daemon.

Options:

-p, --password string Password

--password-stdin Take the password from stdin

-u, --username string Username

Lod in dockerhub

```
[root@localhost tomcat]# docker login -u #####
```

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

4. After login, we can submit the image, that is, one-step docker push.

`docker push diytomcat`

```
[root@localhost tomcat]# docker tag bb64ab96b432 ztxl15/tomcat:1.0
[root@localhost tomcat]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
diytomcat	latest	bb64ab96b432	5 hours ago	695MB
ztxl15/tomcat	1.0	bb64ab96b432	5 hours ago	695MB
mycentos	0.1	c987078b06cb	20 hours ago	321MB
ztx/centos	latest	717fc66b623b	45 hours ago	237MB
tomcat	9.0	2eb5a120304e	3 days ago	647MB

4. Container volumes

4.1 what is a container data volume?

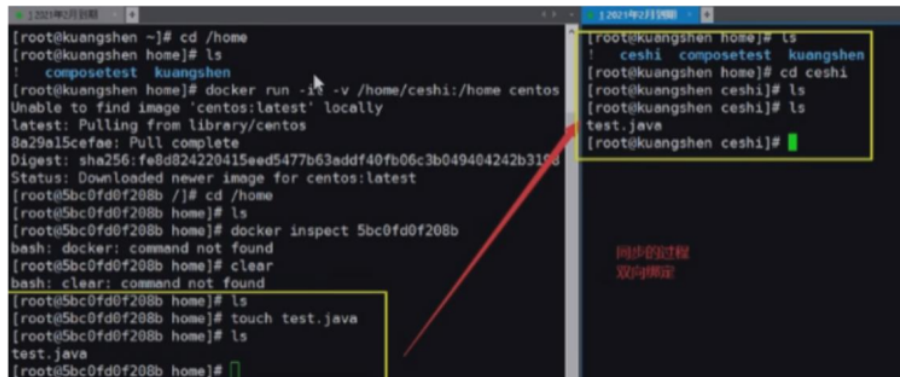
There can be a data sharing technology between containers! The data generated in the docker container is synchronized locally. This is volume technology. Directory mount, the directory in our container, mount to Linux. To sum up, container persistence and synchronization! Data can also be shared between containers. To sum up: container persistence and synchronization! Data can also be shared between containers.

Method 1: directly use the command to mount

Docker run - it - v Host Directory: container directory test[root@localhost When the docker run - it - v / home / ceshi / home CentOS / bin / Bash is started, we can

check the mount status through the docker inspect container ID:

Synchronization of test files



The screenshot shows two terminal windows. The left window is a host terminal with the following commands and output:

```
[root@kuangshen ~]# cd /home
[root@kuangshen home]# ls
! composetest kuangshen
[root@kuangshen home]# docker run -it -v /home/ceshi:/home centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31f4
Status: Downloaded newer image for centos:latest
[root@5bc0fd0f208b /]# cd /home
[root@5bc0fd0f208b home]# ls
[root@5bc0fd0f208b home]# docker inspect 5bc0fd0f208b
bash: docker: command not found
[root@5bc0fd0f208b home]# clear
bash: clear: command not found
[root@5bc0fd0f208b home]# ls
[root@5bc0fd0f208b home]# touch test.java
[root@5bc0fd0f208b home]# ls
test.java
[root@5bc0fd0f208b home]#
```

The right window is a container terminal with the following commands and output:

```
[root@kuangshen home]# ls
! ceshi composetest kuangshen
[root@kuangshen home]# cd ceshi
[root@kuangshen ceshi]# ls
[root@kuangshen ceshi]# ls
test.java
[root@kuangshen ceshi]#
```

A red arrow points from the 'test.java' file in the container terminal to the 'test.java' file in the host terminal, indicating file synchronization.

If you add or modify a file in the specified directory of the container, it will be synchronized to the specified directory of the host. On the contrary, if you do related operations in the host directory, it will be synchronized to the corresponding directory of the container.

Methods 2 and 3: named and anonymous mount.

Anonymous mount: `docker run -d -P --name nginx01 -v /etc/nginx nginx`

Mount by name: `docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx nginx`

All the volumes in the docker container are `/var/lib/docker/volumes/xxxx/_data`, We can easily find one of our volumes through a named mount. In most cases, we use a named mount.

How to determine whether it is a named mount or an anonymous mount, or a specified path mount.

`-V path in container # anonymous mount`

`-V volume name: path in container # named mount`

-V host path: in container path # specified path mount.

Others.

Change the read and write permissions through the path in - V container: RO or RW

Ro readonly

RW readwrite is readable and writable.

Once the container permission is set when creating the container, the container will limit the content we mount! `docker run -d -P --name nginx05 -v juming:/etc/ nginx:ro nginx` `docker run -d -P --name nginx05 -v juming:/etc/ nginx:rw nginx`

The default is RW

Ro as long as you see RO, it means that this path can only be operated through the host, but not inside the container.

5. Container networking

`docker run -d -P --name tomcat01 tomcat`

Check the internal network address IP addr of the container, and find that when the container starts, it will get a `eth0@if43` IP address, assigned by docker.

Every time we start a docker container, the docker will assign an IP to the docker container. As long as we install the docker, we will have a docker 01 network card.

Bridge mode, the use of technology is Veth pair technology.

The IP addr is tested again and a pair of network cards are found:

```

root@localhost /# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:00:92:19 brd ff:ff:ff:ff:ff:ff
    inet 192.168.237.128/24 brd 192.168.237.255 scope global noprefixroute dynamic ens33
        valid_lft 1356sec preferred_lft 1356sec
    inet6 fe80::df9d:b875:65dc:c2b9/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:c3:11:ca brd ff:ff:ff:ff:ff:ff
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:c3:11:ca brd ff:ff:ff:ff:ff:ff
5: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:9d:e5:de:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:9dff:fe5:deb4/64 scope link
        valid_lft forever preferred_lft forever
43: veth0d1477901f42: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 02:4e:1b:7e:59:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::4e:1bff:fe7e:5937/64 scope link
        valid_lft forever preferred_lft forever

```

Start a container test again and find another pair of network cards.

```

root@localhost /# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:00:92:19 brd ff:ff:ff:ff:ff:ff
    inet 192.168.237.128/24 brd 192.168.237.255 scope global noprefixroute dynamic ens33
        valid_lft 1724sec preferred_lft 1724sec
    inet6 fe80::df9d:b875:65dc:c2b9/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:c3:11:ca brd ff:ff:ff:ff:ff:ff
4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:c3:11:ca brd ff:ff:ff:ff:ff:ff
5: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:9d:e5:de:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:9dff:fe5:deb4/64 scope link
        valid_lft forever preferred_lft forever
43: veth0d1477901f42: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 02:4e:1b:7e:59:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::4e:1bff:fe7e:5937/64 scope link
        valid_lft forever preferred_lft forever
45: veth45199f001f44: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 5a:fd:c0:2b:29:58 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::58fd:c0ff:fe2b:2958/64 scope link
        valid_lft forever preferred_lft forever

```

We found that the network cards brought by this container are a pair. Veth pair is a pair of virtual device interfaces. They all appear in pairs. One is connected to the protocol, and the other is connected to each other. Because of this feature, Veth pair acts as a bridge to connect various virtual network devices, openstack, docker containers, The connection of OVS is based on Veth pair technology. Docker uses Linux bridge technology, and the host is docker container bridge docker 0

Custom network

View all docker networks

```
| root@localhost ~/ # docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
461bf576946c        bridge             bridge              local
c501704cf28e        host               host                local
9354fbcc160f        none              null                local
```

Network mode

Bridge: Bridge (docker is created by default and also uses bridge mode.)

None: no network configuration

Host: share network with host

Container: container network connectivity, container direct interconnection(Use less!
It's very limited.)

docker run -d -P --name tomcat01 tomcat docker run -d -P --name tomcat01 --net
bridge tomcat.

We can customize a network!

--Driver bridge network mode is defined as bridge

--Subnet 192.168.0.0/16 defines subnet, the range is 192.168.0.2 ~ 192.168.255.255

-- gateway 192.168.0.1 subnet gateway is 192.168.0.1

```
docker network create --driver bridge --subnet 192.168.0.0/16 --gateway
192.168.0.1 mynet
```

#docker network ls

```
root@localhost [/] # docker network inspect mynet
[
  {
    "Name": "mynet",
    "Id": "7ee3adf259c8c3d86fce6fd2c2c9f85df94e6e57c2dce5449e69a5b024efc28c",
    "Created": "2020-06-14T01:03:53.767960765+08:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.0.0/16",
          "Gateway": "192.168.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

When we use the customized network, docker has already helped us maintain the corresponding relationship, so we are recommended to use the network in this way.

Benefits:

Redis - different clusters use different networks to ensure the security and health of the cluster

MySQL -- different clusters use different networks to ensure the security and health of the cluster.

Network connectivity

```
[root@localhost /]# docker exec -it tomcat01 ping tomcat-net-01
ping: tomcat-net-01: Name or service not known
[root@localhost /]# docker network --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect    Connect a container to a network
  create     Create a network
  disconnect Disconnect a container from a network
  inspect    Display detailed information on one or more networks
  ls        List networks
  prune     Remove all unused networks
  rm        Remove one or more networks
```

```
[root@localhost /]# docker network connect --help

Usage:  docker network connect [OPTIONS] NETWORK CONTAINER

Connect a container to a network

Options:
  --alias strings      Add network-scoped alias for the container
  --driver-opt strings driver options for the network
  --ip string          IPv4 address (e.g., 172.30.100.104)
  --ip6 string         IPv6 address (e.g., 2001:db8::33)
  --link list          Add link to another container
  --link-local-ip strings Add a link-local address for the container

[root@localhost /]#
```

```
{
  "Containers": {
    "b168a37d31fcdc2ff172fd969e4de6de731adf53a2960eeae3dd9c24e14fac67": {
      "Name": "tomcat-net-01",
      "EndpointID": "f0aflc33fc5d47031650d07d5bc769e0333da0989f73f4503140151d0e13f789",
      "MacAddress": "02:42:c0:a8:00:02",
      "IPv4Address": "192.168.0.2/16",
      "IPv6Address": ""
    },
    "bd2fff343bc21b3fd7f7be7302b4b4753b372a64da3925fe26e22a951be5e27c1": {
      "Name": "tomcat01",
      "EndpointID": "ee9c5f0a7562563214e2daecc46aca656699d7c4793fed5c998600e94e1fa59e",
      "MacAddress": "02:42:c0:a8:00:04",
      "IPv4Address": "192.168.0.4/16",
      "IPv6Address": ""
    },
    "c07d634e17152ca27e318c6fc6c02e937e6d5e7a1631676a39166049a44c03c": {
      "Name": "tomcat-net-02",
      "EndpointID": "ba114b9bd5f3b75983097aa82f71678653619733efc1835db857b3862e744fbc",
      "MacAddress": "02:42:c0:a8:00:03",
      "IPv4Address": "192.168.0.3/16",
      "IPv6Address": ""
    }
  }
}
```

If you want to operate others across the network, you need to use docker network connect.

6. Creating your own application

I use MySQL and redis tools on docker to do car rental system.

carName	carColor	carType	status	production	operating
TSL	YELLOW	Large truck	out stock	scheduled	edit delete add
TSL	YELLOW	Large truck	in stock	unscheduled	edit delete add
MOORE	BLUE	small truck	in stock	unscheduled	edit delete add
SPYMAN	BLUE	Large truck	in stock	unscheduled	edit delete add
CHINATRUCK	WHITE	big truck	in stock	unscheduled	edit delete add

7. Kubernetes

7.1 The old way of virtualization

In the early days, the only way to use more computing power was to add more hardware in the physical sense (hence the old name "server farm"). Then, virtual machine (VM) was introduced and the concept of virtualization was born. In the past 20 years, this concept has been playing an increasingly important role. This software introduces a virtual hardware abstraction layer (HAL), which in turn enables multiple operating systems (OS) and applications to run on the same underlying hardware of the parent OS. Although this approach does have advantages in the field of current IT infrastructure, especially when it involves specific complex software objectives, and it does continue to have advantages in market share, it is not without defects.

Full implementation of virtual machine (I call it virtualization 2.0 for comparison) can quickly become a greedy monopolist of system resources, so we always need faster and more expensive hardware. This is especially true in mixed settings, where multiple competing operating systems are required on the same computer (not all applications are always available on the same operating system) due to the array of applications that must be running and their nature. As a result, the software of the virtual machine monitor or hypervisor becomes necessary, which makes the system more complex. Once the setup is complete, the VM implementation becomes heavyweight and inflexible. After all, in order to run an application, each VM must contain not only the full space of the operating system itself, but also all the libraries and dependencies (LIBS / bins) of the entire stack (operating system, device drivers, applications, etc.). Each VM must also simulate the virtual version of the underlying

hardware, so it does not directly call IRQ (interrupt request), but further hinders performance through software proxy. Combining this into a real-world cloud scenario is suitable for organizations with large-scale or strict business objectives, all of which eventually turn into tossing implementation, software complexity, limited portability, wasted CPU cycles and GB, and paying more dollars for hardware and labor from the perspective of IT management. Virtualization is neither cheap nor easy to implement.

Framework for running VM.

7.2 New method of virtualization

In the absence of a better term, we use virtualization 3.0, which is containerization. Software container is not a new solution, because the concept has existed for about 20 years. However, in 2004-2008, Solaris introduced Solaris zones and created an independent application environment, while Google used containers to deal with production business. At the same time, Linux released kernel compiler 2.6.24, which laid the foundation for the maturity of LxC (Linux container). Fast forward to 2013 and beyond: virtualization in the form of containers, not just virtual machines, has been widely recognized and has good reasons to use. The kubernetes project was originally designed and created by Google, which opened it up in 2014. It released version 1.0 in 2015 and teamed up with the Linux foundation to form the cloud native Computing Foundation, which is now in charge of kubernetes.

Containers are to VM what ballerinas are to sumo wrestlers, and you'll soon see why I draw that analogy. If your organization needs a very specific form of virtualization implementation, which may be due to a variety of reasons (that is, running a complex software environment), then VM may still be your ticket. For others, containers are the best choice, at least initially. The key difference is that the containers all run on the same host operating system, so there is no need to copy the operating system like every virtual machine. In addition, the container utilizes the LIBS and bins of the host operating system through direct API calls, and only encapsulates some additional LIBS and bins in its local module, which are specific to the application they are running. For these reasons, the container is lightweight and portable, and has far fewer resources than any VM you might run.

These saved CPU cycles and GB can be used in two different schemes: to make the same group of container applications run with higher system performance and response time, or to make better use of the underlying hardware by running more container applications. If you need to restart the container application after a software failure or security update, you only need a few seconds - far less than booting the VM in the same scenario, let alone a mixed VM configuration. Container

virtualization, of course, comes with its own few - arguably less extreme drawbacks.

First, the container should be light, portable and easy to manage. However, if your business relies on complex applications, you need to run multiple containers. This increases diversity and eventually leads to performance bottlenecks as containers compete for access to protected system resources (operating system and hardware). Second, the container is not as secure as VM by default. This is because they only provide process level security, not operating system level security. There is only one operating system running, that is, the host operating system (any security vulnerability of the host operating system will automatically become a threat to the container). In the case of critical security, additional software is reasonable. Third, because it's perfectly reasonable to run the entire collection of containers on any physical system and optimize the delivery of a comprehensive SaaS (software as a service) solution, managing them in real time is no longer a manual task. In fact, it requires software similar to the VMM, such as the software needed to manage multiple VMS.

Author: Tony Bai

Link: <https://www.imooc.com/article/288155>

Source: MOOC

This article was originally published on MOOCS. Please indicate the source for reprint.
Thank you for your cooperation

7.3 K8s

K8s is an open source container cluster management system, which can realize the automatic deployment, expansion and maintenance of container cluster. K8s is the coordination and scheduling container, so that they can run in the best way and access system resources at any time when the load fluctuates all day. Google is an expert in the field because it has deployed a large number (possibly the highest) of servers and applications over the past two decades. The development and design of kubernetes is highly influenced by the Google Borg system, which was created to manage large-scale clusters. K8s can be deployed internally or in the cloud, enabling you to run container applications across multiple physical machines and maintain a fail safe monitoring layer for restarting a collapsed container. It can also easily publish updates (including rollback), and automatically expand to close and start containers based on various performance metrics. Last but not least, k8s allows containerized applications to run in a consistent deployment, making it easier to monitor them through third-party software such as Prometheus, Jaeger, efk (elastic search, fluent,

kibana) stacks or ccadvisor. K8s and simplified cm (Container Management) solutions are basically synonymous.

1. Failover: When a node is shut down or hung up, services on that node are automatically moved to another node without interruption. This is impossible for docker or ordinary virtual machine

2.Resource Scheduling: the node can be extended if the CPU and memory are insufficient on the node. A new pod is scheduled to a newly expanded node by Kube scheduling.

3. Resource isolation: create three namespace: It is an extension, a timer, a friend, and a mask. It is an empty pod, and the operation and the supplier name empty empty pod. This is a mutual association.

In a traditional host or docker environment, logging in will see all the services or containers.

4. Since the docker container is used, the process does not interfere.

5. Security: different roles have different access rights, such as view probe and delete probe; RBAC authentication increases the security of k8s

Deploy applications quickly and accurately

Limit only hardware usage to the resources you need

Cabernet Sauvignon

Mobile: public cloud, private cloud, hybrid cloud, polymodal cloud

Extensible: modular, plug-ins, attachable, combinable

Auto repair: automatic placement, automatic reboot, automatic replication, automatic scaling

Load balancing

The k8s can make updates to new versions more quickly, packaging applications and interrupting service updates without having to break the server, and the transition from the development environment to the test and the environment is very useful.

7.4 Kubernetes components are composed of:

kubectl

The client command-line tool formats the received commands and sends them to the Kube apiserver as the operation entrance of the whole system.

kube-apiserver

As the control entrance of the whole system, rest API service provides the interface.

kube-controller-manager

It is used to perform background tasks in the whole system, including node status, pod number, pods and Service Association, etc.

kube-scheduler

Responsible for node resource management, accept the task of creating pods from Kube apiserver, and assign it to a node.

etcd

Responsible for service discovery and configuration sharing among nodes.

kube-proxy

It runs on each computing node and is responsible for pod network agent. Get service information from etcd regularly to make corresponding policies.

kubelet

Running on each computing node, as an agent, it receives the pods task assigned to the node and manages the container, periodically obtains the container state, and feeds back to the Kube apiserver.

DNS

An optional DNS service is used to create DNS records for each service object, so that all pods can access the service through DNS.

Author: gag

Link: <https://www.imooc.com/article/275614>

Source: MOOC

8. The following will introduce a simple example of MySQL deployment

JSP pages directly access MySQL database and display data through JDBC.

In this example, we need to start two containers: webapp container and MySQL container.

Download MySQL and Tomcat images:

```
# docker pull mysql:5.5

# docker pull kubeguide/tomcat-app:v1
```

Step 2: start MySQL service

First, we need to create an RC definition file for MySQL service: mysql-rc.yaml. The code content is as follows:

```
1 apiVersion: v1
2 kind: ReplicationController      ## 副本控制器RC
3 metadata:
4   name: mysql                  ## RC的名称，全局唯一
5 spec:
6   replicas: 1                  ## Pod副本期待数量
7   selector:
8     app: mysql                  ## 符合目标的Pod拥有此标签
9   template:                    ## 根据此模板创建Pod实例
10    metadata:
11      labels:
12        app: mysql              ## 副本Pod拥有的标签对应RC的selector
13    spec:
14      containers:               ## Pod内容器定义部分
15        - name: mysql           ## 容器的名称
16          image: mysql: 5.5      ## 容器的镜像
17          ports:
18            - containerPort: 3306 ## 容器暴露的端口号
19          env:                   ## 注入容器的环境变量
20            - name: MYSQL_ROOT_PASSWORD
21              value: "123456"
```

After creating mysql-rc.yaml file, publish it to kubernetes cluster.

```
kubect1 create -f mysql-rc.yaml
```

Next, look at the created RC and pod:

```
1 ]# kubectl get rc
2 NAME          DESIRED   CURRENT   READY   AGE
3 mysql         1         1         1       1h
4 ]# kubectl get po
5 NAME          READY   STATUS    RESTARTS   AGE
6 mysql-px9th   1/1     Running   0          1h
```

After the above creation has no problem, we will create a MySQL service file associated with it: mysql-svc.yaml. The code is as follows:

```
1 apiVersion: v1
2 kind: Service  ## 声明是Kubernetes的service文件
3 metadata:
4   name: mysql  ## service的全局唯一名称
5 spec:
6   ports:
7     - port: 3306  ## 服务暴露的端口号
8     selector:    ## service对应的Pod标签名称
9       app: mysql
```

Run the kubectl command to create a service:

```
kubectl create -f mysql-svc.yaml
```

View the created service:

```
]# kubectl get svc
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
mysql     ClusterIP   10.107.188.209  <none>        3306/TCP   1h
```

Step 3: start Tomcat application

Create the RC file of Tomcat: tomcat-rc.yaml. The code content is as follows:

```

1 apiVersion: v1
2 kind: ReplicationController
3 metadata:
4   name: myweb
5 spec:
6   replicas: 1      # spec.selector与spec.template.metadata.labels, 这两个字段必须相同, 否则下一步创
7   selector:
8     app: myweb
9   template:
10    metadata:
11      labels:
12        app: myweb
13    spec:
14      containers:
15        - name: myweb
16          image: kubeguide/tomcat-app:v1
17          ports:
18            - containerPort: 8080
19          env:
20            - name: MYSQL_SERVICE_HOST
21              value: 'mysql'
22            - name: MYSQL_SERVICE_PORT
23              value: '3306'

```

Create RC:

```
# kubectl create -f myweb-rc.yaml
```

Check whether Tomcat RC is created successfully:

```

1 j# kubectl get rc
2 NAME      DESIRED  CURRENT  READY  AGE
3 mysql     1        1        1      1h
4 myweb     2        2        2      1h
5 j# kubectl get po
6 NAME                                READY   STATUS    RESTARTS   AGE
7 mysql-px9th                         1/1     Running   0          1h
8 myweb-fxq9s                         1/1     Running   0          1h
9 myweb-tt5sd                         1/1     Running   0          1h

```

Create the Tomcat service file tomcat-svc.yaml as follows:

```

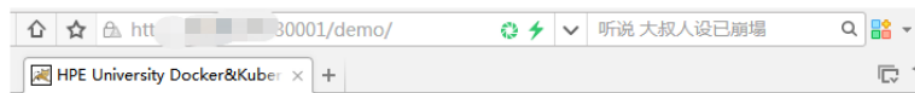
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: myweb
5 spec:
6   type: NodePort
7   ports:
8     - port: 8080
9       nodePort: 30001
10  selector:
11    app: myweb

```

Create and view the created service status:

```
1 ]# kubectl create -f tomcat-svc.yaml
2 ]# kubectl get svc
3 NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
4 mysql         ClusterIP     10.107.188.209  <none>           3306/TCP         1h
5 myweb         NodePort      10.109.40.205   <none>           8080:30001/TCP   1h
```

Step 4: access the page through the browser



Congratulations!!

Add...

Name	Level(Score)
google	100
docker	100
teacher	100
HPE	100
our team	100
me	100

Click add on the page to add a piece of data, refresh the page, and the data is successfully stored. Here, our application service is completed.

FINAL GRADE

50/100

GENERAL COMMENTS

Instructor

A limited attempt at the report made, rather low on the detail expected and a modest amount below the word count. Unfortunately there are also no sources to indicate where your information was derived from. It's imperative that your academic work contains a source list along with inline citations to indicate the origins of your ideas.

If you'd like to discuss this, please don't hesitate to ask - this is the only thing which excluded you from a very high mark in this work.

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
