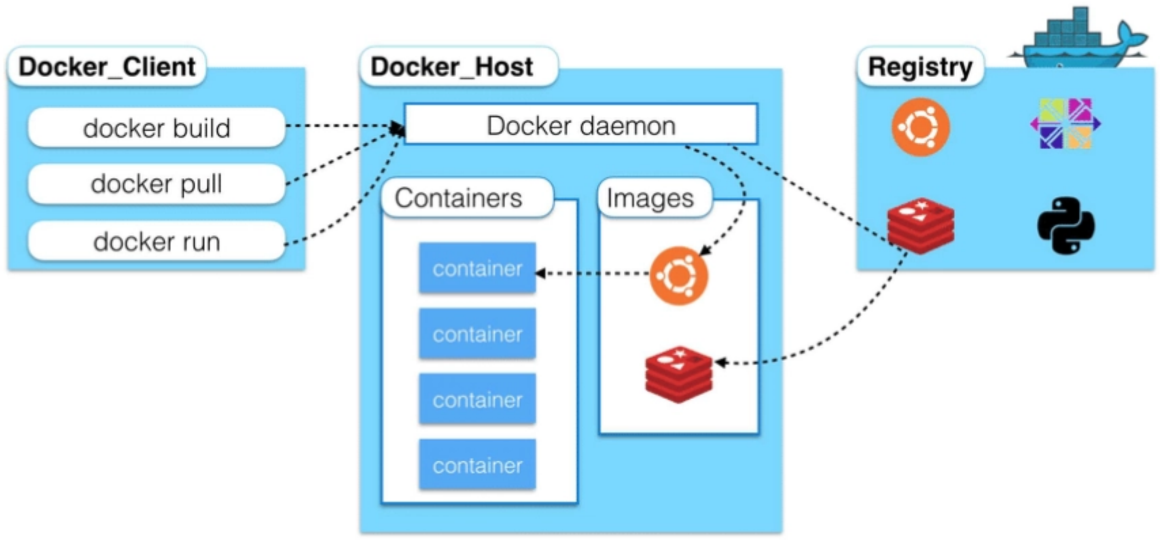
# Docker安装

## Docker的基本组成



**镜像（image）：**

docker镜像就好比是一个模板，可以通过这个模板来创建容器服务，tomcat镜像==>run==>tomcat01容器（提供服务器），通过这个镜像可以创建多个容器（最终服务运行或者项目运行就是在容器中的）。

**容器（container）：**

Docker利用容器技术，独立运行一个或者一个组应用，通过镜像来创建的。

启动、停止、删除、基本命令！

可以理解为一个简易的linux系统。

**仓库（repositoty）：**

仓库就是存放镜像的地方！

仓库分为有共有仓库和私有仓库！

Docker hub(默认是国外的，国内有阿里云，腾讯云等都有容器服务器)。

## 安装Docker

> 环境准备

> 环境查看

```
# 系统内核
youxin@youxin-virtual-machine:/$ uname -r
5.11.0-40-generic
```

```
# 系统版本
youxin@youxin-virtual-machine:/$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.3 LTS (Focal Fossa)"
ID=ubuntu
```

```
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.3 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-
policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

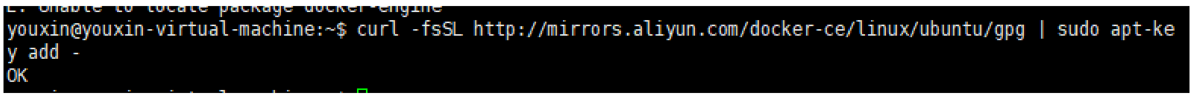
> 安装

帮助文档[https://docs.docker.com/](https://docs.docker.com/)

```
#1、卸载旧的版本
sudo apt-get purge docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin docker-ce-rootless-extras
#主机上的映像、容器、卷或自定义配置文件 不会自动移除。要删除所有映像、容器和卷，请执行以下操作：
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd
#2、更新apt包索引
sudo apt-get update
#3、安装必备的软件包以允许apt通过 HTTPS 使用仓（repository）：
sudo apt-get install ca-certificates curl gnupg lsb-release
#4、添加Docker官方版本库的GPG密钥：
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
#5、使用一下命令设置仓库
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

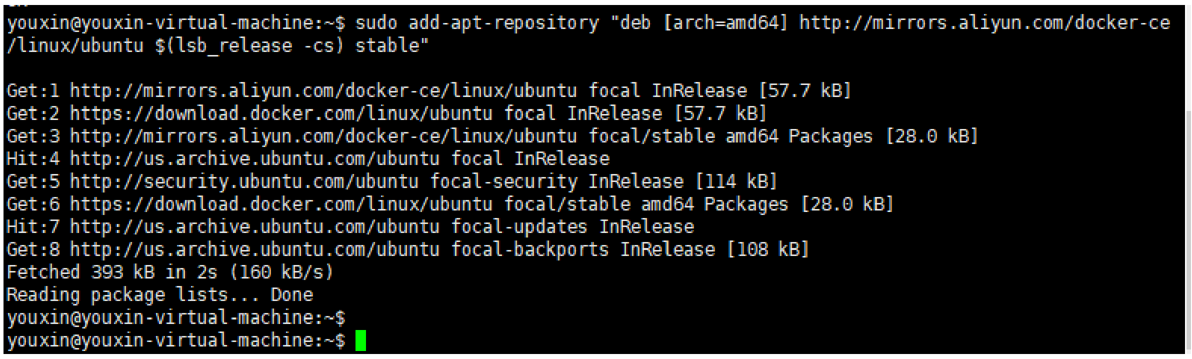
> 切换阿里云镜像源安装docker

```
#1、卸载可能存在的或者为安装成功的Docker版本
sudo apt-get remove docker docker-engine docker-ce docker.io
#2、添加阿里云的GPG密钥
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key
add -
```



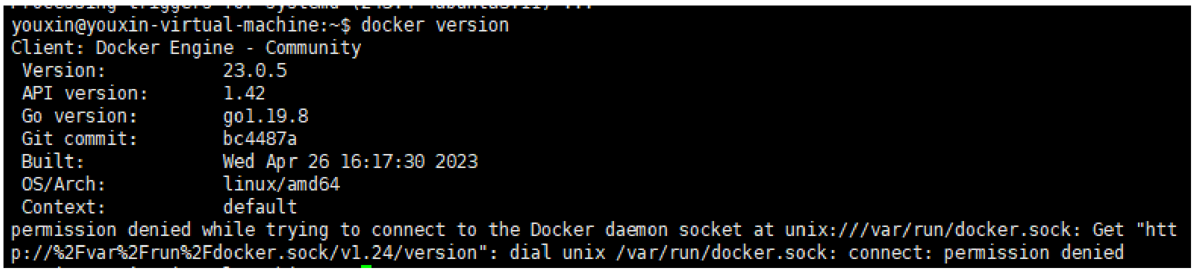
```
E: Unable to locate package docker-engine
youxin@youxin-virtual-machine:~$ curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-ke
y add -
OK
```

```
#3、使用以下命令设置仓库
sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-
ce/linux/ubuntu $(lsb_release -cs) stable"
```



```
youxin@youxin-virtual-machine:~$ sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-ce
/linux/ubuntu $(lsb_release -cs) stable"

Get:1 http://mirrors.aliyun.com/docker-ce/linux/ubuntu focal InRelease [57.7 kB]
Get:2 https://download.docker.com/linux/ubuntu focal InRelease [57.7 kB]
Get:3 http://mirrors.aliyun.com/docker-ce/linux/ubuntu focal/stable amd64 Packages [28.0 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:5 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:6 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [28.0 kB]
Hit:7 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease
Get:8 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Fetched 393 kB in 2s (160 kB/s)
Reading package lists... Done
youxin@youxin-virtual-machine:~$
youxin@youxin-virtual-machine:~$
```

```
#4、安装最新版本的Docker(飞速安装)
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
# 5、验证Docker是否安装成功
#查看docker版本
docker version
```

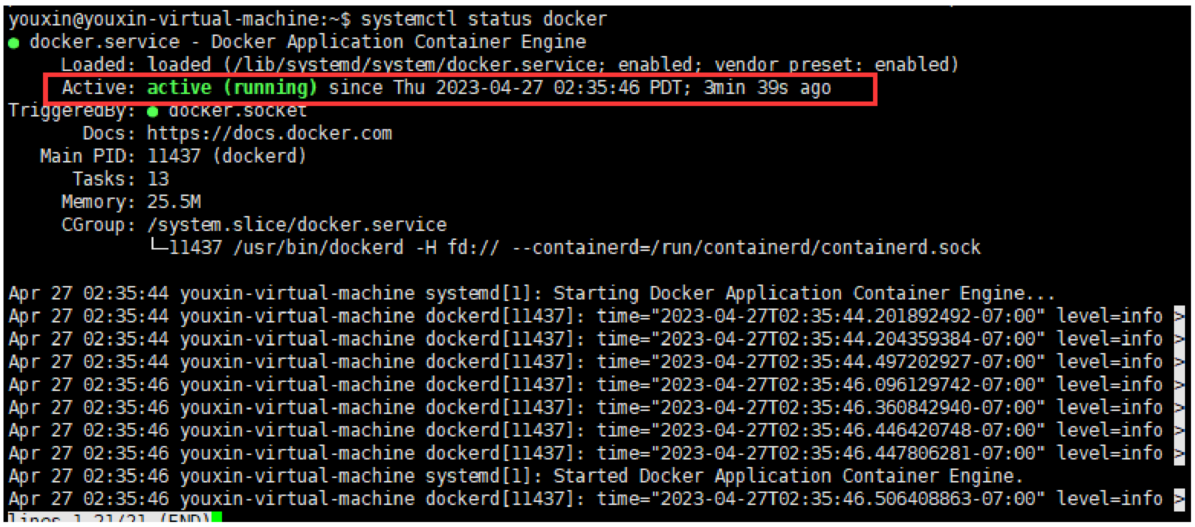


```
youxin@youxin-virtual-machine:~$ docker version
Client: Docker Engine - Community
 Version:           23.0.5
 API version:       1.42
 Go version:        go1.19.8
 Git commit:        bc4487a
 Built:             Wed Apr 26 16:17:30 2023
 OS/Arch:           linux/amd64
 Context:           default
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "htt
p://%2Fvar%2Frun%2Fdocker.sock/v1.24/version": dial unix /var/run/docker.sock: connect: permission denied
```

> 启动docker

1、安装完成后，运行如下命令验证 Docker 服务是否在运行

`systemctl status docker`



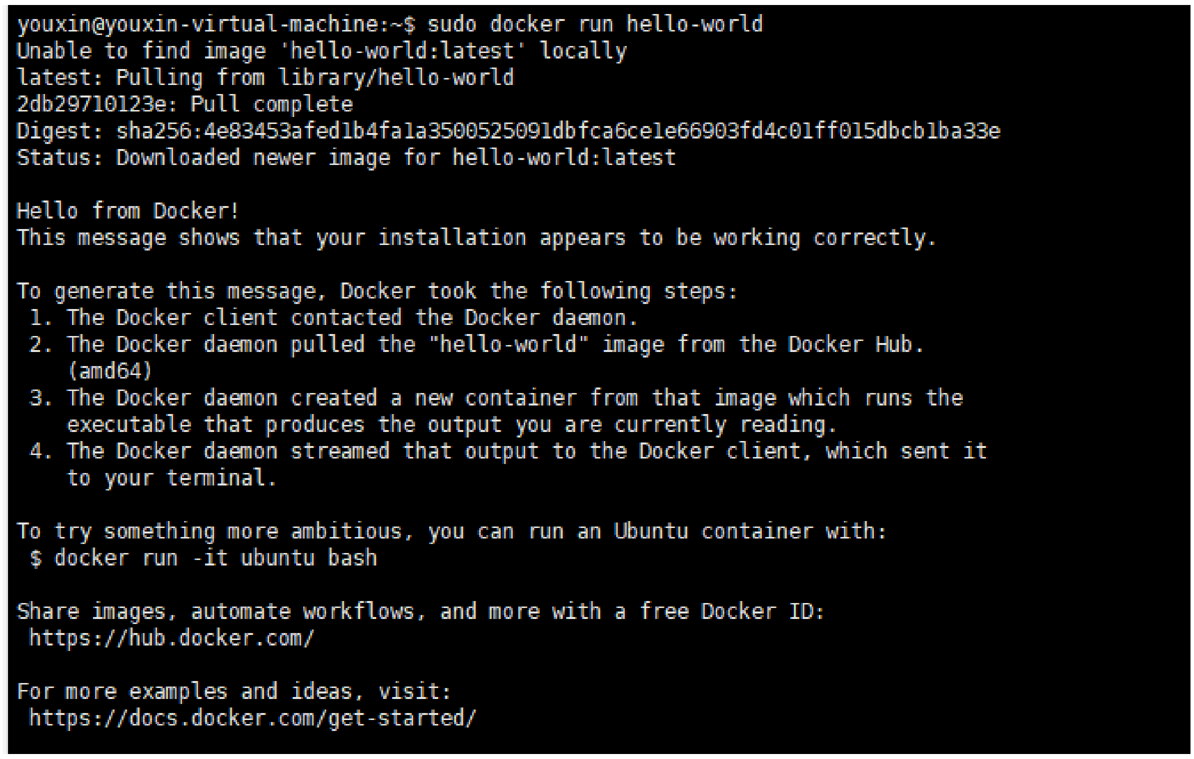
```
youxin@youxin-virtual-machine:~$ systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-04-27 02:35:46 PDT; 3min 39s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 11437 (dockerd)
      Tasks: 13
     Memory: 25.5M
     CGroup: /system.slice/docker.service
             └─11437 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Apr 27 02:35:44 youxin-virtual-machine systemd[1]: Starting Docker Application Container Engine...
Apr 27 02:35:44 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:44.201892492-07:00" level=info >
Apr 27 02:35:44 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:44.204359384-07:00" level=info >
Apr 27 02:35:44 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:44.497202927-07:00" level=info >
Apr 27 02:35:46 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:46.096129742-07:00" level=info >
Apr 27 02:35:46 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:46.360842940-07:00" level=info >
Apr 27 02:35:46 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:46.446420748-07:00" level=info >
Apr 27 02:35:46 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:46.447806281-07:00" level=info >
Apr 27 02:35:46 youxin-virtual-machine systemd[1]: Started Docker Application Container Engine.
Apr 27 02:35:46 youxin-virtual-machine dockerd[11437]: time="2023-04-27T02:35:46.506408863-07:00" level=info >
lines 1-21/21 (END)
```

2、运行docker

`sudo systemctl start docker`

3、运行hallo-world验证Docker是否运行正常

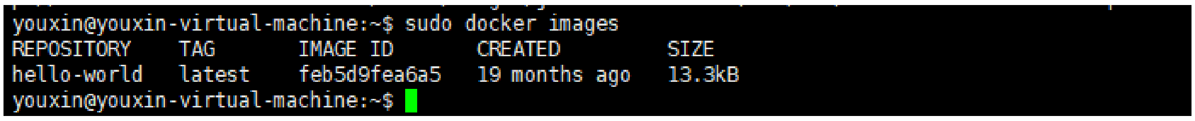
`sudo docker run hello-world`



```
youxin@youxin-virtual-machine:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:4e83453afed1b4fa1a3500525091dbfca6ce1e66903fd4c01ff015dbcb1ba33e
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

4、查看下载的hello-world镜像

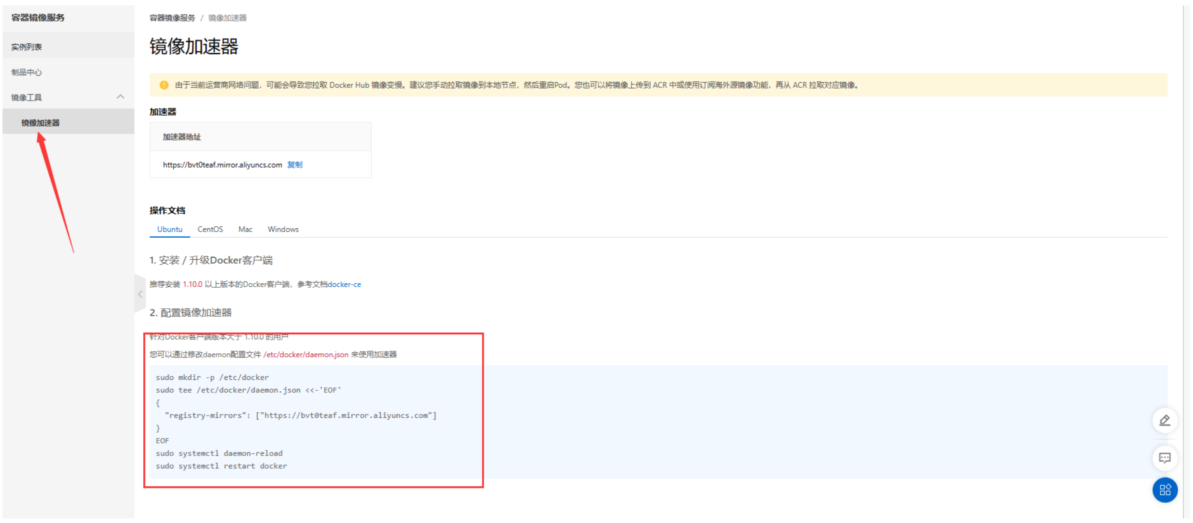
`sudo docker images`



```
youxin@youxin-virtual-machine:~$ sudo docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
hello-world    latest    feb5d9fea6a5   19 months ago    13.3kB
youxin@youxin-virtual-machine:~$ 
```

## 阿里云镜像加速

进入阿里云后台->找到"容器镜像服务"->镜像工具->镜像加速器



镜像加速器

由于当前运营商网络问题，可能会导致您拉取 Docker Hub 镜像变慢。建议您手动拉取镜像到本地节点，然后重启Pod。您也可以将镜像上传到 ACR 中或使用订阅海外源镜像功能，再从 ACR 拉取对应镜像。

加速器

| 加速器地址 |
|---|
| https://bvt0teaf.mirror.aliyuncs.com |

操作文档

1. 安装 / 升级Docker客户端

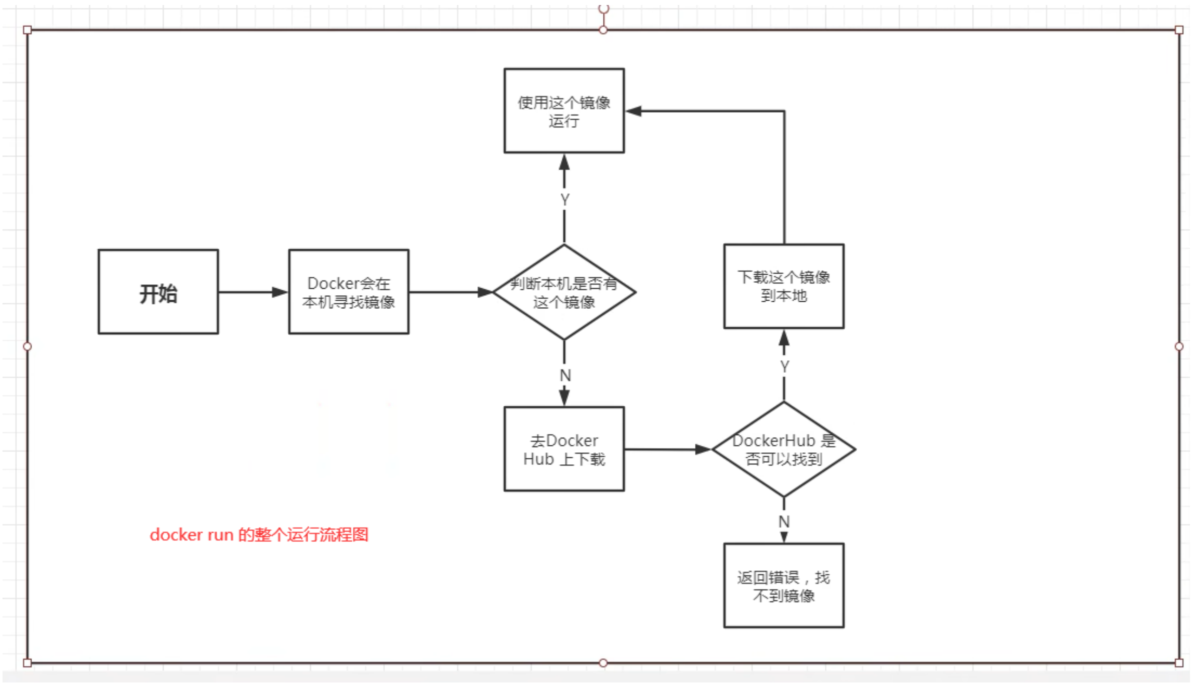
推荐安装 1.10.0 以上版本的Docker客户端，参考文档docker-ce

2. 配置镜像加速器

针对Docker客户端版本大于 1.10.0 的用户

您可以通过修改daemon配置文件 /etc/docker/daemon.json 来使用加速器

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://bvt0teaf.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://bvt0teaf.mirror.aliyuncs.com"]
}
EOF

sudo systemctl daemon-reload

sudo systemctl restart docker
```



## Docker的常用命令https://docs.docker.com/reference/

### 帮助命令

```
docker version #查看docker版本
docker info #显示docker的系统信息，包括镜像和容器的数量
docker 命令 --help #帮助命令
```

### 镜像命令

**sudo docker images** #查看所有本地主机上的镜像

```
root@youxin-virtual-machine:/home/youxin# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
hello-world   latest    feb5d9fea6a5   19 months ago    13.3kB
# 解释
REPOSITORY   镜像的仓库源
TAG   镜像的标签
IMAGE ID   镜像的id
CREATED   镜像的创建时间
SIZE   镜像的大小

-a, --all             展示所有镜像
-q, --quiet           只展示镜像的id
```

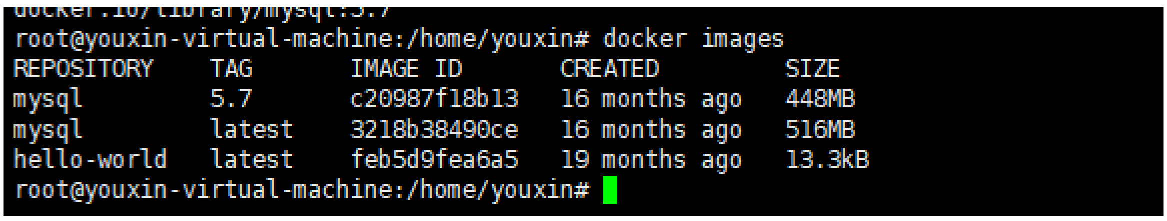
**docker search** 搜索镜像（官方镜像仓库：https://hub.docker.com/）

```
root@youxin-virtual-machine:/home/youxin# docker search mysql
NAME                            DESCRIPTION
STARS     OFFICIAL   AUTOMATED
mysql                           MySQL is a widely used, open-source relation…
14076     [OK]
mariadb                         MariaDB Server is a high performing open sou…
5374      [OK]
percona                         Percona Server is a fork of the MySQL relati…
606       [OK]
phpmyadmin                      phpMyAdmin - A web interface for MySQL and M…
785       [OK]
circleci/mysql                  MySQL is a widely used, open-source relation…
29
bitnami/mysql                   Bitnami MySQL Docker Image
 86                  [OK]
bitnami/mysqld-exporter
5
ubuntu/mysql                    MySQL open source fast, stable, multi-thread…
46


#可选项，通过收藏来过滤
-filter=STARS=3000 #搜索出来的镜像是STARS大于3000的镜像
```

**docker pull** 拉取镜像

```
#下载镜像 docker pull 镜像名[:tag]
root@youxin-virtual-machine:/home/youxin# docker pull mysql
Using default tag: latest #如果不写tag，默认就是latest
latest: Pulling from library/mysql
72a69066d2fe: Pull complete  #分层下载，docker image的核心 联合文件系统
93619dbc5b36: Pull complete
99da31dd6142: Pull complete
626033c43d70: Pull complete
37d5d7efb64e: Pull complete
ac563158d721: Pull complete
d2ba16033dad: Pull complete
688ba7d5c01a: Pull complete
00e060b6d11d: Pull complete
1c04857f594f: Pull complete
4d7cfa90e6ea: Pull complete
e0431212d27d: Pull complete
Digest: sha256:e9027fe4d91c0153429607251656806cc784e914937271037f7738bd5b8e7709
#签名
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:lates #真实地址

docker pull mysql
#等价于
docker pull docker.io/library/mysql:lates

#指定版本下载
root@youxin-virtual-machine:/home/youxin# docker pull mysql:5.7
5.7: Pulling from library/mysql
72a69066d2fe: Already exists
```

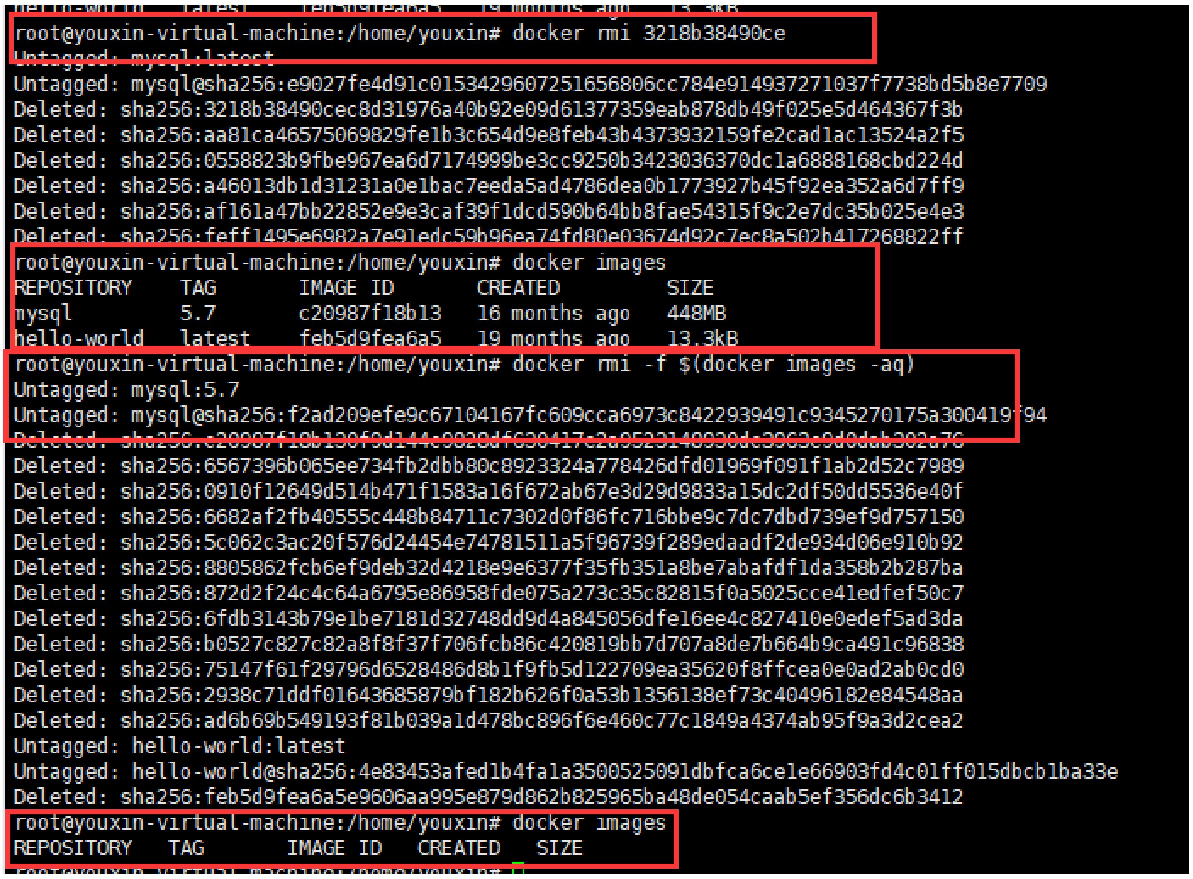
```
93619dbc5b36: Already exists
99da31dd6142: Already exists
626033c43d70: Already exists
37d5d7efb64e: Already exists
ac563158d721: Already exists
d2ba16033dad: Already exists
0ceb82207cd7: Pull complete
37f2405cae96: Pull complete
e2482e017e53: Pull complete
70deed891d42: Pull complete
Digest: sha256:f2ad209efe9c67104167fc609cca6973c8422939491c9345270175a300419f94
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```



```
root@youxin-virtual-machine:/home/youxin# docker images
REPOSITORY     TAG       IMAGE ID       CREATED         SIZE
mysql          5.7       c20987f18b13   16 months ago   448MB
mysql          latest    3218b38490ce   16 months ago   516MB
hello-world    latest    feb5d9fea6a5   19 months ago   13.3kB
root@youxin-virtual-machine:/home/youxin#
```

**docker rmi** 删除镜像

```
root@youxin-virtual-machine:/home/youxin# docker rmi -f 3218b38490ce #删除镜像id为3218b38490ce的镜像
Untagged: mysql:latest
Untagged:
mysql@sha256:e9027fe4d91c0153429607251656806cc784e914937271037f7738bd5b8e7709
Deleted: sha256:3218b38490cec8d31976a40b92e09d61377359eab878db49f025e5d464367f3b
Deleted: sha256:aa81ca46575069829fe1b3c654d9e8feb43b4373932159fe2cad1ac13524a2f5
Deleted: sha256:0558823b9fbe967ea6d7174999be3cc9250b3423036370dc1a6888168cbd224d
Deleted: sha256:a46013db1d31231a0e1bac7eeda5ad4786dea0b1773927b45f92ea352a6d7ff9
Deleted: sha256:af161a47bb22852e9e3caf39f1dcd590b64bb8fae54315f9c2e7dc35b025e4e3
Deleted: sha256:feff1495e6982a7e91edc59b96ea74fd80e03674d92c7ec8a502b417268822ff

==========================
root@youxin-virtual-machine:/home/youxin# docker rmi -f 镜像id 镜像id 镜像id #删除多个镜像
root@youxin-virtual-machine:/home/youxin# docker rmi -f $(docker images -aq) #删除所有镜像
```



```
hello-world   latest    feb5d9fea6a5   19 months ago   13.3kB
root@youxin-virtual-machine:/home/youxin# docker rmi 3218b38490ce
Untagged: mysql:latest
Untagged: mysql@sha256:e9027fe4d91c0153429607251656806cc784e914937271037f7738bd5b8e7709
Deleted: sha256:3218b38490cec8d31976a40b92e09d61377359eab878db49f025e5d464367f3b
Deleted: sha256:aa81ca46575069829fe1b3c654d9e8feb43b4373932159fe2cad1ac13524a2f5
Deleted: sha256:0558823b9fbe967ea6d7174999be3cc9250b3423036370dc1a6888168cbd224d
Deleted: sha256:a46013db1d31231a0e1bac7eeda5ad4786dea0b1773927b45f92ea352a6d7ff9
Deleted: sha256:af161a47bb22852e9e3caf39f1dcd590b64bb8fae54315f9c2e7dc35b025e4e3
Deleted: sha256:feff1495e6982a7e91edc59b96ea74fd80e03674d92c7ec8a502b417268822ff
root@youxin-virtual-machine:/home/youxin# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
mysql         5.7       c20987f18b13   16 months ago   448MB
hello-world   latest    feb5d9fea6a5   19 months ago   13.3kB
root@youxin-virtual-machine:/home/youxin# docker rmi -f $(docker images -aq)
Untagged: mysql:5.7
Untagged: mysql@sha256:f2ad209efe9c67104167fc609cca6973c8422939491c9345270175a300419f94
Deleted: sha256:[illegible]
Deleted: sha256:6567396b065ee734fb2dbb80c8923324a778426dfd01969f091f1ab2d52c7989
Deleted: sha256:0910f12649d514b471f1583a16f672ab67e3d29d9833a15dc2df50dd5536e40f
Deleted: sha256:6682af2fb40555c448b84711c7302d0f86fc716bbe9c7dc7dbd739ef9d757150
Deleted: sha256:5c062c3ac20f576d24454e74781511a5f96739f289edaadf2de934d06e910b92
Deleted: sha256:8805862fcb6ef9deb32d4218e9e6377f35fb351a8be7abafdf1da358b2b287ba
Deleted: sha256:872d2f24c4c64a6795e86958fde075a273c35c82815f0a5025cce41edfef50c7
Deleted: sha256:6fdb3143b79e1be7181d32748dd9d4a845056dfe16ee4c827410e0edef5ad3da
Deleted: sha256:b0527c827c82a8f8f37f706fcb86c420819bb7d707a8de7b664b9ca491c96838
Deleted: sha256:75147f61f29796d6528486d8b1f9fb5d122709ea35620f8ffcea0e0ad2ab0cd0
Deleted: sha256:2938c71ddf01643685879bf182b626f0a53b1356138ef73c40496182e84548aa
Deleted: sha256:ad6b69b549193f81b039a1d478bc896f6e460c77c1849a4374ab95f9a3d2cea2
Untagged: hello-world:latest
Untagged: hello-world@sha256:4e83453afed1b4fa1a3500525091dbfca6ce1e66903fd4c01ff015dbcb1ba33e
Deleted: sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412
root@youxin-virtual-machine:/home/youxin# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
root@youxin-virtual-machine:/home/youxin#
```

## 容器命令

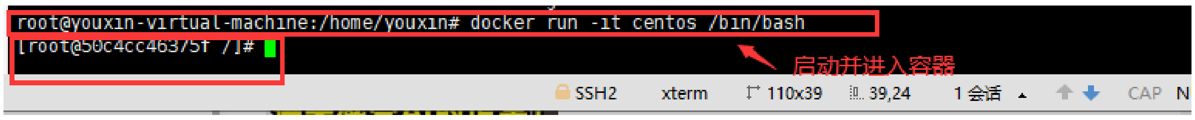
**说明：有了镜像才可以创建容器**

例：下载centos镜像测试学习：

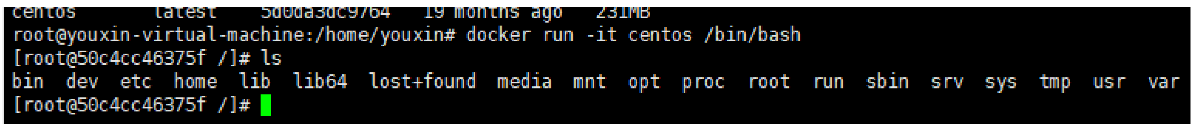
```
docker pull centos
```

**新建容器并启动**

```
docker run [可选参数] image

# 参数说明
--name="name"    #容器名字  tomcat01 tomcat02,用来区分容器
-d               #后台方式运行
-it              #使用交互方式运行，进入容器查看内容
-p               #指定容器的端口 -p 8080:8080
    -p ip:主机端口:容器端口
    -p 主机端口:容器端口
    -p 容器端口
    容器端口
-P               #随机指定端口
```



```
root@youxin-virtual-machine:/home/youxin# docker run -it centos /bin/bash
[root@50c4cc46375f /]#
```



```
centos       latest    5d0da3dc9764   19 months ago   231MB
root@youxin-virtual-machine:/home/youxin# docker run -it centos /bin/bash
[root@50c4cc46375f /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@50c4cc46375f /]#
```

`exit` 退出镜像，返回主机

### 列出所有运行中的容器

```
#docker ps 命令
-a  #列出当前正在运行的容器+带出历史运行过的容器
-n=? #显示最近创建的容器，如-n=1 ，显示最近的一个容器
-q  #只显示容器的编号

root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@youxin-virtual-machine:/home/youxin# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED          STATUS
           PORTS      NAMES
50c4cc46375f   centos         "/bin/bash"   3 minutes ago    Exited (0) About a
minute ago             sad_ganguly
d9b1f9f2b460   feb5d9fea6a5   "/hello"      3 days ago       Exited (0) 3 days
ago                     keen_sammet
2113455f53ba   feb5d9fea6a5   "/hello"      3 days ago       Created
                      optimistic_hellman
775de7f961ad   feb5d9fea6a5   "/hello"      3 days ago       Exited (0) 3 days
ago                     gracious_dhawan
```

### 退出容器

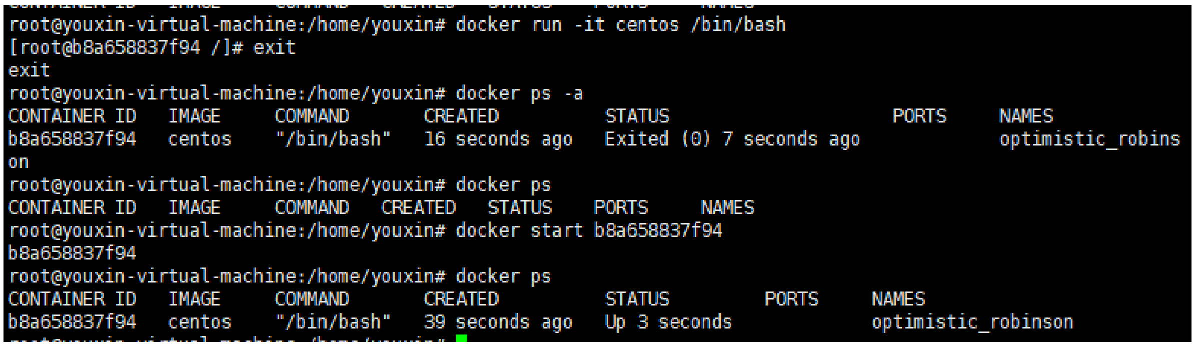
```
exit   #直接容器退出并停止
ctrl + p + q   #容器退出但不停止
```

### 删除容器

```
docker rm 容器id                #删除指定的容器，不能删除正在运行的镜像，如果想要强制删除
rm -f
docker rm -f $(docker ps -aq)  #删除所有的容器,以容器id为参数递归删除
docker ps -a -q|xargs docker rm #删除所有容器，以管道符的形式
```

### 启动和停止容器的操作

```
docker start 容器id     #启动容器
docker restart 容器Id   #重启容器
docker stop 容器id      #停止当前正在运行的容器
docker kill 容器id      #强制停止当前容器
```



```
root@youxin-virtual-machine:/home/youxin# docker run -it centos /bin/bash
[root@b8a658837f94 /]# exit
exit
root@youxin-virtual-machine:/home/youxin# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                     PORTS     NAMES
b8a658837f94   centos    "/bin/bash"   16 seconds ago   Exited (0) 7 seconds ago             optimistic_robins
on
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@youxin-virtual-machine:/home/youxin# docker start b8a658837f94
b8a658837f94
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS         PORTS     NAMES
b8a658837f94   centos    "/bin/bash"   39 seconds ago   Up 3 seconds             optimistic_robinson
```

## 其他常用命令

### 后台启动容器

```
#命令 docker run -d 镜像名!
docker run -d centos

#问题docker ps 发现centos 停止了
#常见的问题，docker 容器使用后台运行，就必须要有一个前台进程，docker发现没有应用，就会自动停止
#nginx,容器启动后，发现自己没有提供服务，就会立刻停止，就是没有程序了
```

### 查看日志

```
docker logs + [参数] + 容器id
--details        Show extra details provided to logs
  -f, --follow         Follow log output
      --since string   Show logs since timestamp (e.g. "2013-01-02T13:23:37Z")
or relative (e.g. "42m"
                       for 42 minutes)
  -n, --tail string    Number of lines to show from the end of the logs (default
"all")
  -t, --timestamps     Show timestamps
      --until string   Show logs before a timestamp (e.g. "2013-01-
02T13:23:37Z") or relative (e.g.
                       "42m" for 42 minutes)



root@youxin-virtual-machine:/home/youxin# docker run -it centos /bin/bash
[root@3a5813680bc5 /]# root@youxin-virtual-machine:/home/youxin# docker logs -f -
t b8a658837f94
2023-04-30T13:42:37.445670139Z [root@b8a658837f94 /]# exit
2023-04-30T13:42:37.445709032Z exit
2023-04-30T13:44:56.364639382Z [root@b8a658837f94 /]# exit


##测试
root@youxin-virtual-machine:/home/youxin# docker run -d centos /bin/sh -c "while
true;do echo youxin;sleep 1;done" #以后台方式运行并执行shell脚本
609ca0b79b91cdc49914753156c8ff17a260369bfaa59c75edec4d037e6261fa
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS     NAMES
609ca0b79b91   centos    "/bin/sh -c 'while t…"   6 seconds ago   Up 3 seconds
          kind_galois
root@youxin-virtual-machine:/home/youxin# docker logs -f -t --tail 10
609ca0b79b91
2023-04-30T14:00:04.184326323Z youxin
2023-04-30T14:00:05.200081871Z youxin
2023-04-30T14:00:06.219946144Z youxin
2023-04-30T14:00:07.228095361Z youxin
2023-04-30T14:00:08.240143363Z youxin
2023-04-30T14:00:09.252176238Z youxin
2023-04-30T14:00:10.268486332Z youxin
2023-04-30T14:00:11.284112126Z youxin
2023-04-30T14:00:12.304059387Z youxin
2023-04-30T14:00:13.320040143Z youxin
```

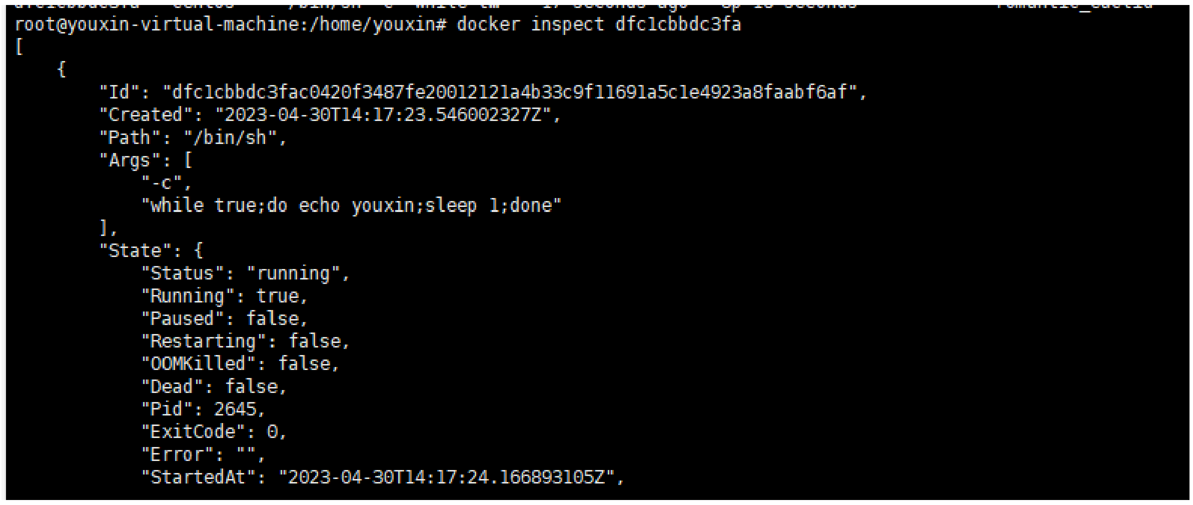
```
2023-04-30T14:00:14.336076601Z youxin
2023-04-30T14:00:15.348368697Z youxin
2023-04-30T14:00:16.368107622Z youxin
2023-04-30T14:00:17.383943908Z youxin
2023-04-30T14:00:18.403979126Z youxin
^C
root@youxin-virtual-machine:/home/youxin# docker stop 609ca0b79b91
609ca0b79b91
```

**查看容器中的进程信息**

```
#命令 docker top 容器id
root@youxin-virtual-machine:/home/youxin# docker top fc909e5e5624
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                4469                4445                3
07:06               ?                   00:00:01
root                4555                4469                22
 07:06               ?                   00:00:00

```

**查看镜像的元数据**

```
#命令 docker inspect [参数] 容器Id
#测试
```



```
root@youxin-virtual-machine:/home/youxin# docker inspect dfc1cbbdc3fa
[
    {
        "Id": "dfc1cbbdc3fac0420f3487fe20012121a4b33c9f11691a5c1e4923a8faabf6af",
        "Created": "2023-04-30T14:17:23.546002327Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "while true;do echo youxin;sleep 1;done"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 2645,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2023-04-30T14:17:24.166893105Z",
```

**进入当前正在运行的容器**

```
#通常容器都是使用后台方式运行的，需要进入容器，修改一些配置

#命令 docker exec -it 容器id bashshell

#测试
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS     NAMES
dfc1cbbdc3fa   centos    "/bin/sh -c 'while t…"   2 minutes ago   Up 2 minutes
          romantic_euclid
root@youxin-virtual-machine:/home/youxin# docker exec -it dfc1cbbdc3fa /bin/bash
[root@dfc1cbbdc3fa /]# ls
```

```
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
[root@dfc1cbbdc3fa /]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 14:17 ?        00:00:00 /bin/sh -c while true;do
echo youxin;sleep 1;done
root         214       0  0 14:20 pts/0    00:00:00 /bin/bash
root         253       1  0 14:21 ?        00:00:00 /usr/bin/coreutils --
coreutils-prog-shebang=sleep /usr/bin
root         254     214  0 14:21 pts/0    00:00:00 ps -ef


#方式二
docker attach 容器id

#测试，进入后是正在执行的代码

#docker exec  #进入容器后开启一个新的终端，可以在里面操作（常用）
#docker attach #进入容器正在执行的终端，不会启动新的进程！
```

**从容器内拷贝文件到主机上**

```
#命令 docker cp 容器id:容器内路径  目的主机路径

#测试
root@youxin-virtual-machine:/home# docker exec -it dfc1cbbdc3fa /bin/bash 进入容器
[root@dfc1cbbdc3fa /]# cd /home
[root@dfc1cbbdc3fa home]# ls
[root@dfc1cbbdc3fa home]# touch test.txt #在容器/home目录下创建test.txt文件
[root@dfc1cbbdc3fa home]# exit
exit
root@youxin-virtual-machine:/home# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS     NAMES
dfc1cbbdc3fa   centos    "/bin/sh -c 'while t…"   14 minutes ago   Up 14 minutes
            romantic_euclid
root@youxin-virtual-machine:/home# docker cp dfc1cbbdc3fa:/home/test.txt /home #
复制容器内文件到主机上
Successfully copied 1.54kB to /home
root@youxin-virtual-machine:/home# ls #复制成功
hello  test.txt  youxin  youxin01  youxin.txt

#拷贝是一个手动过程，使用-v卷的技术，可以实现自动脚本
```

## 小结

```
attach    Attach to a running container  #当前shell下attach连接指定运行镜像
build     Build an image from a Dockerfile  #通过Dockerfile定制镜像
commit    Create a new image from a container's changes  #提交当前容器为新的镜像
cp    Copy files/folders from a container to a HOSTDIR or to STDOUT  #从容器中拷贝
指定文件或者目录到宿主机中
create    Create a new container  #创建一个新的容器，同run 但不启动容器
diff    Inspect changes on a container's filesystem  #查看docker容器变化
events    Get real time events from the server#从docker服务获取容器实时事件
exec    Run a command in a running container#在已存在的容器上运行命令
```

```
export    Export a container's filesystem as a tar archive  #导出容器的内容流作为一个tar归档文件(对应import)
history    Show the history of an image  #展示一个镜像形成历史
images    List images  #列出系统当前镜像
import    Import the contents from a tarball to create a filesystem image  #从tar包中的内容创建一个新的文件系统映像(对应export)
info    Display system-wide information  #显示系统相关信息
inspect    Return low-level information on a container or image  #查看容器详细信息
kill    Kill a running container  #kill指定docker容器
load    Load an image from a tar archive or STDIN  #从一个tar包中加载一个镜像(对应save)
login    Register or log in to a Docker registry#注册或者登陆一个docker源服务器
logout    Log out from a Docker registry  #从当前Docker registry退出
logs    Fetch the logs of a container  #输出当前容器日志信息
pause    Pause all processes within a container#暂停容器
port    List port mappings or a specific mapping for the CONTAINER  #查看映射端口对应的容器内部源端口
ps    List containers  #列出容器列表
pull    Pull an image or a repository from a registry  #从docker镜像源服务器拉取指定镜像或者库镜像
push    Push an image or a repository to a registry  #推送指定镜像或者库镜像至docker源服务器
rename    Rename a container  #重命名容器
restart    Restart a running container  #重启运行的容器
rm    Remove one or more containers  #移除一个或者多个容器
rmi    Remove one or more images  #移除一个或多个镜像(无容器使用该镜像才可以删除，否则需要删除相关容器才可以继续或者-f强制删除)
run    Run a command in a new container  #创建一个新的容器并运行一个命令
save    Save an image(s) to a tar archive#保存一个镜像为一个tar包(对应load)
search    Search the Docker Hub for images  #在docker
hub中搜索镜像
start    Start one or more stopped containers#启动容器
stats    Display a live stream of container(s) resource usage statistics  #统计容器使用资源
stop    Stop a running container  #停止容器
tag         Tag an image into a repository  #给源中镜像打标签
top        Display the running processes of a container #查看容器中运行的进程信息
unpause    Unpause all processes within a container  #取消暂停容器
version    Show the Docker version information#查看容器版本号
wait         Block until a container stops, then print its exit code  #截取容器停止时的退出状态值
```

### nginx部署测试

```
root@youxin-virtual-machine:/home/youxin# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2abf6c4d29d: Pull complete
a9edb18cadd1: Pull complete
589b7251471a: Pull complete
186b1aaa4aa6: Pull complete
b4df32aa5a72: Pull complete
a0bcbecc962e: Pull complete
Digest: sha256:0d17b565c37bcbd895e9d92315a05c1c3c9a29f762b011a10c54a66cd53c9b31
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
root@youxin-virtual-machine:/home/youxin# docker images
```

```
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
nginx        latest    605c77e624dd   16 months ago   141MB
centos       latest    5d0da3dc9764   19 months ago   231MB
root@youxin-virtual-machine:/home/youxin# docker run -d --name nginx01 -p
8081:80 nginx
e4eddc4047ec2523b618249f45d09e72a39e5151566f4c376653af1c13e3b101
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS                                   NAMES
e4eddc4047ec   nginx     "/docker-entrypoint.…"   6 seconds ago   Up 3 seconds
0.0.0.0:8081->80/tcp, :::8081->80/tcp   nginx01
root@youxin-virtual-machine:/home/youxin# curl localhost:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
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>
```

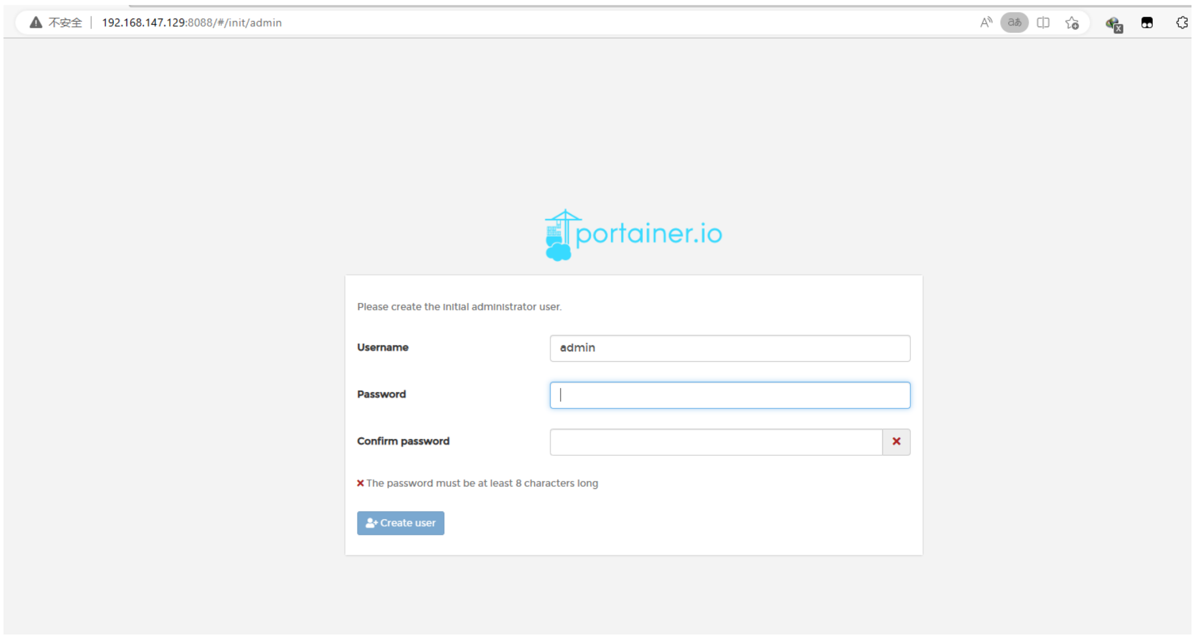
**问题**

```
#在安装例如tomcat时，发现1、linux命令少了 2、没有webapps
#此原因是阿里云镜像的原因，默认是最小的镜像，所有不必要的文件都会被剔除掉，保证最小可运行的环境
#解决：进入tomcat容器，在webapps.dist 目录下存放着root文件
#使用命令cp -r webapps.dist/* webapps将webapps.dist下的文件全部复制即可访问

#命令 docker stats 查看cpu使用情况
```

## 可视化（portainer）

```
docker run -d -p 8088:9000 --restart=always -v
/var/run/docker.sock:/var/run/docker.sock --privileged=true portainer/portainer
```

外网通过8088端口进行访问：


不安全 | 192.168.147.129:8088/#/init/admin

Please create the initial administrator user.

Username: admin

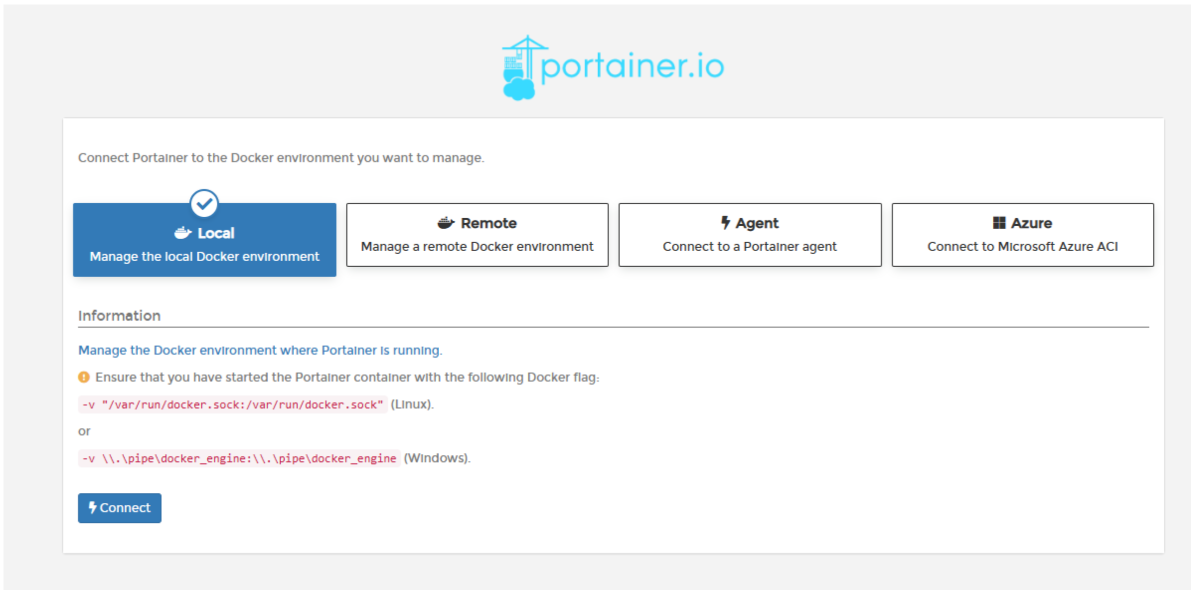
Password:

Confirm password:

✗ The password must be at least 8 characters long

Create user

创建用户->选择本地仓库


Connect Portainer to the Docker environment you want to manage.

Local
Manage the local Docker environment
Remote
Manage a remote Docker environment
Agent
Connect to a Portainer agent
Azure
Connect to Microsoft Azure ACI

Information

Manage the Docker environment where Portainer is running.

Ensure that you have started the Portainer container with the following Docker flag:

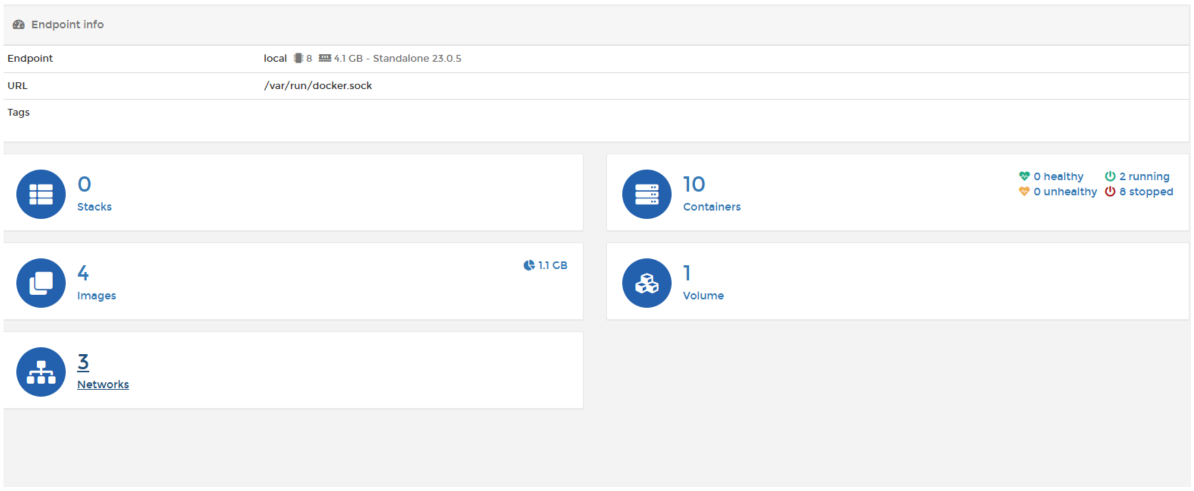
`-v "/var/run/docker.sock:/var/run/docker.sock"` (Linux).

or

`-v \\.\pipe\docker_engine:\\.\pipe\docker_engine` (Windows).

Connect

进入可视化面板：


Endpoint info

| | |
|---|---|
| Endpoint | local 8 4.1 GB - Standalone 23.0.5 |
| URL | /var/run/docker.sock |
| Tags | |

0 Stacks

10 Containers — 0 healthy, 0 unhealthy, 2 running, 8 stopped

4 Images — 1.1 GB

1 Volume

3 Networks

# Docker镜像

## 镜像是什么

镜像是一种轻量级，可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码、运行时库、环境变量和配置文件。

如何得到镜像：

- 从远程仓库下载
- 拷贝
- 自己制作镜像

## Docker镜像加载原理

> UnionFS(联合文件系统)

UnionFS(联合文件系统）：Union文件系统(UnionFS )是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtualfilesystem)。Union文件系统是Docker镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性:一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录

> Docker镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统叫UnionFS(联合文件系统）。（由于docker镜像是由一层一层的文件组成，所以docker镜像之间可以互相共用彼此对自己有用的文件，极大的节省了内存空间，便如若下载Tomcat镜像和MySQL镜像，两个镜像都用到Linux的内核，但下载其中一个镜像时，下载完Linux内核后，再下载另一个镜像，这时就不会再重新下载Linux的内核）bootfs(boot file system)主要包含bootloader（加载器）和kernel（内核）, bootloader（加载器）主要是引导加载kernel（内核）, Linux刚启动时会加载bootfs文件系统，Docker镜像的最底层是bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含bootloader（加载器）和kernel（内核）。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs（平时开机即启动系统时需要用到bootloader（加载器），开机的速度非常的久，但Docker的容器的内核用的是主机的内核，主机已经启动内核，容器不需要再用到bootloader（加载器）来启动内核，所以Docker容器的启动速度非常的快）rootfs (root file system)，在bootfs之上。包含的就是典型Linux系统中的/dev, /proc, /bin, /etc等标准目录和文件，简单来说rootfs就是各种不同的操作系统发行版，比如Ubuntu , Centos等等。（rootfs可以很小，只需包含最基本的命令、工具和程序即可，而内核直接使用主机的kernel（内核）即可，自己只需要提供rootfs即可，所以传统安装操作系统镜像需要几个G的大小，而Docker用来安装容器的操作系统镜像只需要几百MB即可；而且对于不同的Linux发行版，它们的内核都是Linux的内核，所以bootfs基本保持一致，rootfs会有差别，因此不同的Linxu发行版可以公用bootfs）

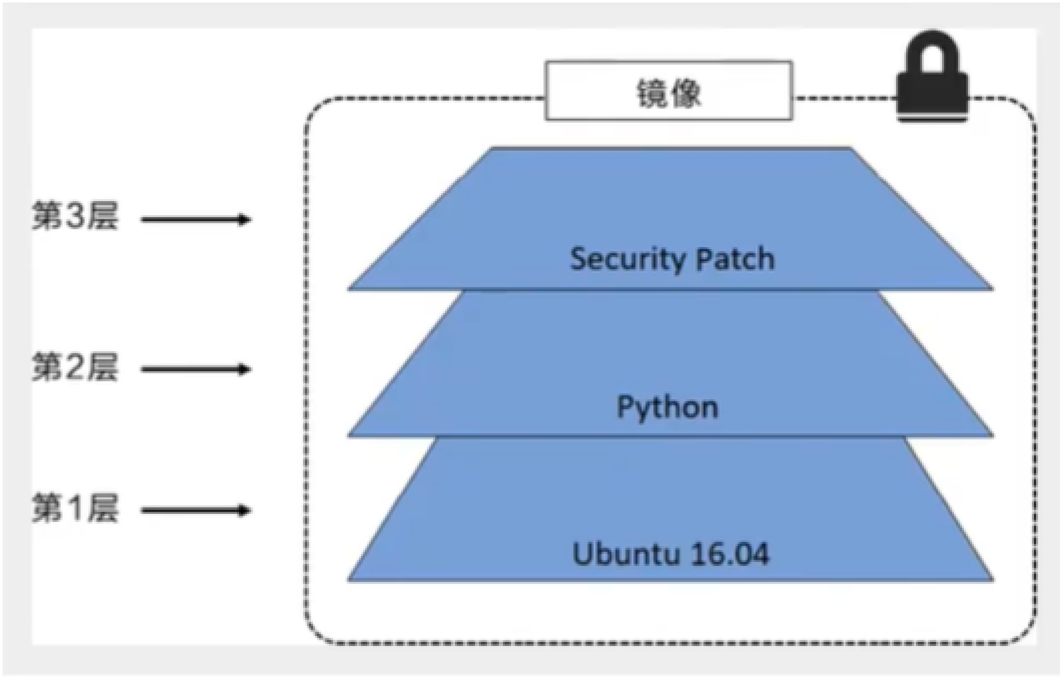
## 分层理解

所有的Docker镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。分层时有文件更新直接替换，基础镜像一样时直接拿过来复用。

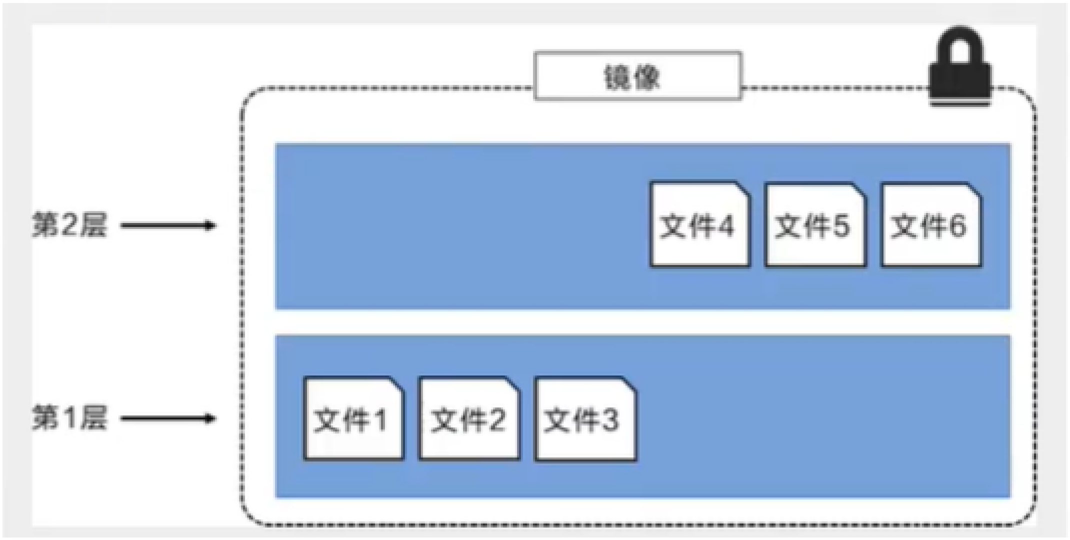
如redis下载时，第一层相同，直接复用，其他几层分层下载。

```
afb6ec6fdc1c: Already exists
608641ee4c3f: Pull complete
668ab9e1f4bc: Pull complete
78a12698914e: Pull complete
d056855f4300: Pull complete
618fdf7d0dec: Pull complete
```

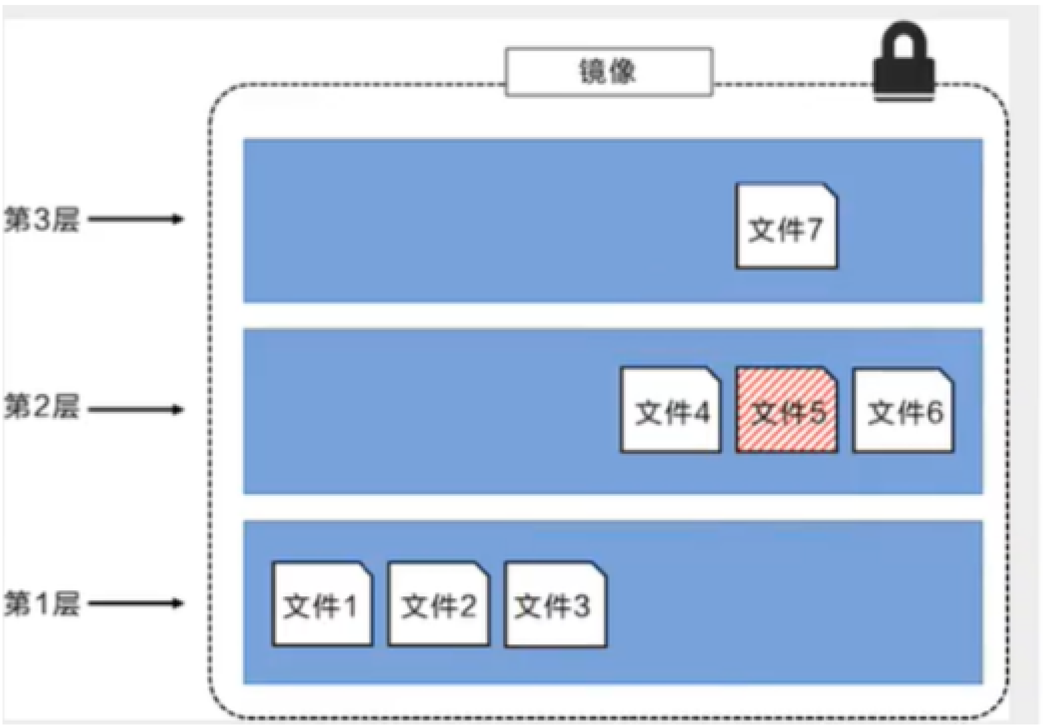
举一个简单的例子，假如基于 Ubuntu16.04创建一个新的镜像，这就是新镜像的第一层。如果在该镜像中添加Python包，就会在基础镜像层之上创建第二个镜像层。如果继续添加一个安全补丁，就会创健第三个镜像层该像当前已经包含3个镜像层，如下图所示（这只是一个用于演示的很简单的例子）。



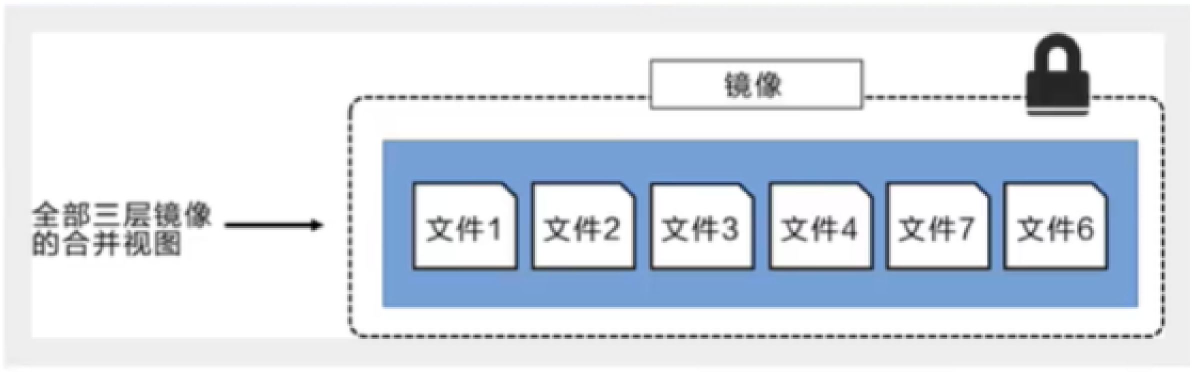
在添加额外的镜像层的同时，镜像始终保持是**当前所有镜像的组合**。一个简单的例子，每个镜像层包含3个文件，而镜像包含了来自两个镜像层的6个文件。



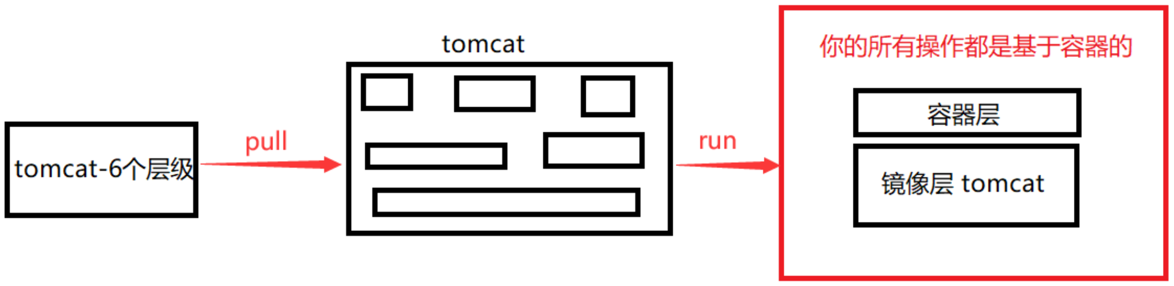
下图中展示了一个稍微复杂的三层镜像，在外部看来整个镜像只有6个文件，这是因为最上层中的文件7是文件5的一个更新版。
上层镜像层中的文件覆盖了底层镜像层中的文件。这样就使得文件的更新版本作为一个新镜像层添加到镜像当中。



所有镜像层堆并合并，对外提供统一的视图。



Docker镜像都是只读的，当容器启动时，一个新的可写层加载到镜像的顶部。这一层就是我们通常说的**容器层**，容器之下的都叫镜像层。



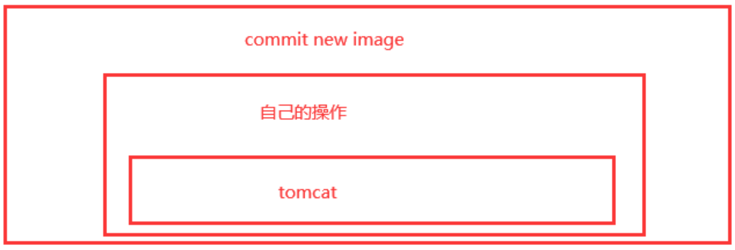
## commit镜像

```
#命令 docker commit 提交容器成为一个新的副本
docker commit -m="提交的描述信息" -a="作者" 容器id 目标镜像名：[TAG] #命令和git相似
```

测试

```
#启动一个默认的tomcat

#发现默认的tomcat没有webapps应用

#拷贝webapps.dist/* 到webapps中

```

```
#提交自己的镜像
root@youxin-virtual-machine:/home/youxin# docker commit -a="youxin" -m="add
webapps applications" 00a2e99d89b3 youxintomcat:1.0
sha256:9cfeb6321ea8937e5a64ac6ade45e43f333923c127530de976e5ca5f71289f4c
root@youxin-virtual-machine:/home/youxin# docker images #查看镜像
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
youxintomcat         1.0       9cfeb6321ea8   6 seconds ago    685MB
nginx                latest    605c77e624dd   16 months ago    141MB
tomcat               9.0       b8e65a4d736d   16 months ago    680MB
centos               latest    5d0da3dc9764   19 months ago    231MB
portainer/portainer  latest    580c0e4e98b0   2 years ago      79.1MB
```



## 容器数据卷

### 什么是容器数据卷

如果数据都在容器中，那么容器一被删除，数据就会丢失！此时需要数据可以持久化，例如mysql的数据可以存储在本地或者其他地方。

容器之间可以有一个数据共享的技术！Docker容器中产生的数据同步到本地。数据就不会丢失-->卷技术，即目录的挂载，将容器内的目录，挂载到Linux上面。

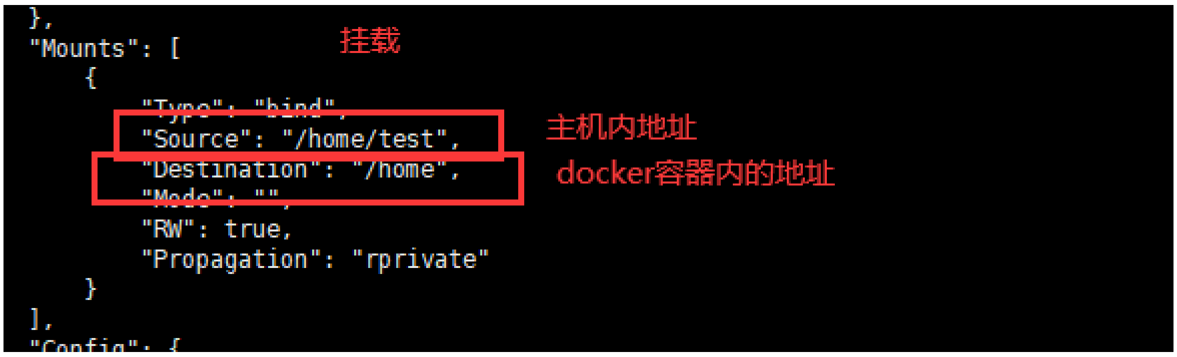
**总结：卷的作用是容器的持久化和同步操作，容器之间也是可以数据共享的。**

### 使用数据卷

> 方式一：直接使用命令来挂载 -v ，参考-p

```
docker run -it -v 主机目录:容器内目录 -p 主机端口:容器内端口

#测试
docker run -it -v /home/test:/home centos /bin/bash
```

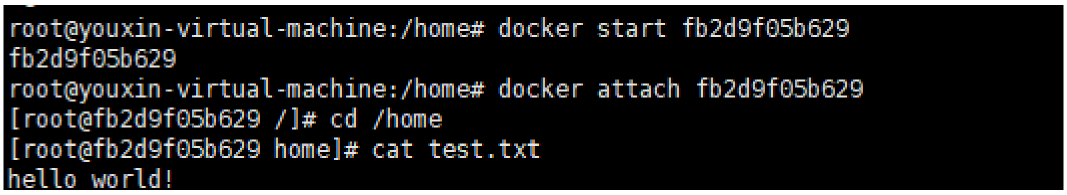
docker inspect 查看centos容器：



```
    },
    "Mounts": [
        {
            "Type": "bind",
            "Source": "/home/test",
            "Destination": "/home",
            "Mode": "",
            "RW": true,
            "Propagation": "rprivate"
        }
    ],
    "Config": {
```

在centos容器的/home文件夹下中创建test.txt文件，发现在宿主机中/home/test文件夹下也同时创建test.txt文件

将容器停止后宿主机中文件依然存在，同时在宿主机中修改test.txt文件，容器启动后也跟着修改了。（可以双向操作）好处：以后修改只需要在本地修改即可，容器内会自动同步。

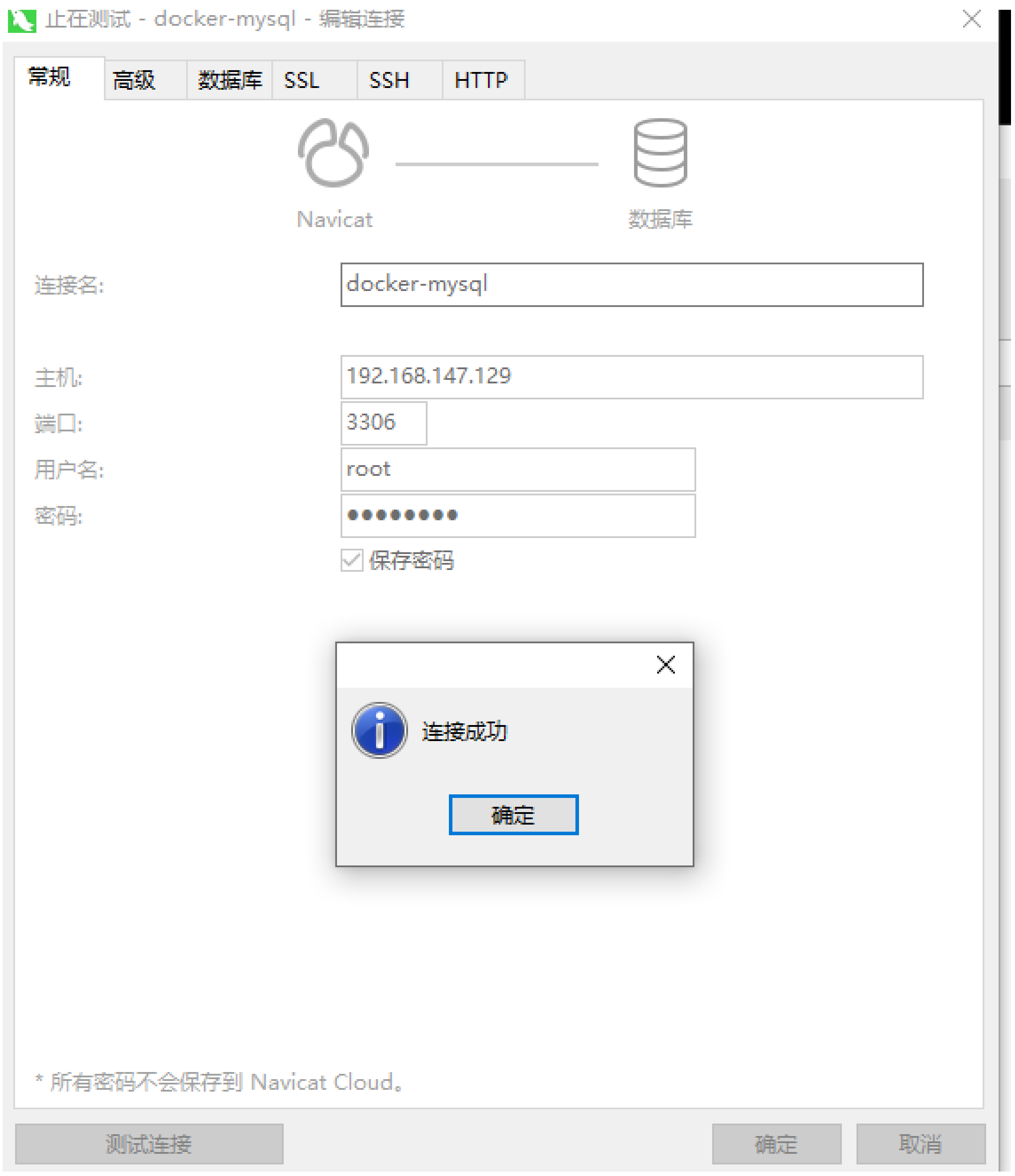


```
root@youxin-virtual-machine:/home# docker start fb2d9f05b629
fb2d9f05b629
root@youxin-virtual-machine:/home# docker attach fb2d9f05b629
[root@fb2d9f05b629 /]# cd /home
[root@fb2d9f05b629 home]# cat test.txt
hello world!
```

## 测试部署Mysql

Mysql的数据持久化问题

```
#获取镜像
docker pull mysql:5.7

#运行容器，需要做数据挂载！注意：安装mysql时需要配置密码！
#官方测试
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag

#实际启动
docker run -d -p 3306:3306 -v /home/mysql/conf:/etc/mysql/comf.d -v
/home/mysql/data:/var/lib/mysql --name="mysql01" -e MYSQL_ROOT_PASSWORD=1234
mysql:5.7
#测试使用navicat连接mysql,发现成功连接
```



正在测试 - docker-mysql - 编辑连接

常规 | 高级 | 数据库 | SSL | SSH | HTTP

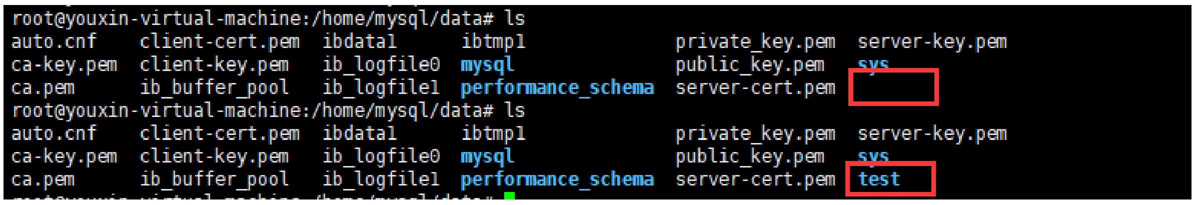
连接名: docker-mysql

主机: 192.168.147.129
端口: 3306
用户名: root
密码: ●●●●●●●●●
☑ 保存密码

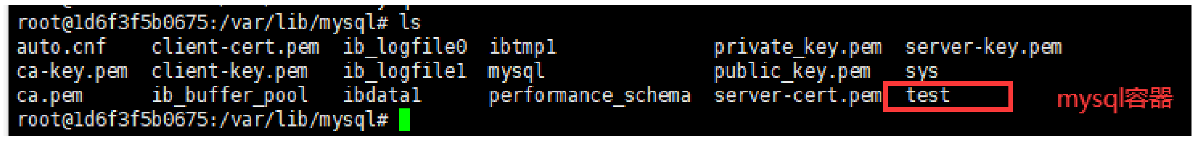
连接成功
确定

* 所有密码不会保存到 Navicat Cloud。

且在navicat中创建数据库：宿主机的/home/mysql/data下也会创建数据库且mysql容器中也会创建数据库：



```
root@youxin-virtual-machine:/home/mysql/data# ls
auto.cnf    client-cert.pem  ibdata1      ibtmp1              private_key.pem  server-key.pem
ca-key.pem  client-key.pem   ib_logfile0  mysql               public_key.pem   sys
ca.pem      ib_buffer_pool   ib_logfile1  performance_schema  server-cert.pem
root@youxin-virtual-machine:/home/mysql/data# ls
auto.cnf    client-cert.pem  ibdata1      ibtmp1              private_key.pem  server-key.pem
ca-key.pem  client-key.pem   ib_logfile0  mysql               public_key.pem   sys
ca.pem      ib_buffer_pool   ib_logfile1  performance_schema  server-cert.pem  test
```

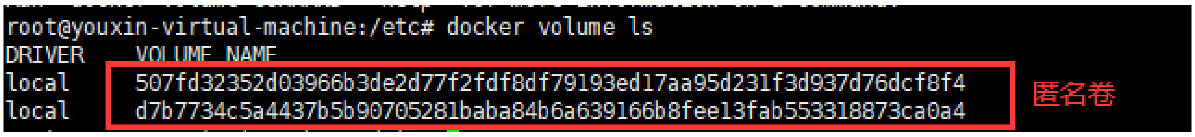


```
root@1d6f3f5b0675:/var/lib/mysql# ls
auto.cnf    client-cert.pem  ib_logfile0  ibtmp1              private_key.pem  server-key.pem
ca-key.pem  client-key.pem   ib_logfile1  mysql               public_key.pem   sys
ca.pem      ib_buffer_pool   ibdata1      performance_schema  server-cert.pem  test
root@1d6f3f5b0675:/var/lib/mysql#
```

即使容器被彻底删除，宿主机上的卷映射文件依然存在，不会被删除！即实现了容器数据持久化功能！

## 具名挂载和匿名挂载

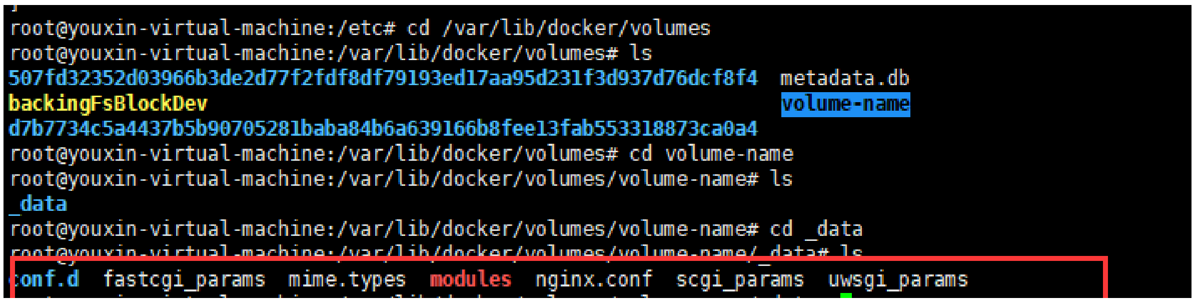
```
#匿名挂载
-v 容器内路径!
docker run -d -p --name ngin01 -v /etc/nginx nginx

#查看所有的卷volume的情况
root@youxin-virtual-machine:/etc# docker volume ls
```



```
root@youxin-virtual-machine:/etc# docker volume ls
DRIVER    VOLUME NAME
local     507fd32352d03966b3de2d77f2fdf8df79193ed17aa95d231f3d937d76dcf8f4
local     d7b7734c5a4437b5b90705281baba84b6a639166b8fee13fab553318873ca0a4
```

```
#具名挂载 通过-v 卷名:容器内路径
root@youxin-virtual-machine:/etc# docker run -d -p 8082:80 --name nginx02 -v
volume-name:/etc/nginx nginx #在-v后面加上卷的名字（名字前不加'/',以'/'开始表示宿主机根路
径下，形成映射，而直接用名字表示给这个卷命名）
3ee566668740f130f3871e8360fb33aa4741a5628bcdf6c9c699882e340ea294
root@youxin-virtual-machine:/etc# docker volume ls
DRIVER    VOLUME NAME
local     507fd32352d03966b3de2d77f2fdf8df79193ed17aa95d231f3d937d76dcf8f4
local     d7b7734c5a4437b5b90705281baba84b6a639166b8fee13fab553318873ca0a4
local     volume-name
#通过卷名查看对应路径
root@youxin-virtual-machine:/etc# docker volume inspect volume-name
[
    {
        "CreatedAt": "2023-05-02T04:28:06-07:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/volume-name/_data",
        "Name": "volume-name",
        "Options": null,
        "Scope": "local"
    }
]
```

所有的Docker容器内的卷，没有指定目录的情况下都是在/var/lib/docker/volumes/xxxx/_data下。



```
root@youxin-virtual-machine:/etc# cd /var/lib/docker/volumes
root@youxin-virtual-machine:/var/lib/docker/volumes# ls
507fd32352d03966b3de2d77f2fdf8df79193ed17aa95d231f3d937d76dcf8f4  metadata.db
backingFsBlockDev                                                 volume-name
d7b7734c5a4437b5b90705281baba84b6a639166b8fee13fab553318873ca0a4
root@youxin-virtual-machine:/var/lib/docker/volumes# cd volume-name
root@youxin-virtual-machine:/var/lib/docker/volumes/volume-name# ls
_data
root@youxin-virtual-machine:/var/lib/docker/volumes/volume-name# cd _data
root@youxin-virtual-machine:/var/lib/docker/volumes/volume-name/_data# ls
conf.d  fastcgi_params  mime.types  modules  nginx.conf  scgi_params  uwsgi_params
```

通过具名挂载可以方便的找到一个卷，大多数情况使用的都是具名挂载。

**小结**

```
#如何确定是具名挂载还是匿名挂载，或者是指定路径挂载
-v 容器内路径          #匿名挂载
-v 卷名:容器内路径     #具名挂载
-v 宿主机路径:容器内路径 #指定路径挂载
```

**扩展**

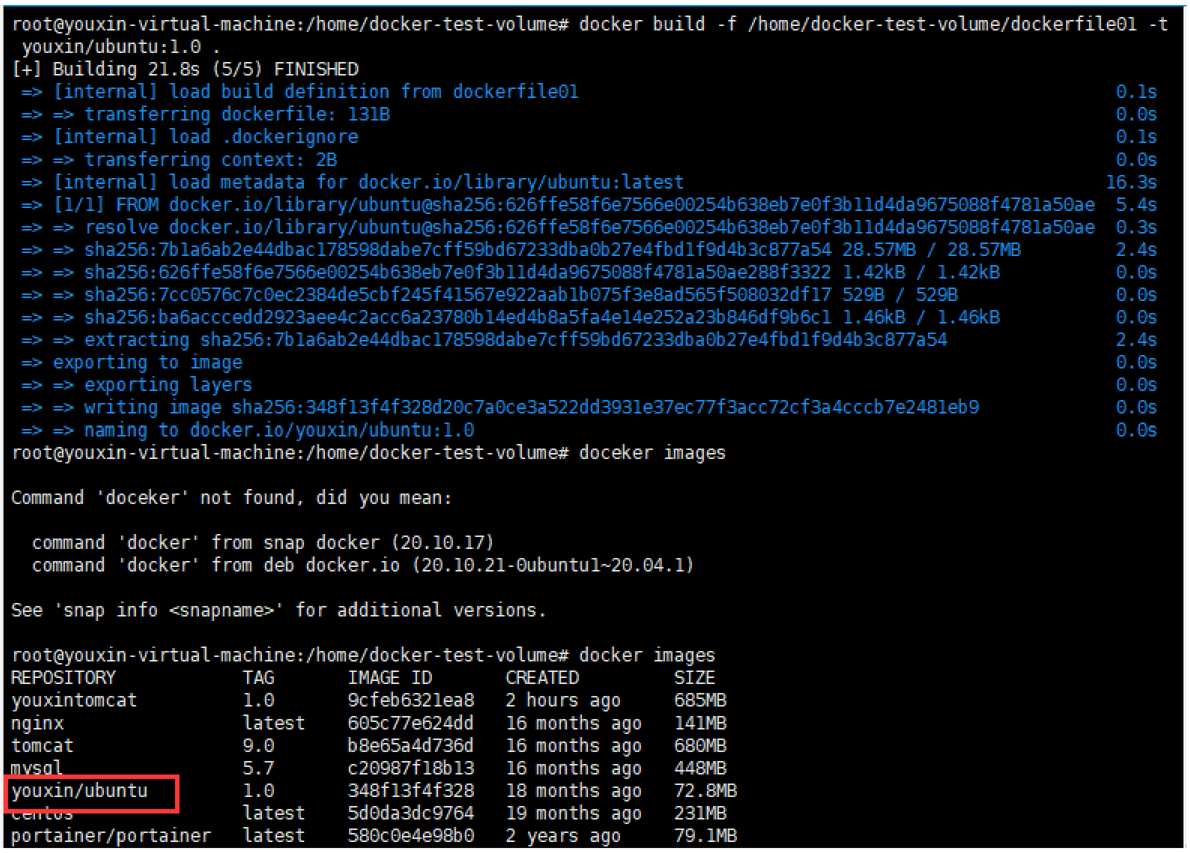
```
#通过 -v 容器内路径:ro/rw 改变读写权限
ro    readonly
rw    readwrite

#一旦设置了容器权限，容器对挂载出来的内容就有限定了！
docker run -d -p 8082:80 --name nginx02 -v volume-name:/etc/nginx:ro nginx #只能从宿主机来改变，容器内部无法操作，只能读操作
docker run -d -p 8082:80 --name nginx02 -v volume-name:/etc/nginx:rw nginx
```

> 方式二：使用DockerFile来挂载卷

DockerFile就是用来构建docker镜像的构建文件，命令脚本。

通过脚本生成镜像，镜像是一层一层的，脚本是一个个的命令，每个命令都是一层。

```
#创建一个DockerFile文件，名字可以随意
#文件内容：指令（大写） 参数
FROM ubuntu

VOLUME ["volume01","volume02"] #匿名挂载

CMD echo "--------end----------"
CMD /bin/bash

#这里的每个命令，就是镜像的一层！
root@youxin-virtual-machine:/home/docker-test-volume# docker build -f /home/docker-test-volume/dockerfile01 -t youxin/ubuntu:1.0 .

```

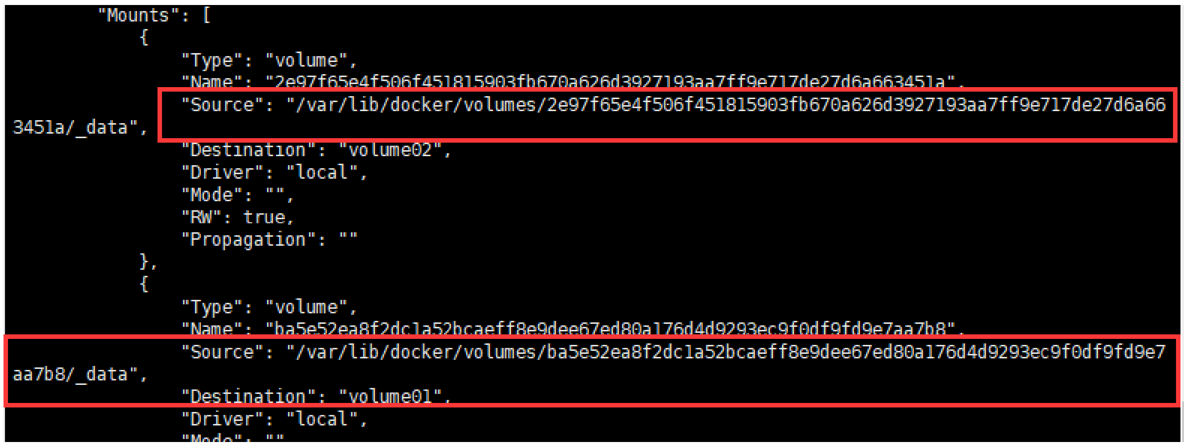


```
root@youxin-virtual-machine:/home/docker-test-volume# docker build -f /home/docker-test-volume/dockerfile01 -t
 youxin/ubuntu:1.0 .
[+] Building 21.8s (5/5) FINISHED
 => [internal] load build definition from dockerfile01                                                    0.1s
 => => transferring dockerfile: 131B                                                                      0.0s
 => [internal] load .dockerignore                                                                         0.1s
 => => transferring context: 2B                                                                           0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest                                         16.3s
 => [1/1] FROM docker.io/library/ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae  5.4s
 => => resolve docker.io/library/ubuntu@sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae  0.3s
 => => sha256:7b1a6ab2e44dbac178598dabe7cff59bd67233dba0b27e4fbd1f9d4b3c877a54 28.57MB / 28.57MB          2.4s
 => => sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322 1.42kB / 1.42kB            0.0s
 => => sha256:7cc0576c7c0ec2384de5cbf245f41567e922aab1b075f3e8ad565f508032df17 529B / 529B                0.0s
 => => sha256:ba6acccedd2923aee4c2acc6a23780b14ed4b8a5fa4e14e252a23b846df9b6c1 1.46kB / 1.46kB            0.0s
 => => extracting sha256:7b1a6ab2e44dbac178598dabe7cff59bd67233dba0b27e4fbd1f9d4b3c877a54                 2.4s
 => exporting to image                                                                                    0.0s
 => => exporting layers                                                                                   0.0s
 => => writing image sha256:348f13f4f328d20c7a0ce3a522dd3931e37ec77f3acc72cf3a4cccb7e2481eb9              0.0s
 => => naming to docker.io/youxin/ubuntu:1.0                                                              0.0s
root@youxin-virtual-machine:/home/docker-test-volume# doceker images

Command 'doceker' not found, did you mean:

  command 'docker' from snap docker (20.10.17)
  command 'docker' from deb docker.io (20.10.21-0ubuntu1~20.04.1)

See 'snap info <snapname>' for additional versions.

root@youxin-virtual-machine:/home/docker-test-volume# docker images
REPOSITORY            TAG       IMAGE ID       CREATED          SIZE
youxintomcat          1.0       9cfeb6321ea8   2 hours ago      685MB
nginx                 latest    605c77e624dd   16 months ago    141MB
tomcat                9.0       b8e65a4d736d   16 months ago    680MB
mysql                 5.7       c20987f18b13   16 months ago    448MB
youxin/ubuntu         1.0       348f13f4f328   18 months ago    72.8MB
centos                latest    5d0da3dc9764   19 months ago    231MB
portainer/portainer   latest    580c0e4e98b0   2 years ago      79.1MB
```

```
#启动自己写的容器
root@youxin-virtual-machine:/home/docker-test-volume# docker run -it
youxin/ubuntu:1.0 /bin/bash
root@108398c22328:/# ls #发现可以进入
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var       volume02 #这个目录就是我们生成镜像的时候自动挂载的，数据卷目录
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volume01 #这个目录就是我们生成镜像的时候自动挂载的，数据卷目录
```

这个卷和外部一定有一个同步的目录

查看卷挂载路径：



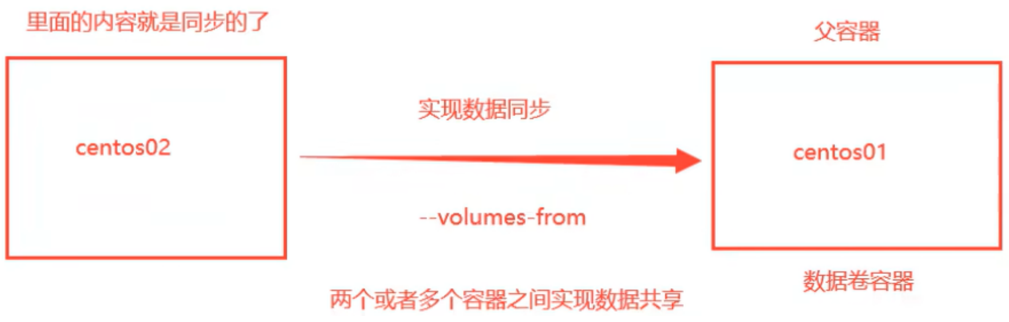
```
        "Mounts": [
            {
                "Type": "volume",
                "Name": "2e97f65e4f506f451815903fb670a626d3927193aa7ff9e717de27d6a663451a",
                "Source": "/var/lib/docker/volumes/2e97f65e4f506f451815903fb670a626d3927193aa7ff9e717de27d6a66
3451a/_data",
                "Destination": "volume02",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            },
            {
                "Type": "volume",
                "Name": "ba5e52ea8f2dc1a52bcaeff8e9dee67ed80a176d4d9293ec9f0df9fd9e7aa7b8",
                "Source": "/var/lib/docker/volumes/ba5e52ea8f2dc1a52bcaeff8e9dee67ed80a176d4d9293ec9f0df9fd9e7
aa7b8/_data",
                "Destination": "volume01",
                "Driver": "local",
                "Mode": ""
```

这种方式使用的十分多，因为我们通常会构建自己的镜像！

假设构建镜像的时候没有挂载卷，要手动镜像挂载-v 卷名:容器内路径！

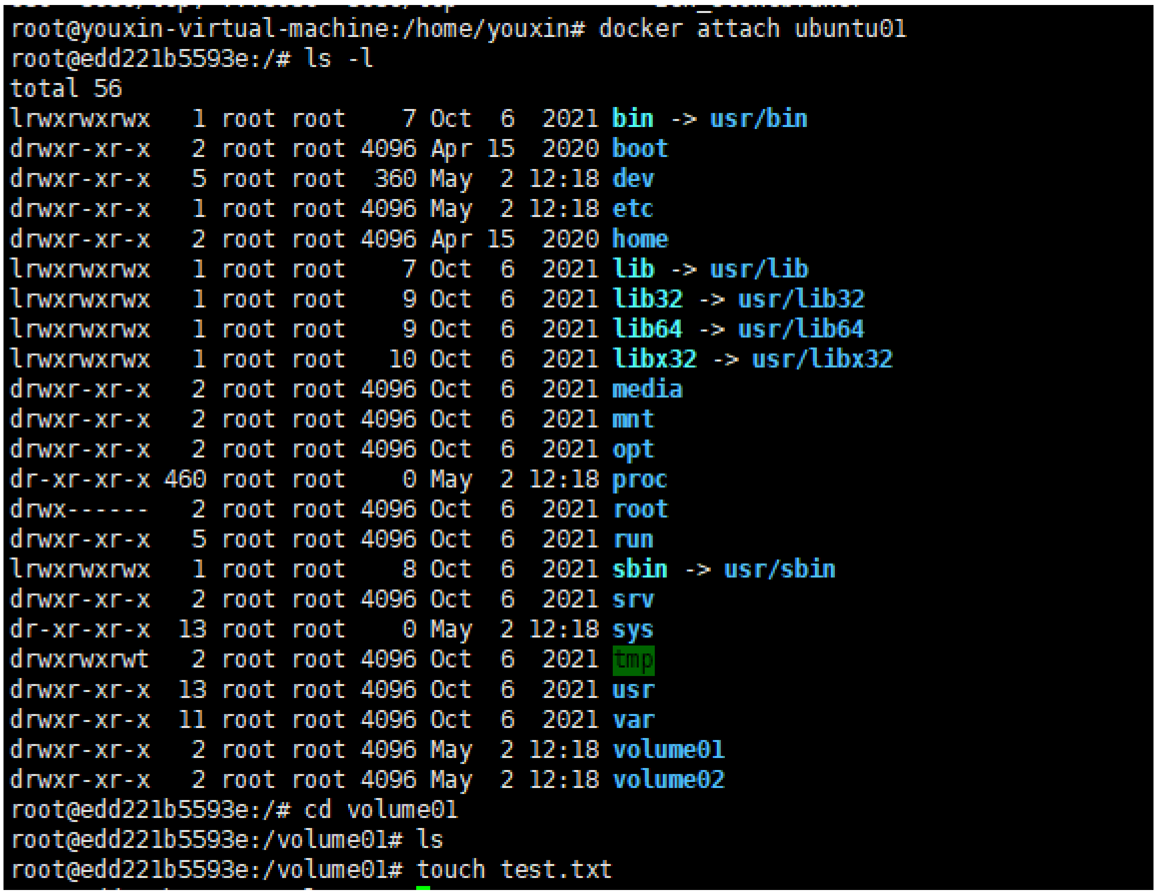
## 数据卷容器（容器和容器之间同步）

如：多个mysql同步数据



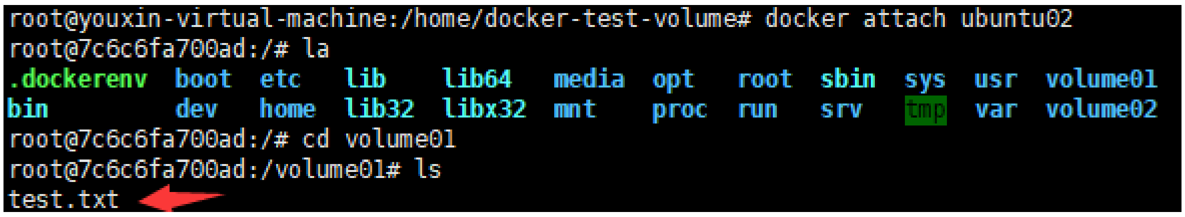
```
#启动3个容器，且通过自己写的镜像启动

#创建ubuntu01容器：
docker run -it --name ubuntu01 youxin/ubuntu:1.0 /bin/bash
#创建ubuntu02容器并将ubuntu01容器作为自己的父容器
docker run -it --name ubuntu02 --volumes-from ubuntu01 youxin/ubuntu:1.0
/bin/bash
```

在ubuntu01容器下的volume01目录下创建test文件



```
root@youxin-virtual-machine:/home/youxin# docker attach ubuntu01
root@edd221b5593e:/# ls -l
total 56
lrwxrwxrwx   1 root root    7 Oct  6  2021 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 15  2020 boot
drwxr-xr-x   5 root root  360 May  2 12:18 dev
drwxr-xr-x   1 root root 4096 May  2 12:18 etc
drwxr-xr-x   2 root root 4096 Apr 15  2020 home
lrwxrwxrwx   1 root root    7 Oct  6  2021 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Oct  6  2021 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Oct  6  2021 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Oct  6  2021 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Oct  6  2021 media
drwxr-xr-x   2 root root 4096 Oct  6  2021 mnt
drwxr-xr-x   2 root root 4096 Oct  6  2021 opt
dr-xr-xr-x 460 root root    0 May  2 12:18 proc
drwx------   2 root root 4096 Oct  6  2021 root
drwxr-xr-x   5 root root 4096 Oct  6  2021 run
lrwxrwxrwx   1 root root    8 Oct  6  2021 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Oct  6  2021 srv
dr-xr-xr-x  13 root root    0 May  2 12:18 sys
drwxrwxrwt   2 root root 4096 Oct  6  2021 tmp
drwxr-xr-x  13 root root 4096 Oct  6  2021 usr
drwxr-xr-x  11 root root 4096 Oct  6  2021 var
drwxr-xr-x   2 root root 4096 May  2 12:18 volume01
drwxr-xr-x   2 root root 4096 May  2 12:18 volume02
root@edd221b5593e:/# cd volume01
root@edd221b5593e:/volume01# ls
root@edd221b5593e:/volume01# touch test.txt
```

发现在ubuntu02容器下的volume01目录下也自定创建了test文件（反向操作同样可以）：



```
root@youxin-virtual-machine:/home/docker-test-volume# docker attach ubuntu02
root@7c6c6fa700ad:/# la
.dockerenv  boot  etc   lib    lib64   media  opt   root  sbin  sys  usr  volume01
bin         dev   home  lib32  libx32  mnt    proc  run   srv   tmp  var  volume02
root@7c6c6fa700ad:/# cd volume01
root@7c6c6fa700ad:/volume01# ls
test.txt
```

删除ubuntu01后，ubuntu02的文件依旧存在！

例如：多个mysql实现数据共享

```
docker run -d -p 3306:3306 -v /home/mysql/conf:/etc/mysql/comf.d -v
/home/mysql/data:/var/lib/mysql --name="mysql01" -e MYSQL_ROOT_PASSWORD=1234
mysql:5.7

docker run -d -p 3307:3306 --name mysql02 -e MYSQL_ROOT_PASSWORD=1234 --volumes-
from mysql01 mysql:5.7

#这个时候，可以实现两个数据库同步操作。
```

结论：

容器之间配置信息的传递，数据卷容器的生命周期一直持续到没有容器使用为止。一旦持久化到了本地，这时，本地的容器是不会删除的。如果在容器中，容器删除数据就没了。

# DockerFile

## DockerFile介绍

DockerFile是用来构建docker镜像的文件！命令参数脚本！

构建步骤：

1、编写dockerfile文件

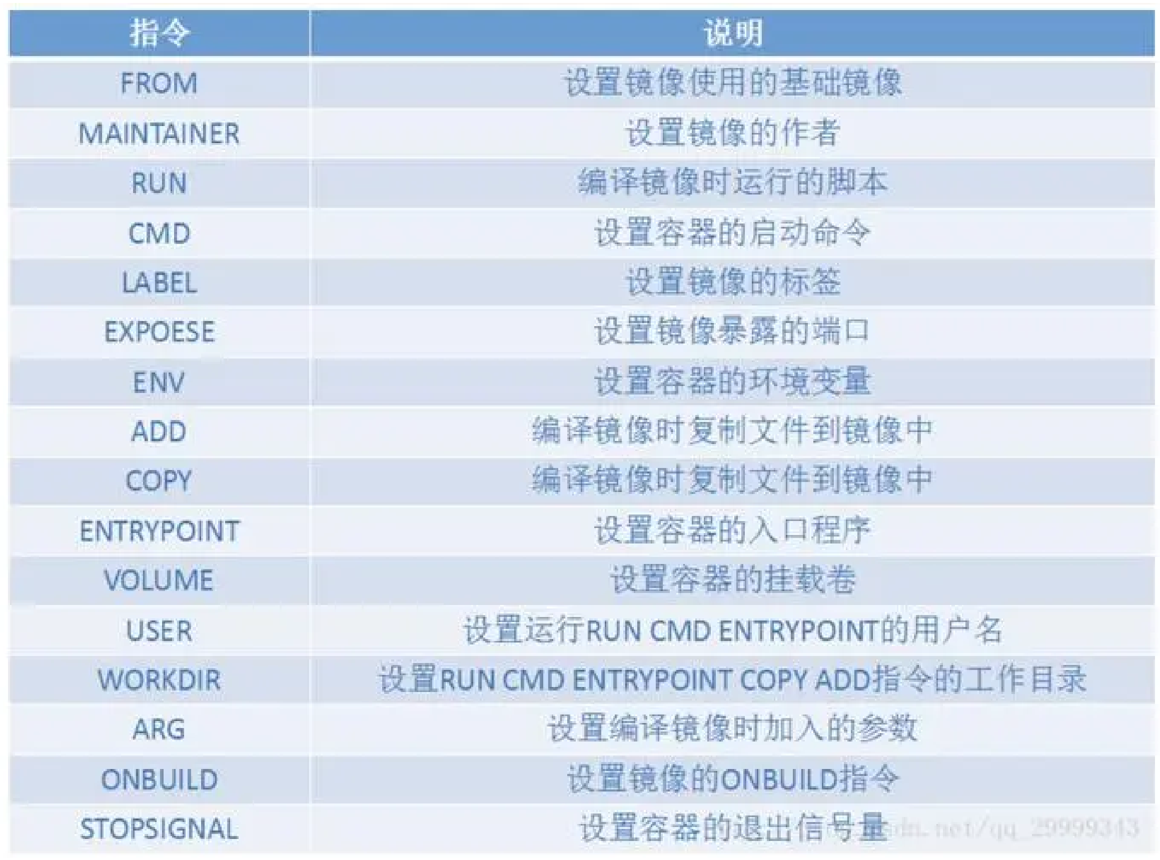
2、docker build构建称为一个镜像

3、docker run运行镜像

4、docker push发布镜像（dockerhub 或者阿里云镜像）

## DockerFile构建过程

指令：



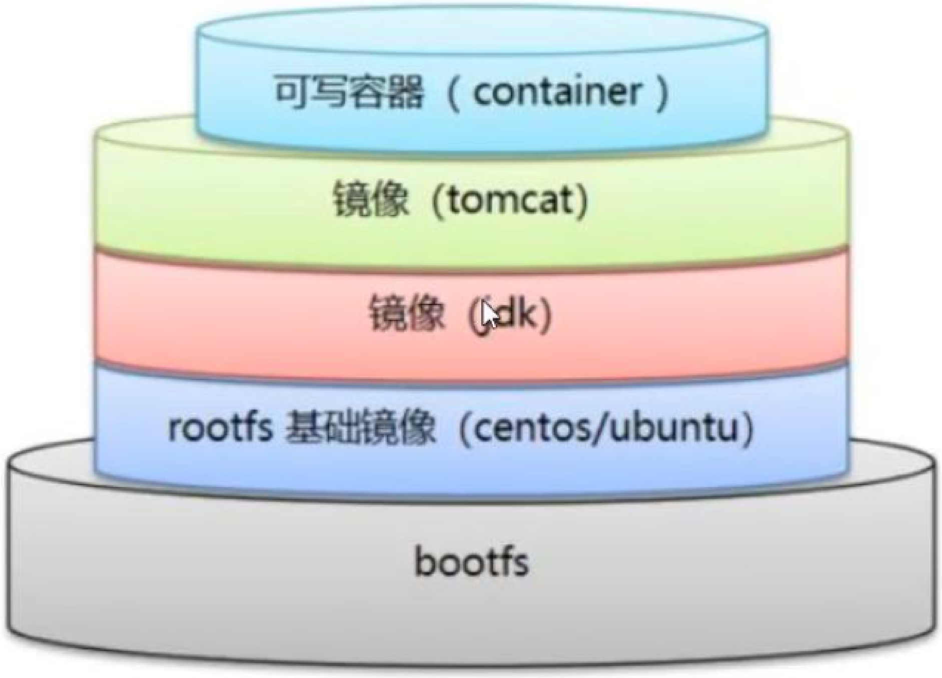
| 指令 | 说明 |
|---|---|
| FROM | 设置镜像使用的基础镜像 |
| MAINTAINER | 设置镜像的作者 |
| RUN | 编译镜像时运行的脚本 |
| CMD | 设置容器的启动命令 |
| LABEL | 设置镜像的标签 |
| EXPOESE | 设置镜像暴露的端口 |
| ENV | 设置容器的环境变量 |
| ADD | 编译镜像时复制文件到镜像中 |
| COPY | 编译镜像时复制文件到镜像中 |
| ENTRYPOINT | 设置容器的入口程序 |
| VOLUME | 设置容器的挂载卷 |
| USER | 设置运行RUN CMD ENTRYPOINT的用户名 |
| WORKDIR | 设置RUN CMD ENTRYPOINT COPY ADD指令的工作目录 |
| ARG | 设置编译镜像时加入的参数 |
| ONBUILD | 设置镜像的ONBUILD指令 |
| STOPSIGNAL | 设置容器的退出信号量 |

1、每个保留关键字（指令）都是必须大写字母

2、执行从上到下顺序执行

3、#表示注释

4、每一个指令都会创建一个新的镜像层，并提交



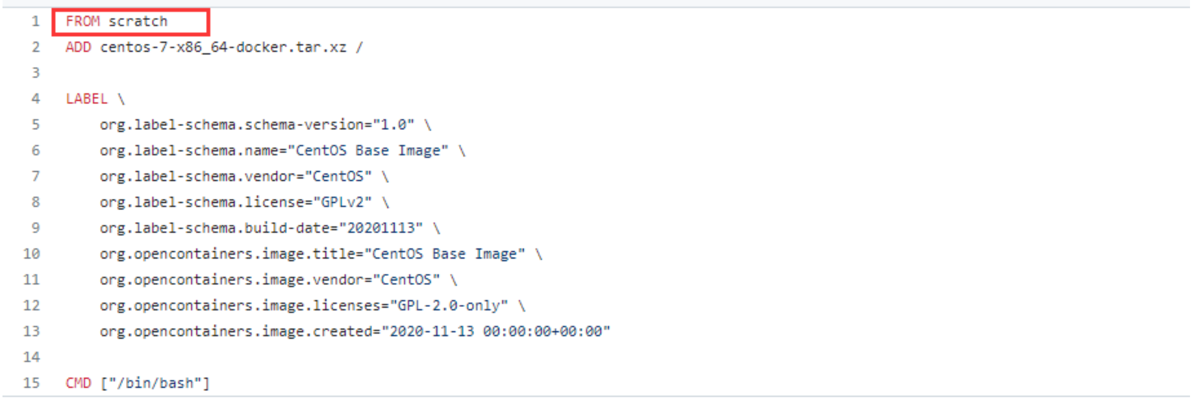
dockerfile是面向开发的，以后发布项目，做镜像就需要编写docekrfile文件。

## DockerFile的指令

```
FROM                #基础镜像，一切从这里开始构建
MAINTAINER          #镜像是谁写的，姓名+邮箱
RUN                 #镜像构建的时候需要运行的命令
ADD                 #步骤，tomcat镜像，这个tomcat压缩包就是添加内容
WORKDIR             #镜像的工作目录
VOLUME              #挂载的目录位置
EXPOSE              #暴露端口配置
CMD                 #指定这个容器启动的时候要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT          #指定这个容器启动的时候要运行的命令，可以追加命令
ONBUILD             #当构建一个被继承DockerFile，这个时候就会运行ONBUILD的指令，触发指令
COPY                #类似ADD命令，将文件拷贝到镜像中
ENV                 #构建的时候设置环境变量
```

**测试：**

Docker Hub中99%镜像都是从这个基础镜像过来的FROM scratch ,然后配置需要的软件和配置来进行的构建



```
FROM scratch
ADD centos-7-x86_64-docker.tar.xz /

LABEL \
    org.label-schema.schema-version="1.0" \
    org.label-schema.name="CentOS Base Image" \
    org.label-schema.vendor="CentOS" \
    org.label-schema.license="GPLv2" \
    org.label-schema.build-date="20201113" \
    org.opencontainers.image.title="CentOS Base Image" \
    org.opencontainers.image.vendor="CentOS" \
    org.opencontainers.image.licenses="GPL-2.0-only" \
    org.opencontainers.image.created="2020-11-13 00:00:00+00:00"

CMD ["/bin/bash"]
```

> 创建一个自己的centos

```
#在/home/dockerfile下新建文件mydockerfile

#1、编写dockerfile文件
FROM centos:7
MAINTAINER youxin<3228105317@qq.com>

ENV MYPATH /usr/local
WORKDIR $MYPATH #默认路径为/，即根目录，设置WORKDIR后，进入镜像时会自动进入设置的工作目录中

RUN yum -y install vim #新增vim和网络工具功能
RUN yum -y install net-tools

EXPOSE 80

CMD echo $MYPATH
CMD echo "===========end=========="
CMD /bin/bash

#2、通过这个文件构建镜像
#命令 docker build -f dockerfile文件路径 -t 镜像名:[tag]
root@youxin-virtual-machine:/home/dockerfile# docker build -f mydockerfile -t
mycentos:0.1 . #后面加个点表示当前目录
[+] Building 73.3s (8/8) FINISHED

 => [internal] load build definition from mydockerfile
                        0.0s
 => => transferring dockerfile: 263B
                        0.0s
 => [internal] load .dockerignore
                         0.0s
 => => transferring context: 2B
                         0.0s
 => [internal] load metadata for docker.io/library/centos:7
                        16.4s
 => [1/4] FROM
docker.io/library/centos:7@sha256:9d4bcbbb213dfd745b58be38b13b996ebb5ac315fe7571
1bd6184  13.9s
 => => resolve
docker.io/library/centos:7@sha256:9d4bcbbb213dfd745b58be38b13b996ebb5ac315fe7571
1bd61842  0.0s
 => => sha256:2d473b07cdd5f0912cd6f1a703352c82b512407db6b05b43f2553732b55df3bc
76.10MB / 76.10MB         5.5s
 => => sha256:9d4bcbbb213dfd745b58be38b13b996ebb5ac315fe75711bd618426a630e0987
1.20kB / 1.20kB           0.0s
 => => sha256:dead07b4d8ed7e29e98de0f4504d87e8880d4347859d839686a31da35a3b532f
529B / 529B               0.0s
 => => sha256:eeb6ee3f44bd0b5103bb561b4c16bcb82328cfe5809ab675bb17ab3a16c517c9
2.75kB / 2.75kB           0.0s
 => => extracting
sha256:2d473b07cdd5f0912cd6f1a703352c82b512407db6b05b43f2553732b55df3bc
     7.1s
 => [2/4] WORKDIR /usr/local
                        1.8s
 => [3/4] RUN yum -y install vim
                       32.3s
 => [4/4] RUN yum -y install net-tools
                        5.4s
 => exporting to image
                       3.4s
 => => exporting layers
                        3.2s
 => => writing image
sha256:0506e6c0051d3cd0f666a54231c201a7e85b226c9642d25a7198e3d13024a0bb
  0.0s
 => => naming to docker.io/library/mycentos:0.1
                        0.2s

 #3、测试运行
 root@youxin-virtual-machine:/home/dockerfile# docker run -it mycentos:0.1
[root@3c7635b6d6c1 local]# ifconfig #原本的centos是没有ifconfig命令的
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.4  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:04  txqueuelen 0  (Ethernet)
        RX packets 21  bytes 2723 (2.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@3c7635b6d6c1 local]# pwd #直接进入了WORKDIR声明的目录
/usr/local
```

列出本地镜像的变更历史：

```
#命令 docker history 镜像id
root@youxin-virtual-machine:/home/dockerfile# docker history 0506e6c0051d
IMAGE          CREATED          CREATED BY
SIZE      COMMENT
0506e6c0051d   12 minutes ago   CMD ["/bin/sh" "-c" "/bin/bash"]
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   CMD ["/bin/sh" "-c" "echo \"===========end==…
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   CMD ["/bin/sh" "-c" "echo $MYPATH"]
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   EXPOSE map[80/tcp:{}]
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   RUN /bin/sh -c yum -y install net-tools # bu…
177MB     buildkit.dockerfile.v0
<missing>      12 minutes ago   RUN /bin/sh -c yum -y install vim # buildkit
259MB     buildkit.dockerfile.v0
<missing>      12 minutes ago   WORKDIR /usr/local
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   ENV MYPATH=/usr/local
0B        buildkit.dockerfile.v0
<missing>      12 minutes ago   MAINTAINER youxin<3228105317@qq.com>
0B        buildkit.dockerfile.v0
```

```
<missing>       19 months ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]
0B
<missing>       19 months ago    /bin/sh -c #(nop)  LABEL org.label-schema.sc…
0B
<missing>       19 months ago    /bin/sh -c #(nop) ADD file:b3ebbe8bd304723d4…
204MB
```

> CMD和ENTRYPOINT区别

```
CMD                 #指定这个容器启动的时候要运行的命令，只有最后一个会生效，可被替代
ENTRYPOINT          #指定这个容器启动的时候要运行的命令，可以追加命令
```

测试cmd

```
#编写dockerfile文件
root@youxin-virtual-machine:/home/dockerfile# cat dockerfile-cmd-test  #编写
DockerFile
FROM centos:7

CMD ["ls","-a"]  #run的时候执行ls -a命令

#构建镜像

#run运行，发现CMD命令生效了
root@youxin-virtual-machine:/home/dockerfile# docker run d53776618e7c
.
..
.dockerenv
anaconda-post.log
bin
dev
etc
home
lib
lib64
media
#想要追加一个命令-l 即run后执行ls -al
root@youxin-virtual-machine:/home/dockerfile# docker run d53776618e7c -l
docker: Error response from daemon: failed to create shim task: OCI runtime
create failed: runc create failed: unable to start container process: exec: "-
l": executable file not found in $PATH: unknown.

#cmd的情况下 -l替换了CMD["ls", "-a"]命令，-l不是命令所以会报错
#此时只有在run后启动完整的ls -al命令才可以正确执行并且替换cmd命令
```

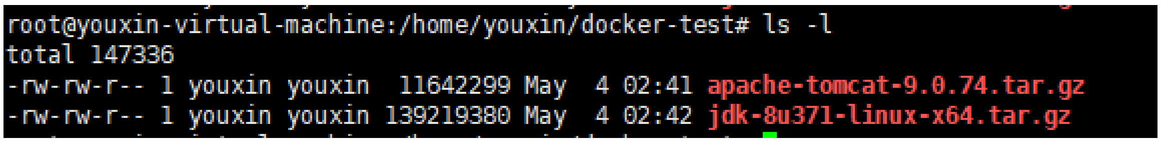
测试ENTRYPOINT

```
#使用entrypoint编写dockerfile
root@youxin-virtual-machine:/home/dockerfile# cat dockerfile-cmd-entrypoint
FROM centos:7

ENTRYPOINT ["ls", "-a"]

#构建镜像

#追加-l命令启动容器，此时没有报错并且在原来的ls -a命令后添加-l命令
```

```
root@youxin-virtual-machine:/home/dockerfile# docker run entrypointtest -l
total 64
drwxr-xr-x   1 root root  4096 May  4 08:40 .
drwxr-xr-x   1 root root  4096 May  4 08:40 ..
-rwxr-xr-x   1 root root     0 May  4 08:40 .dockerenv
-rw-r--r--   1 root root 12114 Nov 13  2020 anaconda-post.log
lrwxrwxrwx   1 root root     7 Nov 13  2020 bin -> usr/bin
drwxr-xr-x   5 root root   340 May  4 08:40 dev
drwxr-xr-x   1 root root  4096 May  4 08:40 etc
drwxr-xr-x   2 root root  4096 Apr 11  2018 home
lrwxrwxrwx   1 root root     7 Nov 13  2020 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Nov 13  2020 lib64 -> usr/lib64
drwxr-xr-x   2 root root  4096 Apr 11  2018 media
drwxr-xr-x   2 root root  4096 Apr 11  2018 mnt
```

## 实战：Tomcat镜像

1、准备镜像文件tomcat压缩包，jdk压缩包



```
root@youxin-virtual-machine:/home/youxin/docker-test# ls -l
total 147336
-rw-rw-r-- 1 youxin youxin  11642299 May  4 02:41 apache-tomcat-9.0.74.tar.gz
-rw-rw-r-- 1 youxin youxin 139219380 May  4 02:42 jdk-8u371-linux-x64.tar.gz
```

2、编写dockerfile文件，官方命名`Dockerfile`，build会自动寻找这个文件，不需要-f指定了

```
FROM ubuntu
MAINTAINER youxin<3228105317@qq.com>

COPY readme.txt /usr/local/readme.txt
ADD jdk-8u371-linux-x64.tar.gz /usr/local/
ADD apache-tomcat-9.0.74.tar.gz /usr/local/

RUN apt-get -y install vim #配置-y选项自动确认不用输入y确认下载
ENV MYPATH /usr/local
WORKDIR $MYPATH

ENV JAVA_HOME /usr/local/jdk1.8.0_371
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar

ENV CATALINA_HOME /usr/local/apache-tomcat-9.0.74
ENV CATALINA_BASH /usr/local/apache-tomcat-9.0.74
ENV PATH $PATH:$JAVA_HOME/bin:$CATALINA_HOME/lib:$CATALINA_HOME/bin

EXPOSE 8080
CMD //usr/local/apache-tomcat-9.0.74/bin/startup.sh && tail -F
/usr/local/apache-tomcat-9.0.74/bin/logs/catali
ne.out
```

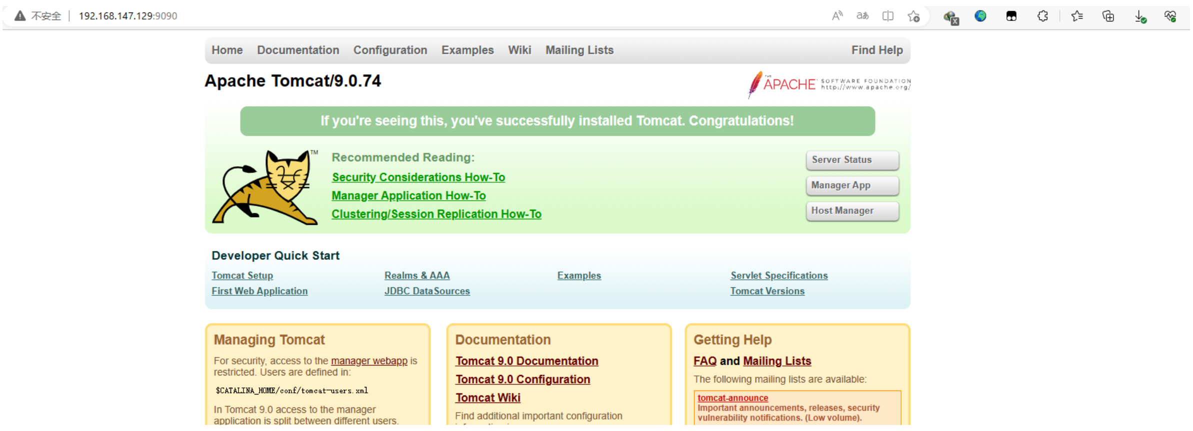
3、构建镜像

```
#docker build
root@youxin-virtual-machine:/home/youxin/docker-test# docker build -t diytomcat
.
[+] Building 71.3s (12/12) FINISHED
```

4、启动镜像

```
root@youxin-virtual-machine:/home/youxin/docker-test# docker run -d -p 9090:8080
--name youxintomcat -v /home/
youxin/docker-test/test:/usr/local/apache-tomcat-9.0.74/webapps/test -v
/home/youxin/docker-test/tomcatlogs:/usr/local/apache-tomcat-9.0.74/logs
diytomcat
```

5、访问


不安全 | 192.168.147.129:9090

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/9.0.74

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Server Status
Manager App
Host Manager

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
$CATALINA_HOME/conf/tomcat-users.xml
In Tomcat 9.0 access to the manager application is split between different users.

Documentation
Tomcat 9.0 Documentation
Tomcat 9.0 Configuration
Tomcat Wiki
Find additional important configuration

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).

6、发布项目（由于做了卷挂载，我们直接在本地编写项目就可以发布）

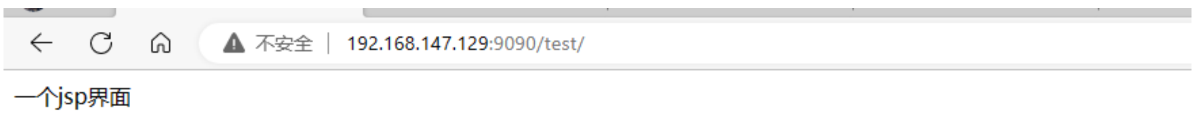
在挂载的/home/youxin/docker-test/test下创建WEB-INF添加web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
</web-app>
```

在test下创建index.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    一个jsp界面
</body>
</html>
```

6、访问http://192.168.147.129:9090/test


不安全 | 192.168.147.129:9090/test/

一个jsp界面

## 发布自己的镜像

> DockerHub

1、dockerhub注册自己账号

2、在服务器上提交自己的镜像

```
# docker login
root@youxin-virtual-machine:/home/youxin/docker-test/tomcatlogs# docker login -u
x1nzz
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
```

3、登录完成后就可以提交镜像了

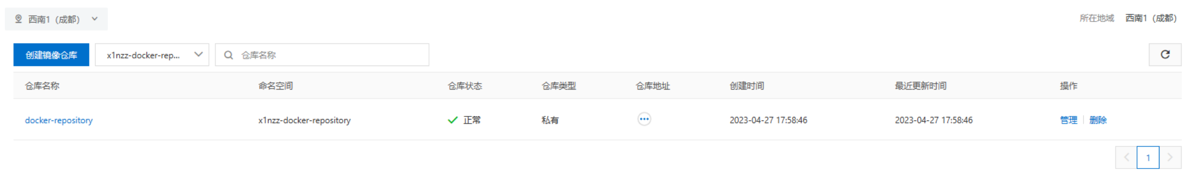
```
#docker push
#因为DockerHub提交有规定，所以需要先使用docker tag 修改镜像名
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
#规定必须为[dockerhub用户名]/imagename:[tag] 格式
root@youxin-virtual-machine:/home/youxin/docker-test# docker tag diytomcat
x1nzz/diydocker:1.0.0
root@youxin-virtual-machine:/home/youxin/docker-test# docker push
x1nzz/diydocker:1.0.0
The push refers to repository [docker.io/x1nzz/diydocker]
5f70bf18a086: Pushed
c712057f14c6: Pushed
f9488092b308: Pushed
5afe91784d1a: Pushing  16.75MB
a676904c31d1: Pushing   4.42MB/343.1MB
942d189c6685: Pushing   2.56kB
9f54eef41275: Pushed
#提交成功
#此时提交太慢修改dns
root@youxin-virtual-machine:/etc# vim resolv.conf
nameserver 8.8.8.8
nameserver 4.4.4.4
```

> 阿里云镜像服务器上

1、进入阿里云镜像服务

2、创建命名空间

4、创建容器镜像

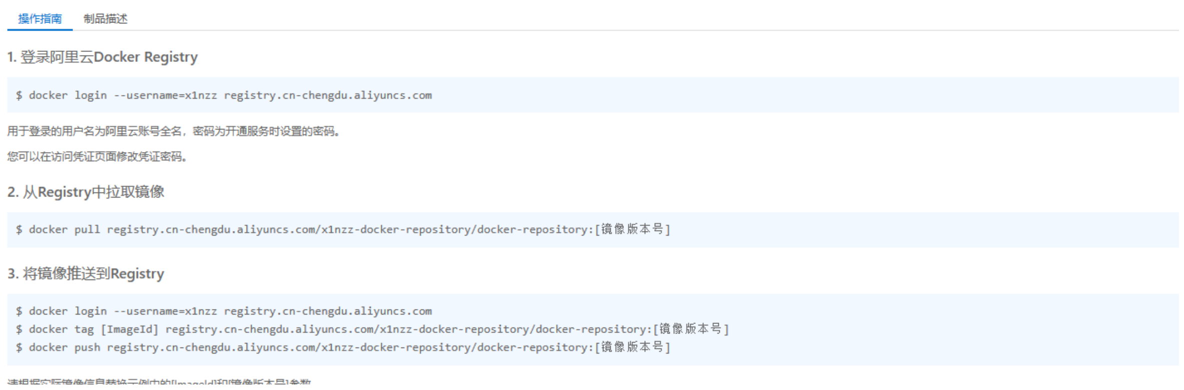


西南1（成都）　　　　所在地域　西南1（成都）

创建镜像仓库　x1nzz-docker-rep...　仓库名称

| 仓库名称 | 命名空间 | 仓库状态 | 仓库类型 | 仓库地址 | 创建时间 | 最近更新时间 | 操作 |
|---|---|---|---|---|---|---|---|
| docker-repository | x1nzz-docker-repository | ✓ 正常 | 私有 | ··· | 2023-04-27 17:58:46 | 2023-04-27 17:58:46 | 管理 \| 删除 |

## 5、推送镜像



操作指南　制品描述

1. 登录阿里云Docker Registry

```
$ docker login --username=x1nzz registry.cn-chengdu.aliyuncs.com
```

用于登录的用户名为阿里云账号全名，密码为开通服务时设置的密码。

您可以在访问凭证页面修改凭证密码。

2. 从Registry中拉取镜像

```
$ docker pull registry.cn-chengdu.aliyuncs.com/x1nzz-docker-repository/docker-repository:[镜像版本号]
```

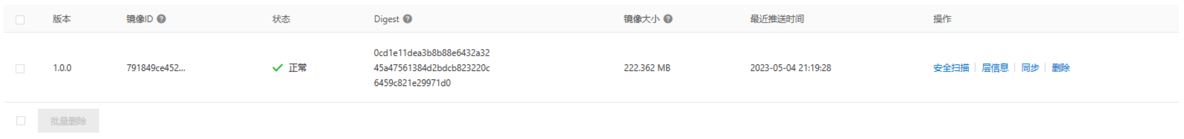
3. 将镜像推送到Registry

```
$ docker login --username=x1nzz registry.cn-chengdu.aliyuncs.com
$ docker tag [ImageId] registry.cn-chengdu.aliyuncs.com/x1nzz-docker-repository/docker-repository:[镜像版本号]
$ docker push registry.cn-chengdu.aliyuncs.com/x1nzz-docker-repository/docker-repository:[镜像版本号]
```

```
$ docker login --username=x1nzz registry.cn-chengdu.aliyuncs.com
$ docker tag [ImageId] registry.cn-chengdu.aliyuncs.com/x1nzz-docker-
repository/docker-repository:[镜像版本号]
$ docker push registry.cn-chengdu.aliyuncs.com/x1nzz-docker-repository/docker-
repository:[镜像版本号]
```

## 6、push成功

```
root@youxin-virtual-machine:/home/youxin/docker-test# docker push registry.cn-
chengdu.aliyuncs.com/x1nzz-docker-repository/docker-repository:1.0.0
The push refers to repository [registry.cn-chengdu.aliyuncs.com/x1nzz-docker-
repository/docker-repository]
5f70bf18a086: Pushed
c712057f14c6: Pushed
f9488092b308: Pushed
5afe91784d1a: Pushed
a676904c31d1: Pushed
942d189c6685: Pushed
9f54eef41275: Pushed
1.0.0: digest:
sha256:0cd1e11dea3b8b88e6432a3245a47561384d2bdcb823220c6459c821e29971d0 size:
1791
```



| | 版本 | 镜像ID | 状态 | Digest | 镜像大小 | 最近推送时间 | 操作 |
|---|---|---|---|---|---|---|---|
| ☐ | 1.0.0 | 791849ce452... | ✓ 正常 | 0cd1e11dea3b8b88e6432a32 45a47561384d2bdcb823220c 6459c821e29971d0 | 222.362 MB | 2023-05-04 21:19:28 | 安全扫描 \| 层信息 \| 同步 \| 删除 |

批量删除

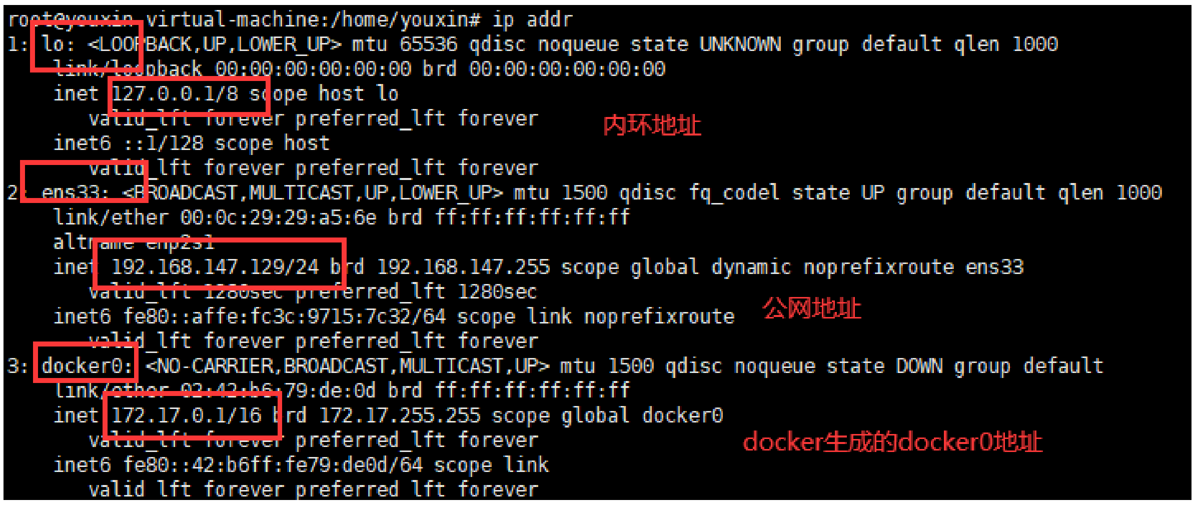
## 小结

img

# Docker网络

## 理解Docker0

> 测试



```
root@youxin-virtual-machine:/home/youxin# 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 fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:29:a5:6e brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.147.129/24 brd 192.168.147.255 scope global dynamic noprefixroute ens33
       valid_lft 1280sec preferred_lft 1280sec
    inet6 fe80::affe:fc3c:9715:7c32/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:b6:79:de:0d 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:b6ff:fe79:de0d/64 scope link
       valid_lft forever preferred_lft forever
```

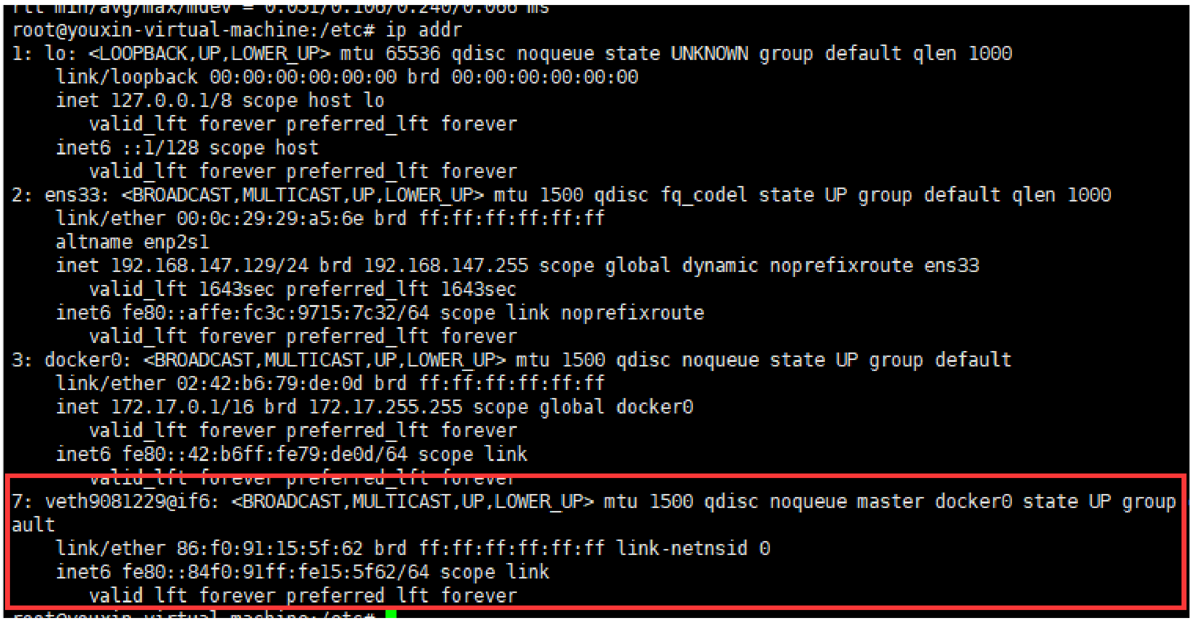
三个网络

```
#问题：Docker是如何处理容器网络访问的？

#启动tomcat
docker run -d -p 8080:8080 --name tomcat01 tomcat:9.0

#容器内查看ip地址（安装net-tools, iproute2,且更新apt-get）
root@130ac519b5f8:/usr/local/tomcat# 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
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever

#思考容器外能不能直接ping通容器内部，发现可以ping通
root@youxin-virtual-machine:/etc# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.197 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.094 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.072 ms
```

> 原理

1、每启动一个docker，docker就会给docker容器分配一个ip，只要安装了docker，就会有一个网卡docker0桥接模式，使用的技术是evth-pair技术。

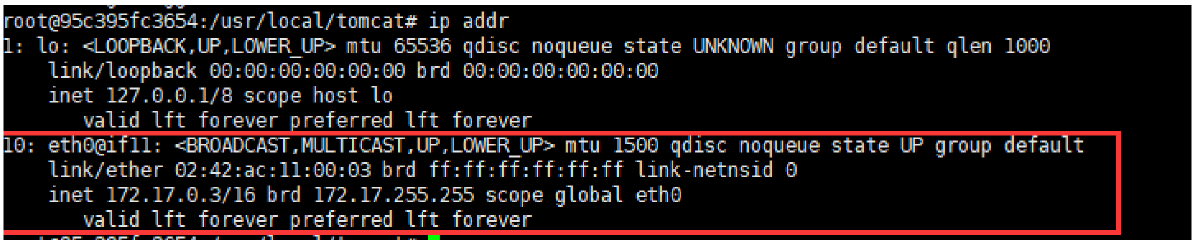
再次测试ip addr



```
rtt min/avg/max/mdev = 0.051/0.106/0.240/0.066 ms
root@youxin-virtual-machine:/etc# 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 fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:29:a5:6e brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.147.129/24 brd 192.168.147.255 scope global dynamic noprefixroute ens33
       valid_lft 1643sec preferred_lft 1643sec
    inet6 fe80::affe:fc3c:9715:7c32/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:b6:79:de:0d 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:b6ff:fe79:de0d/64 scope link
       valid_lft forever preferred_lft forever
7: veth9081229@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group def
ault
    link/ether 86:f0:91:15:5f:62 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::84f0:91ff:fe15:5f62/64 scope link
       valid_lft forever preferred_lft forever
root@youxin-virtual-machine:/etc#
```

2、再启动一个容器测试

```
root@youxin-virtual-machine:/etc# docker run -d -p 8081:8080 --name tomcat02
tomcat:9.0
```



```
root@95c395fc3654:/usr/local/tomcat# 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
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

发现容器生成的网卡，都是一对一对的，evth-paor就是一对的虚拟接口设备，他们都是成对出现的，一端连着协议，一端彼此相连，正因为有这个特性，evth-pair充当一个桥梁，连接各种虚拟设备。

3、通过tomcat01去ping tomcat02容器

```
#通过tomcat01去ping tomcat02容器
#发现同样可以ping的通（安装iputils-ping）
root@130ac519b5f8:/usr/local/tomcat# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.199 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.171 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.142 ms
```

结论：tomcat01和tomcat02是共用的一个路由器，docker0

所有容器在不指定网络的情况下，都是docker0路由的，docker会容器分配一个默认的可用IP。只要容器被删除，对应网桥一对就没了！

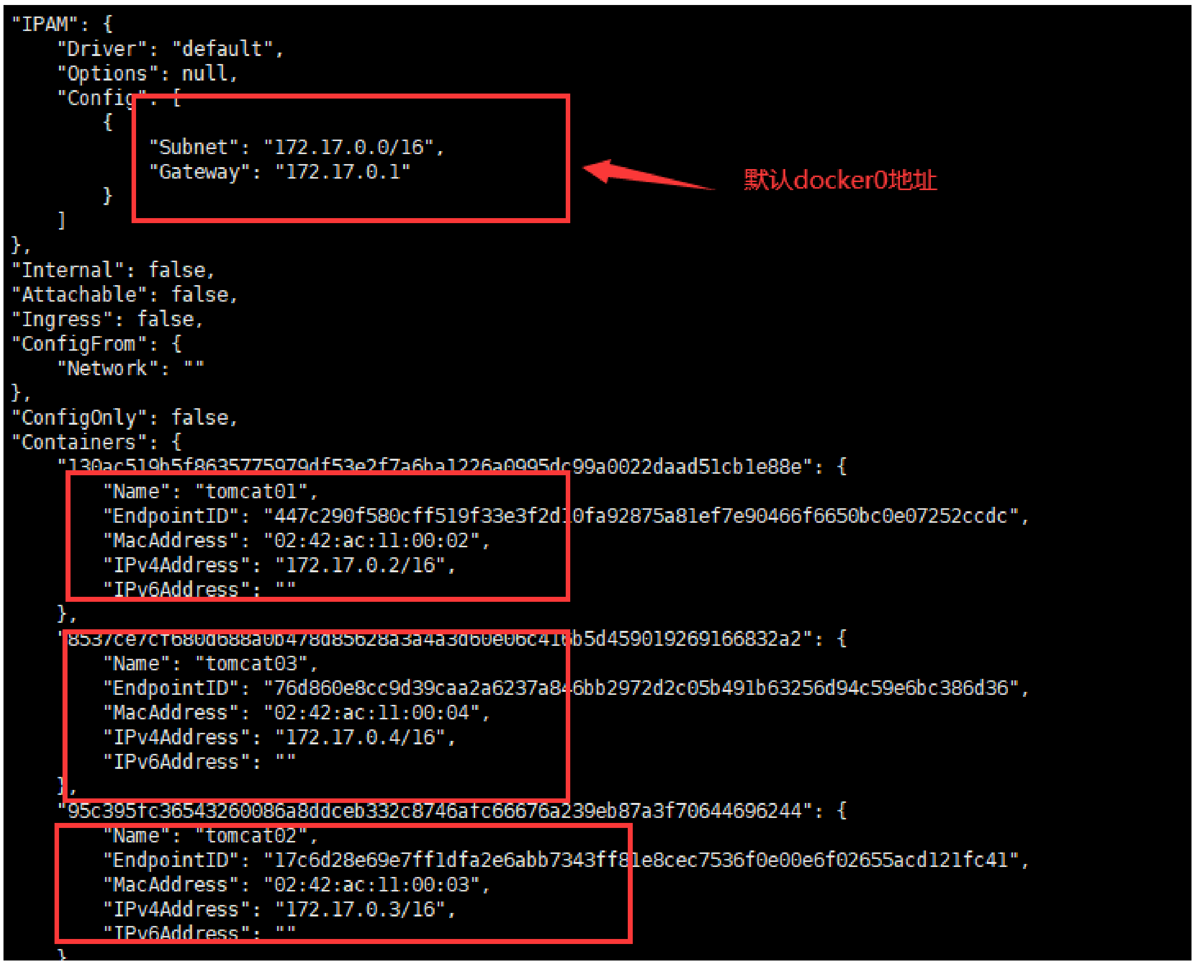
## --link

问题：一个微服务，重启时ip地址换掉了，但不希望整个项目重启，而是通过微服务名来进行访问。

```
#在tomcat01中通过tomcat02直接Ping
root@130ac519b5f8:/usr/local/tomcat# ping tomcat02
ping: tomcat02: Name or service not known
```

```
#解决，启动容器时添加需要link的容器名
root@youxin-virtual-machine:/home/youxin# docker run -d -p 8082:8080 --name 
tomcat03 --link tomcat01 tomcat:9.0

#测试：通过tomcat03连接tomcat01，此时tomcat03能够直接通过容器名来ping通tomcat01
root@8537ce7cf680:/usr/local/tomcat# ping tomcat01
PING tomcat01 (172.17.0.2) 56(84) bytes of data.
64 bytes from tomcat01 (172.17.0.2): icmp_seq=1 ttl=64 time=0.204 ms
64 bytes from tomcat01 (172.17.0.2): icmp_seq=2 ttl=64 time=0.190 ms
64 bytes from tomcat01 (172.17.0.2): icmp_seq=3 ttl=64 time=0.087 ms
64 bytes from tomcat01 (172.17.0.2): icmp_seq=4 ttl=64 time=0.072 ms
#相反通过tomcat01是无法直接通过tomcat03来ping通tomcat03的
root@youxin-virtual-machine:/home/youxin# docker exec -it tomcat01 ping tomcat03
ping: tomcat03: Name or service not known
```

通过docker network inspect 查看桥接网络具体信息



```
"IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
        {
            "Subnet": "172.17.0.0/16",
            "Gateway": "172.17.0.1"
        }
    ]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {
    "130ac519b5f8635775979df53e2f7a6ba1226a0995dc99a0022daad51cb1e88e": {
        "Name": "tomcat01",
        "EndpointID": "447c290f580cff519f33e3f2d10fa92875a81ef7e90466f6650bc0e07252ccdc",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    },
    "8537ce7cf680d688a0b478d85628a3a4a3d60e06c416b5d459019269166832a2": {
        "Name": "tomcat03",
        "EndpointID": "76d860e8cc9d39caa2a6237a846bb2972d2c05b491b63256d94c59e6bc386d36",
        "MacAddress": "02:42:ac:11:00:04",
        "IPv4Address": "172.17.0.4/16",
        "IPv6Address": ""
    },
    "95c395fc36543260086a8ddceb332c8746afc66676a239eb87a3f70644696244": {
        "Name": "tomcat02",
        "EndpointID": "17c6d28e69e7ff1dfa2e6abb7343ff81e8cec7536f0e00e6f02655acd121fc41",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
    }
```

查看tomcat03容器的/etc/hosts文件：发现tomcat01的地址直接被写入到了hosts中

```
root@8537ce7cf680:/usr/local/tomcat# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  tomcat01 130ac519b5f8
172.17.0.4  8537ce7cf680
```

--link本质上就是修改host配置，真实开发已经不建议使用--link了，他不支持容器名连接访问。

## 自定义网络

**网络模式**

bridge： 桥接docker（默认），自己创建也是使用bridge模式

none： 不配置网络

host： 和宿主机共享网络

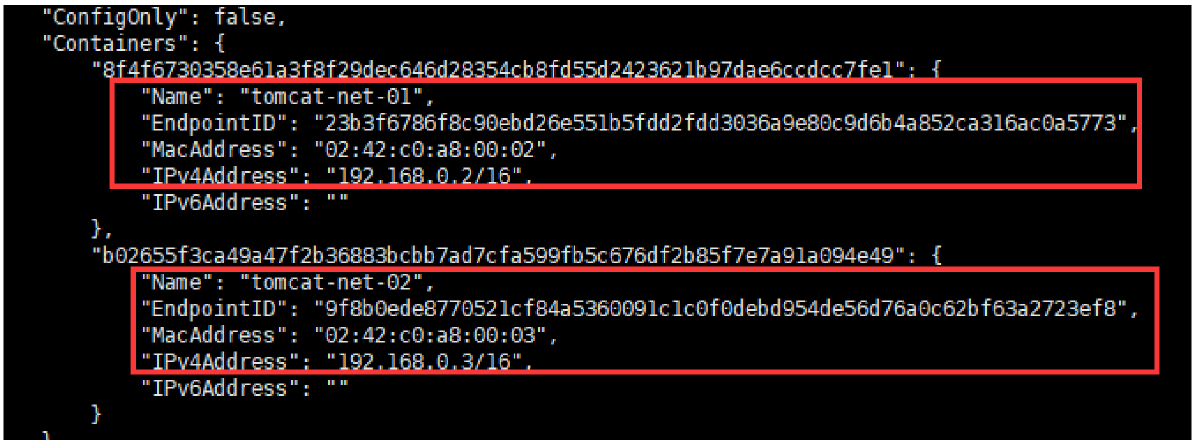
container: 容器网络连通（局限很大）

**测试**

```
#直接启动的命令，--net bridge 是默认的
root@youxin-virtual-machine:/home/youxin# docker run -d -p 8080:8080 --name
tomcat01 tomcat:9.0

root@youxin-virtual-machine:/home/youxin# docker run -d -p 8080:8080 --name
tomcat01 --net bridge tomcat:9.0

#docker0的特点，默认，域名不能访问，--link可以连接

#命令docker network
root@youxin-virtual-machine:/home/youxin# docker network create --help

Usage:  docker network create [OPTIONS] NETWORK

Create a network

Options:
      --attachable           Enable manual container attachment
      --aux-address map      Auxiliary IPv4 or IPv6 addresses used by Network
driver (default map[])
      --config-from string   The network from which to copy the configuration
      --config-only          Create a configuration only network
  -d, --driver string        Driver to manage the Network (default "bridge") #默
认为bridge
      --gateway strings      IPv4 or IPv6 Gateway for the master subnet  #网关，如
192.168.0.1
      --ingress              Create swarm routing-mesh network
      --internal             Restrict external access to the network
      --ip-range strings     Allocate container ip from a sub-range
      --ipam-driver string   IP Address Management Driver (default "default")
      --ipam-opt map         Set IPAM driver specific options (default map[])
      --ipv6                 Enable IPv6 networking
      --label list           Set metadata on a network
  -o, --opt map              Set driver specific options (default map[])
      --scope string         Control the network's scope
      --subnet strings       Subnet in CIDR format that represents a network
segment #子网掩码，如192.168.0.0/16

#自定义一个网络
root@youxin-virtual-machine:/home/youxin# docker network create --driver bridge
--subnet 192.168.0.1/16 --gateway 192.168.0.1 mynet
```

```
root@youxin-virtual-machine:/home/youxin# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
946155ba8b66   bridge    bridge    local
54848616ca5d   host      host      local
873261a72553   mynet     bridge    local
0ddc7d444b33   none      null      local

#启动容器包含在自己创建的网络中
root@youxin-virtual-machine:/home/youxin# docker run -d -p 8080:8080 --name
tomcat-net-01 --net mynet tomcat:9.0
8f4f6730358e61a3f8f29dec646d28354cb8fd55d2423621b97dae6ccdcc7fe1
root@youxin-virtual-machine:/home/youxin# docker run -d -p 8081:8080 --name
tomcat-net-02 --net mynet tomcat:9.0
b02655f3ca49a47f2b36883bcbb7ad7cfa599fb5c676df2b85f7e7a91a094e49
```

```
#使用docker network inspect 查看自己配置的网络
```



```
 "ConfigOnly": false,
 "Containers": {
     "8f4f6730358e61a3f8f29dec646d28354cb8fd55d2423621b97dae6ccdcc7fe1": {
         "Name": "tomcat-net-01",
         "EndpointID": "23b3f6786f8c90ebd26e551b5fdd2fdd3036a9e80c9d6b4a852ca316ac0a5773",
         "MacAddress": "02:42:c0:a8:00:02",
         "IPv4Address": "192.168.0.2/16",
         "IPv6Address": ""
     },
     "b02655f3ca49a47f2b36883bcbb7ad7cfa599fb5c676df2b85f7e7a91a094e49": {
         "Name": "tomcat-net-02",
         "EndpointID": "9f8b0ede8770521cf84a5360091c1c0f0debd954de56d76a0c62bf63a2723ef8",
         "MacAddress": "02:42:c0:a8:00:03",
         "IPv4Address": "192.168.0.3/16",
         "IPv6Address": ""
     }
```

```
#此时在tomcat-net-01中直接ping tomcat-net-02可以直接Ping通
root@dcd1cd88c137:/usr/local/tomcat# ping tomcat-net-02
PING tomcat-net-02 (192.168.0.3) 56(84) bytes of data.
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=1 ttl=64 time=0.154 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from tomcat-net-02.mynet (192.168.0.3): icmp_seq=3 ttl=64 time=0.172 ms
```

好处：不同的集群使用不同的网络，保证集群是安全和健康的。

## 网络连通

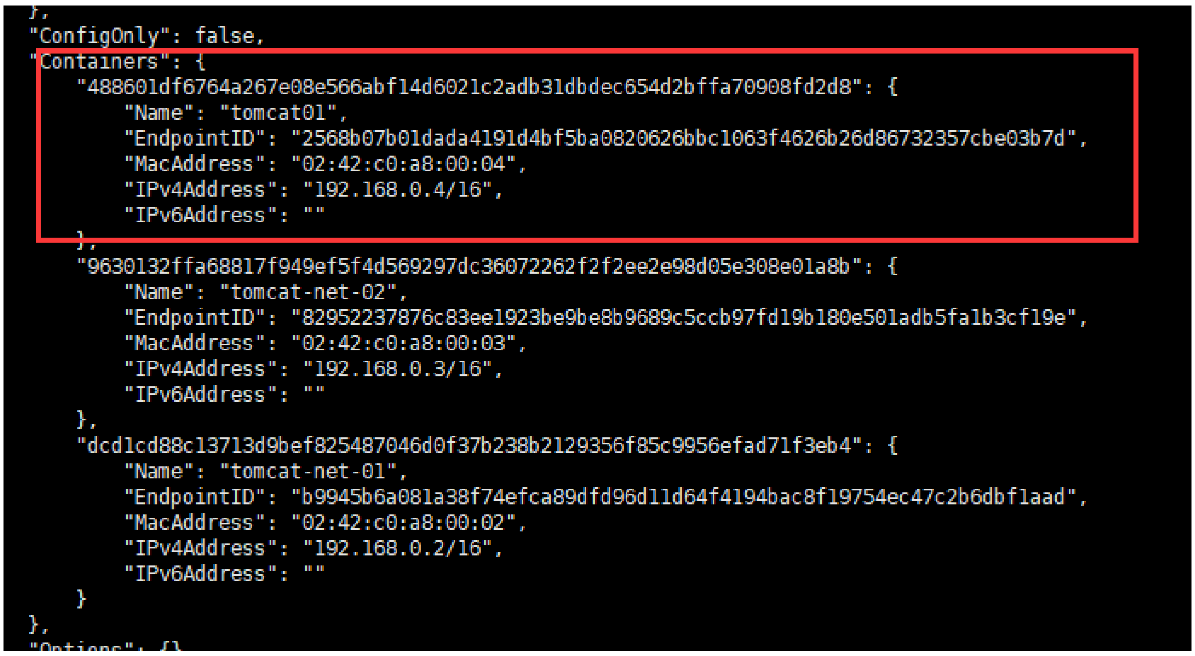
问题：默认的net连接中的容器无法连接上自定义的mynet网络中的容器。

```
root@youxin-virtual-machine:/home/youxin# docker exec -it tomcat01 ping tomcat-
net-01
ping: tomcat-net-01: Name or service not known
```

解决：使用`docker network connect`将容器连接进另一个网络中

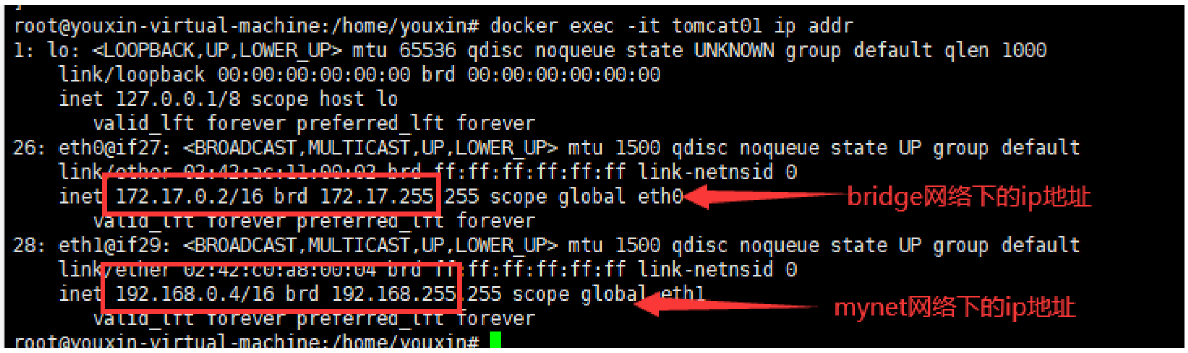
```
root@youxin-virtual-machine:/home/youxin# docker network connect --help

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


#连接
root@youxin-virtual-machine:/home/youxin# docker network connect mynet tomcat01

#将默认网络中的tomcat01连接至mynet下后，tomcat01就可以访问到自定义网络mynet的容器了
root@youxin-virtual-machine:/home/youxin# docker exec -it tomcat01 ping tomcat-net-01
PING tomcat-net-01 (192.168.0.2) 56(84) bytes of data.
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=1 ttl=64 time=0.173 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=3 ttl=64 time=0.171 ms
64 bytes from tomcat-net-01.mynet (192.168.0.2): icmp_seq=4 ttl=64 time=0.062 ms

#此时查看mynet inspect，发现将tomcat01添加到mynet的容器中了
```



```
},
"ConfigOnly": false,
"Containers": {
    "488601df6764a267e08e566abf14d6021c2adb31dbdec654d2bffa70908fd2d8": {
        "Name": "tomcat01",
        "EndpointID": "2568b07b01dada4191d4bf5ba0820626bbc1063f4626b26d86732357cbe03b7d",
        "MacAddress": "02:42:c0:a8:00:04",
        "IPv4Address": "192.168.0.4/16",
        "IPv6Address": ""
    },
    "9630132ffa68817f949ef5f4d569297dc36072262f2f2ee2e98d05e308e01a8b": {
        "Name": "tomcat-net-02",
        "EndpointID": "82952237876c83ee1923be9be8b9689c5ccb97fd19b180e501adb5fa1b3cf19e",
        "MacAddress": "02:42:c0:a8:00:03",
        "IPv4Address": "192.168.0.3/16",
        "IPv6Address": ""
    },
    "dcd1cd88c13713d9bef825487046d0f37b238b2129356f85c9956efad71f3eb4": {
        "Name": "tomcat-net-01",
        "EndpointID": "b9945b6a081a38f74efca89dfd96d11d64f4194bac8f19754ec47c2b6dbf1aad",
        "MacAddress": "02:42:c0:a8:00:02",
        "IPv4Address": "192.168.0.2/16",
        "IPv6Address": ""
    }
},
"Options": {},
```

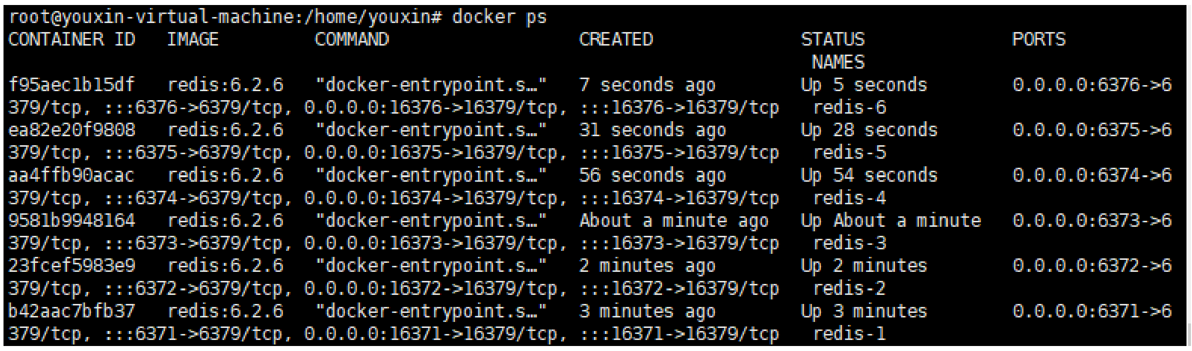
```
#一个容器两个ip地址，此时tomcat01有两个ip地址
```



```
root@youxin-virtual-machine:/home/youxin# docker exec -it tomcat01 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
26: eth0@if27: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
28: eth1@if29: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:a8:00:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.4/16 brd 192.168.255.255 scope global eth1
       valid_lft forever preferred_lft forever
root@youxin-virtual-machine:/home/youxin#
```

**实战：redis集群部署**

```
#创建redis网络
root@youxin-virtual-machine:/home/youxin# docker network create redisnet --
subnet 172.58.0.0/16 --gateway 172.58.0.1

#编写6个配置文件
for port in $(seq 1 6); \
do \
mkdir -p /home/youxin/redis/node-${port}/conf
touch /home/youxin/redis/node-${port}/conf/redis.conf
cat  << EOF >> /home/youxin/redis/node-${port}/conf/redis.conf
port 6379
bind 0.0.0.0
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 172.58.0.1${port}
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
EOF
done

#启动redis
docker run -p 6371:6379 -p 16371:16379 --name redis-1 \
    -v /home/youxin/redis/node-1/data:/data \
    -v /home/youxin/redis/node-1/conf/redis.conf:/etc/redis/redis.conf \
    -d --net redisnet --ip 172.58.0.11 redis:6.2.6 redis-server
/etc/redis/redis.conf
```



```
root@youxin-virtual-machine:/home/youxin# docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED              STATUS               PORTS
                                                                              NAMES
f95aec1b15df   redis:6.2.6   "docker-entrypoint.s…"   7 seconds ago        Up 5 seconds         0.0.0.0:6376->6
379/tcp, :::6376->6379/tcp, 0.0.0.0:16376->16379/tcp, :::16376->16379/tcp   redis-6
ea82e20f9808   redis:6.2.6   "docker-entrypoint.s…"   31 seconds ago       Up 28 seconds        0.0.0.0:6375->6
379/tcp, :::6375->6379/tcp, 0.0.0.0:16375->16379/tcp, :::16375->16379/tcp   redis-5
aa4ffb90acac   redis:6.2.6   "docker-entrypoint.s…"   56 seconds ago       Up 54 seconds        0.0.0.0:6374->6
379/tcp, :::6374->6379/tcp, 0.0.0.0:16374->16379/tcp, :::16374->16379/tcp   redis-4
9581b9948164   redis:6.2.6   "docker-entrypoint.s…"   About a minute ago   Up About a minute    0.0.0.0:6373->6
379/tcp, :::6373->6379/tcp, 0.0.0.0:16373->16379/tcp, :::16373->16379/tcp   redis-3
23fcef5983e9   redis:6.2.6   "docker-entrypoint.s…"   2 minutes ago        Up 2 minutes         0.0.0.0:6372->6
379/tcp, :::6372->6379/tcp, 0.0.0.0:16372->16379/tcp, :::16372->16379/tcp   redis-2
b42aac7bfb37   redis:6.2.6   "docker-entrypoint.s…"   3 minutes ago        Up 3 minutes         0.0.0.0:6371->6
379/tcp, :::6371->6379/tcp, 0.0.0.0:16371->16379/tcp, :::16371->16379/tcp   redis-1
```

```
#进入redis-1容器
root@youxin-virtual-machine:/etc# docker exec -it redis-1 /bin/sh
#创建集群
# redis-cli --cluster create 172.58.0.11:6379 172.58.0.12:6379 172.58.0.13:6379
172.58.0.14:6379 172.58.0.15:6379 172.58.0.16:6379 --cluster-replicas 1

[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.

#进入集群
# redis-cli -c #不加-c是进入单机
127.0.0.1:6379> cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
```

```
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:1
cluster_stats_messages_ping_sent:155
cluster_stats_messages_pong_sent:151
cluster_stats_messages_sent:306
cluster_stats_messages_ping_received:146
cluster_stats_messages_pong_received:155
cluster_stats_messages_meet_received:5
cluster_stats_messages_received:306

#添加信息
127.0.0.1:6379> set name zhangsan
-> Redirected to slot [5798] located at 172.58.0.12:6379 #此时是2号redis处理的
OK

#停掉2号redis容器
root@youxin-virtual-machine:/home/youxin# docker stop 23fcef5983e9

#再次get name
127.0.0.1:6379> get name
-> Redirected to slot [5798] located at 172.58.0.15:6379 #此时从2号redis的从机5号机
上仍然拿到了value
"zhangsan"
```

## SpringBoot微服务打包Docker镜像

1、构建SpringBoot项目

2、打包应用

3、编写Dockerfile

```
FROM java:8

COPY *.jar /app.jar

CMD ["========server.port=8080========"]

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

4、构建镜像

5、发布运行

```
docker run -p 6379:6379 -p 16371:16379 --name redis \
    -v /home/youxin/alumni_management/redis/data:/data \
    -v
/home/youxin/alumni_management/redis/conf/redis.conf:/etc/redis/redis.conf \
    -d --net redisnet --ip 172.58.0.22 redis:6.2.6 redis-server
/etc/redis/redis.conf


docker run -d -p 3306:3306 -v
/home/youxin/alumni_management/mysql/conf:/etc/mysql/comf.d -v
/home/youxin/alumni_management/mysql/data:/var/lib/mysql --name="mysql01" -e
MYSQL_ROOT_PASSWORD=1234 --net redisnet --ip 172.58.0.21 mysql:5.7

docker run -d -p 8080:8080 -v
/home/youxin/alumni_management/files:/usr/local/alumni_management_files --name
alumni_management01 --net redisnet --ip 172.58.0.23 alumni_management:1.0.0

docker build -t alumni_management .
```