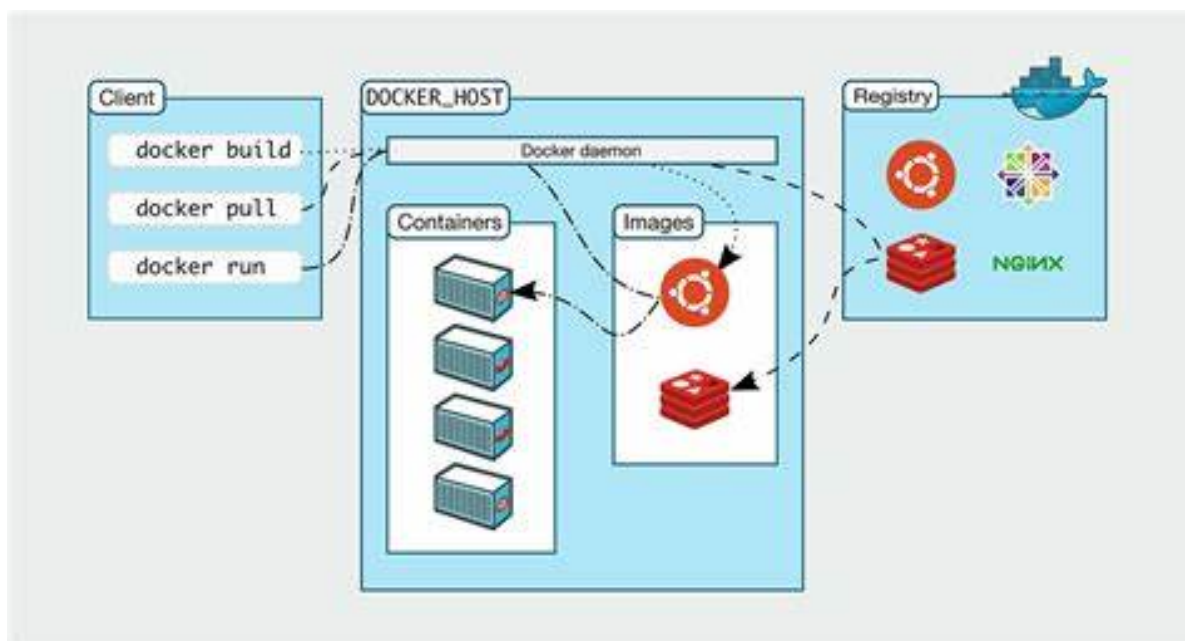


Docker 基本组成



镜像 (image) :

相当于一个模板，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），通过镜像可以创建多个容器。

服务运行在容器中运行

容器(container):

docker利用容器技术，独立运行一个或一组应用，通过镜像来创建的，

启动，停止，删除，基本命令

可以理解为建议的Linux系统

仓库(repository):

存放镜像的地方

分为公有仓库和私有仓库

需要配置镜像加速

安装docker

环境准备

1. 会linux系统
2. CentOS 7
3. 使用Xshell连接远程服务器（作为学习，我使用的是虚拟机里面安装centOS 7）

环境查看

1 | 系统内核是3.10以上的

安装时出现错误 如下图:

```

[docker01@docker01 ~]$ sudo yum install -y yum-utils \
> device-mapper-persistent-data \
> lvm2
[sudo] docker01 的密码:
docker01 不在 sudoers 文件中。此事将被报告。

```

- 1 解决办法:
- 2 1.先用su命令进入root用户下
- 3 2.输入visudo 进入配置文件
- 4 3.在配置文件中找到“rootALL = (ALL) ALL”行,在下面添加用户“ 用户名 ALL=(ALL) ALL ”
- 5 4.退出,重新输入命令即可

2、确定你是CentOS7及以上版本,我们已经做过了

I 3、yum安装gcc相关环境 (需要确保 虚拟机可以上外网)

```

1 yum -y install gcc
2 yum -y install gcc-c++

```

4、卸载旧版本

- 1 安装命令:
- 2 (卸载之前的版本):
- 3 `sudo yum remove docker \`
- 4 `docker-client \`
- 5 `docker-client-latest \`
- 6 `docker-common \`
- 7 `docker-latest \`
- 8 `docker-latest-logrotate \`
- 9 `docker-logrotate \`
- 10 `docker-engine`
- 11
- 12 1.需要的安装包
- 13 `sudo yum install -y yum-utils \`
- 14 `device-mapper-persistent-data \`
- 15 `lvm2`
- 16
- 17 2.设置镜像仓库
- 18 (国外的仓库,比较慢)
- 19 `sudo yum-config-manager \`
- 20 `--add-repo \`
- 21 `https://download.docker.com/linux/centos/docker-ce.repo`
- 22
- 23 (阿里云的镜像仓库,比较快,推荐使用)
- 24 `sudo yum-config-manager \`
- 25 `--add-repo \`
- 26 `http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo`
- 27
- 28 # 更新yum软件包的索引
- 29 `sudo yum makecache fast`
- 30
- 31 3.安装docker最新版本的
- 32
- 33 `sudo yum install docker-ce docker-ce-cli containerd.io`
- 34

```
35 | yum install -y docker-ce-20.10.7 docker-ce-cli-20.10.7 containerd.io-1.4.6
```

启动docker

- 1 | 4.运行 `sudo systemctl start docker`
- 2 | `systemctl enable d`
- 3 |
- 4 | 5.输入 `docker version` 判断docker 是不是启动成功

```
[root@docker01 docker01]# docker version
Client: Docker Engine - Community
Version: 20.10.11
API version: 1.41
Go version: go1.16.9
Git commit: dea9396
Built: Thu Nov 18 00:38:53 2021
OS/Arch: linux/amd64
Context: default
Experimental: true

Server: Docker Engine - Community
Engine:
Version: 20.10.11
API version: 1.41 (minimum version 1.12)
Go version: go1.16.9
Git commit: 847da18
Built: Thu Nov 18 00:37:17 2021
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.4.12
GitCommit: 7b11cfaabd73bb80907dd23182b9347b4245eb5d
runc:
Version: 1.0.2
GitCommit: v1.0.2-0-g52b36a2
docker-init:
Version: 0.19.0
GitCommit: de40ad0
[root@docker01 docker01]#
```

- 1 | 6.运行一个 `hello-world`程序
- 2 | `# docker run hello-world`

```
[root@docker01 docker01]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256: cc15c5b292d8525effc0f89cb299f1804f3a725c8d05e158653a563f15e4f685
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/>

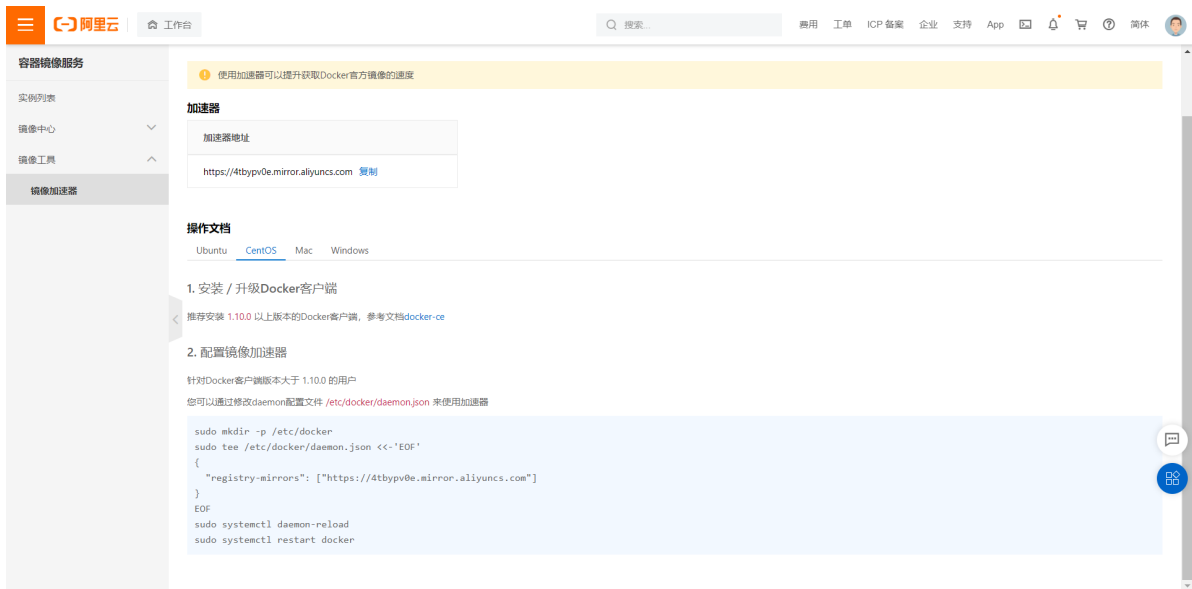
```
1 7.查看下载的 hello-world 镜像
2 输入的命令:
3 [root@docker01 docker01]# docker images
4
5 显示的结果:
6 REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
7 hello-world    latest    feb5d9fea6a5   2 months ago   13.3kB
8
```

docker的卸载

```
1 停止docker运行:
2 systemctl stop docker
3 删除安装包:
4 yum remove docker-ce docker-ce-cli containerd.io
5 删除镜像, 容器, 配置文件等内容:
6 rm -rf /var/lib/docker
```

阿里云镜像加速

- 1.登录阿里云找到容器服务, 找到镜像加速地址



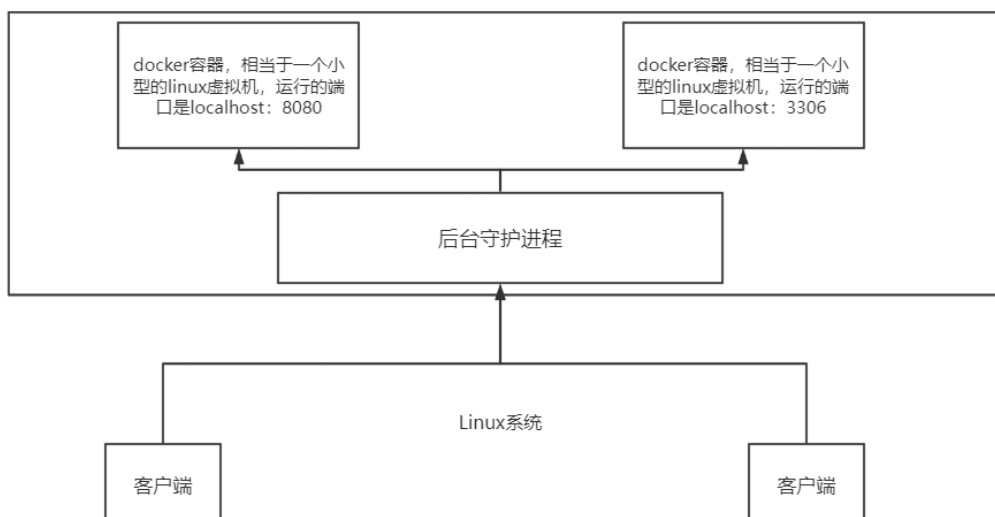
2.配置使用

```
1 1.sudo mkdir -p /etc/docker
2
3 2.sudo tee /etc/docker/daemon.json <<- 'EOF'
4 {
5     "registry-mirrors": ["https://4tbypv0e.mirror.aliyuncs.com"]
6 }
7 EOF
8
9 3.sudo systemctl daemon-reload
10
11 4.sudo systemctl restart docker
```

底层原理

Docker是一个Client-Server结构的系统，Docker的守护进程运行在主机上。通过Socket从客户端访问。

DockerServer接收到Docker-Client的指令，就会执行这个命令。



docker的常用命令

帮助命令

```
1 docker version    #显示docker的版本信息
2 docker info       #显示docker的系统信息，包括镜像和容器的数量
3 docker 命令 --help  #查询各种命令
```

docker命令帮助文档的地址：<https://docs.docker.com/reference/>

镜像命令

docker images 查看所有本地主机上的镜像

```
1 [root@docker01 docker01]# docker images
2 REPOSITORY      TAG          IMAGE ID      CREATED      SIZE
3 hello-world     latest      feb5d9fea6a5  2 months ago 13.3kB
4
5 # 解释
6 REPOSITORY      镜像的仓库源
7 TAG             镜像的标签
8 IMAGE ID        镜像的ID
9 CREATED         镜像的创建时间
10 SIZE           镜像的大小
11
12 # 可选项
13 -a, --all        列出所有镜像
14 -q, --quiet      只显示镜像的ID
15
```

docker search 搜索镜像命令

```
1 NAME                DESCRIPTION
2 mysql               MySQL is a widely used, open-source
3 relation... 11777 [OK]
4 mariadb             MariaDB Server is a high performing open
5 sou... 4487 [OK]
```

docker pull 下载镜像

```
1 docker pull mysql [:tag] 下载mysql镜像,如果不写tag,会默认下载最新版本
2 docker pull mysql:5.7    下载mysql5.7版本的镜像
```

docker rmi 删除镜像

```
1 docker rmi -f 镜像id           #删除指定镜像
2 docker rmi -f 镜像id 镜像id 镜像id   #删除多个镜像
3 docker rmi -f $(docker images -aq)    #删除全部镜像
```

容器命令

说明：有了镜像才能创建容器，下载一个Linux，centOS镜像来测试学习

```
1 docker pull centos
```

新建容器并启动

```
1 docker run [可选参数] image
2
3 # 参数说明
4 --name="Name"    容器的名字， tomcat01 tomcat02 ， 用来区分容器
5 -d              后台交互式运行
6 -it             使用交互方式运行，进入容器查看内容
7 -p              指定容器端口 -p 8080:8080
8     -p ip: 主机端口:容器端口
9     -p 主机端口:容器端口 （常用）
10    -p 容器端口
11    容器端口
12 -p              随机指定端口
13
14
15
16 # 测试，启动并进入容器
17 [root@docker01 docker01]# docker run -it centos /bin/bash
18 [root@870a4fc7d3c0 /]# ls #查看容器内的centos，基础版本
19 bin etc lib lost+found mnt proc run srv tmp var
20 dev home lib64 media opt root sbin sys usr
21
22
23 # 从容器中退出
24 [root@870a4fc7d3c0 /]# quit
25
```

列出所有运行过的容器

```
1 # docker ps 命令
2     #列出正在运行的容器o
3 -a    # 列出运行过的容器
4 -n=?  #显示最近创建的容器
5 -q    # 只显示容器的编号
6
7 #命令可以混合使用 docker ps -aq 显示所有运行过的容器的编号
8
9 [root@docker01 docker01]# docker ps # 列出正在运行的容器
10 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
11
```

```

12 [root@docker01 docker01]# docker ps -a # 列出运行过的容器
13 CONTAINER ID   IMAGE     COMMAND                  CREATED     STATUS
14 870a4fc7d3c0   centos    "/bin/bash"             3 minutes ago    Exited (130)
    About a minute ago    optimistic_kalam
15 e767aeb92078   hello-world "/hello"                7 minutes ago    Exited (0) 7
    minutes ago          keen_yonath
16 eaa74af3c185   hello-world "/hello"                2 hours ago      Exited (0) 2
    hours ago            cranky_goldwasser
17

```

退出容器

```

1 exit #直接停止容器退出
2 Ctrl + q + p #容器不停止但退出

```

删除容器

```

1 docker rm 容器id #删除指定的容器，不能删除正在运行的容器，强制删除 rm
  -f
2 docker rm -f $(docker ps -aq) #删除全部的容器
3 docker ps -a -q|xargs docker rm #删除全部的容器

```

启动和停止容器的操作

```

1 docker start 容器id #启动容器
2 docker restart 容器id #重新启动容器
3 docker stop 容器id #停止当前正在运行的容器
4 docker kill 容器id #强制停止容器

```

常用其他命令

后台启动容器

```

1 # docker run -d 镜像名
2
3 #问题docker ps ,发现 镜像停止了
4
5 #常见的坑: docker 容器在后端运行，就必须要有有一个前台的进程，就会自动停止
6 #nginx, 容器启动后，发现自己没有提供服务，就会立即停止，就是没有程序了

```

查看日志

```

1 # docker logs -t -f --tail=n -a(n代表查询日志的数量)
2 # docker logs -t -f --tail=n 容器id
3
4 #显示日志
5 -a # 查询所有容器的日志
6 -tf # 显示所有日志
7 --tail num # 要显示日志的条数
8

```


查看容器中的进程信息

```
1 | docker top 容器id
```

查看镜像的源数据

```
1 # 命令
2 docker inspect 容器id
3
4 # 测试
5 [root@docker01 docker01]# docker inspect 42f22ae12579
6 [
7     {
8         "Id":
9         "42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626",
10        "Created": "2021-12-04T09:54:04.229855998Z",
11        "Path": "/bin/bash",
12        "Args": [],
13        "State": {
14            "Status": "exited",
15            "Running": false,
16            "Paused": false,
17            "Restarting": false,
18            "OOMKilled": false,
19            "Dead": false,
20            "Pid": 0,
21            "ExitCode": 0,
22            "Error": "",
23            "StartedAt": "2021-12-04T09:54:04.56940188Z",
24            "FinishedAt": "2021-12-04T09:55:31.08471425Z"
25        },
26        "Image":
27        "sha256:5d0da3dc976460b72c77d94c8a1ad043720b0416bfc16c52c45d4847e53fadb6",
28        "ResolveConfPath":
29        "/var/lib/docker/containers/42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626/resolve.conf",
30        "HostnamePath":
31        "/var/lib/docker/containers/42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626/hostname",
32        "HostsPath":
33        "/var/lib/docker/containers/42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626/hosts",
34        "LogPath":
35        "/var/lib/docker/containers/42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626/42f22ae12579008a7c3d161273cce8bf4ad4746d9acd2e40923f2745d463a626-json.log",
36        "Name": "/hungry_noether",
37        "RestartCount": 0,
38        "Driver": "overlay2",
39        "Platform": "linux",
40        "MountLabel": "",
41        "ProcessLabel": "",
42        "AppArmorProfile": "",
43        "ExecIDs": null,
44        "HostConfig": {
45            "Binds": null,
46            "ContainerIDFile": "",
```

```
41     "LogConfig": {
42         "Type": "json-file",
43         "Config": {}
44     },
45     "NetworkMode": "default",
46     "PortBindings": {},
47     "RestartPolicy": {
48         "Name": "no",
49         "MaximumRetryCount": 0
50     },
51     "AutoRemove": false,
52     "VolumeDriver": "",
53     "VolumesFrom": null,
54     "CapAdd": null,
55     "CapDrop": null,
56     "CgroupnsMode": "host",
57     "Dns": [],
58     "DnsOptions": [],
59     "DnsSearch": [],
60     "ExtraHosts": null,
61     "GroupAdd": null,
62     "IpcMode": "private",
63     "Cgroup": "",
64     "Links": null,
65     "OomScoreAdj": 0,
66     "PidMode": "",
67     "Privileged": false,
68     "PublishAllPorts": false,
69     "ReadonlyRootfs": false,
70     "SecurityOpt": null,
71     "UTSMode": "",
72     "UsernsMode": "",
73     "ShmSize": 67108864,
74     "Runtime": "runc",
75     "ConsoleSize": [
76         0,
77         0
78     ],
79     "Isolation": "",
80     "CpuShares": 0,
81     "Memory": 0,
82     "NanoCpus": 0,
83     "CgroupParent": "",
84     "Blkioweight": 0,
85     "BlkioweightDevice": [],
86     "BlkiodeviceReadBps": null,
87     "BlkiodeviceWriteBps": null,
88     "BlkiodeviceReadIOps": null,
89     "BlkiodeviceWriteIOps": null,
90     "CpuPeriod": 0,
91     "CpuQuota": 0,
92     "CpuRealtimePeriod": 0,
93     "CpuRealtimeRuntime": 0,
94     "CpusetCpus": "",
95     "CpusetMems": "",
96     "Devices": [],
97     "DeviceCgroupRules": null,
98     "DeviceRequests": null,
```

```

99         "KernelMemory": 0,
100         "KernelMemoryTCP": 0,
101         "MemoryReservation": 0,
102         "MemorySwap": 0,
103         "MemorySwappiness": null,
104         "OomKillDisable": false,
105         "PidsLimit": null,
106         "Ulimits": null,
107         "CpuCount": 0,
108         "CpuPercent": 0,
109         "IOMaximumIOps": 0,
110         "IOMaximumBandwidth": 0,
111         "MaskedPaths": [
112             "/proc/asound",
113             "/proc/acpi",
114             "/proc/kcore",
115             "/proc/keys",
116             "/proc/latency_stats",
117             "/proc/timer_list",
118             "/proc/timer_stats",
119             "/proc/sched_debug",
120             "/proc/scsi",
121             "/sys/firmware"
122         ],
123         "ReadonlyPaths": [
124             "/proc/bus",
125             "/proc/fs",
126             "/proc/irq",
127             "/proc/sys",
128             "/proc/sysrq-trigger"
129         ]
130     },
131     "GraphDriver": {
132         "Data": {
133             "LowerDir":
134                 "/var/lib/docker/overlay2/31be9ff3ac2da6207f80d25f317db896cfc693020a84eb098
135                 efb024c8e0df2e4-
136                 init/diff:/var/lib/docker/overlay2/c34b4aa3040fde719ffc648c55ec1bb8f4ba831f
137                 980fcbb9b37a30051b4d5910/diff",
138             "MergedDir":
139                 "/var/lib/docker/overlay2/31be9ff3ac2da6207f80d25f317db896cfc693020a84eb098
140                 efb024c8e0df2e4/merged",
141             "UpperDir":
142                 "/var/lib/docker/overlay2/31be9ff3ac2da6207f80d25f317db896cfc693020a84eb098
143                 efb024c8e0df2e4/diff",
144             "WorkDir":
145                 "/var/lib/docker/overlay2/31be9ff3ac2da6207f80d25f317db896cfc693020a84eb098
146                 efb024c8e0df2e4/work"
147         },
148         "Name": "overlay2"
149     },
150     "Mounts": [],
151     "Config": {
152         "Hostname": "42f22ae12579",
153         "Domainname": "",
154         "User": "",
155         "AttachStdin": true,
156         "AttachStdout": true,

```

```
147         "AttachStderr": true,
148         "Tty": true,
149         "OpenStdin": true,
150         "StdinOnce": true,
151         "Env": [
152
153             "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
154         ],
155         "Cmd": [
156             "/bin/bash"
157         ],
158         "Image": "centos",
159         "Volumes": null,
160         "WorkingDir": "",
161         "Entrypoint": null,
162         "OnBuild": null,
163         "Labels": {
164             "org.label-schema.build-date": "20210915",
165             "org.label-schema.license": "GPLv2",
166             "org.label-schema.name": "CentOS Base Image",
167             "org.label-schema.schema-version": "1.0",
168             "org.label-schema.vendor": "CentOS"
169         },
170         "NetworkSettings": {
171             "Bridge": "",
172             "SandboxID":
173             "e4fd3d35da7d401a907c0fb46b987679106b02613ed4bcc3b83fbc2ddbccff72",
174             "HairpinMode": false,
175             "LinkLocalIPv6Address": "",
176             "LinkLocalIPv6PrefixLen": 0,
177             "Ports": {},
178             "SandboxKey": "/var/run/docker/netns/e4fd3d35da7d",
179             "SecondaryIPAddresses": null,
180             "SecondaryIPv6Addresses": null,
181             "EndpointID": "",
182             "Gateway": "",
183             "GlobalIPv6Address": "",
184             "GlobalIPv6PrefixLen": 0,
185             "IPAddress": "",
186             "IPPrefixLen": 0,
187             "IPv6Gateway": "",
188             "MacAddress": "",
189             "Networks": {
190                 "bridge": {
191                     "IPAMConfig": null,
192                     "Links": null,
193                     "Aliases": null,
194                     "NetworkID":
195                     "a52d4e4bfff610bdcba9763774038725794ae4910aeac74f5fe3b7587d2509f3",
196                     "EndpointID": "",
197                     "Gateway": "",
198                     "IPAddress": "",
199                     "IPPrefixLen": 0,
200                     "IPv6Gateway": "",
201                     "GlobalIPv6Address": "",
202                     "GlobalIPv6PrefixLen": 0,
203                     "MacAddress": "",
```

```

202         "DriverOpts": null
203     }
204 }
205 }
206 }
207 ]

```

进入正在运行的容器

```

1  # 容器通常都是使用后台运行的，需要进入容器，修改一些配置
2
3  # 命令
4  docker exec -it 容器id bashShell
5
6  # 测试
7  [root@docker01 docker01]# docker ps
8  CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
9  1c1e914eb3cb   centos    "/bin/bash"             12 seconds ago Up 11 seconds
10                  romantic_bartik
11 [root@docker01 docker01]# docker exec -it 1c1e914eb3cb /bin/bash
12 [root@1c1e914eb3cb /]# ps -ef
13 UID            PID     PPID  C  STIME TTY          TIME CMD
14 root             1         0  0  10:00 pts/0        00:00:00 /bin/bash
15 root            15         0  0  10:01 pts/1        00:00:00 /bin/bash
16 root            29        15  0  10:02 pts/1        00:00:00 ps -ef
17
18 # 方式二
19 docker attach 容器id
20
21 # 测试
22 [root@docker01 docker01]# docker attach 1c1e914eb3cb
23 正在执行当前的代码....
24
25 # docker exec #进入一个新的终端，可以在里面操作（常用）
26 # docker attach # 进入容器中正在执行的终端，不会启动新的进程

```

从容器内拷贝文件到主机

```

1  docker cp 容器id:容器内路径 目的主机路径
2
3  # 查看当前主机目录
4  [root@docker01 home]# ls
5  docker01  likunsong.java
6  [root@docker01 home]# docker ps
7  CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
8  620084e2d74d   centos    "/bin/bash"             3 minutes ago Up 3 minutes
9                  elated_antone11i
10
11 # 进入docker 内部
12 [root@docker01 home]# docker attach 20084e2d74d
13 Error: No such container: 20084e2d74d
14 [root@docker01 home]# docker attach 620084e2d74d
15 [root@620084e2d74d /]# cd /home
16 [root@620084e2d74d home]# ls
17 # 在容器内新建一个文件

```

```

17 [root@620084e2d74d home]# touch text.java
18 [root@620084e2d74d home]# exit
19 exit
20 [root@docker01 home]# docker ps
21 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
22 [root@docker01 home]# docker ps -a
23 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
24 620084e2d74d   centos    "/bin/bash"   5 minutes ago   Exited (0) 22 seconds ago   elated_antone11i
25
26 # 将容器内的文件拷贝到主机上
27 [root@docker01 home]# docker cp 620084e2d74d:/home/text.java /home
28 [root@docker01 home]# ls
29 docker01  likunsong.java  text.java
30

```

练习

Docker 安装 Nginx

```

1  # 1. 搜索镜像, search
2  # 2. 下载镜像 pull
3  # 3. 运行测试
4
5  # -d 后台运行
6  # -- name 给容器命名
7  # -p 宿主机端口, 容器内部端口
8  [root@iz0jl26t6u41561b52dskvz ~]# docker images
9  REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
10 nginx          latest    f652ca386ed1   2 days ago     141MB
11
12 [root@iz0jl26t6u41561b52dskvz ~]# docker ps
13 CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
14 52cc76a3f6e5   nginx     "/docker-entrypoint..."   14 seconds ago   Up 14 seconds   0.0.0.0:3344->80/tcp   nginx01
15
16 [root@iz0jl26t6u41561b52dskvz ~]# curl localhost:3344
17 <!DOCTYPE html>
18 <html>
19 <head>
20 <title>welcome to nginx!</title>
21 <style>
22 html { color-scheme: light dark; }
23 body { width: 35em; margin: 0 auto;
24 font-family: Tahoma, Verdana, Arial, sans-serif; }
25 </style>
26 </head>
27 <body>
28 <h1>welcome to nginx!</h1>
29 <p>If you see this page, the nginx web server is successfully installed and
30 working. Further configuration is required.</p>
31
32 <p>For online documentation and support please refer to

```

```

33 <a href="http://nginx.org/">nginx.org</a>.<br/>
34 Commercial support is available at
35 <a href="http://nginx.com/">nginx.com</a>.</p>
36
37 <p><em>Thank you for using nginx.</em></p>
38 </body>
39 </html>
40
41 # 进入容器
42
43 [root@iz0jl26t6u41561b52dskvz ~]# docker ps
44 CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
45 52cc76a3f6e5   nginx     "/docker-entrypoint...." 12 minutes ago Up 12
46 minutes       0.0.0.0:3344->80/tcp   nginx01
47 [root@iz0jl26t6u41561b52dskvz ~]# docker exec -it nginx01 /bin/bash
48 root@52cc76a3f6e5:/# whereis nginx
49 nginx: /usr/sbin/nginx /usr/lib/nginx /etc/nginx /usr/share/nginx

```

docker 安装 tomcat

```

1 # 官方的使用
2 docker run -it --rm tomcat:9.0
3 （一般用于测试，用完自动删除）
4
5 # 下载再启动
6 docker pull tomcat
7
8 # 启动
9 docker run -d -p 3355:8080 --name tomcat01 tomcat
10
11 # 测试访问，没有问题
12
13 #进入容器
14 docker exec -it tocat01 /bin/bash
15

```

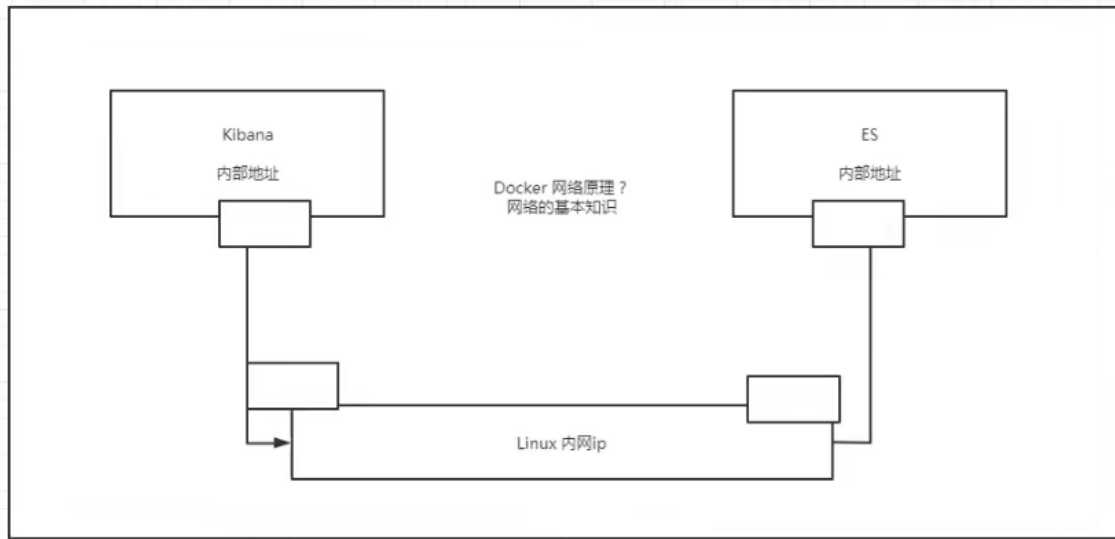
部署 es + kibana

```

1 # es 暴露的端口很多
2 # es 十分耗内存
3 # es 的数据一般需要放置在安全目录 挂载
4
5 # --net somenetwork 网络配置
6
7
8 # 启动 elasticsearch
9 $ docker run -d --name elasticsearch --net somenetwork -p 9200:9200 -p
10 9300:9300 -e "discovery.type=single-node" elasticsearch:tag
11
12 # 启动之后 Linux服务器就会卡 docker stats 查看cpu的状态
13
14 # es 是十分耗内存的

```

```
15 $ docker run -d --name elasticsearch --net somenetwork -p 9200:9200 -p
    9300:9300 -e "discovery.type=single-node" -e ES_JAVA_OPTS="-Xms64m _Xmx512m"
    elasticsearch:tag
16
17 # -e ES_JAVA_OPTS="-Xms64m _Xmx512m" 设置配置文件，为了不让服务器被占用内存太大
```



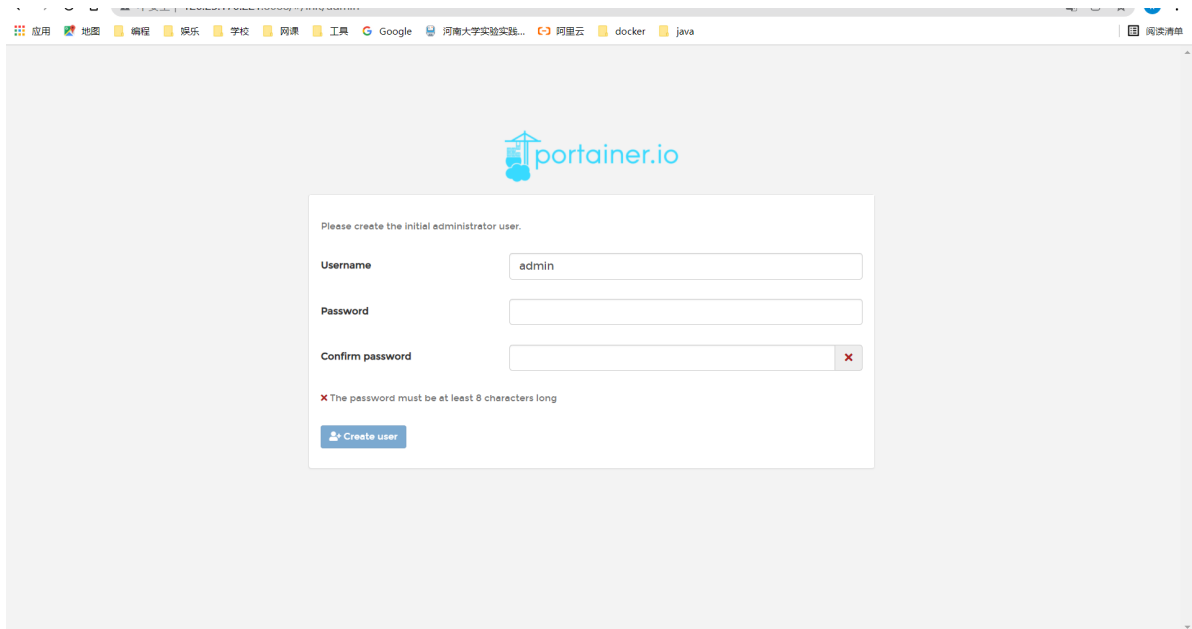
可视化

- portainer (不是最佳选择)
- Rancher (CI/CD再用)
Docker是图形化界面管理工具

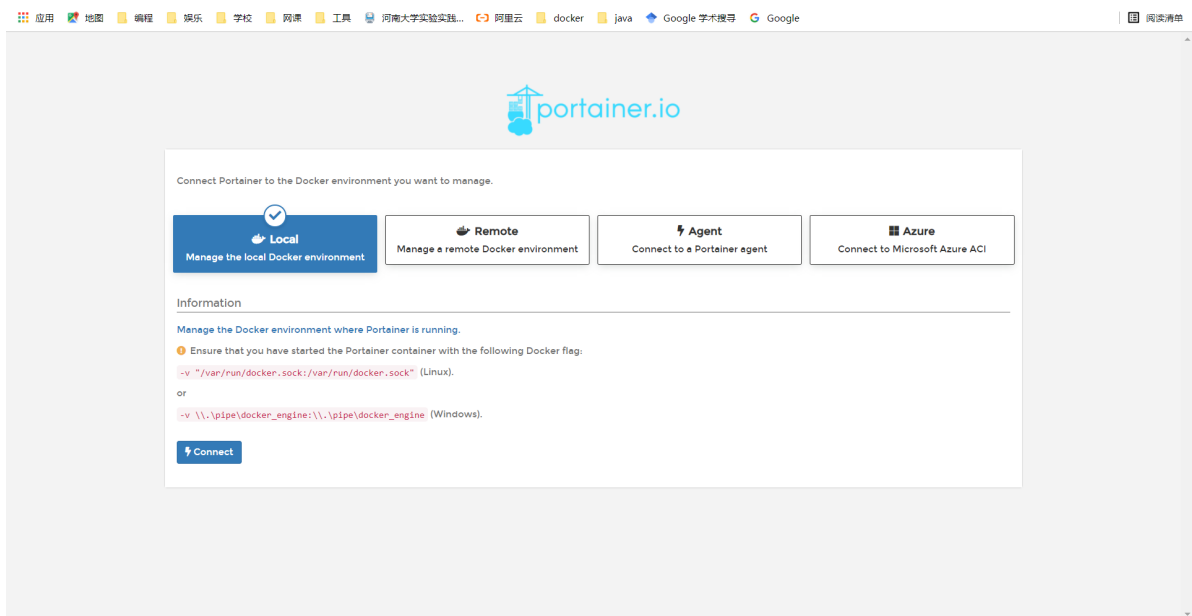
```
1 docker run -d -p 8088:9000 \  
2 --restart=always -v /var/run/docker.sock:/var/run/docker.sock --  
  privileged=true portainer/portainer  
3
```

访问测试：自己外网端口:8088/

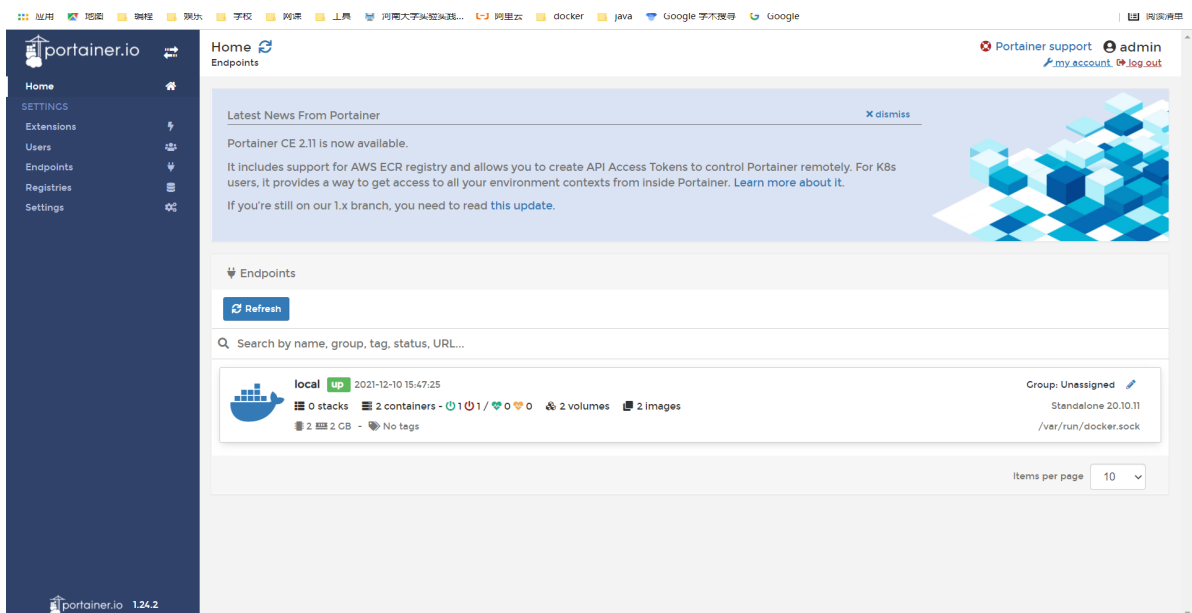
通过它来访问（访问界面）：（账号： admin 密码： 12345678）



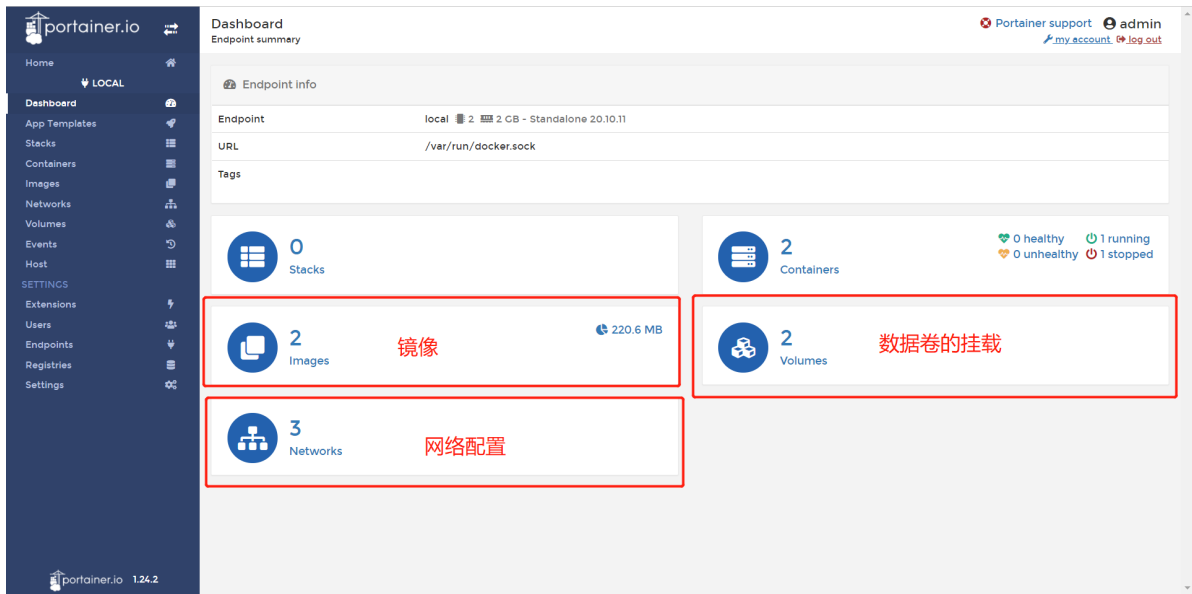
注册完密码后，选择本地的



进去之后会展示这样的一个面板



点开



可视化一般不会用

Docker镜像

镜像是什么

所有的应用，直接打包docker镜像，就可以直接跑起来

如何得到镜像：

- 从远程仓库下载
- 朋友 拷贝
- 自己制作一个镜像 DockerFile

镜像加载原理

UnionFS（联合文件系统）

联合文件系统（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 virtual filesystem）。联合文件系统是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，但从外面只能看见一个文件，联合加载会把各个文件系统叠加起来，这样最终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UnionFS（联合文件系统）。

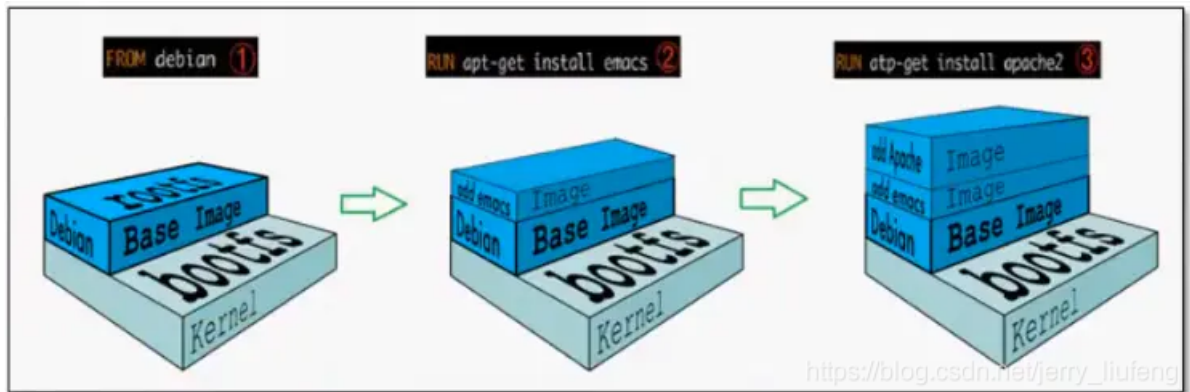
分为两个部分：

- bootfs（boot file system）：主要包含bootloader和kernel（Linux内核），bootloader主要是引导加载kernel，Linux刚启动时会加载bootfs文件系统，而在Docker镜像的最底层也是bootfs这一层，这与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。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。

即：系统启动时需要的引导加载，这个过程会需要一定时间。就是黑屏到开机之间的这么一个过程。电脑、虚拟机、Docker容器启动都需要的过程。在说回镜像，所以这一部分，无论是什么镜像都是公用的。

- rootfs (root file system)：rootfs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。rootfs就是各种不同的操作系统发行版，比如Ubuntu, Centos等等。

即：镜像启动之后的一个小的底层系统，这就是我们之前所说的，容器就是一个小的虚拟机环境，比如Ubuntu, Centos等，这个小的虚拟机环境就相当于rootfs。



对于一个精简的OS系统，rootfs可以很小，只需要包含最基本的命令、工具和程序库就可以了，因为底层直接用Host（宿主机）的kernel（也就是宿主机或者服务器的bootfs+内核），自己只需要提供rootfs就可以了。

由此可见对于不同的linux发行版，bootfs基本是一致的，rootfs会有差别，因此不同的发行版可以公用bootfs部分

这就是我们之前说：虚拟机的启动是分钟级的，容器的启动是秒级的**

分层理解

分层的镜像

```
[root@nanxing ~]# docker pull redis
Using default tag: latest
latest: Pulling from library/redis
e5ae68f74026: Already exists
37c4354629da: Pull complete
b065b1b1fa0f: Pull complete
6954d19bb2e5: Pull complete
6333f8baaf7c: Pull complete
f9772c8a44e7: Pull complete
Digest: sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d4076b796
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
[root@nanxing ~]#
```

是一层一层的安装的

```
1 [root@nanxing ~]# docker image inspect redis:latest
2 [
3   {
4     "Id":
5     "sha256:aea9b698d7d1d2fb22fe74868e27e767334b2cc629a8c6f9db8cc1747ba299fd",
6     "RepoTags": [
7       "redis:latest"
8     ],
9     "RepoDigests": [
```

```

9      "redis@sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d4076b
10      ],
11      ...# 中间省略
12      },
13      "RootFS": {
14          "Type": "layers",
15          "Layers": [
16
17              "sha256:9321ff862abbe8e1532076e5fdc932371eff562334ac86984a836d77dfb717f5",
18
19              "sha256:aa2858ea5edc9c0981901a1b63b49a8f4a6e7099b4304b49e680ffdcc6b71b3e",
20
21              "sha256:93079bf13a6d5fe7c4bd9f00cb96183f9d1db9968c4bd15b395df2f3867bf8e5",
22
23              "sha256:9ca504b88e256aa6f6c04ec65aeed6b926661ea30a0b97f829fbe230155241a",
24
25              "sha256:9468a3f0498bd5cc298ce25ea6ce9c6adf14aa2ce152856b5f389510a9bb9e01",
26
27              "sha256:b7851a62867d82784052d7662862adc0b47b2bddcddc89ae78307f75ba1b29ae"
28          ]
29      },
30      "Metadata": {
31          "LastTagTime": "0001-01-01T00:00:00Z"
32      }
33  }
34  ]

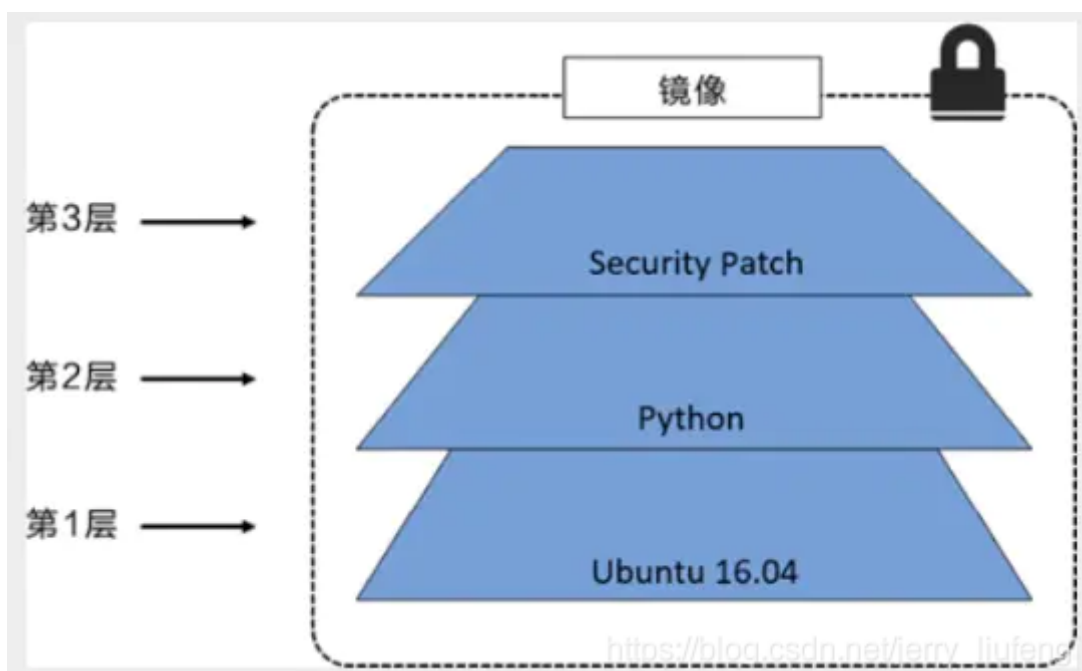
```

理解

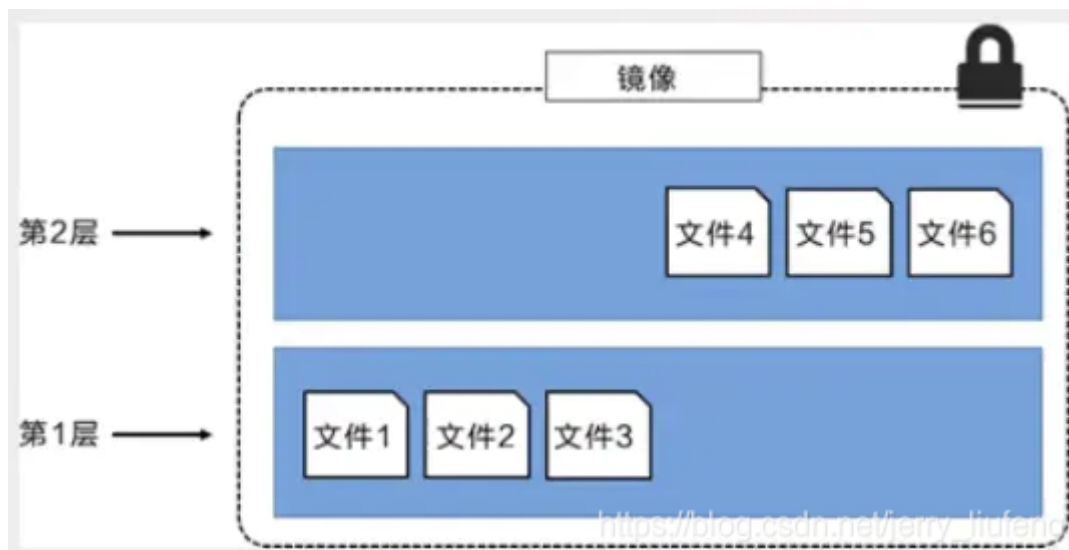
所有的Docker镜像都起始于一个基础镜像层，当进行修改或增加新的内容时，就会在当前镜像层之上，创建新的镜像层。

举一个简单的例子，假如基于Ubuntu Linux 16.04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，就会创建第三个镜像层。

该镜像当前已经包含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）。

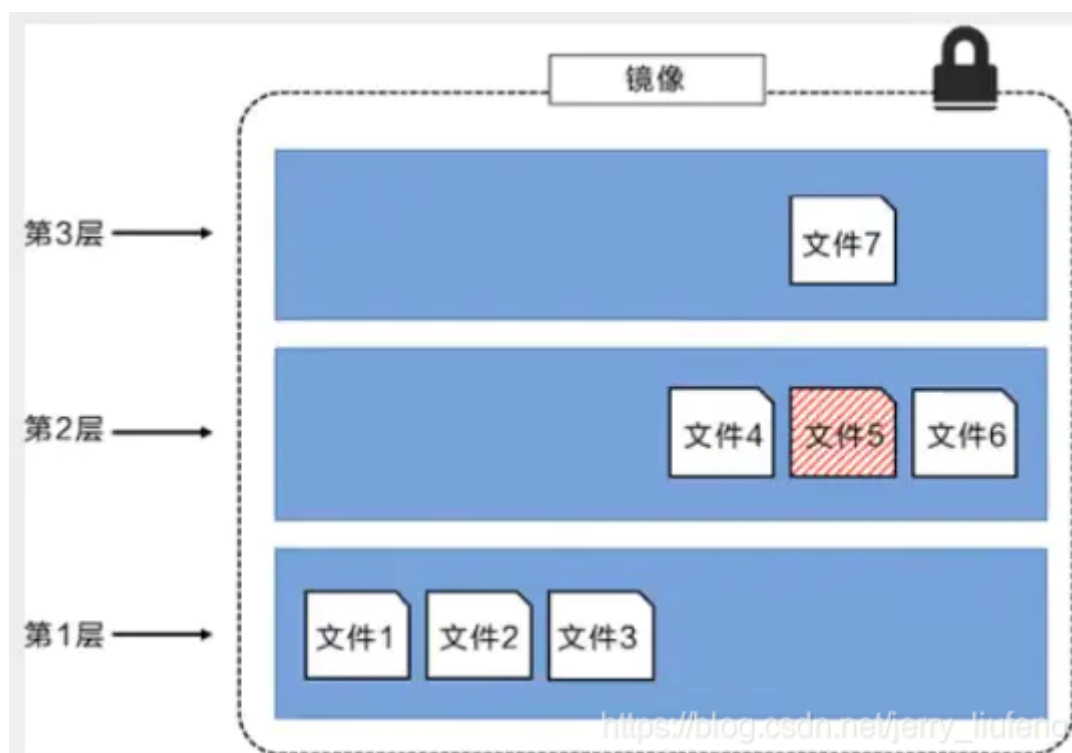


在添加额外的镜像层的同时，镜像始终保持是当前所有镜像的组合，理解这一点非常重要。下图中举了一个简单的例子，每个镜像层包含3个文件，而整体的镜像包含了来自两个镜像层的6个文件。



上图中的镜像层跟之前图中的略有区别，主要目的是便于展示文件。

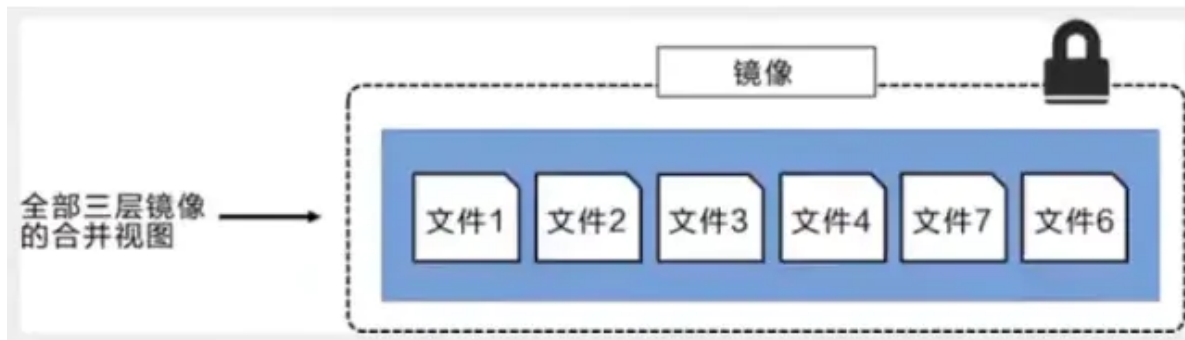
下图中展示了一个稍微复杂的三层镜像，在外部看来整个镜像只有6个文件，这是因为最上层中的文件7是文件5的一个更新版本。



这种情况下，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。

Docker通过存储引擎（新版本采用快照机制）的方式来实现镜像层堆栈，并保证多镜像层对外展示为统一的文件系统。

如上边的三层镜像，Docker最终会把所有镜像层堆叠并合并，对外提供统一的视图，如下图



特点

Docker镜像都是只读的，当容器启动时，一个新的可写层会被加载到镜像的顶部

这一层是我们所说的容器层，容器之下都叫镜像层！

commit镜像

```
1 docker commit 提交容器成为一个新副本
2
3 # 命令和git相似
4 docker commit -m="提交描述信息" -a="作者" 容器id 目标镜像名: [TAG]
```

测试

```
1 # 启动一个tomcat
2 docker run -it -p 8080:8080 tomcat
3 # 发现默认的tomcat 没有webapps应用 镜像的原因 官方镜像下默认tomcat是没有文件的
4
5 # 自己拷贝进去基本的文件
```

```
[root@nanxing ~]# docker exec -it 42348c61acc2 /bin/bash
root@42348c61acc2:/usr/local/tomcat# cd webapps
root@42348c61acc2:/usr/local/tomcat/webapps# ls
root@42348c61acc2:/usr/local/tomcat/webapps# cd ..
root@42348c61acc2:/usr/local/tomcat# cp -r webapps.dist/* webapps
root@42348c61acc2:/usr/local/tomcat# cd webapps
root@42348c61acc2:/usr/local/tomcat/webapps# ls
ROOT docs examples host-manager manager
root@42348c61acc2:/usr/local/tomcat/webapps#
```

```
1 # 将我们修改后的容器通过commit 提交作为一个镜像。我们以后可以使用我们修改过的镜像，这是我们自己修改的镜像
```

```
[root@nanxing ~]# docker commit -m="add webapps app" -a="likunsong" 42348c61acc2 tomcat02:1.0
sha256:f8a6214ee9d8d7183deff963c839936eb8ffef9d683d8b185be680fa2c9b6443
[root@nanxing ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tomcat02	1.0	f8a6214ee9d8	28 seconds ago	684MB
tomcat	latest	24207ccc9cce	36 hours ago	680MB
redis	latest	aea9b698d7d1	7 days ago	113MB
nginx	latest	f652ca386ed1	7 days ago	141MB
portainer/portainer	latest	580c0e4e98b0	8 months ago	79.1MB

```
[root@nanxing ~]#
```

修改后的

容器数据卷

什么是容器数据卷

docker理念回顾

Docker容器数据卷，即Docker Volume（卷）。

当Docker容器运行的时候，会产生一系列的数据文件，这些数据文件会在关闭Docker容器时，直接消失的。但是其中产生部分的数据内容，我们是希望能够把它给保存起来，另作它用的。关闭Docker容器=删除内部除了image底层数据的其他全部内容，即删库跑路

1. 将应用与运行的环境打包形成容器运行，伴随着容器运行产生的数据，我们希望这些数据能够持久化。
2. 希望容器之间也能够实现数据的共享

Docker容器产生的数据同步到本地,这样关闭容器的时候，数据是在本地的，不会影响数据的安全性。docker的容器卷技术也就是将容器内部目录和本地目录进行一个同步，即挂载。

总结：容器的持久化和同步化操作，容器之间也是可以数据共享的（但是注意挂载不是等同于同步！！！）

使用数据卷

直接使用命令进行挂载 -v

```
1 docker run -it -v 主机目录: 容器内目录
2
3 [root@nanxing home]# docker run -it -v /home/ceshi:/home centos /bin/bash
4
```

```
[root@nanxing ~]# cd /home
[root@nanxing home]# ls
[root@nanxing home]# ls
ceshi
[root@nanxing home]#
```

```
[root@nanxing home]# docker run -it -v /home/ceshi:/home centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
[root@df4af4576f1a /]# cd /home
[root@df4af4576f1a home]# ls
[root@df4af4576f1a home]# ^C
[root@df4af4576f1a home]# clear
bash: clear: command not found
[root@df4af4576f1a home]#
```

- 1 # 启动起来后可以通过docker inspect 容器id 查看是否挂载成功


```
    },
    "Name": "overlay2"
  },
  "Mounts": [
    {
      "Type": "bind",
      "Source": "/home/ceshi",
      "Destination": "/home",
      "Mode": "",
      "RW": true,
      "Propagation": "rprivate"
    }
  ],
}
```

挂载 -v 卷

主机地址

docker 容器地址

测试文件的同步

1 myself

```
cation directory [/usr/local/tomcat/webapps/examples]
10-Dec-2021 09:06:20.738 INFO [main] org.apache.catal
pplication directory [/usr/local/tomcat/webapps/examp
10-Dec-2021 09:06:20.738 INFO [main] org.apache.catal
cation directory [/usr/local/tomcat/webapps/host-mana
10-Dec-2021 09:06:20.795 INFO [main] org.apache.catal
pplication directory [/usr/local/tomcat/webapps/host-
10-Dec-2021 09:06:20.815 INFO [main] org.apache.coyot
io-8080"]
10-Dec-2021 09:06:20.854 INFO [main] org.apache.catal
econds
10-Dec-2021 09:06:57.054 INFO [Thread-2] org.apache.c
p-nio-8080"]
10-Dec-2021 09:06:57.064 INFO [Thread-2] org.apache.c
[Catalina]
10-Dec-2021 09:06:57.171 INFO [Thread-2] org.apache.c
p-nio-8080"]
10-Dec-2021 09:06:57.216 INFO [Thread-2] org.apache.c
["http-nio-8080"]
[root@nanxing ~]# clear
[root@nanxing ~]# cd /home
[root@nanxing home]# ls
[root@nanxing home]# docker run -it -v /home/ceshi:/h
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67
Status: Downloaded newer image for centos:latest
[root@df4af4576f1a /]# cd /home
[root@df4af4576f1a home]# ls
[root@df4af4576f1a home]# ^C
[root@df4af4576f1a home]# clear
bash: clear: command not found
[root@df4af4576f1a home]# ls
[root@df4af4576f1a home]# touch test.java
[root@df4af4576f1a home]#
```

1 新建会话 (2)

```
[root@nanxing home]# cd ceshi
[root@nanxing ceshi]# ls
[root@nanxing ceshi]# ls
test.java
[root@nanxing ceshi]#
```

双向同步

再来测试

1. 停止容器
2. 宿主机上修改文件
3. 启动容器
4. 容器内的数据依旧是同步的


```
PORTS      NAMES
[root@nanxing home]# ls
ceshi
[root@nanxing home]# cd ceshi
[root@nanxing ceshi]# ls
test.java
[root@nanxing ceshi]# vim test.java
[root@nanxing ceshi]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
df4af4576f1a	centos	"/bin/bash"	15 minutes ago	Exited (0) 4 minutes ago
	distracted_ritchie			
c155f026b2d5	tomcat@2:1.0	"catalina.sh run"	36 minutes ago	Exited (143) 35 minutes ago
42348c61acc2	serene_darwin	"catalina.sh run"	About an hour ago	Exited (143) 38 minutes ago
	tomcat			
e071f4dde249	sleepy_ardinghelli			
	portainer/portainer	"/portainer"	2 hours ago	Exited (2) 2 hours ago
	peaceful_tharp			
cfe9659199ae	nginx	"/docker-entrypoint..."	4 days ago	Exited (0) 4 days ago
	webserver			

```
[root@nanxing ceshi]# docker start df4af4576f1a
df4af4576f1a
[root@nanxing ceshi]# docker attach df4af4576f1a
[root@df4af4576f1a /]# cd /home
[root@df4af4576f1a home]# ls
test.java
[root@df4af4576f1a home]# cat test.java
hello linux update
[root@df4af4576f1a home]#
```

优点：只用在本地修改，容器上就会自动同步

安装MySQL

MySQL的数据持久化的问题

```
1  # 获取镜像
2  [root@nanxing ceshi]# docker pull mysql:5.7
3
4  # 运行容器， 做数据挂载 # 安装启动mysql需要设置密码
5
6  # 官方测试 docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
7
8  # 启动mysql
9  -d 后台运行
10 -p 端口映射
11 -v 卷挂载 通过两个-v 挂载两个目录
12 -e 环境配置
13 --name 容器的名字
14 # 该命令行无法正常启动mysql
15
16 [root@nanxing ~]# docker run -d -p 3310:3306 -v /home/mysql/conf:/etc/mysql/conf.d -v /home/mysql/data:/var/lib/mysql -e SQL_ROOT_PASSWORD=123456 --name mysql01 mysql:5.7
17 （问题：该段代码运行后发现没有docker容器运行，通过查阅书籍，发现应该挂载一个地址即可正确运行mysql）
18
19 # 正确运行的命令
20
21 docker run --name mysql03 -p 3310:3306 -e MYSQL_ROOT_PASSWORD=123456 -v /my/custom:/etc/mysql/conf.d -d mysql:5.7
22
23 # 测试
24 通过Navicat软件进行连接测试，
```

具名和匿名挂载

```
1 # 匿名挂载
2 -v 容器内路径
3 docker run -d -P --name nginx01 -v /etc/nginx nginx
4
5 # 查看所有volume的情况
6 docker volume ls
7
8 [root@nanxing ~]# docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx
nginx
9 666e563c18473783613a5c11459ac7cd869742eb74b5b77b4218fdbf3b8487e4
10 [root@nanxing ~]# docker volume ls
11 DRIVER      VOLUME NAME
12 local       juming-nginx
13
14 # 通过 -v 卷名: 容器内路径
15 # 查看一下卷
16
```

```
local       juming-nginx
[root@nanxing ~]# docker volume inspect juming-nginx
[
  {
    "CreatedAt": "2021-12-11T00:07:50+08:00",
    "Driver": "local",
    "Labels": null,
    "Mountpoint": "/var/lib/docker/volumes/juming-nginx/_data",
    "Name": "juming-nginx",
    "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/xxx

我们通过具名挂载找到我们的卷，大多数情况下使用具名挂载

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

拓展:

```
1 # 通过 -v 容器内路径 , ro rw 改变权限
2 ro  readonly # 只读
3 rw  readwrite # 可读可写
4
5 # 一旦设定了容器的权限，容器对我们挂载出来的文件就有限定了
6 docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx:ro nginx
7 docker run -d -P --name nginx02 -v juming-nginx:/etc/nginx:rw nginx
8 # ro 只要看到了ro就说明只能通过宿主机进行操作，容器内是无法操作的
```

初识DockerFile

DockerFile 就是用来构建docker 镜像的构建文件，是一个命令脚本

通过这个脚本可以生成镜像，镜像是一层一层的，而脚本也是一个一个的命令，每个命令都是一层。

```
1 # 创建一个dockerfile文件，名字随机，建议dockerfile
2 # 文件中的内容 指令（大写） 参数
3
4 FROM centos # 添加基础镜像
5 VOLUME ["volume01", "volume02"] # 添加两个容器数据卷，属于匿名挂载
6 CMD echo "--finished----success--" # 容器构建完成输出的信息
7 CMD /bin/bash # 指定终端命令
8
9 # 这里的每个命令就是单个的一层
```

```
[root@nanxing docker-test-volume]# docker build -f /home/docker-test-volume/dockerfile1 -t likunsong/centos:1.0
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM centos
----> 300da3dc9764
Step 2/4 : VOLUME ["volume01", "volume02"]
----> Running in bec52558e0b0
Removing intermediate container bec52558e0b0
----> c1c034751030
Step 3/4 : CMD echo "-----end-----"
----> Running in 0992bdeafa4e
Removing intermediate container 0992bdeafa4e
----> 3b5bae7d1977
Step 4/4 : CMD /bin/bash
----> Running in 617345c88e96
Removing intermediate container 617345c88e96
----> c5fa6427a406
Successfully built c5fa6427a406
Successfully tagged likunsong/centos:1.0
[root@nanxing docker-test-volume]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
likunsong/centos	1.0	c5fa6427a406	2 minutes ago	231MB
mysql	5.7	730e71014906	8 days ago	140MB
nginx	latest	f652ca386ed1	8 days ago	141MB
centos	latest	5d0da3dc9764	2 months ago	231MB

```
[root@nanxing docker-test-volume]#
```

```
1 # 启动我们自己写的容器
```

```
[root@nanxing docker-test-volume]# docker run -it c5fa6427a406 /bin/bash
[root@0d130537c6cd /]# ls -l
total 56
lrwxrwxrwx 1 root root 7 Nov 3 2020 bin -> usr/bin
drwxr-xr-x 5 root root 360 Dec 11 03:14 dev
drwxr-xr-x 1 root root 4096 Dec 11 03:14 etc
drwxr-xr-x 2 root root 4096 Nov 3 2020 home
lrwxrwxrwx 1 root root 7 Nov 3 2020 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Nov 3 2020 lib64 -> usr/lib64
drwx----- 2 root root 4096 Sep 15 14:17 lost+found
drwxr-xr-x 2 root root 4096 Nov 3 2020 media
drwxr-xr-x 2 root root 4096 Nov 3 2020 mnt
drwxr-xr-x 2 root root 4096 Nov 3 2020 opt
dr-xr-xr-x 196 root root 0 Dec 11 03:14 proc
dr-xr-xr-x 2 root root 4096 Sep 15 14:17 root
drwxr-xr-x 11 root root 4096 Sep 15 14:17 run
lrwxrwxrwx 1 root root 8 Nov 3 2020 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 Nov 3 2020 srv
dr-xr-xr-x 13 root root 0 Dec 11 03:14 sys
drwxrwxrwt 7 root root 4096 Sep 15 14:17 tmp
drwxr-xr-x 12 root root 4096 Sep 15 14:17 usr
drwxr-xr-x 20 root root 4096 Sep 15 14:17 var
drwxr-xr-x 2 root root 4096 Dec 11 03:14 volume01
drwxr-xr-x 2 root root 4096 Dec 11 03:14 volume02
[root@0d130537c6cd /]#
```

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

查看一下挂载的路径

```
    },
    "Mounts": [
      {
        "Type": "volume",
        "Name": "3a18f9a6907243e912b17a6464955d7d823f7fc82cdec206b1774d4e0fa13032",
        "Source": "/var/lib/docker/volumes/3a18f9a6907243e912b17a6464955d7d823f7fc82cdec206b1774d4e0fa13032/_data",
        "Destination": "volume01",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
      },
      {
        "Type": "volume",
        "Name": "3e15f14054a921d61ab860cf57128b56a52d3ac60cac8a1c34a7fcf05f55235",
        "Source": "/var/lib/docker/volumes/3e15f14054a921d61ab860cf57128b56a52d3ac60cac8a1c34a7fcf05f55235/_data",
        "Destination": "volume02",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
      }
    ]
  },
  "Config": {
```

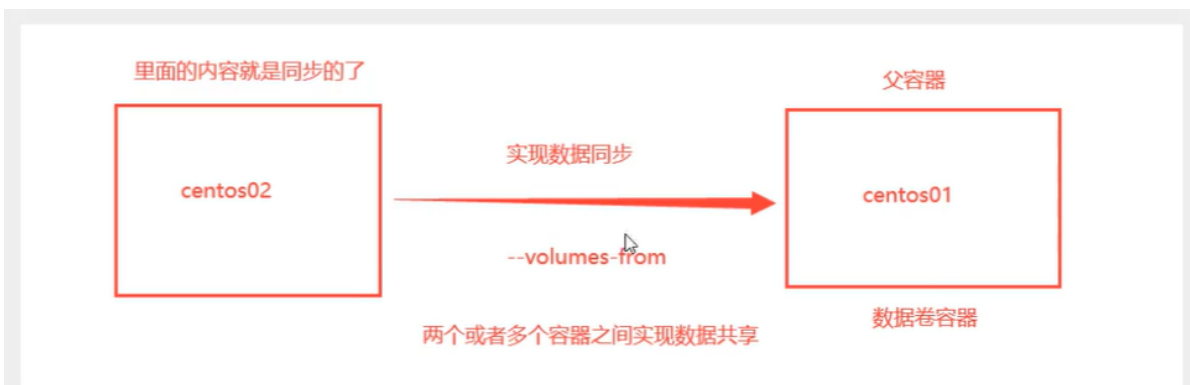
测试一下是否同步了

这种方式我们未来使用的十分多，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，手动镜像挂载 -v 卷名：容器内路径


数据卷容器

两个mysql 同步数据



1 | # 启动三个容器，通过自己写的镜像启动

```
[root@nanxing ~]# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
likunsong/centos    1.0            c5fa6427a406   28 minutes ago 231MB
mysql               5.7            738e7101490b   8 days ago     448MB
nginx               latest         f652ca386ed1   8 days ago     141MB
centos               latest         5d0da3dc9764   2 months ago   231MB
[root@nanxing ~]# docker run -it --name docker01 likunsong/centos:1.0
[root@4fef5d308c48 /]# ls
bin  etc  lib  lost+found  mnt  proc  run  srv  tmp  var  volume02
dev  home lib64 media      opt  root  sbin sys  usr  volume01
[root@4fef5d308c48 /]# ls -l
total 56
lrwxrwxrwx 1 root root 7 Nov 3 2020 bin -> usr/bin
drwxr-xr-x 5 root root 360 Dec 11 03:34 dev
drwxr-xr-x 1 root root 4096 Dec 11 03:34 etc
drwxr-xr-x 2 root root 4096 Nov 3 2020 home
lrwxrwxrwx 1 root root 7 Nov 3 2020 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Nov 3 2020 lib64 -> usr/lib64
drwx----- 2 root root 4096 Sep 15 14:17 lost+found
drwxr-xr-x 2 root root 4096 Nov 3 2020 media
drwxr-xr-x 2 root root 4096 Nov 3 2020 mnt
drwxr-xr-x 2 root root 4096 Nov 3 2020 opt
dr-xr-xr-x 206 root root 0 Dec 11 03:34 proc
dr-xr-xr-x 2 root root 4096 Sep 15 14:17 root
drwxr-xr-x 11 root root 4096 Sep 15 14:17 run
lrwxrwxrwx 1 root root 8 Nov 3 2020 sbin -> usr/sbin
drwxr-xr-x 2 root root 4096 Nov 3 2020 srv
dr-xr-xr-x 13 root root 0 Dec 11 03:34 sys
drwxrwxrwt 7 root root 4096 Sep 15 14:17 tmp
drwxr-xr-x 12 root root 4096 Sep 15 14:17 usr
drwxr-xr-x 20 root root 4096 Sep 15 14:17 var
drwxr-xr-x 2 root root 4096 Dec 11 03:34 volume01
drwxr-xr-x 2 root root 4096 Dec 11 03:34 volume02
[root@4fef5d308c48 /]#
```



数据卷

```
[C:\~]$
Connecting to 120.25.170.221:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+J'.

WARNING! The remote SSH server rejected X11 forwarding request.

Welcome to Alibaba Cloud Elastic Compute Service !

Last login: Sat Dec 11 11:40:35 2021 from 221.176.159.73
[root@nanxing ~]# docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS
4fef5d308c48   likunsong/centos:1.0 "/bin/sh -c /bin/bash"   4 hours ago    Up 4 hou
[root@nanxing ~]# docker run -it --name docker02 --volumes-from docker01 likunsong/ce
ntos:1.0
[root@dad5ecd43811 /]# ls -l
total 56
lrwxrwxrwx    1 root root    7 Nov  3  2020 bin -> usr/bin
drwxr-xr-x    5 root root  360 Dec 11 07:09 dev
drwxr-xr-x    1 root root 4096 Dec 11 07:09 etc
drwxr-xr-x    2 root root 4096 Nov  3  2020 home
lrwxrwxrwx    1 root root    7 Nov  3  2020 lib -> usr/lib
lrwxrwxrwx    1 root root    9 Nov  3  2020 lib64 -> usr/lib64
drwx-----   2 root root 4096 Sep 15 14:17 lost+found
drwxr-xr-x    2 root root 4096 Nov  3  2020 media
drwxr-xr-x    2 root root 4096 Nov  3  2020 mnt
drwxr-xr-x    2 root root 4096 Nov  3  2020 opt
dr-xr-xr-x  185 root root    0 Dec 11 07:09 proc
dr-xr-xr-x    2 root root 4096 Sep 15 14:17 root
drwxr-xr-x   11 root root 4096 Sep 15 14:17 run
lrwxrwxrwx    1 root root    8 Nov  3  2020 sbin -> usr/sbin
drwxr-xr-x    2 root root 4096 Nov  3  2020 srv
dr-xr-xr-x   13 root root    0 Dec 11 07:09 sys
drwxrwxrwt    7 root root 4096 Sep 15 14:17 tmp
drwxr-xr-x   12 root root 4096 Sep 15 14:17 usr
drwxr-xr-x   20 root root 4096 Sep 15 14:17 var
drwxr-xr-x    2 root root 4096 Dec 11 03:34 volume01
drwxr-xr-x    2 root root 4096 Dec 11 03:34 volume02
[root@dad5ecd43811 /]# cd volume01
[root@dad5ecd43811 volume01]# ls
[root@dad5ecd43811 volume01]# ls
docker01
[root@dad5ecd43811 volume01]#
```

docker01上创建的在
docker02上也显示了

- 1 # 测试，删除docker01（父容器） docker02 和docker03 文件依旧存在，且可以访问

用于实现多个mysql 实现数据共享

DockerFile

dockerfile 是用来构建docker镜像的文件，命令参数脚本

构建步骤：

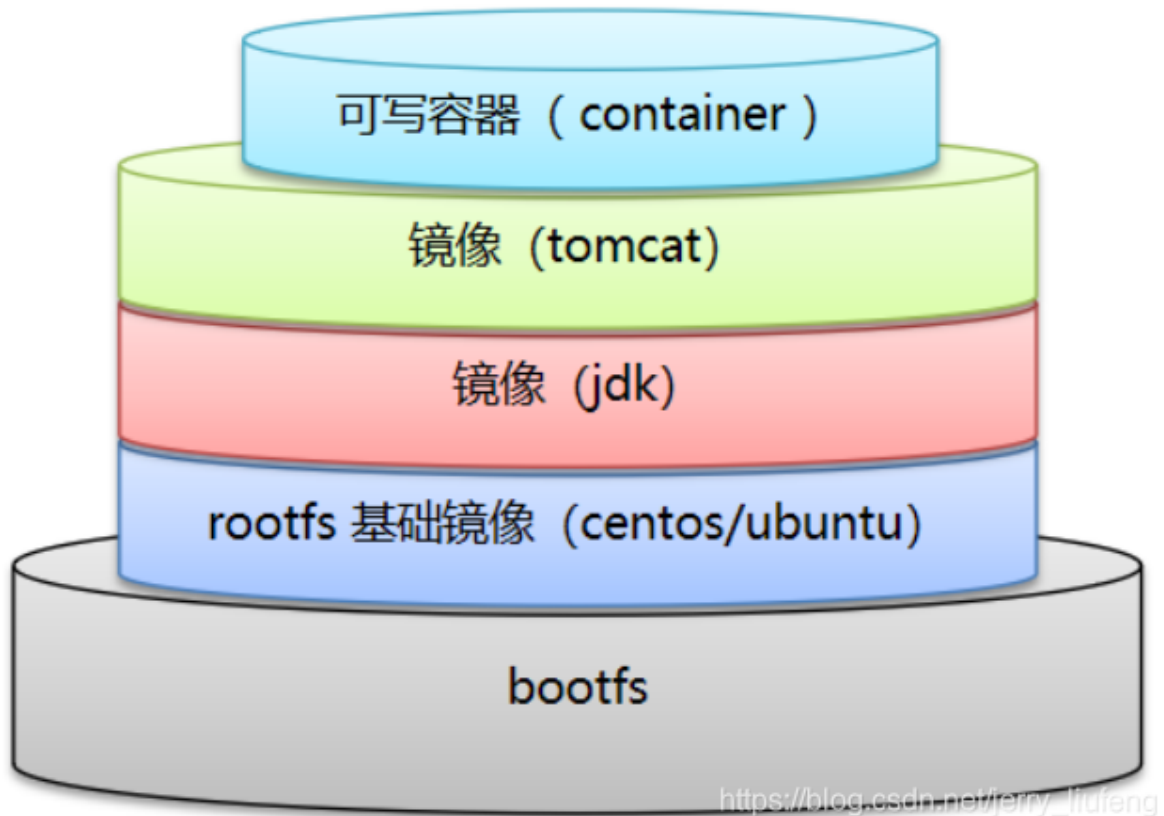
1. 编写一个Dockerfile 文件
2. docker build 构建成为一个镜像
3. docker run 运行镜像
4. docker push 发布镜像（DockerHub,阿里云镜像仓库）

官方的镜像都是基础包，很多功能都没有，我们通常要自己搭建自己的镜像！

DockerFile的构建过程

基础知识

1. 每个保留关键字（指令）都必须是大写字母
2. 指令是从上到下顺序执行的
3. #表示注释
4. 每个指令都会提交一个镜像层



dockerfile是面向开发的，我们以后要发布项目，做镜像，就需要编写dockerfile文件，这个文件十分简单。

步骤：开发，部署，运维

DokerFile：构建文件，定义了一切的步骤，源代码

DockerImags：通过dockerfile构建生成的镜像，最终发布和运行的产品

Docker容器：容器就是镜像运行起来提供服务器

DockerFile的指令

```

1 FROM          # 基础镜像，一切从这里开始构建
2 MAINTAINER    # 镜像是谁写的，姓名+邮箱
3 RUN           # 镜像构建的时候需要运行的命令
4 ADD           # 步骤，tomcat镜像，这个tomcat压缩包，添加内容
5 WORKDIR       # 镜像的工作目录
6 VOLUME        # 设置容器卷，挂载到目录
7 EXPOSE        # 暴露端口
8 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
9 ENTRYPOINT    # 指定容器启动的时候需要运行的命令，可以追加命令
10 ONBUILD       # 当构建一个被继承 DockerFile 这个时候就会运行ONBUILD 命令。触发命令
11 COPY         # 类似ADD，  将我们文件拷贝到镜像中
12 ENV          # 狗酱的时候设置环境变量  用户名，密码

```



实战测试

Docker Hub 中大部分都是从一个基础镜像出来的FROM scratch，然后配置需要的软件和配置进行构建

创建一个自己的centos

```

1 # 1.编写一个dockerfile 文件
2 [root@nanxing dockerfile]# cat mydockerfile-centos
3 FROM centos
4 MAINTAINER likunsong<1537628435@qq.com>
5
6 ENV MYPATH /usr/local
7 WORKDIR $MYPATH
8
9 RUN yum -y install vim
10 RUN yum -y install net-tools
11
12 EXPOSE 80
13
14 CMD echo $MYPATH
15 CMD echo "-----end-----"
16 CMD /bin/bash

```



```

17
18 # 2.通过这个文件构建镜像
19 # 命令 docker build -f dockerfile文件 -t 镜像名:[tag]
20
21 Successfully built 298fb113f5dd
22 Successfully tagged mycentos:0.1
23
24 # 3.测试运行
25 docker run -it centos
26 docker run -it mycentos:0.1

```

对比：我们之前原生的centos

```

[root@nanxing /]# docker run -it centos
[root@24312c0acb65 /]# pwd
/
[root@24312c0acb65 /]# vim
bash: vim: command not found
[root@24312c0acb65 /]# ifconfig
bash: ifconfig: command not found
[root@24312c0acb65 /]# exit
exit
[root@nanxing /]#

```

默认的centos没有这些功能

我们增加后的镜像

```

[root@nanxing dockerfile]# docker run -it mycentos:0.1
[root@8b2c1e200700 local]# pwd
/usr/local
[root@8b2c1e200700 local]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.2 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:ac:11:00:02 txqueuelen 0 (Ethernet)
    RX packets 24 bytes 2017 (1.9 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@8b2c1e200700 local]#

```

我们可以列出我们的给本地镜像的一个变更历史

```

[root@nanxing dockerfile]# docker history b9f15b77ec14
IMAGE          CREATED          CREATED BY          SIZE      COMMENT
b9f15b77ec14   6 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "/bin... 0B
2affa0eed05d   6 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo... 0B
fe9f8ac36d01   6 minutes ago   /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "echo... 0B
371c167ab4b9   6 minutes ago   /bin/sh -c #(nop)  EXPOSE 80             0B
811e8dc0fa62   6 minutes ago   /bin/sh -c yum -y install net-tools 14.6MB
8aadfeb8eaf2   7 minutes ago   /bin/sh -c yum -y install vim      64MB
3a226158a6c2   7 minutes ago   /bin/sh -c #(nop)  WORKDIR /usr/local    0B
523425f94b24   7 minutes ago   /bin/sh -c #(nop)  ENV MYPATH=/usr/local  0B
9527ba890461   7 minutes ago   /bin/sh -c #(nop)  MAINTAINER likunsong<1537... 0B
5d0da3dc9764   2 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]     0B
<missing>       2 months ago   /bin/sh -c #(nop)  LABEL org.label-schema.sc... 0B
<missing>       2 months ago   /bin/sh -c #(nop)  ADD file:805cb5e15fb6e0bb0... 231MB

```

可以研究镜像是怎么做的

CMD 和 ENTRYPOINT 的区别

- | | | |
|---|------------|------------------------------------|
| 1 | 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 |
| 2 | ENTRYPOINT | # 指定容器启动的时候需要运行的命令，可以追加命令 |

测试cmd

```
1 # 编写dockerfile文件
2 [root@nanxing dockerfile]# vim dockerfile-cmd-test
3 FROM centos
4 CMD ["ls","-a"]
5
6 # 构建镜像
7 [root@nanxing dockerfile]# docker build -f dockerfile-cmd-test -t mdtest .
8
9
10 # run运行 发现ls-a实现成功
11 Successfully built fb8dd831cb5d
12 Successfully tagged mdtest:latest
13 [root@nanxing dockerfile]# docker run fb8dd831cb5d
14 .
15 ..
16 .dockerenv
17 bin
18 dev
19 etc
20 home
21 lib
22 lib64
23
24 # 想追加一个命令 -l ls-al
25 [root@nanxing dockerfile]# docker run fb8dd831cb5d -l
26 docker: Error response from daemon: OCI runtime create failed:
  container_linux.go:380: starting container process caused: exec: "-l":
  executable file not found in $PATH: unknown.
27 ERRO[0000] error waiting for container: context canceled
28
29 # cmd的清理下 -l 替换了CMD["ls","-a"]命令，-l不是命令，所以报错
```

测试ENTRYPOINT

```
1 [root@nanxing dockerfile]# vim dockerfile-cmd-entrpoint
2 FROM centos
3 ENTRYPOINT ["ls","-a"]
4
5 [root@nanxing dockerfile]# docker build -f dockerfile-cmd-entrpoint -t
  entrpoint-test .
6 Sending build context to Docker daemon 4.096kB
7 Step 1/2 : FROM centos
8 ----> 5d0da3dc9764
9 Step 2/2 : ENTRYPOINT ["ls","-a"]
10 ----> Running in e97b12187fc8
11 Removing intermediate container e97b12187fc8
12 ----> 9d0e6bdfc7d7
13 Successfully built 9d0e6bdfc7d7
14 Successfully tagged entrpoint-test:latest
```

```

15 [root@nanxing dockerfile]# docker run 9d0e6bdfc7d7
16 .
17 ..
18 .dockerenv
19 bin
20 dev
21 etc
22 home
23 lib
24 lib64
25
26 # 我们的追加命令是直接拼接到我们的 ENTRYPOINT 命令后面
27 [root@nanxing dockerfile]# docker run 9d0e6bdfc7d7 -l
28 total 56
29 drwxr-xr-x  1 root root 4096 Dec 12 01:54 .
30 drwxr-xr-x  1 root root 4096 Dec 12 01:54 ..
31 -rwxr-xr-x  1 root root  0 Dec 12 01:54 .dockerenv
32 lrwxrwxrwx  1 root root  7 Nov  3  2020 bin -> usr/bin
33 drwxr-xr-x  5 root root 340 Dec 12 01:54 dev
34 drwxr-xr-x  1 root root 4096 Dec 12 01:54 etc
35 drwxr-xr-x  2 root root 4096 Nov  3  2020 home
36 lrwxrwxrwx  1 root root  7 Nov  3  2020 lib -> usr/lib
37 lrwxrwxrwx  1 root root  9 Nov  3  2020 lib64 -> usr/lib64
38 drwx----- 2 root root 4096 Sep 15 14:17 lost+found
39 drwxr-xr-x  2 root root 4096 Nov  3  2020 media
40 drwxr-xr-x  2 root root 4096 Nov  3  2020 mnt
41 drwxr-xr-x  2 root root 4096 Nov  3  2020 opt
42 dr-xr-xr-x 187 root root  0 Dec 12 01:54 proc
43 dr-xr-x---  2 root root 4096 Sep 15 14:17 root
44 drwxr-xr-x 11 root root 4096 Sep 15 14:17 run
45 lrwxrwxrwx  1 root root  8 Nov  3  2020/sbin -> usr/sbin
46 drwxr-xr-x  2 root root 4096 Nov  3  2020 srv
47 dr-xr-xr-x 13 root root  0 Dec 12 01:54 sys
48 drwxrwxrwt  7 root root 4096 Sep 15 14:17 tmp
49 drwxr-xr-x 12 root root 4096 Sep 15 14:17 usr
50

```

实战：制作tomcat镜像

1. 准备一个镜像文件 tomcat 的压缩包, jdk的压缩包 （下面是jdk和tomcat的压缩包）

```

[root@nanxing tomcat]# ls
apache-tomcat-9.0.22.tar.gz  jdk-8u11-linux-x64.tar.gz
[root@nanxing tomcat]# ll
total 165976
-rw-r--r-- 1 root root 10929702 Dec 12 11:54 apache-tomcat-9.0.22.tar.gz
-rw-r--r-- 1 root root 159019376 Dec 12 12:25 jdk-8u11-linux-x64.tar.gz
[root@nanxing tomcat]#

```

- 1 | 链接: <https://pan.baidu.com/s/1kvwwj1xS5DVL-uGCQYQESA>
abcd

提取码:

2. 编写dockerfile, 官方命名 Dockerfile, build的时候会自动寻找这个文件, 不需要 -f 指定

```

1 COPY readme.txt /usr/local/readme.txt
2
3 ADD jdk-8u11-linux-x64.tar.gz /usr/local/
4 ADD apache-tomcat-9.0.22.tar.gz /usr/local/

```

```

5
6 RUN yum -y install -vim
7
8 ENV MYPATH /usr/local
9 WORKDIR $MYPATH
10
11 ENV JAVA_HOME /usr/local/jdk1.8.0_11
12 ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
13 ENV CATALINA_HOME /usr/local/apache-tomcat-9.0.22
14 ENV CATALINA_BASH /usr/local/apache-tomcat-9.0.22
15 ENV PATH $PATH:$JAVA_HOME/bin/:$CATALINA_HOME/lib:$CATALINA_HOME/bin
16
17 EXPOSE 8080
18
19 CMD /usr/local/apache-tomcat-9.0.22/bin/startup.sh && tail -F
    /usr/local/apache-tomcat-9.0.22/bin/logs/catalina.out
20

```

3. 构建镜像

```
1 # docker build -t diytomcat .
```

4. 启动镜像

5. 访问测试

6. 发布项目（由于做了卷挂载，我们在本地编写项目就可以发布）

```

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

```

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>九九乘法口诀表</title>
5 </head>
6 <meta charset="utf-8">
7 <body bgcolor="white">
8 <h1 align="center" >
9 <font color="purple red" face="华文楷体" size="10">乘法口诀表</font>
10 </h1>
11 <p>
12 <table border="1" width="700" height="100" bgcolor="white" align="center">
13 <tr bgcolor="red purple">
14 <td>1*1=1</td>
15 </tr>
16 <tr bgcolor="purple">
17 <td>1*2=2</td>
18 <td>2*2=4</td>
19 </tr>
20 <tr bgcolor="purple green">
21 <td>1*3=3</td>

```

```
22 <td>2*3=6</td>
23 <td>3*3=9</td>
24 </tr>
25 <tr bgcolor="white purple">
26 <td>1*4=4</td>
27 <td>2*4=8</td>
28 <td>3*4=12</td>
29 <td>4*4=16</td>
30 </tr>
31 <tr bgcolor="purple green">
32 <td>1*5=5</td>
33 <td>2*5=10</td>
34 <td>3*5=15</td>
35 <td>4*5=20</td>
36 <td>5*5=25</td>
37 </tr>
38 <tr bgcolor="green purple">
39 <td>1*6=6</td>
40 <td>2*6=12</td>
41 <td>3*6=18</td>
42 <td>4*6=24</td>
43 <td>5*6=30</td>
44 <td>6*6=36</td>
45 </tr>
46 <tr bgcolor="black blue">
47 <td>1*7=7</td>
48 <td>2*7=14</td>
49 <td>3*7=21</td>
50 <td>4*7=28</td>
51 <td>5*7=35</td>
52 <td>6*7=42</td>
53 <td>7*7=49</td>
54 </tr>
55 <tr bgcolor="blue black">
56 <td>1*8=8</td>
57 <td>2*8=16</td>
58 <td>3*8=24</td>
59 <td>4*8=32</td>
60 <td>5*8=40</td>
61 <td>6*8=48</td>
62 <td>7*8=56</td>
63 <td>8*8=64</td>
64 </tr>
65 <tr bgcolor="white blue">
66 <td>1*9=9</td>
67 <td>2*9=18</td>
68 <td>3*9=27</td>
69 <td>4*9=36</td>
70 <td>5*9=45</td>
71 <td>6*9=54</td>
72 <td>7*9=63</td>
73 <td>8*9=72</td>
74 <td>9*9=81</td>
75 </tr>
76 </table>
77 </p>
78 </body>
79 </html>
```

项目部署完成，直接访问就ok

需要掌握Dockerfile的编写，之后都是使用docker镜像来进行发布运行

发布自己的镜像

DockerHub

1. 注册自己的账号
2. 登录账号
3. 在我们服务器上提交镜像

```
1 [root@nanxing tomcat]# docker login --help
2
3 Usage:  docker login [OPTIONS] [SERVER]
4
5 Log in to a Docker registry.
6 If no server is specified, the default is defined by the daemon.
7
8 Options:
9   -p, --password string    Password
10  --password-stdin          Take the password from stdin
11  -u, --username string     Username
12
```

4. 登录完之后就可以提交镜像了

```
1 1. 在dockerhub上创建一个自己仓库
2 2. 给本地镜像加标签
3 # docker tag 本地镜像名字 [hub-name]/[repository-name]:[Tag]
4 # [hub-name]为注册时使用的账号
5 # [repository-name]为建立的仓库名
6 # [Tag]为版本号
7 docker tag diytomcat lilunsong/ditomcat:1.0
8 3. 上传镜像
9 # docker push [hub-name]/[repository-name]:[Tag]
10 # 其中[hub-name]/[repository-name]:[Tag]为上文Tag完成的镜像
11
12 docker push lilunsong/ditomcat:1.0
```

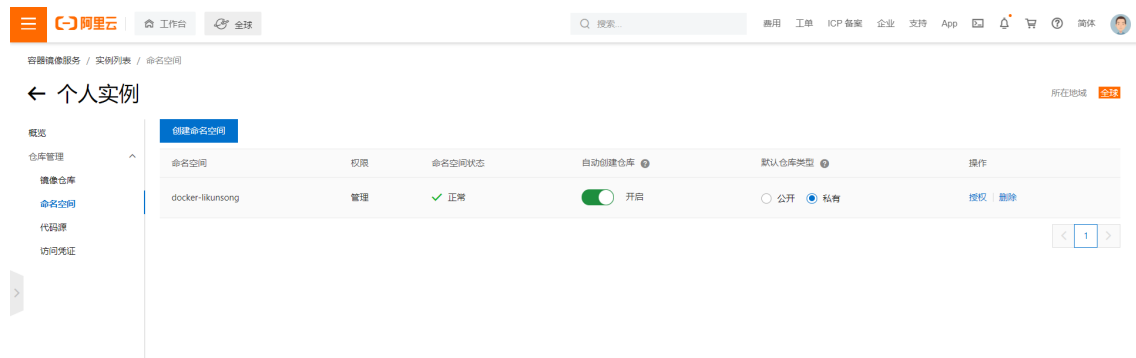
```

[root@nanxing tomcat]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
diytomcat            latest             34399f13f5ab       2 hours ago        635MB
entrypoint-test      latest             9d0e6bdfc7d7       4 hours ago        231MB
mdtest               latest             fb8dd831cb5d       5 hours ago        231MB
mycentos              0.1                b9f15b77ec14       22 hours ago       310MB
likunsong/centos     1.0                c5fa6427a406       27 hours ago       231MB
mysql                 5.7                738e7101490b       9 days ago         448MB
nginx                 latest             f652ca386ed1       9 days ago         141MB
centos                latest             5d0da3dc9764       2 months ago       231MB
[root@nanxing tomcat]# docker tag diytomcat:latest likunsong/ditomcat:1.0
[root@nanxing tomcat]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
diytomcat            latest             34399f13f5ab       2 hours ago        635MB
likunsong/ditomcat   1.0                34399f13f5ab       2 hours ago        635MB
entrypoint-test      latest             9d0e6bdfc7d7       4 hours ago        231MB
mdtest               latest             fb8dd831cb5d       5 hours ago        231MB
mycentos              0.1                b9f15b77ec14       22 hours ago       310MB
likunsong/centos     1.0                c5fa6427a406       27 hours ago       231MB
mysql                 5.7                738e7101490b       9 days ago         448MB
nginx                 latest             f652ca386ed1       9 days ago         141MB
centos                latest             5d0da3dc9764       2 months ago       231MB
[root@nanxing tomcat]# docker push likunsong/ditomcat:1.0
The push refers to repository [docker.io/likunsong/ditomcat]
8a1abaf783f7: Pushing 40.87MB/63.98MB
3822a920b181: Pushed
6bc42c7b47f4: Pushing 35.89MB/324MB
7730ec70caca: Pushed
74ddd0ec08fa: Mounted from library/centos

```

发布阿里云镜像服务

1. 登录阿里云
2. 找到容器镜像服务
3. 创建一个命名空间



4. 创建容器镜像



5. 浏览功能

1. 登录阿里云Docker Registry

```
$ docker login --username=likun**** registry.cn-shenzhen.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-shenzhen.aliyuncs.com/docker-likunsong/nanxing:[镜像版本号]
```

3. 将镜像推送到Registry

```
$ docker login --username=likun**** registry.cn-shenzhen.aliyuncs.com
$ docker tag [ImageId] registry.cn-shenzhen.aliyuncs.com/docker-likunsong/nanxing:[镜像版本号]
$ docker push registry.cn-shenzhen.aliyuncs.com/docker-likunsong/nanxing:[镜像版本号]
```

请根据实际镜像信息替换示例中的[ImageId]和[镜像版本号]参数。

4. 选择合适的镜像仓库地址

从ECS推送镜像时，可以选择使用镜像仓库内网地址。推送速度将得到提升并且将不会消耗您的公网流量。

如果您使用的机器位于VPC网络，请使用 registry-vpc.cn-shenzhen.aliyuncs.com 作为Registry的域名登录。

5. 示例

使用'docker tag'命令重命名镜像，并将它通过专有网络地址推送至Registry。

```
1  # 按照步骤，一步一步的做，就可以发布镜像成功到阿里云
2
3  docker login --username=likun**** registry.cn-shenzhen.aliyuncs.com
4
5  tag [ImageId] registry.cn-shenzhen.aliyuncs.com/docker-likunsong/nanxing:[镜像
   版本号]
6
7  push registry.cn-shenzhen.aliyuncs.com/docker-likunsong/nanxing:[镜像版本号]
```

阿里云容器镜像的参考官方文档

小结

Docker网络

docker0

清空所有环境

测试

查看ip地址

1 | ip addr

```
[root@nanxing 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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:16:3e:02:83:c4 brd ff:ff:ff:ff:ff:ff
    inet 172.25.231.57/20 brd 172.25.255.255 scope global dynamic noprefixroute eth0
        valid_lft 312373489sec preferred_lft 312373489sec
    inet6 fe80::216:3eff:fe02:83c4/64 scope link
        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:bc:9c:38:1b 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:bbff:fe8c:381b/64 scope link
        valid_lft forever preferred_lft forever
```

lo 是本地回环地址 eth0 是阿里云内网地址 docker0 是docker的地址

三个网络

1 | # 问题: docker是如何处理网络访问的

```
1 # docker run -d -P --name tomcat01 tomcat
2
3 # 查看容器内部网络地址 ip addr
4 docker exec -it tomcat01 ip addr
5
6
7 # 无法使用ip命令
8 #OCI runtime exec failed: exec failed: container_linux.go:380: starting
  container process caused: exec: "ip": executable file not found in $PATH:
  unknown
9 解决方法: 进入容器内部 docker exec -it tomcat01 /bin/bash
10 然后运行命令: apt update && apt install -y iproute2 更新下载结束后即可使用 ip addr
  的命令
11
12 附:
13 该方法也可获取容器的IP地址
14 docker inspect --format='{{.NetworkSettings.IPAddress}}' [容器名] 即可获取容器
  id
15
16
17
18 # Linux能不能ping通容器内部
19
20 [root@nanxing /]# ping 173.17.0.3
```

```

21 | PING 173.17.0.3 (173.17.0.3) 56(84) bytes of data.
22 | 64 bytes from 173.17.0.3: icmp_seq=1 ttl=44 time=248 ms
23 | 64 bytes from 173.17.0.3: icmp_seq=3 ttl=44 time=256 ms
24 |
25 | # Linux可以ping通docker容器内部

```

原理

1、我们每启动一个docker容器，docker就会给容器分配一个ip，我们只要安装了docker，就会有一个网卡docker0的桥接模式，使用的是veth-pair技术

2、再次测试IP addr

```

[root@nanxing /]# 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
80: eth0@if81: <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
[root@nanxing /]# 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:16:3e:02:83:c4 brd ff:ff:ff:ff:ff:ff
    inet 172.25.231.57/20 brd 172.25.239.255 scope global dynamic noprefixroute eth0
        valid_lft 312361338sec preferred_lft 312361338sec
    inet6 fe80::216:3eff:fe02:83c4/64 scope link
        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:bb:8c:38:1b 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:bbff:fe8c:381b/64 scope link
        valid_lft forever preferred_lft forever
81: vetha32c7ff8@if80: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 06:2d:ce:1d:17:1a brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b42d:ceff:feld:171a/64 scope link
        valid_lft forever preferred_lft forever
83: veth6c9236a@if82: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 2e:9a:26:70:d2:b4 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::2c9a:26ff:fe70:d2b4/64 scope link
        valid_lft forever preferred_lft forever

```

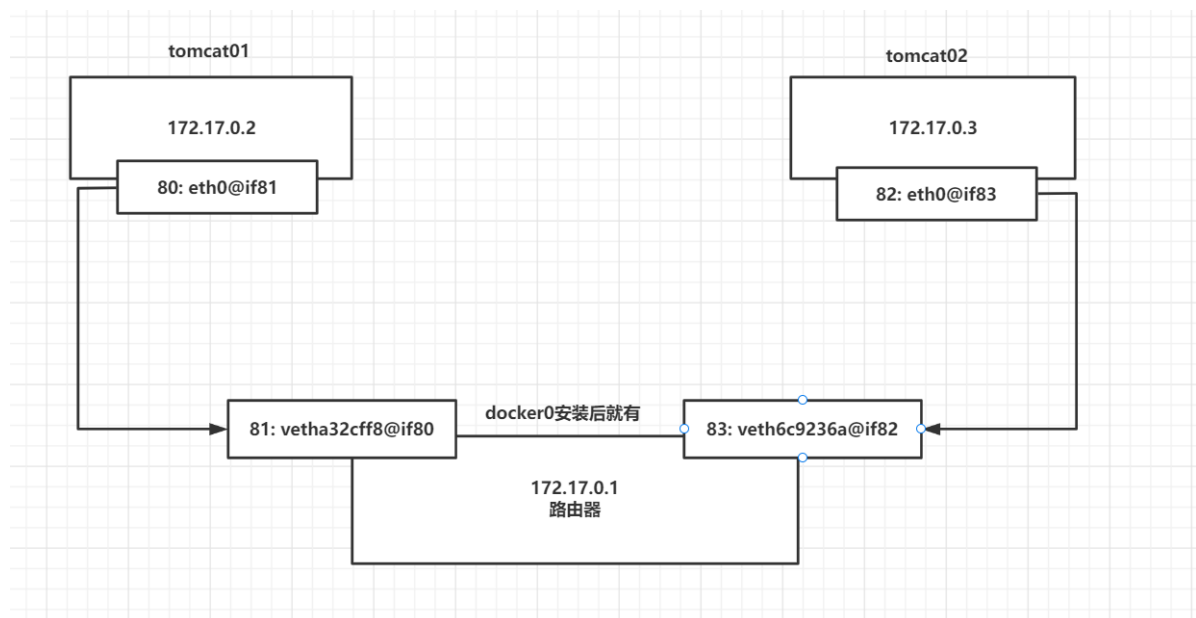
- 1 | # 我们发现容器带来的网卡都是一对一对的
- 2 | # veth-pair 就是一对虚拟设备接口，他们都是成对出现的，一段连着协议，一段彼此相连
- 3 | # 因为这个特性，veth-pair 充当一个桥梁，连接各种虚拟的网络设备
- 4 | # openstack, docker之间的连接，ovs之间的连接，都是使用veth-pair技术

3、测试一下tomcat01 和 tomcat02 是否可以ping通

- 1 | 无法使用ping命令
- 2 |
- 3 | 1. 进入容器 docker exec -it tomcat01 /bin/bash
- 4 | 2. 下载 apt-get update
- 5 | apt install net-tools #ifconfig
- 6 | apt install iputils-ping #ping
- 7 |
- 8 | 即可使用ping命令

- 1 | [root@nanxing /]# docker exec -it tomcat01 ping 172.17.0.3
- 2 |
- 3 | # 结论容器和容器之间是可以互相ping通的

绘制一个网络模型图：

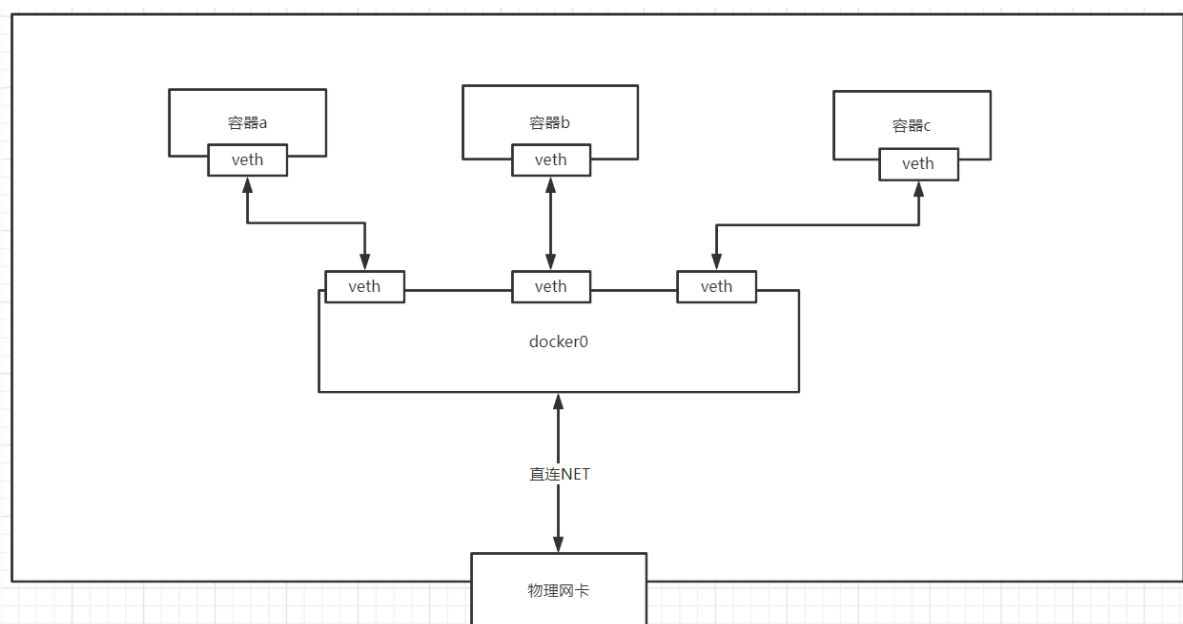


结论：tomcat01 和 tomcat02 是共用的的一个路由器，docker0

所有容器在不指定网络的情况下，都是docker0路由的，docker会给我们分配一个默认的可用ip

小结

docker 使用的是linux的桥接，宿主机中是Docker容器的网桥 docker0.



docker所有的网络接口都是虚拟的。虚拟的转化效率高（内网传递）

只要容器被删除，对应网桥这一对就没有了

--link

```

1 [root@nanxing ~]# docker exec -it tomcat01 ping tomcat02
2 ping: tomcat02: Name or service not known
3
4
5 # 如何解决无法ping通
6
7 # docker run -d -P --name tomcat02 --link tomcat02 tomcat
8
9 # docker exec -it tomcat03 ping tomcat02
10 即可ping成功
11
12 # 但是反向不能ping通

```

不建议使用--link，使用自定义网络

自定义网络

查看所有的docker网络

```

[root@nanxing ~]# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
0f3646f0819a    bridge    bridge       local
1ccd4c2658c5    host      host         local
0907491689cd    none      null          local
[root@nanxing ~]#

```

网络模式

bridge：桥接模式 在docker（默认）上面搭桥

none：不配置网络

host：和宿主机共享网络

container：容器内可以网络连通（用的少）

默认只有前三个

测试

```

1 # 我们直接启动的命令 --net bridge，而这个就是我们的docker0
2 docker run -d -P --name tomcat01 --net bridge tomcat
3
4 #docker0 特点：默认，域名不能访问，--link可以打通连接
5
6 #我们可以自定义一个网络服务
7 # --driver bridge
8 # --subnet 192.168.0.0/16
9 # --gateway 192.168.0.1
10 [root@nanxing /]# docker network create --driver bridge --subnet
11 192.168.0.0/16 --gateway 192.168.0.1 mynet
12 [root@nanxing /]# docker network ls
13 NETWORK ID      NAME      DRIVER      SCOPE
14 0f3646f0819a    bridge    bridge       local
15 1ccd4c2658c5    host      host         local
16 3e2e3c86c5c2    mynet     bridge       local

```

```
17 0907491689cd  none      null      local
18
```

```
1 [root@nanxing /]# docker run -d -P --name tomcat-net-01 --net mynet tomcat
2 4a698ed723008a4d37b100f0f39d3caf36d2d2ba4bcf7d341ee3f47f3675fd39
3 [root@nanxing /]# docker run -d -P --name tomcat-net-02 --net mynet tomcat
4 805675d14ed36d849041b93920525f04c594bb8d2d9879fd96400fb55ecea3b3
5 [root@nanxing /]# docker network inspect mynet
6 [
7     {
8         "Name": "mynet",
9         "Id":
10      "3e2e3c86c5c2a5ccac0b77af80d0266b769fadbddef00a4f558ca9909fcc72c0e",
11         "Created": "2022-01-10T18:11:21.494578404+08:00",
12         "Scope": "local",
13         "Driver": "bridge",
14         "EnableIPv6": false,
15         "IPAM": {
16             "Driver": "default",
17             "Options": {},
18             "Config": [
19                 {
20                     "Subnet": "192.168.0.0/16",
21                     "Gateway": "192.168.0.1"
22                 }
23             ],
24             "Internal": false,
25             "Attachable": false,
26             "Ingress": false,
27             "ConfigFrom": {
28                 "Network": ""
29             },
30             "ConfigOnly": false,
31             "Containers": {
32
33      "4a698ed723008a4d37b100f0f39d3caf36d2d2ba4bcf7d341ee3f47f3675fd39": {
34          "Name": "tomcat-net-01",
35          "EndpointID":
36      "b519d341f1a5c7419bbc973bd60c033a9a985fdcc0501e9dcdb150ea3620f494",
37          "MacAddress": "02:42:c0:a8:00:02",
38          "IPv4Address": "192.168.0.2/16",
39          "IPv6Address": ""
40      },
41
42      "805675d14ed36d849041b93920525f04c594bb8d2d9879fd96400fb55ecea3b3": {
43          "Name": "tomcat-net-02",
44          "EndpointID":
45      "52351b141864e710219743a6d474715eb7316a05995784d0548f4f654219dd35",
46          "MacAddress": "02:42:c0:a8:00:03",
47          "IPv4Address": "192.168.0.3/16",
48          "IPv6Address": ""
49      }
50     },
51     "Options": {},
52     "Labels": {}
53 }
```

```
50 ]
51
52 现在使用ping可以直接连接两个容器，不需要使用--link
```

自定义网络已经帮我们维护好了对应的关系，平时就使用这种自定义网络

好处：不同的集群使用不同的网络，保证集群是安全的

网络联通

```
[root@nanxing /]# 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

Run 'docker network COMMAND --help' for more information on a command.
```

```
[root@nanxing /]# 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@nanxing /]#
```

```
1  # 打通 容器 和 自定义网络
2
3  # 连通之后，就是将容器 放到了 自定义网络下
4
5  # 一个容器，两个ip
```

redis集群部署

```
1  # 创建网卡
2  docker network redis --subnet 172.38.0.0/16
3
4  # 通过脚本创建六个redis配置
5  for port in $(seq 1 6); \
6  do \
```

```

7  mkdir -p /mydata/redis/node-`${port}`/conf
8  touch /mydata/redis/node-`${port}`/conf/redis.conf
9  cat << EOF >/mydata/redis/node-`${port}`/conf/redis.conf
10 port 6379
11 bind 0.0.0.0
12 cluster-enabled yes
13 cluster-config-file nodes.conf
14 cluster-node-timeout 5000
15 cluster-announce-ip 172.38.0.1`${port}`
16 cluster-announce-port 6379
17 cluster-announce-bus-port 16379
18 appendonly yes
19 EOF
20 done
21
22 for port in $(seq 1 6); \
23 do \
24 cat << EOF >/mydata/redis/node-`${port}`/conf/redis.conf
25 port 6379
26 bind 0.0.0.0
27 cluster-enabled yes
28 cluster-config-file nodes.conf
29 cluster-node-timeout 5000
30 cluster-announce-ip 172.38.0.1`${port}`
31 cluster-announce-port 6379
32 cluster-announce-bus-port 16379
33 appendonly yes
34 EOF
35 done
36
37 # 启动容器
38 for port in $(seq 1 6);
39 do
40 docker run -p 637`${port}`:6379 -p 1637`${port}`:16379 --name redis-`${port}` \
41 -v /mydata/redis/node-`${port}`/data:/data \
42 -v /mydata/redis/node-`${port}`/conf/redis.conf:/etc/redis/redis.conf \
43 -d --net redis --ip 172.38.0.1`${port}` redis:5.0.9-alpine3.11 redis-server
44 /etc/redis/redis.conf
45 done
46
47 docker run -p 6371:6379 -p 16371:16379 --name redis-1 \
48 -v /mydata/redis/node-1/data:/data \
49 -v /mydata/redis/node-1/conf/redis.conf:/etc/redis/redis.conf \
50 -d --net redis --ip 172.38.0.11 redis:5.0.9-alpine3.11 redis-server
51 /etc/redis/redis.conf
52
53 #c
54 [root@nanxing conf]# docker exec -it redis-1 /bin/sh
55 /data # ls
56 appendonly.aof nodes.conf
57 /data # redis-cli --cluster create 172.38.0.11:6379 172.38.0.12:6379
58 172.38.0.13:6379 172.38.0.14:6379 172.38.0.15:6379 172.38.0.16:6379 --
59 cluster-replicas 1

```