

NSD CLOUD DAY06

1. [案例1：制作自定义镜像](#)
2. [案例2：创建私有镜像仓库](#)
3. [案例3：NFS共享存储](#)
4. [案例4：创建自定义网桥](#)

1 案例1：制作自定义镜像

1.1 问题

本案例要求制作自定义镜像：

- 基于centos镜像使用commit创建新的镜像文件
- 基于centos镜像使用Dockerfile文件创建一个新的镜像文件

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：使用镜像启动容器

1) 在该容器基础上修改yum源

```
01. [ root@docker1 docker_images ] # docker run - it centos
02. [ root@8d07ecd7e345 / ] # rm - rf /etc/yum.repos.d/*
03. [ root@8d07ecd7e345 / ] # vi /etc/yum.repos.d/dvd.repo
04. [ dvd]
05. name=dvd
06. baseurl=ftp://192.168.1.254/system
07. enabled=1
08. gpgcheck=0
09. [ root@8d07ecd7e345 / ] # yum clean all
10. [ root@8d07ecd7e345 / ] # yum repolist
```

2) 安装测试软件

```
01. [ root@8d07ecd7e345 / ] # yum - y install net- tools iproute psmisc vim- enhanced
```

3) ifconfig查看

```
01. [ root@8d07ecd7e345 / ] # if config
```

[Top](#)

```

02. eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
03.      inet 172.17.0.3 netmask 255.255.0.0 broadcast 0.0.0.0
04.      inet6 fe80::42:acff:fe11:3 prefixlen 64 scopeid 0x20<link>
05.      ether 02:42:ac:11:00:03 txqueuelen 0 (Ethernet)
06.      RX packets 2488 bytes 28317945 (27.0 MB)
07.      RX errors 0 dropped 0 overruns 0 frame 0
08.      TX packets 1858 bytes 130264 (127.2 KiB)
09.      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
10. [root@8d07ecd7e345 /] # exit
11. exit

```

步骤二：另存为另外一个镜像

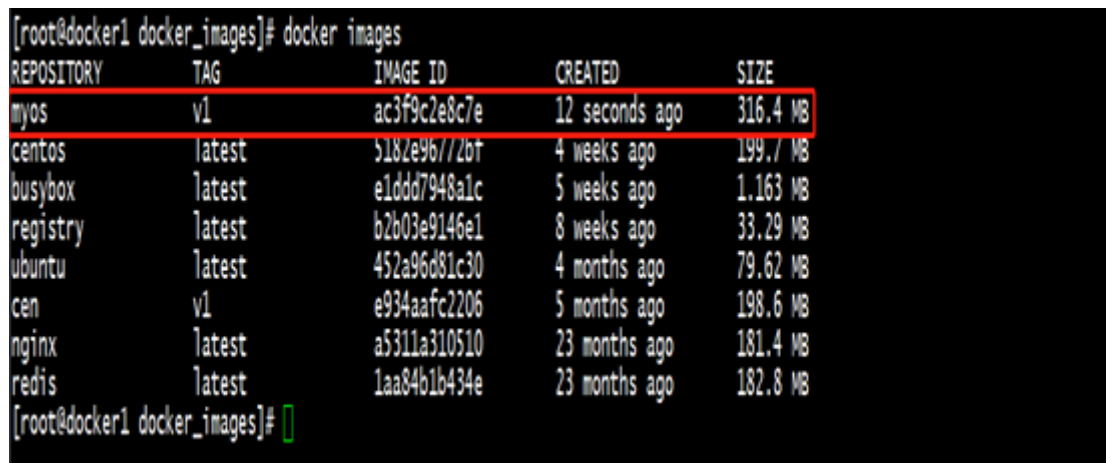
1) 创建新建镜像

```

01. [root@docker1 docker_images] # docker start 8d07ecd7e345
02. //可以简写为8d，要保证唯一性
03. 8d07ecd7e345
04. [root@docker1 docker_images] # docker commit 8d07ecd7e345 my os: v1
05. sha256: ac3f9c2e8c7e13db183636821783f997890029d687b694f5ce590a473ad82c5f

```

2) 查看新建的镜像，如图-1所示：



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myos	v1	ac3f9c2e8c7e	12 seconds ago	316.4 MB
centos	latest	5182e9677267	4 weeks ago	199.7 MB
busybox	latest	e1ddd7948a1c	5 weeks ago	1.163 MB
registry	latest	b2b03e9146e1	8 weeks ago	33.29 MB
ubuntu	latest	452a96d81c30	4 months ago	79.62 MB
cen	v1	e934aafc2206	5 months ago	198.6 MB
nginx	latest	a5311a310510	23 months ago	181.4 MB
redis	latest	1aa84b1b434e	23 months ago	182.8 MB

图-1

3) 验证新建镜像

```

01. [root@docker1 docker_images] # docker run -it my os: v1
02. [root@497c7b4664bf /] # ifconfig
03. eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
04.      inet 172.17.0.6 netmask 255.255.0.0 broadcast 0.0.0.0

```

[Top](#)

05. inet6 fe80::42:acff:fe11:6 prefixlen 64 scopeid 0x20<link>
06. ether 02:42:ac:11:00:06 txqueuelen 0 (Ethernet)
07. RX packets 0 bytes 0 (0.0 B)
08. RX errors 0 dropped 0 overruns 0 frame 0
09. TX packets 7 bytes 578 (578.0 B)
10. TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

步骤三：使用Dockerfile文件创建一个新的镜像文件

Dockerfile语法格式：

- FROM:基础镜像
- MAINTAINER:镜像创建者信息(说明)
- EXPOSE:开放的端口
- ENV:设置环境变量
- ADD:复制文件到镜像
- RUN:制作镜像时执行的命令,可以有多个
- WORKDIR:定义容器默认工作目录
- CMD:容器启动时执行的命令,仅可以有一条CMD

1) 创建一个Apache的镜像文件

01. [root@docker1 ~] # mkdir oo
02. [root@docker1 ~] # cd oo
03. [root@docker1 oo] # touch Dockerfile //Dockerfile文件第一个字母要大写
04. [root@docker1 oo] # cp /etc/yum.repos.d/local.repo ./
05. [root@docker1 oo] # vi Dockerfile
06. FROM myos:v1
07. RUN yum -y install httpd
08. ENV EnvironmentFile=/etc/sysconfig/httpd
09. WORKDIR /var/www/html/ //定义容器默认工作目录
10. RUN echo "test" > /var/www/html/index.html
11. EXPOSE 80 //设置开放端口号
12. CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
13. [root@docker1 oo] # docker build -t myos:http .
14. [root@docker1 oo] # docker run -d myos:http
15. d9a5402709b26b42cd304c77be442559a5329dc784ec4f6c90e4abac1c88e206
16. [root@docker1 oo] # docker inspect d9
17. [root@docker1 oo] # curl 172.17.0.7
18. test

[Top](#)

2 案例2：创建私有镜像仓库

2.1 问题

本案例要求创建私有的镜像仓库：

- Docker主机：192.168.1.20
- 镜像仓库服务器：192.168.1.10

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：自定义私有仓库

1) 定义一个私有仓库

```

01. [ root@docker1 oo] # vim /etc/docker/daemon.json //不写这个文件会报错
02. {
03.     "insecure-registries": [ "192.168.1.10:5000" ] //使用私有仓库运行容器
04. }
05. [ root@docker1 oo] # systemctl restart docker
06. [ root@docker1 oo] # docker run -d -p 5000:5000 registry
07. 273be3d1f3280b392cf382f4b74fea53aed58968122eff69fd016f638505ee0e
08. [ root@docker1 oo] # curl 192.168.1.10:5000/v2/
09. {} //出现括号
10. [ root@docker1 oo] # docker tag busybox:latest 192.168.1.10:5000/busybox:latest
11. //打标签
12. [ root@docker1 oo] # docker push 192.168.1.10:5000/busybox:latest //上传
13. [ root@docker1 oo] # docker tag myos:http 192.168.1.10:5000/myos:http
14. [ root@docker1 oo] # docker push 192.168.1.10:5000/myos:http
  
```

2) 在docker2上面启动

```

01. [ root@docker2 ~] # scp 192.168.1.10:/etc/docker/daemon.json /etc/docker/
02. [ root@docker2 ~] # systemctl restart docker
03. [ root@docker2 ~] # docker images
04. [ root@docker2 ~] # docker run -it 192.168.1.10:5000/myos:http /bin/bash
05. //直接启动
  
```

步骤二：查看私有仓库

1) 查看里面有什么镜像

[Top](#)

```
01. [root@docker1 oo] # curl http://192.168.1.10:5000/v2/_catalog
02. {"repositories":["busy box","my os"]}
```

2) 查看里面的镜像标签

```
01. [root@docker1 oo] # curl http://192.168.1.10:5000/v2/busy box/tags/list
02. {"name":"busy box","tags":["latest"]}
03. [root@docker1 oo] # curl http://192.168.1.10:5000/v2/my os/tags/list
04. {"name":"my os","tags":["http"]}
```

3 案例3：NFS共享存储

3.1 问题

本案例要求创建NFS共享，能映射到容器里：

- 服务器创建NFS共享存储，共享目录为/content，权限为rw
- 客户端挂载共享，并将共享目录映射到容器中

3.2 方案

本方案要求需要一台NFS服务器（NFS用真机代替），ip为192.168.1.254，一台客户端docker1主机，ip为192.168.1.10，一台客户端docker2主机，ip为192.168.1.20，实现客户端挂载共享，并将共享目录映射到容器中，docker1更新文件时，docker2实现同步更新，方案如图-2所示：

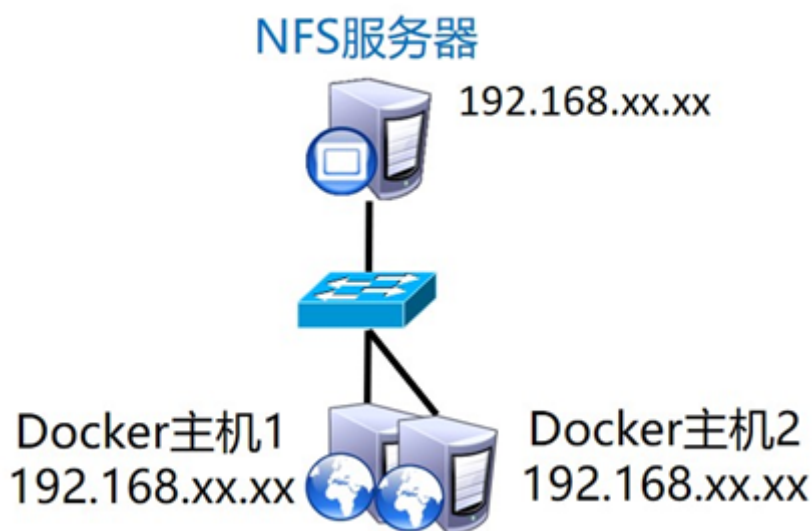


图-2

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：配置NFS服务器

[Top](#)

```

01. [ root@room9pc01 ~] # yum -y install nfs-utils
02. [ root@room9pc01 ~] # mkdir /content
03. [ root@room9pc01 ~] # vim /etc/exports
04. /content *(rw,no_root_squash)
05. [ root@room9pc01 ~] # systemctl restart nfs-server.service
06. [ root@room9pc01 ~] # systemctl restart nfs-secure.service
07. [ root@room9pc01 ~] # exportfs -rv
08. exporting */content
09. [ root@room9pc01 ~] # chmod 777 /content
10. [ root@room9pc01 ~] # echo 11 > /content/index.html

```

步骤二：配置客户端

```

01. [ root@docker100 ~] # yum -y install nfs-utils
02. [ root@docker100 ~] # systemctl restart nfs-server.service
03. [ root@docker100 ~] # showmount -e 192.168.1.254
04. Export list for 192.168.1.254:
05. /content *
06. [ root@docker1 ~] # mkdir /mnt/qq
07. [ root@docker1 ~] # mount -t nfs 192.168.1.254:/content /mnt/qq
08. [ root@docker1 ~] # ls /mnt/qq
09. index.html
10. [ root@docker1 ~] # cat /mnt/qq/index.html
11. 11
12. [ root@docker1 ~] # docker run -d -p 80:80 -v /mnt/qq:/var/www/html -it myos: http
13. 224248f0df5d795457c43c2a7dad0b7e5ec86abdc3f31d577e72f7929f020e01
14. [ root@docker1 ~] # curl 192.168.1.10
15. 11
16. [ root@docker2 ~] # yum -y install nfs-utils
17. [ root@docker2 ~] # showmount -e 192.168.1.254
18. Export list for 192.168.1.254:
19. /content *
20. [ root@docker2 ~] # mkdir /mnt/qq
21. [ root@docker2 ~] # mount -t nfs 192.168.1.254:/content /mnt/qq
22. [ root@docker2 ~] # docker run -d -p 80:80 -v /mnt/qq:/var/www/html -it 192.168.1.10
23. 00346dabec2c7a12958da4b7fee6551020249cdcb111ad6a1058352d2838742a
24. [ root@docker2 ~] # curl 192.168.1.20
25. 11
26. [ root@docker1 ~] # touch /mnt/qq/a.sh
27. [ root@docker1 ~] # echo 22 > /mnt/qq/index.html

```

[Top](#)

```

28. [root@docker2 ~] #s /mnt/qq/
29. a.sh index.html
30. [root@docker2 ~] # cat /mnt/qq/index.html
31. 22

```

4 案例4：创建自定义网桥

4.1 问题

本案例要求：

- 创建网桥设备docker01
- 设定网段为172.30.0.0/16
- 启动nginx容器，nginx容器桥接docker01设备
- 映射真实机8080端口与容器的80端口

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建Docker网络模型

1) 新建docker1网络模型

```

01. [root@docker1 ~] # docker network create -- subnet=172.30.0.0/16 docker01
02. c9cf 26f 911ef 2dccb1f d1f 670a6c51491e72b49133246f 6428dd732c44109462
03. [root@docker1 ~] # docker network list
04. NETWORK ID          NAME           DRIVER         SCOPE
05. bc189673f959         bridge        bridge         local
06. 6622752788ea         docker01     bridge         local
07. 53bf43bdd584         host         host           local
08. ac52d3151ba8         none         null           local
09. [root@docker1 ~] # ip a s
10. [root@docker1 ~] # docker network inspect docker01
11. [
12. {
13.     "Name": "docker01",
14.     "Id": "c9cf 26f 911ef 2dccb1f d1f 670a6c51491e72b49133246f 6428dd732c44109462",
15.     "Scope": "local",
16.     "Driver": "bridge",
17.     "EnableIPv6": false,
18.     "IPAM": {
19.         "Driver": "default",
20.         "Options": {},

```

[Top](#)

```

21.         "Config": [
22.             {
23.                 "Subnet": "172.30.0.0/16"
24.             }
25.         ]
26.     },
27.     "Internal": false,
28.     "Containers": {},
29.     "Options": {},
30.     "Labels": {}
31. }
32. ]

```

2) 使用自定义网桥启动容器

```
01. [root@docker1 ~]# docker run -- network=docker01 -id nginx
```

3) 端口映射

```

01. [root@docker1 ~]# docker run -p 8080:80 -id nginx
02. e523b386f9d6194e53d0a5b6b8f5ab4984d062896bab10639e41aef657cb2a53
03. [root@docker1 ~]# curl 192.168.1.10:8080

```

步骤二：扩展实验

1) 新建一个网络模型docker02

```

01. [root@docker1 ~]# docker network create --driver bridge docker02
02. //新建一个 名为docker02的网络模型
03. 5496835bd3f53ac220ce3d8be71ce6afc919674711ab3f94e6263b9492c7d2cc
04. [root@docker1 ~]# ifconfig
05. //但是在用ifconfig命令查看的时候，显示的名字并不是docker02，而是br-5496835bd3f5
06. br-5496835bd3f5: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
07.     inet 172.18.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
08.     ether 02:42:89:6a:a2:72 txqueuelen 0 (Ethernet)
09.     RX packets 8 bytes 496 (496.0 B)
10.     RX errors 0 dropped 0 overruns 0 frame 0
11.     TX packets 8 bytes 496 (496.0 B)

```

[Top](#)


```

12.          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
13.
14.  [ root@docker1 ~] # docker network list           //查看显示docker02 (查看加粗字样)
15.  NETWORK ID      NAME      DRIVER      SCOPE
16.  bc189673f959    bridge   bridge      local
17.  5496835bd3f5    docker02 bridge      local
18.  53bf43bdd584    host     host        local
19.  ac52d3151ba8    none     null        local

```

2) 若要解决使用ifconfig命令可以看到docker02的问题，可以执行以下几步命令

```

01.  [ root@docker1 ~] # docker network list           //查看docker02的NETWORK ID (加粗字样)
02.  NETWORK ID      NAME      DRIVER      SCOPE
03.  bc189673f959    bridge   bridge      local
04.  5496835bd3f5    docker02 bridge      local
05.  53bf43bdd584    host     host        local
06.  ac52d3151ba8    none     null        local

```

3) 查看16dc92e55023的信息，如图-3所示：

```

01.  [ root@docker2 ~] # docker network inspect bc189673f959

```

[Top](#)

```
[
{
  "Name": "bridge",
  "Id": "bc189673f959bf338d9fa2a70186d9632b0107667eb8434d6851916408ab1aa4",
  "Scope": "local",
  "Driver": "bridge",
  "EnableIPv6": false,
  "IPAM": {
    "Driver": "default",
    "Options": null,
    "Config": [
      {
        "Subnet": "172.17.0.0/16",
        "Gateway": "172.17.0.1"
      }
    ]
  },
  "Internal": false,
  "Containers": {
    "224248f0df5d795457c43c2a7dad0b7e5ec86abdc3f31d577e72f7929f020e01": {
      "Name": "happy_lichterman",
      "EndpointID": "3932768411fbc9593238615150704d01d4bbf2f84e60e7c6dba6e8b604353dbc",
      "MacAddress": "02:42:ac:11:00:02",
      "IPv4Address": "172.17.0.2/16",
      "IPv6Address": ""
    }
  },
  "Options": {
    "com.docker.network.bridge.default_bridge": "true"
  }
}
```

图-3

4) 查看图片的倒数第六行有"com.docker.network.bridge.name": "docker0"字样

5) 把刚刚创建的docker02网桥删掉

```
01. [root@docker1 ~]# docker network rm docker02 //删除docker02
02. docker02
03. [root@docker1 ~]# docker network create \
04. docker02 -o com.docker.network.bridge.name=docker02
05. //创建docker02网桥
06. 648bd5da03606d5a1a395c098662b5f820b9400c6878e2582a7ce754c8c05a3a
07. [root@docker1 ~]# ifconfig //ifconfig查看有docker02
08. docker02: flags=4096<UP,BROADCAST,MULTICAST> mtu 1500
09.     inet 172.18.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
10.     ether 02:42:94:27:a0:43 txqueuelen 0 (Ethernet)
11.     RX packets 0 bytes 0 (0.0 B)
12.     RX errors 0 dropped 0 overruns 0 frame 0
13.     TX packets 0 bytes 0 (0.0 B)
14.     TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

[Top](#)

6) 若想在创建docker03的时候自定义网段(之前已经创建过docker01和02,这里用docker03),执行以下命令

```
01. [root@docker1 ~]# docker network create docker03 -- subnet=172.30.0.0/16 - o com.dock
02. f003aa1c0fa20c81e4f73c12dcc79262f1f1d67589d7440175ea01dc0be4d03c
03. [root@docker1 ~]# ifconfig //ifconfig查看,显示的是自己定义的网段
04. docker03: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
05.     inet 172.30.0.1 netmask 255.255.0.0 broadcast 0.0.0.0
06.     ether 02:42:27:9b:95:b3 txqueuelen 0 (Ethernet)
07.     RX packets 0 bytes 0 (0.0 B)
08.     RX errors 0 dropped 0 overruns 0 frame 0
09.     TX packets 0 bytes 0 (0.0 B)
10.     TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```



[Top](#)