

🔗 动手实战学Docker (/courses/498)

网络管理

1. 课程说明

课程为纯动手实验教程，为了能说清楚实验中的一些操作会加入理论内容。理论内容我们不会写太多，已经有太多好文章了，会精选最值得读的文章推荐给你，在动手实践的同时扎实理论基础。

实验环境中可以联网，不受实验楼网络限制。

2. 学习方法

实验楼的Docker课程包含14 个实验，每个实验都提供详细的步骤和截图，适用于有一定Linux系统基础，想快速上手Docker的同学。

学习方法是多实践，多提问。启动实验后按照实验步骤逐步操作，同时理解每一步的详细内容。

如果实验开始部分有推荐阅读的材料，请务必先阅读后再继续实验，理论知识是实践必要的基础。

3. 本节内容简介

本节中，我们需要依次完成下面几项任务：

1. docker 容器端口映射
2. 自定义网络实现容器互联
3. host 和 none 网络的使用

4. 网络

在开始下面的内容之前，为了不出现命名上的冲突，也为了显示更为直观并且方便演示示例，首先需要将前面创建或启动的容器全部删除。可以使用下面两条命令达到这一效果：

```
# 暂停所有运行中的容器
$ docker container ls -q | xargs docker container stop

# 删除所有的容器
$ docker container ls -aq | xargs docker container rm
```

在我们安装 Docker 后，会自动创建三个网络。我们可以使用下面的命令来查看这些网络：

👉 动手实战学Docker (/courses/498)

```
$ docker network ls
```

```
shiyancelou:~/ $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
43baf0b22ca7        bridge             bridge             local
04dcaff0d5a4        host              host              local
4ce7c23db6fc        none              null              local
shiyancelou:~/ $
```

如上图所示，三种默认的网络，分别为 bridge，host，none。

4.1 bridge

bridge，即桥接网络，在安装 docker 后会创建一个桥接网络，该桥接网络的名称为 docker0。我们可以通过下面两条命令去查看该值。

```
# 查看 bridge 网络的详细信息，并通过 grep 获取名称项
$ docker network inspect bridge | grep name
```

```
# 使用 ifconfig 查看 docker0 网络
ifconfig
```

```
shiyancelou:~/ $
shiyancelou:~/ $ docker network inspect bridge | grep name
"com.docker.network.bridge.name": "docker0",
shiyancelou:~/ $
shiyancelou:~/ $ ifconfig
docker0  Link encap:以太网 硬件地址 02:42:a0:25:e1:e1
          inet 地址:192.168.0.1 广播:0.0.0.0 掩码:255.255.240.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
          接收数据包:34366 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:49438 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:0
          接收字节:1905102 (1.9 MB) 发送字节:151036308 (151.0 MB)


eth0     Link encap:以太网 硬件地址 00:16:3e:00:7a:33
          inet 地址:10.174.32.81 广播:10.174.39.255 掩码:255.255.248.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
          接收数据包:51054 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:51395 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:1000
          接收字节:40040187 (40.0 MB) 发送字节:3961213 (3.9 MB)
```

在上图中，我们可以查看到对应的值。默认情况下，我们创建一个新的容器都会自动连接到 bridge 网络。其详细信息如下所示：

```
shiyanolou:~/ $ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "43baf0b22ca7e6c3bdf52f859e1353f8baa628885a0e66952eddfce0ba32ce10",
    "Created": "2018-01-10T13:22:00.234970024+08:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "192.168.0.0/20",
          "Gateway": "192.168.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

子网

名称



我们可以尝试创建一个容器，该容器会自动连接到 bridge 网络，例如我们创建一个名为 shiyanlou001 的容器：

```
$ docker container run -itd --name shiyanlou001 ubuntu /bin/bash
```

上述命令中默认使用 `--network bridge`，即指定 bridge 网络，与下面的命令等同

```
$ docker container run -itd --name shiyanlou001 --network bridge ubuntu /bin/bash
```


创建后，再次查看 bridge 的信息：

```

shiyancelou:~/ $ docker container run -itd --name shiyancelou001 ubuntu /bin/bash
6d3b8e96345df145686a8bc953d0f50243099a673c5c985842fdf7ad8c0ed041
shiyancelou:~/ $
shiyancelou:~/ $ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "43baf0b22ca7e6c3bdf52f859e1353f8baa628885a0e66952eddf0ba32ce10",
    "Created": "2018-01-10T13:22:00.234970024+08:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "192.168.0.0/20",
          "Gateway": "192.168.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "Containers": {
      "6d3b8e96345df145686a8bc953d0f50243099a673c5c985842fdf7ad8c0ed041": {
        "Name": "shiyancelou001",
        "EndpointID": "dee0c05270ae5671681c8e0ad9479807f102d64d5fe145d9f01894684e4443d1",
        "MacAddress": "02:42:c0:a8:00:02",
        "IPv4Address": "192.168.0.2/20",
        "IPv6Address": ""
      }
    }
  }
]

```

该容器的网络信息



这时可以查看到相应的容器的网络信息，该容器在连接到 bridge 网络后，会从子网的地址池中获得一个 IP 地址，即上图中的 192.168.0.2。

使用 `docker container attach shiyancelou001` 命令，也可查看相应的地址信息：

```

shiyancelou:~/ $
shiyancelou:~/ $ docker container attach shiyancelou001
root@6d3b8e96345d:/#
root@6d3b8e96345d:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:c0:a8:00:02
          inet addr:192.168.0.2  Bcast:0.0.0.0  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

root@6d3b8e96345d:/# %
shiyancelou:~/ $
shiyancelou:~/ $

```

如果提示没有找到 `ifconfig` 命令，可以通过如下命令安装：

动手实战学 Docker (/courses/498)

```
$ sudo apt update
$ sudo apt install net-tools
```

并且对于连接到默认的 bridge 之间的容器可以通过 IP 地址互相通信。例如我们启动一个 shiyanlou002 的容器，它可以与 shiyanlou001 通过 IP 地址进行通信。

```
shiyanlou:~/ $ docker container run -it --name shiyanlou002 ubuntu /bin/bash
root@5088ecf032da:/#
root@5088ecf032da:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:c0:a8:00:03
          inet addr:192.168.0.3  Bcast:0.0.0.0  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

root@5088ecf032da:/#
root@5088ecf032da:/# ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.193 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.101 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.119 ms
^C
--- 192.168.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.101/0.137/0.193/0.042 ms
root@5088ecf032da:/#
```



仅将文本发送到当前终端卡

如果提示没有找到 `ping` 命令，可使用如下命令安装：

```
$ sudo apt update
$ sudo apt install iputils-ping
```

其具体的实现原理可以参考链接 Linux 上的基础网络设备

(https://www.ibm.com/developerworks/cn/linux/1310_xiawc_networkdevice/index.html)，以及涉及到网桥的工作原理

(<https://www.jianshu.com/p/9070f4bfeddf>)

上述的操作我们通过 `ping` 命令演示了 IP 相关的内容。但是对于应用程序来讲，如果需要在外部进行访问，我们还会涉及到端口的使用，而 Docker 对于 bridge 网络使用端口的方式为设置端口映射，通过 iptables 实现。

下面我们通过 iptables 来为大家演示 docker 实现端口映射的方式，主要针对 nat 表和 filter 表。
动手实战学 Docker (/courses/498)

1. 首先删除掉上面创建的两个容器。这里不再给出具体的命令
2. 这时，我们查看 nat 表的转发规则，使用如下命令：

```
$ sudo iptables -t nat -nvl
```

3. 由于此时并未创建 docker 容器，nat 表中没有什么特殊的规则。接下来，我们使用上一节构建的 shiyanlou:1.0 镜像创建一个容器 shiyanlou001，并将本机的端口 10001 映射到容器中的 80 端口上，在浏览器中可以通过 localhost:10001 访问容器 shiyanlou001 的 apache 服务，命令如下：

```
$ docker run -d -p 10001:80 --name shiyanlou001 shiyanlou:1.0
```

docker run 命令的 -p 参数是通过端口映射的方式，将容器的端口发布到主机的端口上。其使用格式为 -p ip:hostPort:containerPort。并且还可以指定范围，例如 -p 10001-10100:1-100，代表将容器 1-100 的端口映射到主机上的 10001-10100 端口上，两者一一对应。

4. 创建成功后，我们可以在浏览器中输入 localhost:10001 访问到容器 shiyanlou001 的 apache 服务，并查看此时 iptables 中 nat 表和 filter 表的规则，其中分别新增了一条比较重要的内容，如下图所示：

```
shiyanlou:~/ $ sudo iptables -t nat -nvl
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination
 106  5544 DOCKER    all  --  *      *        0.0.0.0/0  0.0.0.0/0          ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination
   2   120 DOCKER    all  --  *      *        0.0.0.0/0  !127.0.0.0/8       ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination
   90  5629 MASQUERADE all  --  *      *        !docker0  192.168.0.0/20     0.0.0.0/0
    0     0 MASQUERADE tcp  --  *      *        192.168.0.2  192.168.0.2       tcp dpt:80

Chain DOCKER (2 references)
 pkts bytes target    prot opt in     out     source    destination
    1    84 RETURN    all  --  !docker0 *        0.0.0.0/0  0.0.0.0/0
    7   364 DNAT     tcp  --  !docker0 *        0.0.0.0/0  0.0.0.0/0       tcp dpt:10001 to:192.168.0.2:80
```

shiyanlou:~/ \$
shiyanlou:~/ \$


```

shiyanolou:~/ $ sudo iptables -nvL
Chain INPUT (policy ACCEPT 318K packets, 140M bytes)
 pkts bytes target    prot opt in     out     source         destination

Chain FORWARD (policy DROP 30 packets, 1560 bytes)
 pkts bytes target    prot opt in     out     source         destination
182K 268M DOCKER-ISOLATION all -- *      *        0.0.0.0/0      0.0.0.0/0
96722 263M ACCEPT    all -- *      docker0 0.0.0.0/0      0.0.0.0/0      ctstate RELATED,ESTABLISHED
88 4608 DOCKER    all -- *      docker0 0.0.0.0/0      0.0.0.0/0
84860 5153K ACCEPT    all -- docker0 !docker0 0.0.0.0/0      0.0.0.0/0
1 84 ACCEPT    all -- docker0 docker0 0.0.0.0/0      0.0.0.0/0

Chain OUTPUT (policy ACCEPT 259K packets, 115M bytes)
 pkts bytes target    prot opt in     out     source         destination

Chain DOCKER (1 references)
 pkts bytes target    prot opt in     out     source         destination
7 364 ACCEPT    tcp -- !docker0 docker0 0.0.0.0/0      192.168.0.2      tcp dpt:80

Chain DOCKER-ISOLATION (1 references)
 pkts bytes target    prot opt in     out     source         destination
182K 268M RETURN    all -- *      *        0.0.0.0/0      0.0.0.0/0
shiyanolou:~/ $
shiyanolou:~/ $

```



5. 接下来，再次使用镜像 `shiyanlou:1.0` 来启动一个容器 `shiyanlou002`，这次我们不指定端口映射，通过手动修改 `nat` 表的方式来模拟实现：

```
$ docker run -d --name shiyanlou002 shiyanlou:1.0
```

6. 获取容器 `shiyanlou002` 的 `ip` 地址，如果按步骤操作此 `ip` 为 `192.168.0.3`。此时我们想通过主机的 `10002` 端口访问容器 `shiyanlou002` 的 `80` 端口，就可以添加一条规则：

```
# 添加一条规则，大致解释为将从非 docker0 接口上，目的端口为 10002 的 tcp 报文，修改其目的地址为 192.168.0.3:80
```

```
$ sudo iptables -t nat -A DOCKER ! -i docker0 -p tcp --dport 10002 -j DNAT --to-destination 192.168.0.3:80
```

7. 添加成功后我们在主机发出的本地公网或内网 `ip` 加端口号 `10002` 的请求会被定位到 `192.168.0.3:80` 上，但是在将请求转发到 `docker0` 网桥上时，对于默认的 `filter` 表中的 `FORWARD` 链的规则是 `DROP`，因此我们还需要在 `filter` 表中设置相应的规则：

```
$ sudo iptables -t filter -A FORWARD ! -i docker0 -o docker0 -p tcp -d 192.168.0.3 -j ACCEPT --dport 80
```

或者你也可以选择将其加到由 `docker` 定义的 `DOCKER` 链中，上面的命令和下面的命令选择其中的一个即可

```
$ sudo iptables -t filter -A DOCKER ! -i docker0 -o docker0 -p tcp -d 192.168.0.3 -j ACCEPT --dport 80
```

9. 此时我们就能够通过 `192.168.0.3:80` 访问容器 `shiyanlou002` 中的 `apache` 服务了。即通过 `iptables` 的方式实现了容器 `shiyanlou002` 上 `80` 端口到主机 `10002` 端口的映射。

10. 最后，为了不影响后面实验的进行，这里我们删除掉手动添加的规则，并删除容器。

删除手动添加的规则可使用如下方法：

🔍 点击查看规则

🔗 动手实战学 Docker (/courses/498)

```
$ sudo iptables -t nat -nvL --line-numbers
```

#比如删除 DOCKER 链的第 2 条规则

```
$ sudo iptables -t nat -D DOCKER 2
```



```
#查看 filter 规则
```

```
$ sudo iptables -nvL --line-numbers
```

#比如删除 DOCKER 链第 1 条规则

```
$ sudo iptables -D DOCKER 1
```

4.2 自定义网络

对于默认的 bridge 网络来说，使用端口可以通过端口映射的方式来实现，并且在上面的内容中我们也演示了容器之间通过 IP 地址互相进行通信。但是对于默认的 bridge 网络来说，每次重启容器，容器的 IP 地址都是会发生变化的，因为对于默认的 bridge 网络来说，并不能在启动容器的时候指定 ip 地址，在启动单个容器时并不容易看到这一区别。

旧版的容器互联

容器间都是通过通过在 /etc/hosts 文件中添加相应的解析，通过容器名，别名，服务名等来识别需要通信的容器。

这里，我们启动两个容器，来演示旧的容器互联：

1. 首先启动一个名为 shiyanlou001 的容器，使用镜像 busybox：

```
$ docker run -it --rm --name shiyanlou001 busybox /bin/sh
```

2. 这时打开一个新的终端，启动一个名为 shiyanlou002 的容器，并使用 --link 参数与容器 shiyanlou001 互联。

```
$ docker run -it --rm --name shiyanlou002 --link shiyanlou001 busybox /bin/sh
```

docker run 命令的 --link 参数的格式为 --link <name or id>:alias。格式中的 name 为容器名，alias 为别名。即可以通过 alias 访问到该容器。

如下图所示，左侧为 shiyanlou001，右侧为 shiyanlou002：


```

shiyanlou:~/ $ docker run -it --name shiyanlou001 --rm busybox /bin/sh
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1
    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
183: eth0@if184: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:c0:a8:00:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.2/20 scope global eth0
        valid_lft forever preferred_lft forever
/ # 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
192.168.0.2    c172b8e571e1
/ #

shiyanlou:~/ $ docker run -it --rm --name shiyanlou002 --link shiyanlou001 busybox /bin/sh
/ # ping -c 3 shiyanlou001
PING shiyanlou001 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=64 time=0.148 ms
64 bytes from 192.168.0.2: seq=1 ttl=64 time=0.130 ms
64 bytes from 192.168.0.2: seq=2 ttl=64 time=0.126 ms

--- shiyanlou001 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.126/0.134/0.148 ms
/ # 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
192.168.0.2    shiyanlou001 c172b8e571e1
192.168.0.3    492de4eb0451
/ #
  
```

通过别名访问容器 shiyanlou001

能够访问因为使用 --link 后 /etc/hosts 配置了相应的解析

3. 如果此时 shiyanlou001 容器退出，这时我们启动一个 shiyanlou003，再次启动一个 shiyanlou001：

```

$ docker run -itd --name shiyanlou003 --rm busybox /bin/sh

$ docker run -it --name shiyanlou001 --rm busybox /bin/sh
  
```

按照顺序分配的原则，此时 shiyanlou003 的 IP 地址为 192.168.0.2，容器 shiyanlou001 的 IP 地址为 192.168.0.4。并且此时容器 shiyanlou002 中 /etc/hosts 文件的解析依旧不变，所以不能获取到正确的解析：

```

shiyanlou:~/ $ docker run -itd --name shiyanlou003 --rm busybox /bin/sh
/sh
eb7f6c3f2bde5704871aa91314c8864164c745b7077b3a494a7273b8e9f0602f
shiyanlou:~/ $ [10:46:54]
shiyanlou:~/ $ docker run -it --name shiyanlou001 --rm busybox /bin/sh
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1
    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
199: eth0@if200: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:c0:a8:00:04 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.4/20 scope global eth0
        valid_lft forever preferred_lft forever
/ #
/ #
/ #
/ #

shiyanlou:~/ $ docker run -it --rm --name shiyanlou002 --link shiyanlou001 busybox /bin/sh
/ # ping shiyanlou001
PING shiyanlou001 (192.168.0.2): 56 data bytes
64 bytes from 192.168.0.2: seq=0 ttl=64 time=0.155 ms
64 bytes from 192.168.0.2: seq=1 ttl=64 time=0.142 ms
64 bytes from 192.168.0.2: seq=2 ttl=64 time=0.132 ms
^C

--- shiyanlou001 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.132/0.143/0.155 ms
/ #
/ #
/ #
/ #
/ #
/ #
/ #
  
```

此时 shiyanlou001 的 IP 为 192.168.0.4

此时 shiyanlou002 容器中解析依旧是 192.168.0.2

如上所示，旧的容器 shiyanlou002 通过 --link 连接到 shiyanlou001。而在 shiyanlou001 重启后，由于 IP 地址的变化，此时 shiyanlou002 并不能正确的访问到 shiyanlou001。

除了使用 --link 链接的方式来达到容器间互联的效果，在 docker 中，容器间的通信更应该使用的是自定义网络。

自定义网络

docker 在安装时会默认创建一个桥接网络，除了使用默认网络之外，我们还可以创建自己的 bridge 或 overlay 网络。

如下所示，我们创建一个名为 network1 的桥接网络，简单命令如下：

```
$ docker network create network1
```

```
$ docker network ls
```

```
shiyanolou:~/ $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
aaaa8ebfd2bf        bridge             bridge             local
04dcaff0d5a4        host              host              local
4ce7c23db6fc        none              null              local
shiyanolou:~/ $
shiyanolou:~/ $ docker network create network1
a0714f9f4d8f0a96705b61485c593f0f51ab7f47342a2a6f10247ca5f6335ebc
shiyanolou:~/ $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
aaaa8ebfd2bf        bridge             bridge             local
04dcaff0d5a4        host              host              local
a0714f9f4d8f        network1           bridge             local
4ce7c23db6fc        none              null              local
shiyanolou:~/ $
shiyanolou:~/ $
```

创建成功后，可以使用 ifconfig 或者 ip addr show 命令查看该桥接网络的网络接口信息，如下所示：

```
shiyanolou:~/ $ ip addr show [11:09:00]
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    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
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:16:3e:00:7a:33 brd ff:ff:ff:ff:ff:ff
    inet 10.174.32.81/21 brd 10.174.39.255 scope global eth0
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:16:3f:00:8a:56 brd ff:ff:ff:ff:ff:ff
    inet 120.55.125.199/22 brd 120.55.127.255 scope global eth1
        valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:a0:25:e1:e1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.1/20 scope global docker0
        valid_lft forever preferred_lft forever
201: br-a0714f9f4d8f: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:06:9c:13:cc brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.1/20 scope global br-a0714f9f4d8f
        valid_lft forever preferred_lft forever
shiyanolou:~/ $
shiyanolou:~/ $
```



而对于该网络的详细信息可以通过 `docker network inspect network1` 命令来查看，如下图所示：

```
shiyanolou:~/ $ docker network inspect network1 [11:11:44]
[
  {
    "Name": "network1",
    "Id": "a0714f9f4d8f0a96705b61485c593f0f51ab7f47342a2a6f10247ca5f6335ebc",
    "Created": "2018-01-29T11:07:09.750893762+08:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.16.0/20",
          "Gateway": "192.168.16.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

→ 子网信息



其相应的网络接口名称和子网都是由 docker 随机生成，当然，我们也可以手动指定：

👉 动手实战学 Docker (/courses/498)

```
# 首先删除掉刚刚创建的 network1
$ docker network rm network1

# 再次创建 network1，指定子网
$ docker network create -d bridge --subnet=192.168.16.0/24 --gateway=192.168.16.1 network1
```

此时，我们可以运行一个容器 shiyanlou001，指定其网络为 network1，使用 --network network1：


```
$ docker run -it --name shiyanlou001 --network network1 --rm busybox /bin/sh
```

```
shiyanlou:~/ $ docker run -it --name shiyanlou001 --network network1 --rm busybox /bin/sh
sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:C0:A8:10:02
          inet addr:192.168.16.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

/ #
/ #
/ #
```

ip 地址从指定的子网中获取




使用 exit 退出该容器使其自动删除，这时我们再次创建该容器，但是不指定其 --network：

```
$ docker run -it --name shiyanlou001 --rm busybox /bin/sh
```

```
shianlou:~/ $ docker run -it --name shianlou001 --rm busybox /bin/sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:C0:A8:00:02
          inet addr:192.168.0.2  Bcast:0.0.0.0  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

/ #
```



此时，该容器连接到默认的 bridge 网络，这时，可以新打开一个终端，在其中运行如下命令，将 shianlou001 连接到 network1 网络中：

```
# 在新打开的终端中运行，将容器 shianlou001 连接到 network1 网络中
$ docker network connect network1 shianlou001

# 这时再次在容器 `shianlou001` 中使用 `ifconfig` 命令
```



```
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:C0:A8:00:02
          inet addr:192.168.0.2  Bcast:0.0.0.0  Mask:255.255.240.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:C0:A8:10:02
          inet addr:192.168.16.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

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

/ #
/ #
/ #
```

eth1 连接到 network1



如上图所示，出现了一个 eth1 接口，此时，eth0 连接到默认的 bridge 网络，eth1 连接到 network1 网络。

对于自定义的网络来说，docker 嵌入的 DNS 服务支持连接到该网络的容器名的解析。这意味着连接到同一个网络的容器都可以通过容器名去 ping 另一个容器。

如下所示，启动两个容器，连接到 network1：

```
$ docker run -itd --name shiyanlou_1 --network network1 --rm busybox /bin/sh

$ docker run -it --name shiyanlou_2 --network network1 --rm busybox /bin/sh
```

启动之后，由于上述的两个容器都是连接到 network1 网络，所以可以通过容器名 ping 通：


```

shiyanolou:~/ $ docker run -itd --name shiyanolou_1 --network network1 --rm busybox /bin/
sh
ae435d63edc7a036034cf8cae21efc0448c692044c6a4a5542f3bb450f066953
shiyanolou:~/ $ docker run -it --name shiyanolou_2 --network network1 --rm busybox /bin/s
h
/ #
/ # ping shiyanolou_1
PING shiyanolou_1 (192.168.16.2): 56 data bytes
64 bytes from 192.168.16.2: seq=0 ttl=64 time=0.119 ms
64 bytes from 192.168.16.2: seq=1 ttl=64 time=0.136 ms
^C
--- shiyanolou_1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.119/0.127/0.136 ms
/ #
/ # exit
shiyanolou:~/ $
shiyanolou:~/ $ docker stop shiyanolou_1
shiyanolou_1
shiyanolou:~/ $

```

除此之外，在用户自定义的网络中，是可以通过 `--ip` 指定 IP 地址的，而在默认的 bridge 网络不能指定 IP 地址：

```

# 连接到 network1 网络，运行成功
$ docker run -it --network network1 --ip 192.168.16.100 --rm busybox /bin/sh

# 连接到默认的 bridge 网络，下面的命令运行失败
$ docker run -it --rm busybox --ip 192.168.0.100 --rm busybox /bin/sh

```

```

shiyanolou:~/ $ docker run -it --network network1 --ip 192.168.16.100 --rm busybox /bin/
sh
/ #
/ # ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1
    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
219: eth0@if220: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:c0:a8:10:64 brd ff:ff:ff:ff:ff:ff
    inet 192.168.16.100/24 scope global eth0
        valid_lft forever preferred_lft forever
/ #
/ # exit
shiyanolou:~/ $
shiyanolou:~/ $ docker run -it --ip 192.168.0.100 --rm busybox /bin/sh
docker: Error response from daemon: user specified IP address is supported on user defi
ned networks only.
ERRO[0000] error getting events from daemon: context canceled
shiyanolou:~/ $
shiyanolou:~/ $
shiyanolou:~/ $

```

提示仅支持用户自定义的网络

4.3 host 和 none

host 网络，容器可以直接访问主机上的网络。

例如，我们启动一个容器，指定网络为 host：

👉 动手实战学Docker (/courses/498)

```
$ docker run -it --network host --rm busybox /bin/sh
```

如下所示，该容器可以直接访问主机上的网络：

```
shiyanolou:~/ $ docker run -it --network host --rm busybox /bin/sh
/ #
/ # ifconfig
br-eb046e748d51 Link encap:Ethernet HWaddr 02:42:8A:F6:D6:11
    inet addr:192.168.16.1 Bcast:0.0.0.0 Mask:255.255.255.0
    UP BROADCAST MULTICAST MTU:1500 Metric:1
    RX packets:1 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0
    RX bytes:28 (28.0 B) TX bytes:0 (0.0 B)

docker0    Link encap:Ethernet HWaddr 02:42:A0:25:E1:E1
    inet addr:192.168.0.1 Bcast:0.0.0.0 Mask:255.255.240.0
    UP BROADCAST MULTICAST MTU:1500 Metric:1
    RX packets:132235 errors:0 dropped:0 overruns:0 frame:0
    TX packets:147390 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0
    RX bytes:8209284 (7.8 MiB) TX bytes:415278076 (396.0 MiB)

eth0       Link encap:Ethernet HWaddr 00:16:3E:00:7A:33
    inet addr:10.174.32.81 Bcast:10.174.39.255 Mask:255.255.248.0
    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
    RX packets:210626 errors:0 dropped:0 overruns:0 frame:0
    TX packets:173033 errors:0 dropped:0 overruns:0 carrier:0
```

none 网络，容器中不提供其它网络接口。

```
$ docker run -it --network none --rm busybox /bin/sh
```

```
shiyanolou:~/ $ docker run -it --network none --rm busybox /bin/sh
/ #
/ # ifconfig
lo         Link encap:Local Loopback
    inet addr:127.0.0.1 Mask:255.0.0.0
    UP LOOPBACK RUNNING MTU:65536 Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1
    RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

/ #
/ #
```



5. 总结

本节实验中我们学习了以下内容：

1. docker 容器端口映射
2. 自定义网络实现容器互联
3. host 和 none 网络的使用

请务必保证自己能够动手完成整个实验，只看文字很简单，真正操作的时候会遇到各种各样的问题，解决问题的过程才是收获的过程。

**本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。*

本课程为训练营课程，查看完整
内容请

[加入训练营 \(/courses/498\)](/courses/498)

上一节：[Docker 存储管理 \(/courses/498/labs/1706/document\)](/courses/498/labs/1706/document)

下一节：[编写 DockerFile \(/courses/498/labs/1708/document\)](/courses/498/labs/1708/document)