Docker Networking

For Docker containers to communicate with each other and the outside world via the host machine, there has to be a layer of networking involved. Docker supports different types of networks, each fit for certain use cases.

When Docker is installed, a default bridge network named docker0 is created. Each new Docker container is automatically attached to this network, unless a custom network is specified.

Besides **docker0** , two other networks get created automatically by Docker: **host** (no isolation between host and containers on this network, to the outside world they are on the same network) and **none** (attached containers run on container-specific network stack).

Network drivers

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

bridge: The default network driver. If you don't specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. host is only available for swarm services on Docker 17.06 and higher.

overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.

macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.

none: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.

```
[ec2-user@ip-172-31-3-152 ~]$ ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:A5:A9:B5:43
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          inet6 addr: fe80::42:a5ff:fea9:b543/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3106 (3.0 KiB)  TX bytes:976 (976.0 b)

eth0      Link encap:Ethernet  HWaddr 0A:C0:FA:9B:45:1E
          inet addr:172.31.3.152  Bcast:172.31.15.255  Mask:255.255.240.0
          inet6 addr: fe80::8c0:faff:fe9b:451e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
          RX packets:507 errors:0 dropped:0 overruns:0 frame:0
          TX packets:536 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:67196 (65.6 KiB)  TX bytes:57740 (56.3 KiB)
```

Check the default network list in your machine.

docker network ls

```
[ec2-user@ip-172-31-3-152 ~]$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c3effe4048a7        bridge              bridge              local
f03a7e936764        host                host                local
a4e16ce92147        none                null                local
[ec2-user@ip-172-31-3-152 ~]$
```

The **docker0** interface is a virtual Ethernet bridge that connects our containers and the local host network. If we look further at the other interfaces on our Docker host, we'll find a series of interfaces starting with **veth**.

Every time Docker creates a container, it creates a pair of peer interfaces that are like opposite ends of a pipe (i.e., a packet sent on one will be received on the other). It gives one of the peers to the container to become its **eth0** interface and keeps the other peer, with a unique name like **veth***, out on the host machine. You can think of a **veth** interface as one end of a virtual network cable. One end is plugged into the **docker0** bridge, and the other end is plugged into the container. By binding every **veth*** interface to the **docker0** bridge, Docker creates a virtual subnet shared between the host machine and every Docker container.

Check veth interface using below command.

$ sudo yum install bridge-utils -y

$ brctl show

```
[ec2-user@ip-172-31-3-152 ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242a5a9b543       no
```

Here there is no container on this bridge.

Note: there is no veth* interfaces means there is no containers are running for confirmation check docker ps command.

Let run one container and see.

$ sudo docker run -t -i ubuntu /bin/bash

If you want to check **ip addr show eth0** you need install network tools ..

$ apt-get update && apt install -y iputils-ping && apt install iproute2 -y

root@b78a7ea8f398:/# ip addr show eth0

```
root@b78a7ea8f398:/# ip addr show eth0
8: eth0@if9: <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
```

Come out from the container without stopping using **ctrl+p+q**

Now check **brtcl show** command this time we are able see one veth interface. Because of one container is running.

[ec2-user@ip-172-31-3-152 ~]$ brctl show

```
[ec2-user@ip-172-31-3-152 ~]$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242a5a9b543       no              vethf521e20
```

And do ifconfig in host machine

Do docker network inspect bridge for more info

[ec2-user@ip-172-31-3-152 ~]$ docker network inspect bridge

$ apt-get update && apt install -y iputils-ping && apt install iproute2 -y

```
[ec2-user@ip-172-31-3-152 ~]$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "c3effe4048a7ecbc0445520e5cdf35dd1b7c6455c5da4b941a31467990bdc529",
        "Created": "2019-09-26T09:52:49.508589772Z",
        "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,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "b78a7ea8f398f91c071313cbf4ec44094d6be49edfaa7e78715e6a4847f30fb9": {
                "Name": "wonderful_feynman",
                "EndpointID": "4041c92124bece02db8893b42d7d20ace13d934fd118f5eb1372e2ff173e0d9b",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "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": {}
    }
]
```

How many containers are running in that network it will show all containers.

And run another container

[ec2-user@ip-172-31-3-152 ~]$ docker run --rm -it --name myubuntu2 ubuntu

Ping the first container from second container

root@6e847848b743:/# **ping 172.17.0.2**

PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.

64 bytes from 172.17.0.2: icmp_seq=1 ttl=255 time=0.060 ms

64 bytes from 172.17.0.2: icmp_seq=2 ttl=255 time=0.050 ms

64 bytes from 172.17.0.2: icmp_seq=3 ttl=255 time=0.050 ms

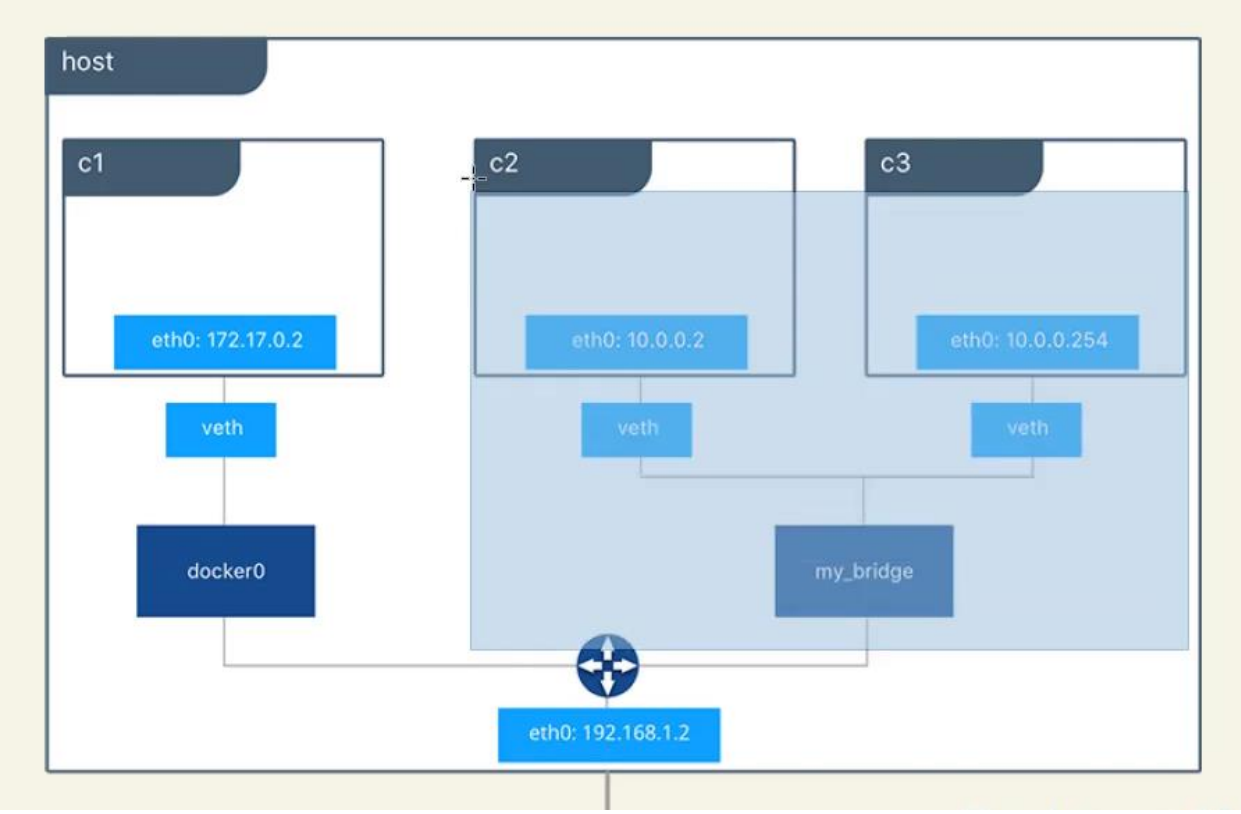
64 bytes from 172.17.0.2: icmp_seq=4 ttl=255 time=0.052 ms

^C

It will connect because of both are in the same network.

Create custom network - bridge network

$ docker network create mynetwork --subnet=10.0.0.1/16 --gateway=10.0.10.100



$ docker network ls

```
[ec2-user@ip-172-31-3-152 ~]$ docker network create  mynetwork --subnet=10.0.0.1/16 --gateway=10.0.10.100
59fcc8909a03518d53ec2c784d0b7103bbae03f3aa2faa6e67636e353d690264
[ec2-user@ip-172-31-3-152 ~]$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c3effe4048a7        bridge              bridge              local
f03a7e936764        host                host                local
59fcc8909a03        mynetwork           bridge              local
a4e16ce92147        none                null                local
```

Then inspect and check

$ docker network inspect mynetwork


```
[ec2-user@ip-172-31-3-152 ~]$ docker network inspect mynetwork
[
    {
        "Name": "mynetwork",
        "Id": "59fcc8909a03518d53ec2c784d0b7103bbae03f3aa2faa6e67636e353d690264",
        "Created": "2019-09-26T11:24:31.880905885Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.0.1/16",
                    "Gateway": "10.0.10.100"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```


Then create a container with our new network

$ docker run -it --net mynetwork --name myubuntu3 ubuntu

Once login the container try to ping the default bridge container ip…nope it won't work because both are different network.

To get the particular container ip we can use docker inspect command to get ipaddress like below.

[ec2-user@ip-172-31-3-152 ~]$ docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' 5676df12b29d
172.17.0.2

To login the container you can you docker exec command

[ec2-user@ip-172-31-3-152 ~]$ docker exec -it 72cf3c6c1058 bash

Then try to ping the default bridge container ip from inside custom network container.

Note ping command not works need to install ping utilities.

$ apt-get update && apt install -y iputils-ping && apt install iproute2 -y

root@72cf3c6c1058:/# ping 172.17.0.2

```
root@72cf3c6c1058:/# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
^C
--- 172.17.0.2 ping statistics ---
81 packets transmitted, 0 received, 100% packet loss, time 81899ms
```

Shhhhhh….nope it won't work because both are different network.

Then what we do we need to add the out custom network container to default network bridge.

[ec2-user@ip-172-31-3-152 ~]$ docker network connect bridge myubuntu3

Then login my custom network docker container and ping to default bridge docker container ip.

```
root@1c6a3e45c997:/# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=255 time=0.064 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=255 time=0.049 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=255 time=0.051 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=255 time=0.061 ms
64 bytes from 172.17.0.2: icmp_seq=5 ttl=255 time=0.052 ms
64 bytes from 172.17.0.2: icmp_seq=6 ttl=255 time=0.069 ms
64 bytes from 172.17.0.2: icmp_seq=7 ttl=255 time=0.051 ms
^C
--- 172.17.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6126ms
rtt min/avg/max/mdev = 0.049/0.056/0.069/0.011 ms
```

yessss…its connecting. Now both network connected each other.

even we can reconfirm it docker inspect bridge

[ec2-user@ip-172-31-3-152 ~]$ docker inspect bridge

```
[ec2-user@ip-172-31-3-152 ~]$ docker inspect bridge
[
    {
        "Name": "bridge",
        "Id": "c3effe4048a7ecbc0445520e5cdf35dd1b7c6455c5da4b941a31467990bdc529",
        "Created": "2019-09-26T09:52:49.508589772Z",
        "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,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "1c6a3e45c9971a947c4aeb1b573f5b16e029c44b67426a574d2677d92c933bf0": {
                "Name": "myubuntu3",
                "EndpointID": "116cc817cd63be03f8db0034457efad4a0b8ab916141d29a69a63e6b65127c55",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "5676df12b29dd29a1d6cbf621a151e3c6881453fc2a8dc1ff675a92d5e563afb": {
                "Name": "myubuntu1",
                "EndpointID": "d5f96c71c0f080b3d4980076c80bf19715c47d830dacba4d14cd56bd691a3360",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
```

we are done….

we can start the container's with different modes.

1. **--net=bridge →** this is the default mode that we just saw. so we can start container like below.

   $ docker run -it --net=bridge ubuntu

2. **--net=host** → with this option docker does not create any network it will use host network.
   $ docker run -it --net=host ubuntu

3. **--net=container:<containername>_or<containerid>** → with this option docker does not create a new network namespace while starting the container but shares it form another container.

   $ docker run -it --name myubuntu ubuntu

   now start another container as follows.

   $ docker run -it --net=container:myubuntu ubuntu

   you will find that both containers have same ip address

4. **--net=none** → with this option, Docker creates the network namespace inside the container but does not configure networking.

Network driver summary

User-defined bridge networks are best when you need multiple containers to communicate on the same Docker host.

Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.

Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.

Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.

Third-party network plugins allow you to integrate Docker with specialized network stacks.

Differences between user-defined bridges and the default bridge

https://docs.docker.com/network/bridge/