Lab: Docker Networking

Introduction:

Docker abstracts the underling networking of the host to provide the networking capability to the containers running. Docker networking allows you to attach a container to as many networks as you like. You can also attach an already running container.

Objective:

- **Use of the default bridge network**
- **Use of the user-defined bridge networks**

1. Ensure that you have logged-in as **root** user with password as **linux**.

1.1 Let's **list** current **networks** available on host.

```
# docker network ls
```

Output:

```
[root@docker-engine ~]#docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
b1ee32f20ac9   bridge    bridge    local
ba91bd2d6e84   host      host      local
8e0b1f08ca38   none      null      local
```

1.2 Let's **inspect** the bridge network by using below command.

```
# docker network inspect bridge
```

Output:

```
[root@docker-engine ~]#docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "b1ee32f20ac97b506535ef09ibe6d593bdd563af3e76d1de3e27134a559c3874",
        "Created": "2023-02-06T11:03:32.232578833+05:30",
        "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": {},
        "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": {}
    }
]
```

1.3 Let's **create two containers** to verify the default bridge network behavior.

```
# docker container run --name server01 -dit centos
```

Output:

```
[root@docker-engine ~]#docker container run --name server01 -dit centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
a1d0c7532777: Already exists
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
7413e49a7cbc3569108a7606e6780168352d9e87bee0d6826fce7b0e34e86a9e
```

```
# docker container run --name server02 -dit centos
```

Output:

```
[root@docker-engine ~]#docker container run --name server02 -dit centos
7c423a65424e98bb522cd44310b8ad55369c08c6f6bb068832456daf621c7ebb
```

1.4 Let's **Inspect** the bridge network to see which containers are connected to it.

```
# docker network inspect bridge | grep Containers -A 15
```

Output:

```
[root@docker-engine ~]#docker network inspect bridge | grep Containers -A 15
        "Containers": {
            "7413e49a7cbc3569108a7606e6780168352d9e87bee0d6826fce7b0e34e86a9e": {
                "Name": "server01",
                "EndpointID": "0412b7ca4553cb504f2a234af2c9c288a604335f8337f70562ee065a8b56d9e5",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "7c423a65424e98bb522cd44310b8ad55369c08c6f6bb068832456daf621c7ebb": {
                "Name": "server02",
                "EndpointID": "49b8b3abf53304f426abe9bee2d55a17c4a555414bfe55eafe99d3addbd6fe3c",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
        },
```

1.5 By using docker **attach** command enter inside **server01** container**,** from within server01, make sure you can connect to the internet by pinging google.com. The **-c 2** flag limits the command to two ping attempts.

```
# docker container attach server01
```

Output:

```
[root@docker-engine ~]#docker container attach server01
[root@7413e49a7cbc /]#
```

```
# ip a s
```

Output:

```
[root@7413e49a7cbc /]# ip a s
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
28: eth0@if29: <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
```

```
# ping -c 2 172.17.0.2
```

Output:

```
[root@7413e49a7cbc /]# ping -c 2 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.117 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.362 ms

--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.117/0.239/0.362/0.123 ms
```

```
# ping -c 2 172.17.0.3
```

Output:

```
[root@7413e49a7cbc /]# ping -c 2 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.289 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.262 ms

--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1039ms
rtt min/avg/max/mdev = 0.262/0.275/0.289/0.021 ms
```

```
# ping -c 2 www.google.com
```

Output:

```
[root@7413e49a7cbc /]# ping -c 2  www.google.com
PING www.google.com (172.217.166.100) 56(84) bytes of data.
64 bytes from maa05s09-in-f4.1e100.net (172.217.166.100): icmp_seq=1 ttl=127 time=10.2 ms
64 bytes from maa05s09-in-f4.1e100.net (172.217.166.100): icmp_seq=2 ttl=127 time=11.3 ms

--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 10.210/10.732/11.254/0.522 ms
```

Note: Press ctrl+p+q to come out of the container gracefully.

1.6 By using docker **attach** command dive inside **server02,** From within server02, make sure you can connect to the internet by pinging google.com. The **-c 2** flag limits the command to two ping attempts.

```
# docker container attach server02
```

Output:

```
[root@docker-engine ~]#docker container attach server02
[root@7c423a65424e /]#
```

```
# ip a s
```

Output:

```
[root@7c423a65424e /]# ip a s
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
30: eth0@if31: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

```
# ping -c 2 172.17.0.2
```

Output:

```
[root@7c423a65424e /]# ping -c 2 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.149 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.055 ms

--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.055/0.102/0.149/0.047 ms
```

```
# ping -c 2 172.17.0.3
```

Output:

```
[root@7c423a65424e /]# ping -c 2 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.043 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.160 ms

--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1036ms
rtt min/avg/max/mdev = 0.043/0.101/0.160/0.059 ms
```

```
# ping -c 2 www.google.com
```

Output:

```
[root@7c423a65424e /]# ping -c 2 www.google.com
PING www.google.com (142.250.67.164) 56(84) bytes of data.
64 bytes from bom12s07-in-f4.1e100.net (142.250.67.164): icmp_seq=1 ttl=127 time=26.2 ms
64 bytes from bom12s07-in-f4.1e100.net (142.250.67.164): icmp_seq=2 ttl=127 time=28.2 ms

--- www.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 26.208/27.179/28.151/0.985 ms
```

Note: Press ctrl+pq to come out of the container gracefully.

1.7 Let's **clean-up** the containers, by executing the below command.

```
# docker container rm `docker container ls -a -q` -f
```

Output:

```
[root@docker-engine ~]#docker container rm `docker container ls -a -q` -f
7c423a65424e
7413e49a7cbc
```

2. Customized or user-defined Bridge Networks.

2.1 Let's **create** a bridge by the name dev-network.

```
# docker network create --driver bridge dev-network
```

Output:

```
[root@docker-engine ~]#docker network create --driver bridge dev-network
9c1be601e9fb1c1f099cc416031269da26846ba1a3adcaba2a66fda023dd7bf7
```

2.2 Let's **list** the docker networks.

```
# docker network ls
```

Output:

```
[root@docker-engine ~]#docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
b1ee32f20ac9        bridge              bridge              local
9c1be601e9fb        dev-network         bridge              local
ba91bd2d6e84        host                host                local
8e0b1f08ca38        none                null                local
```

2.3 Let's Inspect the **dev-network** network. This shows you its IP address and the fact that no containers are connected to it.

```
# docker network inspect dev-network
```

Output:

```
[root@docker-engine ~]#docker network inspect dev-network
[
    {
        "Name": "dev-network",
        "Id": "9c1be601e9fb1c1f099cc416031269da26846ba1a3adcaba2a66fda023dd7bf7",
        "Created": "2023-02-06T18:15:40.740695085+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

2.4 Let's create **four containers**. Notice the --network flags. You can only connect to one network during the docker run command, so you need to use docker network connect afterwards to connect one of the container to the bridge network as well.

```
# docker container run --name dev-container1 -dit --network
dev-network centos
```

Output:

```
[root@docker-engine ~]#docker container run --name dev-container1 -dit --network dev-network centos
8854c7ec1ed88339e6a02e5758e25485b37723b9692716d3ace8ddd868cad5f8
```

```
# docker container run --name dev-container2 -dit --network
dev-network centos
```

Output:

```
[root@docker-engine ~]#docker container run --name dev-container2 -dit --network dev-network centos
346d3a3c830c9554941083f9746d3e134473695e6212710e483e32e79ba160d1
```

```
# docker container run --name dev-container3 -dit --network
dev-network centos
```

Output:

```
[root@docker-engine ~]#docker container run --name dev-container3 -dit --network dev-network centos
1f3228f47caa2bc920960837decd004ec944a983cf75d9daeed6a297759fc896
```

```
# docker container run --name centos-4 -dit centos
```

Output:

```
[root@docker-engine ~]#docker container run --name centos-4 -dit centos
e3ebde87f62a446c3848ffc11638f66df07287bfca2e1213e54089f339a1e270
```

Note: In above steps we have create a bunch of containers where for some containers we have specified network bridge.

```
# docker network connect bridge dev-container3
```

2.5 Let's **list** the created containers by using bellow command.

```
# docker container ls -a
```

Output:

```
[root@docker-engine ~]#docker container ls -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS              PORTS     NAMES
e3ebde87f62a   centos    "/bin/bash"   About a minute ago   Up About a minute             centos-4
1f3228f47caa   centos    "/bin/bash"   4 minutes ago        Up 4 minutes                  dev-container3
346d3a3c830c   centos    "/bin/bash"   4 minutes ago        Up 4 minutes                  dev-container2
8854c7ec1ed8   centos    "/bin/bash"   6 minutes ago        Up 6 minutes                  dev-container1
```

2.6 Let's **Inspect** the dev-network to verify the containers.

```
# docker network inspect dev-network | grep -i name -A 4
```

Output:

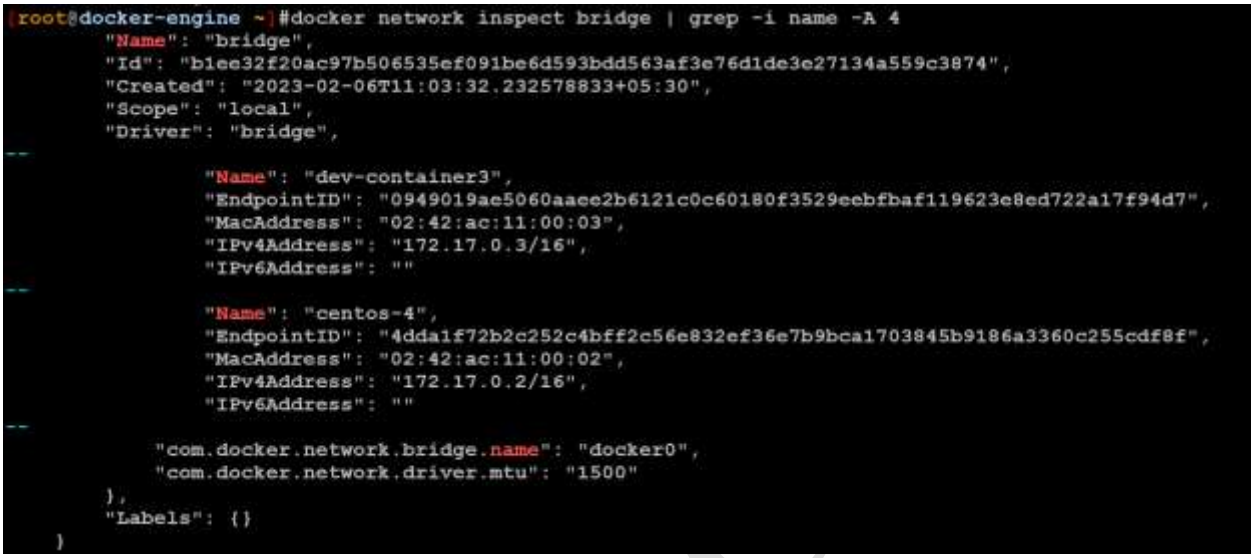
```
[root@docker-engine ~]#docker network inspect dev-network | grep -i name -A 4
        "Name": "dev-network",
        "Id": "9c1be601e9fb1c1f099cc416031269da26846ba1a3adcaba2a66fda023dd7bf7",
        "Created": "2023-02-06T18:15:40.740695085+05:30",
        "Scope": "local",
        "Driver": "bridge",
--
                "Name": "dev-container3",
                "EndpointID": "e05495a2405b1f48fa7c9d05a1e99547b72dd5eb50d2c8d8d86e40353700f54e",
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
--
                "Name": "dev-container2",
                "EndpointID": "090bb70f381674f244cb8b27a9a5675b184b5c9665862e6409ea8e5c78bec158",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
--
                "Name": "dev-container1",
                "EndpointID": "af3acd70a04a32fae94e720daeb8c78fe9c7b973fef1122a64ebb7089a91e77d",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
```

Note: Container's dev-container1, dev-container2, and dev-container3 are connected to the dev-network network.

2.7 Let's **Inspect** the bridge network to verify the containers attached to the default bridge.

```
# docker network inspect bridge | grep -i name -A 4
```

Output:



Note: Container's dev-container3, and centos-4 are connected to the default bridge network.

Info: On **user-defined** networks like **"dev-network"**, containers can not only communicate by IP address but can also resolve a container name to an **IP address**. This capability is called **automatic service discovery**.

2.8 Let's attach to **dev-container1** and test this out. **dev-container1** should be able to resolve **dev-container2** and **dev-container3**(and **dev-container1**, itself) to IP addresses.

```
# docker container attach dev-container1
```

Output:


```
# ping -c 2 dev-container1
```

Output:

```
[root@8854c7ec1ed8 /]# ping -c 2 dev-container1
PING dev-container1 (172.18.0.2) 56(84) bytes of data.
64 bytes from 8854c7ec1ed8 (172.18.0.2): icmp_seq=1 ttl=64 time=0.243 ms
64 bytes from 8854c7ec1ed8 (172.18.0.2): icmp_seq=2 ttl=64 time=0.164 ms

--- dev-container1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1057ms
rtt min/avg/max/mdev = 0.164/0.203/0.243/0.042 ms
```

```
# ping -c 2 dev-container2
```

Output:

```
[root@8854c7ec1ed8 /]# ping -c 2 dev-container2
PING dev-container2 (172.18.0.3) 56(84) bytes of data.
64 bytes from dev-container2.dev-network (172.18.0.3): icmp_seq=1 ttl=64 time=0.223 ms
64 bytes from dev-container2.dev-network (172.18.0.3): icmp_seq=2 ttl=64 time=0.243 ms

--- dev-container2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.223/0.233/0.243/0.010 ms
```

```
# ping -c 2 dev-container3
```

Output:

```
[root@8854c7ec1ed8 /]# ping -c 2 dev-container3
PING dev-container3 (172.18.0.4) 56(84) bytes of data.
64 bytes from dev-container3.dev-network (172.18.0.4): icmp_seq=1 ttl=64 time=0.172 ms
64 bytes from dev-container3.dev-network (172.18.0.4): icmp_seq=2 ttl=64 time=0.316 ms

--- dev-container3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 0.172/0.244/0.316/0.072 ms
```

2.9 From **dev-container1**, you should not be able to connect to centos-4 at all, since it is not on the **dev-network** network.

```
# ping -c 2 centos-4
```

Output:

```
[root@8854c7ec1ed8 /]# ping -c 2 centos-4
ping: centos-4: Name or service not known
```

Note: Detach from dev-container1 using detach sequence Ctrl+ pq

2.10 Remember that **dev-container3** is connected to both the **default bridge** network and **"dev-network"**. It should be able to reach all of the other containers.

However, you will need to address **centos-4** by its **IP address**. **Attach** to it and run the tests.

```
# docker container attach dev-container3
```

Output:

```
[root@docker-engine ~]#docker container attach dev-container3
[root@1f3228f47caa /]#
```

```
# ping -c 2 dev-container1
```

Output:

```
[root@1f3228f47caa /]# ping -c 2 dev-container1
PING dev-container1 (172.18.0.2) 56(84) bytes of data.
64 bytes from dev-container1.dev-network (172.18.0.2): icmp_seq=1 ttl=64 time=0.091 ms
64 bytes from dev-container1.dev-network (172.18.0.2): icmp_seq=2 ttl=64 time=0.209 ms

--- dev-container1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.091/0.150/0.209/0.059 ms
```

```
# ping -c 2 dev-container2
```

Output:

```
[root@1f3228f47caa /]# ping -c 2 dev-container2
PING dev-container2 (172.18.0.3) 56(84) bytes of data.
64 bytes from dev-container2.dev-network (172.18.0.3): icmp_seq=1 ttl=64 time=0.256 ms
64 bytes from dev-container2.dev-network (172.18.0.3): icmp_seq=2 ttl=64 time=0.338 ms

--- dev-container2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1052ms
rtt min/avg/max/mdev = 0.256/0.297/0.338/0.041 ms
```

```
# ping -c 2 172.17.0.2
```

Output:

```
[root@1f3228f47caa /]# ping -c 2 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.380 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.253 ms

--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1059ms
rtt min/avg/max/mdev = 0.253/0.316/0.380/0.065 ms
```

```
# ping -c 2 centos-4
```

Output:

```
[root@1f3228f47caa /]# ping -c 2 centos-4
ping: centos-4: Name or service not known
```

Note: **Detach from dev-container3 using detach sequence ctrl+pq**

Note: User defined bridge can resolve hostname (Service Discovery) where default bridge can't.

2.11 Let's **create** new network with custom subnet.

```
# docker network create --subnet 172.25.0.0/16 new-net01
```

Output:

```
[root@docker-engine ~]#docker network create --subnet 172.25.0.0/16 new-net01
c772bd4586da2b4e9a82b83102eb22b93144042791c0ca94a4cd9ab8848d25ca
```

2.12 Let's **inspect** the network.

```
# docker network inspect new-net01
```

Output:

```
[root@docker-engine ~]#docker network inspect new-net01
[
    {
        "Name": "new-net01",
        "Id": "c772bd4586da2b4e9a82b83102eb22b93144042791c0ca94a4cd9ab8848d25ca",
        "Created": "2023-02-06T19:02:46.749889846+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.25.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

2.13 Let's **verify** the new network **new-net01** whether it allocating the ip under the described subnet

```
# docker container run --name web01 -dit --network=new-net01
nginx
```

Output:

```
[root@docker-engine ~]#docker container run --name web01 -dit --network=new-net01 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
01b5b2efb836: Pull complete
db354f722736: Pull complete
abb02e674be5: Pull complete
214be53c3027: Pull complete
a69afcef752d: Pull complete
625184acb94e: Pull complete
Digest: sha256:c54fb26749e49dc2df77c6155e8b5f0f78b781b7f0eadd96ecfabdcdfa5b1ec4
Status: Downloaded newer image for nginx:latest
b4a9b84ae403971386c5134c4636191ebe49309da3f3b0614a55a3485b8ee57c
```

2.14 Let's **inspect** and verify the ip address.

```
# docker container inspect web01 | grep -i ipaddress
```

Output:

```
[root@docker-engine ~]#docker container inspect web01 | grep -i ipaddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.25.0.2",
```

2.15 Let's supply the **static ip** while create the container.

```
# docker container run --name web02 -dit --network=new-net01 --ip 172.25.0.10 nginx
```

Output:

```
[root@docker-engine ~]#docker container run --name web02 -dit --network=new-net01 --ip 172.25.0.10 nginx
9a36d83186ac63d02f4c4f4edf56ed9b5b1e3aaf9a1115deedbb64b673dc1a52
```

2.16 Let's **inspect** and verify the ip address.

```
# docker container inspect web02 | grep -i ipaddress
```

Output:

```
[root@docker-engine ~]#docker container inspect web02 |grep -i ipaddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.25.0.10",
```

2.17 Let's create a new network bridge with custom subnet and gateway.

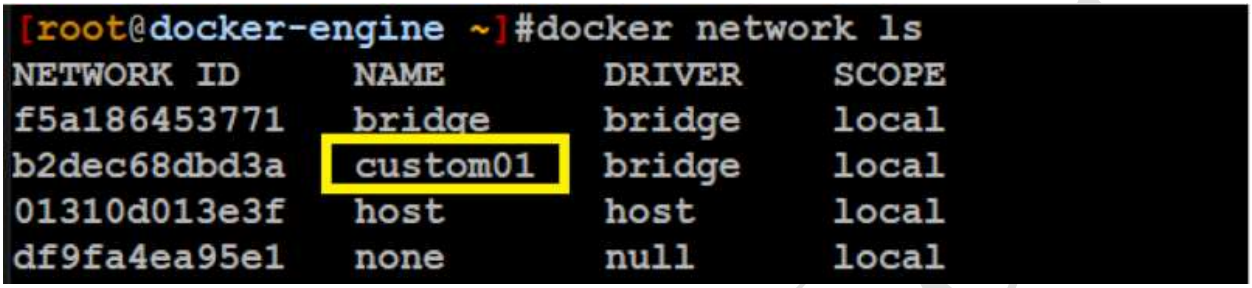
```
# docker network create -o \
"com.docker.network.bridge.name"="docker1" --subnet
172.26.0.0/16 --gateway 172.26.0.1 custom01
```

Output:

2.18 Lets verify the network bridge.

```
# docker network ls
```

Output:



2.19 Let's **disconnect** the **web02** container from the network **new-net01** .

```
# docker network disconnect new-net01 web02
```

2.20 Let's **create** container which is connected to host type network.

```
# docker container run --name web03 -dit --network=host nginx
```

Output:

2.21 Let's **inspect** the container and verify Network mode.

```
# docker container inspect web03 |grep -i NetworkMode
```

Output:

2.22 Let's access web03 by using docker host ip (Below address is the Docker Engine's Private IP, execute "**ip a s eth0**" command to know your Engine IP Address).

```
# curl 192.168.100.10
```

Output:

```
[root@docker-engine ~]#curl 192.168.100.10
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

3. Cleanup.

3.1 Let's **rm** all the containers by executing below commands.

```
# docker container rm `docker container ls -a -q` -f
```

Output:

```
[root@docker-engine ~]#docker container rm `docker container ls -a -q` -f
06cf23811844
9a36d83186ac
b4a9b84ae403
e3ebde87f62a
1f3228f47caa
346d3a3c830c
8854c7ec1ed8
```

```
# docker image rm `docker image ls -a -q` -f
```

3.2 Let's **rm** all the images by executing below command.

Output:

```
[root@docker-engine ~]#docker image rm `docker image ls -a -q` -f
Untagged: nginx:latest
Untagged: nginx@sha256:c54fb26749e49dc2df77c6155e8b5f0f78b781b7f0eadd96ecfabdcdfa5b1ec4
Deleted: sha256:9eee96112defa9d7b1880f18e76e174537491bcec6e31ad9a474cdea40f5abab
Deleted: sha256:9cadfb09eeeb8021f9bcda73607dc61ccff20c8fdc44af61267c788f174e9bdb
Deleted: sha256:f35b4ecd12ae382160d9a00f3b6956fec245c644e79e71d2bd0998403ea93a00
Deleted: sha256:0023ddac86d5e471a726b8b20d0422defc75eadbf85b3f98bfd02351f263bc57
Deleted: sha256:4bb459f6844bfd227980b8037560c81a5bf565fc81fd288052b84acbcf660b42
Deleted: sha256:0e16e930455b149d8ce0b75aaaa31b168c82d658527bfe184db954073457cb60
Deleted: sha256:bd2fe8b74db65d82ea10db97368d35b92998d4ea0e7e7dc819481fe4a68f64cf
Untagged: centos:latest
Untagged: centos@sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Deleted: sha256:5d0da3dc976460b72c77d94c8a1ad043720b0416bfc16c52c45d4847e53fadb6
```

3.3 Let's **rm** all the demo bridge network by executing below command.

```
# docker network rm new-net01 dev-network
```

Output:

```
[root@docker-engine ~]#docker network rm new-net01 dev-network
new-net01
dev-network
```