Configuring IPvlan networking in Docker

https://4sysops.com/archives/configuring-ipvlan-networking-in-docker/

The macvlan network driver allows you to assign a MAC address to Docker containers, which enables your containerized application to connect directly to your physical network.

Contents

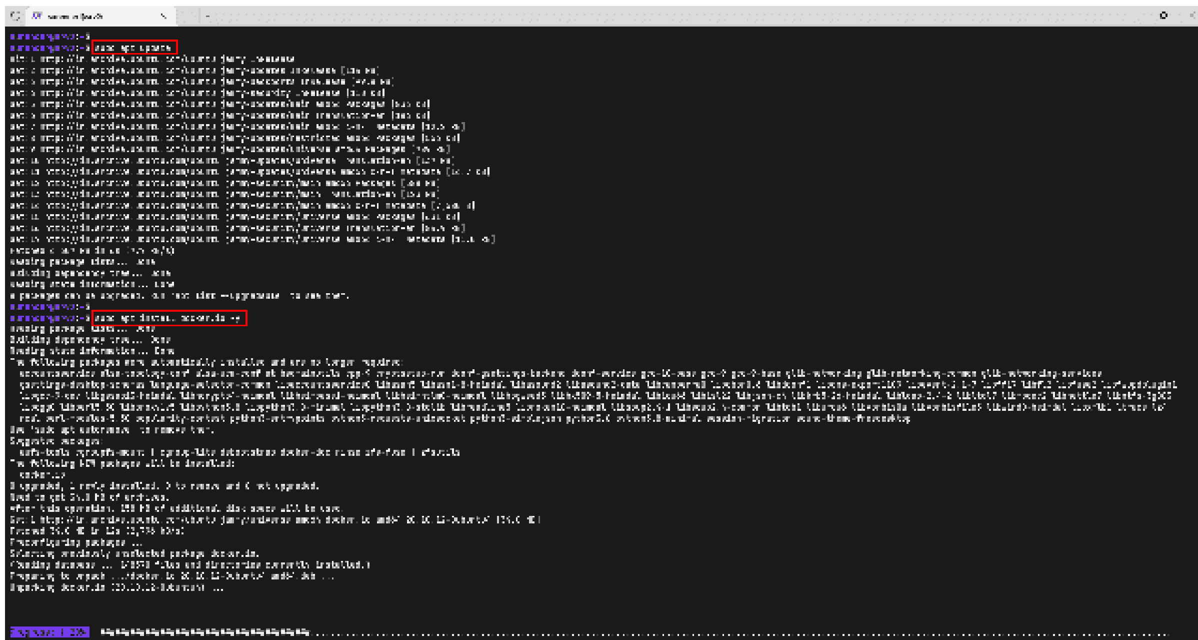
1. Prerequisites
2. Up- and downsides of the default bridge network
   - Pros of the default bridge network
   - Cons of the default bridge network
3. Why use the macvlan network?
4. How does macvlan work?
5. Configure the macvlan network.
6. Conclusion

Docker's networking subsystem works with the help of drivers, where the default driver uses a network bridge. When you create a Docker container without explicitly specifying a network, it is automatically connected to the default bridge network. However, some applications don't work properly in a bridged network using NAT. This is where the macvlan network driver comes in. In this guide, you will learn how to configure macvlan.

Prerequisites

To follow along with this guide, you need a physical server or a VM running on Ubuntu Linux. If you haven't already installed Docker, run the following commands to install it:

```
sudo apt update
sudo apt install docker.io -y
```



Installing Docker in Ubuntu Linux

Up- and downsides of the default bridge network

Before directly jumping into macvlan, let me first cover the advantages and disadvantages of the default *Bridge* and *Host* networks.

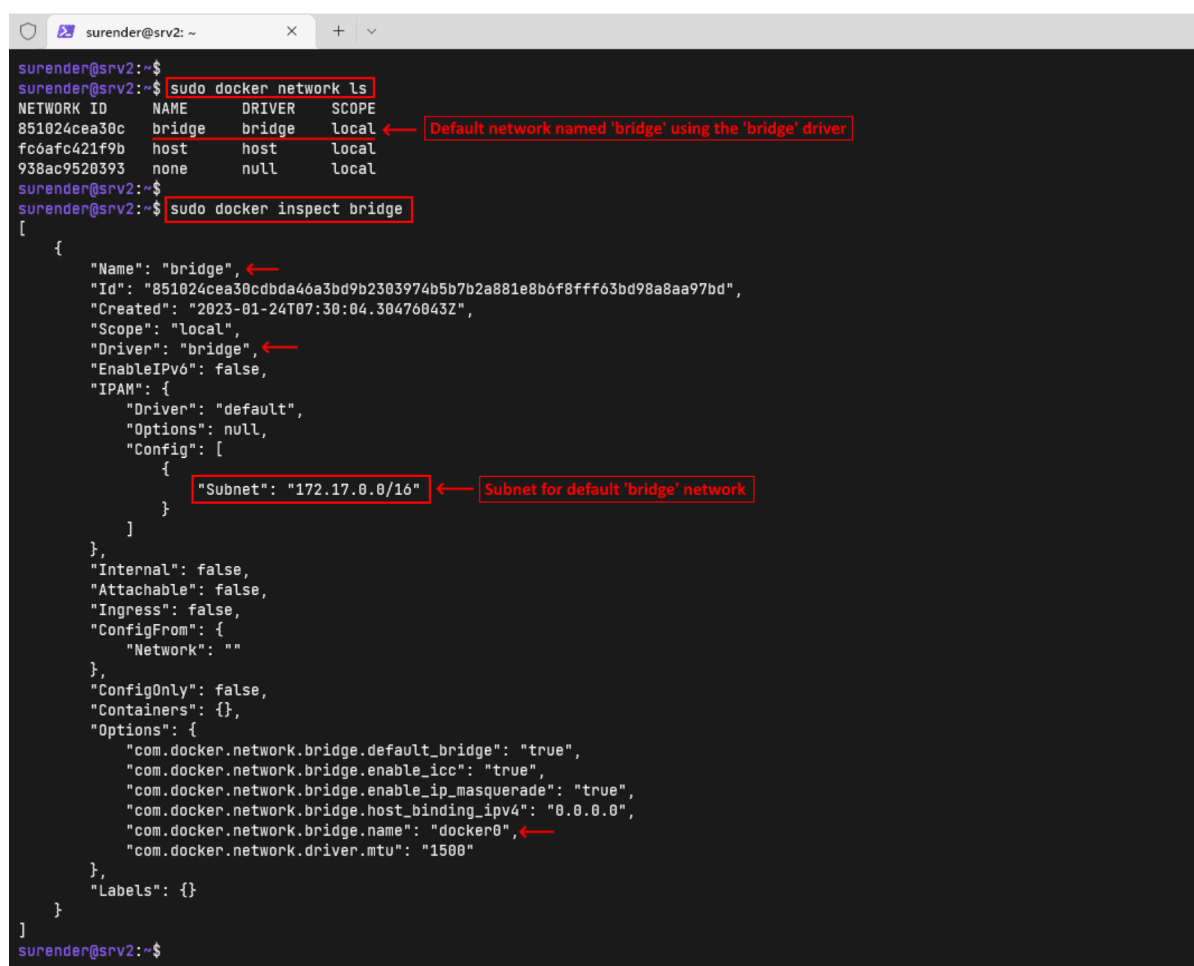
Pros of the default bridge network

- It creates an isolated private network where containers can communicate directly with each other and with the Docker daemon. For internet access, containers need to use NAT.
- It uses an isolated subnet (172.17.0.0/16), which allows containers to communicate using IP address only. It doesn't support container name resolution.

Cons of the default bridge network

- It doesn't support name resolution, so containers cannot communicate using names.
- Every new container is by default connected to this network when the network isn't explicitly specified. So you do not get the best security and isolation.
- It isn't ideal for every application, since containers use NAT for internet access. NAT is known for its slow network performance, and your application might not support NAT (or double NAT).

The following screenshot shows how you can view default Docker networks:



Viewing the default Docker network and its properties

To get complete control of Docker networking, you can use a custom (or user-defined) network. This also allows you to use container names instead of IP addresses because you can then work with name resolution. However, this doesn't solve the NAT performance issue.

To avoid NAT problems, you can use the *Host* network. The *Host* network allows Docker containers to share the same IP address and port number on the Docker host. However, this solution comes with its own problems. The *Host* network doesn't offer any network isolation, and it is prone to port-conflict issues. In a nutshell, the application appears to be running directly in the host operating system.

```
surender@srv2:~$
surender@srv2:~$ sudo docker network ls
NETWORK ID     NAME     DRIVER   SCOPE
851024cea30c   bridge   bridge   local
fc6afc421f9b   host     host     local   <-- 'host' network in Docker
938ac9520393   none     null     local
surender@srv2:~$
```

Viewing the host network in Docker

Why use the macvlan network?

The macvlan network assigns a unique MAC address to each container, making it appear to be a physical device on your network, just like a traditional virtual machine. The Docker daemon then routes the traffic to containers on the basis of their MAC address. It also allows you to assign an IP address from the same subnet in which the Docker host resides. This avoids the use of the host network, there is no NAT overhead, and you won't run into network performance issues.

Let's consider that you have a legacy application like a network traffic monitoring system that you want to containerize. Your application currently runs on a VM, since it requires being directly connected as a separate device on a physical network. In such a situation, macvlan is an ideal choice. Below are some benefits of macvlan:

- Allocates a unique MAC address to each container, allowing it to have a full TCP/IP stack.
- Allows you to allocate IP addresses to containers from the same subnet, which is managed by your enterprise IT.
- Allows a Docker container to become part of a physical network as a separate network device. It allows containers to utilize advanced network services, such as VLANs and IPAM.
- Removes the additional overhead caused by NAT.

How does macvlan work?

To understand how the macvlan network works, take a look at the diagram below.

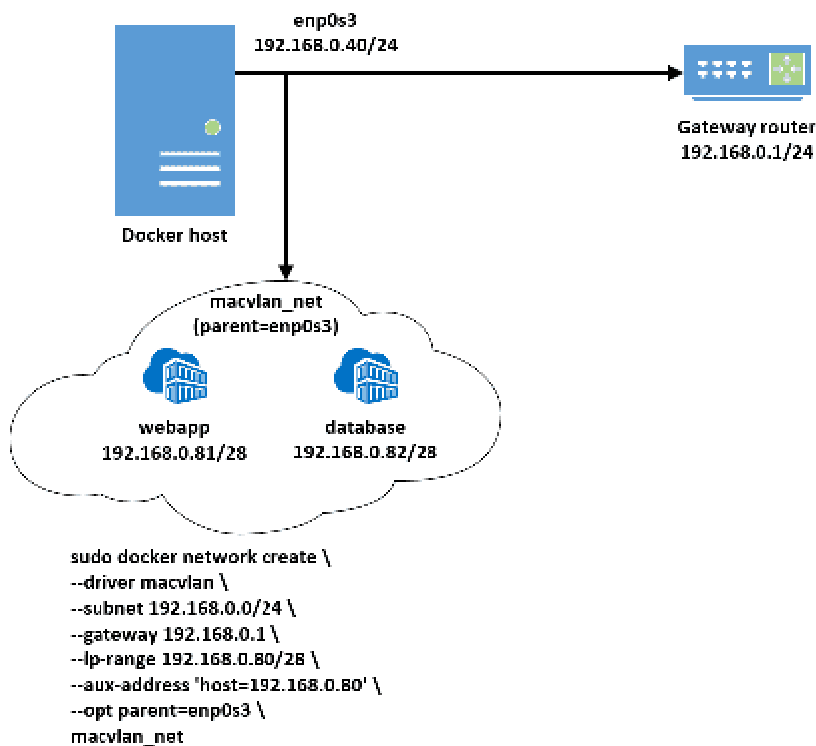


Diagram representing how macvlan works

You can see that the physical network interface (*enp0s3*) in my Docker host has IP address 192.168.0.40/24. It is connected to the gateway router (192.168.0.1), which is in the same subnet.

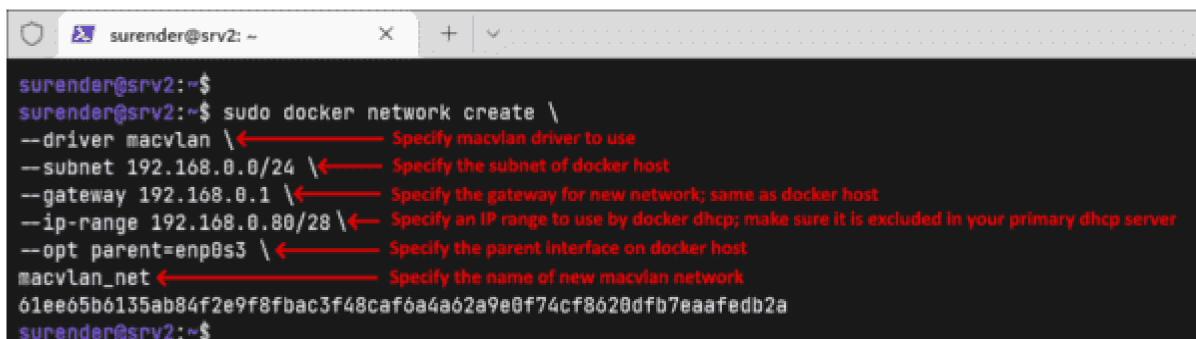
The macvlan network (*macvlan_net*) on the Docker host is configured with the parent interface set as a physical interface (*enp0s3*). At the bottom of the diagram, you can see the Docker command used to create the macvlan network. I will discuss this in more detail in the next section.

Configure the macvlan network.

Now, let's create our macvlan network based on the diagram.

1. To create a macvlan network, use the *docker network create* command with additional options:

```
sudo docker network create \
--driver macvlan \
--subnet 192.168.0.0/24 \
--gateway 192.168.0.1 \
--ip-range 192.168.0.80/28 \
--aux-address 'host=192.168.0.80' \
--opt parent=enp0s3 \
macvlan_net
```



Creating a macvlan network in Docker

As you can see in the screenshot, I've split the *docker network create* command into multiple lines using a backslash (\) for better readability and understanding.

-
  - The --driver option specifies the macvlan driver instead of the default bridge driver.
  - The --*subnet* option specifies the subnet for the new macvlan network, which must be the same as that of the Docker host.
  - The --gateway option specifies the gateway for the macvlan network.
  - The --*ip-range* is optional, and you can skip it if you're planning to manually allocate an IP address to each Docker container. Just make sure that the IP range specified here is excluded from your primary DHCP server to avoid potential IP conflicts. I used the *192.168.0.80/28* subnet, which gives me 16 IP addresses in the macvlan.
  - The --*aux-address 'host=192.168.0.80'* option allows you to exclude a specific IP address from the above mentioned IP range. This will cause the Docker DHCP to start allocating with 192.168.0.81.
  - The --*opt* parameter specifies additional options, such as the parent interface. You can use it multiple times to specify multiple options.
  - The last option sets a name for the new macvlan network.

2. You can now view the new Docker network using the commands shown below:

```
surender@srv2:~$
surender@srv2:~$ sudo docker network ls
NETWORK ID     NAME         DRIVER    SCOPE
851024cea30c   bridge       bridge    local
fc6afc421f9b   host         host      local
61ee65b6135a   macvlan_net  macvlan   local
938ac9520393   none         null      local
surender@srv2:~$
surender@srv2:~$ sudo docker inspect macvlan_net
[
    {
        "Name": "macvlan_net",
        "Id": "c93192ce2e531b632a49df084ce4bc8de1a23c6431c8154ff61625938c15e4b2",
        "Created": "2023-02-20T04:40:31.495429231Z",
        "Scope": "local",
        "Driver": "macvlan",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.0.0/24",
                    "IPRange": "192.168.0.80/28",
                    "Gateway": "192.168.0.1",
                    "AuxiliaryAddresses": {
                        "host": "192.168.0.80"
                    }
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "parent": "enp0s3"
        },
        "Labels": {}
    }
]
```

List all docker networks

Inspect the new macvlan network

Inspecting Docker networks

```
sudo docker network ls
sudo docker inspect macvlan_net
```

3. Let's now create two containers, as per our diagram.

```
sudo docker run --name webapp --rm --detach --network macvlan_net nginx
sudo docker run --name database --rm --detach --network macvlan_net --env
MARIADB_ROOT_PASSWORD=Pass@321 mariadb
```

```
surender@srv2:~$
surender@srv2:~$ sudo docker run --name webapp --rm --detach --network macvlan_net nginx
f8cfcd4a8a3d9d180519f64c06a492d1a33ea4b72042daa117a97b9acf9578eb
surender@srv2:~$
surender@srv2:~$ sudo docker run --name database --rm --detach --network macvlan_net --env MARIADB_ROOT_PASSWORD=Pass@321 mariadb
fe0b2a8cee4695e7779398 4ecdea859baf89833a0203ce6739799636b44f72dd
surender@srv2:~$
surender@srv2:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
fe0b2a8cee46   mariadb   "docker-entrypoint.s…"   10 seconds ago   Up 8 seconds              database
f8cfcd4a8a3d   nginx     "/docker-entrypoint.…"   29 seconds ago   Up 26 seconds             webapp
surender@srv2:~$
```

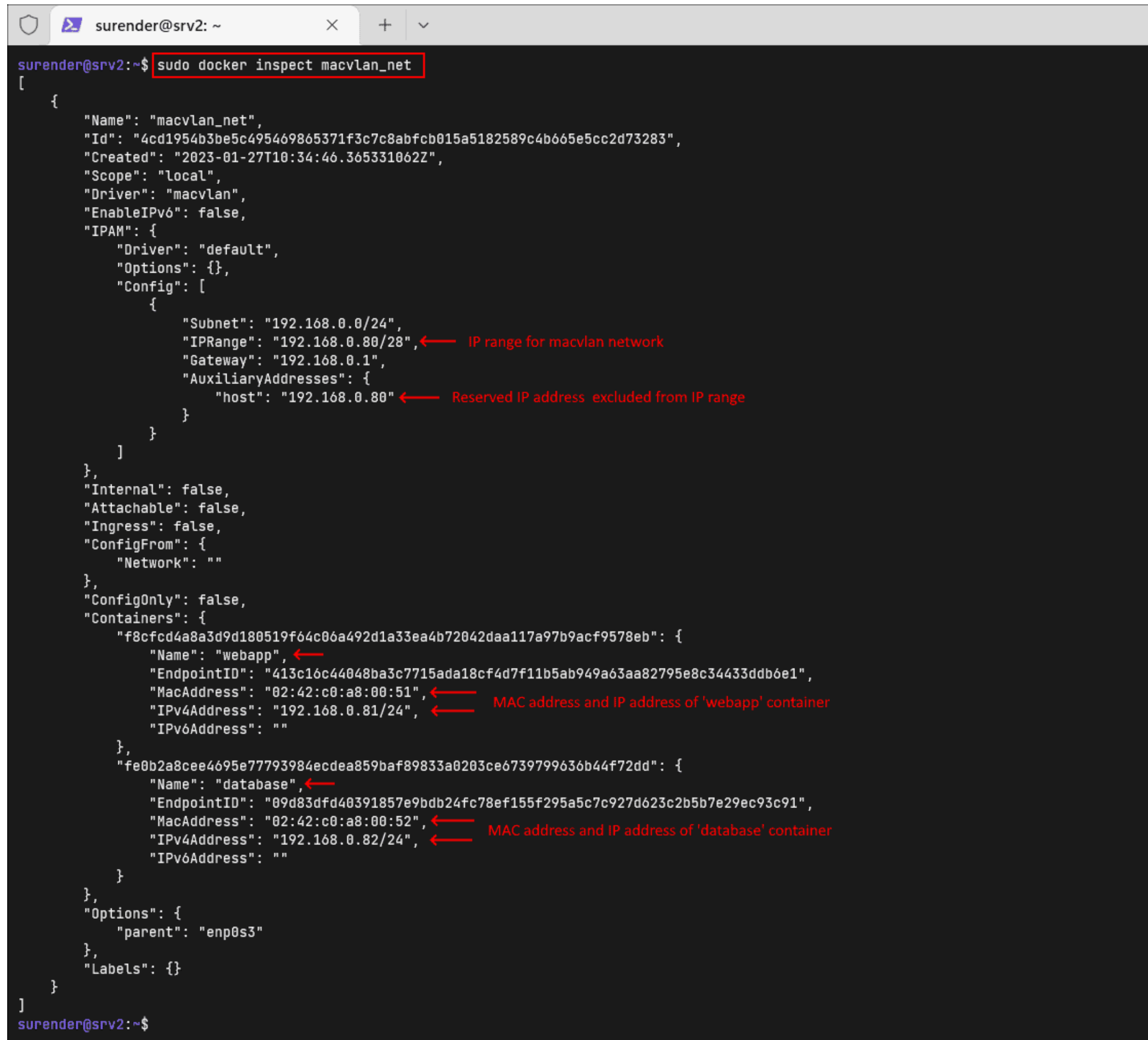
Start a container named 'webapp' connected to 'macvlan' network

Start a container named 'database' connected to 'macvlan' network

Creating Docker containers and connecting to the macvlan network

The first command creates a *webapp* container using an nginx image, and the second command creates a database container using the mariadb image in detach (background) mode. The *--network* option is key here, since it specifies the *macvlan* network to connect the containers instead of the default network.

4. If you run the *docker inspect macvlan*net *command again, you will be able to view the containers and their network configuration, as shown in the screenshot.*



Inspecting the macvlan network to view container configurations

5. You can see that both containers received a unique MAC address. The IP address is allocated from the IP range we used while defining our macvlan network. If you do not want to use an auto-assigned IP address from the Docker DHCP range, you can specify the static IP address of your choice with the *--ip* option when creating the container, as shown in the following command:It is your responsibility to make sure the IP address you allocate this way is not in use by another device in your network. sudo docker run --name alpine --rm -itd --network macvlan_net --ip 192.168.0.100 alpine Now, inside your container, you will be able to ping other containers connected to the macvlan network using their IP addresses and container names.

```
sudo docker exec -it webapp /bin/bash
```

```
surender@srv2:~$ sudo docker exec -it webapp /bin/bash
root@f8cfcd4a8a3d:/#
root@f8cfcd4a8a3d:/# ip add show eth0
34: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:a8:00:51 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.0.81/24 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@f8cfcd4a8a3d:/#
root@f8cfcd4a8a3d:/# ip route
default via 192.168.0.1 dev eth0
192.168.0.0/24 dev eth0 proto kernel scope link src 192.168.0.81
root@f8cfcd4a8a3d:/#
root@f8cfcd4a8a3d:/# ping database    <-- Ping to database container is successful; built-in name resolution working
PING database (192.168.0.82) 56(84) bytes of data.
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=1 ttl=64 time=0.121 ms
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=3 ttl=64 time=0.083 ms
^C
--- database ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.046/0.083/0.121/0.030 ms
root@f8cfcd4a8a3d:/#
root@f8cfcd4a8a3d:/# ping 192.168.0.1    <-- Cannot ping the gateway because parent interface promiscuity is by default '0'
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
From 192.168.0.81 icmp_seq=9 Destination Host Unreachable
From 192.168.0.81 icmp_seq=10 Destination Host Unreachable
From 192.168.0.81 icmp_seq=11 Destination Host Unreachable
^C
--- 192.168.0.1 ping statistics ---
12 packets transmitted, 0 received, +3 errors, 100% packet loss, time 11278ms
```

Verifying connectivity with other containers in the same macvlan network

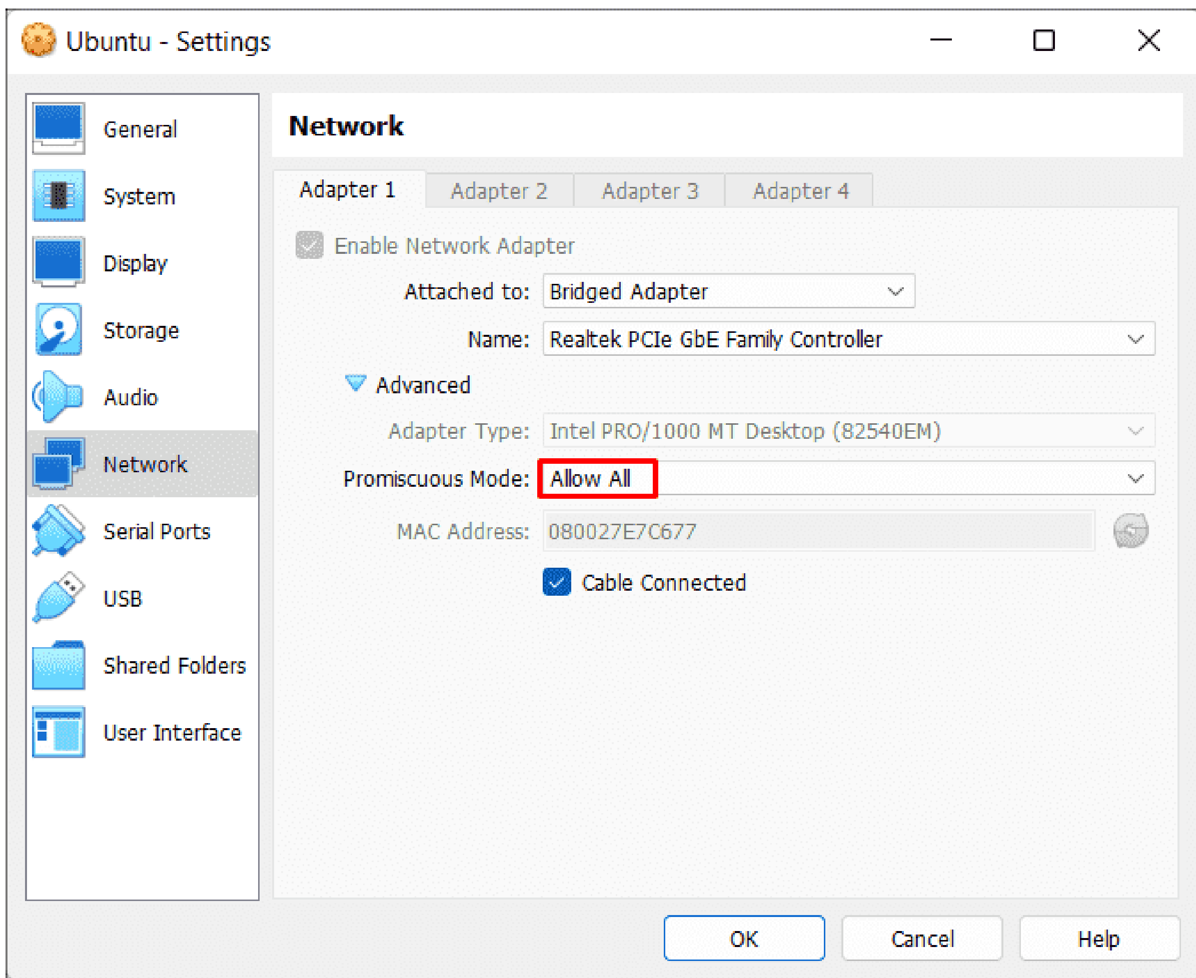
Verifying connectivity with other containers in the same macvlan network
As you can see in the screenshot above, I can ping other containers using their names, which means the name resolution is working. However, I cannot ping the default gateway because the promiscuity of the parent interface is 0, by default. Because of this, the other devices on your network will not be able to ping the Docker containers.

6. To allow communication between the Docker containers and the default gateway, you need to enable promiscuity mode on the host interface (*enp0s3*), as shown in the following commands:

```
sudo ip -detail -color link show enp0s3
sudo ip link set dev enp0s3 promisc on
```

Enabling promiscuity on the Docker host interface

If your Docker host is a virtual machine, you might also need to enable promiscuous mode on the virtual network interface in the VM settings.



Enabling promiscuous mode in VM settings

Now, your container will be able to ping the gateway, and other devices will be able to ping docker containers.

```
surender@srv2: ~
surender@srv2:~$
surender@srv2:~$ sudo docker exec -it webapp /bin/bash
root@3f8901eee120:/#
root@3f8901eee120:/# ping database
PING database (192.168.0.82) 56(84) bytes of data.
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=1 ttl=64 time=0.769 ms
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=2 ttl=64 time=0.036 ms
64 bytes from database.macvlan_net (192.168.0.82): icmp_seq=3 ttl=64 time=0.038 ms
^C
--- database ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2017ms
rtt min/avg/max/mdev = 0.036/0.281/0.769/0.345 ms
root@3f8901eee120:/# ping 192.168.0.1 ⟵ Ping to gateway is now successful
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.44 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.17 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=1.30 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=64 time=1.25 ms
^C
--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3028ms
rtt min/avg/max/mdev = 1.170/1.289/1.437/0.097 ms
root@3f8901eee120:/#
```

Verifying connectivity with other containers and the default gateway

```
C:\Windows\system32\cmd.exe

C:\Users\Surender>ping 192.168.0.81   <- Ping to 'webapp' container is successful

Pinging 192.168.0.81 with 32 bytes of data:
Reply from 192.168.0.81: bytes=32 time<1ms TTL=64
Reply from 192.168.0.81: bytes=32 time<1ms TTL=64
Reply from 192.168.0.81: bytes=32 time<1ms TTL=64
Reply from 192.168.0.81: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.81:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Surender>ping 192.168.0.82   <- Ping to 'database' container is successful

Pinging 192.168.0.82 with 32 bytes of data:
Reply from 192.168.0.82: bytes=32 time<1ms TTL=64
Reply from 192.168.0.82: bytes=32 time<1ms TTL=64
Reply from 192.168.0.82: bytes=32 time<1ms TTL=64
Reply from 192.168.0.82: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.82:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Surender>_
```

Verifying connectivity from other network device to the Docker containers

Similarly, other devices on the network can now ping both Docker containers connected to the macvlan network.

7. There is one last problem. By default, the traffic originating from the Docker containers to the Docker host (and vice versa) is filtered by the kernel to enforce strong network isolation and security. Therefore, our Docker host (192.168.0.40) at this point is not able to ping Docker containers, and containers will not be able to ping the Docker host.To solve this problem, we could create a macvlan interface on the Docker host and tell the Docker host to use this macvlan interface to pass traffic to the containers. Do you remember that we excluded an IP address, *192.168.0.80*, from the macvlan IP range? We can use this IP address for our new macvlan interface on the Docker host. To do so, run the following commands:

```
sudo ip link add macvlan_int link enp0s3 type macvlan mode bridge
sudo ip address add 192.168.0.80/32 dev macvlan_int
sudo ip link set macvlan_int up
sudo ip route add 192.168.0.80/28 dev macvlan_int
```

```
surender@srv2:~$
surender@srv2:~$ sudo ip link add macvlan_int link enp0s3 type macvlan mode bridge   <- Create macvlan interface
surender@srv2:~$
surender@srv2:~$ sudo ip address add 192.168.0.80/32 dev macvlan_int   <- Assign a static IP address to macvlan interface
surender@srv2:~$
surender@srv2:~$ sudo ip link set macvlan_int up   <- Bring the macvlan interface up
surender@srv2:~$
surender@srv2:~$ sudo ip route add 192.168.0.80/28 dev macvlan_int   <- Add a static route for docker macvlan network through macvlan interface on host
surender@srv2:~$
surender@srv2:~$
```

Creating a macvlan interface on the Docker host to allow traffic from host to container and vice versa

8. Take a look at the newly created macvlan interface and route on the Docker host with the following commands:

```
sudo ip -br -col add show
sudo ip route
```

```
surender@srv2:~$
surender@srv2:~$ sudo ip -br -col address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             192.168.0.48/24 fe80::a00:27ff:fea7:c677/64
docker0          DOWN           172.17.0.1/16
macvlan_int@enp0s3 UP             192.168.0.88/32 fe80::7470:e9ff:fe84:a086/64   <- New macvlan interface on docker host
surender@srv2:~$
surender@srv2:~$ sudo ip route
default via 192.168.0.1 dev enp0s3 proto static
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.0.0/24 dev enp0s3 proto kernel scope link src 192.168.0.48
192.168.0.80/28 dev macvlan_int scope link   <- New route for docker macvlan subnet through macvlan interface
surender@srv2:~$
```

Viewing the macvlan interface and route on the Docker host

Your Docker host can now communicate with containers, and containers can communicate with the Docker host without any problem. See the following screenshots for reference: