Macvlan network driver

- Platform support and requirements
- Considerations
- Options
- Create a Macvlan network
  - Bridge mode
  - 802.1Q trunk bridge mode
  - Use an IPvlan instead of Macvlan
- Use IPv6
  - Use router-assigned IPv6 addresses
- Usage examples
  - Bridge mode example
  - 802.1Q trunked bridge mode example

Some applications, especially legacy applications or applications which monitor network traffic, expect to be directly connected to the physical network. In this type of situation, you can use the `macvlan` network driver to assign a MAC address to each container's virtual network interface, making it appear to be a physical network interface directly connected to the physical network. In this case, you need to designate a physical interface on your Docker host to use for the Macvlan, as well as the subnet and gateway of the network. You can even isolate your Macvlan networks using different physical network interfaces.

Platform support and requirements

- The macvlan driver only works on Linux hosts. It is not supported on Docker Desktop for Mac or Windows, or Docker Engine on Windows.
- Most cloud providers block macvlan networking. You may need physical access to your networking equipment.
- Requires at least Linux kernel version 3.9 (version 4.0 or later is recommended).
- The macvlan driver is not supported in rootless mode.

Considerations

- You may unintentionally degrade your network due to IP address exhaustion or to "VLAN spread", a situation that occurs when you have an inappropriately large number of unique MAC addresses in your network.
- Your networking equipment needs to be able to handle "promiscuous mode", where one physical interface can be assigned multiple MAC addresses.
- If your application can work using a bridge (on a single Docker host) or overlay (to communicate across multiple Docker hosts), these solutions may be better in the long term.
- Containers attached to a macvlan network cannot communicate with the host directly, this is a restriction in the Linux kernel. If you need communication between the host and the containers, you can connect the containers to a bridge network as well as the macvlan. It is also possible to create a macvlan interface on the host with the same parent interface, and assign it an IP address in the Docker network's subnet.

Options

The following table describes the driver-specific options that you can pass to `--opt` when creating a network using the `macvlan` driver.

Option	Default	Description
`macvlan_mode`	`bridge`	Sets the Macvlan mode. Can be one of: `bridge`, `vepa`, `passthru`, `private`
`parent`		Specifies the parent interface to use.

Create a Macvlan network

When you create a Macvlan network, it can either be in bridge mode or 802.1Q trunk bridge mode.

- In bridge mode, Macvlan traffic goes through a physical device on the host.
- In 802.1Q trunk bridge mode, traffic goes through an 802.1Q sub-interface which Docker creates on the fly. This allows you to control routing and filtering at a more granular level.

Bridge mode

To create a `macvlan` network which bridges with a given physical network interface, use `--driver macvlan` with the `docker network create` command. You also need to specify the `parent`, which is

the interface the traffic will physically go through on the Docker host.

```
$ docker network create -d macvlan \
  --subnet=172.16.86.0/24 \
  --gateway=172.16.86.1 \
  -o parent=eth0 pub_net
```

If you need to exclude IP addresses from being used in the `macvlan` network, such as when a given IP address is already in use, use `--aux-addresses`:

```
$ docker network create -d macvlan \
  --subnet=192.168.32.0/24 \
  --ip-range=192.168.32.128/25 \
  --gateway=192.168.32.254 \
  --aux-address="my-router=192.168.32.129" \
  -o parent=eth0 macnet32
```

802.1Q trunk bridge mode

If you specify a `parent` interface name with a dot included, such as `eth0.50`, Docker interprets that as a sub-interface of `eth0` and creates the sub-interface automatically.

```
$ docker network create -d macvlan \
    --subnet=192.168.50.0/24 \
    --gateway=192.168.50.1 \
    -o parent=eth0.50 macvlan50
```

Use an IPvlan instead of Macvlan

An `ipvlan` network created with option `-o ipvlan_mode=l2` is similar to a macvlan network. The main difference is that the `ipvlan` driver doesn't assign a MAC address to each container, the layer-2 network stack is shared by devices in the ipvlan network. So, containers use the parent interface's MAC address.

The network will see fewer MAC addresses, and the host's MAC address will be associated with the IP address of each container.

The choice of network type depends on your environment and requirements. There are some notes about the trade-offs in the Linux kernel documentation.

```
$ docker network create -d ipvlan \
    --subnet=192.168.210.0/24 \
    --gateway=192.168.210.254 \
     -o ipvlan_mode=l2 -o parent=eth0 ipvlan210
```

Use IPv6

If you have configured the Docker daemon to allow IPv6, you can use dual-stack IPv4/IPv6 `macvlan` networks.

```
$ docker network create -d macvlan \
    --subnet=192.168.216.0/24 --subnet=192.168.218.0/24 \
    --gateway=192.168.216.1 --gateway=192.168.218.1 \
    --subnet=2001:db8:abc8::/64 --gateway=2001:db8:abc8::10 \
     -o parent=eth0.218 \
     -o macvlan_mode=bridge macvlan216
```

Use router-assigned IPv6 addresses

The previous example assigns IPv6 addresses from a subnet managed by Docker's IPAM. On a `macvlan` network, you can instead let containers receive IPv6 addresses directly from a router on the parent network, using stateless address autoconfiguration (SLAAC).

When a `macvlan` network has no IPv6 subnet, Docker disables IPv6 on the container's interface, so it can't accept the router advertisements that SLAAC relies on. To re-enable IPv6 on the interface, set its `disable_ipv6` sysctl to `0` when you connect the container to the network:

```
$ docker network connect \
    --driver-opt="com.docker.network.endpoint.sysctls=net.ipv6.conf.IFNAME.disable_ipv6=0" \
    my-macvlan-net my-container
```

Use the literal string `IFNAME` in the sysctl name. Docker replaces it with the name of the container's interface on this network. `disable_ipv6` is a per-interface sysctl, so it must be set with the `endpoint.sysctls` driver option rather than `docker run --sysctl`. For more details, see `docker network connect`.

Usage examples

This section provides hands-on examples for working with macvlan networks, including bridge mode and 802.1Q trunk bridge mode.

> [!NOTE] These examples assume your ethernet interface is `eth0`. If your device has a different name, use that instead.

Bridge mode example

In bridge mode, your traffic flows through `eth0` and Docker routes traffic to your container using its MAC address. To network devices on your network, your container appears to be physically attached to the network.

1. Create a macvlan network called `my-macvlan-net`. Modify the `subnet`, `gateway`, and `parent` values to match your environment:

   ```
   $ docker network create -d macvlan \
     --subnet=172.16.86.0/24 \
     --gateway=172.16.86.1 \
     -o parent=eth0 \
     my-macvlan-net
   ```

   Verify the network was created:

   ```
   $ docker network ls
   $ docker network inspect my-macvlan-net
   ```

2. Start an `alpine` container and attach it to the `my-macvlan-net` network. The `-dit` flags start the container in the background. The `--rm` flag removes the container when it stops:

   ```
   $ docker run --rm -dit \
     --network my-macvlan-net \
     --name my-macvlan-alpine \
     alpine:latest \
     ash
   ```

3. Inspect the container and notice the `MacAddress` key within the `Networks` section:

   ```
   $ docker container inspect my-macvlan-alpine
   ```

   Look for output similar to:

```
"Networks": {
  "my-macvlan-net": {
    "Gateway": "172.16.86.1",
    "IPAddress": "172.16.86.2",
    "IPPrefixLen": 24,
    "MacAddress": "02:42:ac:10:56:02",
    ...
  }
}
```

4. Check how the container sees its own network interfaces:

   ```
   $ docker exec my-macvlan-alpine ip addr show eth0

   9: eth0@tunl0: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state
   UP
   link/ether 02:42:ac:10:56:02 brd ff:ff:ff:ff:ff:ff
   inet 172.16.86.2/24 brd 172.16.86.255 scope global eth0
      valid_lft forever preferred_lft forever
   ```

   Check the routing table:

   ```
   $ docker exec my-macvlan-alpine ip route

   default via 172.16.86.1 dev eth0
   172.16.86.0/24 dev eth0 scope link  src 172.16.86.2
   ```

5. Stop the container (Docker removes it automatically) and remove the network:

   ```
   $ docker container stop my-macvlan-alpine
   $ docker network rm my-macvlan-net
   ```

802.1Q trunked bridge mode example

In 802.1Q trunk bridge mode, your traffic flows through a sub-interface of `eth0` (called `eth0.10`) and Docker routes traffic to your container using its MAC address. To network devices on your network, your container appears to be physically attached to the network.

1. Create a macvlan network called `my-8021q-macvlan-net`. Modify the `subnet`, `gateway`, and `parent` values to match your environment:

```
$ docker network create -d macvlan \
  --subnet=172.16.86.0/24 \
  --gateway=172.16.86.1 \
  -o parent=eth0.10 \
  my-8021q-macvlan-net
```

Verify the network was created and has parent `eth0.10`. You can use `ip addr show` on the Docker host to verify that the interface `eth0.10` exists:

```
$ docker network ls
$ docker network inspect my-8021q-macvlan-net
```

2. Start an `alpine` container and attach it to the `my-8021q-macvlan-net` network:

```
$ docker run --rm -itd \
  --network my-8021q-macvlan-net \
  --name my-second-macvlan-alpine \
  alpine:latest \
  ash
```

3. Inspect the container and notice the `MacAddress` key:

```
$ docker container inspect my-second-macvlan-alpine
```

Look for the `Networks` section with the MAC address.

4. Check how the container sees its own network interfaces:

```
$ docker exec my-second-macvlan-alpine ip addr show eth0

11: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state
UP
link/ether 02:42:ac:10:56:02 brd ff:ff:ff:ff:ff:ff
inet 172.16.86.2/24 brd 172.16.86.255 scope global eth0
   valid_lft forever preferred_lft forever
```

Check the routing table:

```
$ docker exec my-second-macvlan-alpine ip route

default via 172.16.86.1 dev eth0
172.16.86.0/24 dev eth0 scope link  src 172.16.86.2
```

5. Stop the container and remove the network:

```
$ docker container stop my-second-macvlan-alpine
$ docker network rm my-8021q-macvlan-net
```

□□□ #2
□ □□□ □□□ 2026-10-31 19:25:56 UTC
□ □□□ □□□ 2026-10-31 19:25:58 UTC