

How to Manage Networking in Docker?

written by sysadmin | 31 May 2025

Docker has a network system to regulate communication between one container and another container, your Docker host(server), and the outside world.

Problem

How to manage networking in Docker?

Solution

Docker provides six network drivers that you can use, as shown in the image below:

Driver	Description
bridge	The default network driver.
host	Remove network isolation between the container and the Docker host.
none	Completely isolate a container from the host and other containers.
overlay	Overlay networks connect multiple Docker daemons together.
ipvlan	IPvlan networks provide full control over both IPv4 and IPv6 addressing.
macvlan	Assign a MAC address to a container.

The six network drivers in Docker (Image credit for docs.docker.com)

You can see from the image above that the bridge is a default network driver, so if you do not specify a driver, then this type of network is created. To see the types of network drivers on your server, use the command below:

```
docker network ls
```

```
sysadmin@docker:~$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
e1d4ed8ee8af	bridge	bridge	local
3fe9d2c18c30	host	host	local
e4e911392a2c	none	null	local

```
sysadmin@docker:~$
```

List the network drivers in Docker on your server

To see the configuration details of each network driver, use the command below:

```
docker network inspect bridge host none
```

```
sysadmin@docker:~$ docker network inspect bridge host none
```

```
[
  {
    "Name": "bridge",
    "Id": "cf222e15cb997e14f4031803146097fa437d0b48f9286ab8933bb8ef3de27927",
    "Created": "2025-04-15T15:55:17.718291848Z",
    "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": {
      "2a4eadafffcd4899dce3201f8e110489e77d5c0f6d4a9bac8af91f48a06adf35": {
        "Name": "webapp1",
        "EndpointID": "2221b3ad69895615cec2fe8214bbb4e44155a153d8429710ce8cb720ea5300a9",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
    }
  }
]
```

Display detailed information about the network on each network driver

To see the type of driver network used in each container, use the command below:

```
docker ps -q | xargs docker inspect | jq '[.[] | {Name: .Name[1:], Networks:
```

```
(.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)]]'
```

```
sysadmin@docker:~$ docker ps -q | xargs docker inspect | jq '[.] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)]]'
```

```
{
  "Name": "webapp1",
  "Networks": {
    "bridge": null
  },
},
{
  "Name": "nginx",
  "Networks": {
    "bridge": null
  },
}
```

Display the network drivers in each container

To see the IP of each container, for example, in the nginx container, use the command below:

```
docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx
```

```
sysadmin@docker:~$ docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx
```

```
172.17.0.3
```

Display of the IP of a container

To communicate between one container with another container, both containers must have the same gateway. For example, you have made two containers, so by default, the two containers will use a network bridge driver so that it has the same gateway. Use the command below to see the two container IPs:

```
docker ps -q | xargs docker inspect | jq '[.] | {Name: .Name[1:], IPAddress: .NetworkSettings.IPAddress, Gateway: .NetworkSettings.Gateway}]'
```

```
sysadmin@docker:~$ docker ps -q | xargs docker inspect | jq '[.] | {Name: .Name[1:], IPAddress: .NetworkSettings.IPAddress, Gateway: .NetworkSettings.Gateway}]'
```

```
[
  {
    "Name": "webapp1",
    "IPAddress": "172.17.0.3",
    "Gateway": "172.17.0.1"
  },
  {
    "Name": "nginx",
    "IPAddress": "172.17.0.2",
    "Gateway": "172.17.0.1"
  }
]
```

Display all the IPs in each container

Then try to enter one of the containers and ping to another container using the command:

```
docker exec webapp1 ping -c2 172.17.0.2
```

```
sysadmin@docker:~$ docker exec webapp1 ping -c2 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=0.141 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 1062ms
rtt min/avg/max/mdev = 0.055/0.098/0.141/0.043 ms
sysadmin@docker:~$
```

Ping between containers

And the containers should be able to communicate with each other like the image above.

WARNING

If you can't ping on your container, then you have to install ping in your container by accessing one of your containers and installing the ping package. If your container uses Ubuntu, then you can install it using the command:

```
apt update;apt install iputils-ping
```

A. Create a new network

You can create a new network on the server for your own needs using the format below:

```
docker network create network_name --driver driver_name
```

For example, you want to create an app-network network using a bridge driver, then use the command below:

```
docker network create app-network --driver bridge
```

```
sysadmin@docker:~$ docker network create app-network --driver bridge
8ec63b0ea106fe3afce45e0b13164dd96050089a7cd7be03cf9a931b289cb641
sysadmin@docker:~$
sysadmin@docker:~$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
8ec63b0ea106	app-network	bridge	local
cf222e15cb99	bridge	bridge	local
3fe9d2c18c30	host	host	local
e4e911392a2c	none	null	local

```
sysadmin@docker:~$
```

Create a new network

INFO

You may not write the option `--driver` if you want to create a new network using a bridge driver because bridge is a default network driver in Docker.

After that, try to use the command below to see the detailed information of the app-network:

```
docker network inspect app-network
```

You can see from the picture above that the IP and Gateway from the app-network have been made automatically. You can make an IP and a Gateway according to what you want. For example, you want to create a new network with the name db-network, which has a range of IP 10.10.1.0/24 and Subnet 10.10.0.0/16 and Gateway 10.10.1.254, then use the command below:

```
docker network create -d bridge db-network \
--subnet=10.10.0.0/16 \
--ip-range=10.10.1.0/24 \
--gateway=10.10.1.254
```

```

sysadmin@docker:~$ docker network inspect app-network
[
  {
    "Name": "app-network",
    "Id": "5953916e1c5328a18728eca908f8a51ae1098fc68979e1dcd2cf2cdba9a4ca56",
    "Created": "2025-04-15T16:45:54.466051473Z",
    "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": {}
  }
]
sysadmin@docker:~$

```

Create a new network with the custom values

Use the command below to display network information details easily:

```
docker network inspect db-network -f '{{json .IPAM}}' | python3 -m json.tool
```

```

sysadmin@docker:~$ docker network inspect db-network -f '{{json .IPAM}}' | python3 -m json.tool
{
  "Driver": "default",
  "Options": {},
  "Config": [
    {
      "Subnet": "10.10.0.0/16",
      "IPRange": "10.10.1.0/24",
      "Gateway": "10.10.1.254"
    }
  ]
}
sysadmin@docker:~$

```

Display the value of the network in a container

From the image above, you can see that the IP, Subnet, and Gateway on the network are the same as what you want.

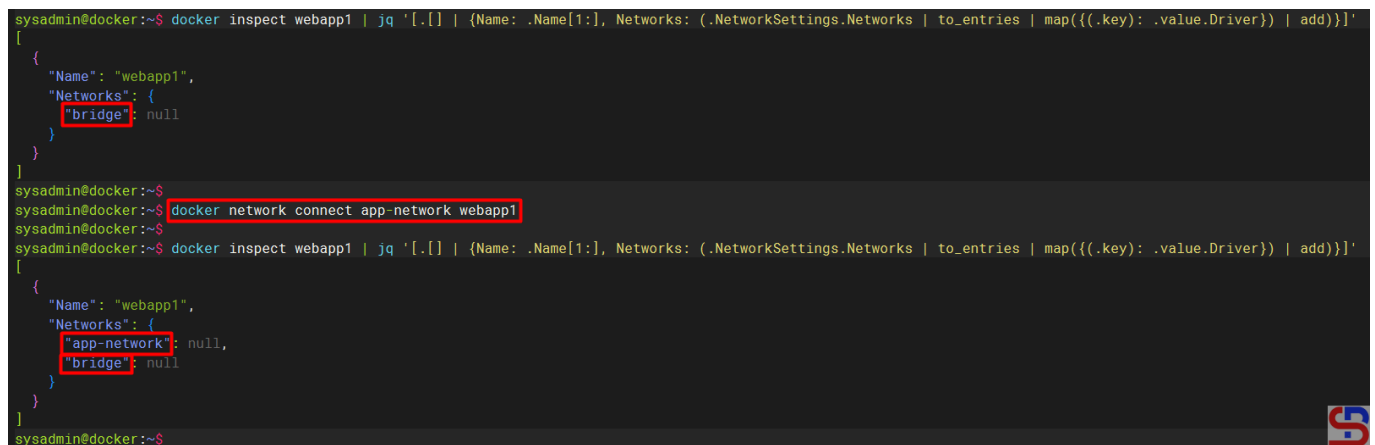
B. Connecting a network to a container

If you connect the current container to a network, use the format below:

```
docker network connect network-name container-name
```

For example, if you want to connect the webapp1 container to the app-network network, use the command below:

```
docker network connect app-network webapp1
```



```
sysadmin@docker:~$ docker inspect webapp1 | jq '[.[] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)}]'
```

```
{
  {
    "Name": "webapp1",
    "Networks": {
      "bridge": null
    }
  }
}
```

```
sysadmin@docker:~$ docker network connect app-network webapp1
```

```
sysadmin@docker:~$ docker inspect webapp1 | jq '[.[] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)}]'
```

```
{
  {
    "Name": "webapp1",
    "Networks": {
      "app-network": null,
      "bridge": null
    }
  }
}
```

Connecting a network to a container

If you want to create a new container by directly connecting to a network, use the format below:

```
docker container run --name container_name --network network_name image:tag
```

For example, if you want to create a db-mysql container on the db-network network, use the command below:

```
docker run -d --name db-mysql --network db-network -e  
MYSQL_ROOT_PASSWORD='q1w2e3r4' mysql
```

```

sysadmin@docker:~$ docker run -d --name db-mysql --network db-network -e MYSQL_ROOT_PASSWORD='q1w2e3r4' mysql
2fd039880269153882f303435bf9a197fd1aefed5b96c5df0fe2a8e291266cb3
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect db-mysql | jq '[.[] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)}}]'
[
  {
    "Name": "db-mysql",
    "Networks": {
      "db-network": null
    }
  }
]
sysadmin@docker:~$

```

Connecting a network when creating a new container

C. Disconnect a network in the container

To disconnect a network in the container, use the format below:

```
docker network disconnect network_name container_name
```

For example, if you want to break the app-network network from the webapp1 container, use the format below:

```
docker network disconnect network1 webapp1
```

```

sysadmin@docker:~$ docker inspect webapp1 | jq '[.[] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)}}]'
[
  {
    "Name": "webapp1",
    "Networks": {
      "app-network": null,
      "bridge": null
    }
  }
]
sysadmin@docker:~$
sysadmin@docker:~$ docker network disconnect app-network webapp1
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect webapp1 | jq '[.[] | {Name: .Name[1:], Networks: (.NetworkSettings.Networks | to_entries | map({(.key): .value.Driver}) | add)}}]'
[
  {
    "Name": "webapp1",
    "Networks": {
      "bridge": null
    }
  }
]
sysadmin@docker:~$

```

Disconnect a network from a container

D. Removing a network

To remove a network, use the format below:

```
docker network rm network_name
```

For example, I want to delete the app-network network, use the command below:

```
docker network rm app-network
```

```

sysadmin@docker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8ec63b0ea106       app-network         bridge              local
f36fe308a5e6       bridge              bridge              local
a292d347df19       db-network          bridge              local
3fe9d2c18c30       host                host                local
e4e911392a2c       none                null                local
sysadmin@docker:~$
sysadmin@docker:~$ docker network rm app-network
app-network
sysadmin@docker:~$
sysadmin@docker:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f36fe308a5e6       bridge              bridge              local
a292d347df19       db-network          bridge              local
3fe9d2c18c30       host                host                local
e4e911392a2c       none                null                local
sysadmin@docker:~$

```



Remove a network

WARNING

You cannot remove a network if the network is still connected to a container. First, disconnect the network connection from the container, and then you can delete the network.

Note

You must be careful when setting the network in Docker because if you set the wrong network, one or several containers will not be connected, which causes the application that runs on the Docker will be disturbed.

References

docs.docker.com
spacelift.io
youtube.dimas-maryanto.com
stackoverflow.com
youtube.com