

Cloud Engineering Principles and Practices

Docker Networking: Commands and Practical Exercises

A step-by-step tutorial and lab

This lab introduces Docker networking from the container point of view and then connects the commands to what happens on the host. You will inspect the default bridge network, understand the Linux docker0 bridge, follow a packet from a container to the Internet, create a user-defined bridge network, use Docker DNS for container-name resolution, publish a service to the host, and compare the main Docker network drivers.

By the end of the lab you should be able to choose an appropriate Docker network type, explain how bridge NAT/masquerading provides outbound Internet access, distinguish outbound connectivity from published inbound ports, and create, inspect, connect, disconnect and remove Docker networks.

Part 0 Getting ready

1. Start Docker Desktop (or Docker Engine on Linux) and wait until the engine is running.
2. Open PowerShell/Windows Terminal on Windows, or Terminal on macOS/Linux.
3. Check that Docker is working:

```
docker --version
docker version
```

List the networks that Docker created for you:

```
docker network ls
```

Platform note: Linux Engine and Docker Desktop

On a native Linux Docker Engine, the default bridge is backed by a host interface normally called docker0. Docker Desktop on Windows and macOS runs Linux containers inside a lightweight Linux VM, so docker0 is inside that VM rather than a normal Windows/macOS host interface. The cross-platform command `docker network inspect bridge` is therefore the best classroom way to inspect it.

Part 1 Docker network types

Docker networking is driver-based. The driver decides how a container is connected, what network namespace it uses, and how it reaches other containers or external networks.

Network type	What it does	Typical use
bridge	Private network on one Docker host. Containers get their own IP addresses.	Most single-host applications and labs.
host	Container shares the host network stack; no separate container IP.	Performance-sensitive tools or large port ranges.
none	No external network interface; only loopback remains.	Maximum network isolation.
overlay	Connects containers/services across multiple Docker hosts in a Swarm.	Multi-host container networking.
macvlan	Container appears directly on the physical LAN with its own MAC and IP.	Legacy/physical-LAN integration.

ipvlan	Container gets its own IP but uses the parent interface Layer-2 identity.	LAN integration with fewer MAC addresses.
--------	---	---

Important distinction

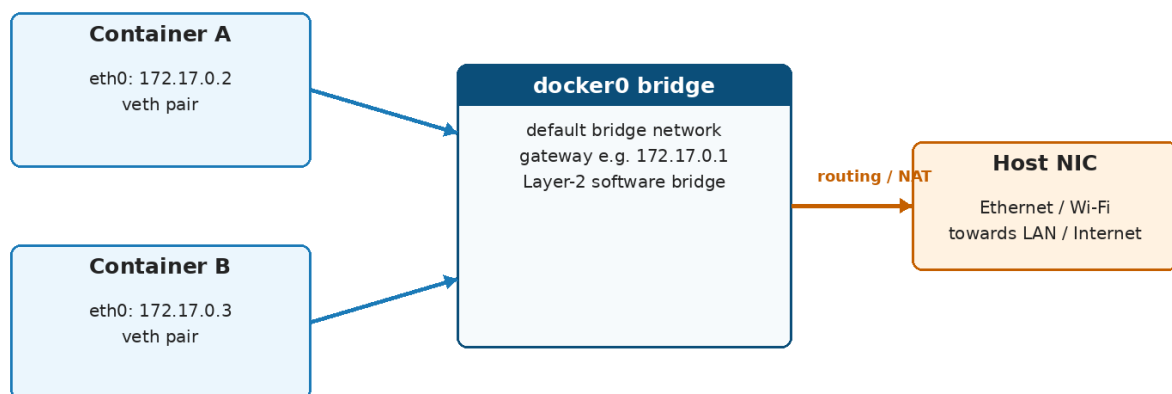
The default bridge network and a user-defined bridge both use the bridge driver, but they are not identical. User-defined bridges provide automatic DNS resolution between containers by name and are the recommended choice for application containers that need to talk to each other.

Part 2 The default bridge and docker0

2.1 What docker0 is

When Docker Engine starts on Linux, it creates a built-in network named bridge. Traditionally this network is implemented by a Linux software bridge interface named docker0. If you run a container without `--network`, Docker attaches it to this default bridge.

Bridge networking on a Linux Docker host



Conceptual Linux Engine view. Exact IP ranges and interface names can vary.

A bridge-connected container normally has an interface named eth0. A virtual Ethernet (veth) pair connects that interface to the host side. For the default bridge, the host-side veth endpoint is attached to docker0. docker0 acts as the Layer-2 switch and usually provides the container default gateway.

Do not memorise 172.17.0.1

Many installations use a default bridge subnet such as 172.17.0.0/16 with docker0 at 172.17.0.1, but Docker can choose a different range. Always inspect your own installation rather than hard-coding these addresses.

2.2 Inspect the default bridge

```
docker network inspect bridge
```

In the output, locate the network driver, subnet, gateway and the Containers section. Then start a small web server on the default bridge and inspect again:

```
docker run -d --name default-web httpd:2.4-alpine
docker network inspect bridge
```

On a native Linux host you can also examine the Linux bridge interface directly:

```
ip address show docker0
```

Exercise 1: inspect docker0 and the default bridge

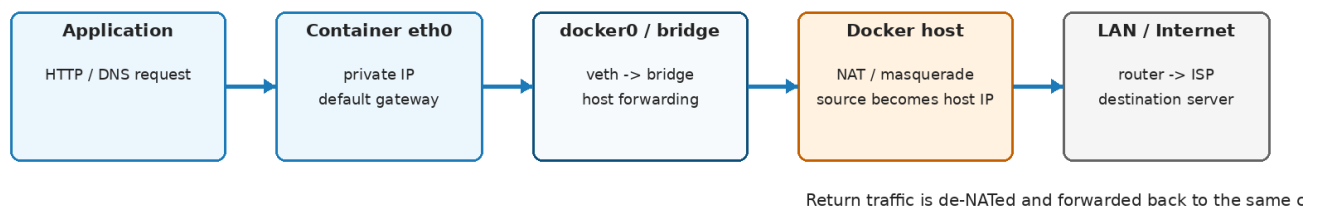
1. Run `docker network ls` and identify bridge, host and none.
2. Inspect bridge and record its subnet and gateway.
3. Start `default-web`, inspect bridge again, and find the container entry and IP address.
4. If you are on native Linux, compare the gateway shown by Docker with `ip address show docker0`.
5. Leave `default-web` running for the next section.

Part 3 How a container reaches the Internet

3.1 Outbound traffic

Containers on a normal bridge network can make outgoing connections by default. The container sends packets to its bridge gateway. The Docker host forwards them towards its external interface and applies source NAT (masquerading), so devices outside the Docker host see the host address rather than the private container address.

How a bridge-connected container reaches the Internet



Inbound connections are different: use `-p HOST_PORT:CONTAINER_PORT` to publish a service.

Example: browser -> host:8080 -> Docker port forwarding/NAT -> container:80

A simplified packet journey is: container application -> container eth0 -> veth pair -> docker bridge/docker0 -> host IP forwarding -> Docker firewall/NAT rules -> host Ethernet/Wi-Fi -> local router -> Internet. Reply traffic follows the reverse path; connection tracking allows the host to translate it back to the originating container.

Docker Desktop adds one more layer

With Docker Desktop, Linux container networking and `docker0` are inside Docker Desktop's Linux VM. Docker Desktop then forwards/NATs traffic from that VM to the Windows or macOS host and onward to the physical network. The container experience is still the same: it sees an interface, a gateway, DNS and a route.

3.2 Outbound access is not the same as inbound access

A container being able to browse the Internet does not mean the Internet or your LAN can automatically initiate a connection to that container. Bridge container addresses are private to the Docker host. To make a service reachable through the host, publish a port.

```
docker rm -f default-web
docker run -d --name default-web -p 8080:80 httpd:2.4-alpine
```

Now open `http://localhost:8080`. The host listens on port 8080 and Docker forwards that traffic to port 80 inside `default-web`. The container still uses its private bridge IP internally.

Exercise 2: prove outbound and inbound are different

1. With `default-web` running, open `http://localhost:8080` and confirm the Apache test page appears.
2. Inspect the container and identify its bridge IP address.
3. Explain why the browser uses `localhost:8080` rather than the container IP address in a normal Docker Desktop lab.
4. State which direction uses masquerading for Internet access and which direction relies on a published port.

Part 4 User-defined bridge networks and Docker DNS

4.1 Create a bridge network

For application containers, create a user-defined bridge rather than relying on the default bridge. Docker gives containers on the same user-defined bridge automatic DNS-based name resolution.

```
docker network create student-net
docker network ls
docker network inspect student-net
```

The following longer command is equivalent because `bridge` is the default driver for `docker network create`:

```
docker network create --driver bridge another-net
```

You can also ask Docker to use a specific subnet, provided it does not overlap an existing host, VPN or Docker network:

```
docker network create --subnet 172.30.10.0/24 --gateway 172.30.10.1 custom-net
```

4.2 Practical: Apache server + curl client

Start an Apache HTTP server on `student-net` and publish it to host port 8081:

```
docker run -d --name web1 --network student-net -p 8081:80 httpd:2.4-alpine
```

Now start a temporary curl container on the same network. It does not need `web1`'s IP address; Docker DNS resolves the container name:

```
docker run --rm --network student-net curlimages/curl:latest http://web1
```

The response should contain the Apache default page HTML. Try the same curl container without joining `student-net`:

```
docker run --rm curlimages/curl:latest http://web1
```

That second command should fail to resolve `web1` because the temporary client is on the default bridge, not `student-net`. This demonstrates network isolation as well as name-based service discovery.

Exercise 3: user-defined bridge and DNS

1. Create a new user-defined bridge named `lab-net`.
2. Run an Apache container named `apache-lab` on `lab-net` and publish it on host port 8082.
3. Use a temporary `curlimages/curl` container on `lab-net` to request `http://apache-lab`.
4. Open `http://localhost:8082` in the host browser.
5. Inspect `lab-net` and record the subnet, gateway and `apache-lab` IP address.
6. Run the curl test again without `--network lab-net` and explain the result.

Part 5 Connect, disconnect and isolate containers

A running container can be attached to or detached from a user-defined network without rebuilding the image.

```
docker run -d --name second-web httpd:2.4-alpine
docker network connect student-net second-web
docker network inspect student-net
```

Test name resolution from a temporary client:

```
docker run --rm --network student-net curlimages/curl:latest http://second-web
```

Disconnect the container and repeat the test:

```
docker network disconnect student-net second-web
docker run --rm --network student-net curlimages/curl:latest http://second-web
```

After the disconnect, second-web is no longer a member of student-net, so clients on student-net should not resolve or reach it by that network identity.

Exercise 4: change network membership

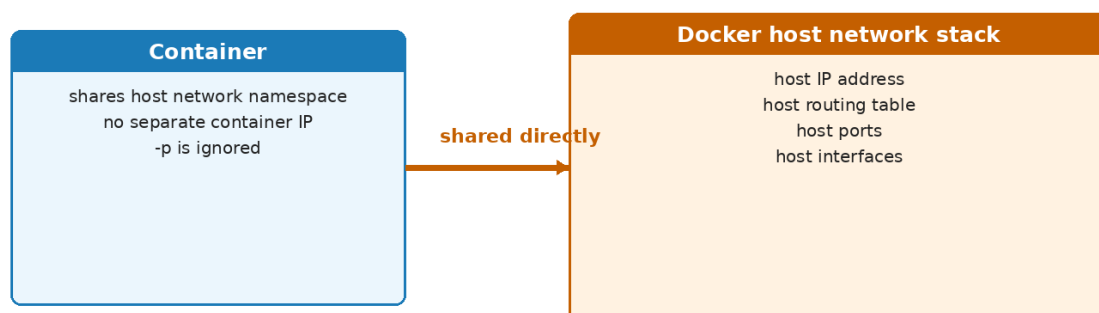
1. Start a new httpd container called movable-web without specifying a network.
2. Connect movable-web to lab-net while it is running.
3. Prove that a curl client on lab-net can reach http://movable-web.
4. Disconnect movable-web from lab-net and prove the same request no longer works.
5. Reconnect it and confirm connectivity returns.

Part 6 Other Docker network drivers

6.1 Host networking

With the host driver, the container shares the Docker host's network namespace instead of receiving a separate bridge IP. On Linux, if the application listens on port 80, it listens directly on the host network stack. Port publishing (-p) is therefore ignored in host mode.

Host network driver



Use when direct host networking or very large port ranges are required; isolation is reduced.

```
docker run --rm --network host httpd:2.4-alpine
```

Platform caution

Treat host networking as a concept unless your lab platform supports it. Native Linux supports host networking directly. Docker Desktop also has host-networking support in recent releases when the feature is enabled, but behaviour and availability can depend on the Docker Desktop version and settings.

6.2 None networking

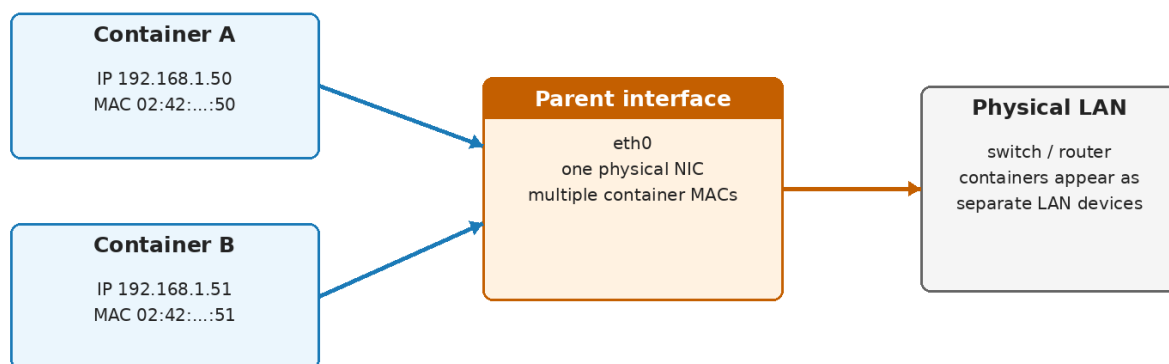
The none driver gives a container no normal network connectivity. It retains only its loopback interface. This is useful when a process should be deliberately isolated from other containers, the host network and the Internet.

```
docker run --rm --network none alpine:latest ip address
```

6.3 Macvlan networking

Macvlan is different from bridge networking: a container can appear on the physical LAN as if it were a separate machine, with its own IP address and its own MAC address. Traffic passes through a specified parent interface such as eth0.

Macvlan network driver



External network sees a unique MAC address for each container.

```
docker network create -d macvlan \
  --subnet 192.168.1.0/24 \
  --gateway 192.168.1.1 \
  -o parent=eth0 macvlan-net
```

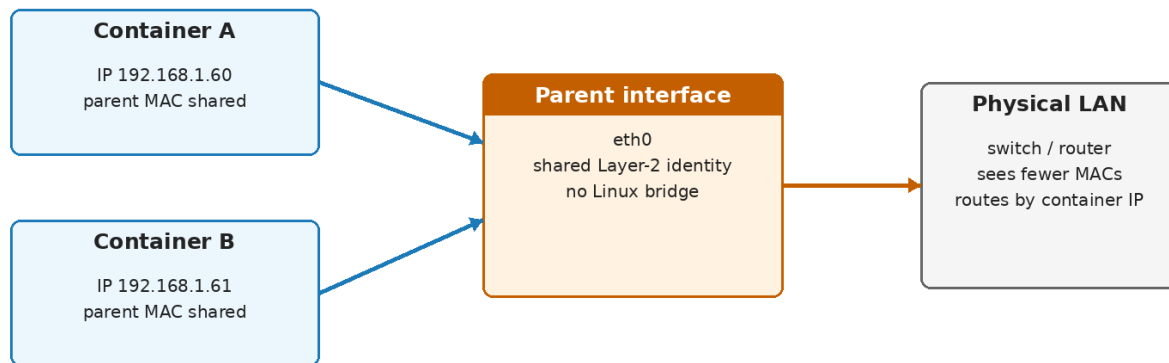
Do not copy this macvlan command blindly

The subnet, gateway and parent interface must match the real LAN. Macvlan can also require the physical network to accept multiple MAC addresses on one port, and Linux prevents direct host-to-macvlan-container communication by default. This is normally demonstrated only in a controlled Linux networking lab.

6.4 IPvlan networking

IPvlan also connects containers closely to an external network, but containers do not each present a unique MAC address. In L2 mode they share the parent interface Layer-2 identity while still using distinct IP addresses. This can be useful when the upstream switch or virtual network limits the number of MAC addresses.

IPvlan network driver (L2 concept)



Useful where the network limits MAC addresses per switch port.

```
docker network create -d ipvlan \
  --subnet 192.168.1.0/24 \
  --gateway 192.168.1.1 \
  -o ipvlan_mode=l2 \
  -o parent=eth0 ipvlan-net
```

6.5 Overlay networking

Overlay networking is for containers or services that must communicate across multiple Docker hosts. Docker uses an overlay data plane over the underlying host network. In Docker Engine, overlay networks are normally associated with Swarm mode and are outside the scope of this single-host practical.

```
docker network create -d overlay --attachable multi-host-net
```

Which network should you choose?

For this module's normal single-host Docker practicals, use a user-defined bridge. Move to host, macvlan, ipvlan or overlay only when the requirements specifically demand direct host networking, physical-LAN identities, MAC scaling, or multi-host communication.

Part 7 Final independent network exercise

Exercise 5: build a small two-container network

1. Create a user-defined bridge network named assessment-net.
2. Run an nginx:alpine container named site-a on assessment-net and publish it on host port 8090.
3. Use a temporary curlimages/curl container attached to assessment-net to request <http://site-a> by name.
4. Prove from the host browser that <http://localhost:8090> reaches the same web server.
5. Inspect assessment-net and record its driver, subnet, gateway and site-a IP address.
6. Run a second web container named site-b on assessment-net without publishing a host port.
7. Prove that the curl client can reach both <http://site-a> and <http://site-b> even though only site-a is published to the host.
8. Disconnect site-b from assessment-net and demonstrate that the client can no longer reach it by name.
9. Reconnect site-b, prove connectivity again, then stop and remove the containers and remove assessment-net.

Evidence to retain

- docker network ls showing the network you created.
- docker network inspect output showing subnet, gateway and connected containers.
- Successful curl output using a container name rather than an IP address.
- A browser screenshot showing the published Apache/nginx service through localhost.
- A short explanation or diagram of the container-to-Internet path through bridge, host forwarding and NAT/masquerading.

Docker network commands at a glance

Command	Purpose
docker network ls	List Docker networks.
docker network inspect NETWORK	Show detailed network configuration and connected containers.
docker network create NETWORK	Create a user-defined bridge network.
docker network create -d DRIVER NETWORK	Create a network using a selected driver.
docker network connect NETWORK CONTAINER	Attach an existing container to a network.
docker network disconnect NETWORK CONTAINER	Detach a container from a network.
docker network rm NETWORK	Remove an unused user-defined network.
docker network prune	Remove all unused user-defined networks; use carefully.
docker run --network NETWORK ...	Start a container already attached to a network.
docker run -p HOST:CONTAINER ...	Publish a container port through the Docker host.

Clean up

```
docker rm -f default-web web1 second-web movable-web
docker network rm student-net another-net custom-net lab-net assessment-net
```

If Docker reports that one of these objects does not exist, continue with the remaining names. A network cannot be removed while a container is still attached to it.

Be careful with prune

docker network prune removes every unused user-defined network on the Docker host, not only networks created in this lab. Do not run it on a shared Docker server unless you know what else is using that host.

Quick knowledge check

1. What is the relationship between Docker's default bridge network and docker0 on native Linux?
2. Why can a bridge-connected container usually reach the Internet even though it has a private IP address?
3. Why does -p 8080:80 matter for inbound connections but not for normal outbound Internet access?
4. What advantage does a user-defined bridge have over the default bridge for container-to-container communication?
5. Why is host networking less isolated than bridge networking?

6. How does macvlan differ from ipvlan at Layer 2?

7. When would an overlay network be more appropriate than a bridge network?

When something goes wrong

Message / symptom	What to check
Could not resolve host / name	Confirm both containers are attached to the same user-defined bridge and check the container name.
port is already allocated	Another container or program already uses that host port. Choose another host port.
network has active endpoints	One or more containers are still connected. Disconnect or remove them before removing the network.
Pool overlaps with other one on this address space	The requested subnet overlaps an existing Docker, host or VPN network. Choose a different subnet.
Container has no Internet access	Check host Internet access, the container route/DNS, and whether Docker firewall/NAT rules are enabled.
host network behaves differently	Host mode is platform-dependent; verify Linux/Docker Desktop support and settings before using it.

Reference

Docker networking overview: <https://docs.docker.com/engine/network/>

Bridge network driver: <https://docs.docker.com/engine/network/drivers/bridge/>

Network drivers: <https://docs.docker.com/engine/network/drivers/>

Packet filtering and firewalls: <https://docs.docker.com/engine/network/packet-filtering-firewalls/>

Docker Desktop networking: <https://docs.docker.com/desktop/features/networking/>