DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Networking

Learning Objectives

By the end of this lesson, you will be able to:

- Explain different types of networks and their architecture
- Describe the architecture of Container Network Model (CNM) and different network drivers
- Explain various use cases for different types of networks
- Identify and publish the ports
- Access logs and troubleshoot services

Network Architecture

Networks: Overview

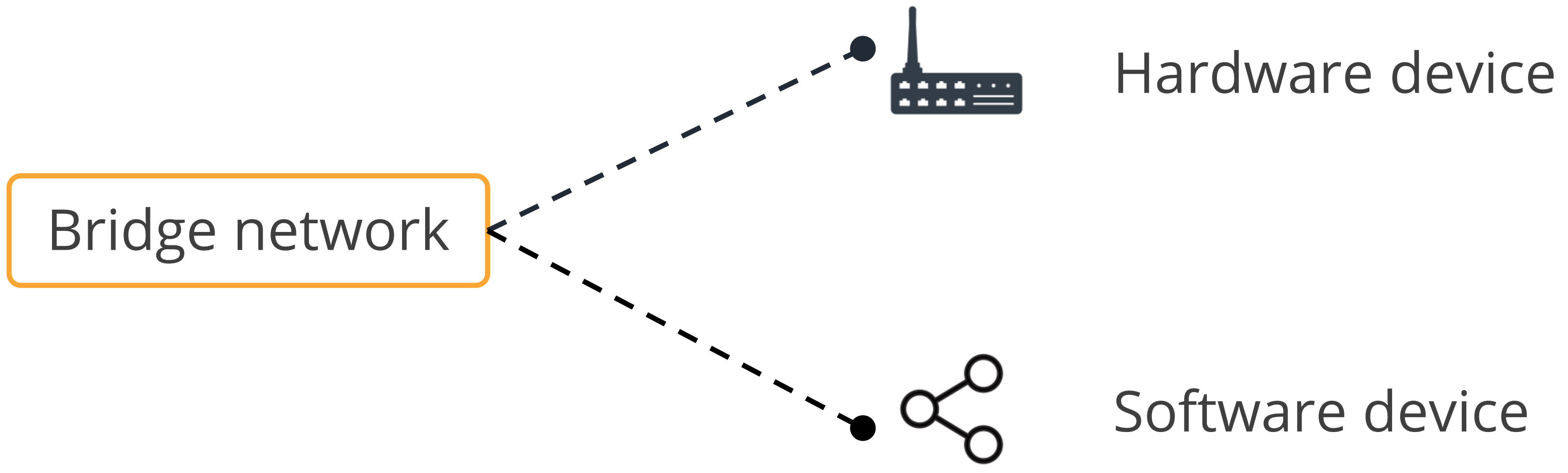
Docker networking subsystem → use → Drivers

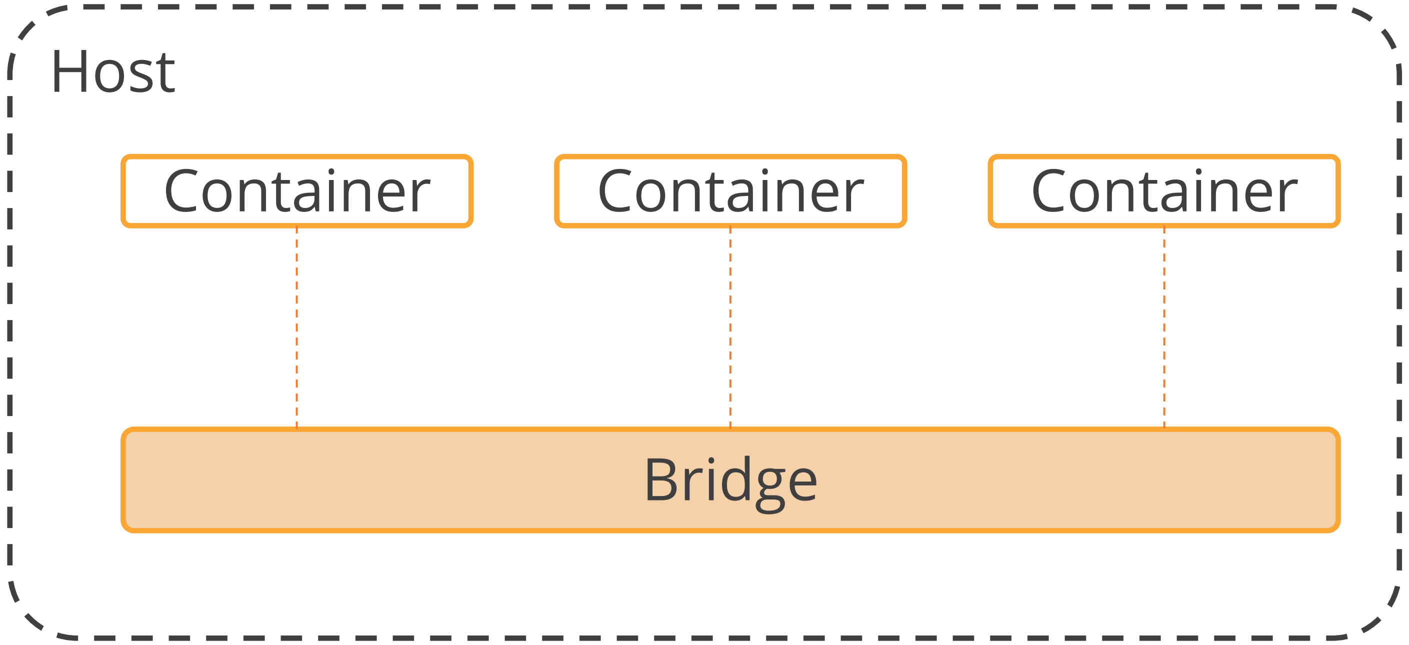
Types of drivers:

- Bridge networks
- Host networks
- Overlay networks
- Macvlan networks
- Third-party network plugins

Bridge Network

Bridge Network: Overview





Source: https://success.docker.com/article/networking#dockerbridgenetworkdriver

Bridge Network

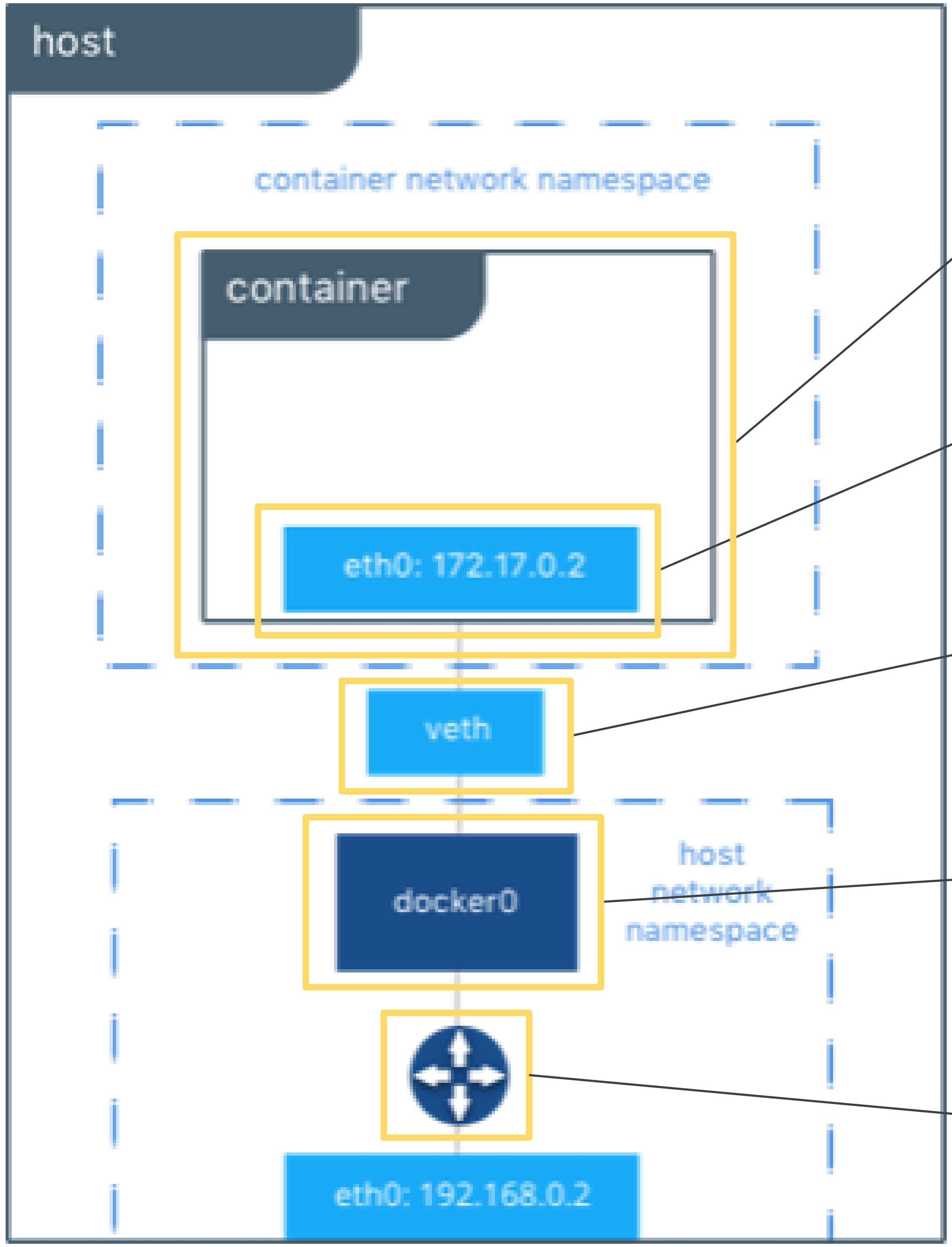
A Bridge is a default Docker network that is present on any Linux host which runs a Docker Engine.

Bridge network driver $\xrightarrow{\text{creates}}$ Bridge network

Understanding correlated terms:

- A *bridge* is a Docker network
- A *bridge* is also a Docker network driver/template, which creates a bridge network
- *docker0* is the kernel building block that is used in implementing the bridge network

Bridge Network



The Docker Engine connects the container to the bridge network by default.

eth0 is created by the bridge driver and an address is given by the Docker native IPAM driver.

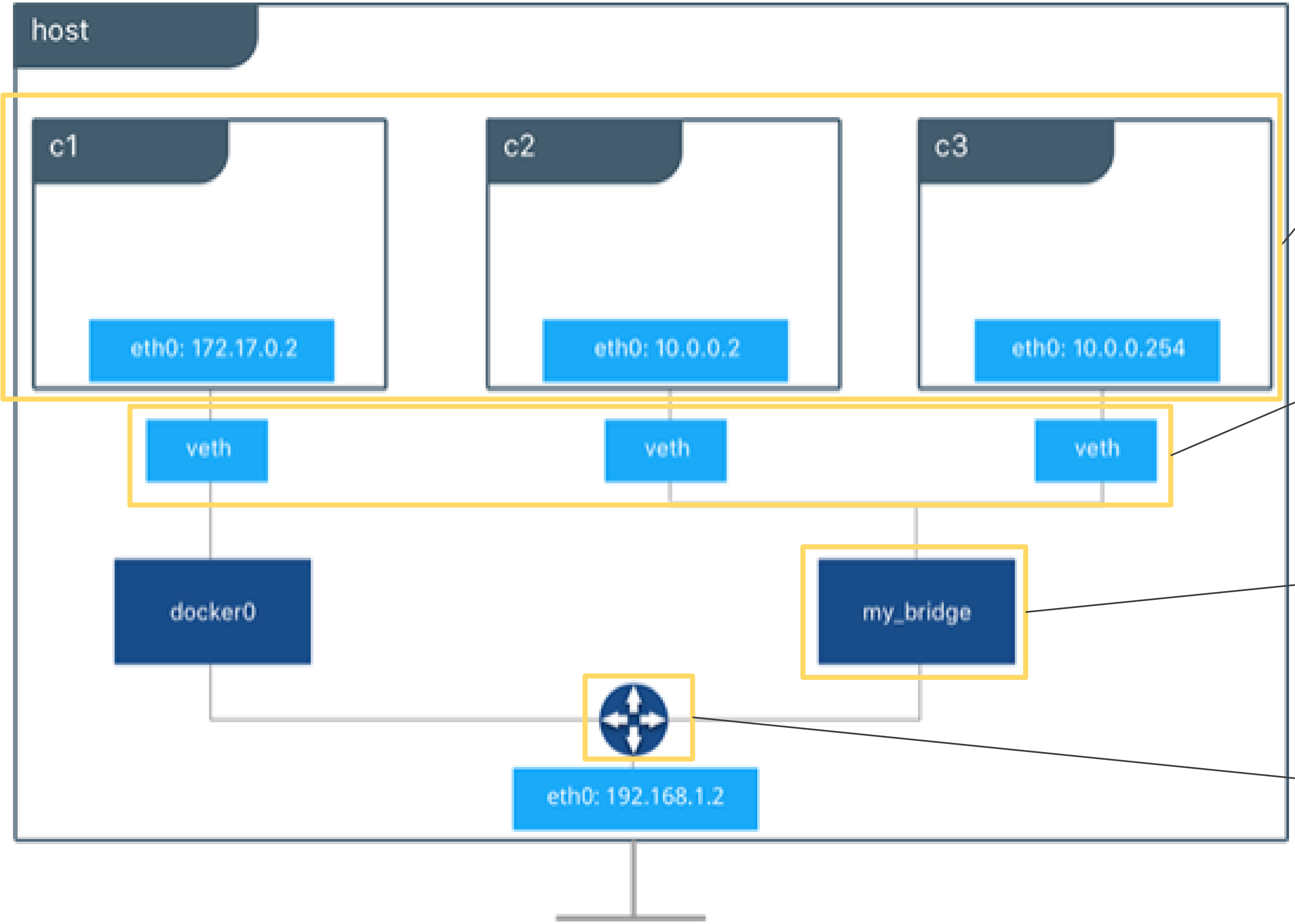
veth is a virtual ethernet interface which connects bridge to the *eth0* interface inside the container.

docker0 is a Linux bridge that exists in the host network namespace.

The routing table connects *docker0* and *eth0* on the external network

Source: https://success.docker.com/article/networking#dockerbridgenetworkdriver

User-Defined Bridge Network



Containers

Virtual ethernet interfaces

A user-defined bridge that connects to two containers via two interfaces

The routing table connects *docker0, my_bridge,* and *eth0* on the external network

Source: https://success.docker.com/article/networking#dockerbridgenetworkdriver

Bridge Network

Difference		
Features	**Default**	**User-defined**
Better isolation and interoperability between containerized applications	No	Yes
Automatic DNS resolution between containers	No	Yes
Attachment and detachment of containers on the fly	No	Yes
Configurable bridge creation	No	Yes
Linked containers share environment variables	Yes	No

Assisted Practice

Create a Bridge Network

Problem Statement: You are required to create a default network in docker and inspect it so that it can be established that the bridge driver is the default network driver.

Steps to Perform:

1. Create a default network.
2. List all the current networks to check whether a network is created or not.
3. Inspect the network created for the *driver* flag.

Assisted Practice
Create a User-Defined Bridge Network

Problem Statement: You have been asked to create a user-defined bridge network that can be used to connect multiple containers to a single network.

Steps to Perform:

1. Create a user-defined bridge network.
2. Create an *nginx* container and connect it to the user-defined bridge network.
3. Connect a running container to an existing network.
4. Disconnect the container from the network.

Host Network

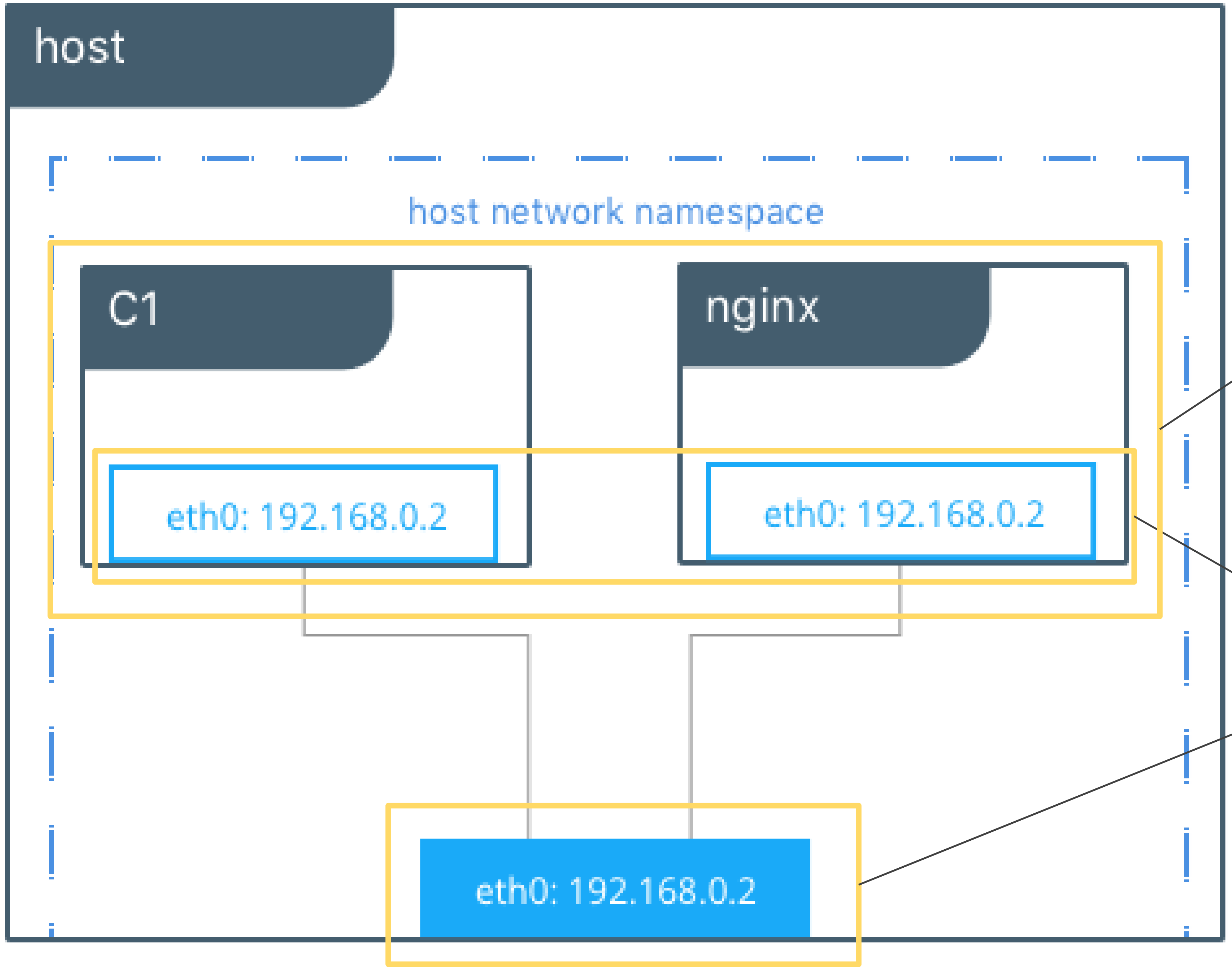
Host Network: Overview

While using a host network, the container shares the host's networking namespace, and the container is not allocated its own IP address.

Advantages:

- Optimizes the performance
- Handles a large range of ports
- Does not require network address translation (NAT)
- Does not require "userland-proxy" for each port

Host Network



These containers connect with each other using a *localhost* on C1.

The containers C1 and nginx are using the host network and share the same interface for *eth0.*

Source: *https://success.docker.com/article/networking#dockerhostnetworkdriver*

Host Network

--network host: This is passed with command *docker service create* to use a host network for a swarm service.

Features:

- An overlay network is used to manage swarm and service-related traffic.
- The Docker daemon host network and ports are used to send data for individual swarm service.

NOTE

The host networking driver only works on Linux hosts.

Assisted Practice
Create a Host Network

Problem Statement: You have been asked to create a host network so that your container gets its own IP address allocation and is not isolated from the host.

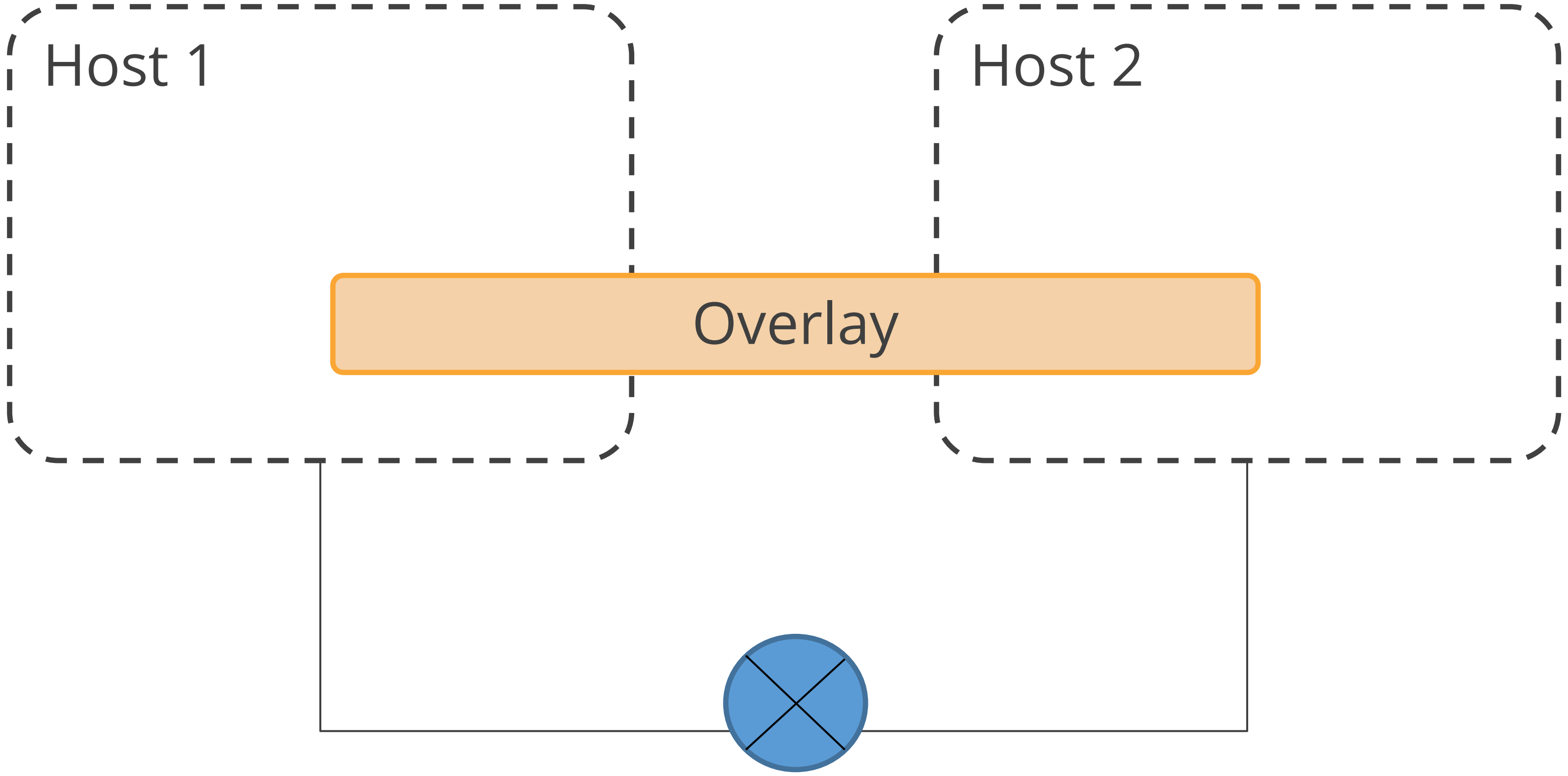
Steps to Perform:

1. Create and start the container as a detached process.
2. Access Nginx by browsing to *http://localhost:80/*.
3. Stop the container.
4. Verify which process is bound to port 80 by using the netstat command.
5. Examine all network interfaces and verify that a new one is not created.

Overlay Network

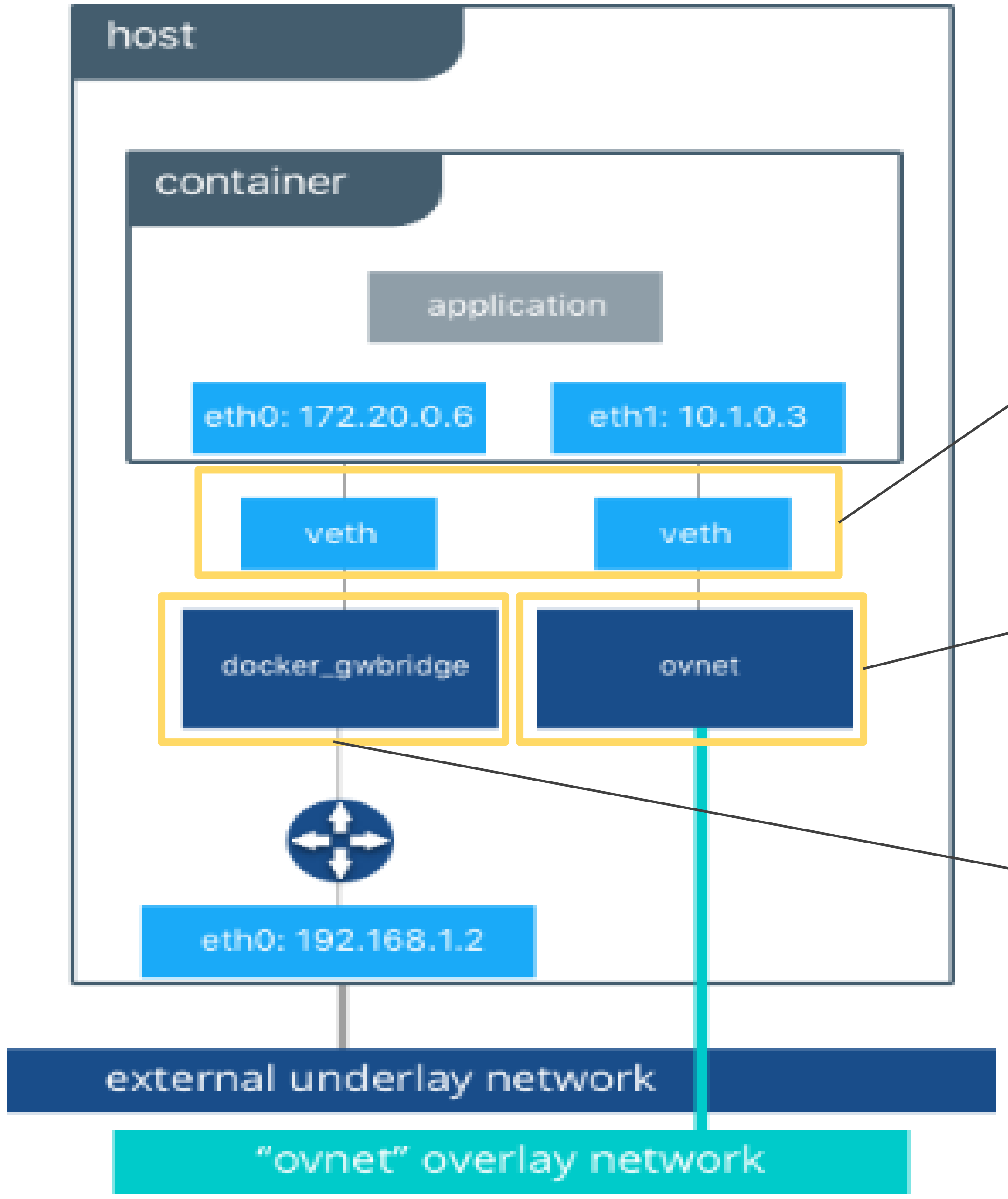
Overlay Network: Overview

Overlay network driver: It creates a distributed network among multiple Docker daemon hosts.



Overlay Network

Provisioning for an overlay network is automated by Docker Swarm control plane.



Virtual ethernet interfaces

An *ovnet* is a bridge with the overlay driver

A *docker_gwbridge* is an egress bridge

https://success.docker.com/article/networking#overlaydrivernetworkarchitecture

Overlay Network

Swarm — creates →

- *Ingress*: It handles data traffic and the controls related to swarm services.
- *docker_gwbridge*: It connects all the Docker daemons which are participating in the swarm.

Overlay Network: Prerequisites

Open ports:

- Open TCP port 2377 for cluster management communications
- Open TCP and UDP port 7946 for communication among nodes
- Open UDP port 4789 for overlay network traffic

Initialize Docker daemons:

Initialize Docker daemon as a swarm manager using *docker swarm init*, or join the Docker daemon to an existing swarm using *docker swarm join*, before creating an overlay network.

Macvlan Network

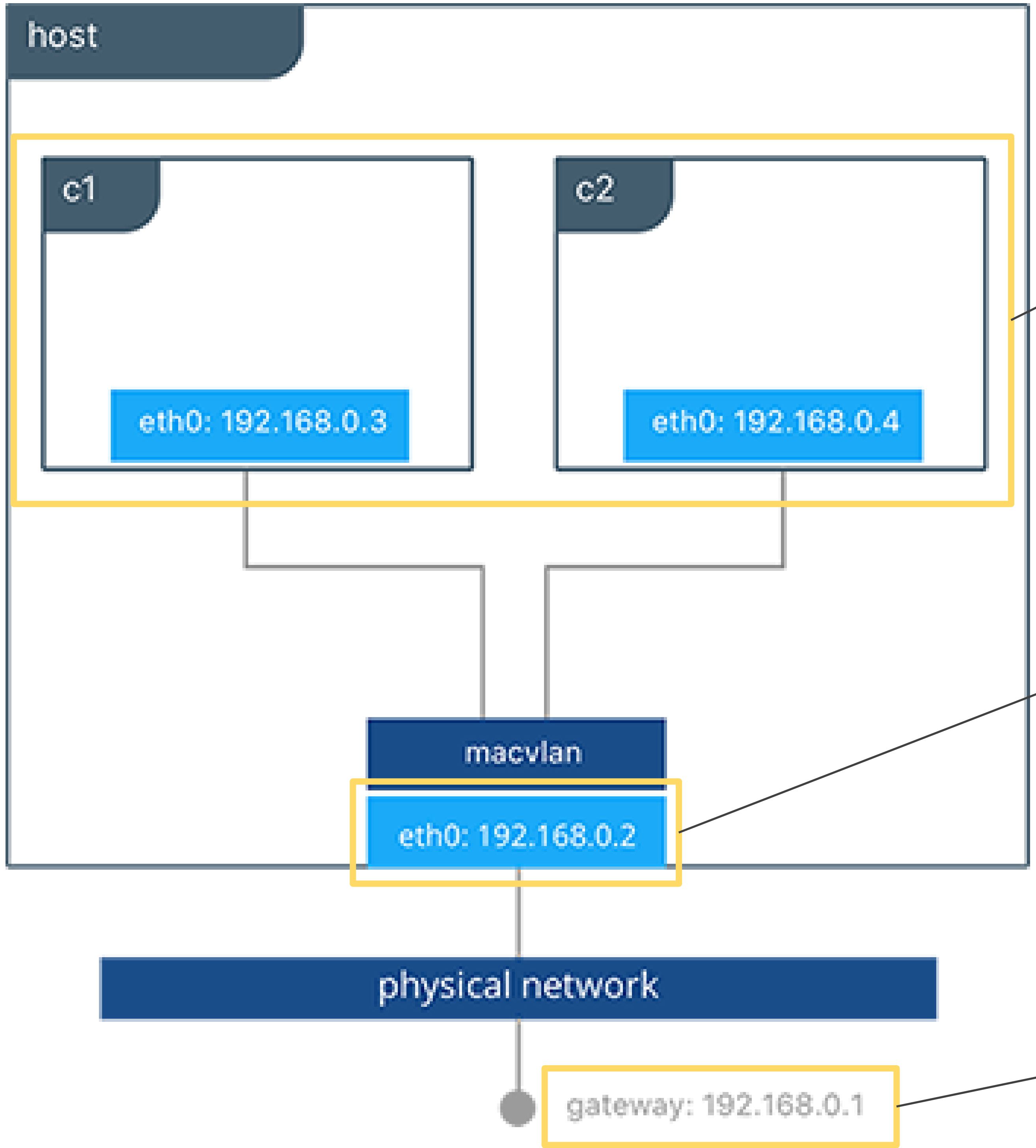
Macvlan Network: Overview

Macvlan network is used to assign MAC address to the virtual network interface of containers. This helps the legacy applications to directly connect to the physical network.

Precautionary measures:

- Cut down the large number of unique MAC address to save the network from damage
- Handle “promiscuous mode” via networking equipment to assign multiple MAC address to single physical interface

MACVLAN Network



Containers formed on *mvnet* network

An interface *eth0* on the host is bound to *mvnet (a MACVLAN network)*

An external gateway is required during MACVLAN network configuration

https://success.docker.com/article/networking#macvlan

MACVLAN Network

Positive performance implications:

- MACVLAN has simple and lightweight architecture
- MACVLAN drivers provide direct access between physical network and containers
- MACVLAN containers receive routable IP addresses that are present on the subnet of the physical network

Use cases of MACVLAN include:

- Low-latency applications
- Network design which needs containers to be on the same subnet and use IPs as the external host network

Assisted Practice
Create a Macvlan Network

Problem Statement: Your manager has asked you to create a macvlan network for legacy applications so that it can be directly connected to a physical network.

Steps to Perform:

1. Create a Macvlan network in bridge mode along with parent name.
2. Exclude IP addresses from the macvlan network.
3. Create a Macvlan network in 802.1q trunk bridge mode.

None Network

None Network: Overview

None provides the functionality of disabling networking.

Form a container with none network:

Command:

```
$ docker run --rm -dit \
  --network none \
  --name no-net-alpine \
  alpine:latest \
  ash
```

Using this will result in a container with no *eth0*

Prune Network

Prune Networks

Docker networks don’t take up much disk space, but they do create iptables rules, bridge network devices, and routing table entries.

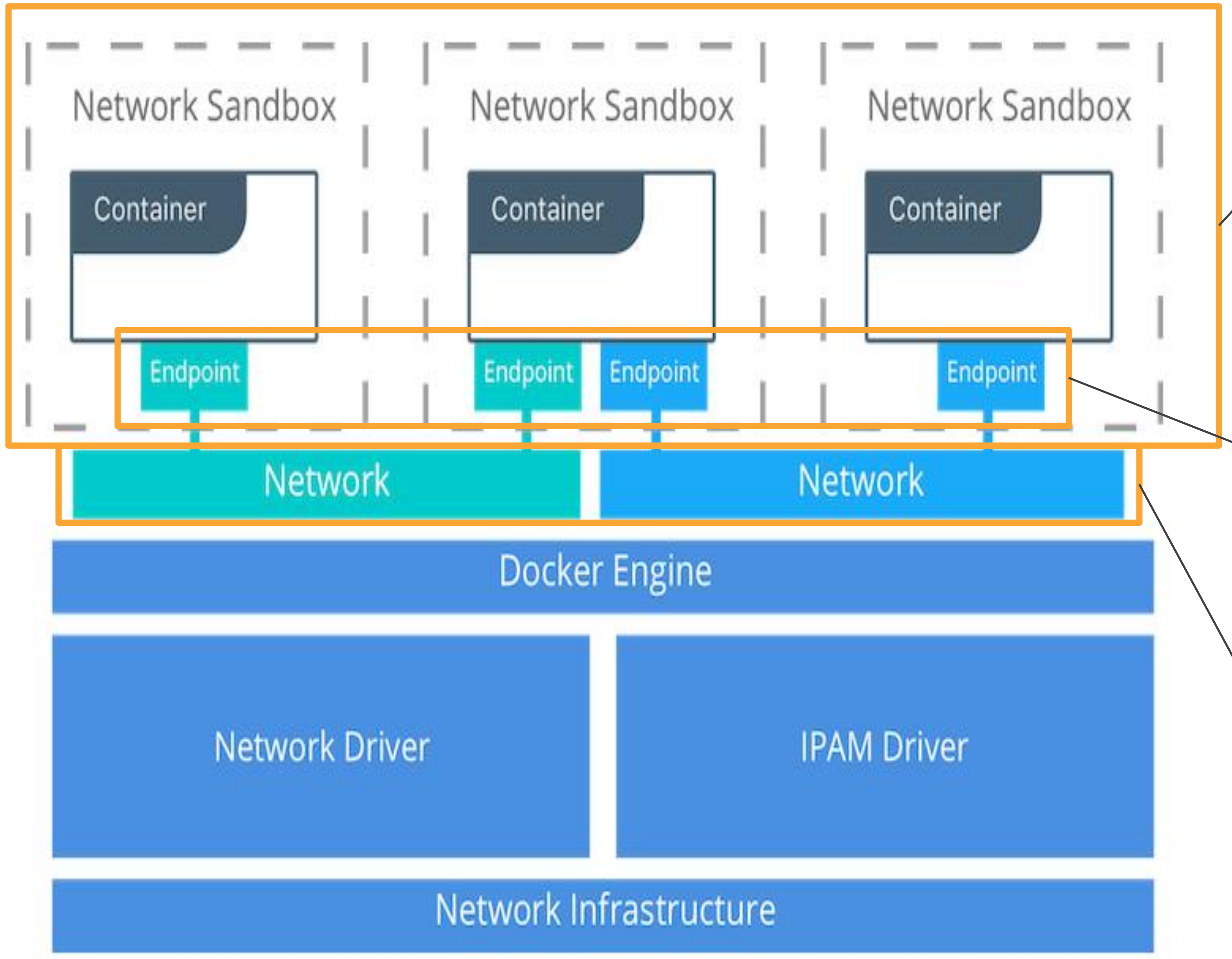
The user can use the following command to clean up networks which aren’t used by any containers:

```
$ docker network prune
```

Container Networking Model

Container Networking Model (CNM)

The CNM provides portability to applications across diverse infrastructures.



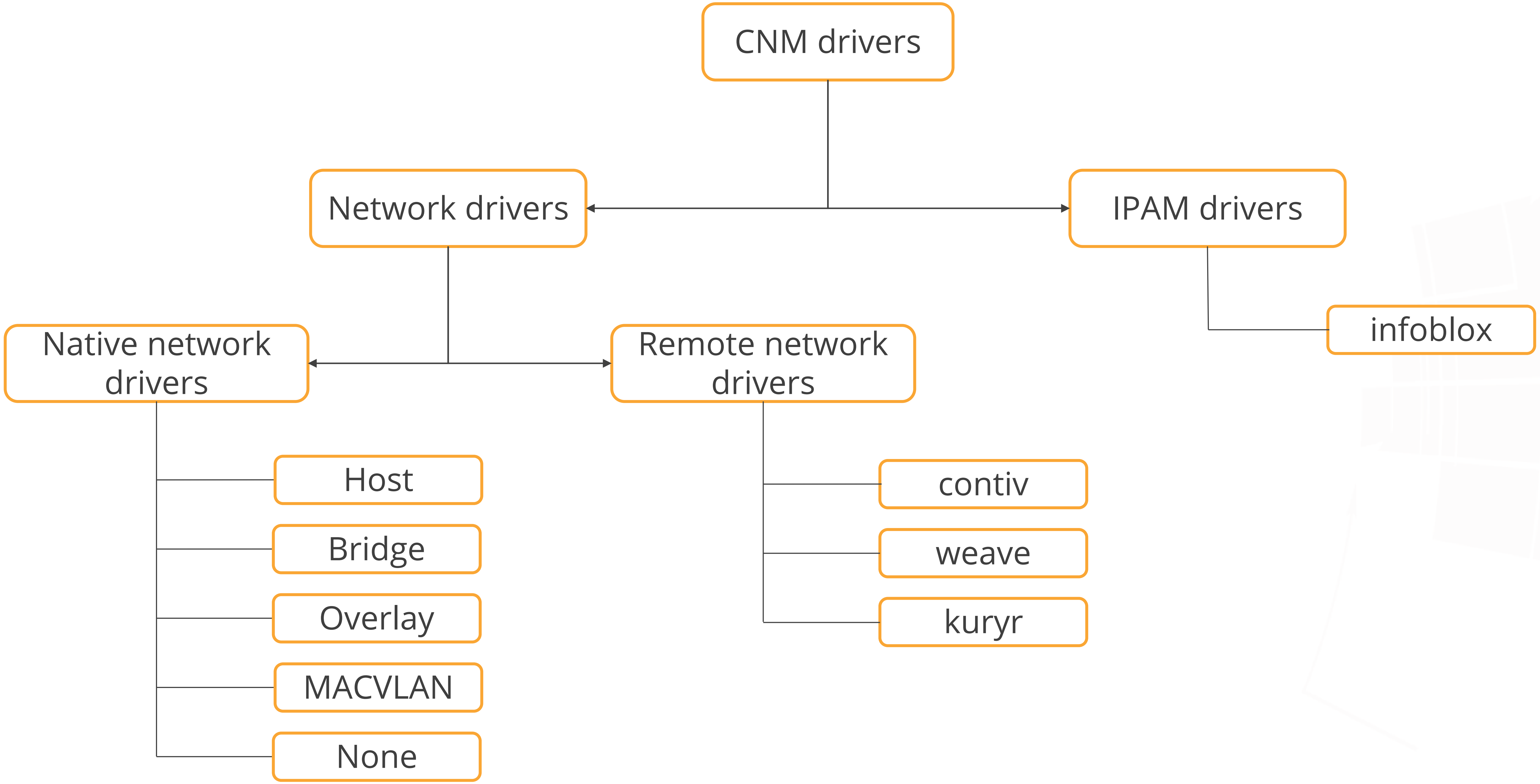
A sandbox is composed of a container's network stack configuration.

These endpoints join the sandboxes to a network.

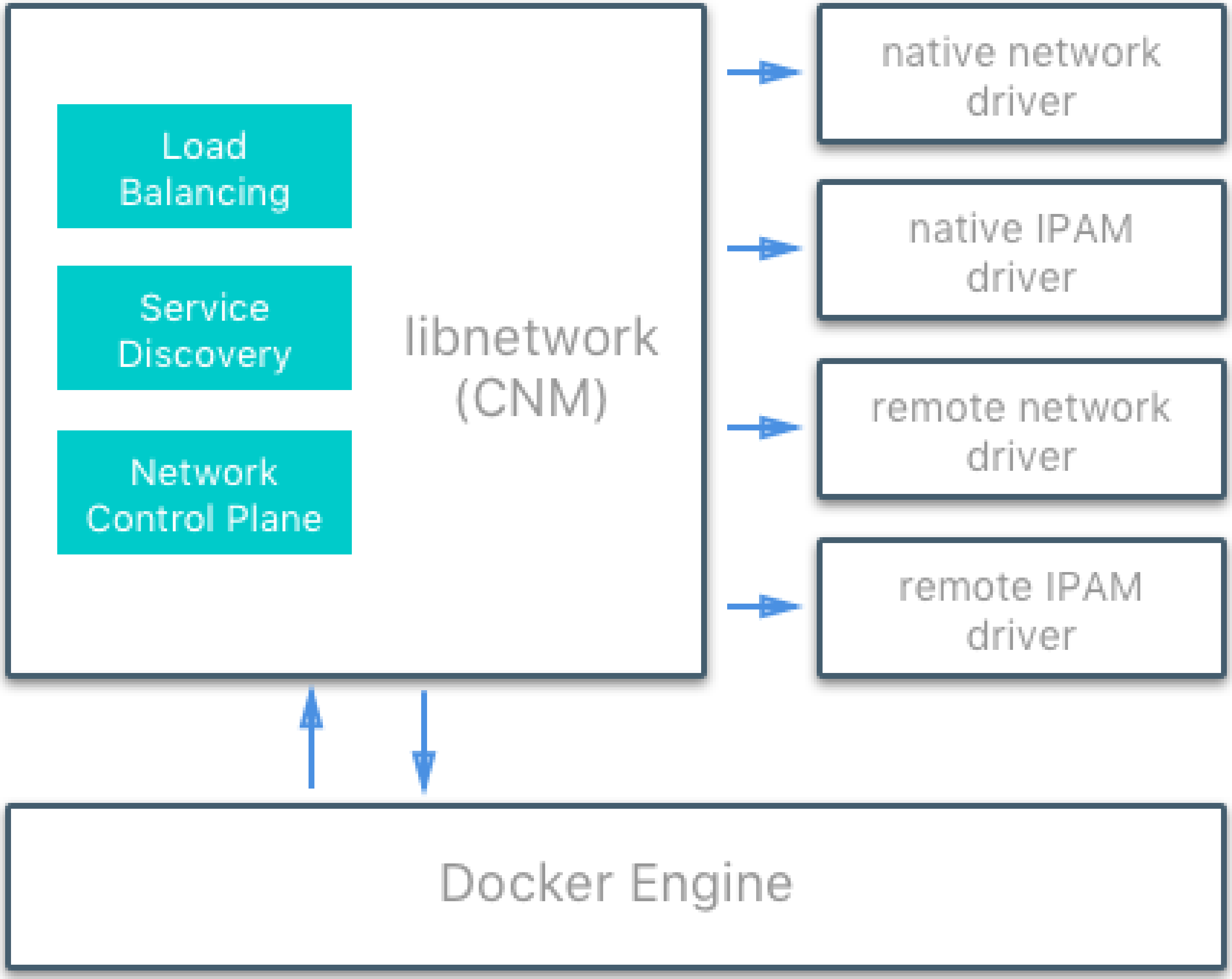
A Network is the collection of endpoints.

https://success.docker.com/article/networking#thecontainernetworkingmodel

Container Networking Model: Drivers



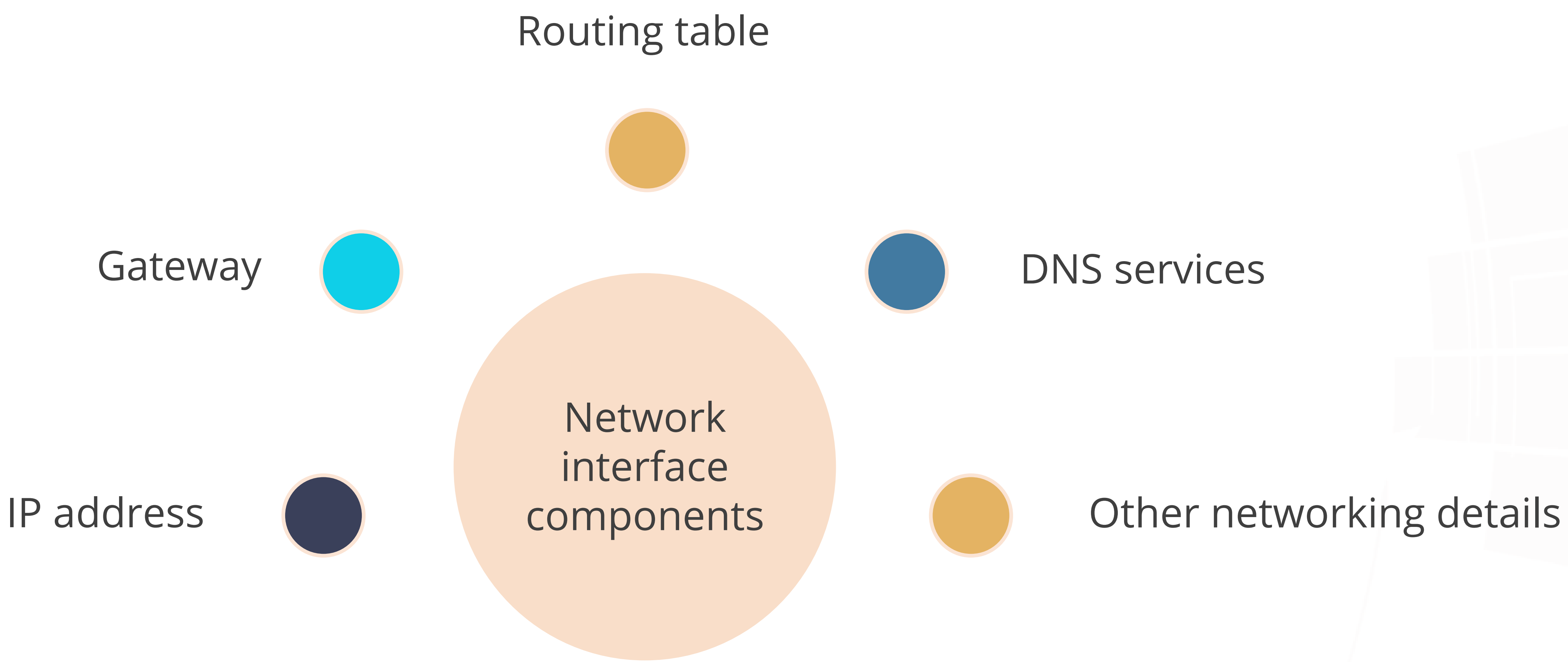
Interaction between Docker Engine, CNM, and Network Drivers



Source: https://success.docker.com/article/networking#thecontainernetworkingmodel

Container Networking

Networking from Container Point of View



Networking from Container Point of View

Published ports:

Making a port available using *--publish* or *-p* will create a firewall rule that map a container port to the port present on a Docker host. Examples are provided in the following table:

Flag value	Description
-p 8080:80	TCP port 80 in the container is mapped to port 8080 on the Docker host.
-p 192.168.1.100:8080:80	TCP port 80 in the container is mapped to port 8080 on the Docker host for connections to host IP 192.168.1.100.
-p 8080:80/udp	UDP port 80 in the container is mapped to port 8080 on the Docker host.
-p 8080:80/tcp -p 8080:80/udp	TCP port 80 in the container is mapped to TCP port 8080 on the Docker host and UDP port 80 in the container is mapped to UDP port 8080 on the Docker host.

Networking from Container Point of View

IP address:

- An IP address is assigned to a container for every Docker network that it connects to.
- *--network* is used to connect a container to a single network.
- The *docker network connect* is used to connect a running container to multiple networks.
- The IP address can be specified while connecting the container to a network by using *--ip* or *--ip6* flags.

Hostname:

- Container ID in the Docker is the default hostname of the container.
- A hostname is overridden by using *--hostname*.
- Additional network alias is specified by using *--alias* flag for the container on an existing network.

Networking from Container Point of View

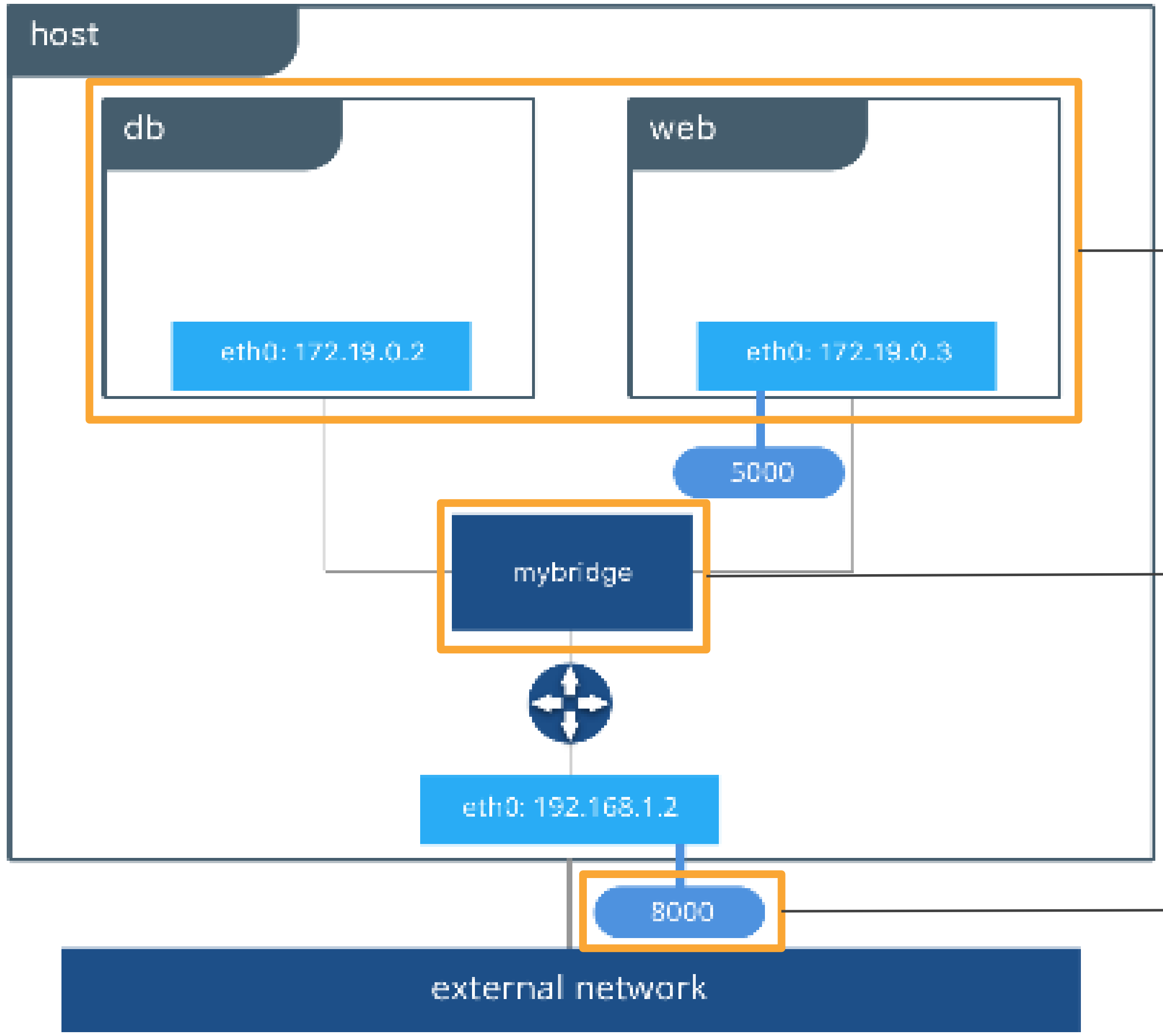
DNS services:

A container inherits the DNS settings of the Docker daemon, including the /etc/hosts and /etc/resolv.conf.

Flag	Description
--dns	IP address of a DNS server. Multiple --*dns* flags are used to specify multiple DNS servers.
--dns-search	Searches non-fully-qualified hostnames. Multiple --*dns*-search flags are used to specify multiple DNS search prefixes.
--dns-opt	Represents a DNS option and its value.
--hostname	Hostname of a container.

Use Cases of Network Drivers

Bridge Network Driver: Use Case



db and *web* are containers of an application called *pets*. This application is available on *<host-ip>:8000.*

mybridge is helping containers *web* to interact with *db* by its container name. This driver is a local scope driver.

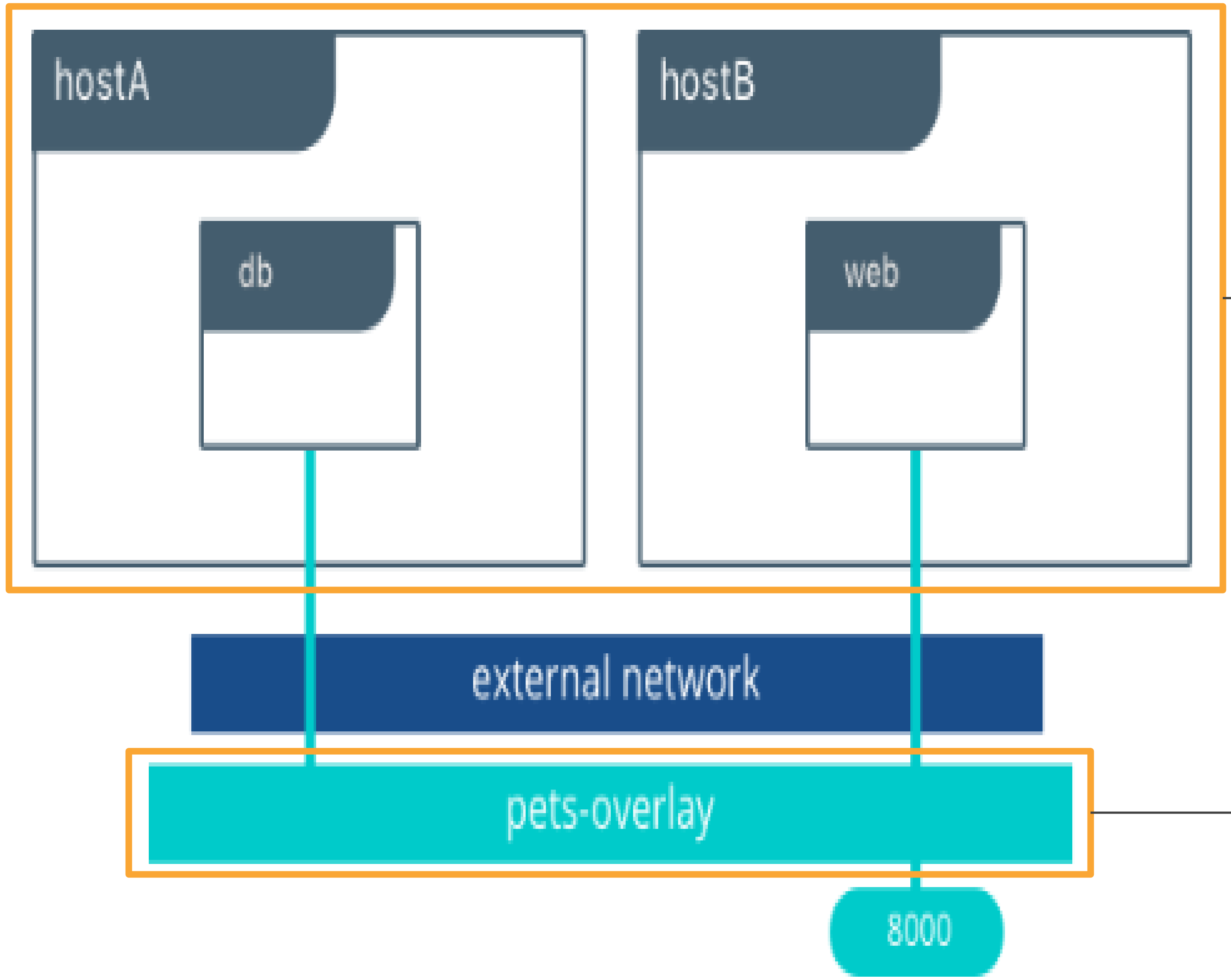
An application *pets* is served on the host at port 8000.

NOTE

The discovery of service is done automatically by the Docker bridge because they are on the same network.

Source: https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/

Overlay Network Driver: Use Case

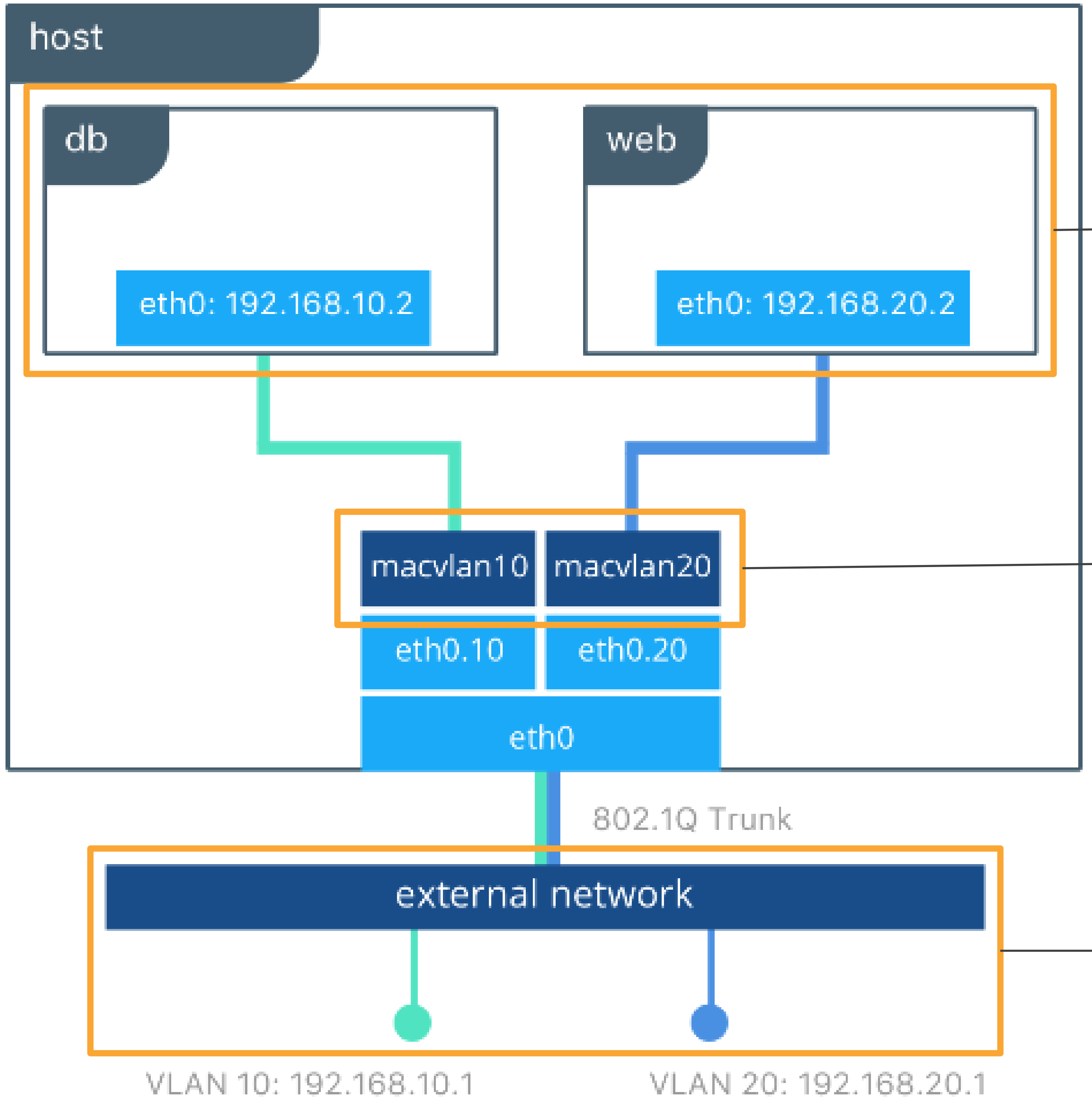


The application *pets* is deployed on *db* and *web.* These two containers are present on different hosts.

The overlay network, which is created by a Universal Control Panel, allows the *db* and *web* to connect to each other.

Source: https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/

MACVLAN Network Driver: Use Case



db and *web* are present on the same host and connected to different MACVLAN networks.

Two MACVLAN networks are created and joined to different sub-interfaces.

The external networks are specific for specific containers.

Source: https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/

Ports

Identifying Ports

Role of port:

The host port is bound to the container's port allowing the container to connect to the external environment.

Use *docker ps* to find all the ports mapped:

Command:

$ docker ps

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
b650456536c7	busybox:latest	top	About an hour ago	Up About an hour	0.0.0.0:1234->9876/tcp, 0.0.0.0:4321->7890/tcp

NAMES
test

Host port

Container port

Publishing and Exposing Ports

Publishing Ports

Ways to publish swarm service port to hosts that are present outside the swarm:

- Using the routing mesh
- Bypassing the routing mesh

Using the routing mesh:

Use *--publish <PUBLISHED-PORT>:<SERVICE-PORT> flag* in order to publish a service's ports externally to the swarm.

Example:

```
$ docker service create --name my_web \
                        --replicas 3 \
                        --publish published=8080,target=80 \
                        nginx
```

Publishing Ports

Bypassing the routing mesh:

Use the *mode=host* option to the *--publish* flag in order to publish a service's port directly on the node where it is running.

Example:

```
$ docker service create \
        --mode global \
        --publish mode=host,target=80,published=8080 \
        --name=nginx \
        nginx:latest
```

Exposing Ports

Exposing ports:

Using --*expose* exposes the ports or range of ports in the container.

Example: Let us expose port 80 without publishing the port

```
$ docker run --expose 80 ubuntu bash
```

Assisted Practice
Publishing Ports

Problem Statement: You are required to publish a swarm service's port to external hosts in different ways so that it can be accessed externally.

Steps to Perform:

1. Publishing a swarm service's port using the routing mesh.
2. Check whether your service has started on published port or not.
3. Publishing a swarm service's port directly on the swarm node.

Traffic

Traffic

Inbound Traffic for Swarm Management:

Swarm mode port	Purpose
TCP port 2377	Cluster management and raft sync communications
TCP and UDP port 7946	Communication between all nodes
UDP port 4789	Overlay network traffic

While using overlay network with the encryption option, ensure that the IP protocol 50 (ESP) traffic is allowed.

Traffic

Network ports and protocols that Swarm cluster components listen on:

Cluster components	Port and protocols	Purpose
Swarm manager	Inbound 80/tcp (HTTP)	Allows *docker pull* commands to work
	Inbound 2375/tcp	Allows Docker Engine CLI commands to the Engine daemon
	Inbound 3375/tcp	Allows Engine CLI commands to the swarm manager
	Inbound 22/tcp	Allows remote management through SSH
Service discovery	Inbound 80/tcp (HTTP)	Allows *docker pull* commands to work
	Inbound *Discovery service port*	Requires setting to the port that the backend discovery service listens on
	Inbound 22/tcp	Allows remote management through SSH

Traffic

Network ports and protocols that Swarm cluster components listen on:

Cluster components	Port and protocols	Purpose
Swarm nodes	Inbound 80/tcp (HTTP)	Allows *docker pull* commands to work
	Inbound 2375/tcp	Allows Engine CLI commands to the Docker daemon
	Inbound 22/tcp	Allows remote management through SSH
Custom, cross-host container networks	Inbound 7946/tcp	Allows discovery of other container networks
	Inbound 7946/udp	Allows discovery of container networks
	Inbound *<store-port>/tcp*	It is a network key-value store service port
	4789/udp	Required for the container overlay network
	ESP packets	Required for encrypted overlay networks

Assisted Practice
Configure Docker to Use External DNS

Problem Statement: You have been asked to configure your Docker daemon to use external DNS so that it can be used to pull images from an external IP address.

Steps to Perform:

1. Navigate to Docker Daemon config file *daemon.json*.
2. In daemon.json file, add the *dns* key with one or more IP addresses.
3. Restart the Docker Daemon.
4. Pull an image from external DNS to check if docker can use external IP address.

Docker Link

Docker Link

Docker containers have other means apart from using the network port mapping to connect to one another. Docker containers also communicate using the linking system. Information can be sent to a recipient container from a source container, when the containers are linked.

Docker link feature allows the containers to:

- Discover each other
- Transfer information between containers in secure manner.

Docker Link

Every container that is created will automatically get a name. The name of a container provides two functions:

- Describes the function of the container
- Provides a reference point to Docker

Name the container using *--name* flag:

$ docker run -d -P --name asper training/webapp python app.py

Find the name of the container:

$ docker ps -l

Communication across Links

Create a new container named *db* containing a database:

```
$ docker run -d --name db training/postgres
```

Create a new *web* container and link it with *db* container:

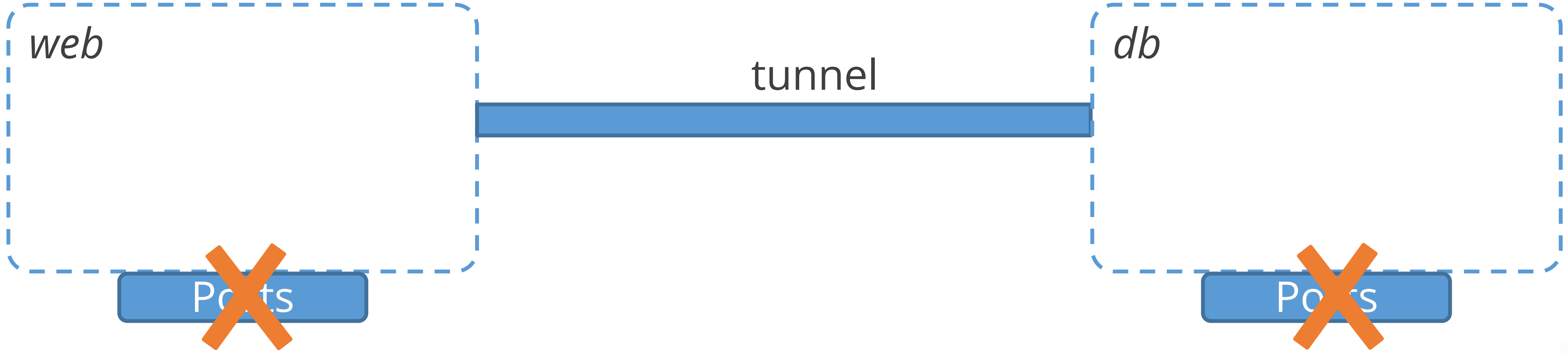
```
$ docker run -d -P --name web --link db:db training/webapp python app.py
```

Inspect the linked containers:

```
$ docker inspect -f "{{ .HostConfig.Links }}" web
```

Communication across Links

How *web* accesses information from the source *db*:



Ways to expose connectivity information between containers by:

- Using Environment variables
- Updating the */etc/hosts* file

Key Takeaways

- A Sandbox contains the configuration of a container's network stack.
- A Network is a collection of endpoints. These endpoints join the Sandbox to the Network.
- A container is given its own networking stack and a network namespace by a none driver, but this driver does not configure interfaces inside the container.
- On creation of a container, the interaction with the outside world is not possible, because the ports are not automatically published.

Thank You