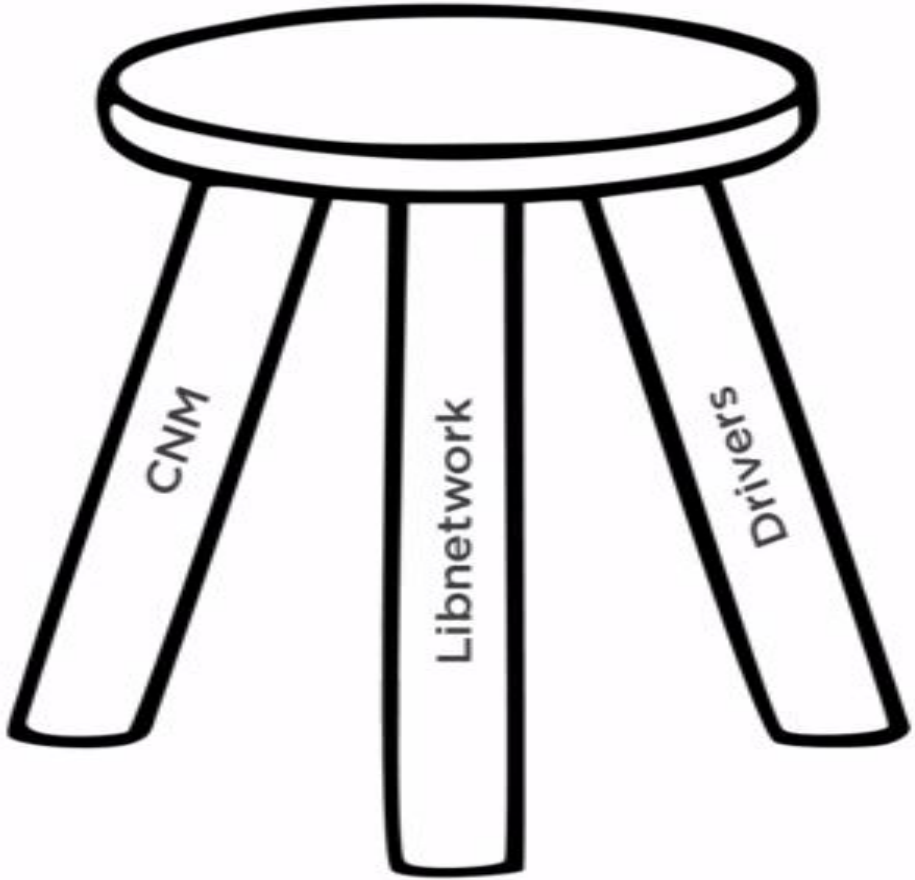
Docker Networking

The Three Pillars of Docker Networking




CNM
Libnetwork
Drivers
Grand design/DNA
of Docker networking


CNM
Libnetwork
Drivers
Spec from Docker,
Inc. Now open source
Grand design/DNA
of Docker networking
https://github.com/docker/libnetwork/blob/master/docs/design.md

CNM VS CNI

Container network model	AKA	Container network interface
Docker	Speciality	Kubernetes
Docker, Inc.	Origin	CoreOS, Inc.

CNM

Sandbox

A.k.a. namespace

Isolated area of OS

Contains full network stack

Endpoint

Network interface

E.g. eth0

Network

Connected endpoints

Central place for all Docker networking logic, API, UX etc...

https://github.com/docker/libnetwork

X-platform Pluggable

Written in Go/Golang

Real-world implementation of CNM by Docker, Inc.

CNM → Libnetwork → Drivers

DNA:
- Sandbox
- Endpoint
- Network

Spec from Docker, Inc.
Now open source

Grand design/DNA of Docker networking

https://github.com/docker/libnetwork/blob/master/docs/design.md

Central place for all Docker networking logic, API, UX etc...

https://github.com/docker/libnetwork

X-platform Pluggable

Written in Go/Golang

Real-world implementation of CNM by Docker, Inc.

CNM → Libnetwork → Drivers

DNA:
- Sandbox
- Endpoint
- Network

Spec from Docker, Inc.
Now open source

Grand design/DNA of Docker networking

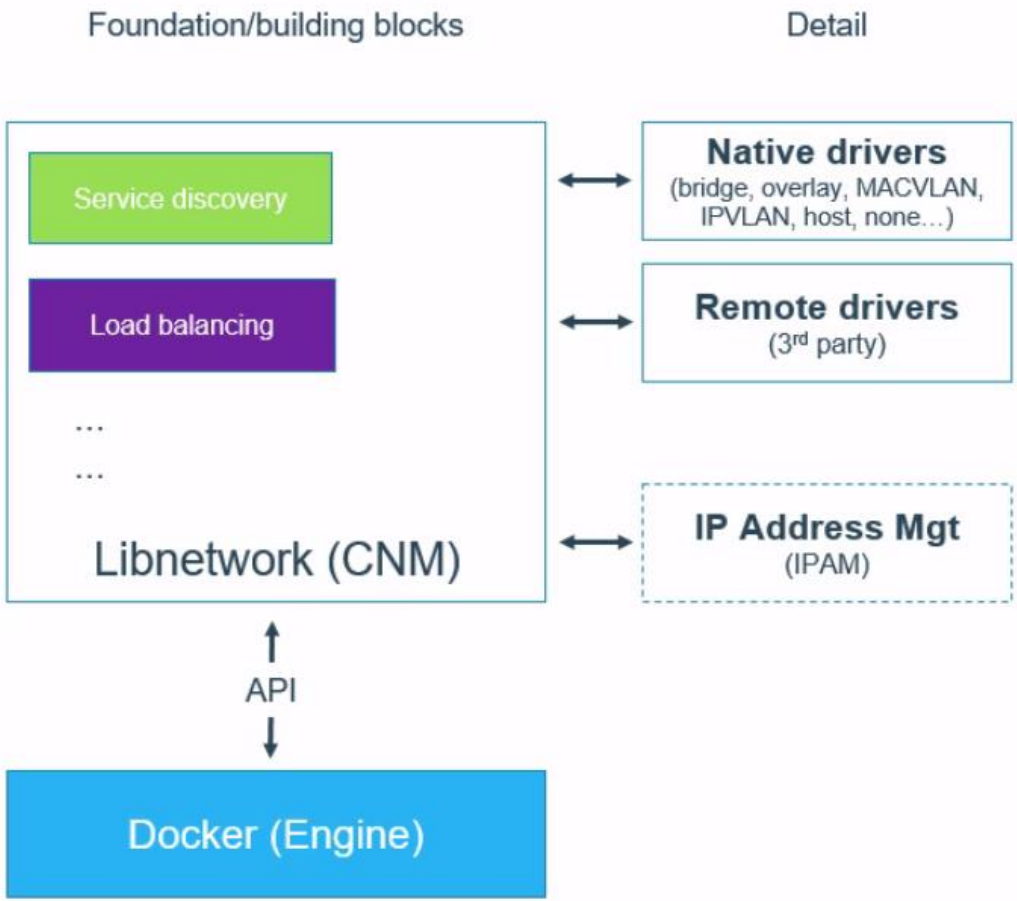
https://github.com/docker/libnetwork/blob/master/docs/design.md

Network-specific detail
- Overlay
- MACVLAN
- IPVLAN
- Bridge

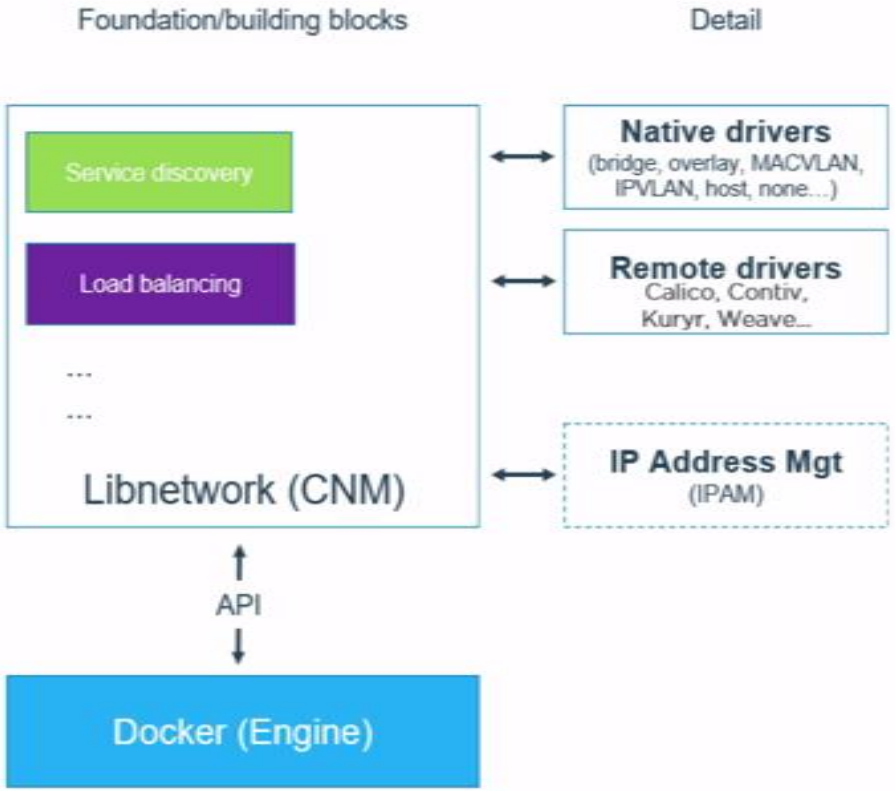
"Local" = native

"Remote" = 3rd party


CNM
Libnetwork
Drivers
Design /DNA
Control plane & management plane
Data plane


Foundation/building blocks
Detail
Service discovery
Load balancing
...
...
Libnetwork (CNM)
Native drivers
(bridge, overlay, MACVLAN, IPVLAN, host, none…)
Remote drivers
(3rd party)
IP Address Mgt
(IPAM)
API
Docker (Engine)

- Distributed KV store
- Network gossip
- Encryption
- Service discovery
- IP address management




$ docker network

```
root@dockerpract1:/home/user# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1487578a0b71        bridge              bridge              local
9ea93df7c85b        host                host                local
9b6b4efe6d84        none                null                local
root@dockerpract1:/home/user#
```

SCOPE

swarm = multi-host

local = single-host

AGENDA

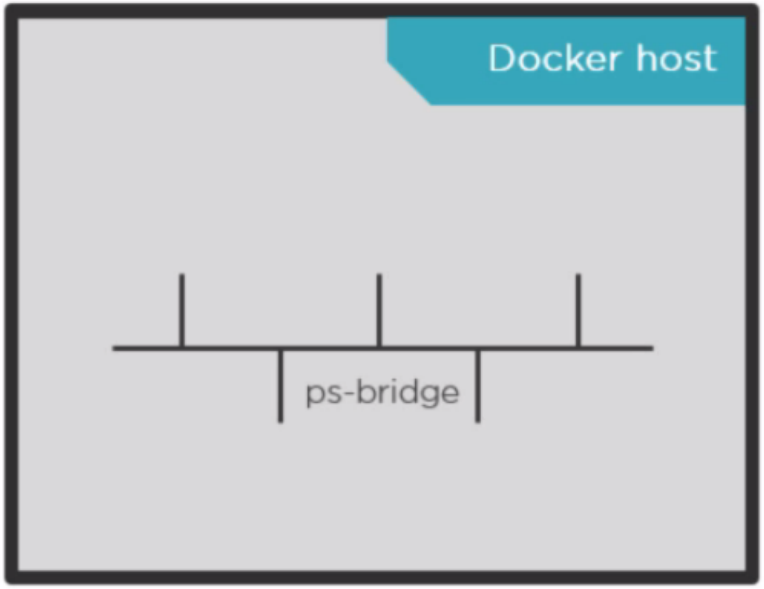
- Single-host networking
- Multi-host networking
- Working with existing networks

Single-host Networking

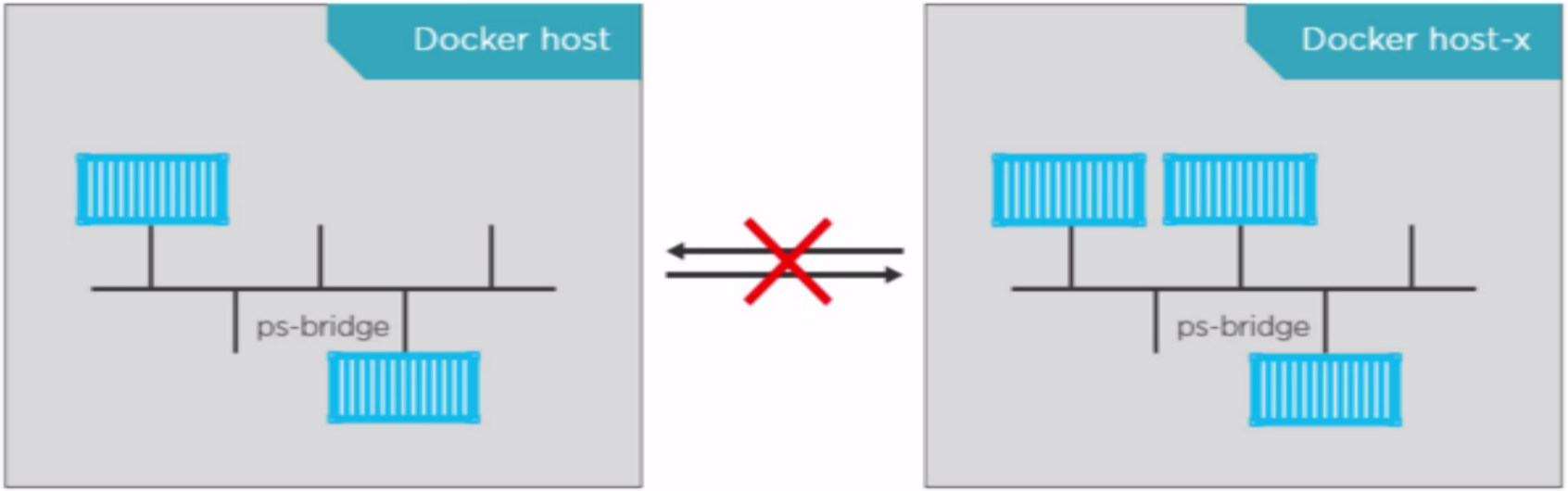
With the **bridge** driver (Linux)

≈ **nat** driver (Windows)

Single-host



Single-host



Create NetWork

Docker network create -d bridge --subnet 10.1.0.0/24 qt-bridge

```
root@dockerpract1:/home/user# docker network create -d bridge --subnet 10.0.0.0/
24 ps-bridge
54ac2de113229954b4b895dd467f985f7bfcf3c34d9ef1d7459c23ebc084e49c
root@dockerpract1:/home/user# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1487578a0b71        bridge              bridge              local
9ea93df7c85b        host                host                local
9b6b4efe6d84        none                null                local
54ac2de11322        ps-bridge           bridge              local
root@dockerpract1:/home/user#
```

CREATE NETWORK (CONTD..)

```
root@dockerpract1:/home/user# docker inspect ps-bridge
[
    {
        "Name": "ps-bridge",
        "Id": "54ac2de113229954b4b895dd467f985f7bfcf3c34d9ef1d7459c23ebc084e49c"
,
        "Created": "2017-02-21T09:37:52.994323939Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.0.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
root@dockerpract1:/home/user#
```

Create Containers in custom Bridge

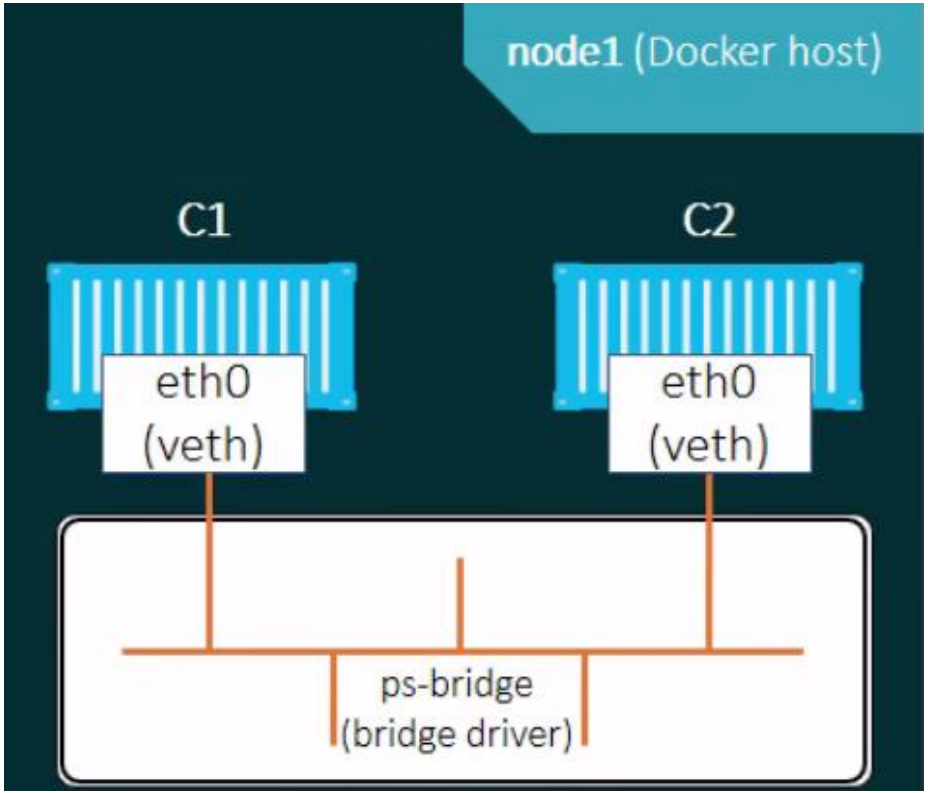
```
docker run -dt --name c1 --network ps-bridge ubuntu sleep 1d

docker run -dt --name c2 --network ps-bridge ubuntu sleep 1d
```

Inspect network (docker network inspect ps-bridge)

Create Containers in custom Bridge


```
        "Containers": {
            "6a3cc4ac3df97141e922dad90169aa565ed230e634e4386a3e85fa6051573948":
{
                "Name": "c1",
                "EndpointID": "e5200afd291b20142b3742be586a18c03b2673f761a6b461c
0ab9b44cb5a7d38",
                "MacAddress": "02:42:0a:00:00:02",
                "IPv4Address": "10.0.0.2/24",
                "IPv6Address": ""
            },
            "c9eff113dd6c1f590f2846346986512a3bbcbe4abd248e62e9d528d682fc022c":
{
                "Name": "c2",
                "EndpointID": "327bef4cfb84490a3a4213a2ce15237ea113173f71c6f473a
1871570821cbf11",
                "MacAddress": "02:42:0a:00:00:03",
                "IPv4Address": "10.0.0.3/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
```

Network Topology For bridge



Test

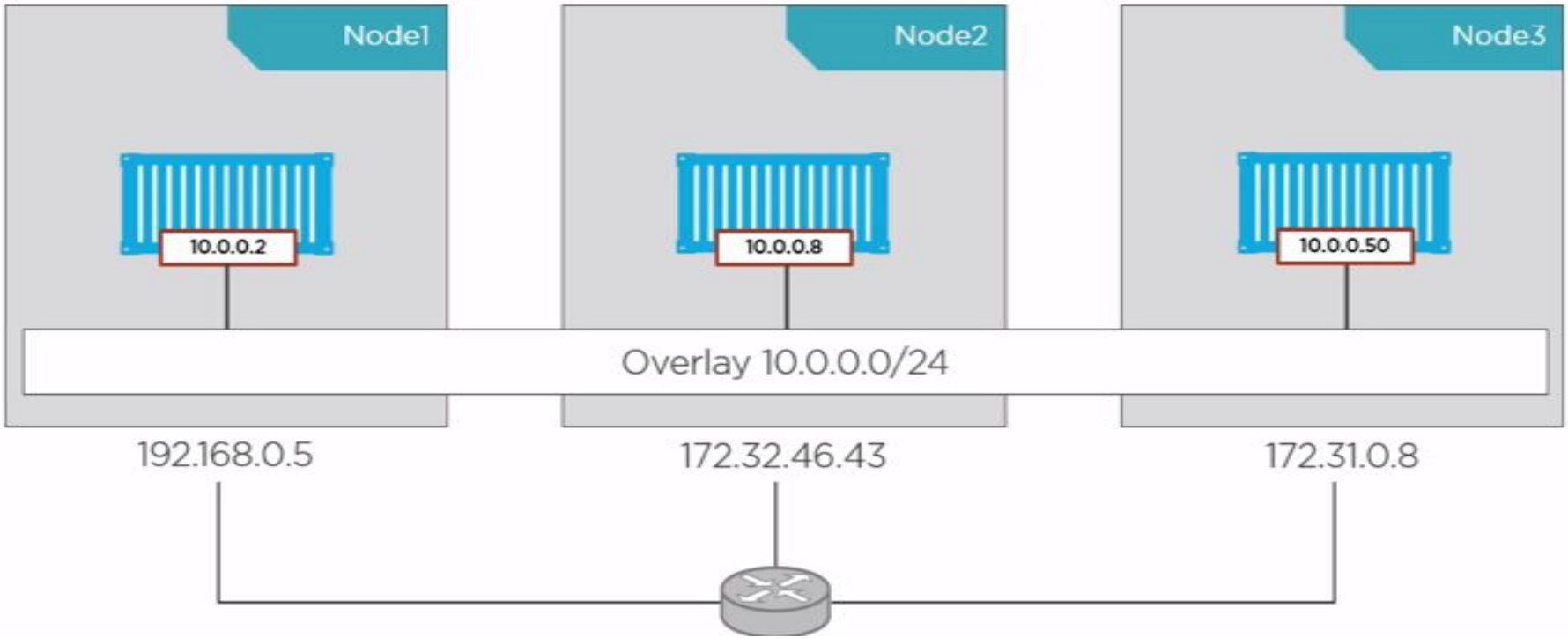
Ping other container (c1) from c2

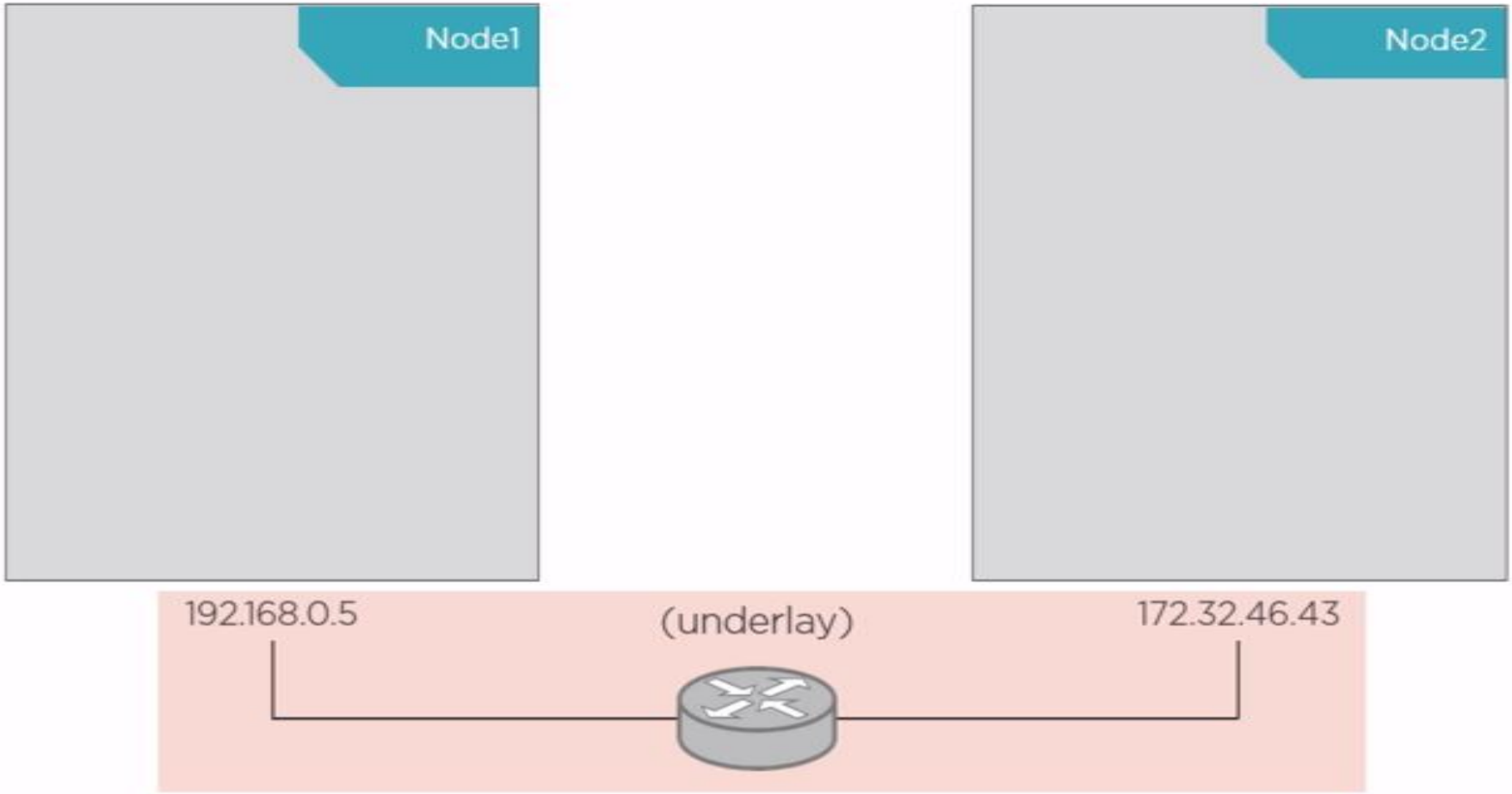
```
root@c9eff113dd6c:/# ping c1
PING c1 (10.0.0.2) 56(84) bytes of data.
64 bytes from c1.ps-bridge (10.0.0.2): icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from c1.ps-bridge (10.0.0.2): icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from c1.ps-bridge (10.0.0.2): icmp_seq=3 ttl=64 time=0.079 ms
64 bytes from c1.ps-bridge (10.0.0.2): icmp_seq=4 ttl=64 time=0.099 ms
^C
--- c1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.047/0.075/0.099/0.018 ms
root@c9eff113dd6c:/#
```

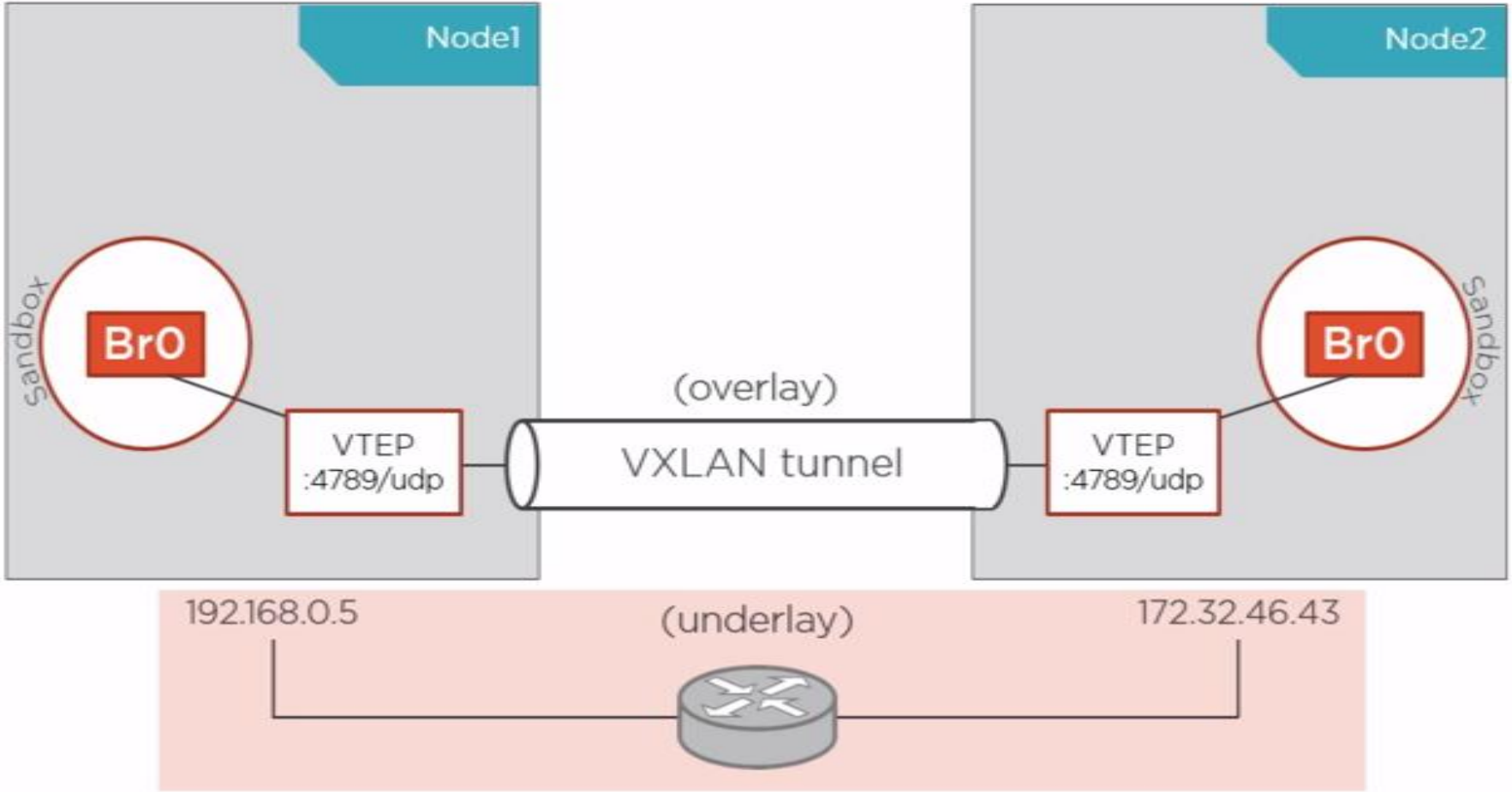
Multi-host Networking

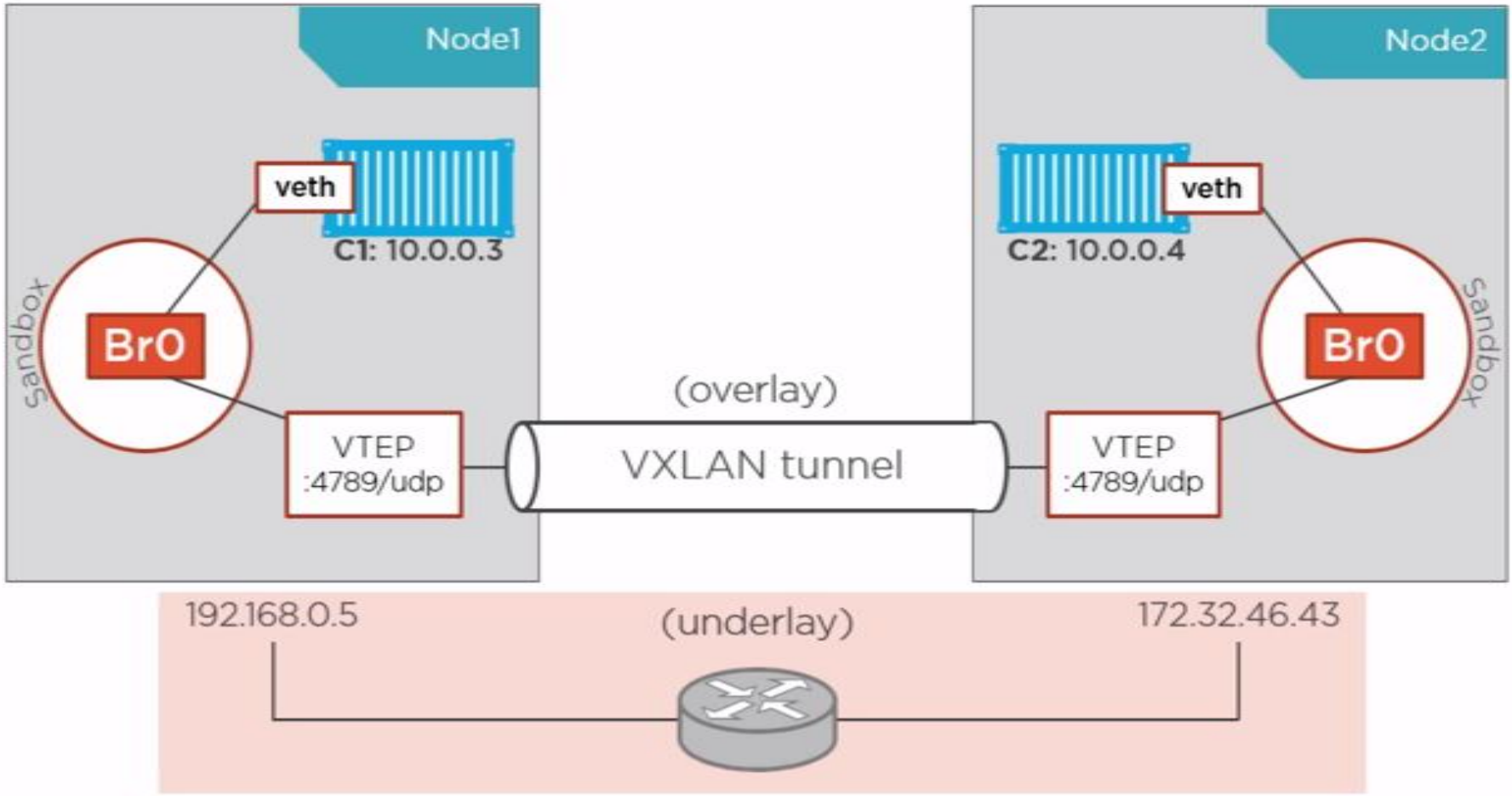
with the Docker overlay driver

Multi-host Overlay




Node1
Node2
192.168.0.5
(underlay)
172.32.46.43


Node1
Node2
sandbox
Br0
VTEP
:4789/udp
(overlay)
VXLAN tunnel
VTEP
:4789/udp
Br0
sandbox
192.168.0.5
(underlay)
172.32.46.43


Node1
veth
C1: 10.0.0.3
sandbox
Br0
VTEP
:4789/udp
(overlay)
VXLAN tunnel
Node2
veth
C2: 10.0.0.4
Br0
sandbox
VTEP
:4789/udp
192.168.0.5
(underlay)
172.32.46.43

Steps for multi-host networking

Install docker on two or more hosts

On the first machine execute “docker swarm init”

Execute docker swarm join in other hosts

Execute docker network ls and you should be able to see ingress with DRIVER as overlay

Execute “docker node ls” to find out nodes

Note: Following ports should be open ====>

2789/udp
7946/tcp/udp
2377/tcp

CREATION OF OVERLAY NETWORK

“Docker network create -d overlay qt-over” to create network

Try executing “docker network ls” on other container you will not see the qt-over as docker follows lazy approach

To see the qt-over start a container.

“docker service create --name qt-svc --network qt-over --replicas 3 hello-world sleep 1d”

To see the status execute on host1 “docker service ps ps-svc”