KODEKLOUD

Course Objectives

- Scheduling
- Logging Monitoring
- Application Lifecycle Management
- Cluster Maintenance
- Security
- Storage
- Troubleshooting
- Core Concepts
- Networking

○ Pre-Requisites – Network, Switching, Routing, Tools

○ Pre-Requisites – DNS, IPAM, Firewalls, LBs

○ Pre-Requisites – Network Namespaces

○ Pre-Requisites – Networking in Docker

○ Networking Configuration on Cluster Nodes

○ Service Networking

○ POD Networking Concepts

○ Network Loadbalancer

○ Ingress

○ Cluster DNS

○ CNI

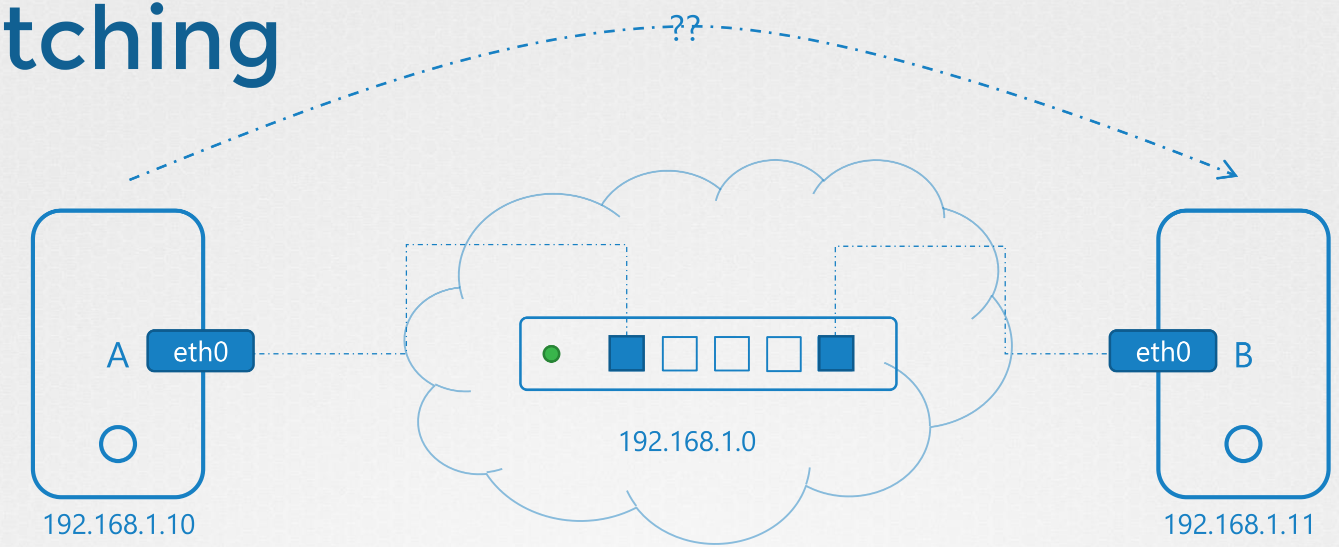
LINUX NETWORKING BASICS

Networking Pre-Requisites

- Switching and Routing
  - Switching
  - Routing
  - Default Gateway
  - NAT
- Linux Interfaces for Virtual Networking
  - Bridge Network
  - VLAN
  - VXLAN
- IP Address Management & Name Resolution
  - DNS
  - IPAM
  - DHCP
- Firewalls
- Load-Balancers

- Tools:
  - Ping
  - NC - NetCat
  - TCPDUMP
  - IPTABLES

Switching



```
ip link
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
```

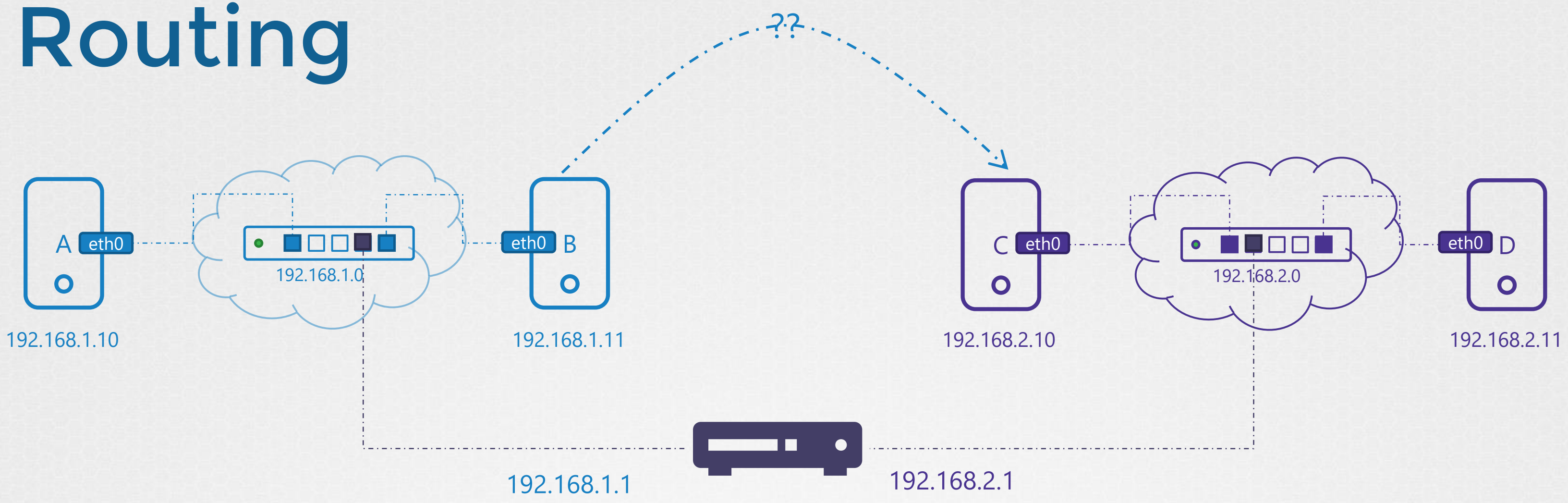
```
ip addr add 192.168.1.10/24 dev eth0
```

```
ip link
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdis
fq_codel state UP mode DEFAULT group default qlen 100
```

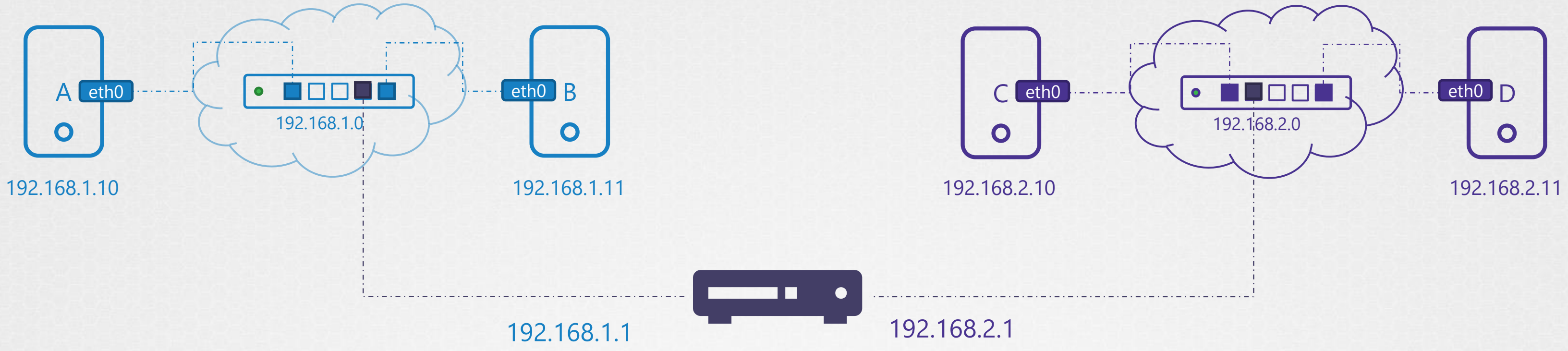
```
ip addr add 192.168.1.11/24 dev eth0
```

```
ping 192.168.1.11
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117
```

Routing


??
A
eth0
192.168.1.0
eth0
B
C
eth0
192.168.2.0
eth0
D
192.168.1.10
192.168.1.11
192.168.2.10
192.168.2.11
192.168.1.1
192.168.2.1

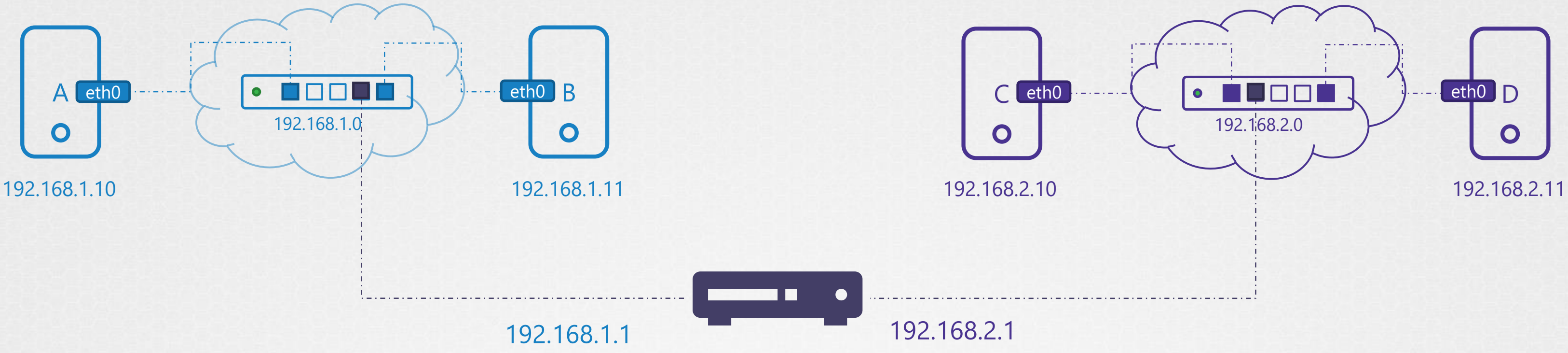
Gateway



```
▶ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

```
▶ ip route add 192.168.2.0/24 via 192.168.1.1
▶ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.2.0     192.168.1.1     255.255.255.0   UG    0      0        0 eth0
```

Gateway

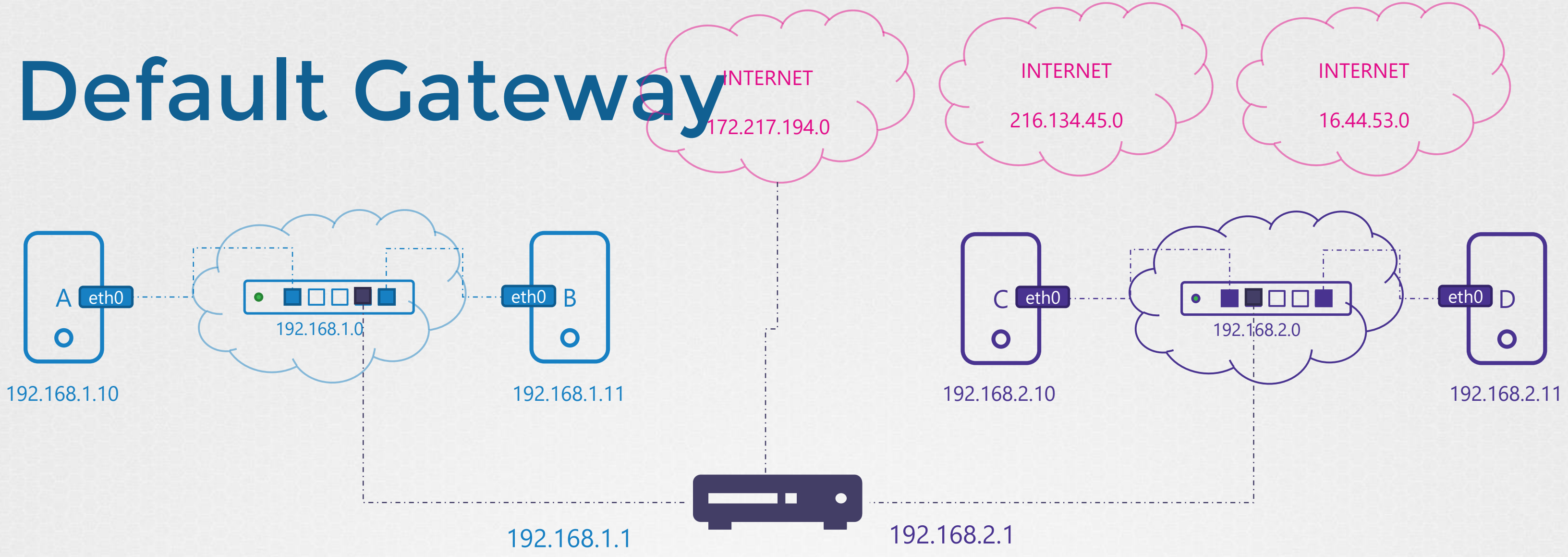


```
ip route add 192.168.1.0/24 via 192.168.2.1
```

```
route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     192.168.2.1     255.255.255.0   UG    0      0        0 eth0
```

Default Gateway



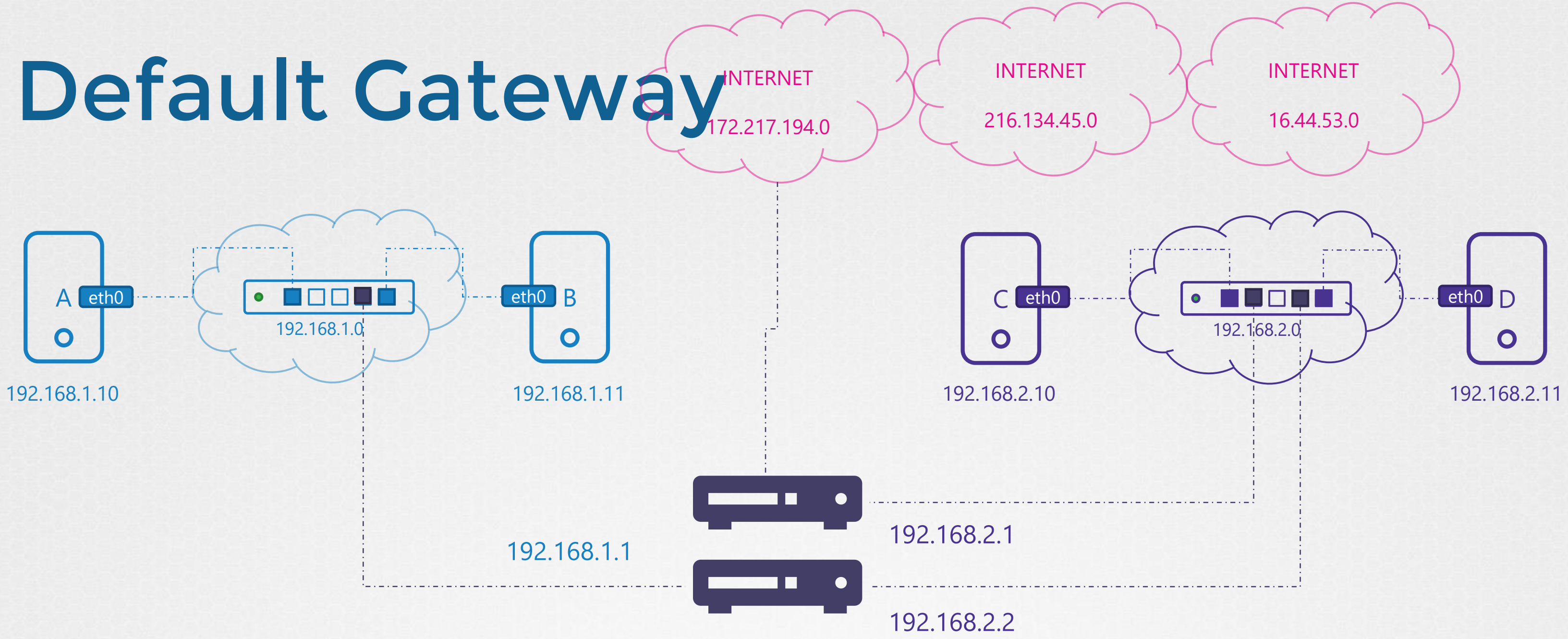
```
ip route add 192.168.1.0/24 via 192.168.2.1
```

```
ip route add default via 192.168.2.1
```

```
route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     192.168.2.1     255.255.255.0   UG    0      0        0 eth0

0.0.0.0         192.168.2.1     255.255.255.0   UG    0      0        0 eth0
192.168.2.0     0.0.0.0         255.255.255.0   UG    0      0        0 eth0
```

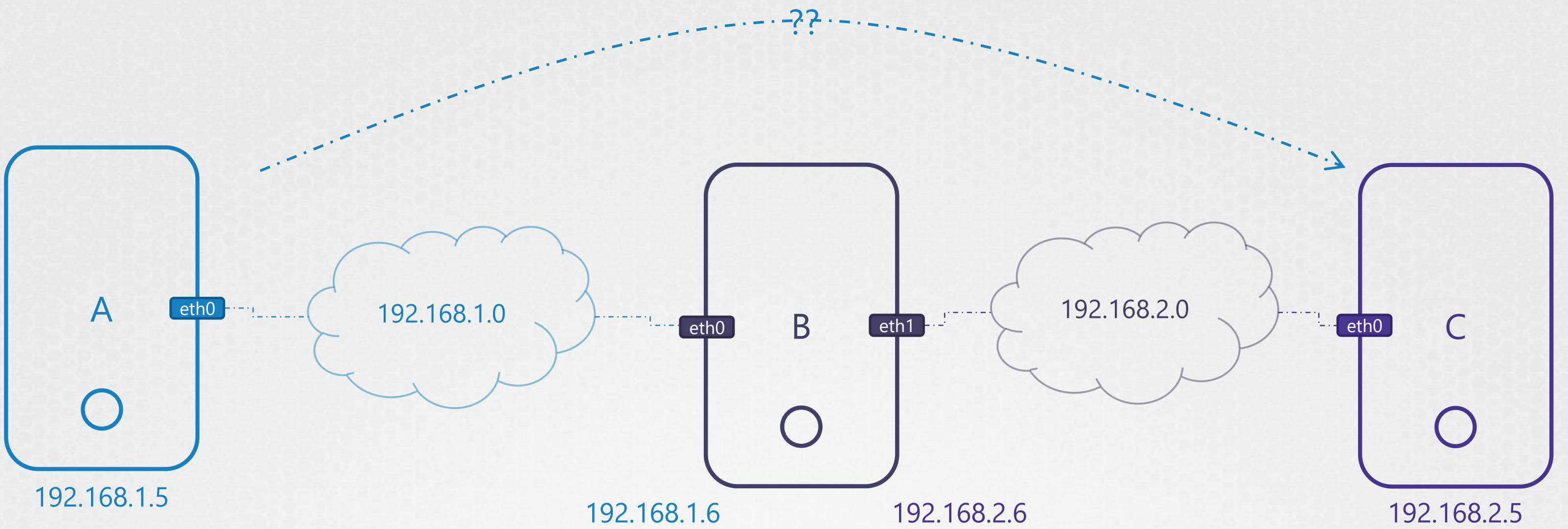
Default Gateway



```
▶ ip route add 192.168.1.0/24 via 192.168.2.2
```

```
▶ route
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.2.1     255.255.255.0   UG    0      0        0 eth0
192.168.1.0     192.168.2.2     255.255.255.0   UG    0      0        0 eth0
```

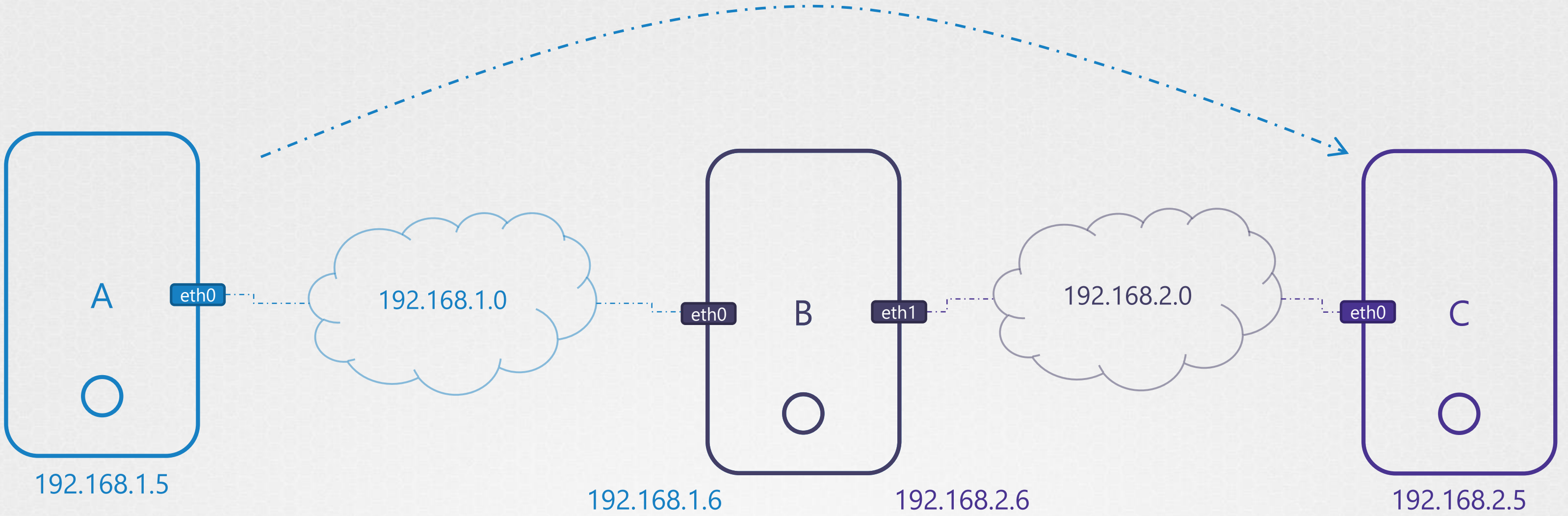


```
ping 192.168.2.5
Connect: Network is unreachable
```

```
ip route add 192.168.2.0/24 via 192.168.1.6
```

```
ip route add 192.168.1.0/24 via 192.168.2.6
```

```
ping 192.168.2.5
```



```
cat /proc/sys/net/ipv4/ip_forward
0
```

```
1.6
```

```
echo 1 > /proc/sys/net/ipv4/ip_forward
1
```

/etc/sysctl.conf

```
…
net.ipv4.ip_forward = 1
…
```

```
ping 192.168.2.5
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
```

Take Aways

```
ip link
```

```
ip addr
```

```
ip addr add 192.168.1.10/24 dev eth0
```

```
ip route
```

```
route
```

```
ip route add 192.168.1.0/24 via 192.168.2.1
```

```
cat /proc/sys/net/ipv4/ip_forward
1
```

KODEKLOUD

Course Objectives

- Scheduling
- Logging Monitoring
- Application Lifecycle Management
- Cluster Maintenance
- Security
- Storage
- Troubleshooting
- Core Concepts
- Networking

○ Pre-Requisites – Network, Switching, Routing, Tools

○ Pre-Requisites – DNS, IPAM, Firewalls, LBs

○ Pre-Requisites – Network Namespaces

○ Pre-Requisites – Networking in Docker

○ Networking Configuration on Cluster Nodes

○ Service Networking

○ POD Networking Concepts

○ Network Loadbalancer

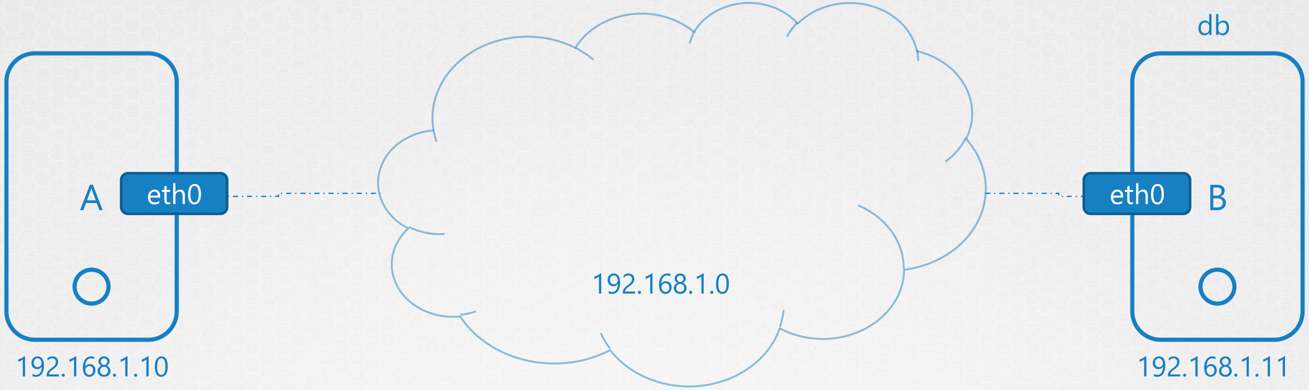
○ Ingress

○ Cluster DNS

○ CNI

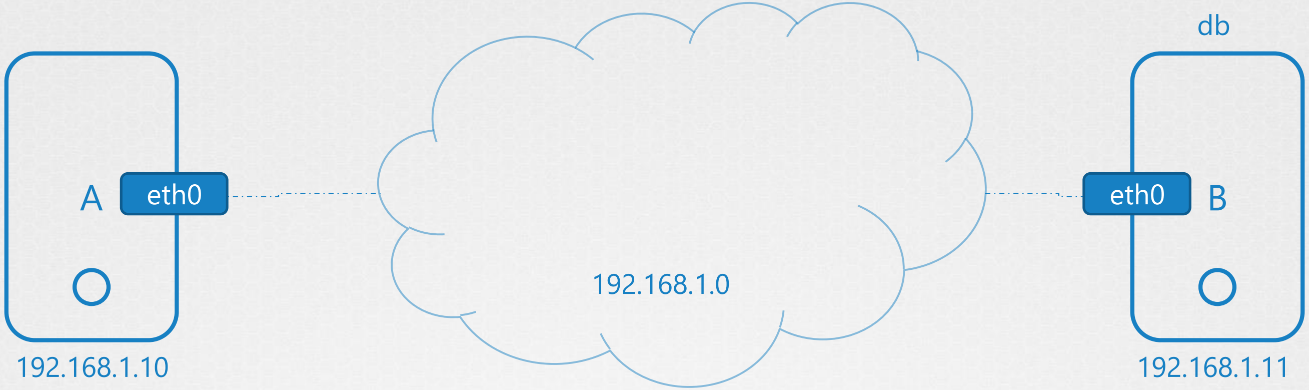
DNS

For the Absolute Beginners



```
▶ ping 192.168.1.11
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117
Reply from 192.168.1.11: bytes=32 time=4ms TTL=117
```

```
▶ ping db
ping: unknown host db
```

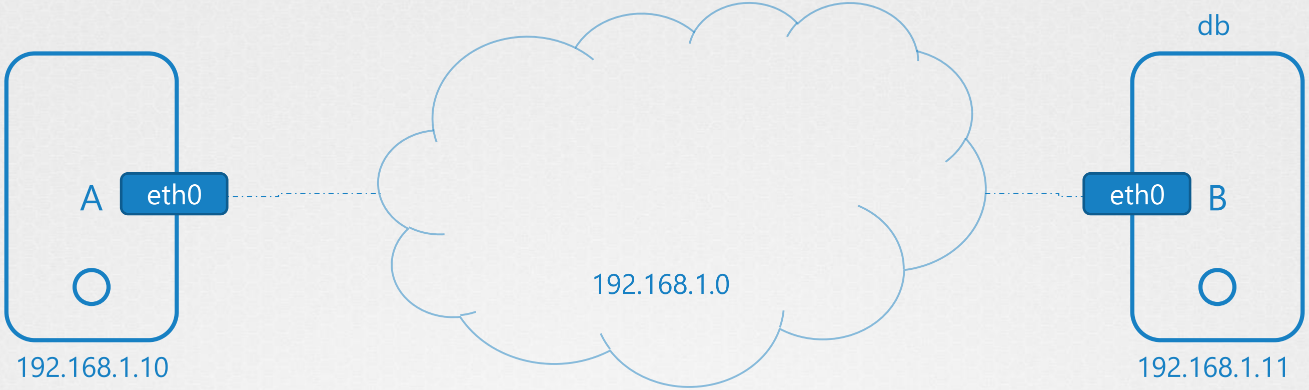


```
▶ ping db
ping: unknown host db
```

```
▶ hostname
host-2
```

```
▶ cat >> /etc/hosts
192.168.1.11          db
```

```
▶ ping db
PING db (192.168.1.11) 56(84) bytes of data.
64 bytes from db (192.168.1.11): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from db (192.168.1.11): icmp_seq=2 ttl=64 time=0.079 ms
```



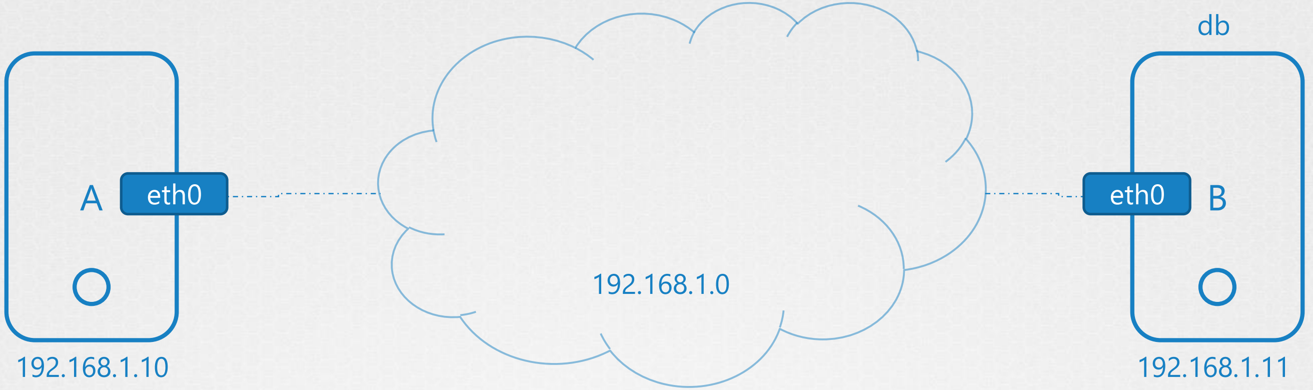
```
▶ cat >> /etc/hosts
192.168.1.11          db
192.168.1.11          www.google.com
```

```
▶ hostname
host-2
```

```
▶ ping db
PING db (192.168.1.11) 56(84) bytes of data.
64 bytes from db (192.168.1.11): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from db (192.168.1.11): icmp_seq=2 ttl=64 time=0.079 ms
```

```
▶ ping www.google.com
PING www.google.com (192.168.1.11) 56(84) bytes of data.
64 bytes from www.google.com (192.168.1.11): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from www.google.com (192.168.1.11): icmp_seq=2 ttl=64 time=0.079 ms
```

Name Resolution



```
▶ cat >> /etc/hosts
192.168.1.11        db
192.168.1.11        www.google.com
```

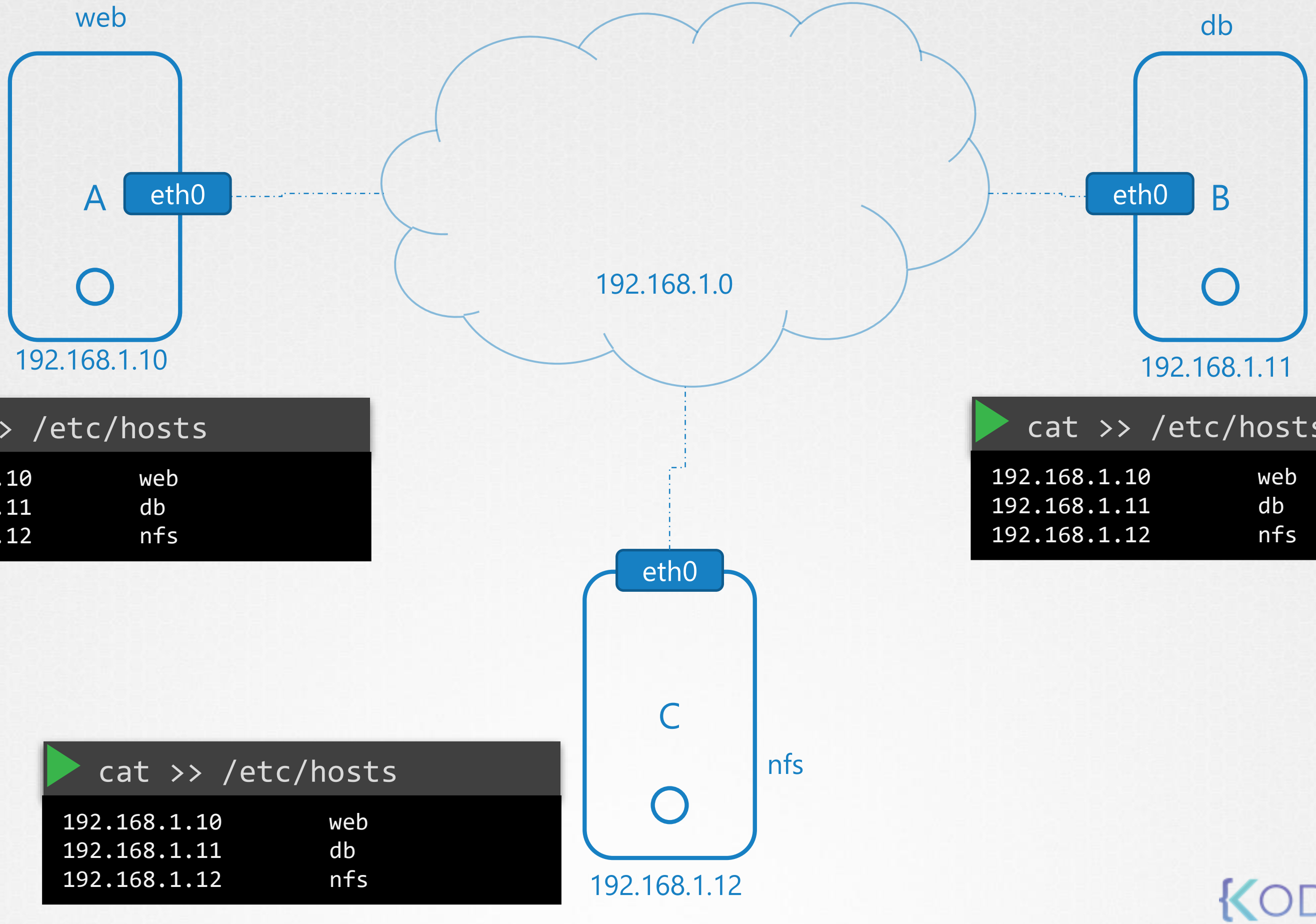
```
▶ ping db
```

```
▶ ssh db
```

```
▶ curl http://www.google.com
```

```
▶ hostname
host-2
```

Name Resolution

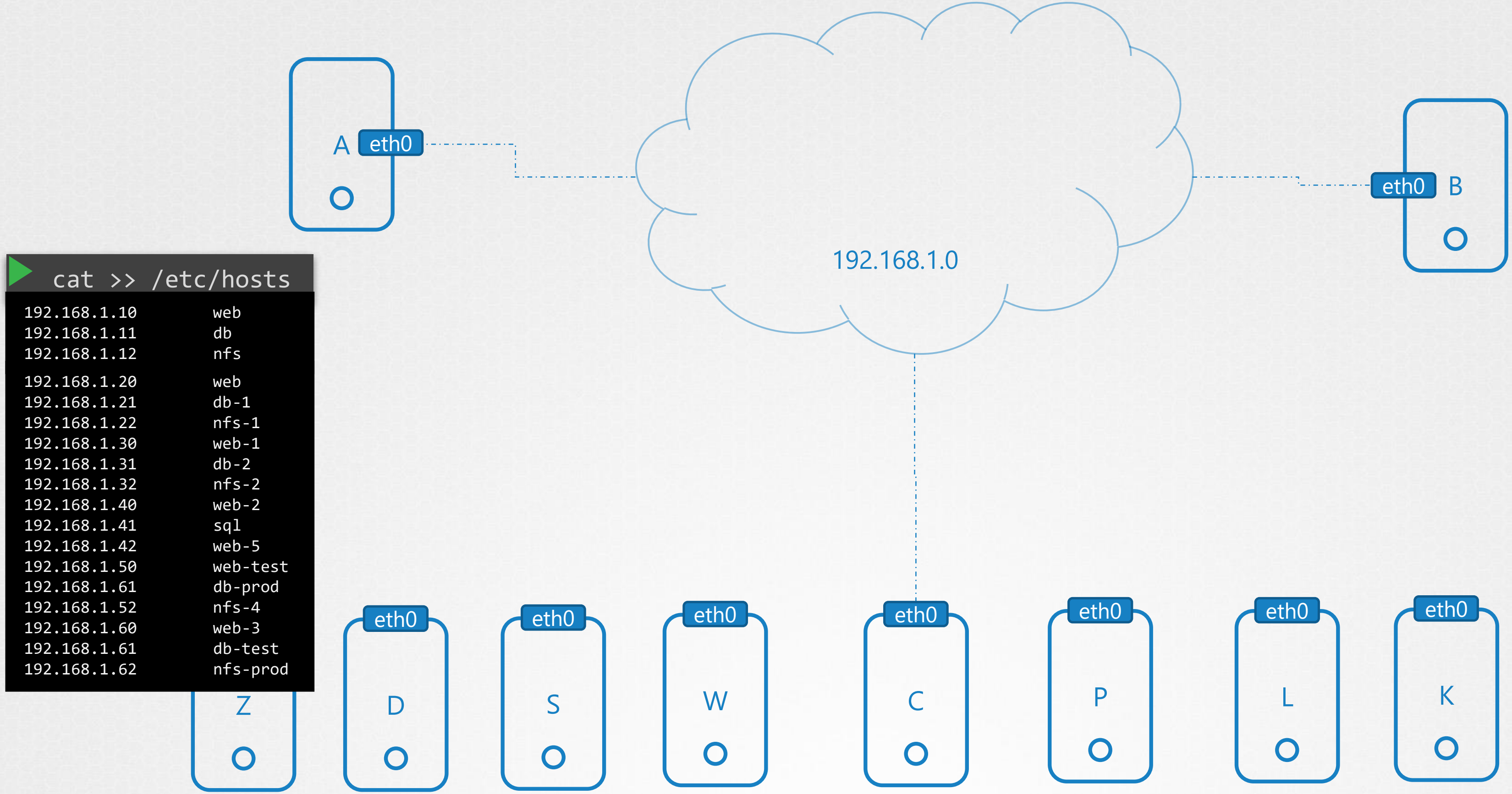


```
▶ cat >> /etc/hosts
192.168.1.10          web
192.168.1.11          db
192.168.1.12          nfs
```

```
▶ cat >> /etc/hosts
192.168.1.10          web
192.168.1.11          db
192.168.1.12          nfs
```

```
▶ cat >> /etc/hosts
192.168.1.10          web
192.168.1.11          db
192.168.1.12          nfs
```

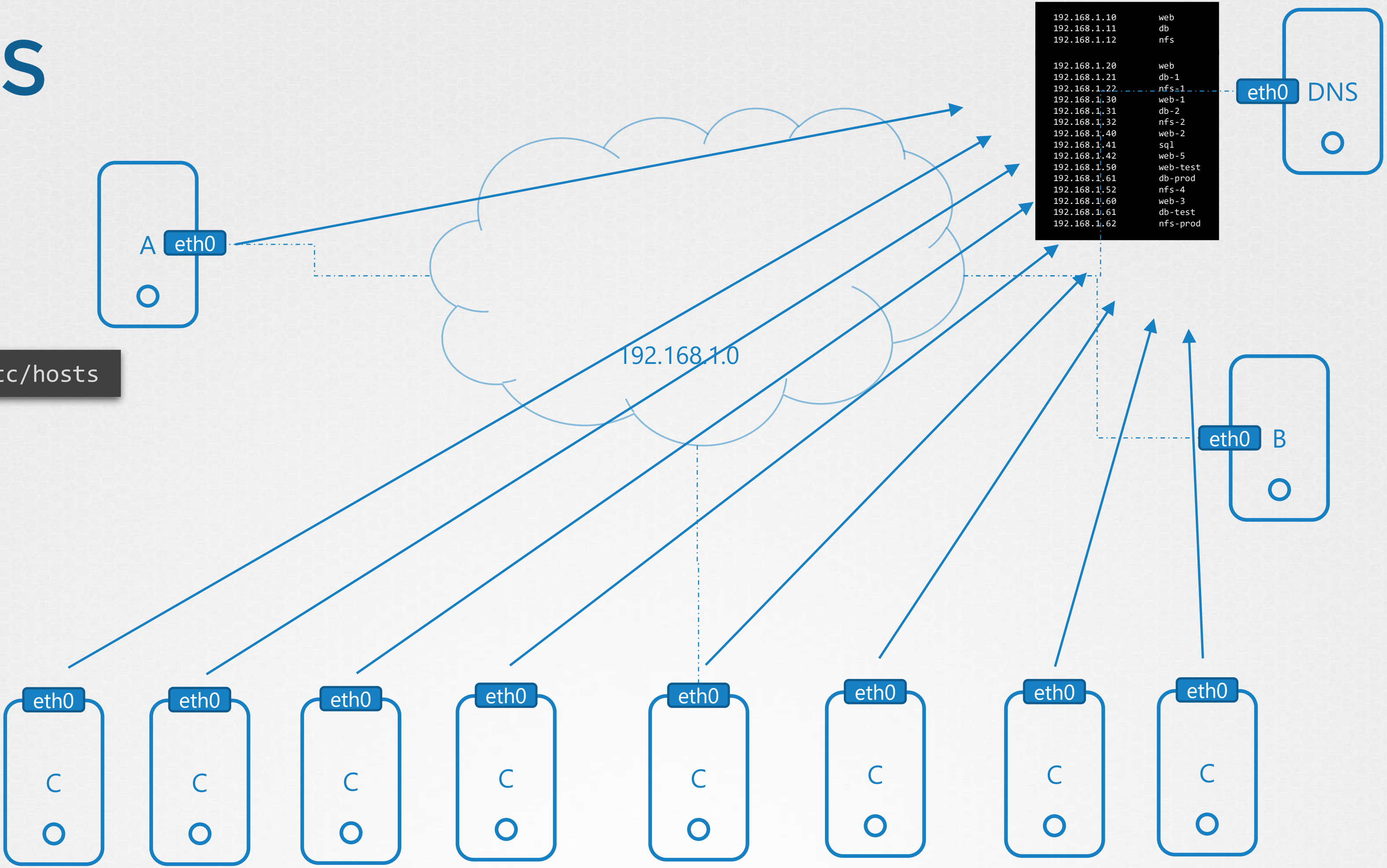
Name Resolution



```
▶ cat >> /etc/hosts
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

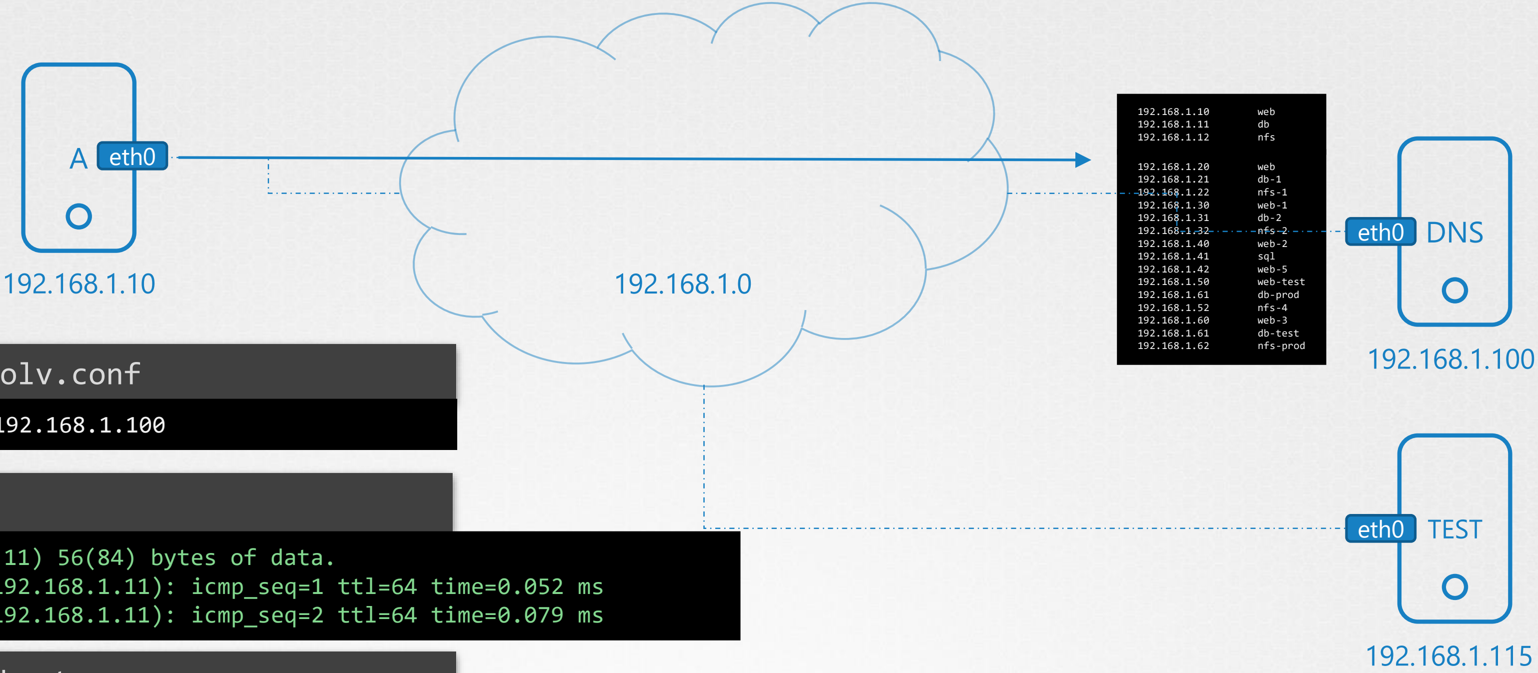
DNS

```
cat >> /etc/hosts
```



```
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

DNS



```
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

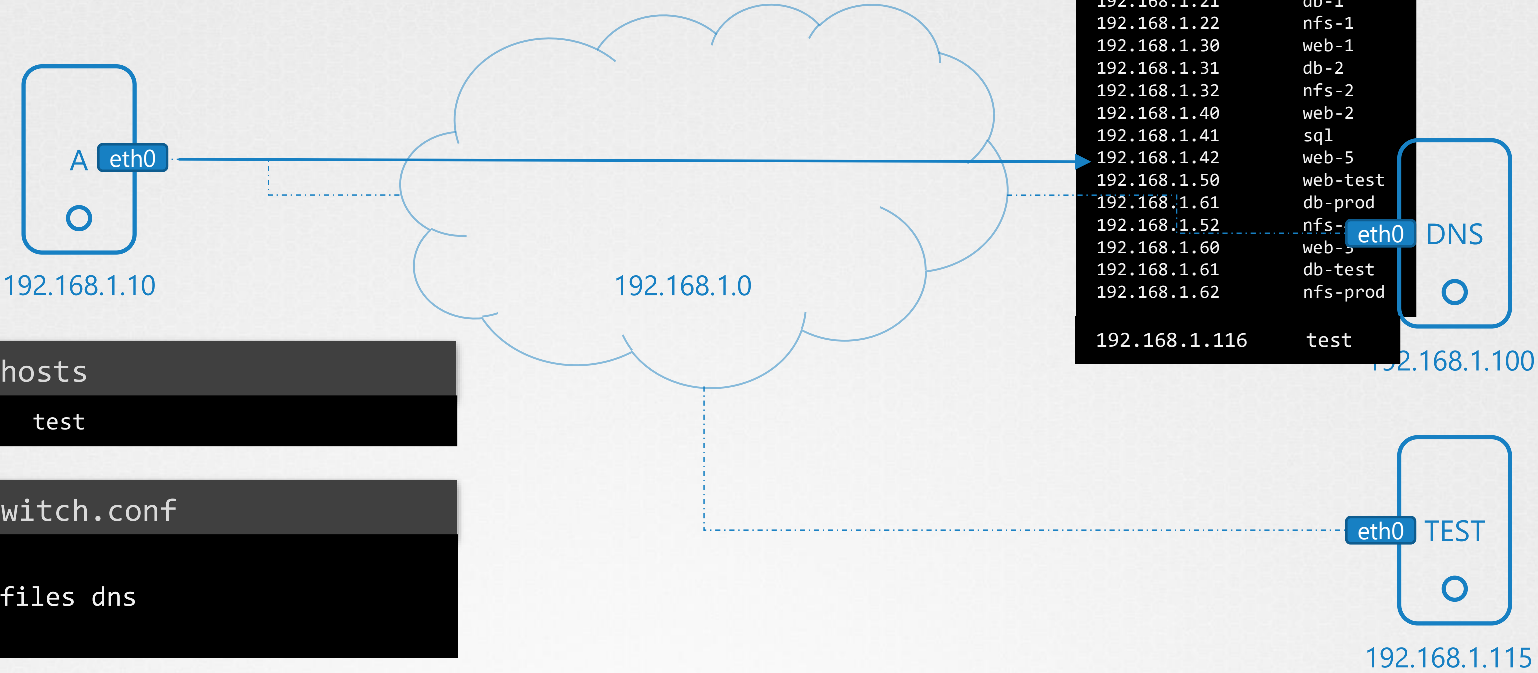
```
▶ cat /etc/resolv.conf
nameserver          192.168.1.100
```

```
▶ ping db
PING db (192.168.1.11) 56(84) bytes of data.
64 bytes from db (192.168.1.11): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from db (192.168.1.11): icmp_seq=2 ttl=64 time=0.079 ms
```

```
▶ cat >> /etc/hosts
192.168.1.115          test
```

```
▶ ping test
PING test (192.168.1.115) 56(84) bytes of data.
64 bytes from test (192.168.1.115): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from test (192.168.1.115): icmp_seq=2 ttl=64 time=0.079 ms
```

DNS

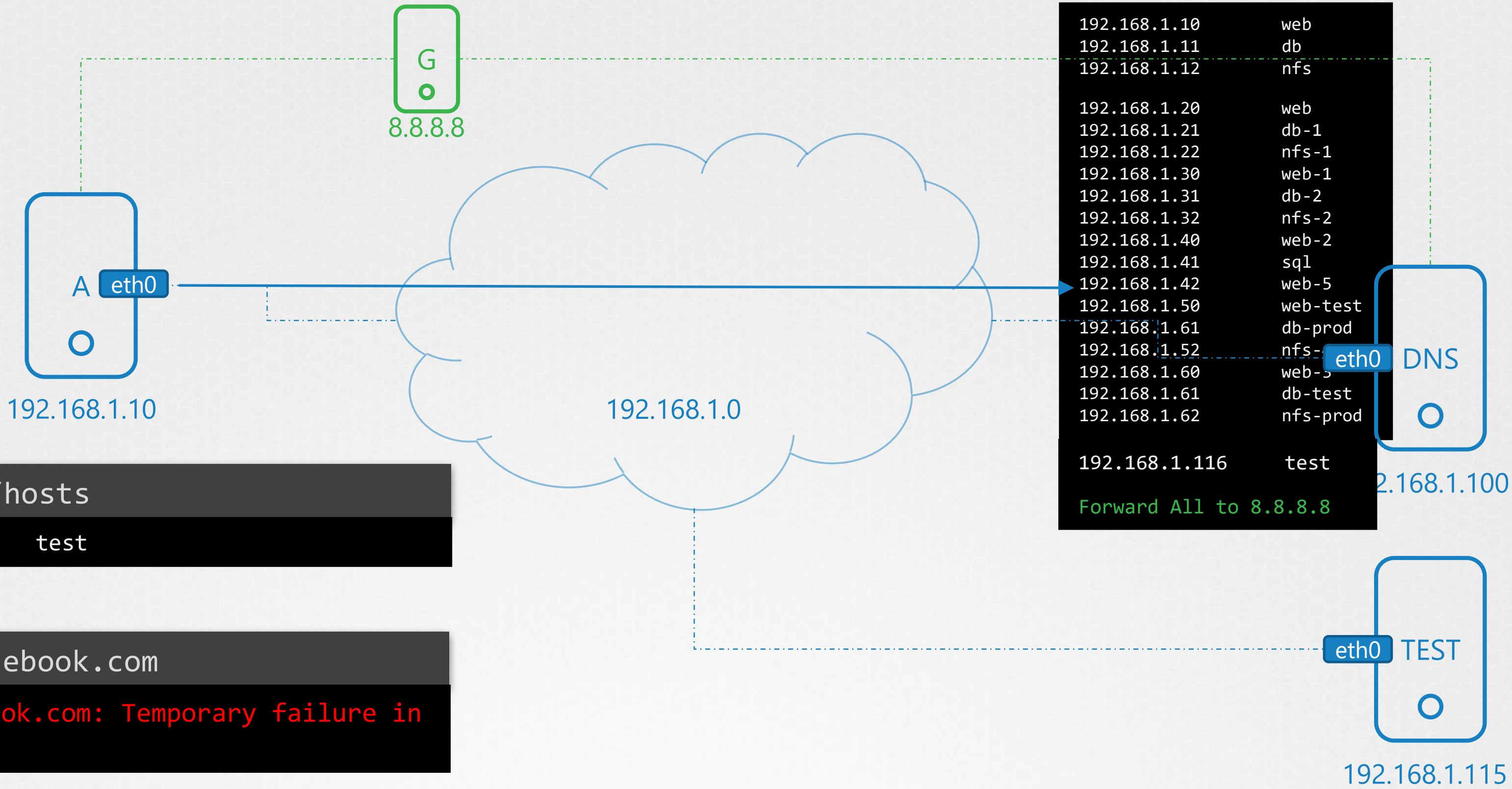


```
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-
192.168.1.60        web-5
192.168.1.61        db-test
192.168.1.62        nfs-prod

192.168.1.116       test
```

```
cat >> /etc/hosts
192.168.1.115          test
```

```
cat /etc/nsswitch.conf
…
hosts:            files dns
…
```

DNS



```
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-
192.168.1.60        web-
192.168.1.61        db-test
192.168.1.62        nfs-prod

192.168.1.116       test

Forward All to 8.8.8.8
```

```
▶ cat >> /etc/hosts
192.168.1.115          test
```

```
▶ ping www.facebook.com
ping: www.facebook.com: Temporary failure in
name resolution
```

```
▶ cat >> /etc/resolv.conf
nameserver          192.168.1.100
```

```
▶ ping www.facebook.com
PING star-mini.c10r.facebook.com (157.240.13.35) 56(84) bytes of data.
64 bytes from edge-star-mini-shv-02-sin6.facebook.com (157.240.13.35): icmp_seq=1 ttl=50 time=5.70 ms
```

Domain Names

www.codepen.io

wwwkubernetes.io

www.facebook.com

www.un.org

www.mit.edu

www.google.com

www.behance.net

www.speedtest.net

www.stanford.edu

www.care.org

Domain Names

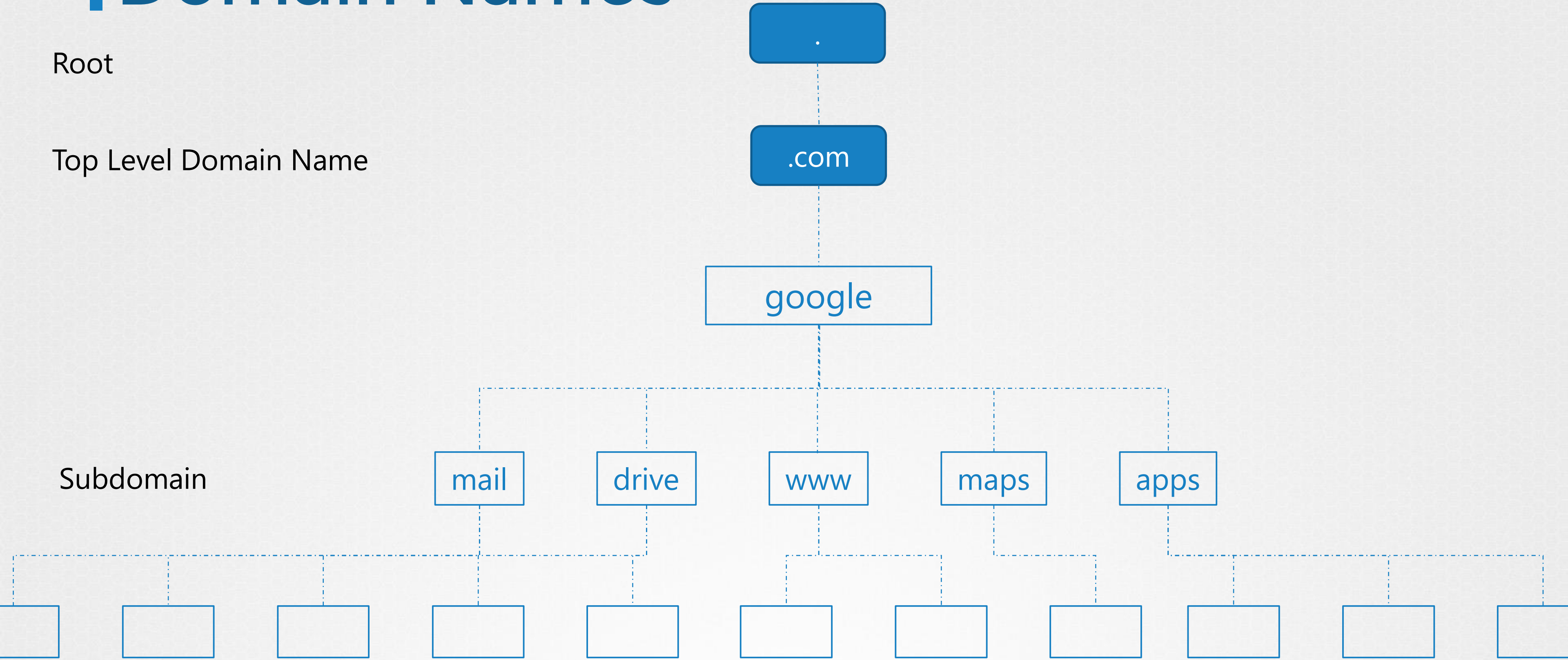
.com	.net	.edu	.org	.io
www.google	www.behance	www.stanford	www.care	www.kubernetes
www.facebook	www.speedtest	www.mit	www.un	www.codepen

Domain Names

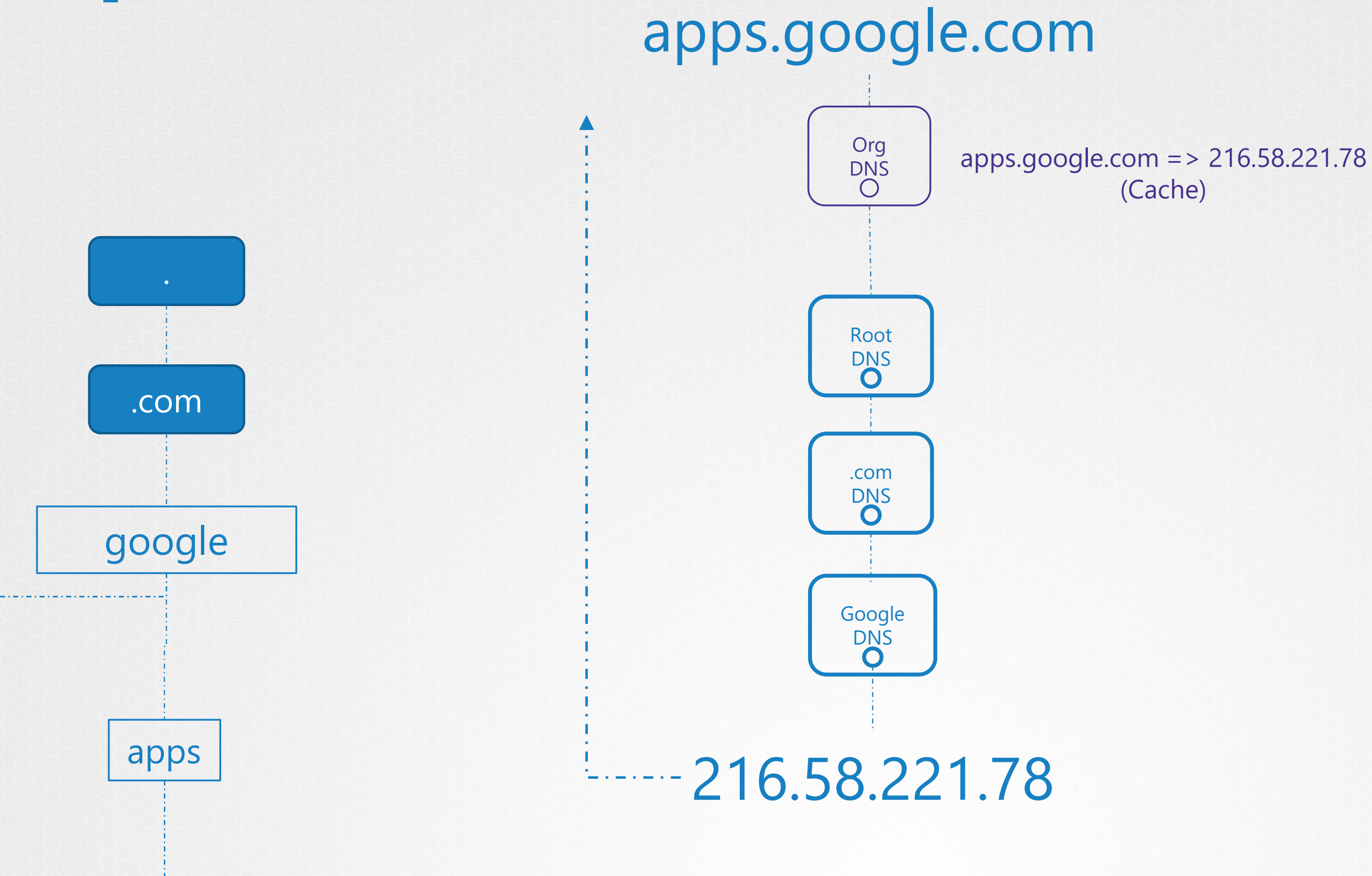
Root

Top Level Domain Name

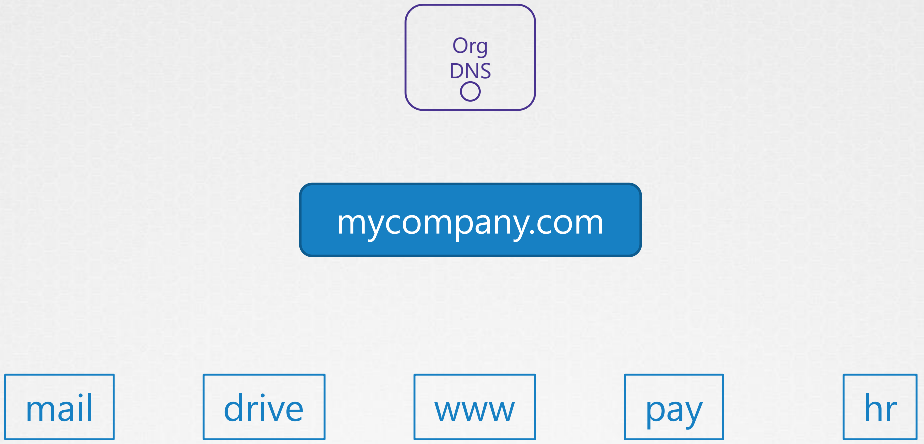
Subdomain



Domain Names



Domain Names



Org DNS

mycompany.com

mail

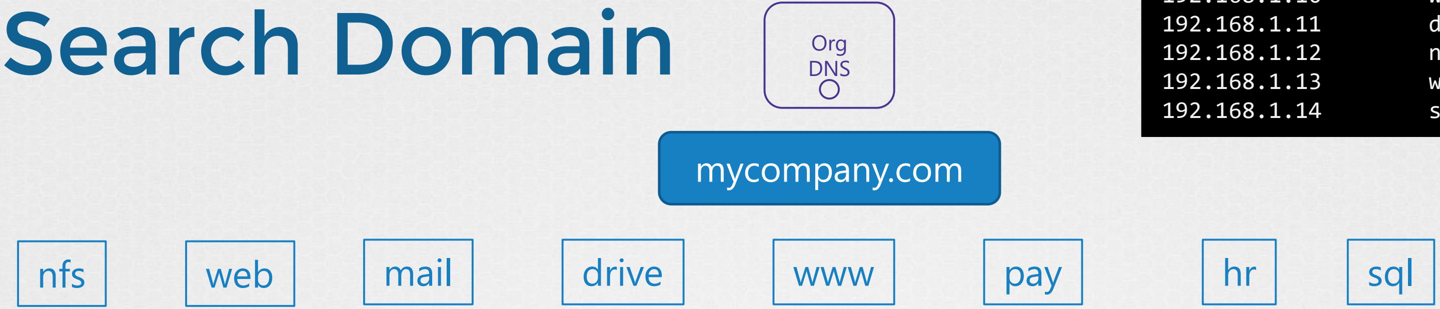
drive

www

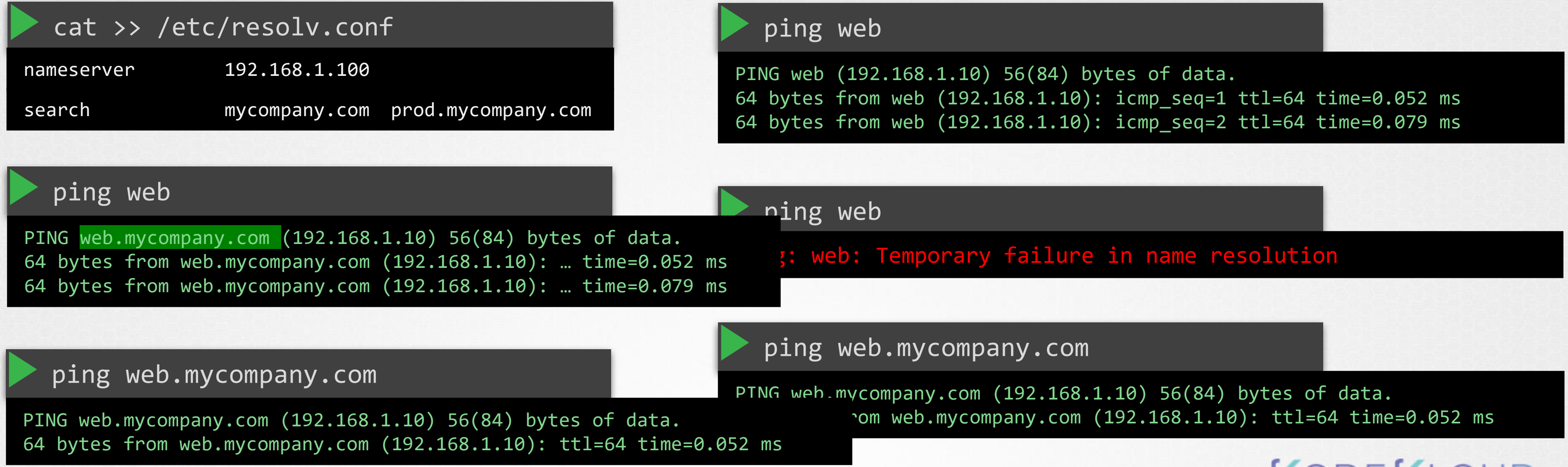
pay

hr

Search Domain



```
192.168.1.10        web.mycompany.com
192.168.1.11        db.mycompany.com
192.168.1.12        nfs.mycompany.com
192.168.1.13        web-1.mycompany.com
192.168.1.14        sql.mycompany.com
```



```
▶ cat >> /etc/resolv.conf
nameserver          192.168.1.100
search              mycompany.com   prod.mycompany.com
```

```
▶ ping web
PING web.mycompany.com (192.168.1.10) 56(84) bytes of data.
64 bytes from web.mycompany.com (192.168.1.10): … time=0.052 ms
64 bytes from web.mycompany.com (192.168.1.10): … time=0.079 ms
```

```
▶ ping web.mycompany.com
PING web.mycompany.com (192.168.1.10) 56(84) bytes of data.
64 bytes from web.mycompany.com (192.168.1.10): ttl=64 time=0.052 ms
```

```
▶ ping web
PING web (192.168.1.10) 56(84) bytes of data.
64 bytes from web (192.168.1.10): icmp_seq=1 ttl=64 time=0.052 ms
64 bytes from web (192.168.1.10): icmp_seq=2 ttl=64 time=0.079 ms
```

```
▶ ping web
g: web: Temporary failure in name resolution
```

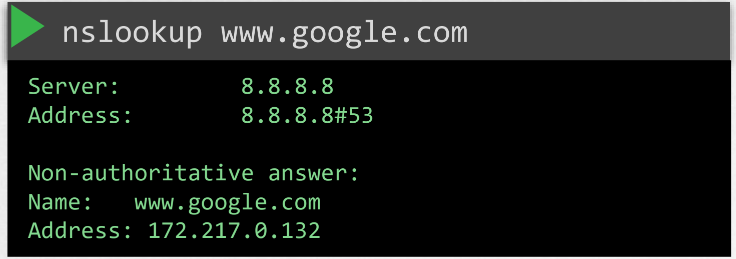
```
▶ ping web.mycompany.com
PING web.mycompany.com (192.168.1.10) 56(84) bytes of data.
rom web.mycompany.com (192.168.1.10): ttl=64 time=0.052 ms
```

KODEKLOUD

Record Types

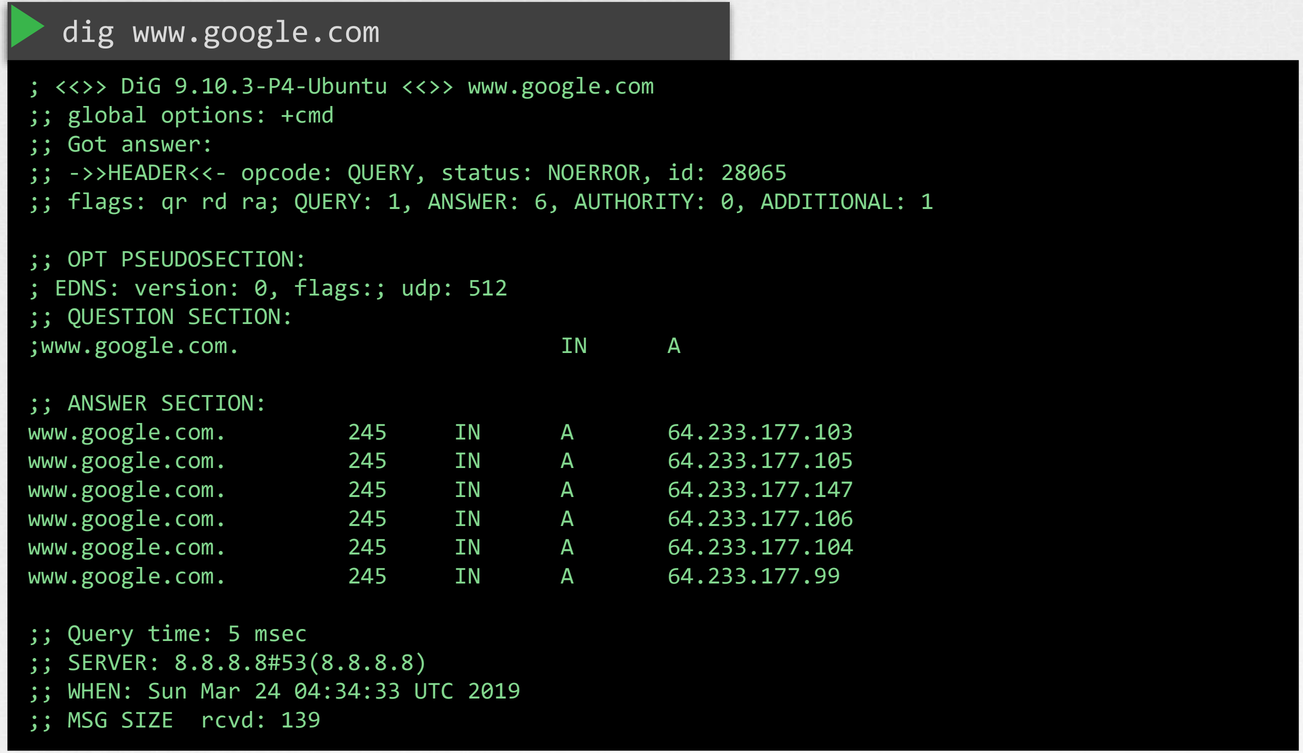
A	web-server	192.168.1.1
AAAA	web-server	2001:0db8:85a3:0000:0000:8a2e:0370:7334
CNAME	food.web-server	eat.web-server, hungry.web-server

nslookup



```
▶ nslookup www.google.com
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.0.132
```

dig



```
dig www.google.com
```

```
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28065
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         245     IN      A       64.233.177.103
www.google.com.         245     IN      A       64.233.177.105
www.google.com.         245     IN      A       64.233.177.147
www.google.com.         245     IN      A       64.233.177.106
www.google.com.         245     IN      A       64.233.177.104
www.google.com.         245     IN      A       64.233.177.99

;; Query time: 5 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Sun Mar 24 04:34:33 UTC 2019
;; MSG SIZE  rcvd: 139
```

KODEKLOUD

Course Objectives

- Scheduling
- Logging Monitoring
- Application Lifecycle Management
- Cluster Maintenance
- Security
- Storage
- Troubleshooting
- Core Concepts
- Networking

○ Pre-Requisites – Network, Switching, Routing, Tools

○ Pre-Requisites – Network Namespaces

○ Pre-Requisites – DNS, IPAM, Firewalls, LBs

○ Pre-Requisites – Networking in Docker

○ Networking Configuration on Cluster Nodes

○ Service Networking

○ POD Networking Concepts

○ Network Loadbalancer

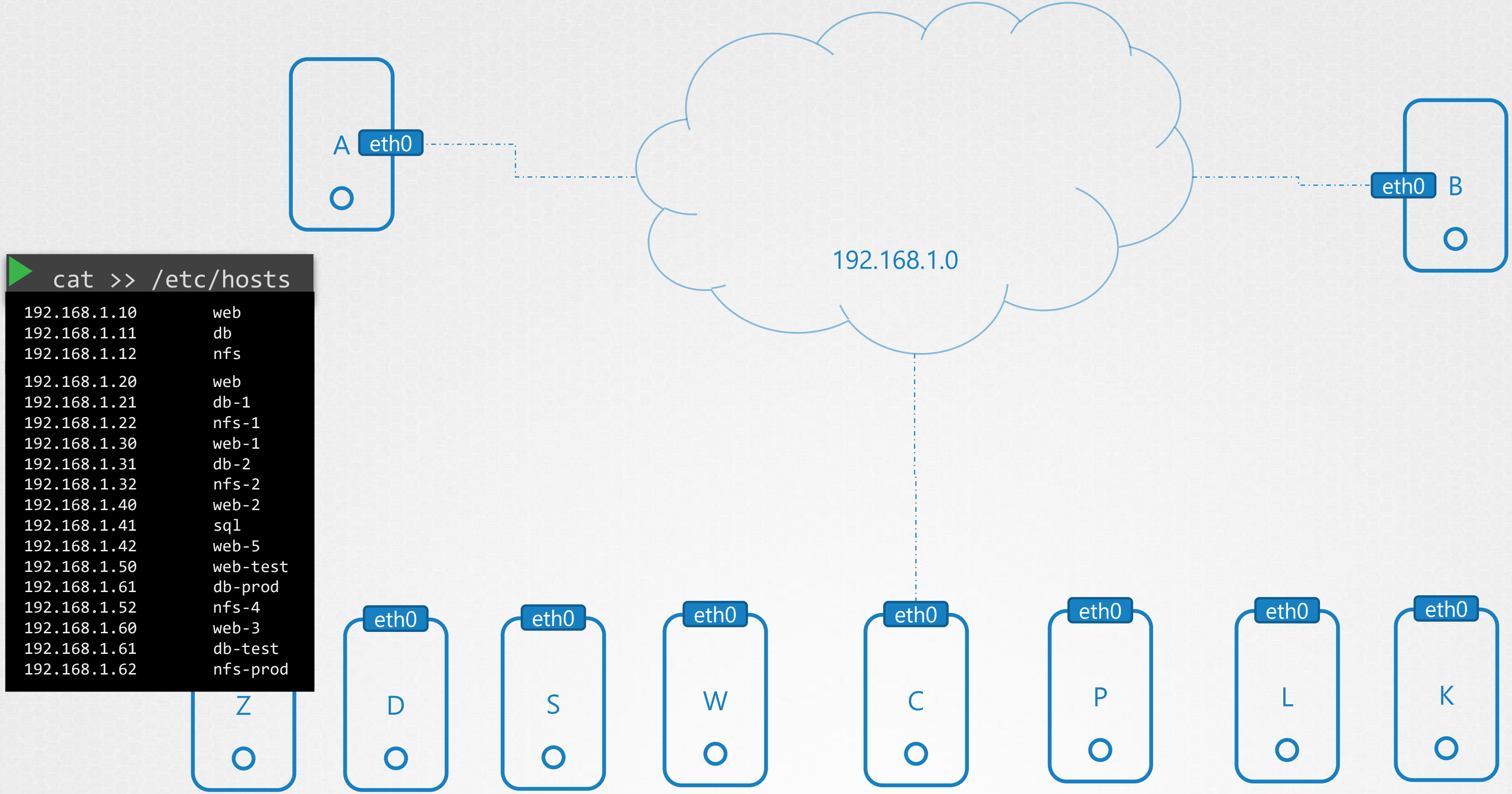
○ Ingress

○ Cluster DNS

○ CNI

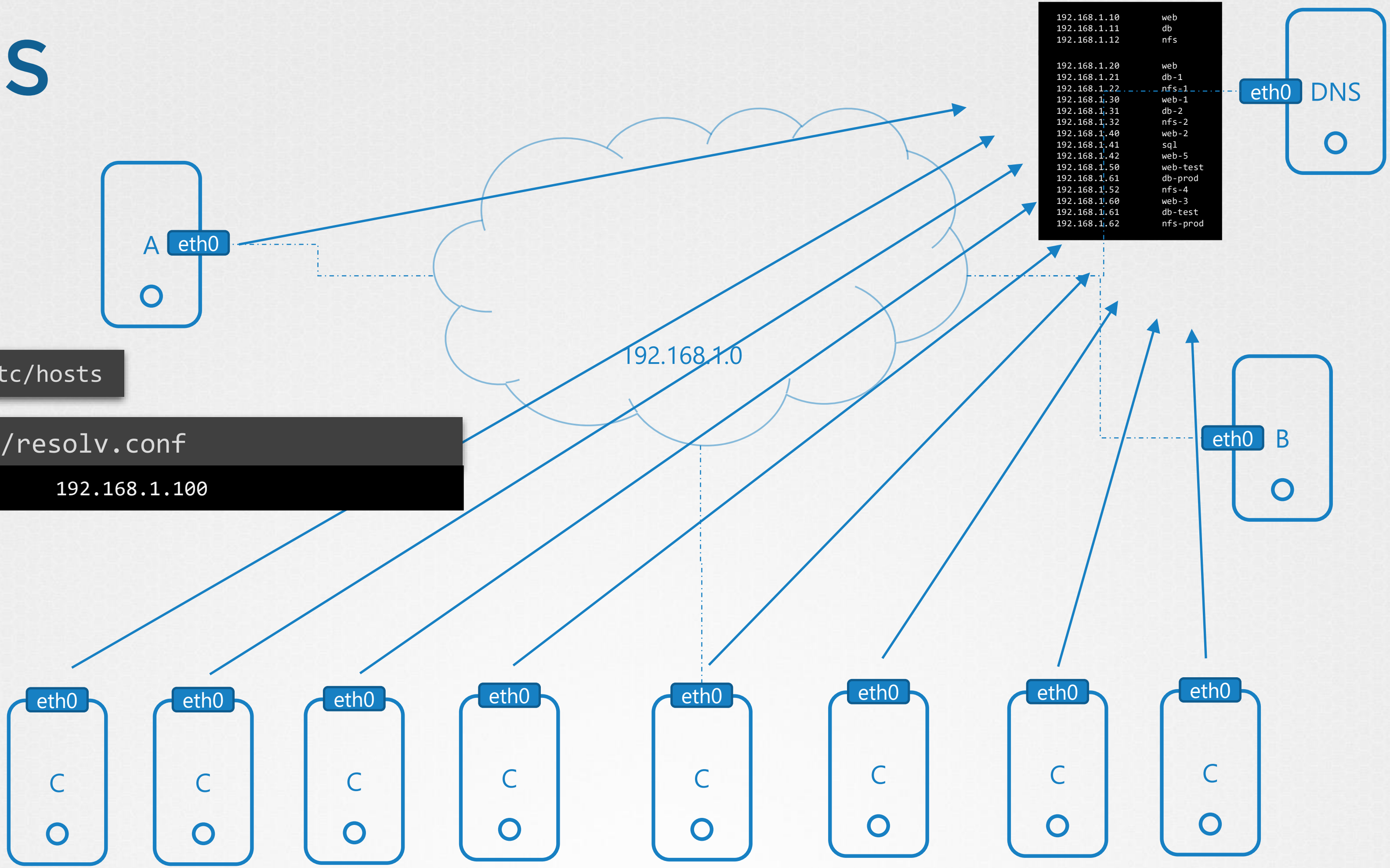
CoreDNS

Name Resolution



```
▶ cat >> /etc/hosts
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

DNS

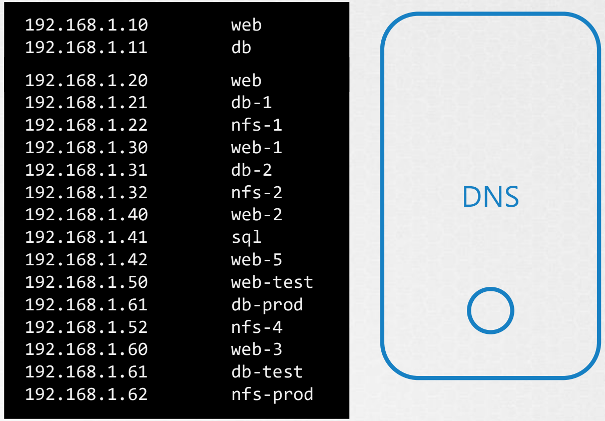


```
192.168.1.10        web
192.168.1.11        db
192.168.1.12        nfs

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

```
▶ cat >> /etc/hosts
```

```
▶ cat /etc/resolv.conf
nameserver        192.168.1.100
```

CoreDNS



```
192.168.1.10        web
192.168.1.11        db

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

CoreDNS

```
wget https://github.com/coredns/coredns/releases/download/v1.4.0/coredns_1.4.0_linux_amd64.tgz
```

```
coredns_1.4.0_linux_amd64.tgz
```

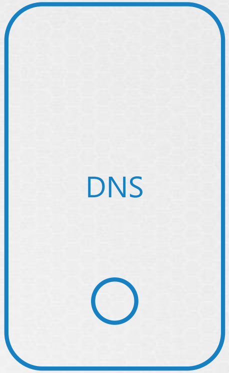
```
tar –xzvf coredns_1.4.0_linux_amd64.tgz
```

```
coredns
```

```
./coredns
```

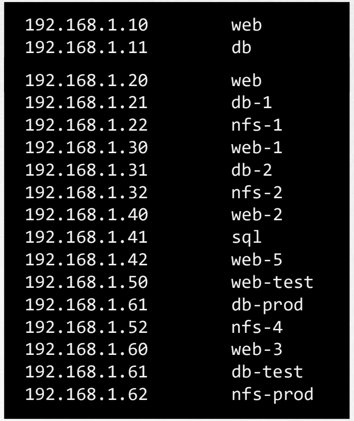
```
.:53
2019-03-04T10:46:13.756Z [INFO] CoreDNS-1.4.0
2019-03-04T10:46:13.756Z [INFO] linux/amd64, go1.12,
8dcc7fc
CoreDNS-1.4.0
linux/amd64, go1.12, 8dcc7fc
```

```
192.168.1.10        web
192.168.1.11        db

192.168.1.20        web
192.168.1.21        db-1
192.168.1.22        nfs-1
192.168.1.30        web-1
192.168.1.31        db-2
192.168.1.32        nfs-2
192.168.1.40        web-2
192.168.1.41        sql
192.168.1.42        web-5
192.168.1.50        web-test
192.168.1.61        db-prod
192.168.1.52        nfs-4
192.168.1.60        web-3
192.168.1.61        db-test
192.168.1.62        nfs-prod
```

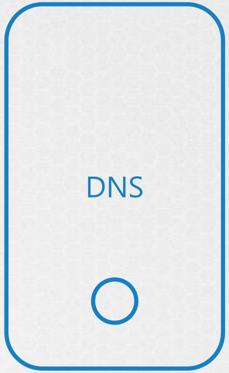


CoreDNS

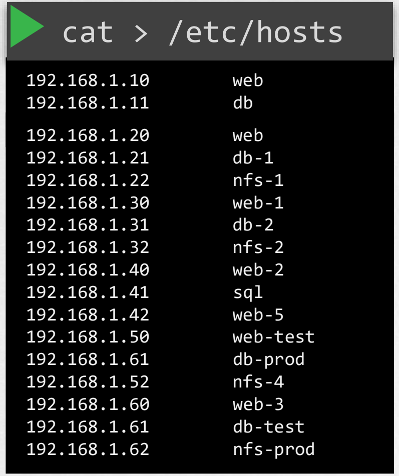
```
cat /etc/hosts
```



```
192.168.1.10          web
192.168.1.11          db

192.168.1.20          web
192.168.1.21          db-1
192.168.1.22          nfs-1
192.168.1.30          web-1
192.168.1.31          db-2
192.168.1.32          nfs-2
192.168.1.40          web-2
192.168.1.41          sql
192.168.1.42          web-5
192.168.1.50          web-test
192.168.1.61          db-prod
192.168.1.52          nfs-4
192.168.1.60          web-3
192.168.1.61          db-test
192.168.1.62          nfs-prod
```

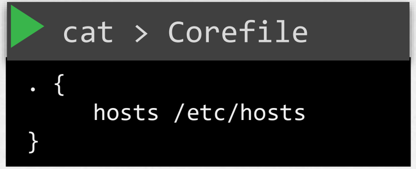


CoreDNS



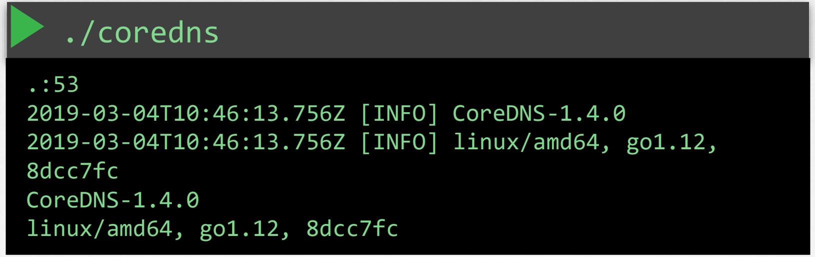
```
cat > /etc/hosts
```

```
192.168.1.10          web
192.168.1.11          db

192.168.1.20          web
192.168.1.21          db-1
192.168.1.22          nfs-1
192.168.1.30          web-1
192.168.1.31          db-2
192.168.1.32          nfs-2
192.168.1.40          web-2
192.168.1.41          sql
192.168.1.42          web-5
192.168.1.50          web-test
192.168.1.61          db-prod
192.168.1.52          nfs-4
192.168.1.60          web-3
192.168.1.61          db-test
192.168.1.62          nfs-prod
```



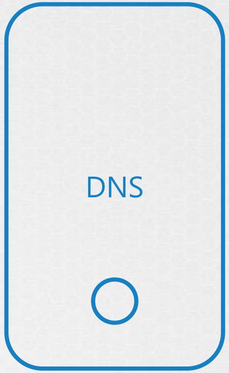
```
cat > Corefile
```

```
. {
      hosts /etc/hosts
}
```



```
./coredns
```

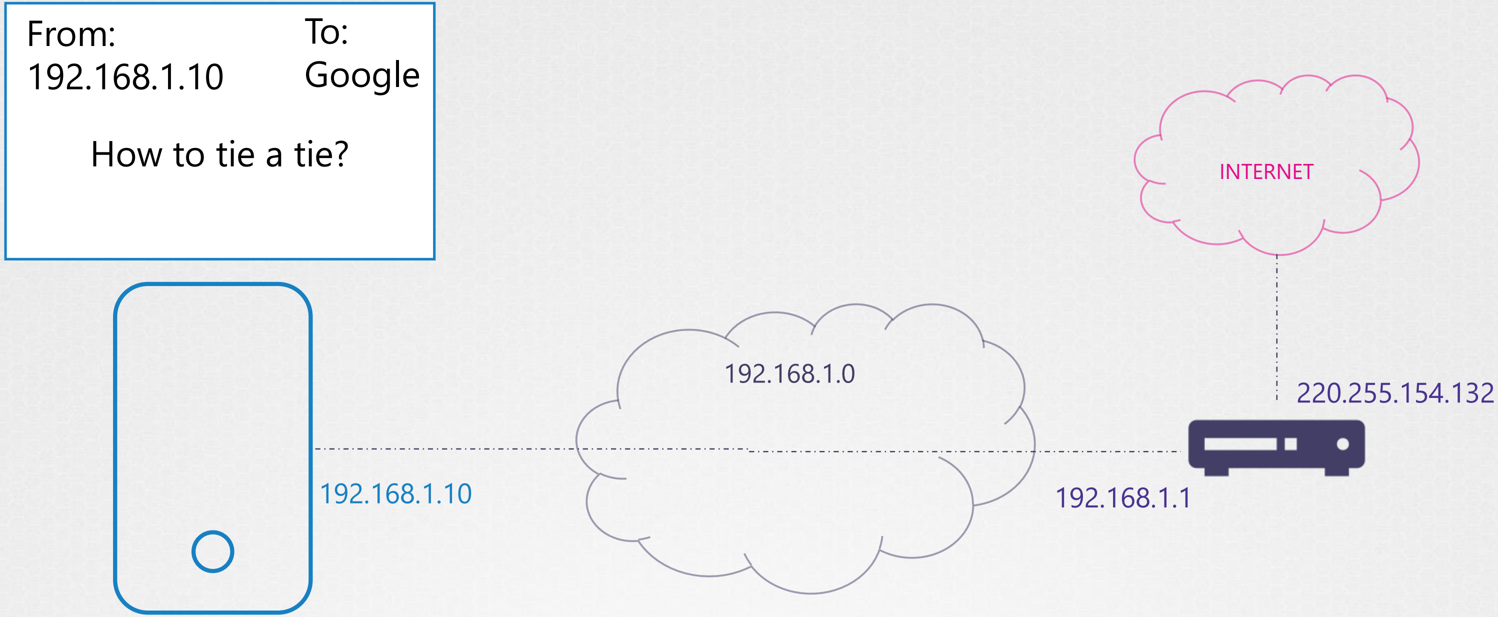
```
.:53
2019-03-04T10:46:13.756Z [INFO] CoreDNS-1.4.0
2019-03-04T10:46:13.756Z [INFO] linux/amd64, go1.12,
8dcc7fc
CoreDNS-1.4.0
linux/amd64, go1.12, 8dcc7fc
```



KODEKLOUD

PRE-REQUISITE

NETWORK ADDRESS TRANSLATION (NAT)

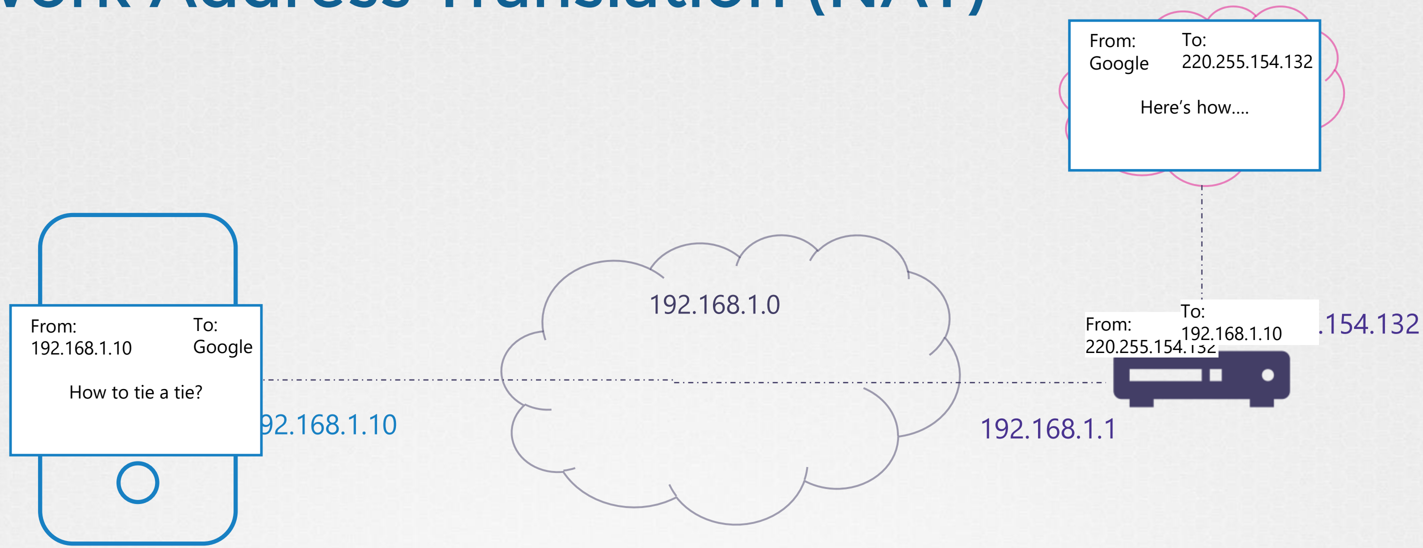


From:
192.168.1.10

To:
Google

How to tie a tie?

Network Address Translation (NAT)



NAT TABLE

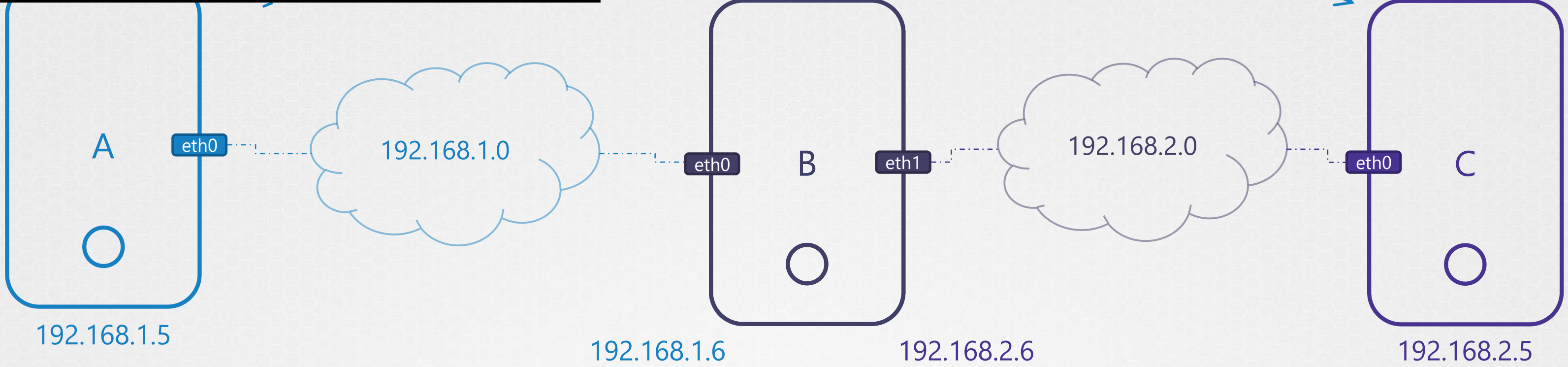
Private IP	Public IP
192.168.1.10	220.255.154.132

```
ping 192.168.2.5
```

```
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
```

```
tcpdump -l
```

```
05:52:50.901754 IP 192.168.1.5 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.1.5 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.1.5 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.1.5 > 192.168.2.5: ICMP echo requh 64
```



```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
1
```

```
ip route add 192.168.2.0/24 via 192.168.1.6
```

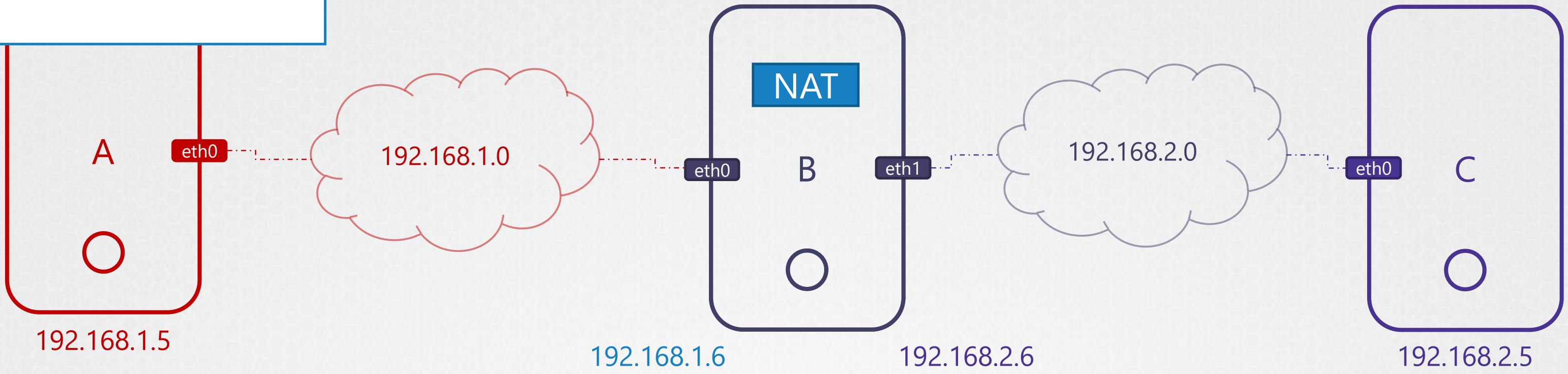
```
ip route add 192.168.1.0/24 via 192.168.2.6
```

From: 192.168.1.5

To: 192.168.2.5

How to tie a tie?

From: 192.168.2.6



```
iptables -t nat -A POSTROUTING -s 192.168.5.0/24 -j MASQUERADE
```

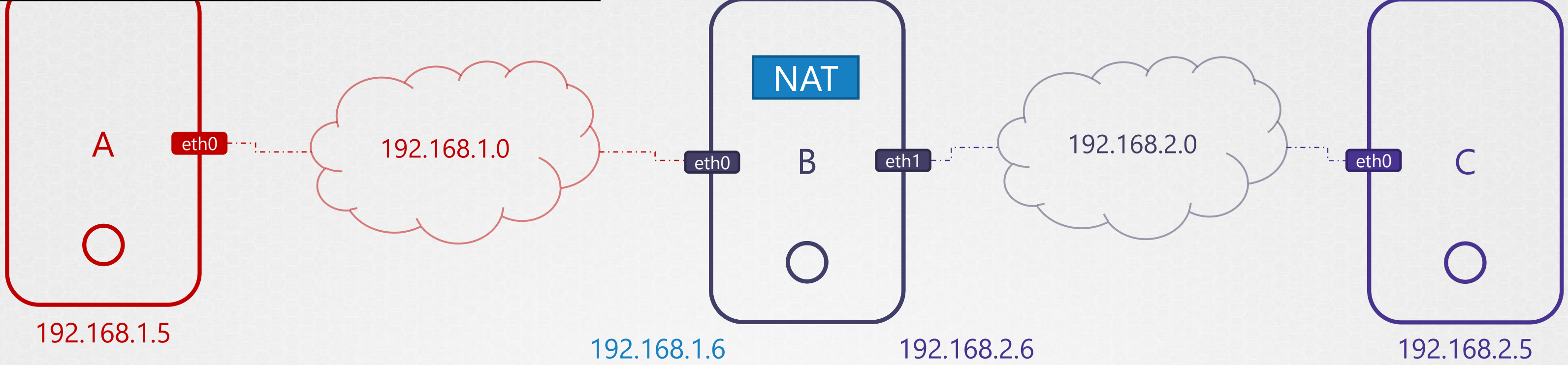
```
echo 1 > /proc/sys/net/ipv4/ip_forward
1
```

```
ip route add 192.168.2.0/24 via 192.168.1.6
```

```
ip route add 192.168.1.0/24 via 192.168.2.6
```

```
ping 192.168.2.5
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
Reply from 192.168.2.5: bytes=32 time=4ms TTL=117
```

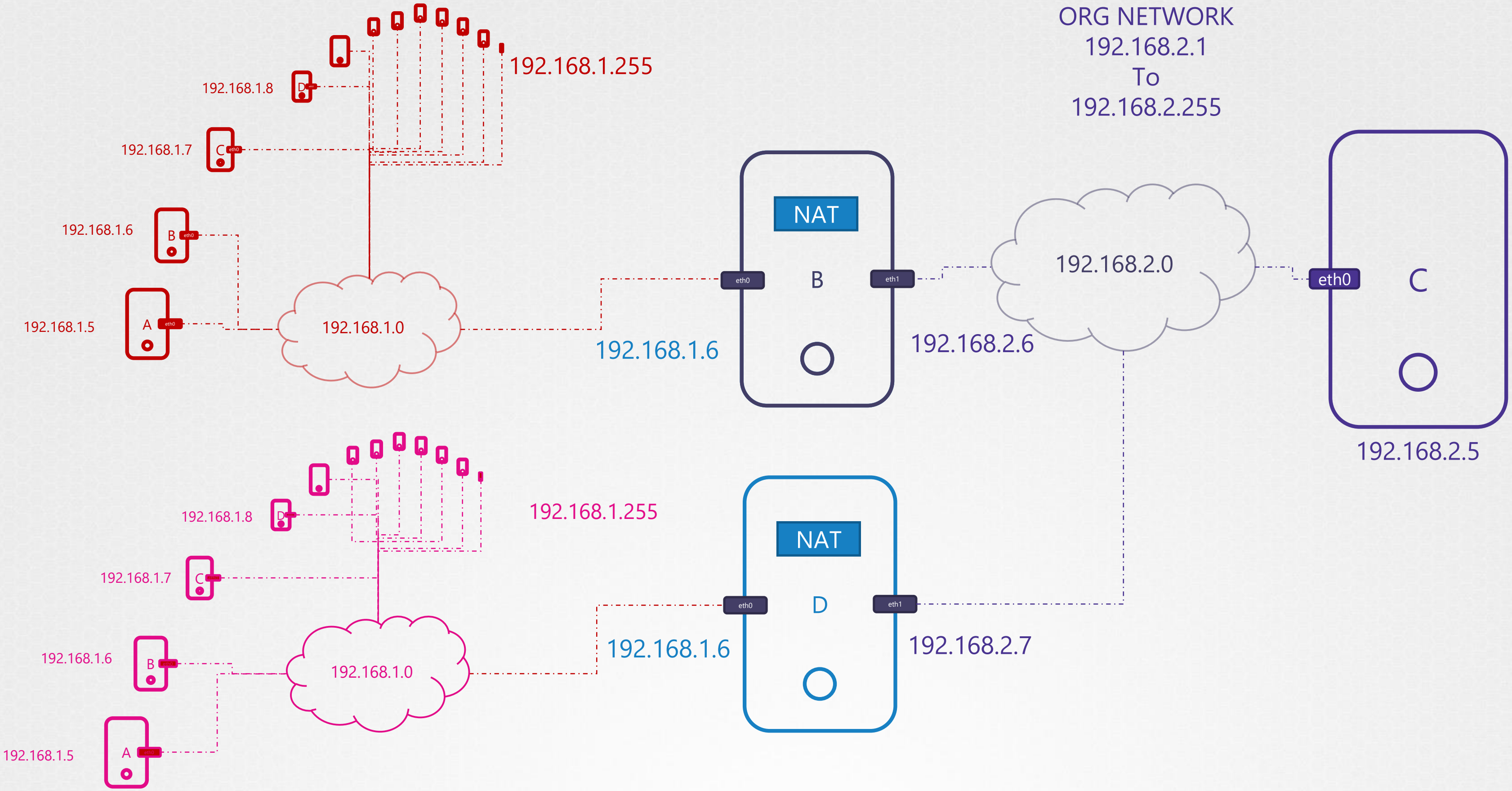
```
tcpdump -l
05:52:50.901754 IP 192.168.2.6 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.2.6 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.2.6 > 192.168.2.5: ICMP echo requh 64
05:52:50.901754 IP 192.168.2.6 > 192.168.2.5: ICMP echo requh 64
```



```
iptables -t nat -A POSTROUTING -s 192.168.5.0/24 -j MASQUERADE
```

```
echo 1 > /proc/sys/net/ipv4/ip_forward
1
```

```
ip route add 192.168.2.0/24 via 192.168.1.6
```



192.168.1.255
192.168.1.8
D
192.168.1.7
C
eth0
192.168.1.6
B
eth0
192.168.1.5
A
eth0
192.168.1.0
192.168.1.6
NAT
eth0
B
eth1
ORG NETWORK
192.168.2.1
To
192.168.2.255
192.168.2.0
192.168.2.6
eth0
C
192.168.2.5
192.168.1.8
D
192.168.1.255
192.168.1.7
C
192.168.1.6
B
192.168.1.5
A
192.168.1.0
192.168.1.6
NAT
eth0
D
eth1
192.168.2.7

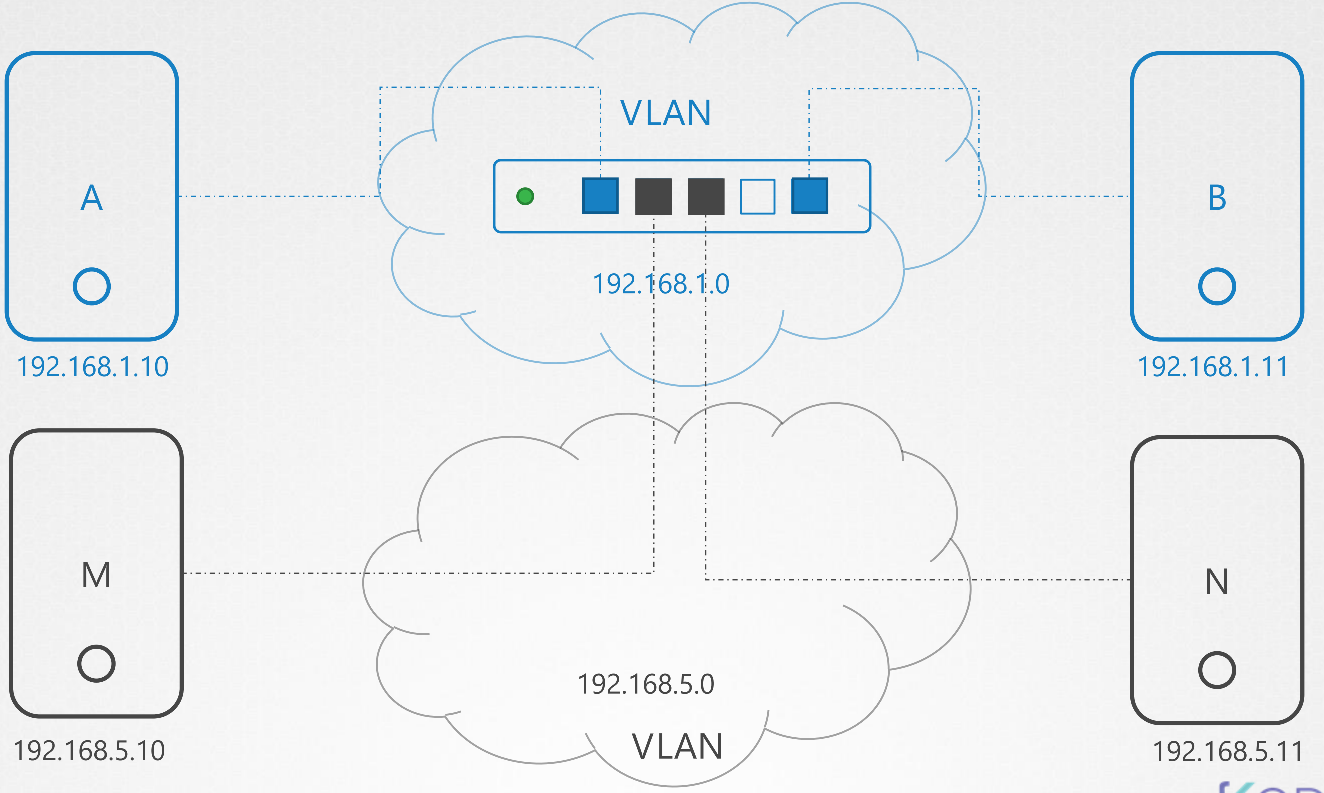
KODEKLOUD

KODEKLOUD

PRE-REQUISITE

NETWORKING VLAN & VXLAN

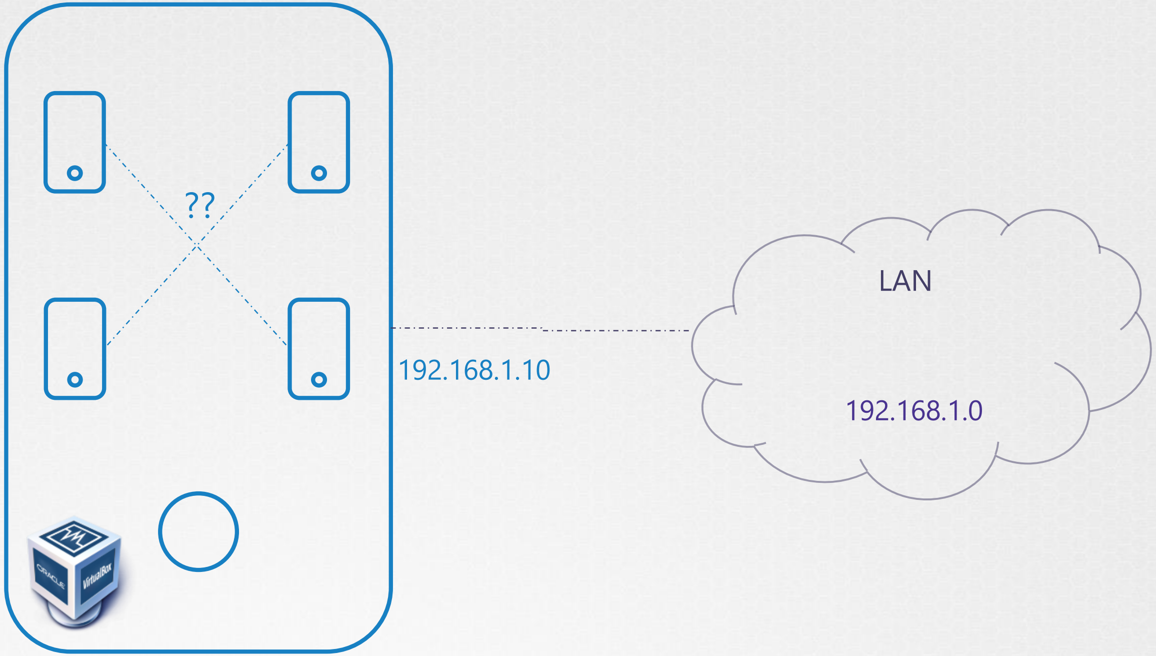
VLAN


VLAN
A
B
192.168.1.0
192.168.1.10
192.168.1.11
M
N
192.168.5.0
VLAN
192.168.5.10
192.168.5.11

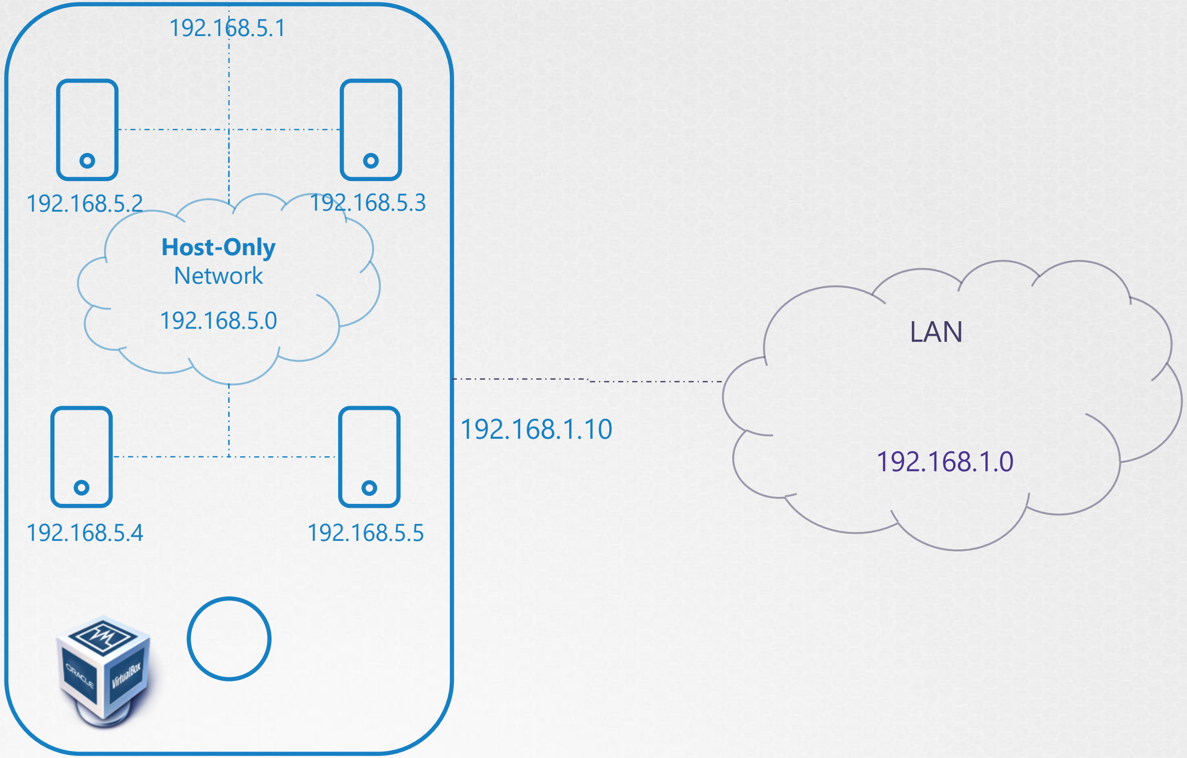
KODEKLOUD

PRE-REQUISITE

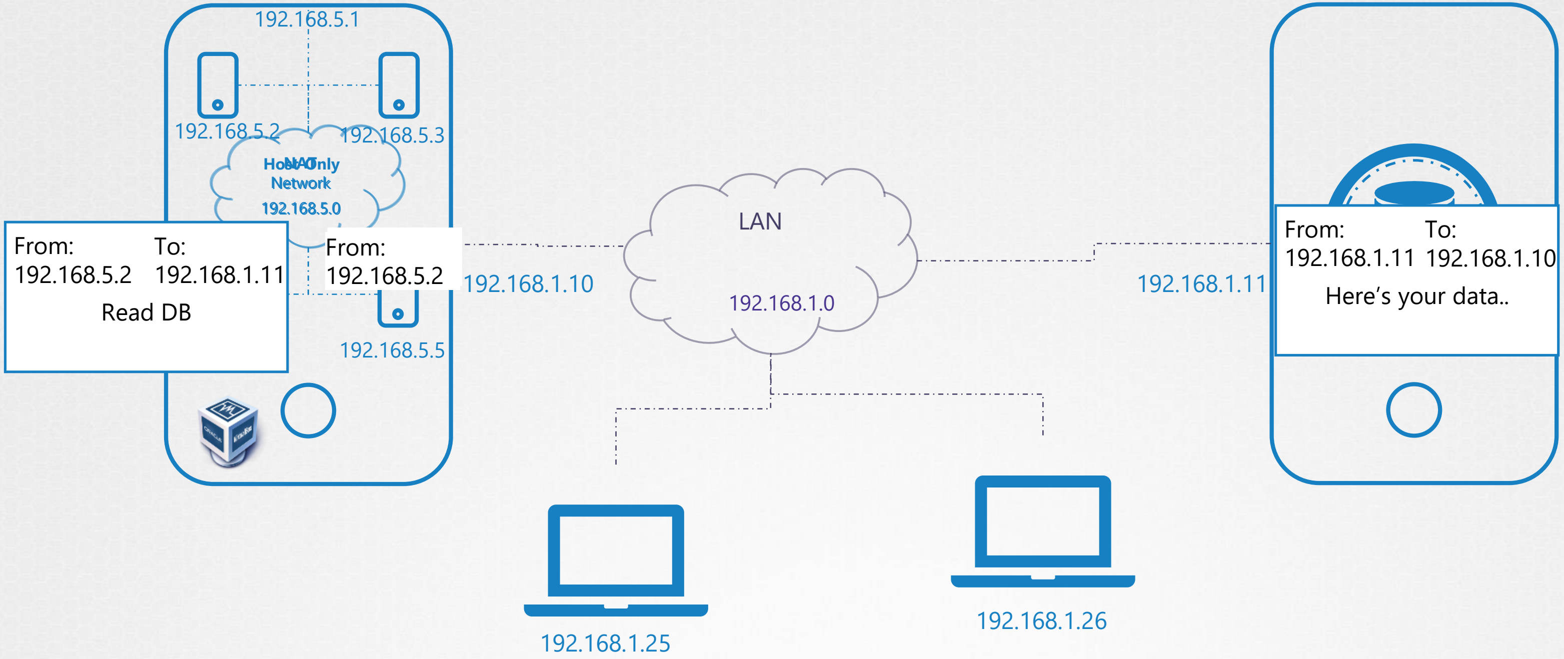
NETWORKING VIRTUAL MACHINES


??
192.168.1.10
LAN
192.168.1.0
ORACLE
VirtualBox

Host Only


192.168.5.1
192.168.5.2
192.168.5.3
Host-Only
Network
192.168.5.0
192.168.5.4
192.168.5.5
192.168.1.10
LAN
192.168.1.0

Network Address Translation (NAT)



From: 192.168.5.2 To: 192.168.1.11

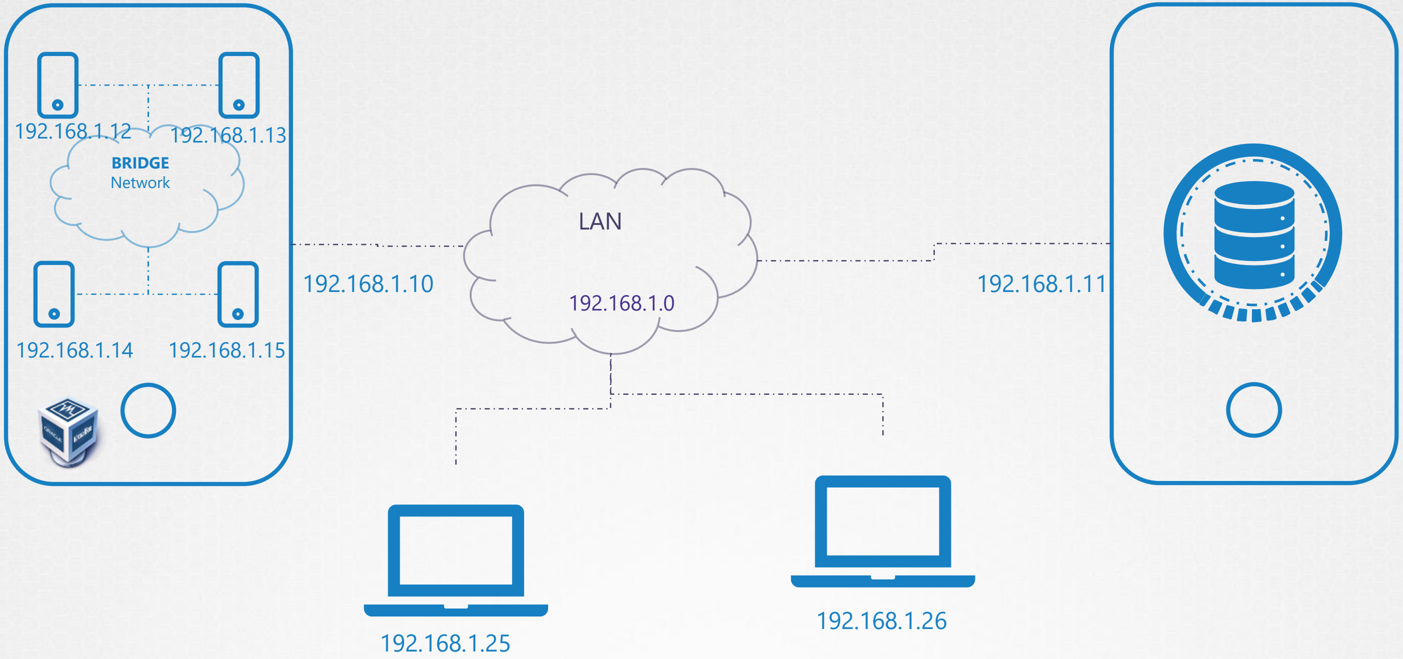
Read DB

From: 192.168.5.2

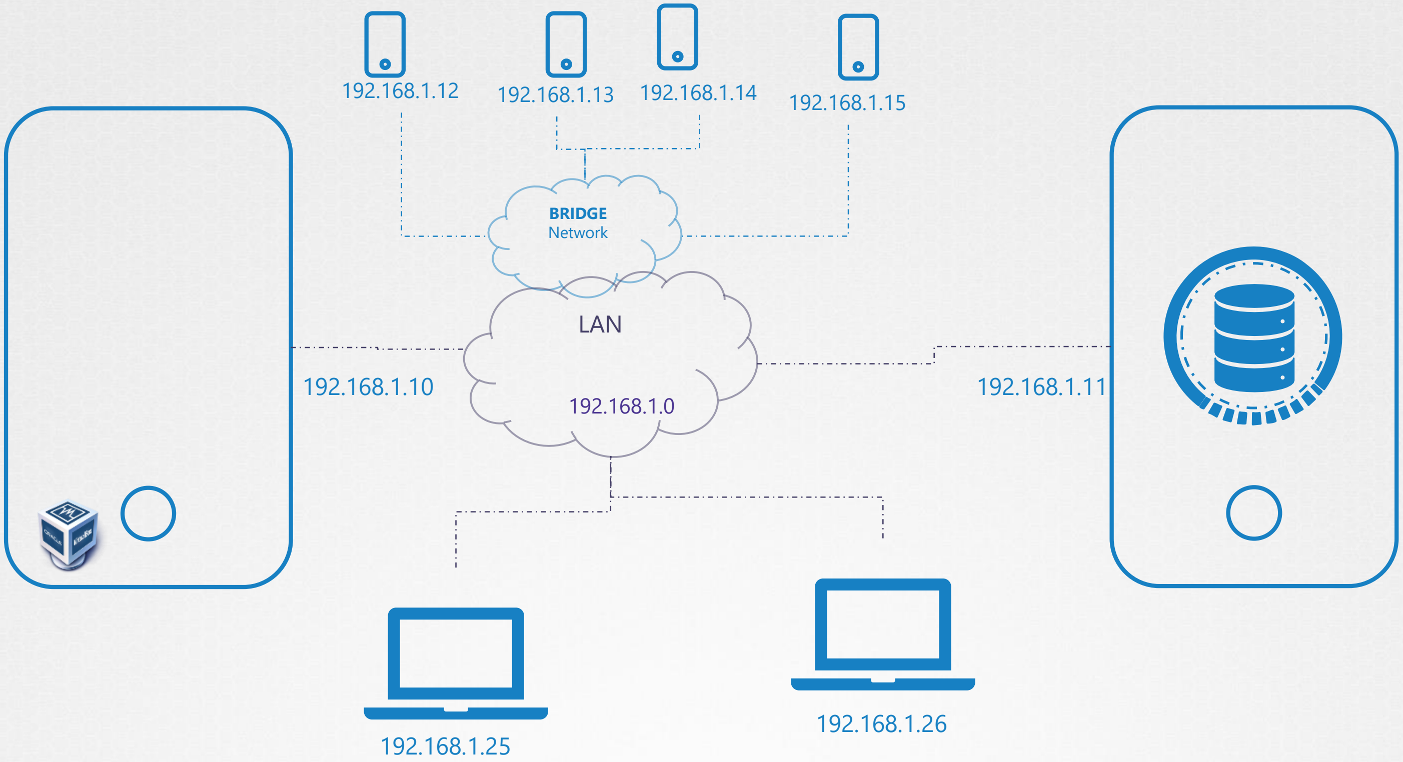
From: 192.168.1.11 To: 192.168.1.10

Here's your data..

Bridge Network



Bridge Network

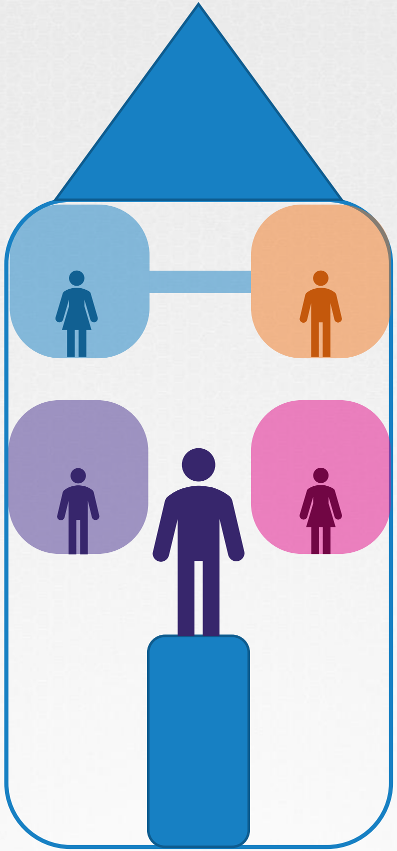


KODEKLOUD

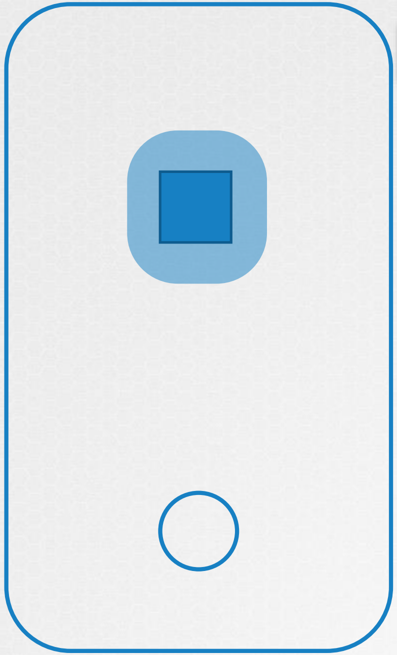
PRE-REQUISITE

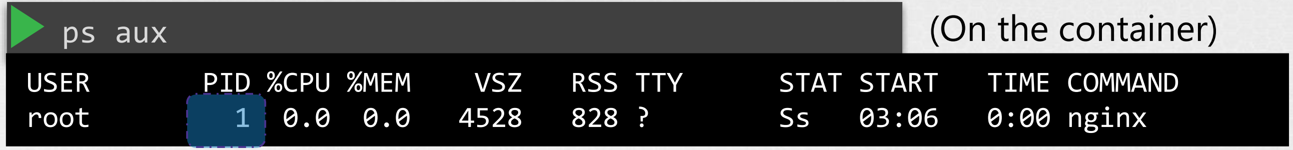
NETWORK NAMESPACES

NAMESPACE



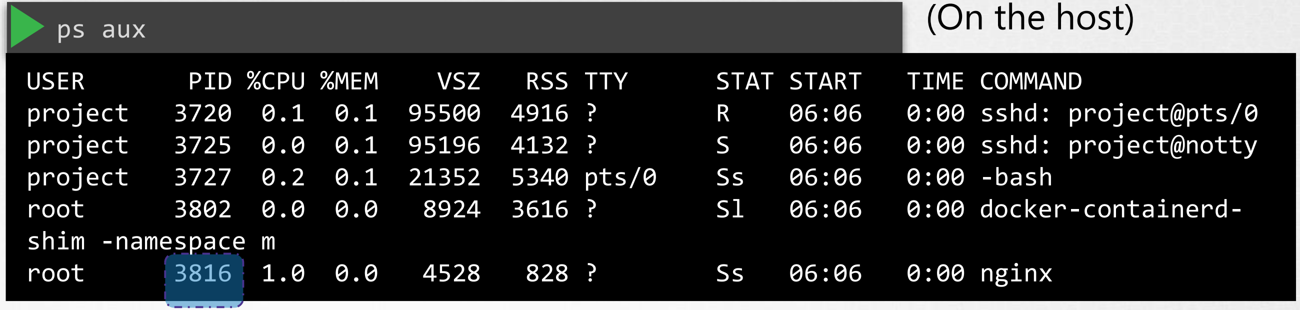
PROCESS NAMESPACE





```
ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0   4528   828 ?        Ss   03:06   0:00 nginx
```

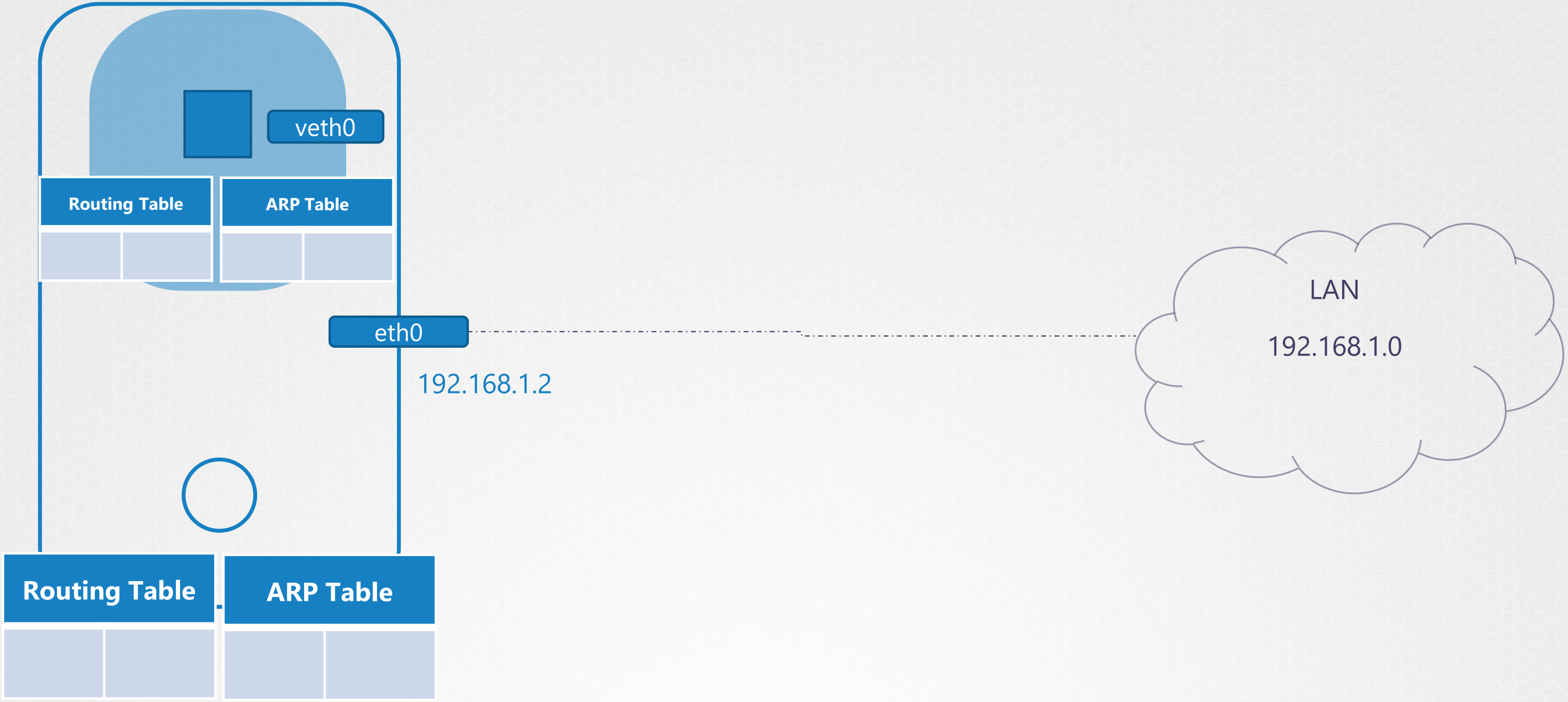
(On the container)



```
ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
project    3720  0.1  0.1  95500  4916 ?        R    06:06   0:00 sshd: project@pts/0
project    3725  0.0  0.1  95196  4132 ?        S    06:06   0:00 sshd: project@notty
project    3727  0.2  0.1  21352  5340 pts/0    Ss   06:06   0:00 -bash
root       3802  0.0  0.0   8924  3616 ?        Sl   06:06   0:00 docker-containerd-
shim -namespace m
root       3816  1.0  0.0   4528   828 ?        Ss   06:06   0:00 nginx
```

(On the host)

NETWORK NAMESPACE

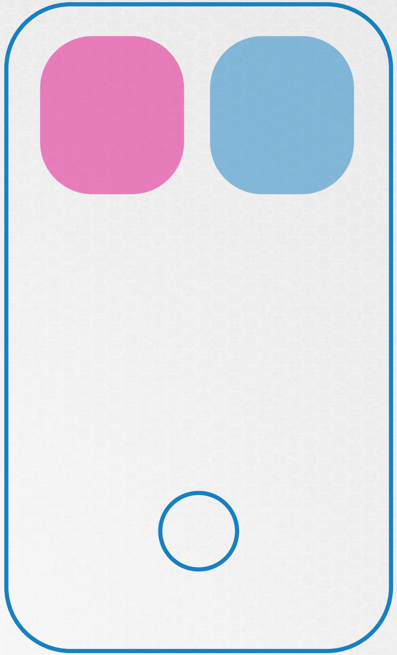


CREATE NETWORK NS

```
▶ ip netns add red
```

```
▶ ip netns add blue
```

```
▶ ip netns
red
blue
```

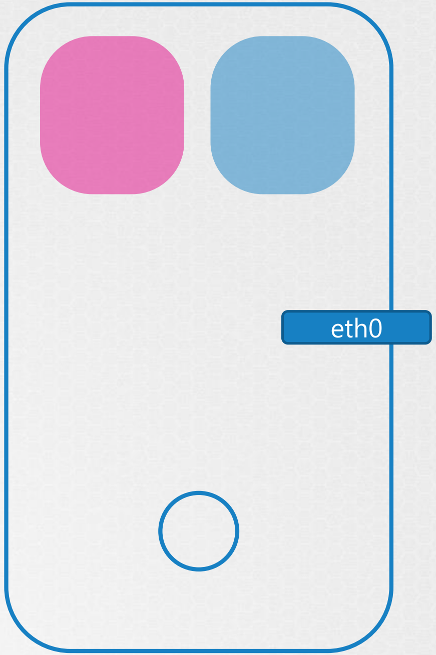


EXEC IN NETWORK NS

```
▶ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc state UP mode DEFAULT qlen 1000
    link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff
```

```
▶ ip netns exec red
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
▶ ip -n red
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

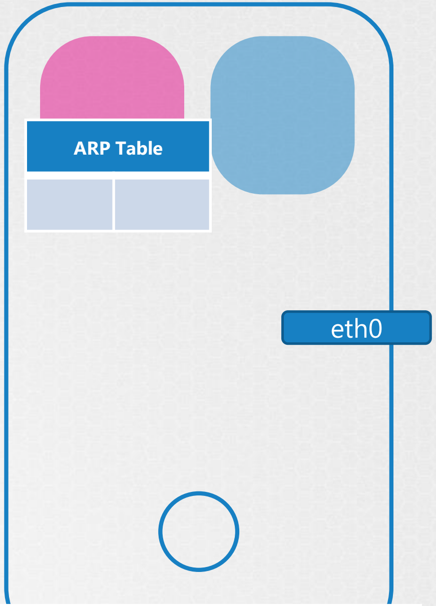


```
ip netns exec red ip link
          =
ip    -n red          link
```

EXEC IN NETWORK NS

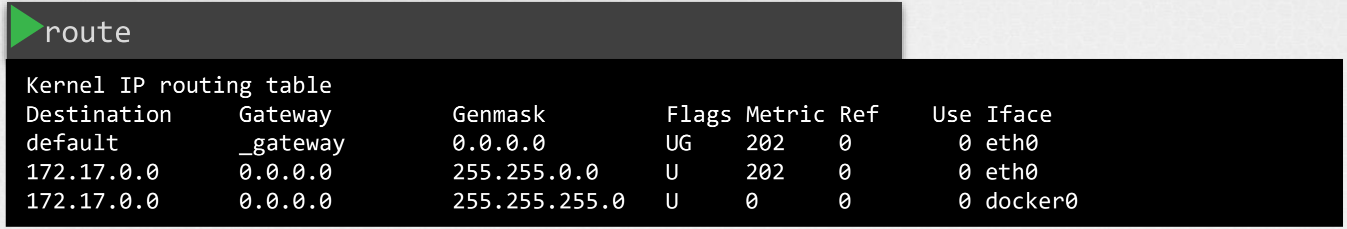
```
arp
Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.0.21              ether   02:42:ac:11:00:15   C                     eth0
172.16.0.8               ether   06:fe:d3:b5:59:65   C                     eth0
_gateway                 ether   02:42:d5:7a:84:8e   C                     eth0
host01                   ether   02:42:ac:11:00:1c   C                     eth0
```

```
ip netns exec red arp
Address                  HWtype  HWaddress           Flags Mask            Iface
```

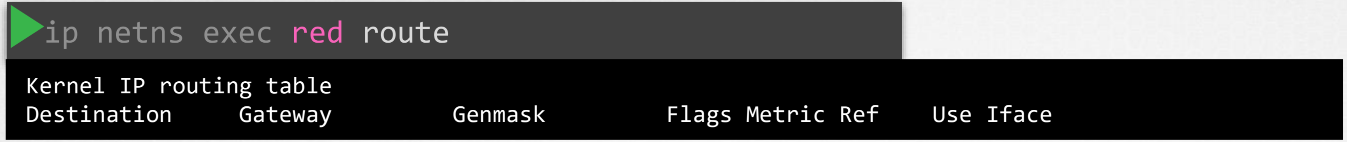


ARP Table	
172.17.0.21	02:42:ac:11:00:15
172.16.0.8	06:fe:d3:b5:59:65

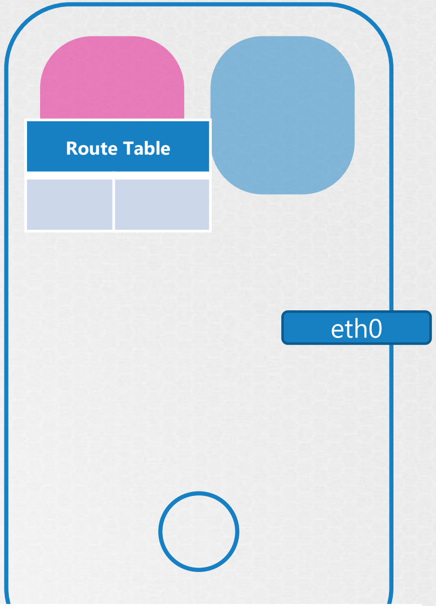
EXEC IN NETWORK NS



```
route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    202    0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     202    0        0 eth0
172.17.0.0      0.0.0.0         255.255.255.0   U     0      0        0 docker0
```

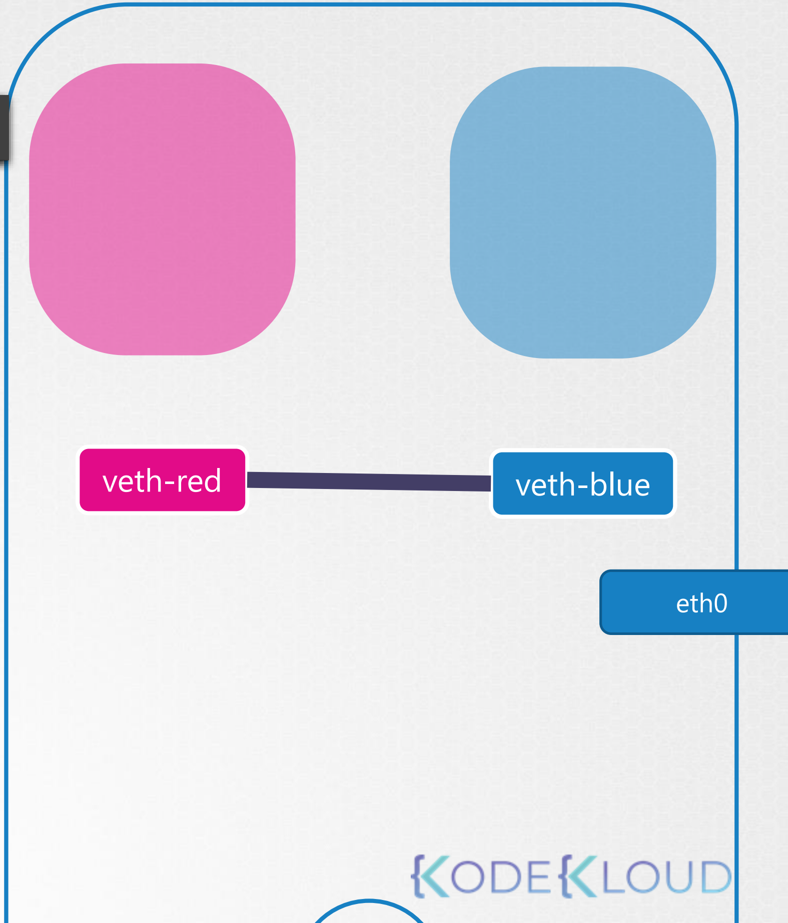


```
ip netns exec red route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```



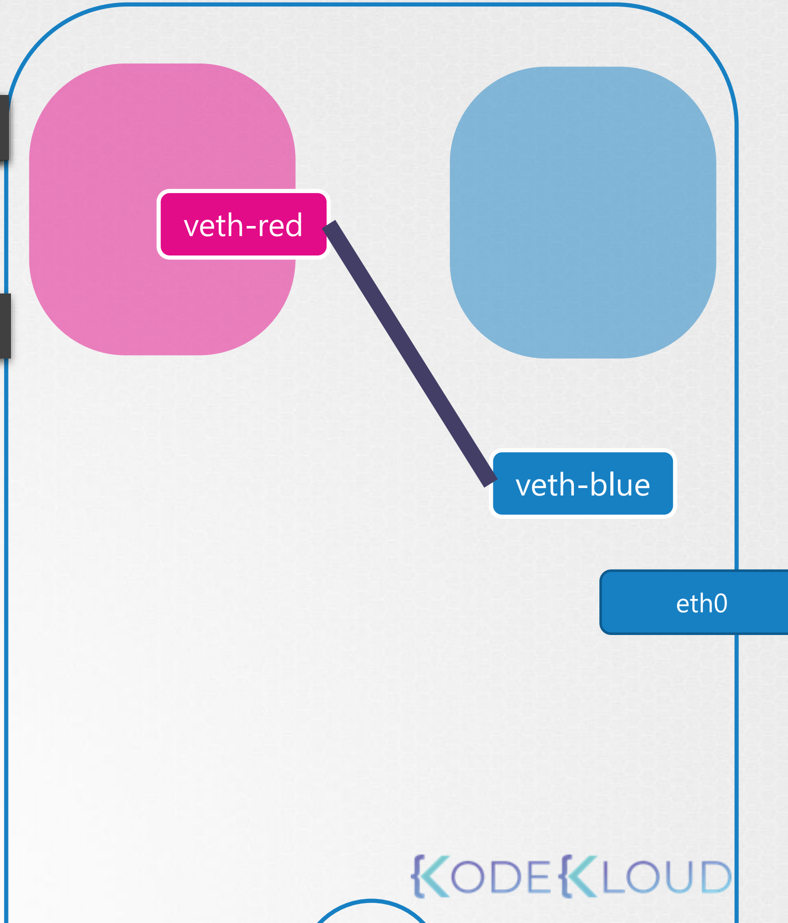
Route Table	
172.17.0.0	0.0.0.0
17.18.0.0	0.0.0.0

```
ip link add veth-red type veth peer name veth-blue
```



```
ip link add veth-red type veth peer name veth-blue
```

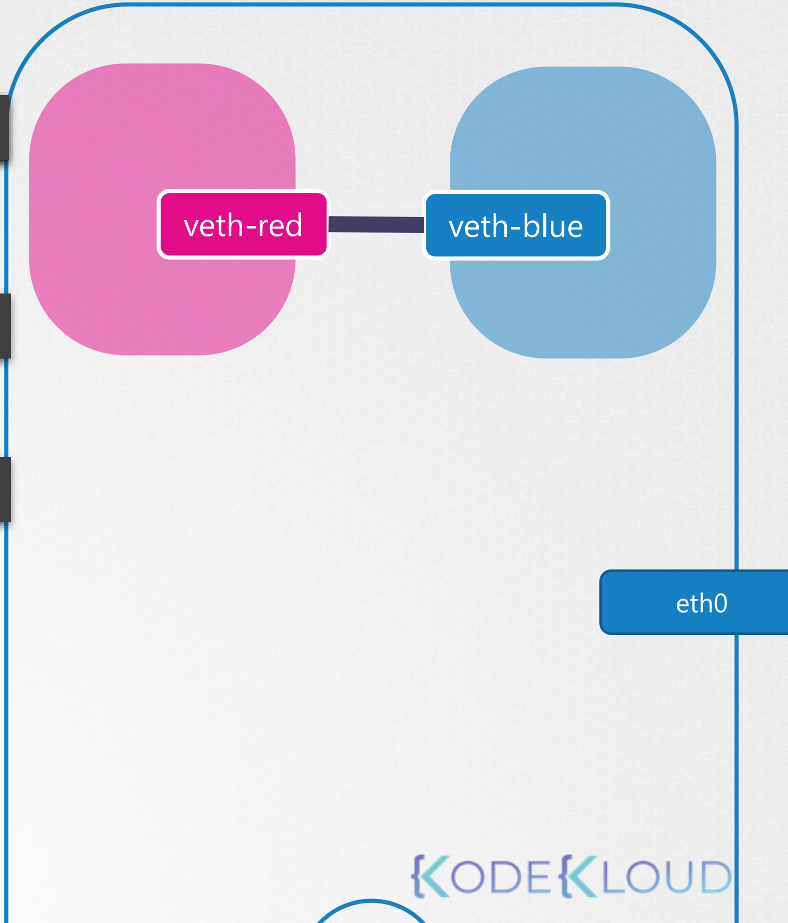
```
ip link set veth-red netns red
```



```
ip link add veth-red type veth peer name veth-blue
```

```
ip link set veth-red netns red
```

```
ip link set veth-blue netns blue
```



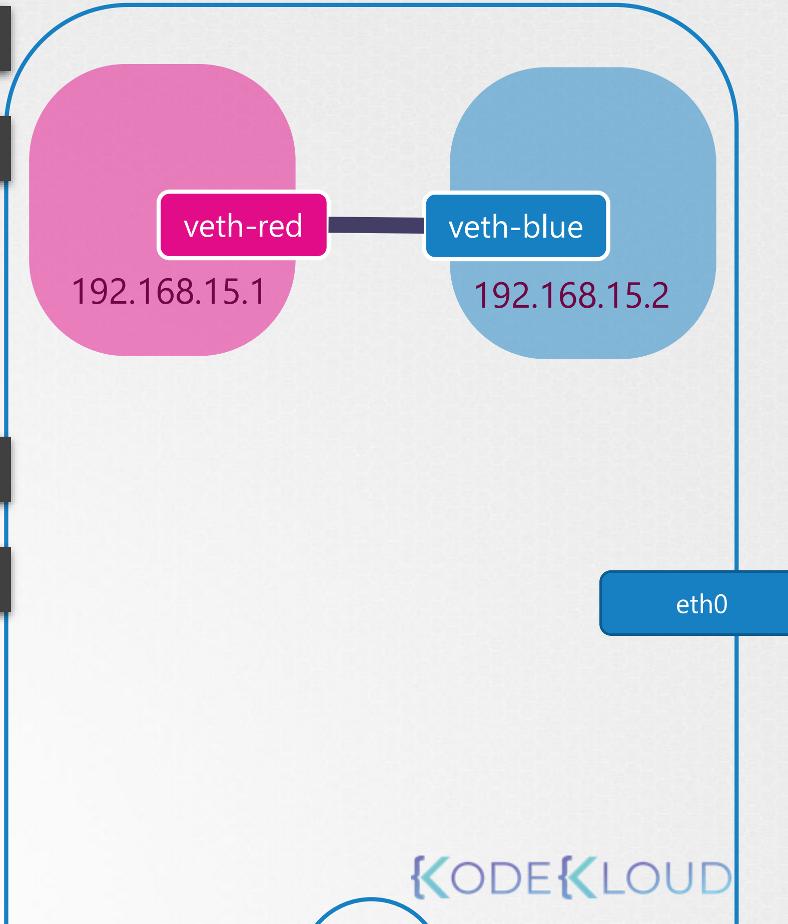
```
ip link add veth-red type veth peer name veth-blue
```

```
ip link set veth-red netns red
```

```
ip link set veth-blue netns blue
```

```
ip -n red  addr add 192.168.15.1 dev veth-red
```

```
ip -n blue addr add 192.168.15.2 dev veth-blue
```



```
ip link add veth-red type veth peer name veth-blue
```

```
ip link set veth-red netns red
```

```
ip link set veth-blue netns blue
```

```
ip -n red  addr add 192.168.15.1 dev veth-red
```

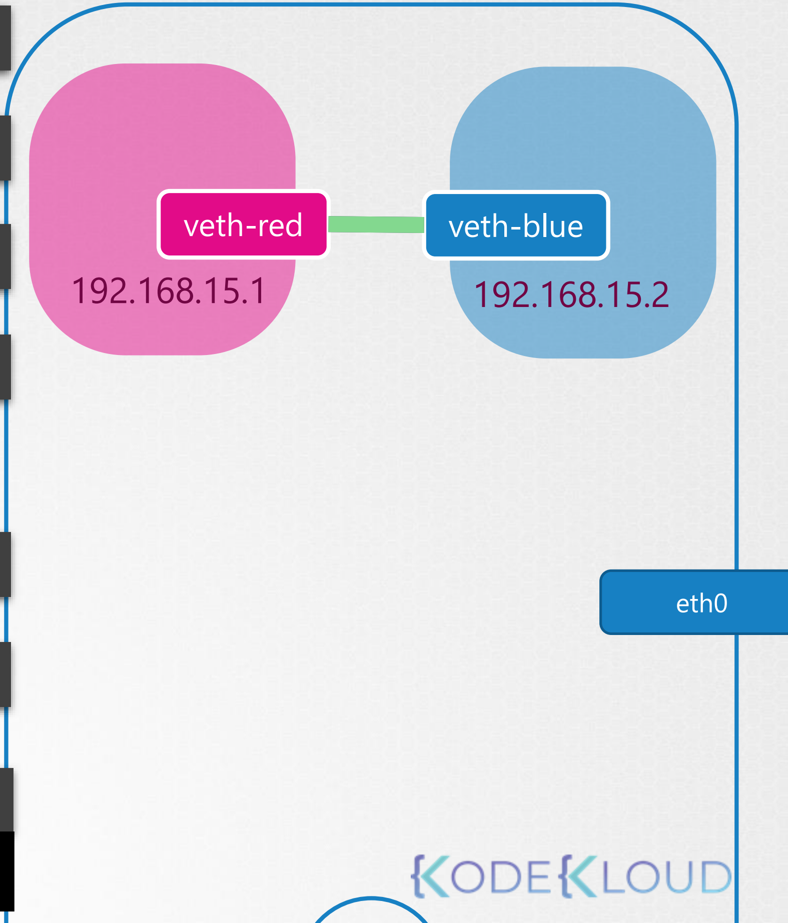
```
ip -n blue addr add 192.168.15.2 dev veth-blue
```

```
ip -n red  link set veth-red up
```

```
ip -n blue link set veth-blue up
```

```
ip netns exec red ping 192.168.15.2
```

```
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data.
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.026 ms
```



KODEKLOUD

```
ip -n red  link set veth-red up
```

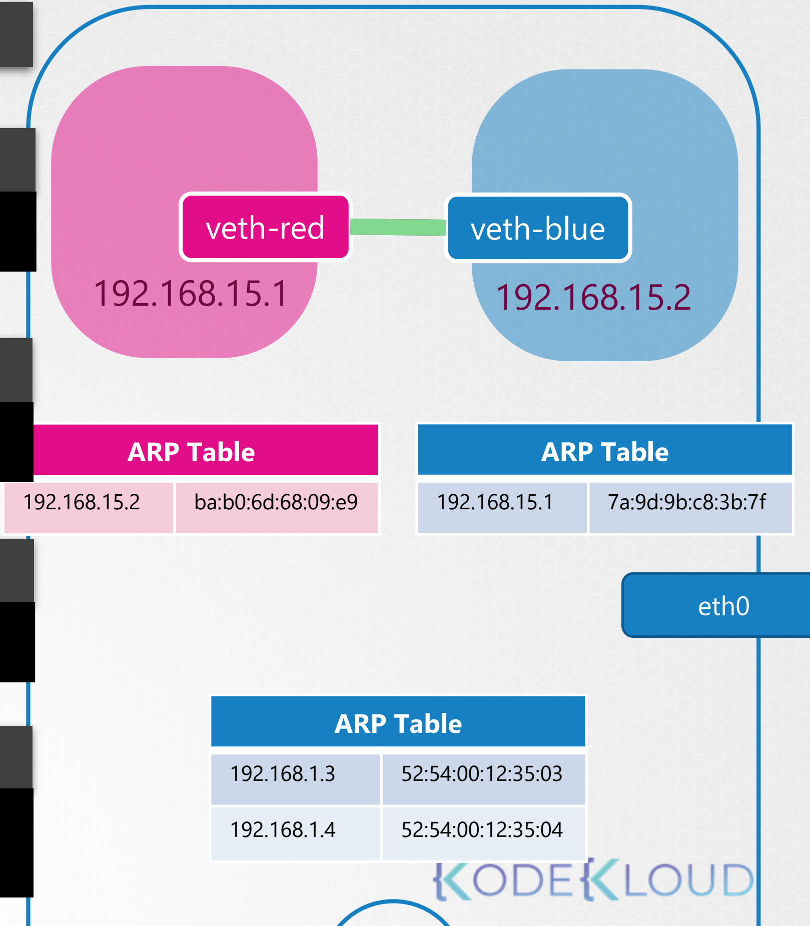
```
ip -n blue link set veth-blue up
```

```
ip netns exec red ping 192.168.15.2
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data.
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.026 ms
```

```
ip netns exec red arp
Address          HWtype  HWaddress          Flags Mask          Iface
192.168.15.2     ether   ba:b0:6d:68:09:e9  C                   veth-red
```

```
ip netns exec blue arp
Address          HWtype  HWaddress          Flags Mask          Iface
192.168.15.1     ether   7a:9d:9b:c8:3b:7f  C                   veth-blue
```

```
 arp
Address                  HWtype  HWaddress          Flags Mask       Iface
192.168.1.3              ether   52:54:00:12:35:03  C                eth0
192.168.1.4              ether   52:54:00:12:35:04  C                eth0
```



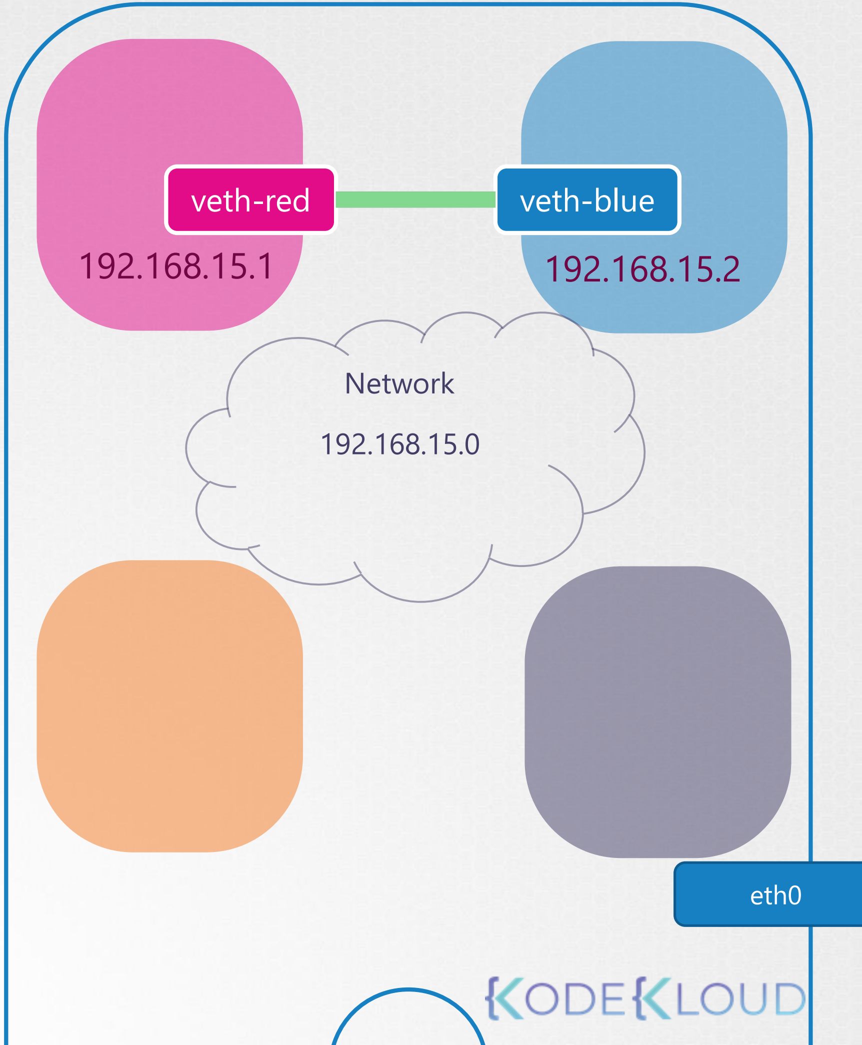
ARP Table	
192.168.15.2	ba:b0:6d:68:09:e9

ARP Table	
192.168.15.1	7a:9d:9b:c8:3b:7f

ARP Table	
192.168.1.3	52:54:00:12:35:03
192.168.1.4	52:54:00:12:35:04

LINUX BRIDGE

Open vSwitch



KODEKLOUD

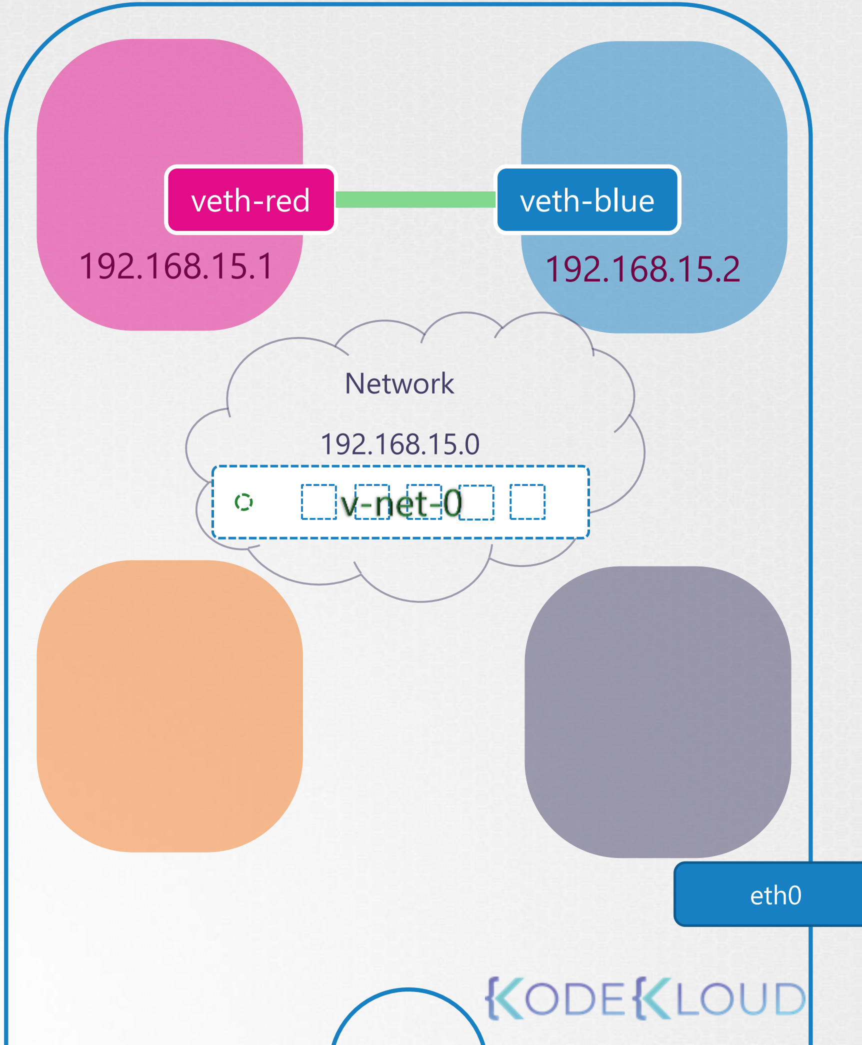
LINUX BRIDGE

```
ip link add v-net-0 type bridge
```

```
ip link
```

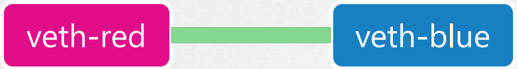
```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:0d:31:14:c7:a7 brd ff:ff:ff:ff:ff:ff
6: v-net-0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
mode DEFAULT group default qlen 1000
    link/ether 06:9d:69:52:6f:61 brd ff:ff:ff:ff:ff:ff
```

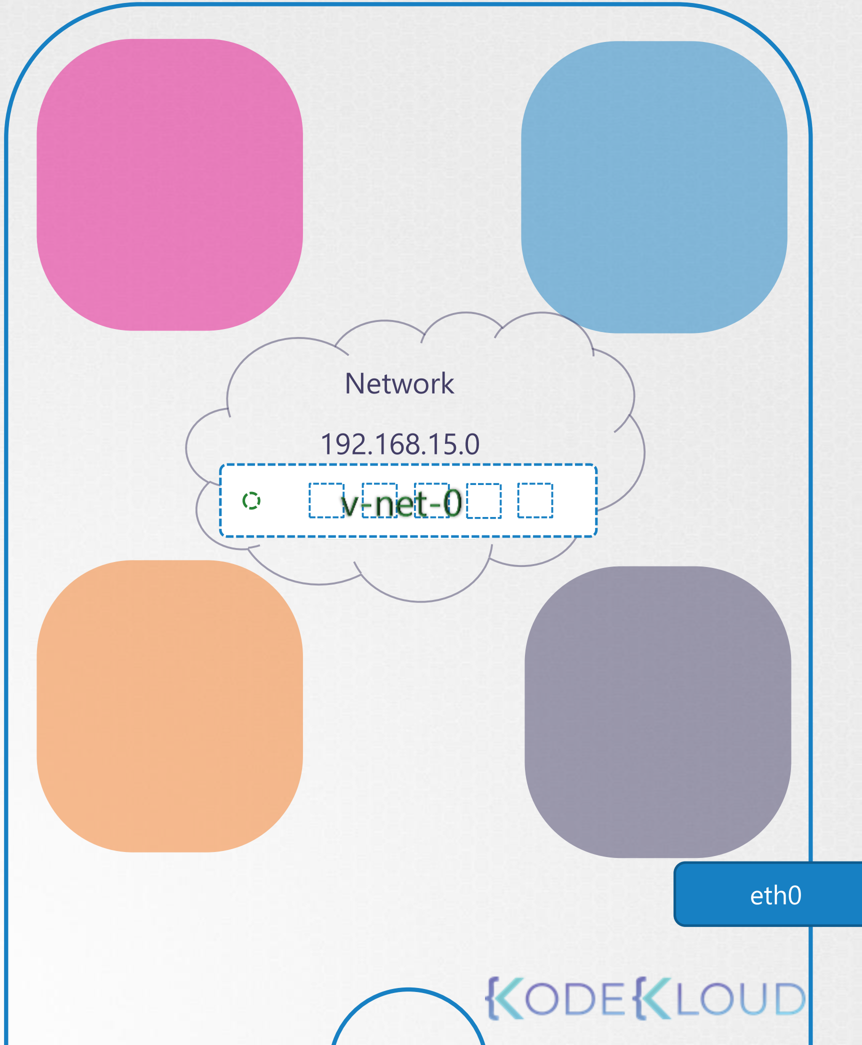
```
ip link set dev v-net-0 up
```



LINUX BRIDGE

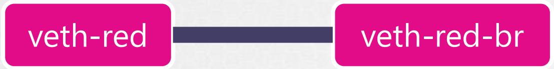
```
ip -n red link del veth-red
```



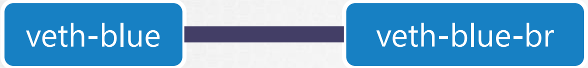


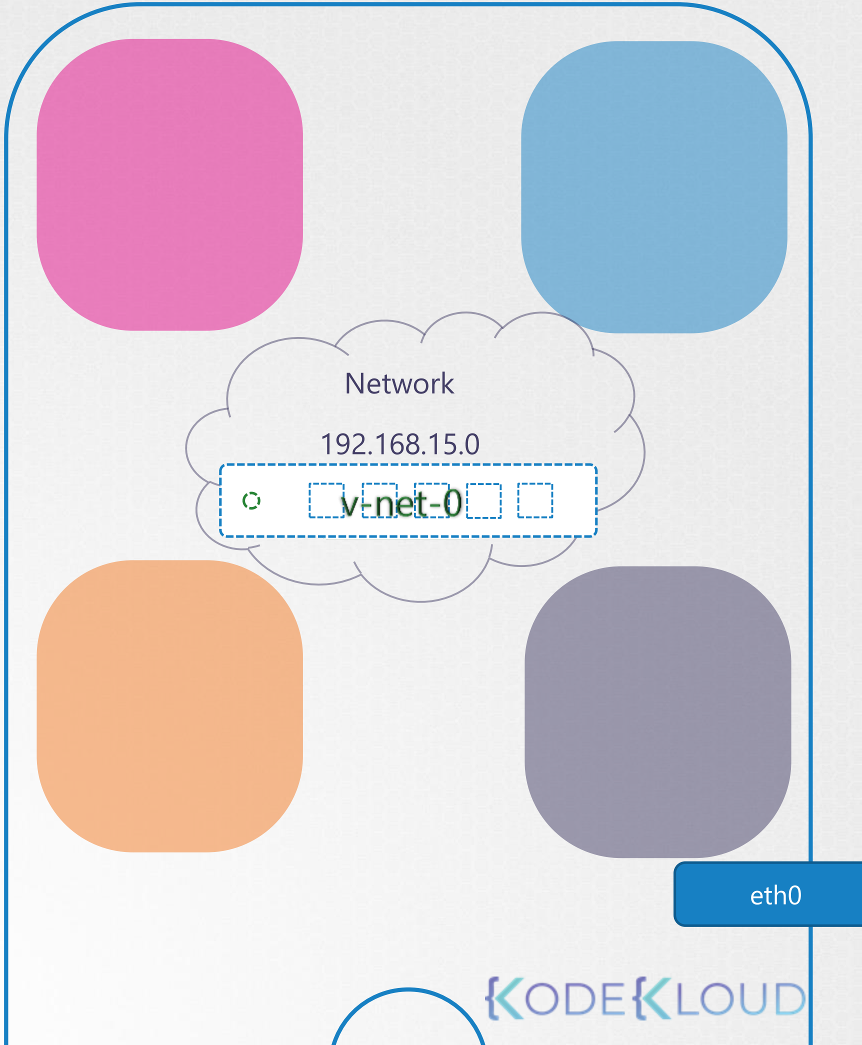
LINUX BRIDGE

```
ip link add veth-red type veth peer name veth-red-br
```



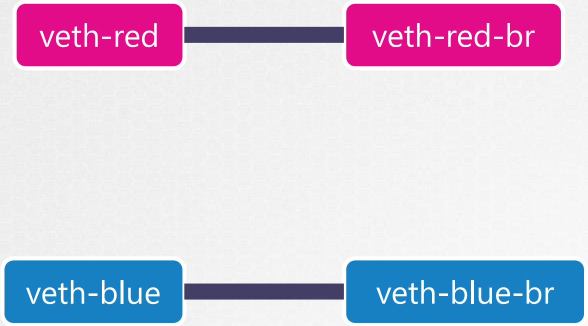
```
ip link add veth-blue type veth peer name veth-blue-br
```

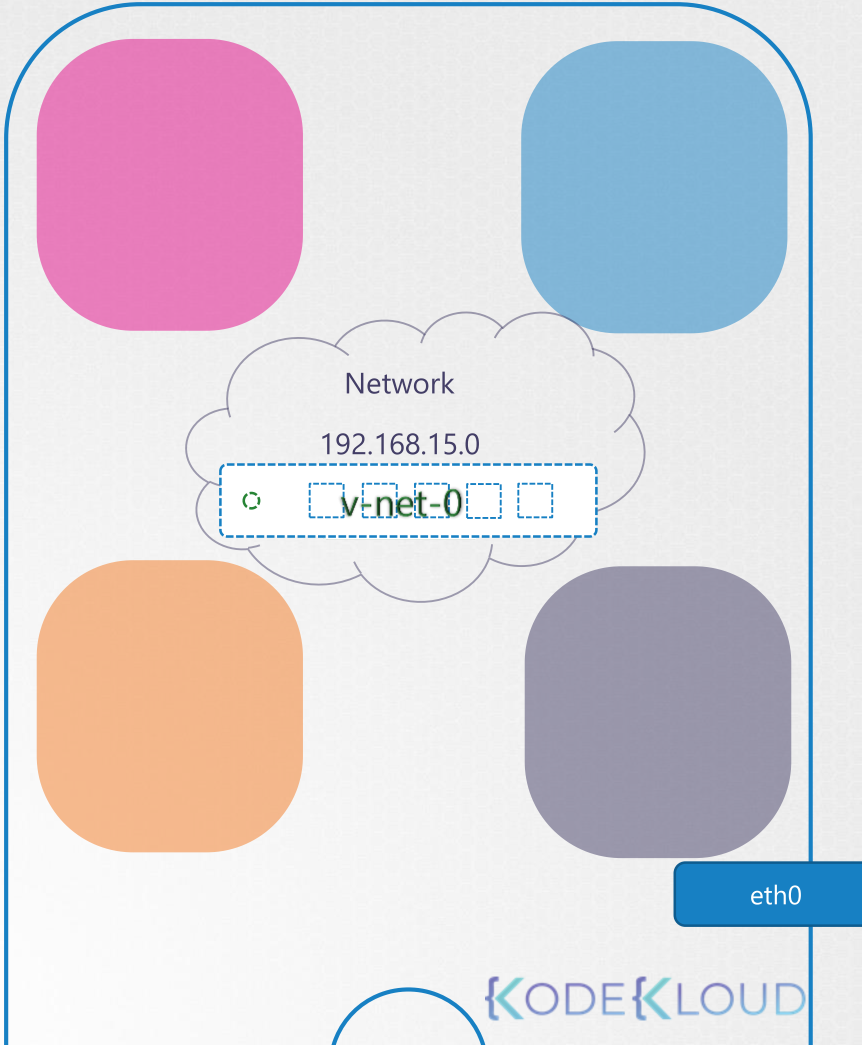




LINUX BRIDGE

```
ip link set veth-red netns red
```

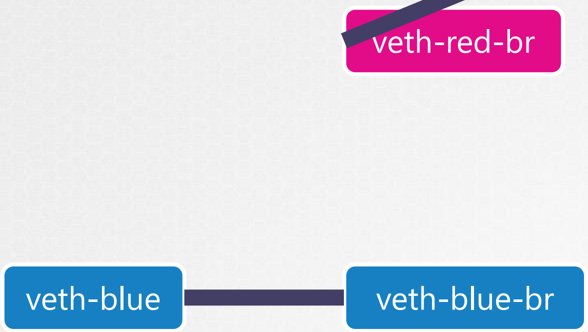


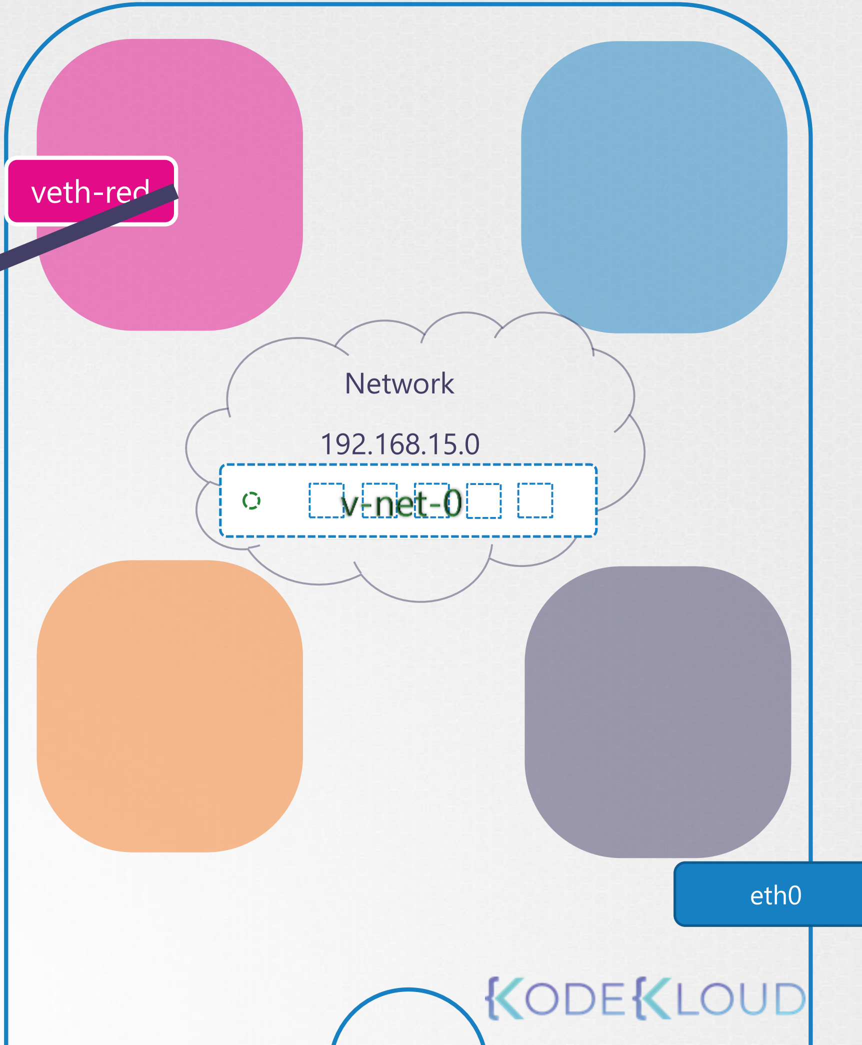


LINUX BRIDGE

```
ip link set veth-red netns red
```

```
ip link set veth-red-br master v-net-0
```



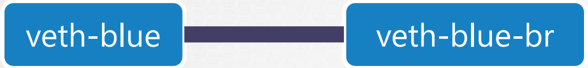


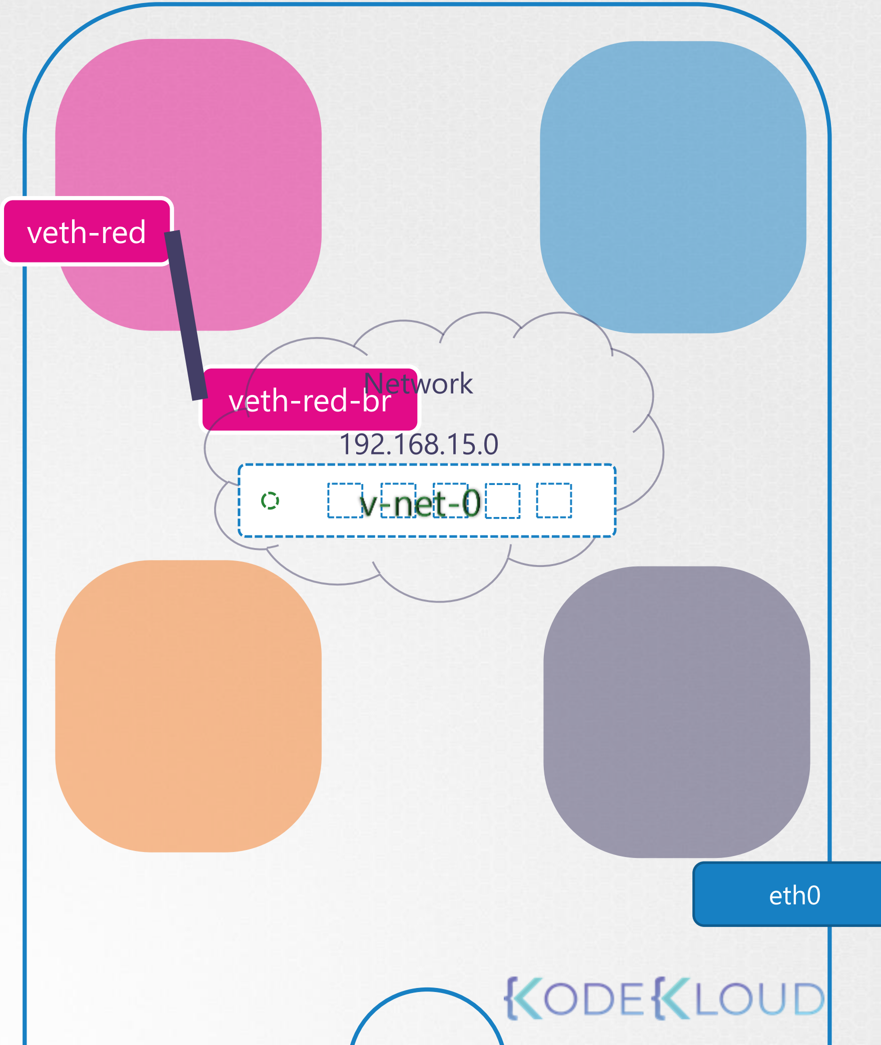
LINUX BRIDGE

```
ip link set veth-red netns red
```

```
ip link set veth-red-br master v-net-0
```

```
ip link set veth-blue netns blue
```





KODEKLOUD

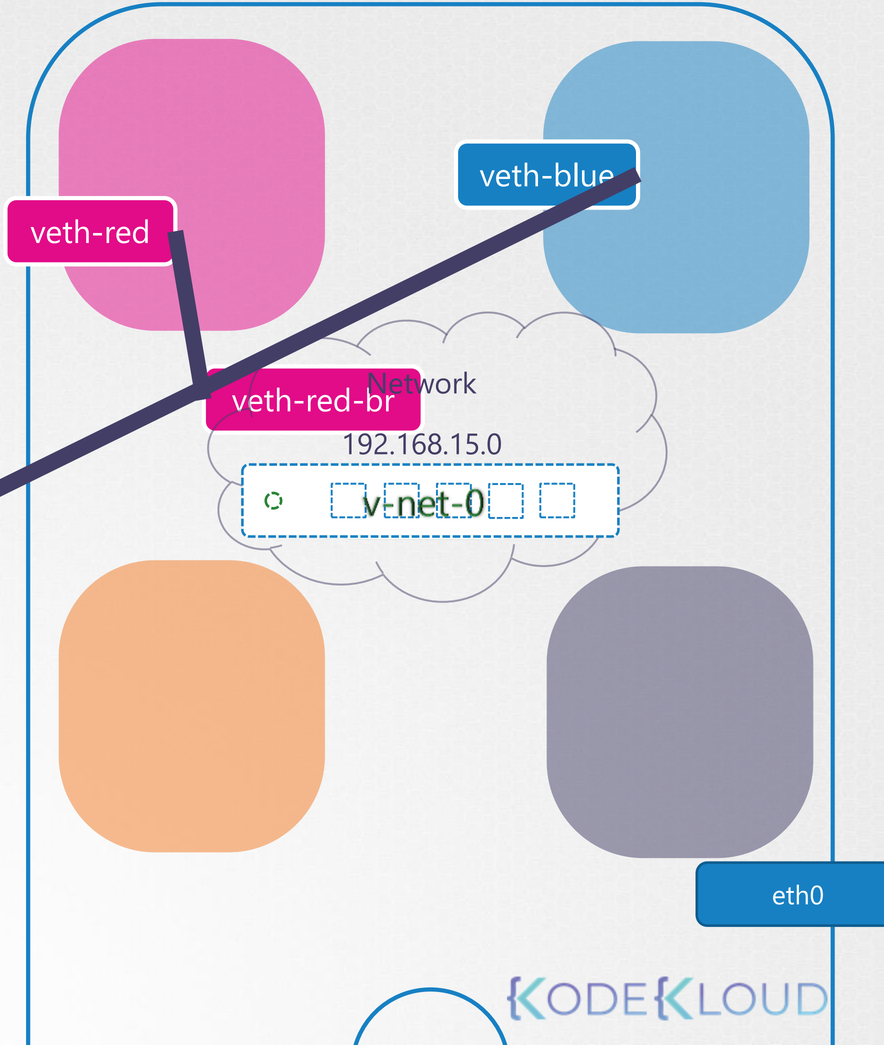
LINUX BRIDGE

```
ip link set veth-red netns red
```

```
ip link set veth-red-br master v-net-0
```

```
ip link set veth-blue netns blue
```

```
ip link set veth-blue-br master v-net-0
```



LINUX BRIDGE

```
ip link set veth-red netns red
```

```
ip link set veth-red-br master v-net-0
```

```
ip link set veth-blue netns blue
```

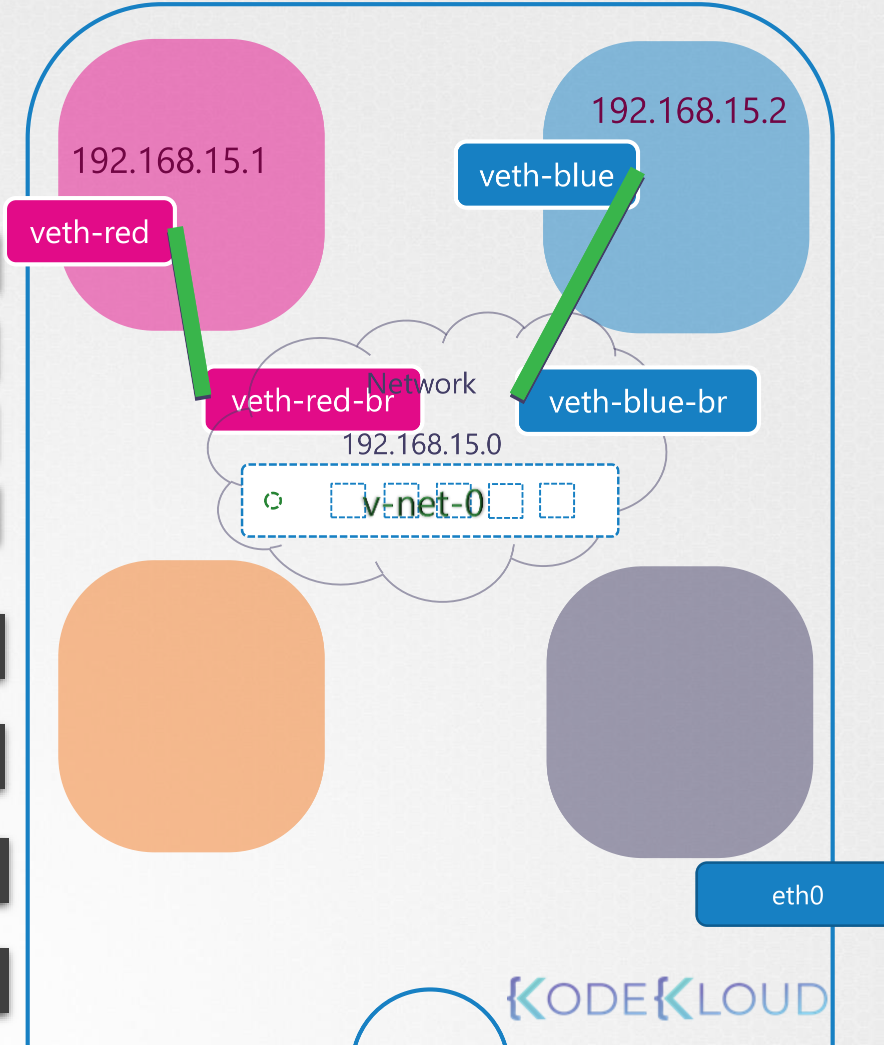
```
ip link set veth-blue-br master v-net-0
```

```
ip -n red addr add 192.168.15.1 dev veth-red
```

```
ip -n blue addr add 192.168.15.2 dev veth-blue
```

```
ip -n red link set veth-red up
```

```
ip -n blue link set veth-blue up
```

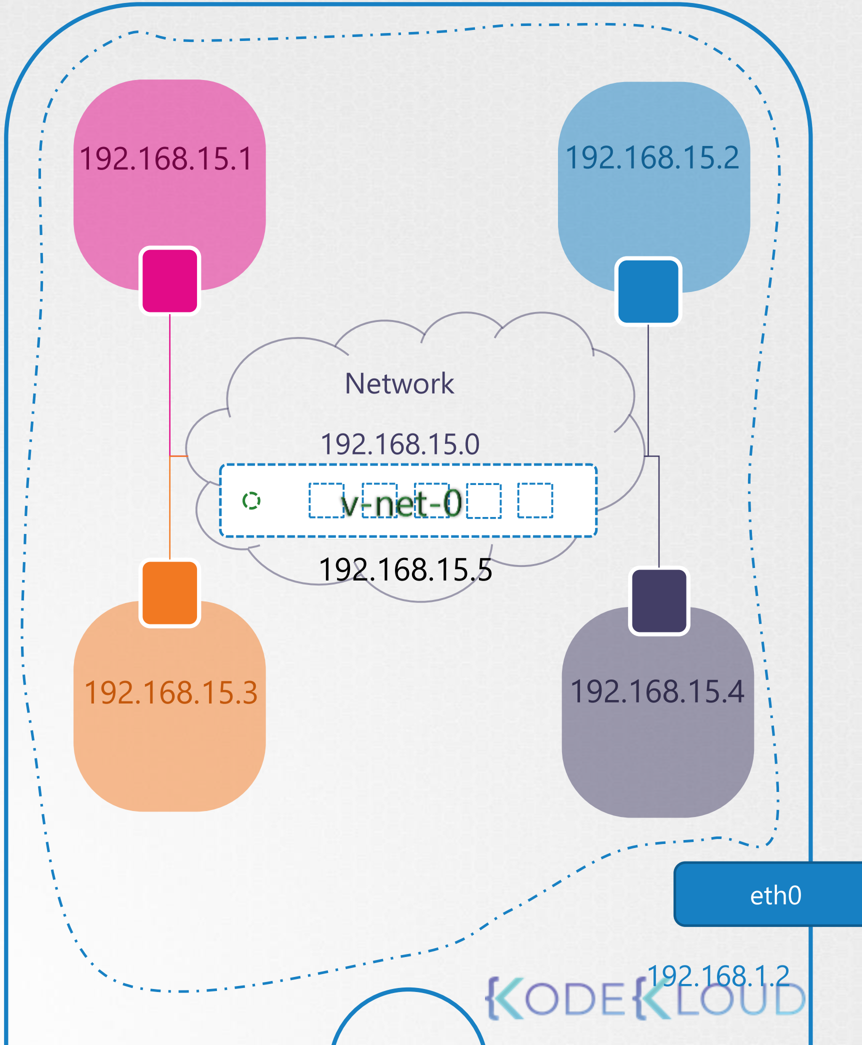


LINUX BRIDGE

```
ping 192.168.15.1
Not Reachable!
```

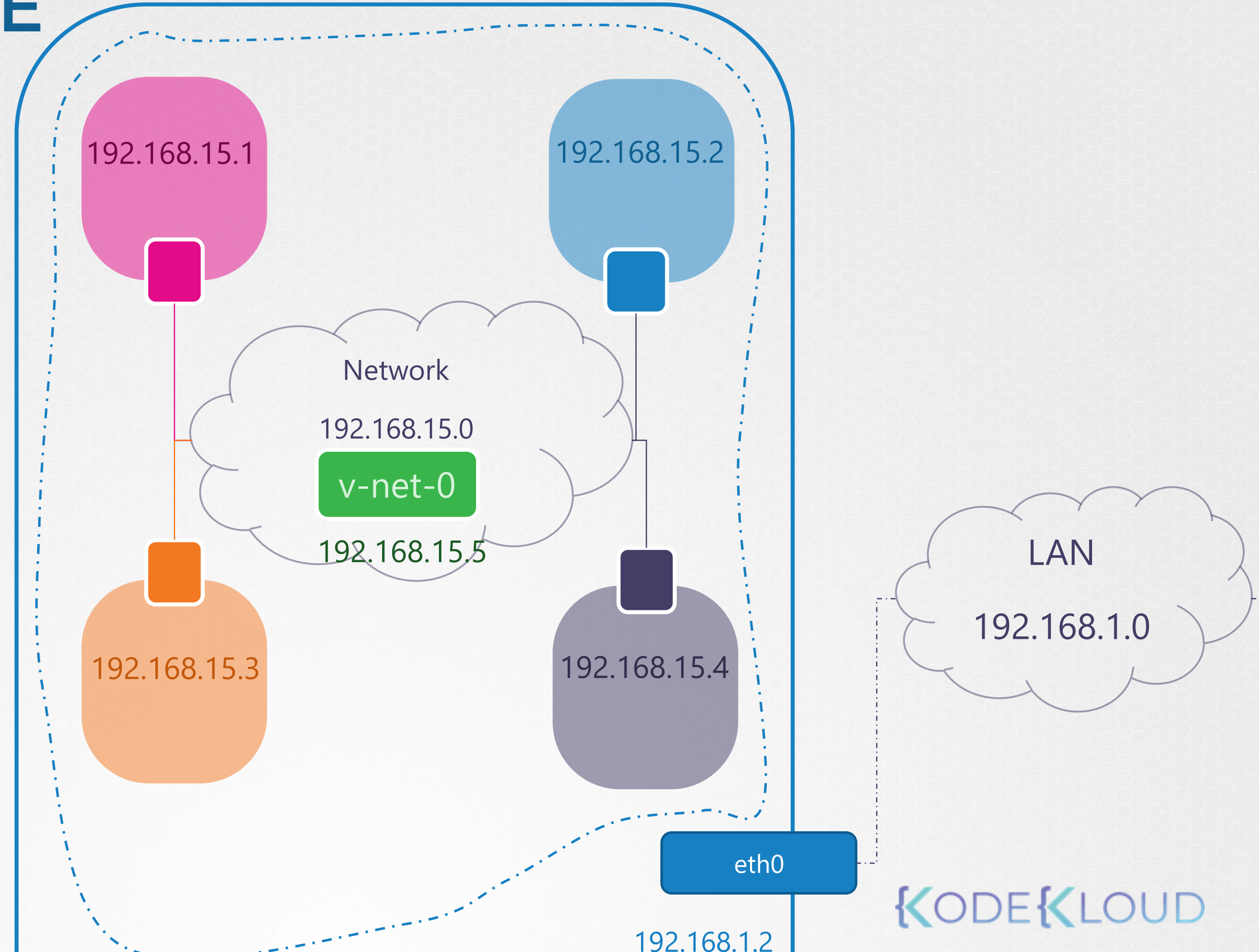
```
ip addr add 192.168.15.5/24 dev v-net-0
```

```
ping 192.168.15.1
PING 192.168.15.1 (192.168.15.1) 56(84) bytes of data.
64 bytes from 192.168.15.1: icmp_seq=1 ttl=64 time=0.026 ms
```



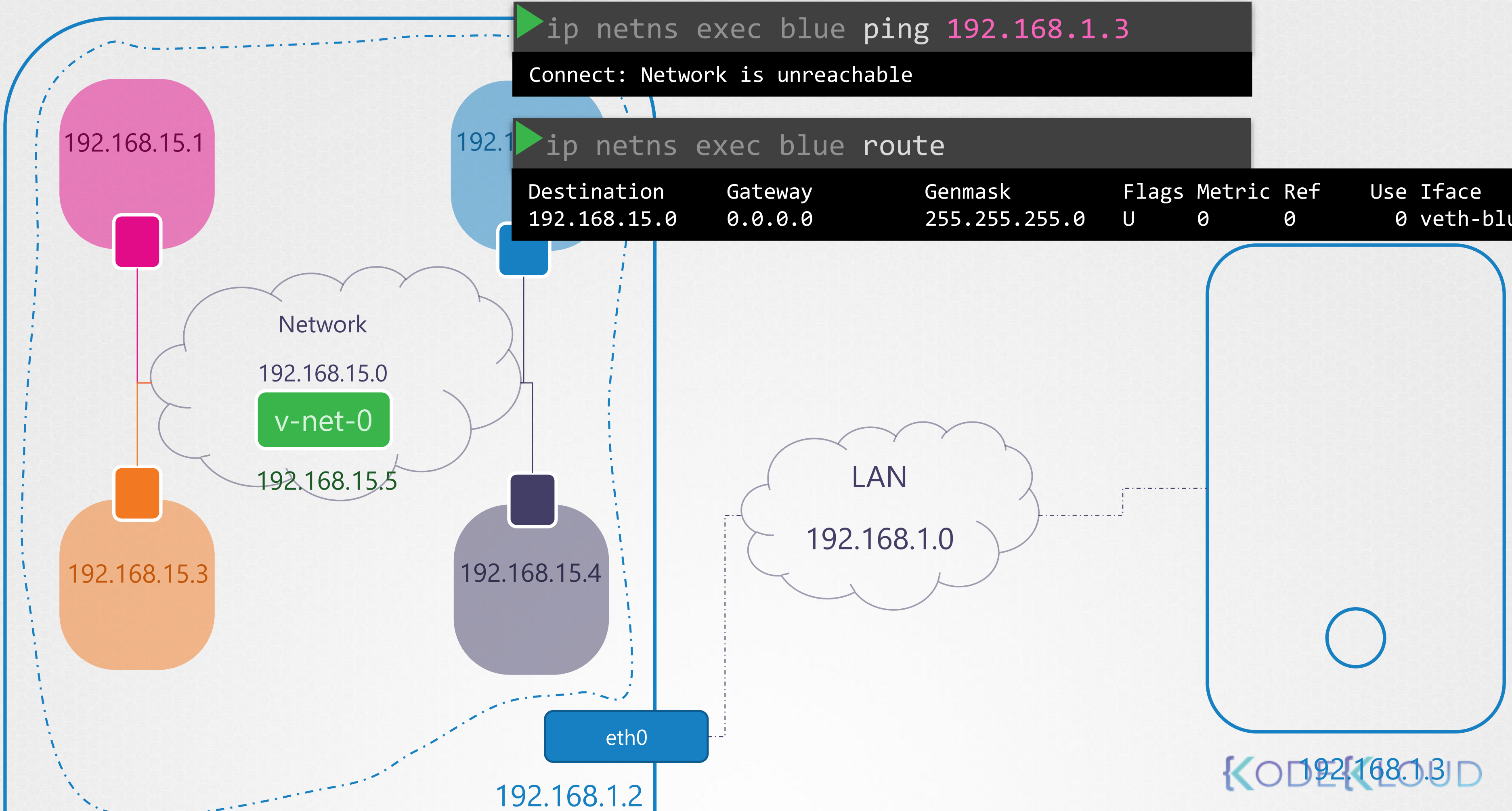
KODEKLOUD

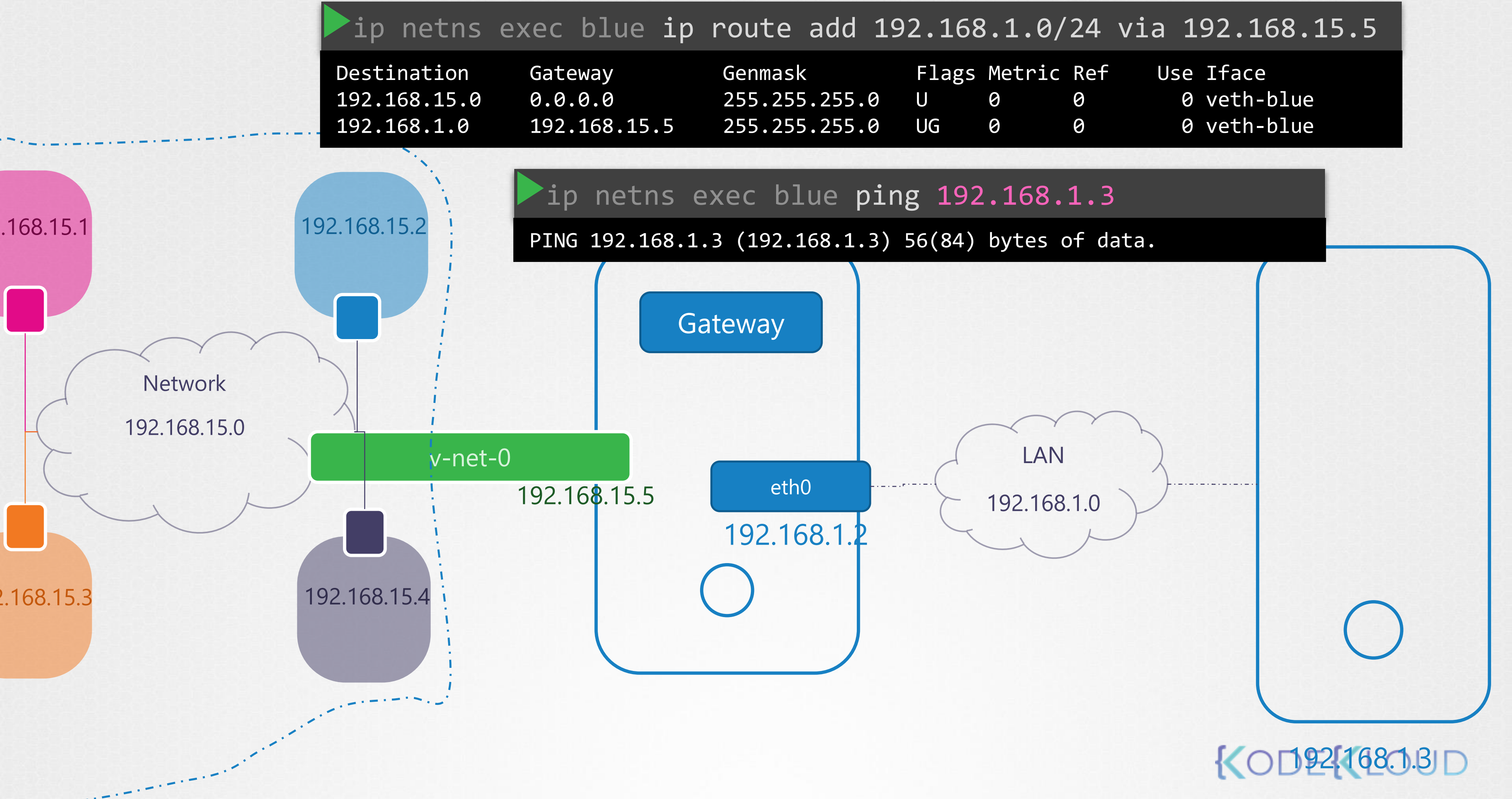
LINUX BRIDGE



```
ip netns exec blue ping 192.168.1.3
Connect: Network is unreachable
```

```
ip netns exec blue route
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0         255.255.255.0   U     0      0        0 veth-blue
```

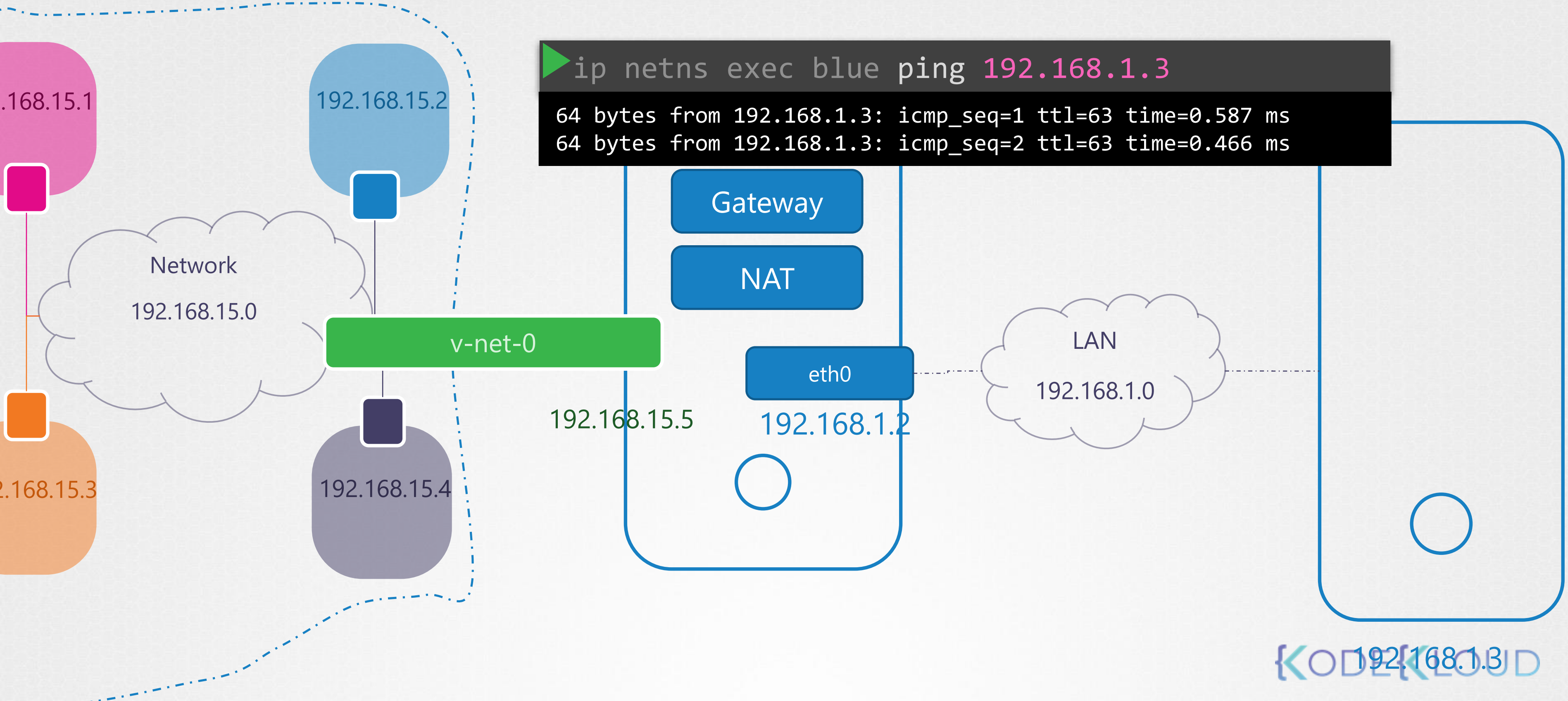




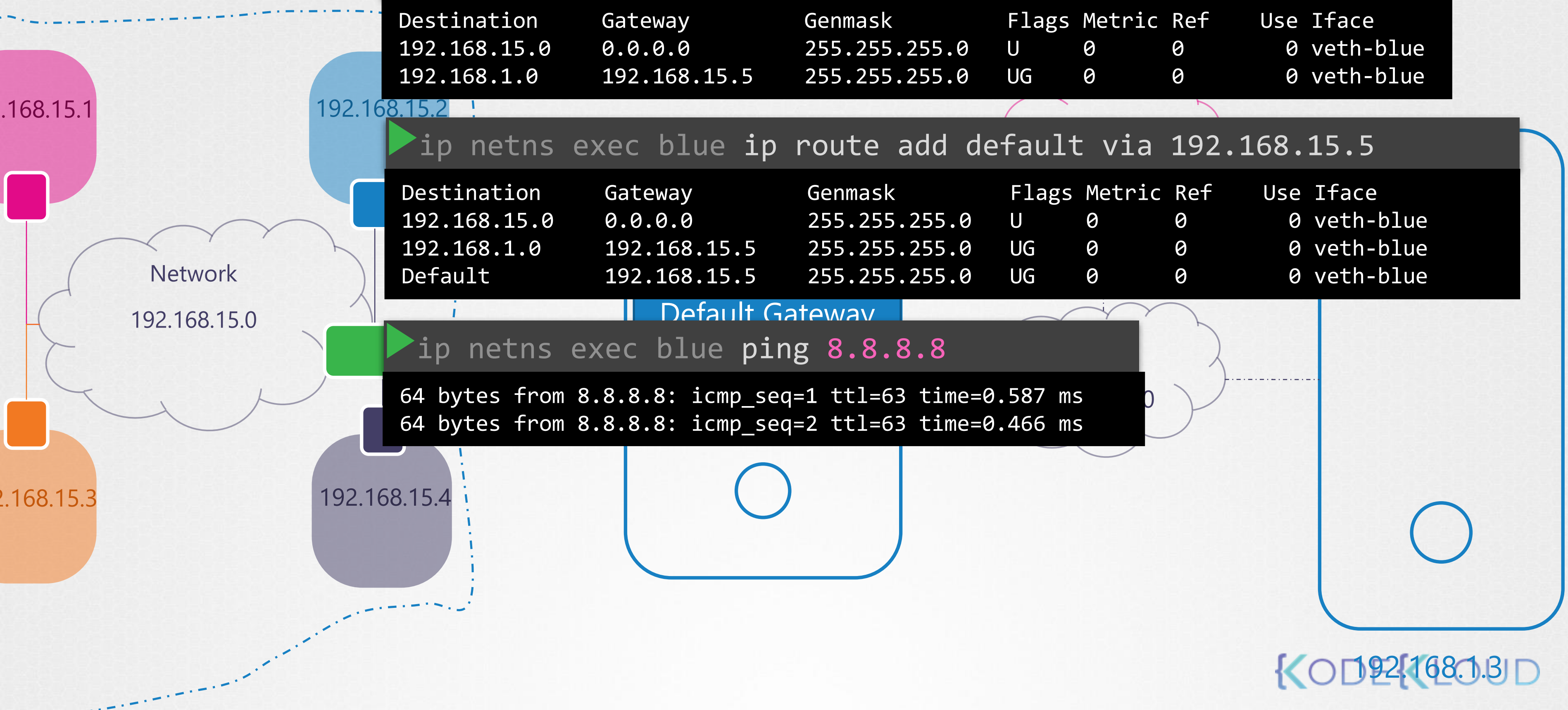
```
▶ip netns exec blue ip route add 192.168.1.0/24 via 192.168.15.5
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0         255.255.255.0   U     0      0        0 veth-blue
192.168.1.0     192.168.15.5    255.255.255.0   UG    0      0        0 veth-blue
```

```
▶ip netns exec blue ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
```

```
▶ iptables -t nat -A POSTROUTING -s 192.168.15.0/24 -j MASQUERADE
```



```
▶ip netns exec blue ping 192.168.1.3
64 bytes from 192.168.1.3: icmp_seq=1 ttl=63 time=0.587 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=63 time=0.466 ms
```



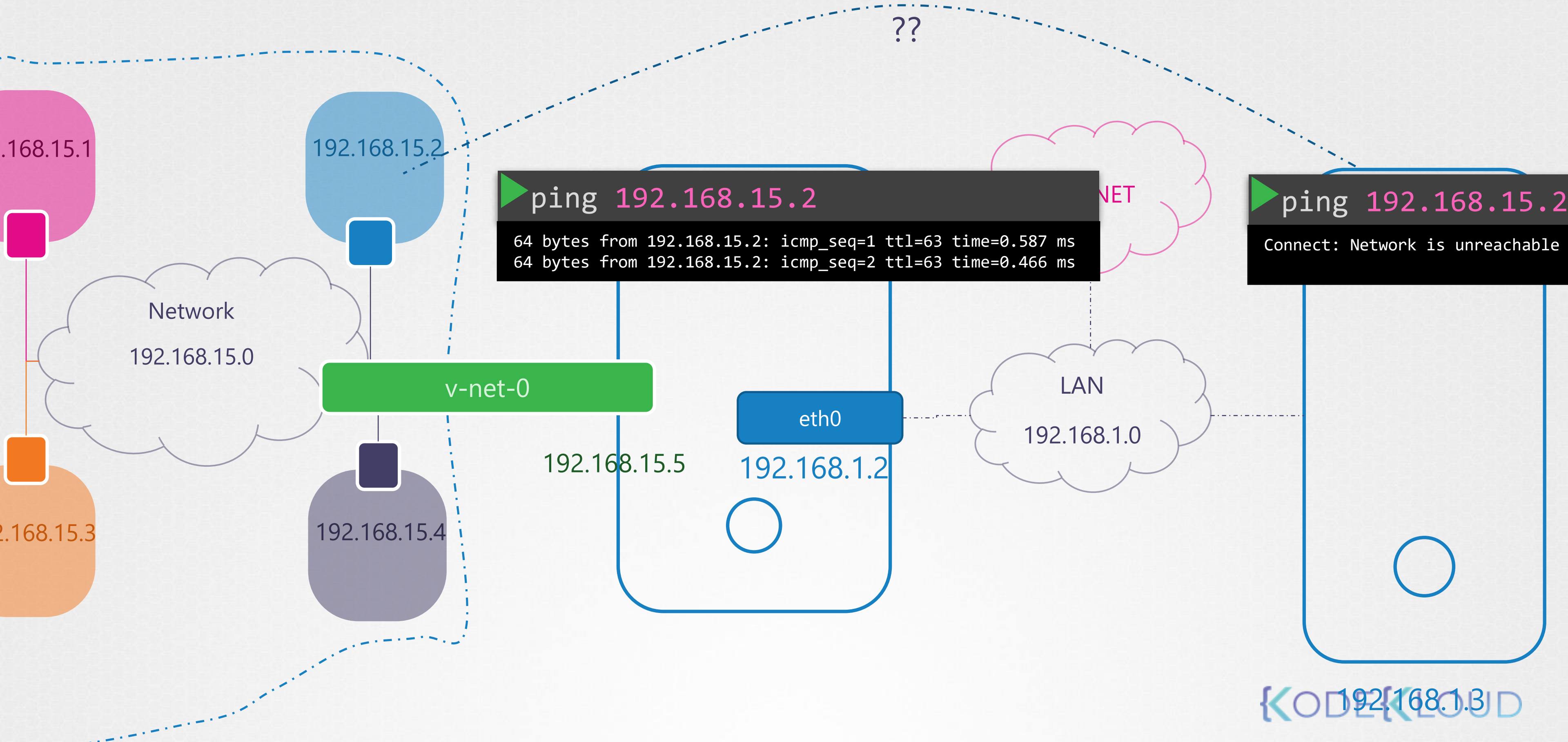
```
ip netns exec blue ping 8.8.8.8
Connect: Network is unreachable
```

```
ip netns exec blue route
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0         255.255.255.0   U     0      0        0 veth-blue
192.168.1.0     192.168.15.5    255.255.255.0   UG    0      0        0 veth-blue
```

```
ip netns exec blue ip route add default via 192.168.15.5
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.15.0    0.0.0.0         255.255.255.0   U     0      0        0 veth-blue
192.168.1.0     192.168.15.5    255.255.255.0   UG    0      0        0 veth-blue
Default         192.168.15.5    255.255.255.0   UG    0      0        0 veth-blue
```

```
ip netns exec blue ping 8.8.8.8
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=0.587 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=0.466 ms
```

```
iptables -t nat -A PREROUTING --dport 80 --to-destination 192.168.15.2:80 -j DNAT
```



```
ping 192.168.15.2
64 bytes from 192.168.15.2: icmp_seq=1 ttl=63 time=0.587 ms
64 bytes from 192.168.15.2: icmp_seq=2 ttl=63 time=0.466 ms
```

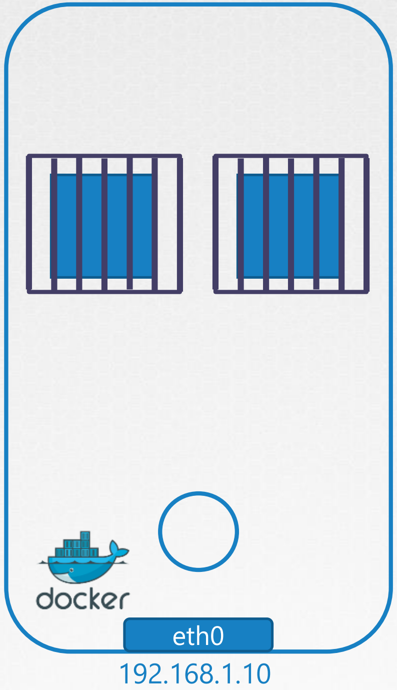
```
ping 192.168.15.2
Connect: Network is unreachable
```

```
# Create network namespaces
ip netns add red
ip netns add blue

# Create veth pairs
ip link add veth-red type veth peer name veth-blue

# Create Add veth to respective namespaces
ip link set veth-red netns red
ip link set veth-blue netns blue

# Set IP Addresses
ip -n red addr add 192.168.1.1 dev veth-red
ip -n blue addr add 192.168.1.2 dev veth-blue

# Check IP Addresses
ip -n red addr
ip -n blue addr

# Bring up interfaces
ip -n red link set veth-red up
ip -n blue link set veth-blue up

# Bring Loopback devices up
ip -n red link set lo up
ip -n blue link set lo up

# Add default gateway
ip netns exec red ip route add default via 192.168.1.1 dev
veth-red
ip netns exec blue ip route add default via 192.168.1.2 dev
ip -n blue link del veth-blue

ip netns del red
ip netns del blue

ip link del v-net-0

iptables -t nat -D POSTROUTING 1

#



ip netns add red
ip netns add blue

ip link add veth-red type veth peer name veth-red-br
ip link add veth-blue type veth peer name veth-blue-br

ip link set veth-red netns red
ip link set veth-blue netns blue

ip -n red addr add 192.168.15.2/24 dev veth-red

ip -n blue addr add 192.168.15.3/24 dev veth-blue


brctl addbr v-net-0

ip link set dev v-net-0 up

ip link set veth-red-br up
ip link set veth-blue-br up
```

PRE-REQUISITE

DOCKER NETWORKING

NONE

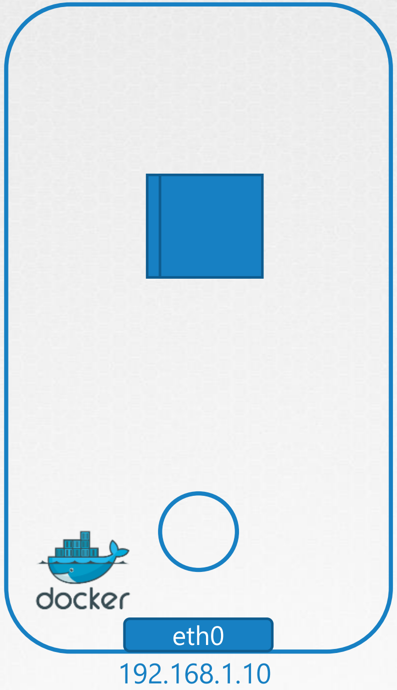


```
docker run --network none nginx
```

```
docker run --network none nginx
```

HOST

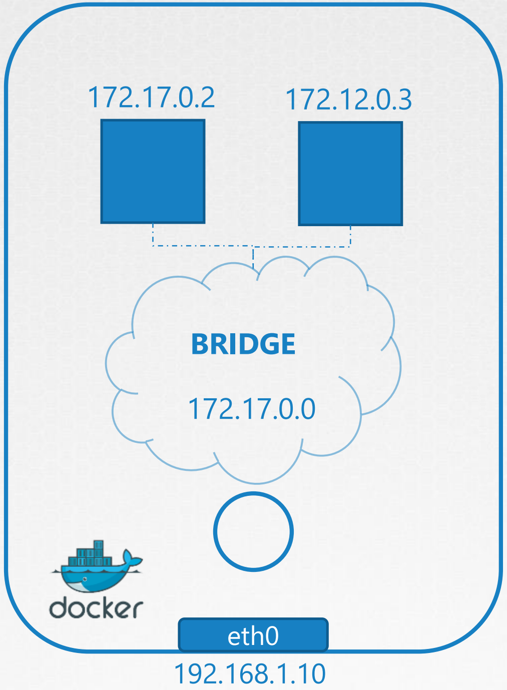
```
http://192.168.1.10:80
```



```
docker run --network host nginx
```

```
docker run --network host nginx
```

BRIDGE



```
▶ docker run nginx
```

```
▶ docker run nginx
```

KODEKLOUD

DOCKER NETWORKING Deep Dive

BRIDGE

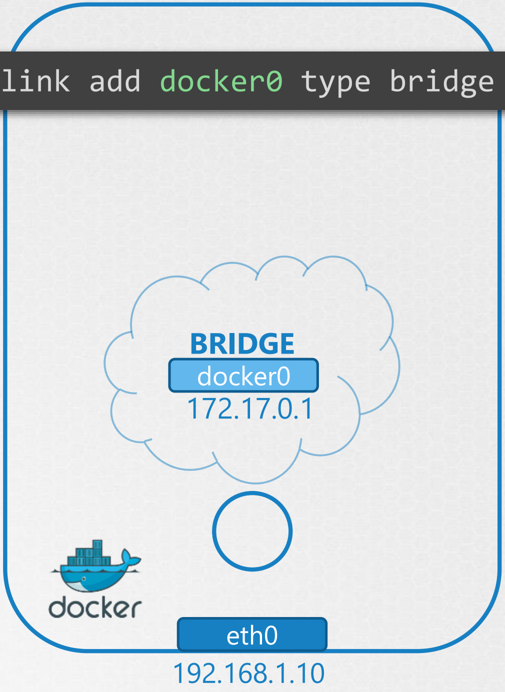
```
▶ docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
2b60087261b2        bridge              bridge              local
0beb4870b093        host                host                local
99035e02694f        none                null                local
```

```
▶ ip link
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:88:56:50:83 brd ff:ff:ff:ff:ff:ff
```

```
▶ ip link add docker0 type bridge
```



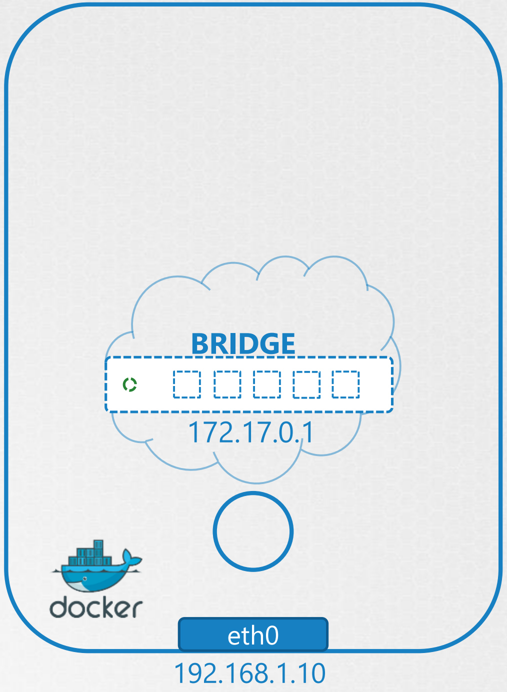
BRIDGE

```
▶ ip link
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:88:56:50:83 brd ff:ff:ff:ff:ff:ff
```

```
▶ ip addr
```

```
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:88:56:50:83 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/24 brd 172.17.0.255 scope global docker0
       valid_lft forever preferred_lft forever
```



BRIDGE

```
▶ ip addr
```

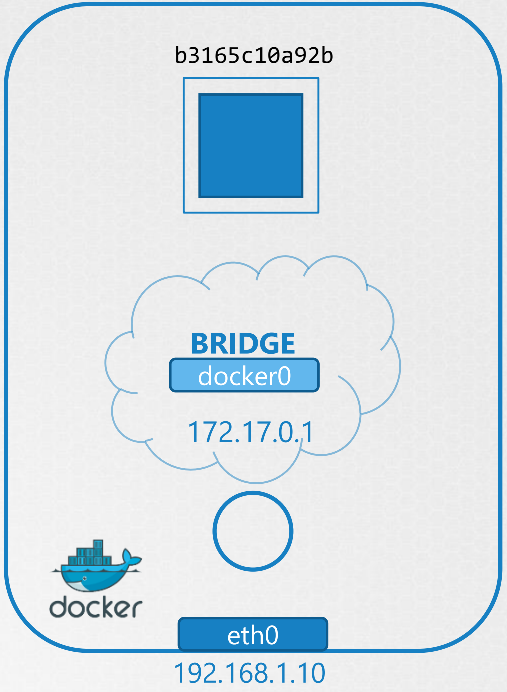
```
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:88:56:50:83 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/24 brd 172.17.0.255 scope global docker0
       valid_lft forever preferred_lft forever
```

```
▶ ip netns
```

```
b3165c10a92b
```

```
▶ docker inspect 942d70e585b2
```

```
 "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "b3165c10a92b50edce4c8aa5f37273e180907ded31",
           "SandboxKey": "/var/run/docker/netns/b3165c10a92b",
```



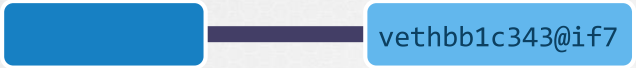
```
▶ docker run nginx
```

```
2e41deb9ef1b8b3d141c7bb55d883541b4
```

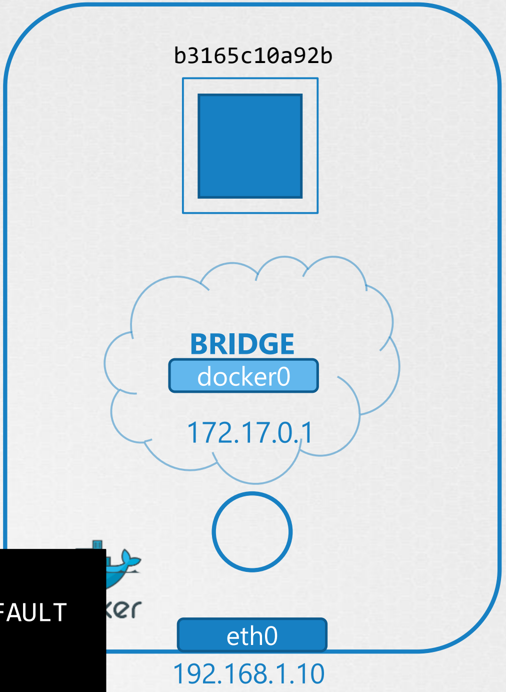
KODEKLOUD

BRIDGE

```
▶ ip netns
b3165c10a92b
```



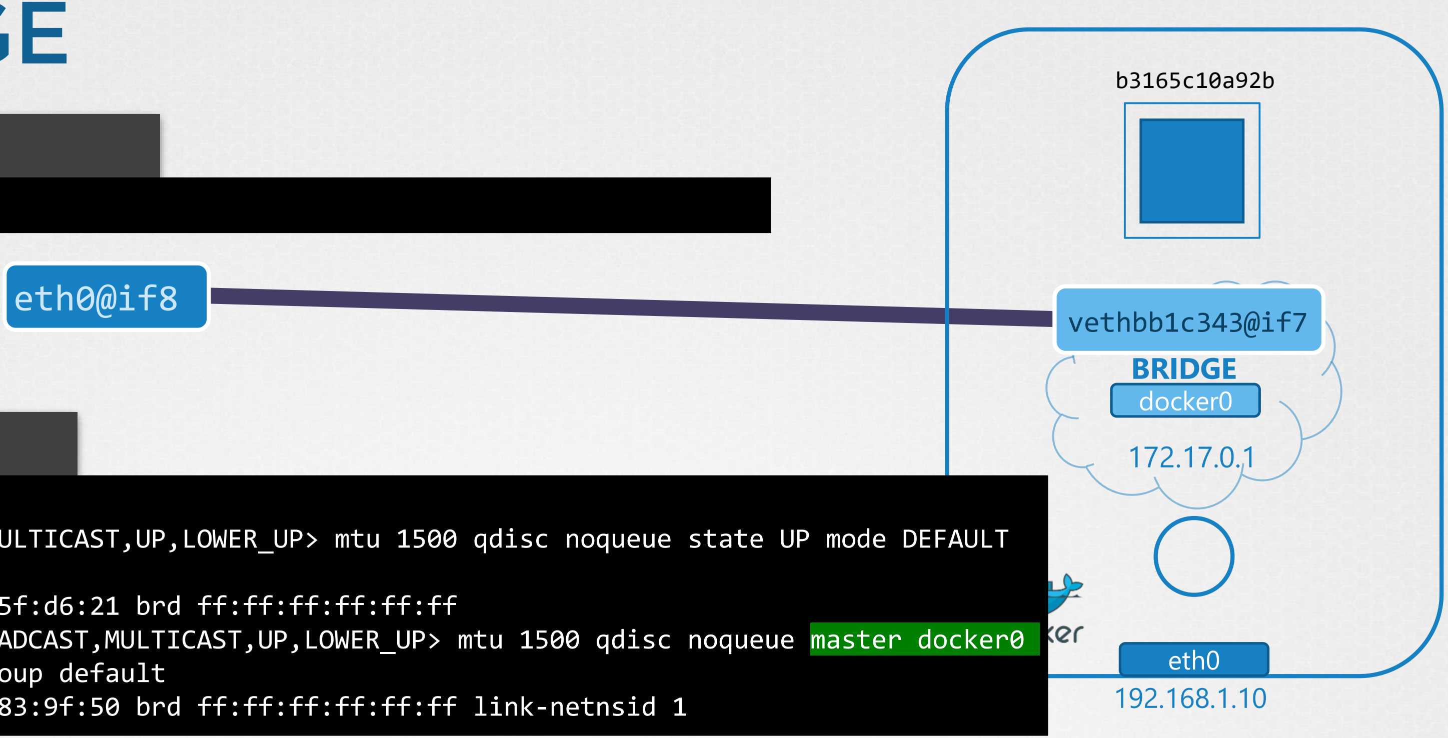
```
▶ ip link
...
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:9b:5f:d6:21 brd ff:ff:ff:ff:ff:ff
8: vethbb1c343@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0
state UP mode DEFAULT group default
    link/ether 9e:71:37:83:9f:50 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```



```
ocker run nginx
2e41deb9ef1b8b3d141c7bb55d883541b4
```

BRIDGE

```
▶ ip netns
b3165c10a92b
```



```
▶ ip link
...
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:9b:5f:d6:21 brd ff:ff:ff:ff:ff:ff
8: vethbb1c343@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0
state UP mode DEFAULT group default
    link/ether 9e:71:37:83:9f:50 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

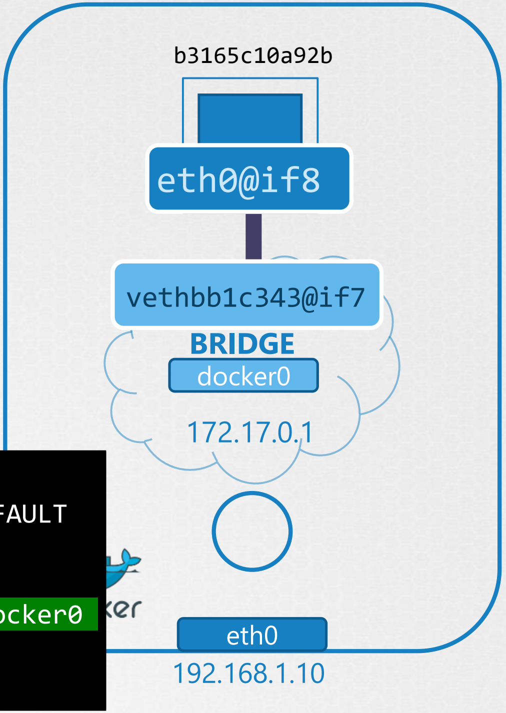
```
▶ ip lin b3165c10a92b
...
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

```
▶ docker run nginx
2e41deb9ef1b8b3d141c7bb55
```

BRIDGE

```
▶ ip netns
```

```
b3165c10a92b
```



```
▶ ip link
```

```
...
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:9b:5f:d6:21 brd ff:ff:ff:ff:ff:ff
8: vethbb1c343@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0
state UP mode DEFAULT group default
    link/ether 9e:71:37:83:9f:50 brd ff:ff:ff:ff:ff:ff link-netnsid 1
```

```
▶ ip -n b3165c10a92b link
```

```
...
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

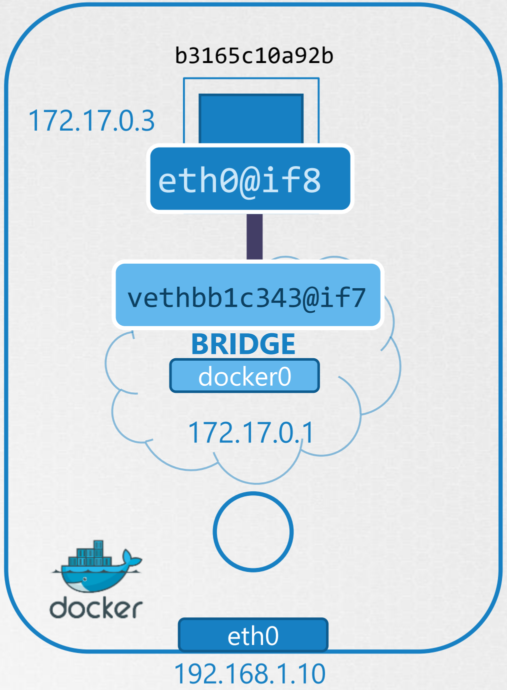
```
▶ docker run nginx
```

```
2e41deb9ef1b8b3d141c7bb55
```

BRIDGE

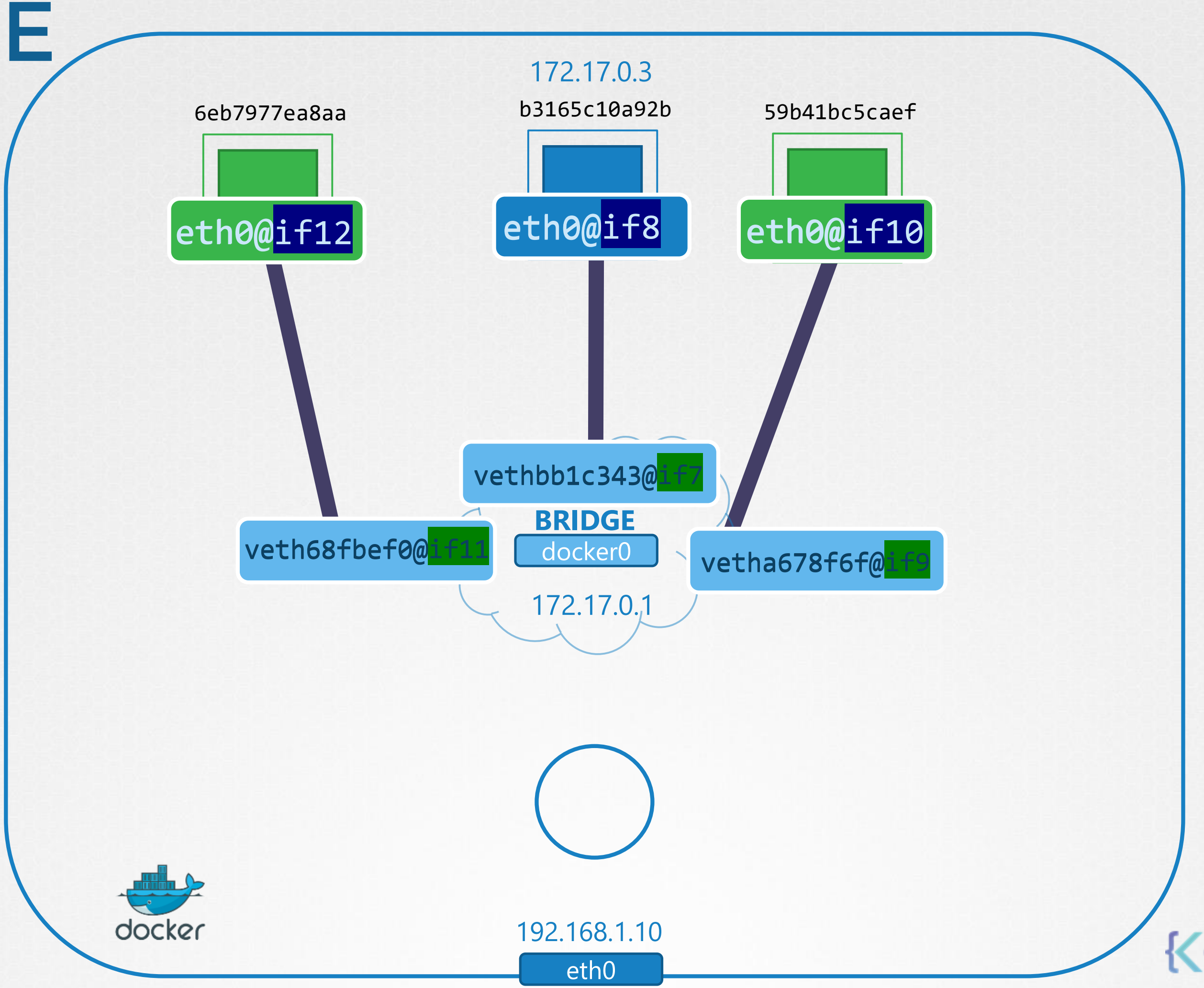
```
▶ ip netns
b3165c10a92b
```

```
▶ ip -n b3165c10a92b addr
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```



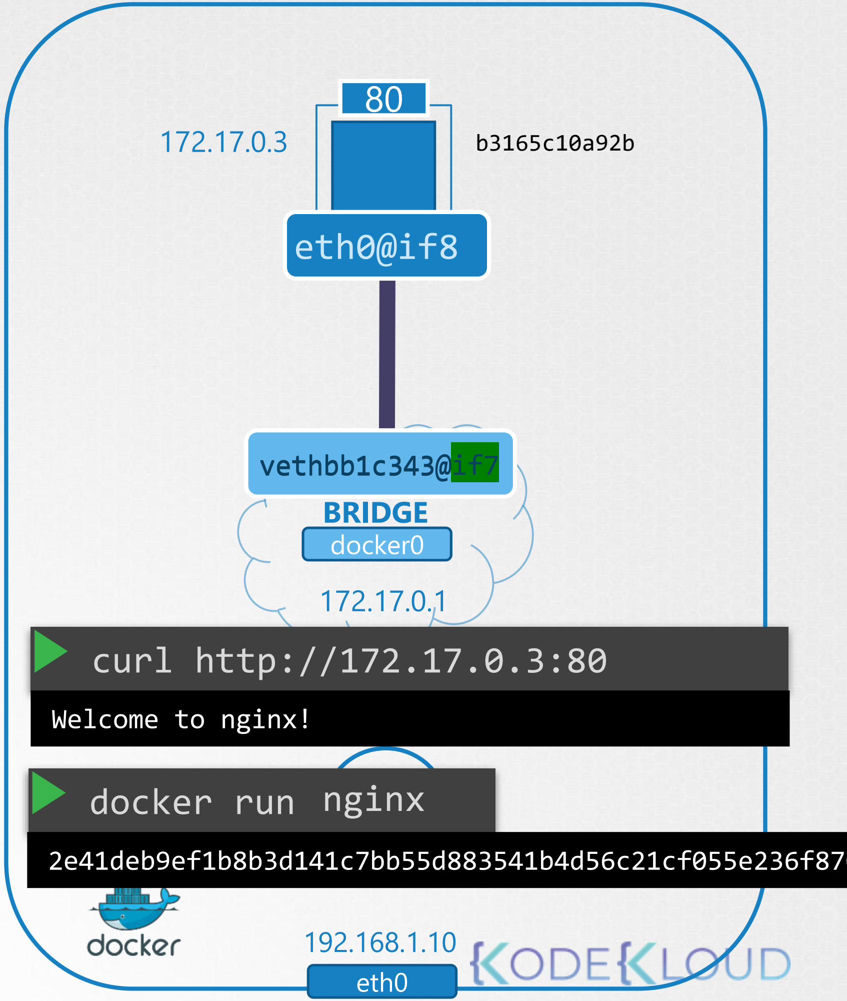
```
▶ docker run nginx
2e41deb9ef1b8b3d141c7bb55
```

BRIDGE



KODEKLOUD

BRIDGE

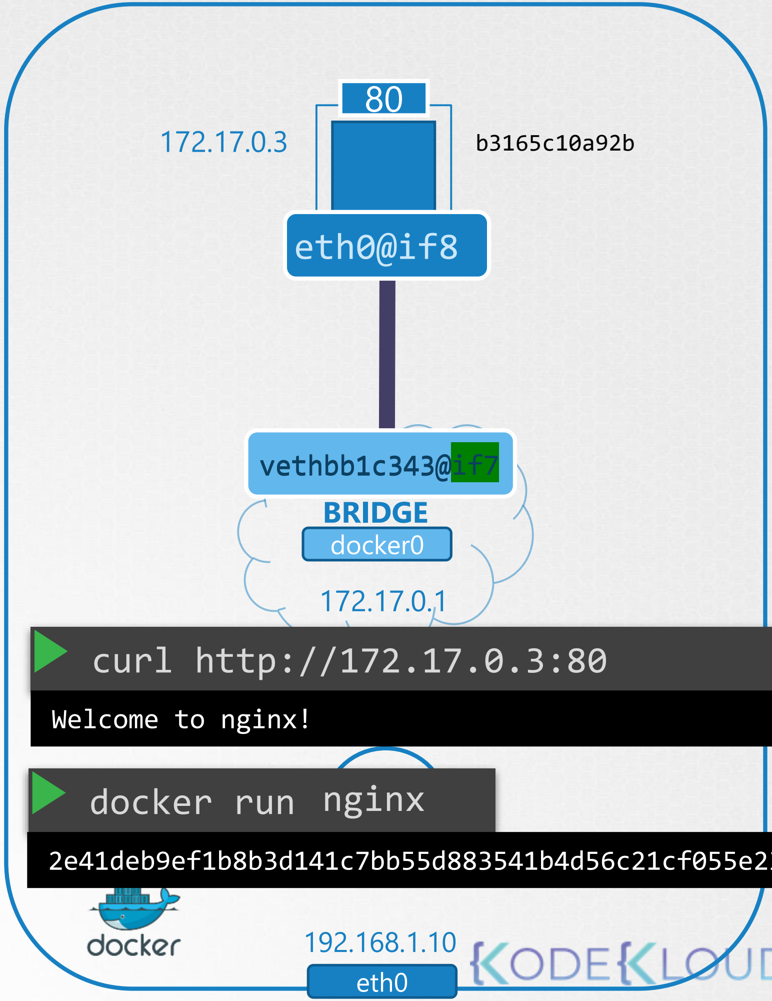


```
▶ curl http://172.17.0.3:80
Welcome to nginx!
```

```
▶ docker run nginx
2e41deb9ef1b8b3d141c7bb55d883541b4d56c21cf055e236f87
```

BRIDGE

```
curl http://172.17.0.3:80
curl: (7) Failed to connect... No route to host
```



```
curl http://172.17.0.3:80
Welcome to nginx!
```

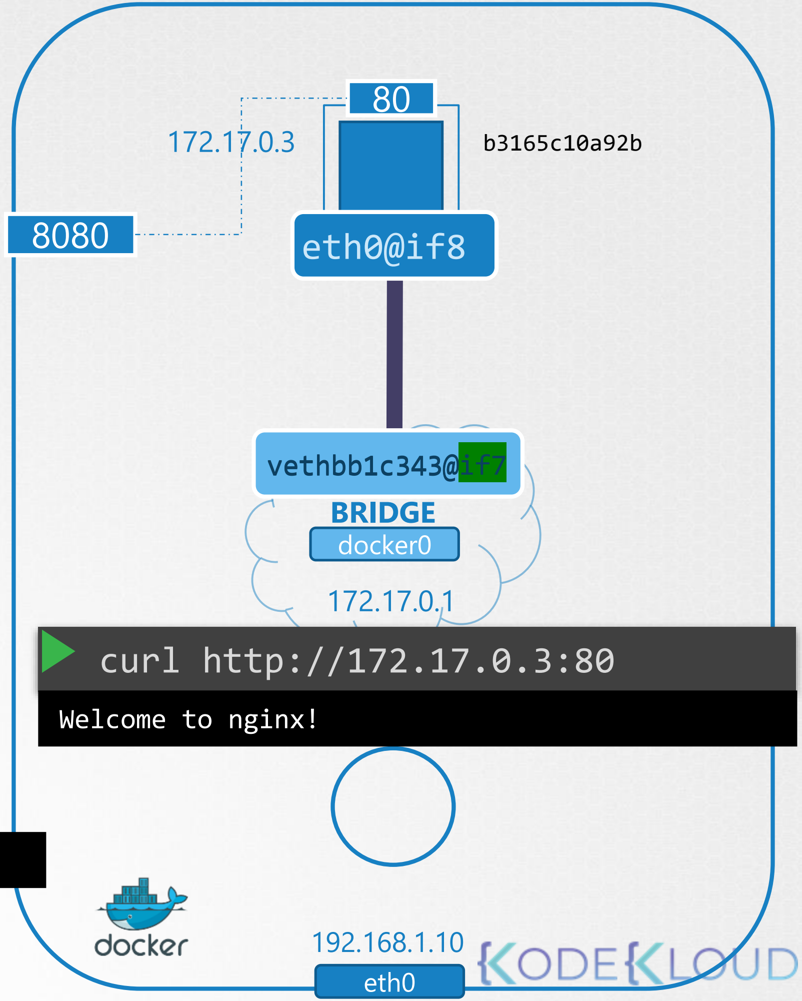
```
docker run nginx
2e41deb9ef1b8b3d141c7bb55d883541b4d56c21cf055e236f87
```

BRIDGE

```
▶ curl http://172.17.0.3:80
curl: (7) Failed to connect... No route to host
```

```
▶ curl http://192.168.1.10:8080
Welcome to nginx!
```

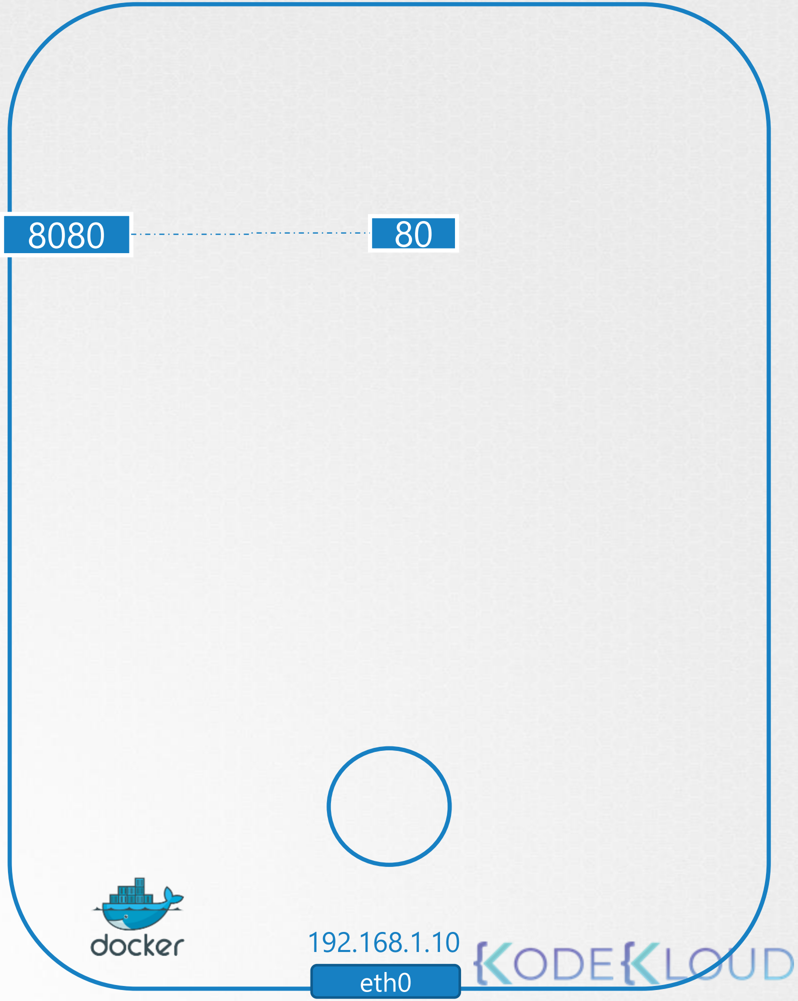
```
▶ docker run  ngi8080:80
2e41deb9ef1b8b3d141c7bb55d883541b4d56c21cf055e236f870bd0f274e52b
```



```
▶ curl http://172.17.0.3:80
Welcome to nginx!
```

BRIDGE

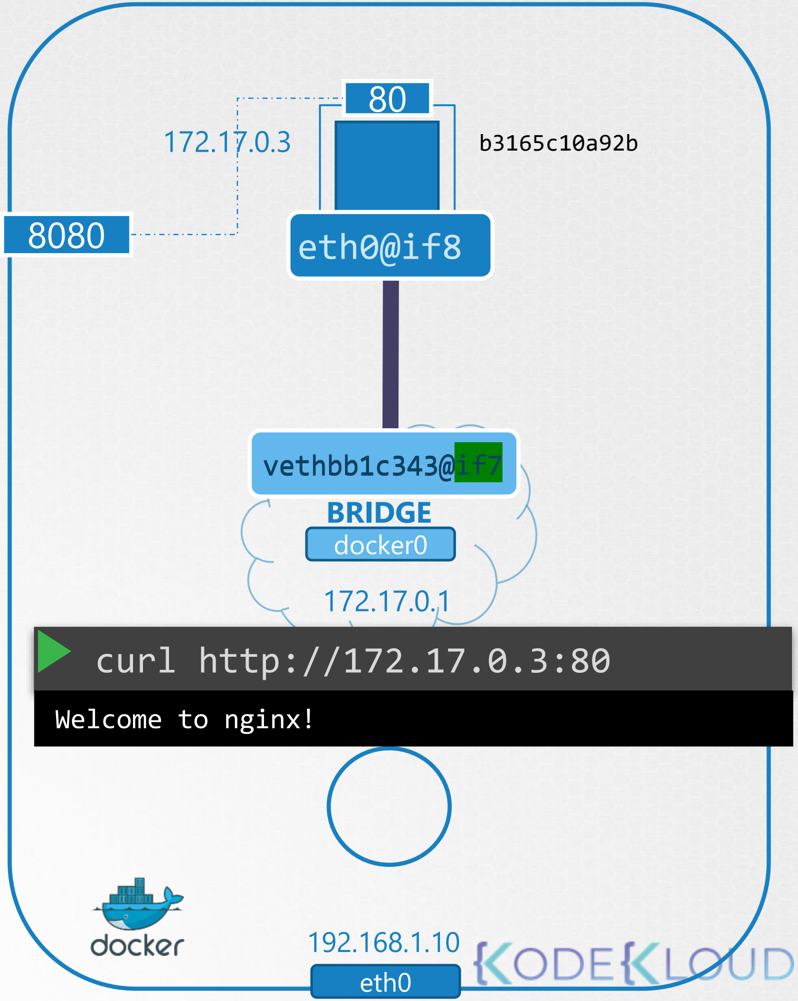
```
iptables \
        -t nat \
        -A PREROUTING \
        -j DNAT \
        --dport 8080 \
        --to-destination 80
```



BRIDGE

```
iptables \
      -t nat \
      -A PREROUTING \
      -j DNAT \
      --dport 8080 \
      --to-destination 80
```

```
iptables \
      -t nat \
      -A DOCKER \
      -j DNAT \
      --dport 8080 \
      --to-destination 172.17.0.3:80
```

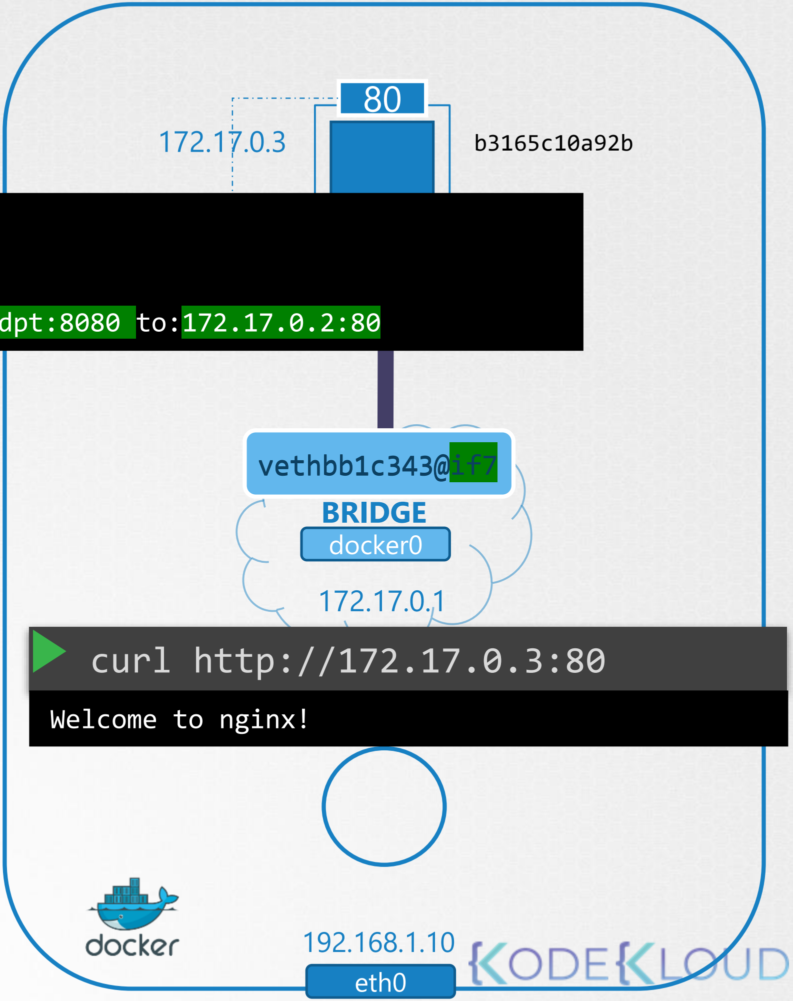


```
curl http://172.17.0.3:80
Welcome to nginx!
```

BRIDGE

```
▶ iptables -nvL -t nat
```

```
Chain DOCKER (2 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere
DNAT       tcp  --  anywhere             anywhere             tcp dpt:8080 to:172.17.0.2:80
```



```
▶ curl http://172.17.0.3:80
```

```
Welcome to nginx!
```

https://docs.docker.com/v17.09/engine/userguide/networking/default_network/container-communication/#communicating-to-the-outside-world

```
ln -s /var/run/docker/netns  /var/run
```

```
sudo route add 172.17.0.6/32 gateway 192.168.176.14 enp0s8

sudo ip neighbor add 172.17.0.6 lladdr 02:42:ac:11:00:06 dev enp0s8

sudo bridge fdb add 02:42:ac:11:00:06 dev enp0s8 self

sudo iptables -P FORWARD ACCEPT
```

KODEKLOUD

Container Networking Interface (CNI)

Network Namespaces

1. Create Network Namespace
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

docker

1. Create Network Namespace
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

rkt

1. Create Network Namespace
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

MESOS

1. Create Network Namespace
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

1. Create Network Namespace
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

Network Namespaces

docker

1. Create Network Namespace

rkt

1. Create Network Namespace

```
bridge add <cid> <namespace>
```

MESOS

1. Create Network Namespace

1. Create Network Namespace

```
bridge add <cid> <namespace>
```

1. Create Network Namespace

```
bridge add 2e34dcf34 /var/run/netns/2e34dcf34
```

BRIDGE

2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

Network Namespaces

1. Create Network Namespace

docker

1. Create Network Namespace

rkt

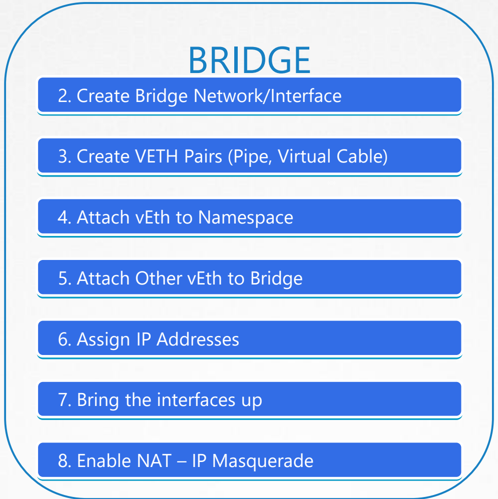
1. Create Network Namespace

MESOS

1. Create Network Namespace

1. Create Network Namespace

CONTAINER NETWORK INTERFACE



BRIDGE

2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

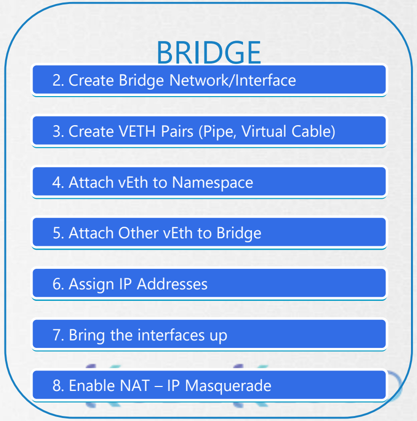
CONTAINER NETWORK INTERFACE

- ☑ Container Runtime must create network namespace
- ☑ Identify network the container must attach to
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is ADDed.
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is DELeted.
- ☑ JSON format of the Network Configuration

rkt

MESOS

- ☑ Must support command line arguments ADD/DEL/CHECK
- ☑ Must support parameters container id, network ns etc..
- ☑ Must manage IP Address assignment to PODs
- ☑ Must Return results in a specific format



BRIDGE

2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

CONTAINER NETWORK INTERFACE

✗
```
docker run --network=cni-bridge nginx
```

▶
```
docker run --network=none nginx
```

▶
```
bridge add 2e34dcf34 /var/run/netns/2e34dcf34
```

docker

rkt

MESOS

CONTAINER NETWORK MODEL (CNM)

BRIDGE

VLAN

IPVLAN

MACVLAN

WINDOWS

DHCP

host-local

weaveworks

flannel

cilium

vmware NSX

CONTAINER NETWORK INTERFACE

```
docker run --network=none nginx
```

```
bridge add 2e34dcf34 /var/run/netns/2e34dcf34
```

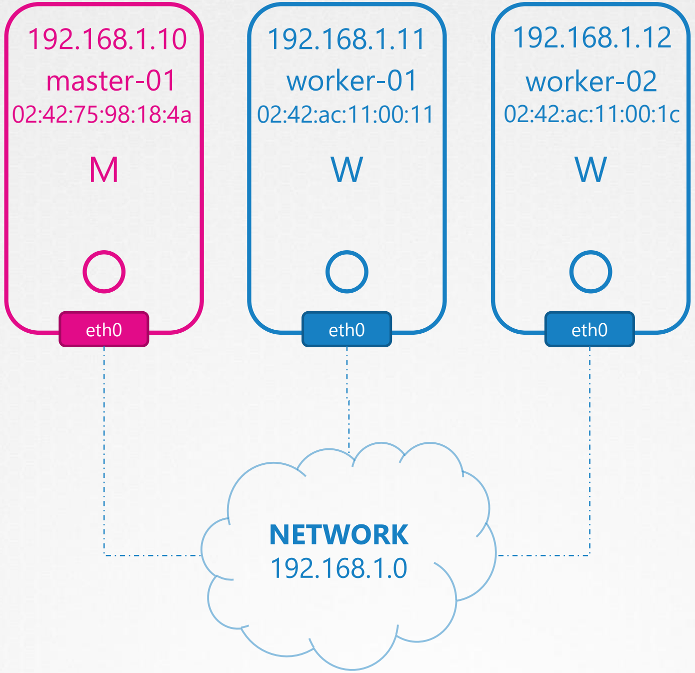
KODEKLOUD

Course Objectives

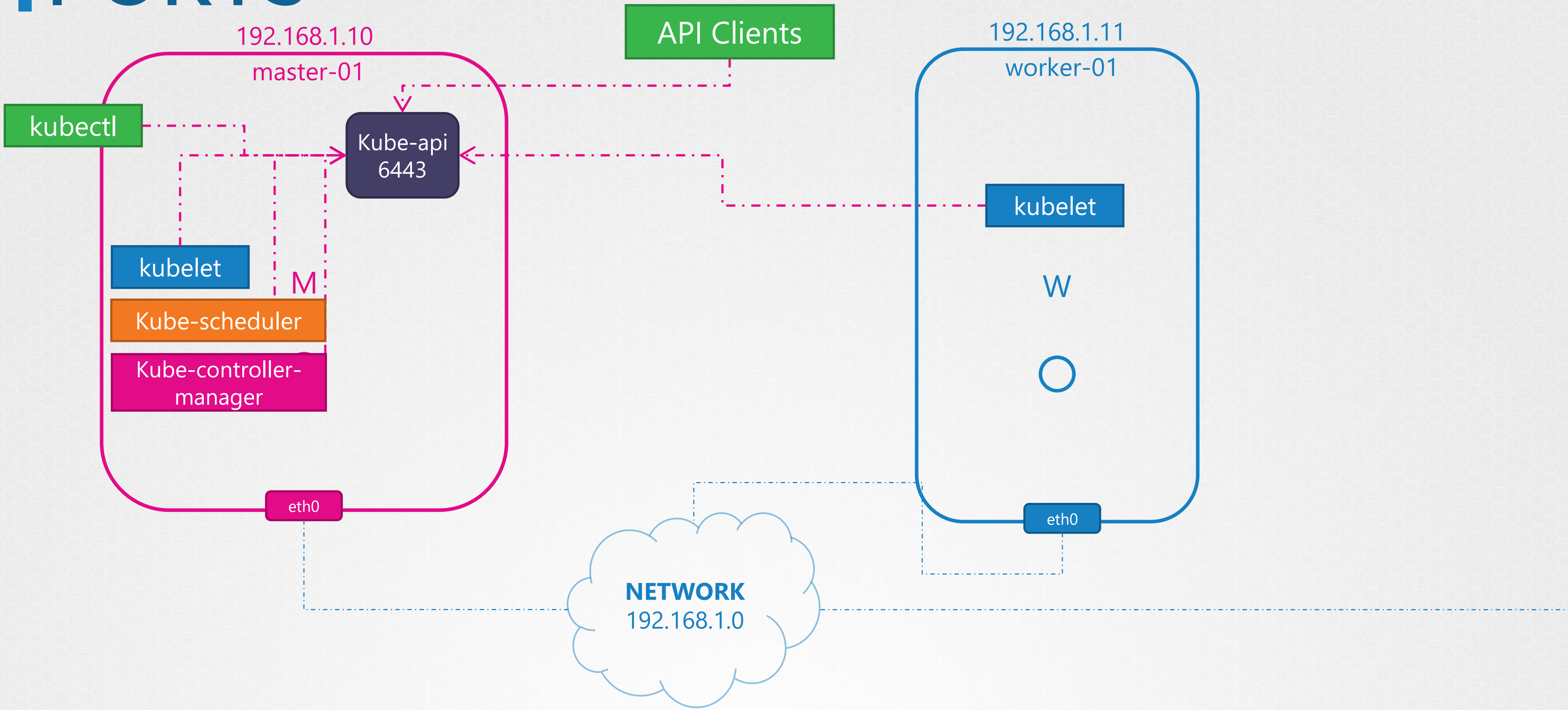
- ☑ Scheduling
- ☑ Logging Monitoring
- ☑ Application Lifecycle Management
- ☑ Cluster Maintenance
- ☑ Security
- ☑ Storage
- ☑ Troubleshooting
- ☑ Core Concepts
- ● Networking
  - ☑ Pre-Requisites – Switching, Routing
  - ☑ Pre-Requisites – Tools
  - ☑ Pre-Requisites CNI
  - ☑ Pre-Requisites – DNS
  - ☑ Pre-Requisites – Networking in Docker
  - ● Networking Configuration on Cluster Nodes
  - ○ POD Networking Concepts
  - ○ CNI in Kubernetes
  - ○ Cluster DNS
  - ○ Service Networking
  - ○ Network Loadbalancer
  - ○ Ingress

Networking Cluster Nodes

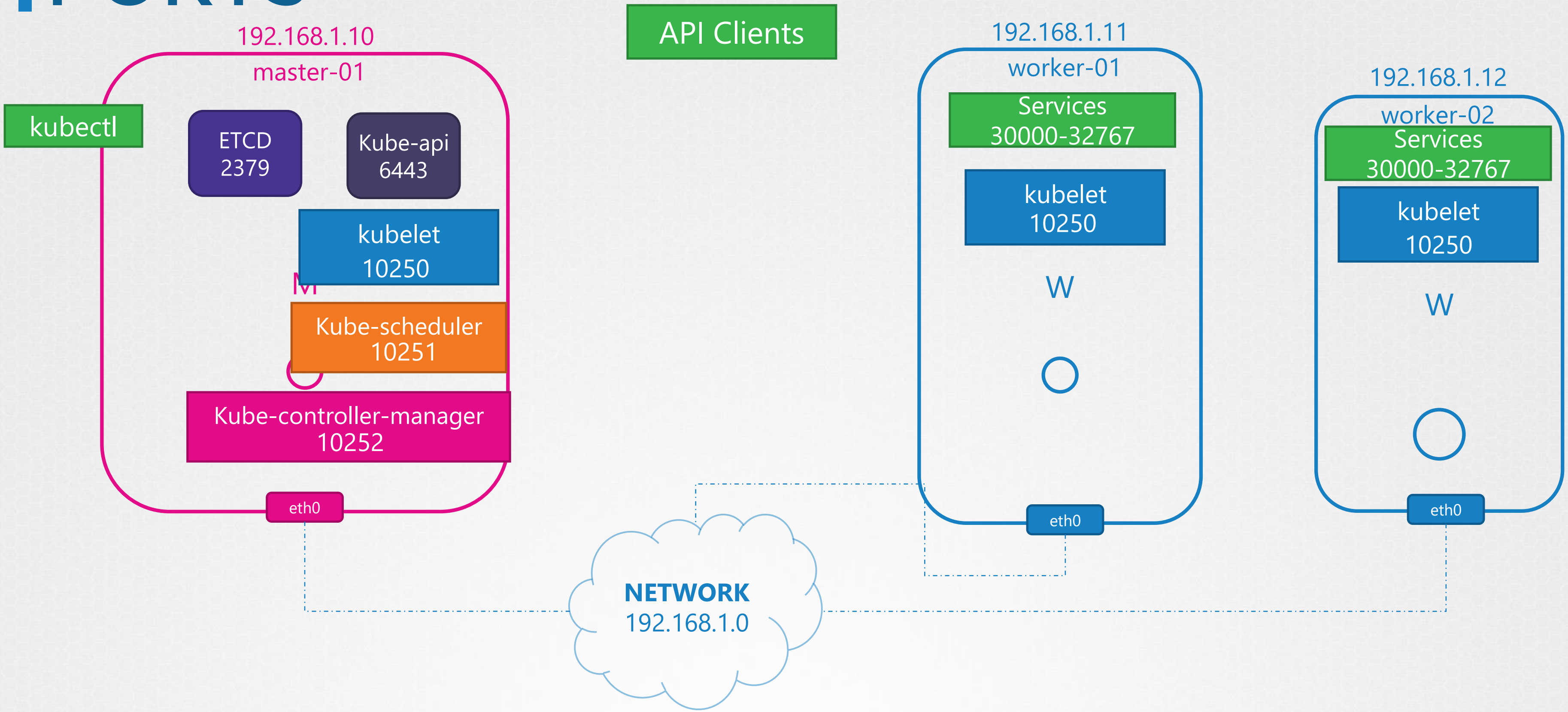
IP & FQDN


192.168.1.10
master-01
02:42:75:98:18:4a
M
eth0
192.168.1.11
worker-01
02:42:ac:11:00:11
W
eth0
192.168.1.12
worker-02
02:42:ac:11:00:1c
W
eth0
NETWORK
192.168.1.0

PORTS

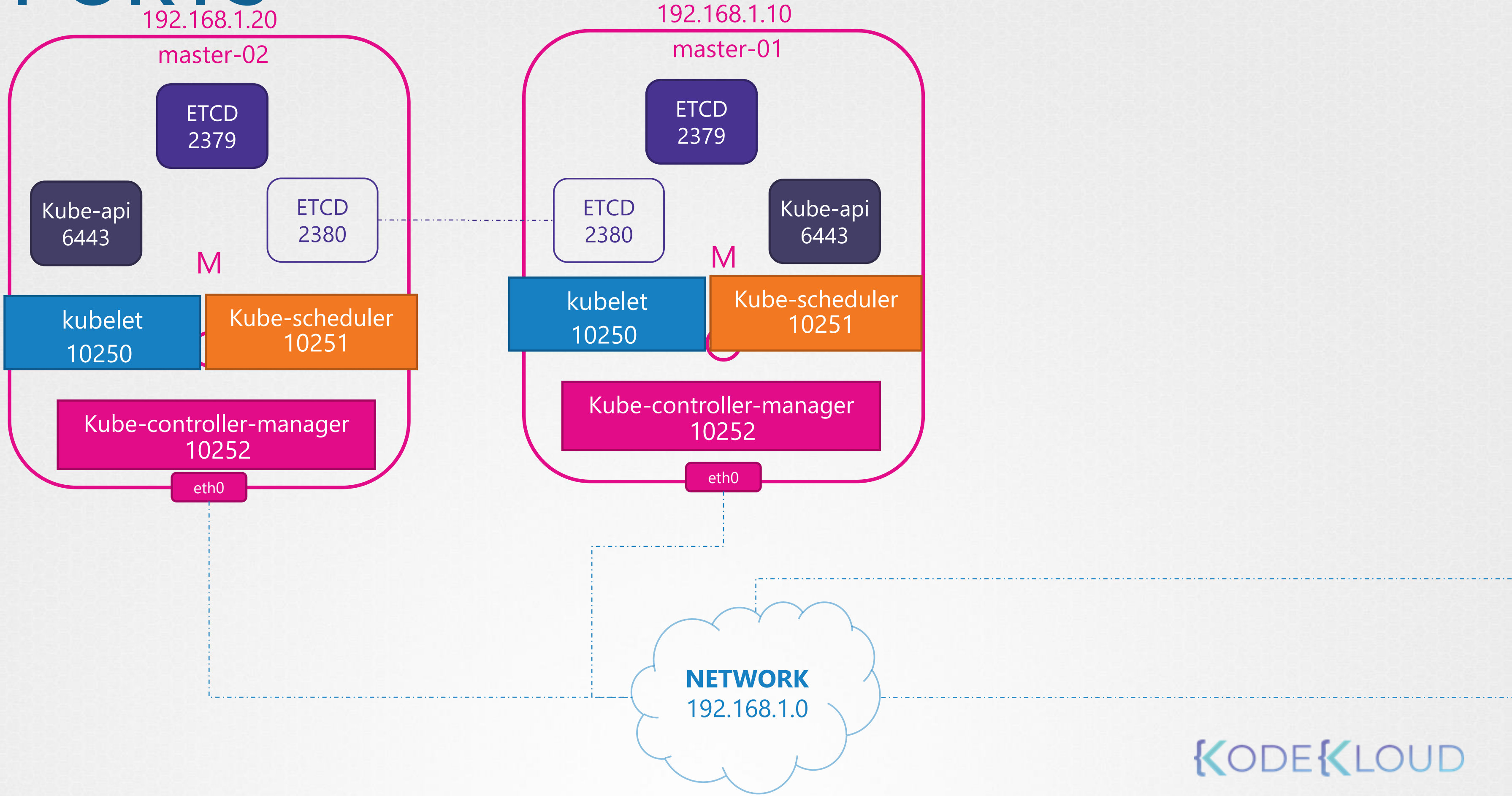

192.168.1.10
master-01
API Clients
192.168.1.11
worker-01
kubectl
Kube-api
6443
kubelet
kubelet
M
W
Kube-scheduler
Kube-controller-manager
eth0
eth0
NETWORK
192.168.1.0

PORTS



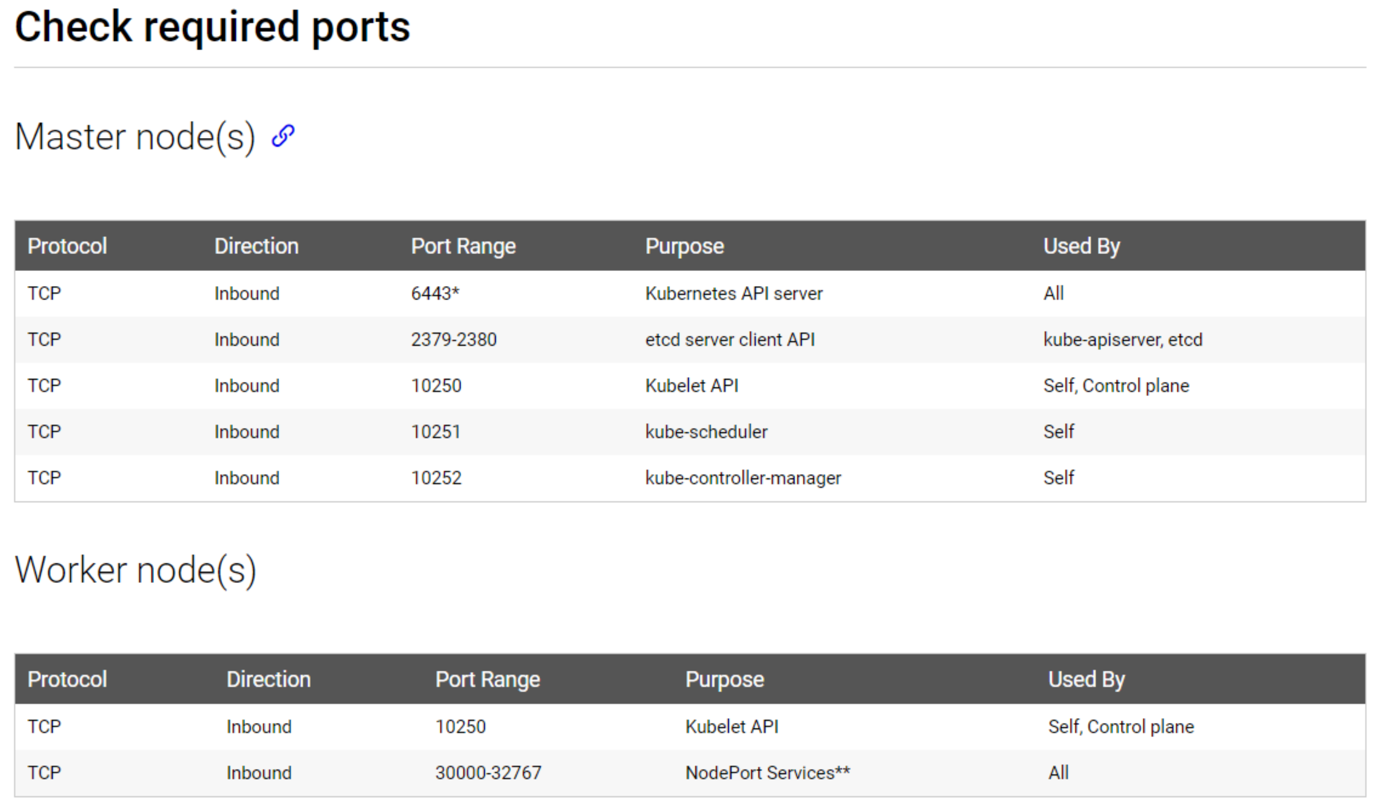
192.168.1.10
master-01
kubectl
ETCD
2379
Kube-api
6443
kubelet
10250
M
Kube-scheduler
10251
Kube-controller-manager
10252
eth0
API Clients
192.168.1.11
worker-01
Services
30000-32767
kubelet
10250
W
eth0
192.168.1.12
worker-02
Services
30000-32767
kubelet
10250
W
eth0
NETWORK
192.168.1.0

PORTS



KODEKLOUD

Documentation



Check required ports

Master node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443*	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10251	kube-scheduler	Self
TCP	Inbound	10252	kube-controller-manager	Self

Worker node(s)

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	30000-32767	NodePort Services**	All

https://kubernetes.io/docs/setup/independent/install-kubeadm/#check-required-ports

COMMMANDS

```
▶ ip link
```

```
▶ ip addr
```

```
▶ ip addr add 192.168.1.10/24 dev eth0
```

```
▶ ip route
```

```
▶ ip route add 192.168.1.0/24 via 192.168.2.1
```

```
▶ route
```

```
▶ cat /proc/sys/net/ipv4/ip_forward
1
```

```
▶ arp
```

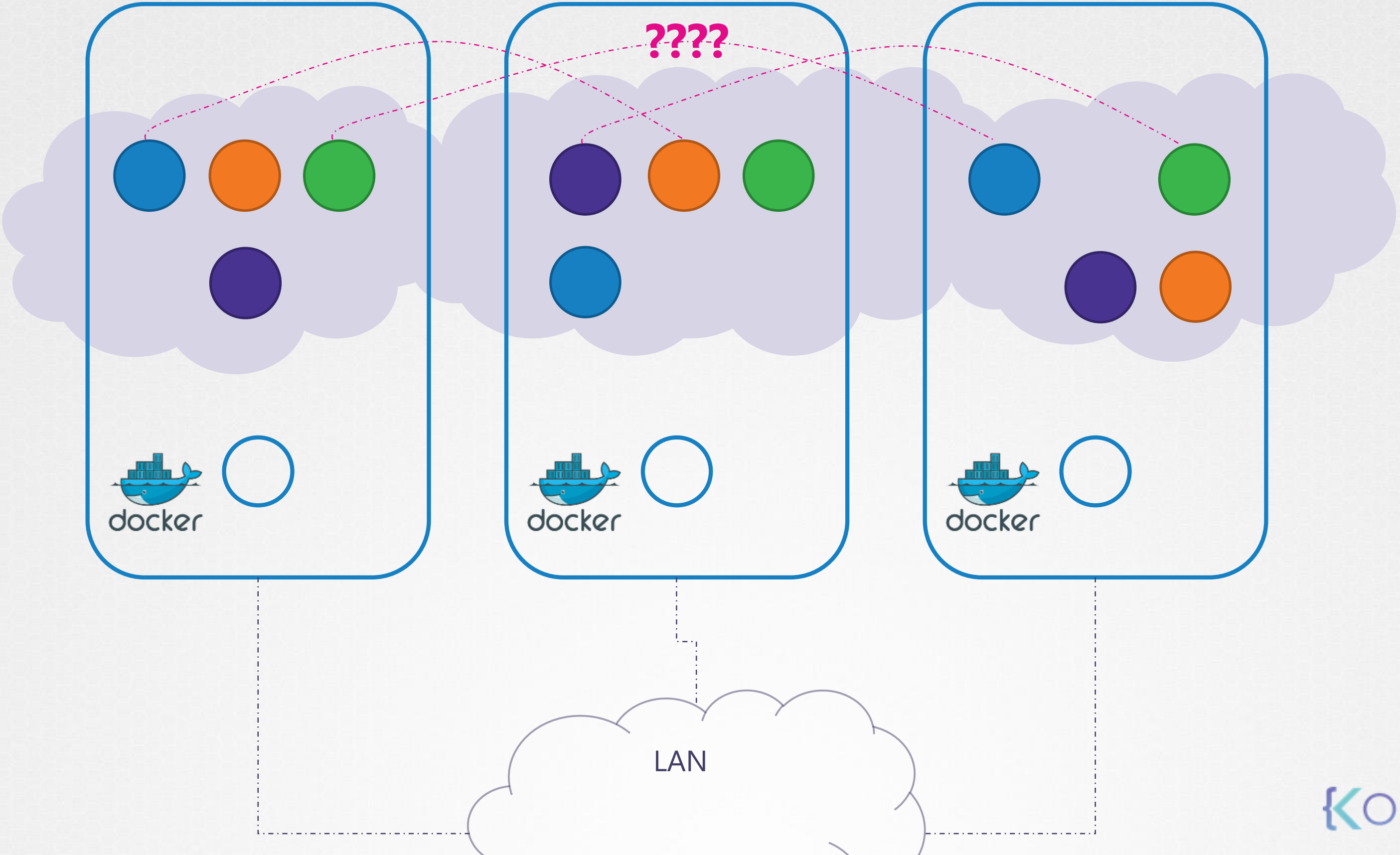
```
▶ netstat -plnt
```

KODEKLOUD

Course Objectives

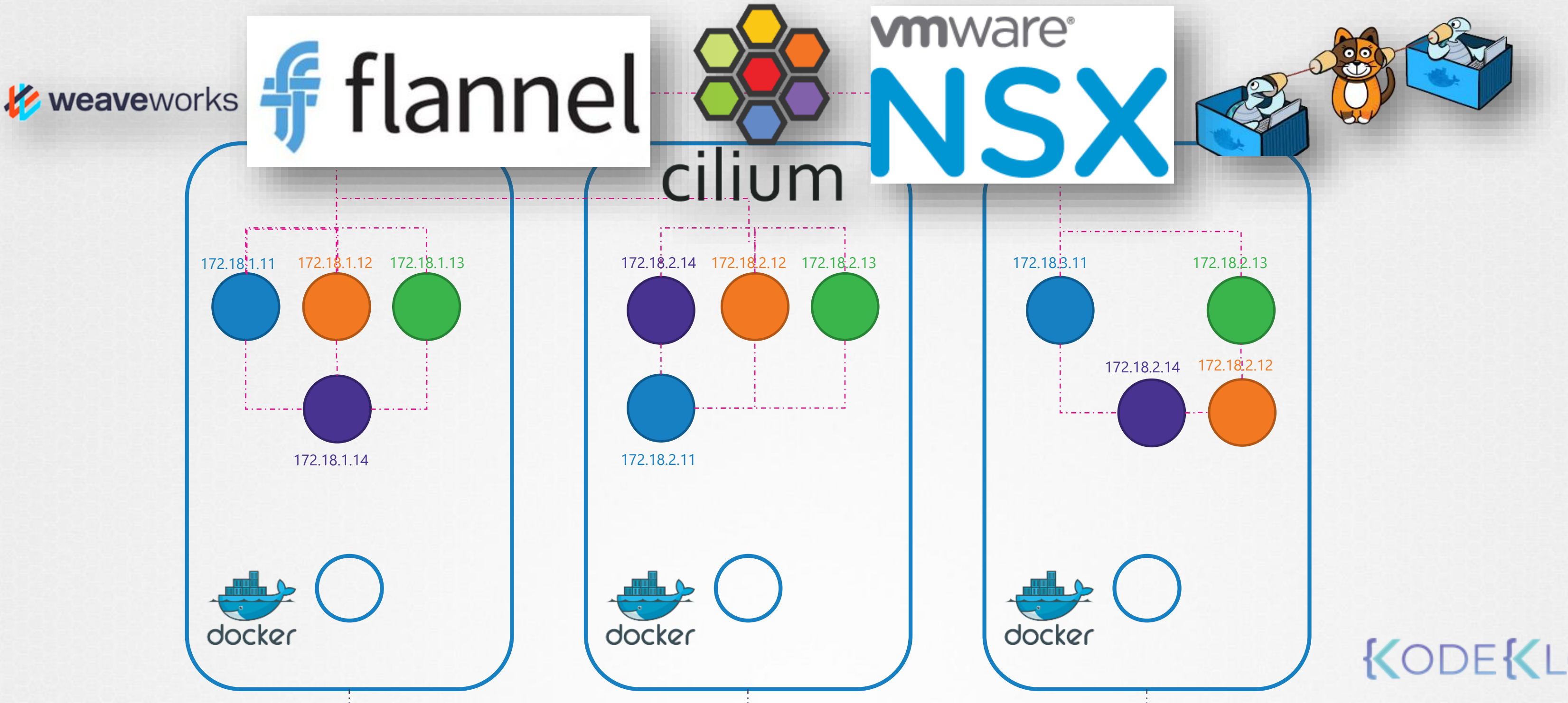
- ✓ Scheduling
- ✓ Logging Monitoring
- ✓ Application Lifecycle Management
- ✓ Cluster Maintenance
- ✓ Security
- ✓ Storage
- ✓ Troubleshooting
- ✓ Core Concepts
- ● Networking
  - ✓ Pre-Requisites – Switching, Routing
  - ✓ Pre-Requisites – Tools
  - ✓ Pre-Requisites CNI
  - ✓ Pre-Requisites – DNS
  - ✓ Pre-Requisites – Networking in Docker
  - ✓ Networking Configuration on Cluster Nodes
  - ● POD Networking Concepts
  - ○ CNI in Kubernetes
  - ○ Cluster DNS
  - ○ Service Networking
  - ○ Network Loadbalancer
  - ○ Ingress

POD Networking Concepts


????
docker
docker
docker
LAN

Networking Model

- ❑ Every POD should have an IP Address
- ❑ Every POD should be able to communicate with every other POD in the same node.
- ❑ Every POD should be able to communicate with every other POD on other nodes without NAT.



Networking Model

```
ip link add v-net-0 type bridge
```

```
ip link set dev v-net-0 up
```

- ❑ Every POD should have an IP Address
- ❑ Every POD shou ame node.
- ❑ Every POD should be able to communicate with every other POD on other nodes without NAT.

```
ip addr add 192.168.15.5/24 dev v-net-0
```

```
ip link add veth-red type veth peer name veth-red-br
```

```
ip link set veth-red netns red
```

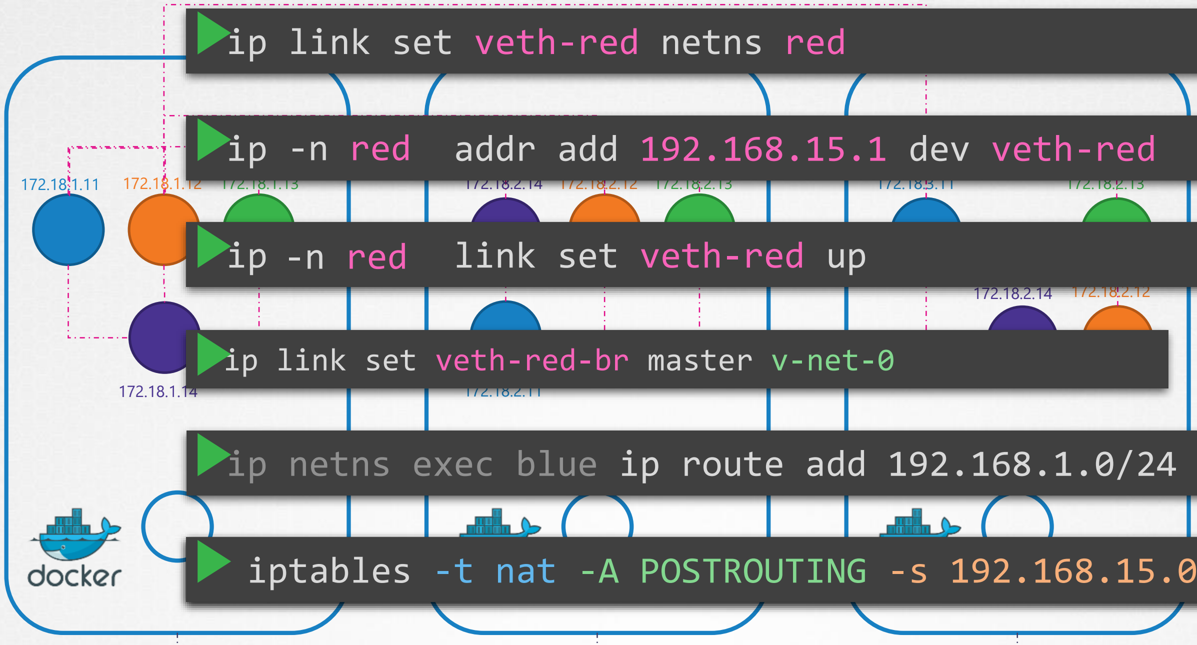
```
ip -n red  addr add 192.168.15.1 dev veth-red
```

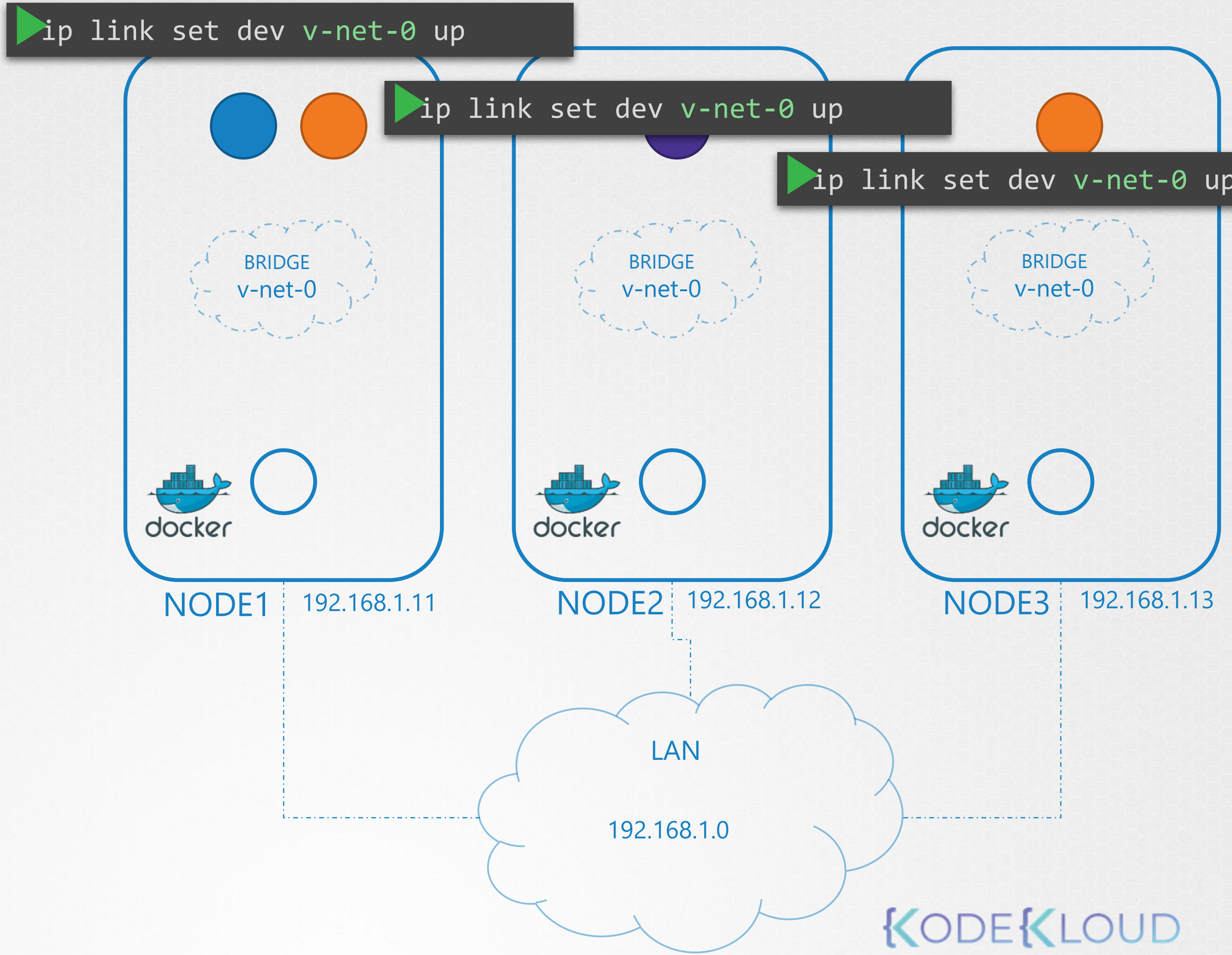
```
ip -n red  link set veth-red up
```

```
ip link set veth-red-br master v-net-0
```

```
ip netns exec blue ip route add 192.168.1.0/24 via 192.168.15.5
```

```
iptables -t nat -A POSTROUTING -s 192.168.15.0/24 -j MASQUERADE
```





```
ip link set dev v-net-0 up
```

```
ip link set dev v-net-0 up
```

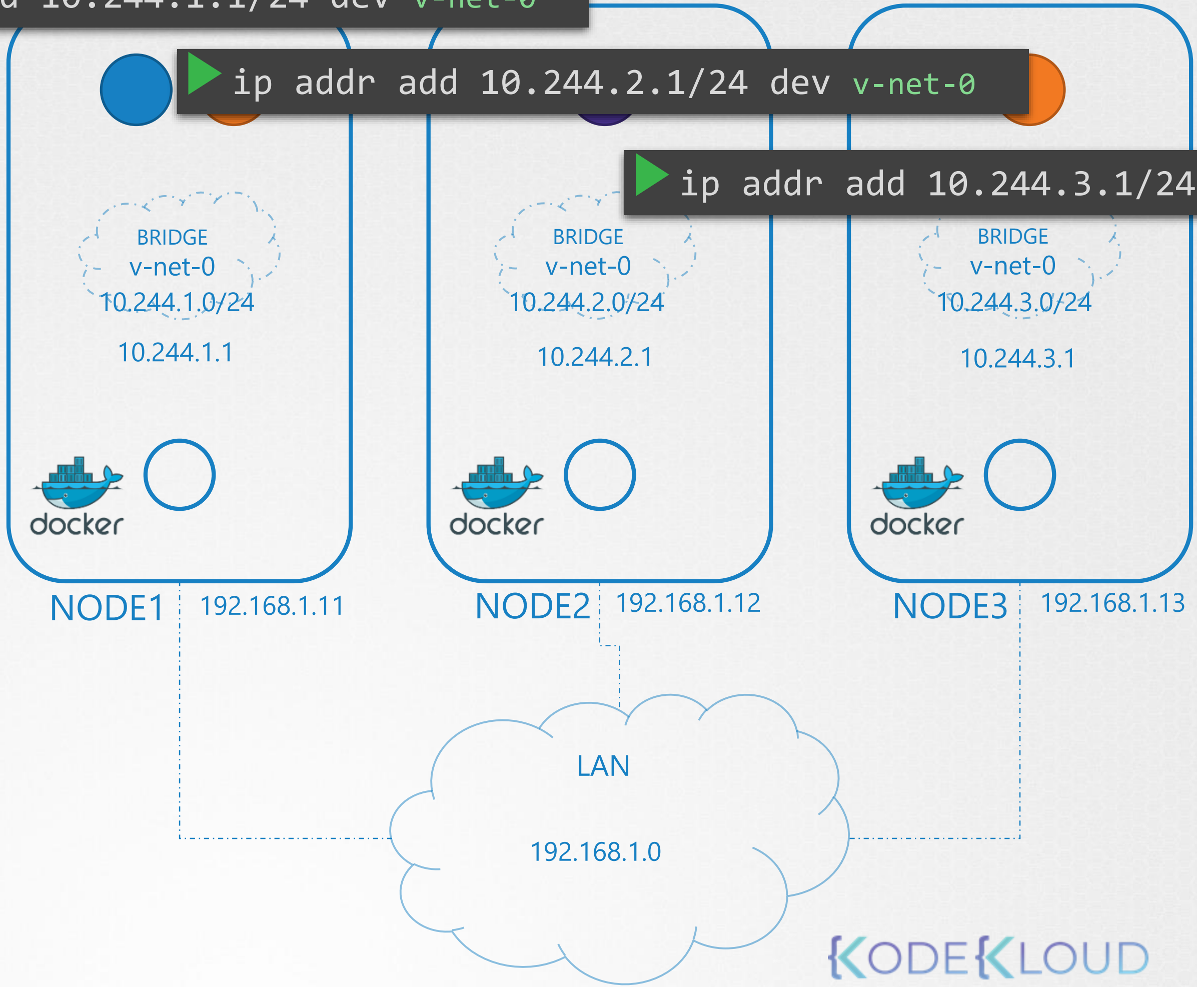
```
ip link set dev v-net-0 up
```

KODEKLOUD

```
ip addr add 10.244.1.1/24 dev v-net-0
```

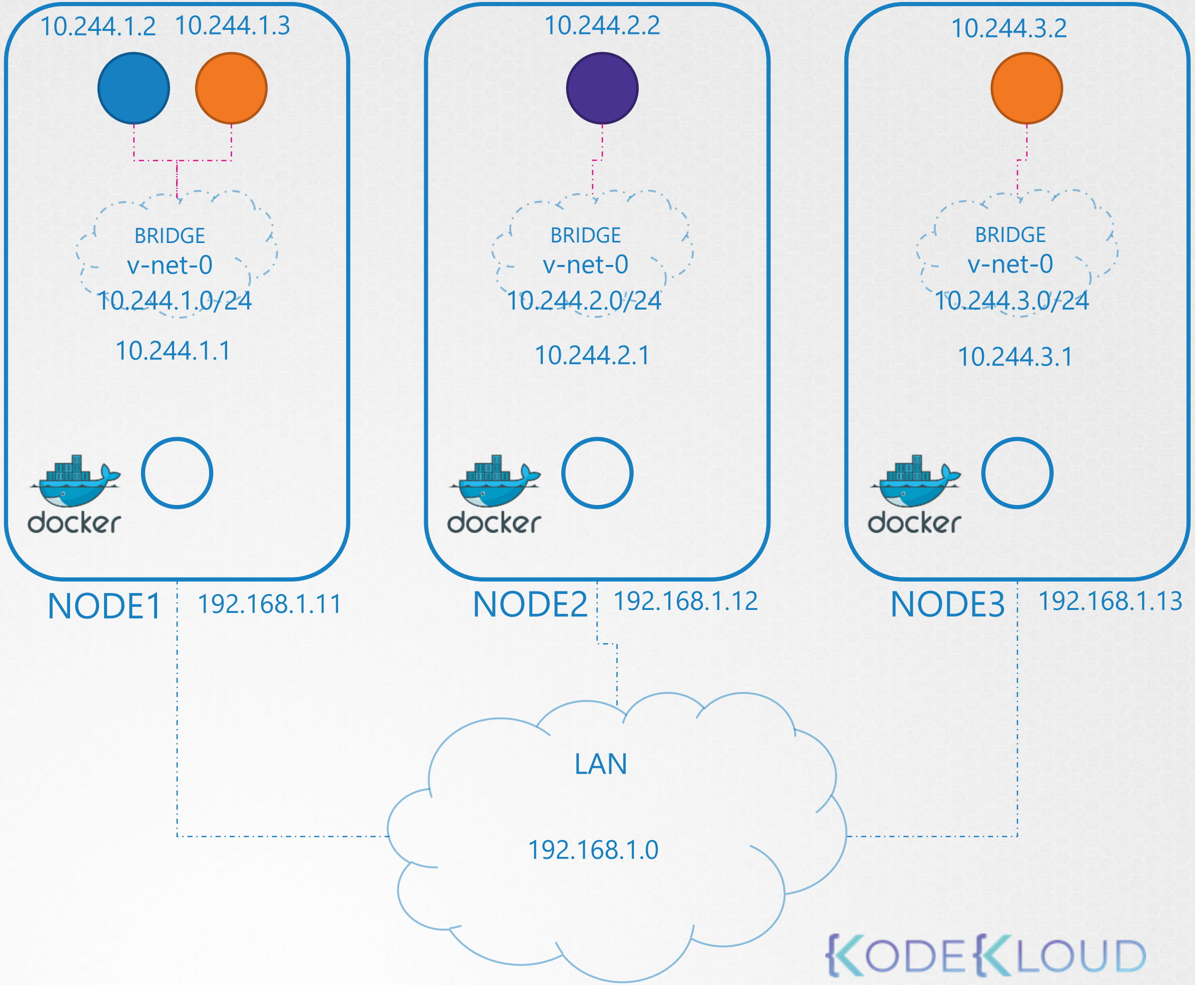
```
ip addr add 10.244.2.1/24 dev v-net-0
```

```
ip addr add 10.244.3.1/24
```



net-script.sh

```
# Create veth pair
ip link add ......

# Attach veth pair
ip link set ......

ip link set ......

 # Assign IP Address
 ip -n <namespace> addr add ......
 ip -n <namespace> route add ......

 # Bring Up Interface
 ip -n <namespace> link set ......
```



KODEKLOUD

```
bluepod$ ping 10.244.2.2
```

```
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=0.587 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=0.466 ms
```

```
node3$ ip route add 10.244.1.2 via 192.168.1.11
```

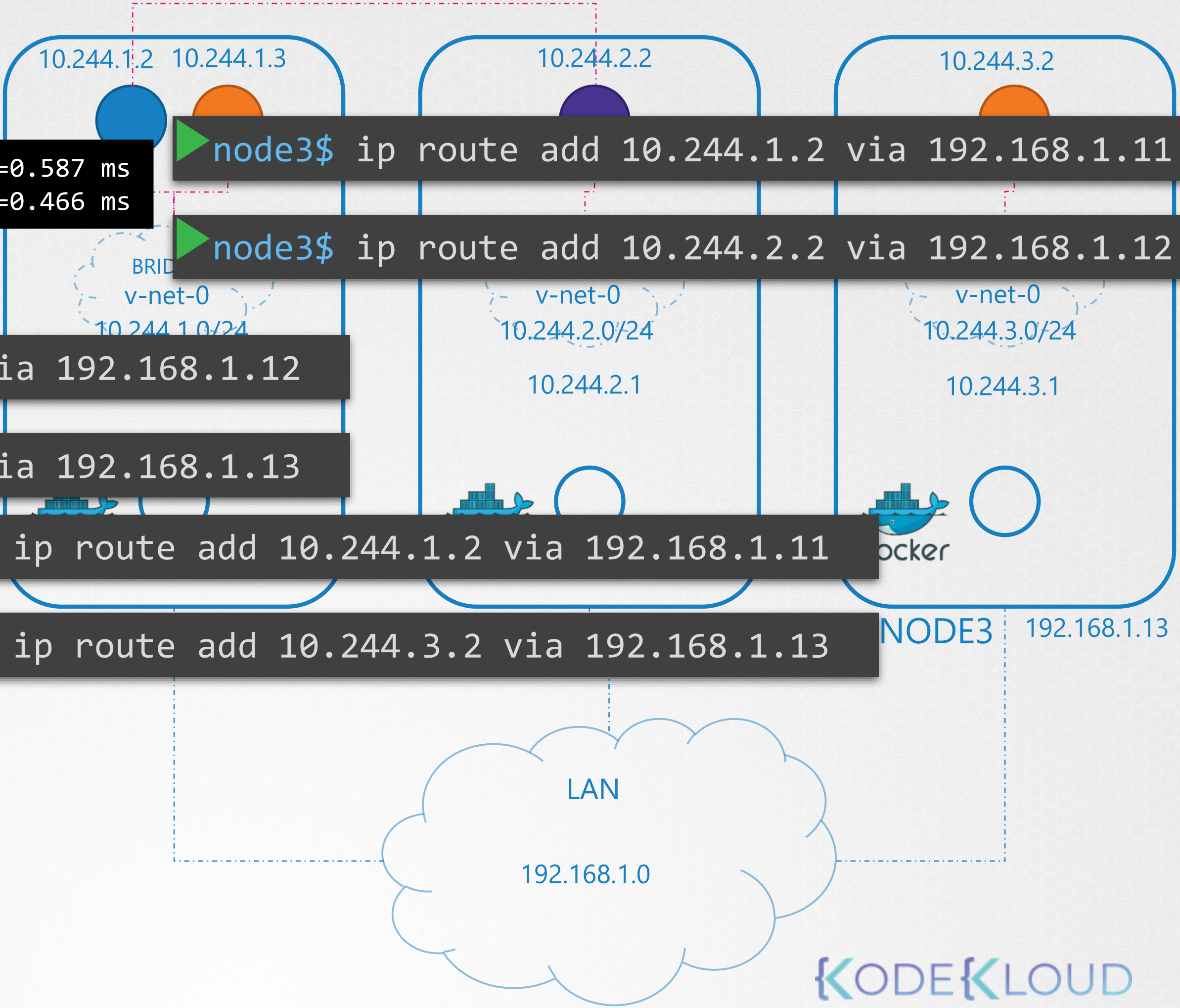
```
node3$ ip route add 10.244.2.2 via 192.168.1.12
```

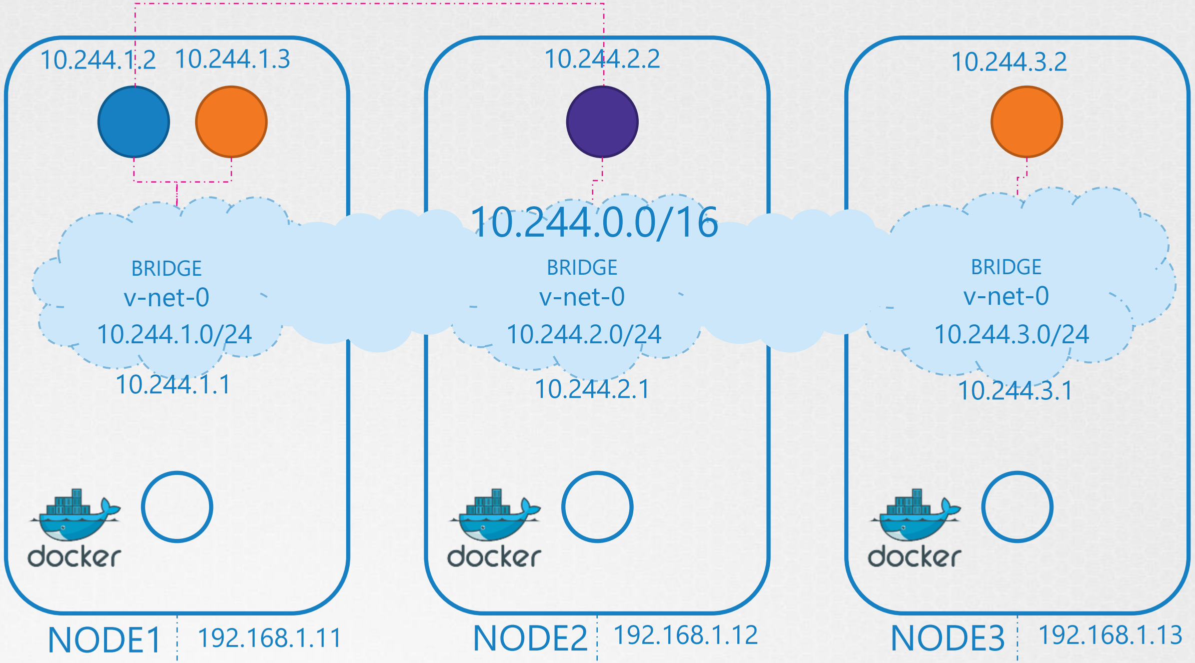
```
node1$ ip route add 10.244.2.2 via 192.168.1.12
```

```
node1$ ip route add 10.244.3.2 via 192.168.1.13
```

```
node2$ ip route add 10.244.1.2 via 192.168.1.11
```

```
node2$ ip route add 10.244.3.2 via 192.168.1.13
```



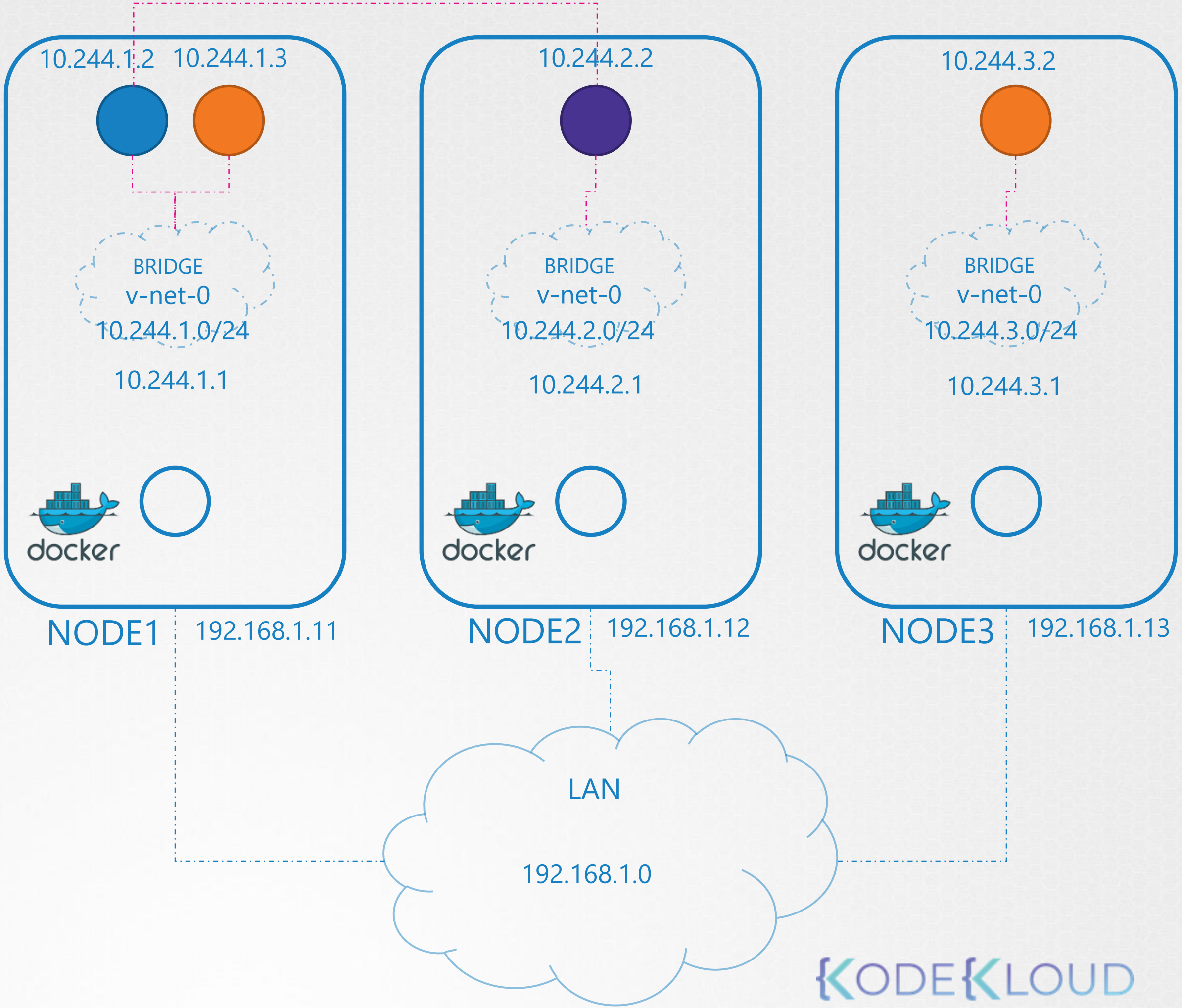


NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12
10.244.3.0/24	192.168.1.13

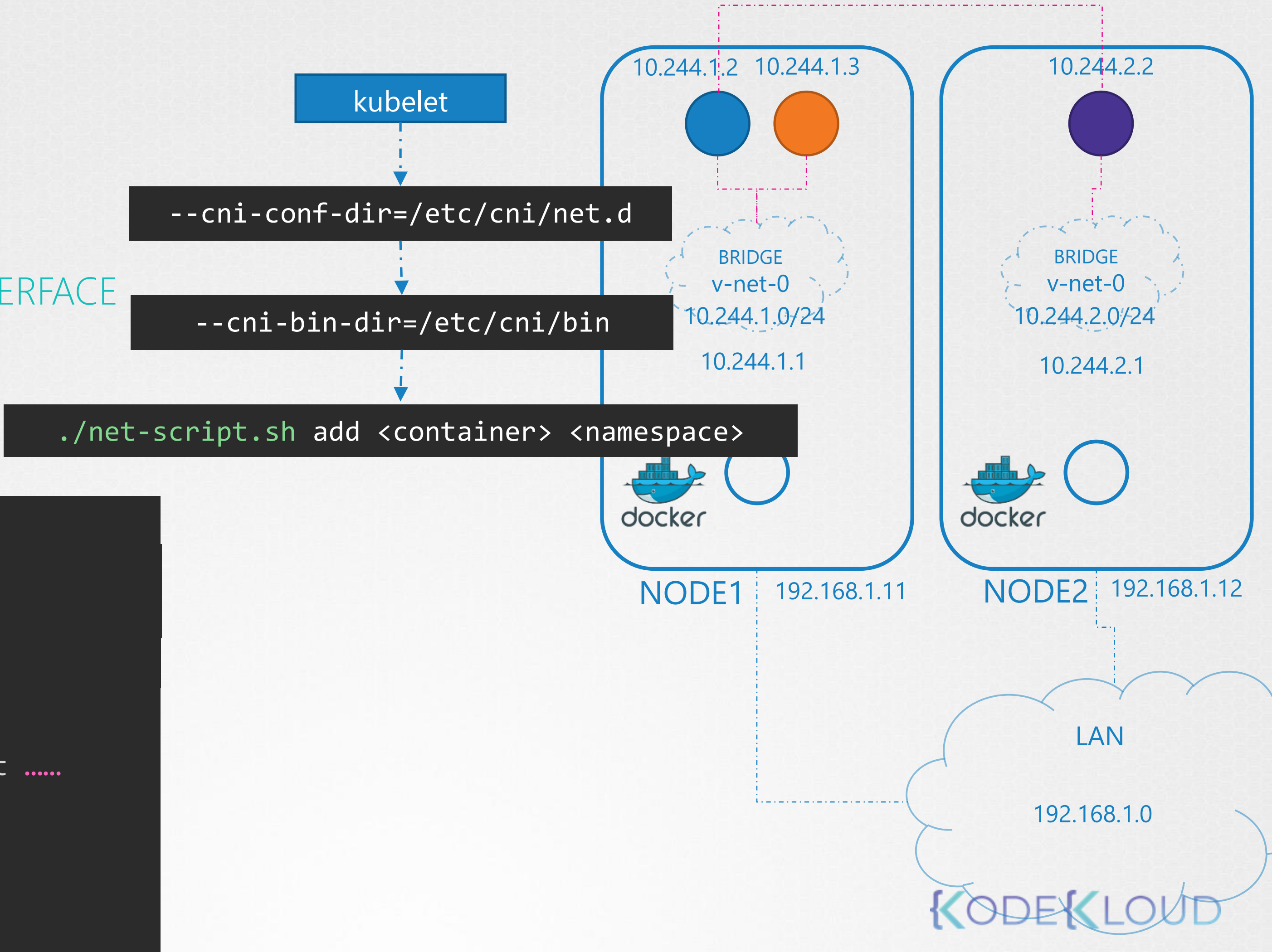
CONTAINER NETWORK INTERFACE (CNI)

net-script.sh

```
# Create veth pair
ip link add ......

# Attach veth pair
ip link set ......

ip link set ......

 # Assign IP Address
 ip -n <namespace> addr add ......
 ip -n <namespace> route add ......
```



CONTAINER NETWORK INTERFACE (CNI)



kubelet

```
--cni-conf-dir=/etc/cni/net.d
--cni-bin-dir=/etc/cni/bin
./net-script.sh add <container> <namespace>
```

net-script.sh

```
ADD)
 # Create veth pair
 # Attach veth pair
 # Assign IP Address
 # Bring Up Interface
 ip -n <namespace> link set ......
DEL)
 # Delete veth pair
 ip link del ......
```

KODEKLOUD

Course Objectives

- ✓ Scheduling
- ✓ Logging Monitoring
- ✓ Application Lifecycle Management
- ✓ Cluster Maintenance
- ✓ Security
- ✓ Storage
- ✓ Troubleshooting
- ✓ Core Concepts
- ● Networking
  - ✓ Pre-Requisites – Switching, Routing
  - ✓ Pre-Requisites – DNS
  - ✓ Networking Configuration on Cluster Nodes
  - ○ Cluster DNS
  - ○ Service Networking
  - ✓ Pre-Requisites – Tools
  - ✓ Pre-Requisites – Networking in Docker
  - ✓ POD Networking Concepts
  - ○ Network Loadbalancer
  - ✓ Pre-Requisites CNI
  - CNI in Kubernetes
  - ○ Ingress

Container Networking Interface (CNI)

IN KUBERNETES

Pre-Requisites

- ✓ Network Namespaces in Linux
- ✓ Networking in Docker
- ✓ Why and what is Container Network Interface (CNI)?
- ✓ CNI Plugins

CONTAINER NETWORK INTERFACE

- ✓ Container Runtime must create network namespace
- ✓ Identify network the container must attach to
- ✓ Container Runtime to invoke Network Plugin (bridge) when container is ADDed.
- ✓ Container Runtime to invoke Network Plugin (bridge) when container is DELeted.
- ✓ JSON format of the Network Configuration

Configuring CNI

kubelet.service

```
ExecStart=/usr/local/bin/kubelet \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --container-runtime=remote \\
  --container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --network-plugin=cni \\
  --cni-bin-dir=/opt/cni/bin \\
  --cni-conf-dir=/etc/cni/net.d \\
  --register-node=true \\
  --v=2
```

View kubelet options

```
▶  ps -aux | grep kubelet
```

```
root      2095  1.8  2.4 960676 98788 ?       Ssl  02:32   0:36 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --cgroup-driver=cgroupfs --cni-bin-dir=/opt/cni/bin --cni-
conf-dir=/etc/cni/net.d --network-plugin=cni
```

```
▶  ls /opt/cni/bin
```

```
bridge  dhcp  flannel  host-local  ipvlan  loopback  macvlan  portmap  ptp  sample  tuning
vlan  weave-ipam  weave-net  weave-plugin-2.2.1
```

```
▶  ls /etc/cni/net.d
```

```
10-bridge.conf
```

View kubelet options

```
▶ ls /etc/cni/net.d
```

```
10-bridge.conf
```

```
▶ cat /etc/cni/net.d/10-bridge.conf
```

```
{
   "cniVersion": "0.2.0",
   "name": "mynet",
   "type": "bridge",
   "bridge": "cni0",
   "isGateway": true,
   "ipMasq": true,
   "ipam": {
      "type": "host-local",
      "subnet": "10.22.0.0/16",
      "routes": [
         { "dst": "0.0.0.0/0" }
      ]
   }
}
```

KODEKLOUD

Course Objectives

- ● Scheduling
- ● Logging Monitoring
- ● Application Lifecycle Management
- ● Cluster Maintenance
- ● Security
- ● Storage
- ● Troubleshooting
- ● Core Concepts
- ● Networking

○ Pre-Requisites – Switching, Routing

○ Pre-Requisites – DNS, IPAM, Firewalls, LBs

○ Networking Configuration on Cluster Nodes

○ Cluster DNS

○ Service Networking

○ Pre-Requisites – Tools

○ Pre-Requisites – Networking in Docker

○ POD Networking Concepts

○ Network Loadbalancer

○ Pre-Requisites CNI

○ CNI in Kubernetes

○ Ingress

WeaveWorks (CNI)

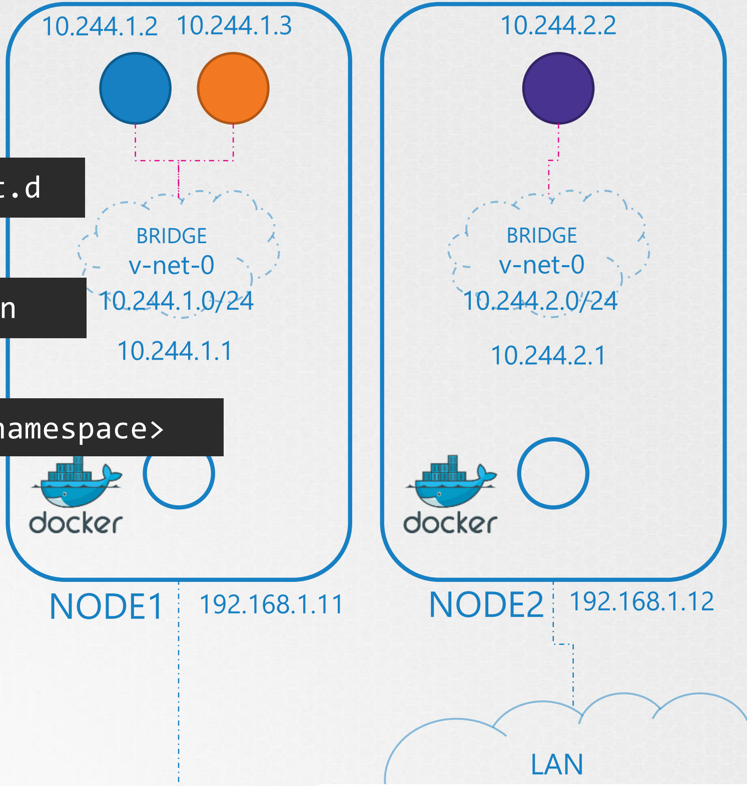
CONTAINER NETWORK INTERFACE (CNI)

kubelet

```
--cni-conf-dir=/etc/cni/net.d
```

```
--cni-bin-dir=/etc/cni/bin
```

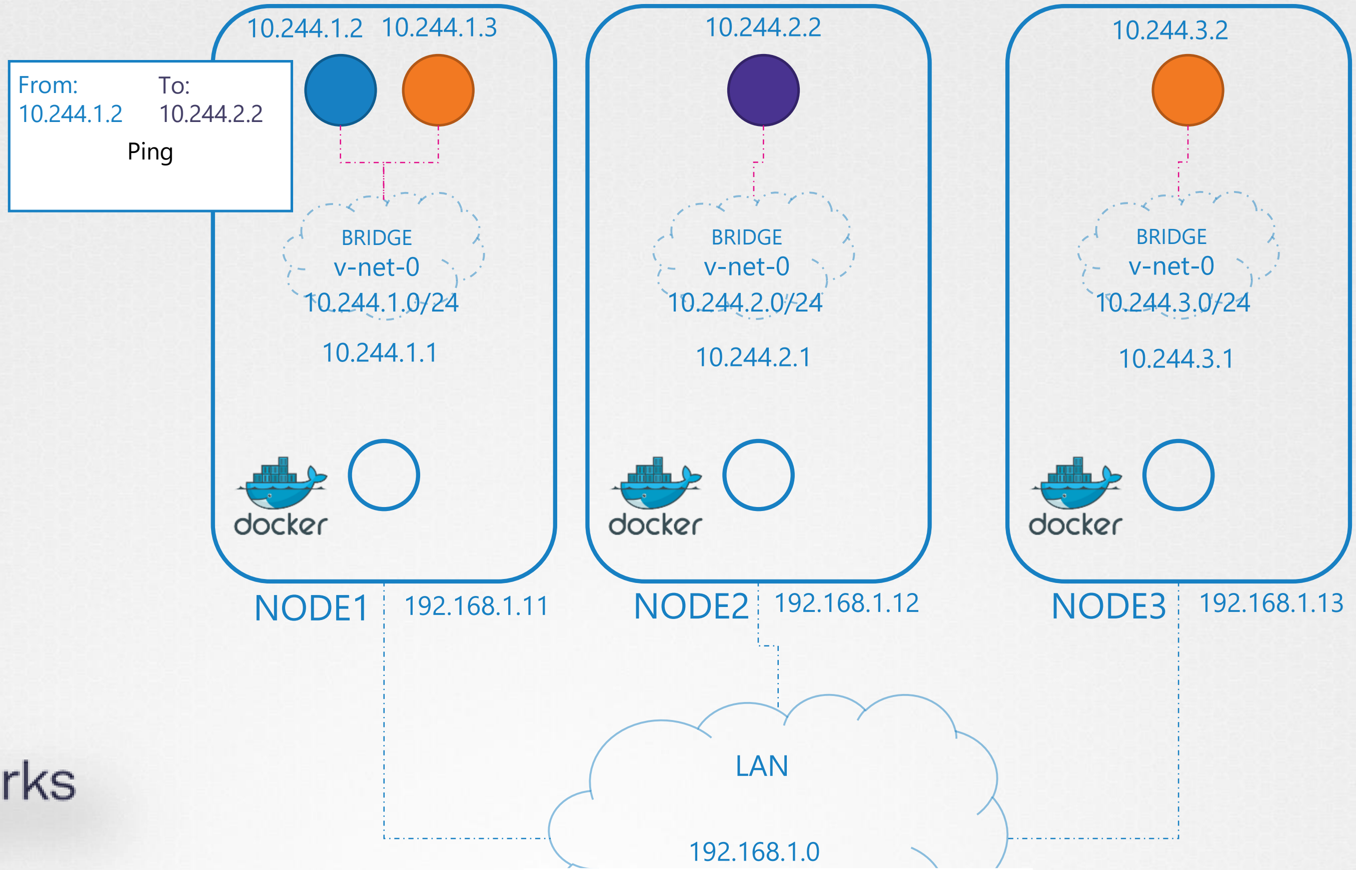
```
./net-script.sh add <container> <namespace>
```



net-script.sh

```
ADD)
  # Create veth pair

  # Attach veth pair

  # Assign IP Address

  # Bring Up Interface
  ip -n <namespace> link set ......
DEL)

  # Delete veth pair
  ip link del ......
```

NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12
10.244.3.0/24	192.168.1.13



From: 10.244.1.2

To: 10.244.2.2

Ping

10.244.1.2 10.244.1.3

BRIDGE
v-net-0
10.244.1.0/24

10.244.1.1

docker

NODE1 192.168.1.11

10.244.2.2

BRIDGE
v-net-0
10.244.2.0/24

10.244.2.1

docker

NODE2 192.168.1.12

10.244.3.2

BRIDGE
v-net-0
10.244.3.0/24

10.244.3.1

docker

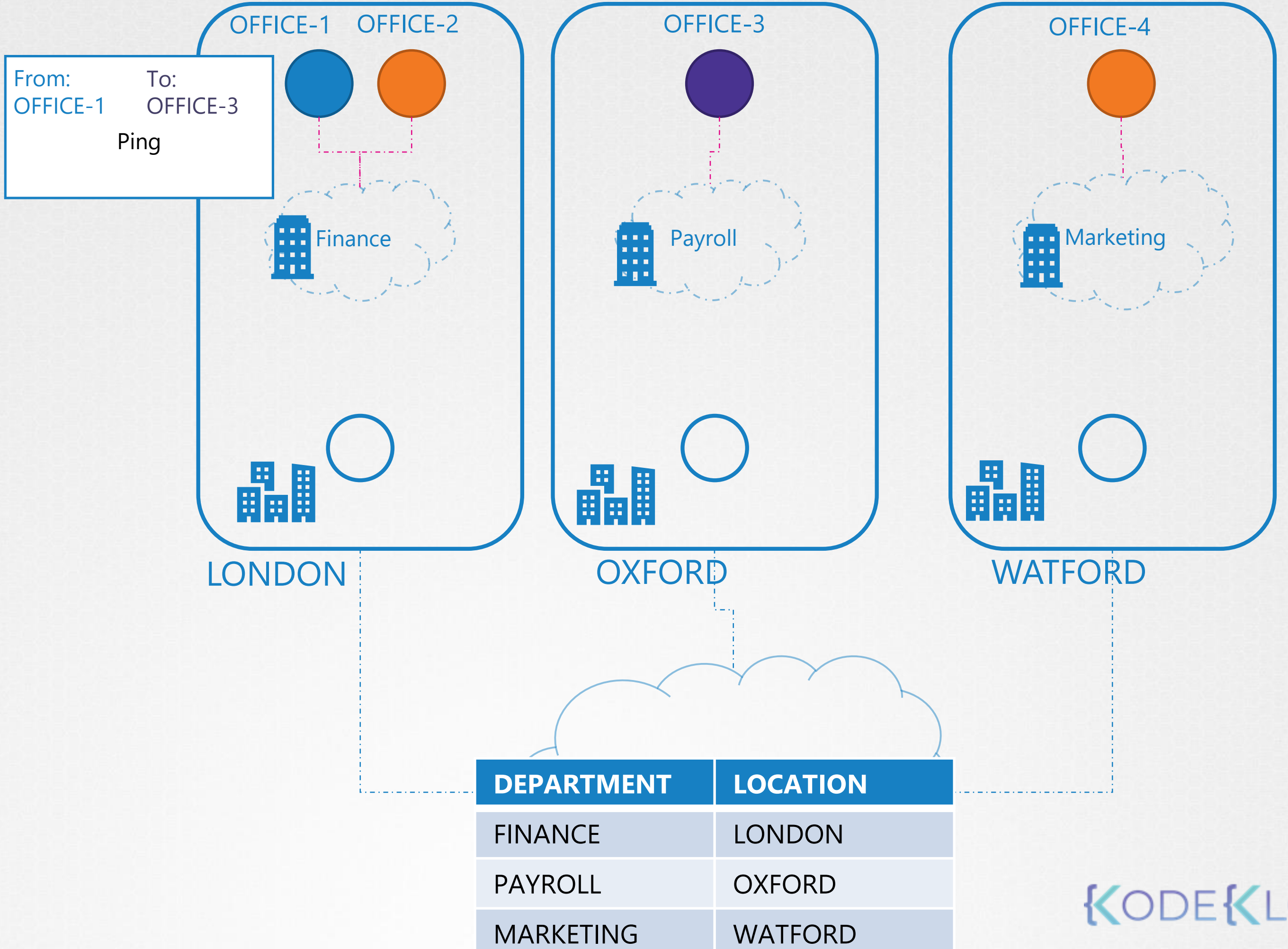
NODE3 192.168.1.13

LAN

192.168.1.0

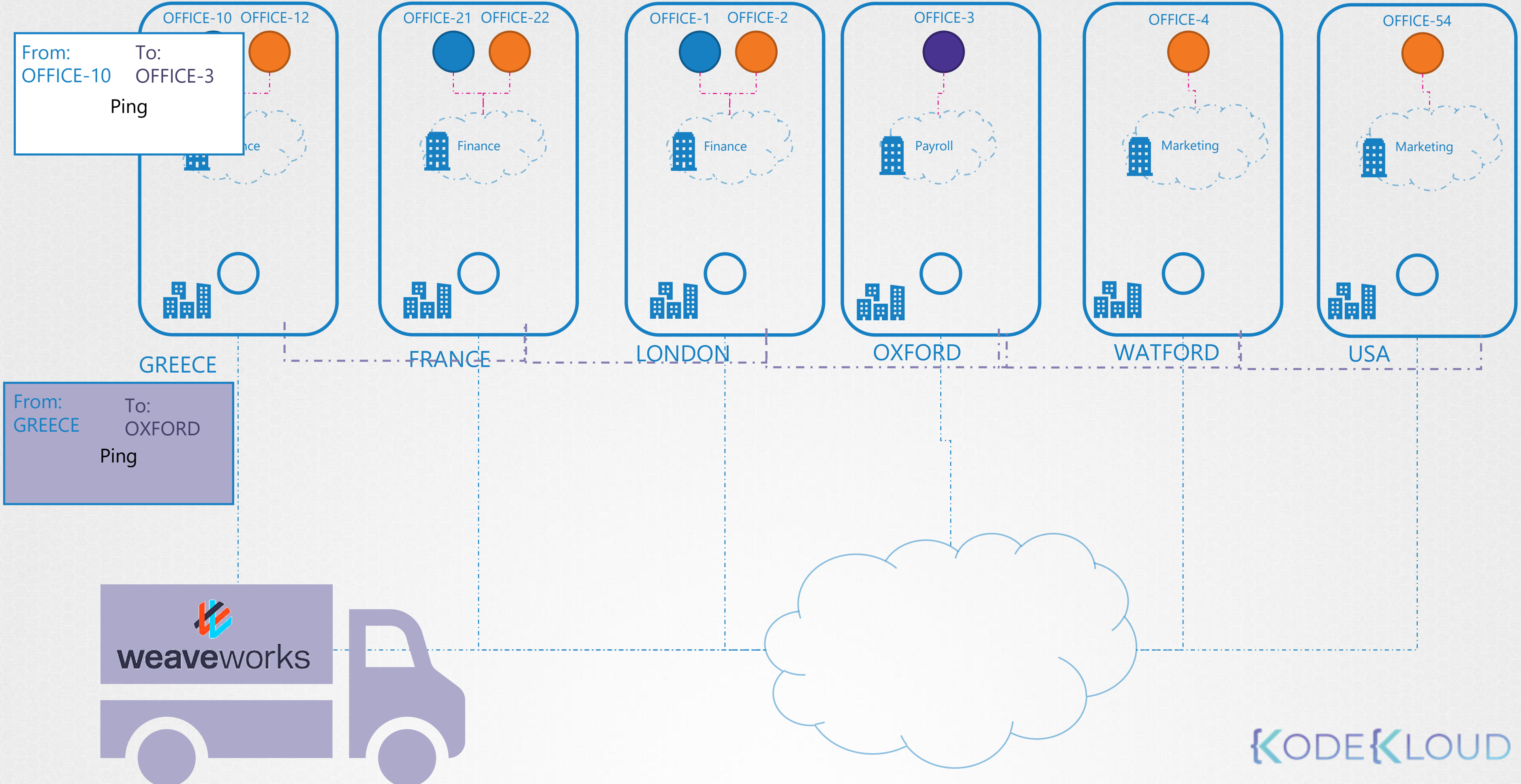
NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12

weaveworks



From: OFFICE-1
To: OFFICE-3
Ping

DEPARTMENT	LOCATION
FINANCE	LONDON
PAYROLL	OXFORD
MARKETING	WATFORD



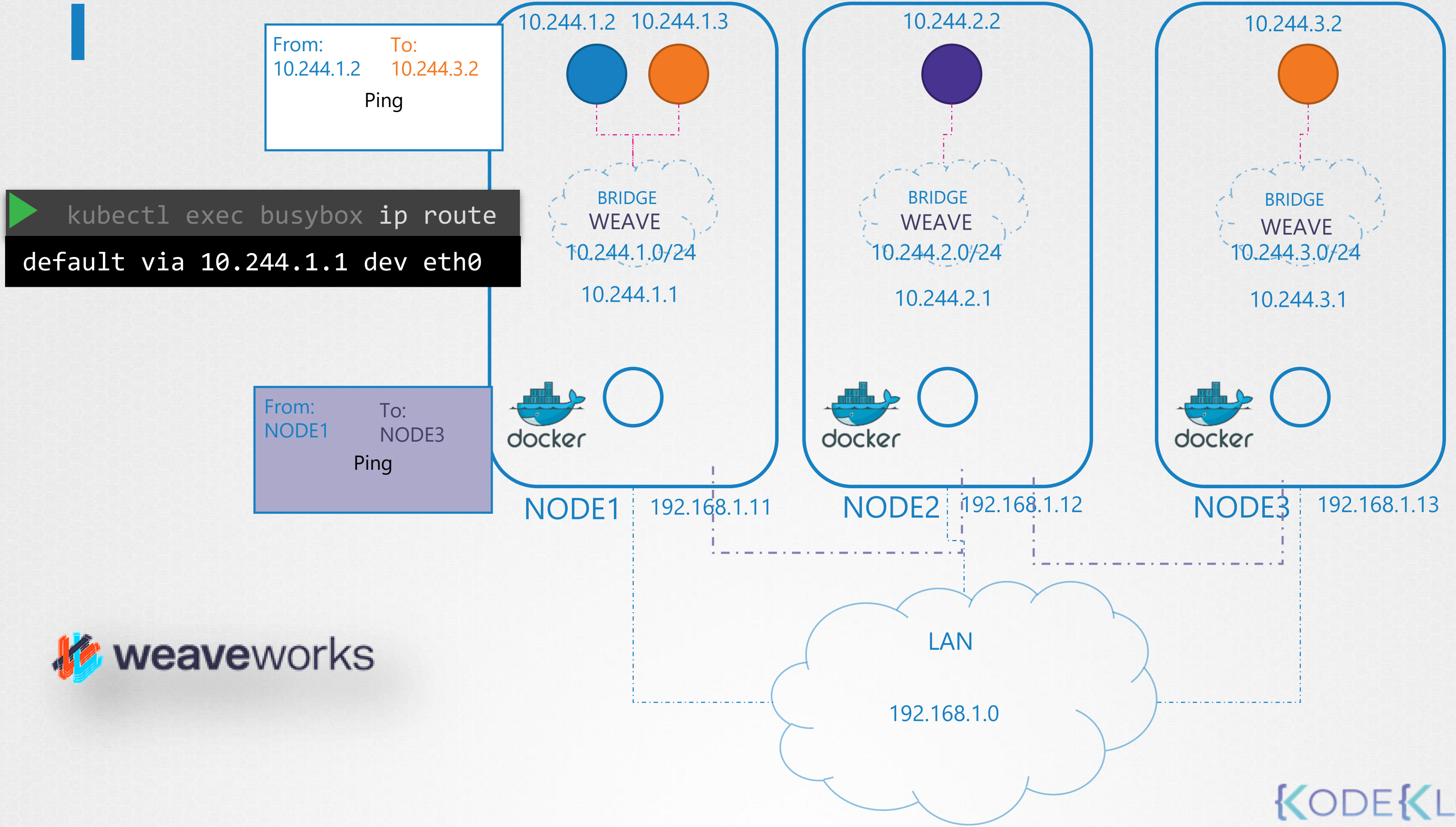
From: OFFICE-10 To: OFFICE-3

Ping

From: GREECE To: OXFORD

Ping

KODEKLOUD

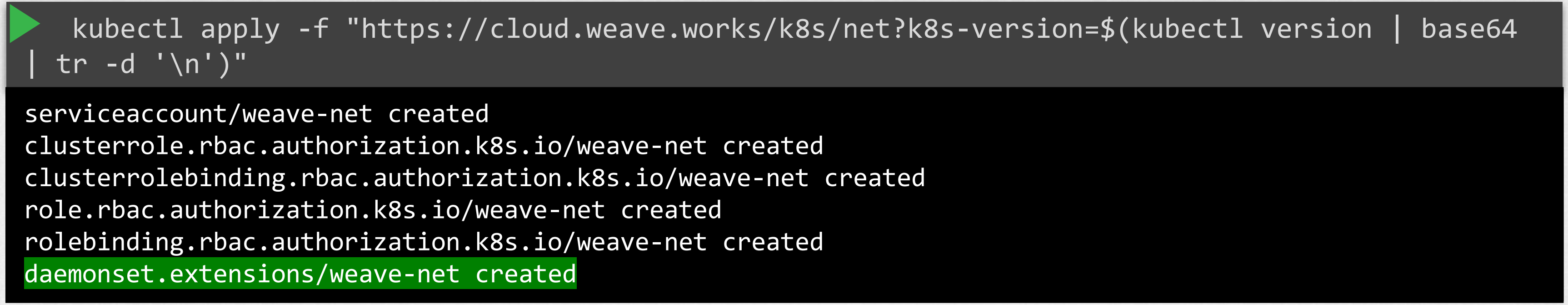


```
▶ kubectl exec busybox ip route
default via 10.244.1.1 dev eth0
```

weaveworks

KODEKLOUD

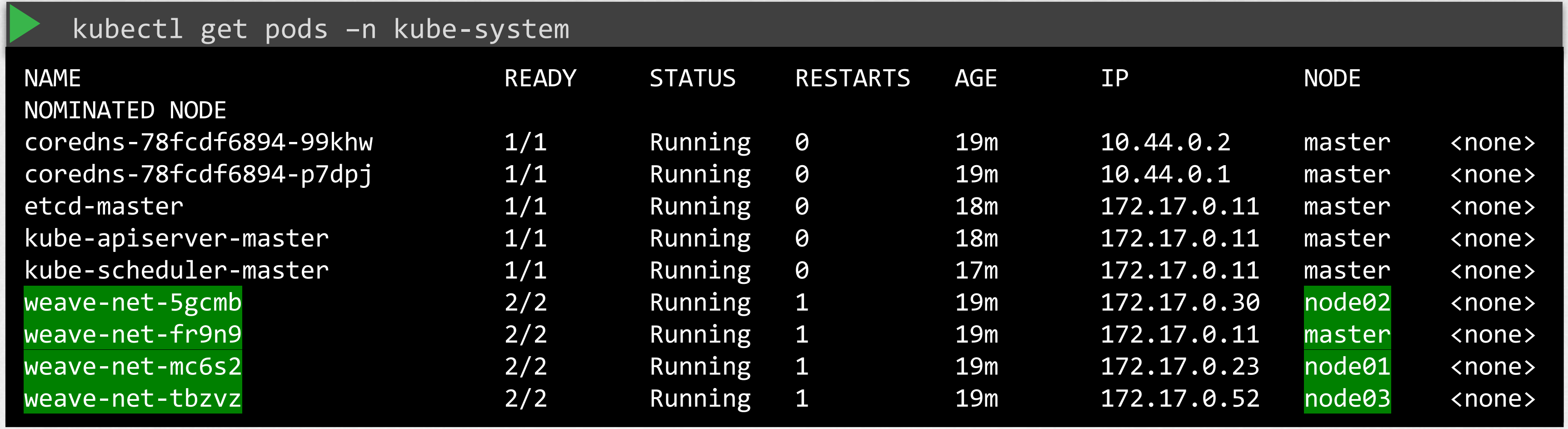
Deploy Weave



```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64
| tr -d '\n')"
```

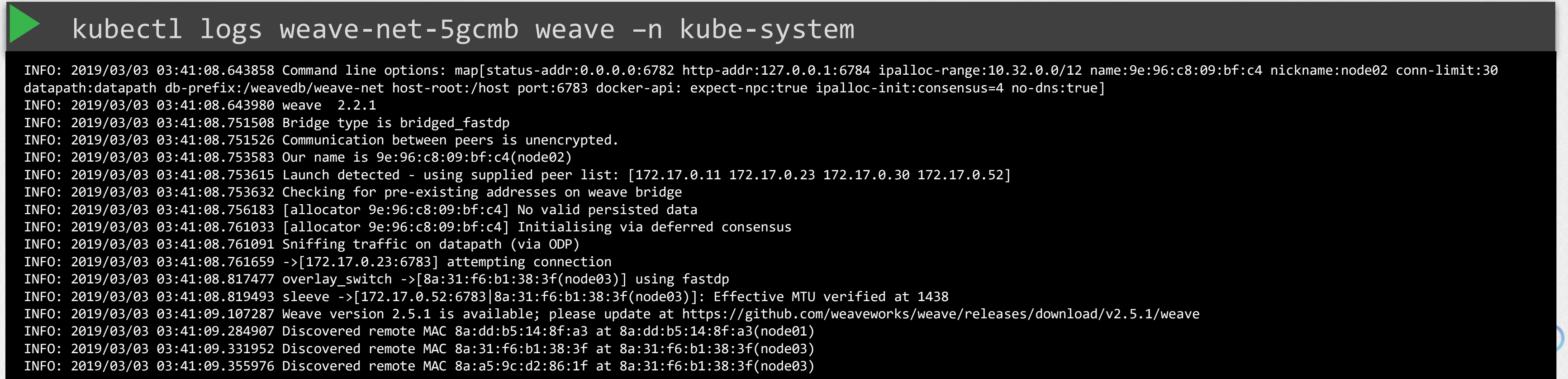
```
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.extensions/weave-net created
```

Weave Peers



```
kubectl get pods -n kube-system
```

```
NAME                         READY     STATUS    RESTARTS   AGE      IP            NODE
NOMINATED NODE
coredns-78fcdf6894-99khw     1/1       Running   0          19m      10.44.0.2     master    <none>
coredns-78fcdf6894-p7dpj     1/1       Running   0          19m      10.44.0.1     master    <none>
etcd-master                  1/1       Running   0          18m      172.17.0.11   master    <none>
kube-apiserver-master        1/1       Running   0          18m      172.17.0.11   master    <none>
kube-scheduler-master        1/1       Running   0          17m      172.17.0.11   master    <none>
weave-net-5gcmb              2/2       Running   1          19m      172.17.0.30   node02    <none>
weave-net-fr9n9              2/2       Running   1          19m      172.17.0.11   master    <none>
weave-net-mc6s2              2/2       Running   1          19m      172.17.0.23   node01    <none>
weave-net-tbzvz              2/2       Running   1          19m      172.17.0.52   node03    <none>
```



```
kubectl logs weave-net-5gcmb weave –n kube-system
```

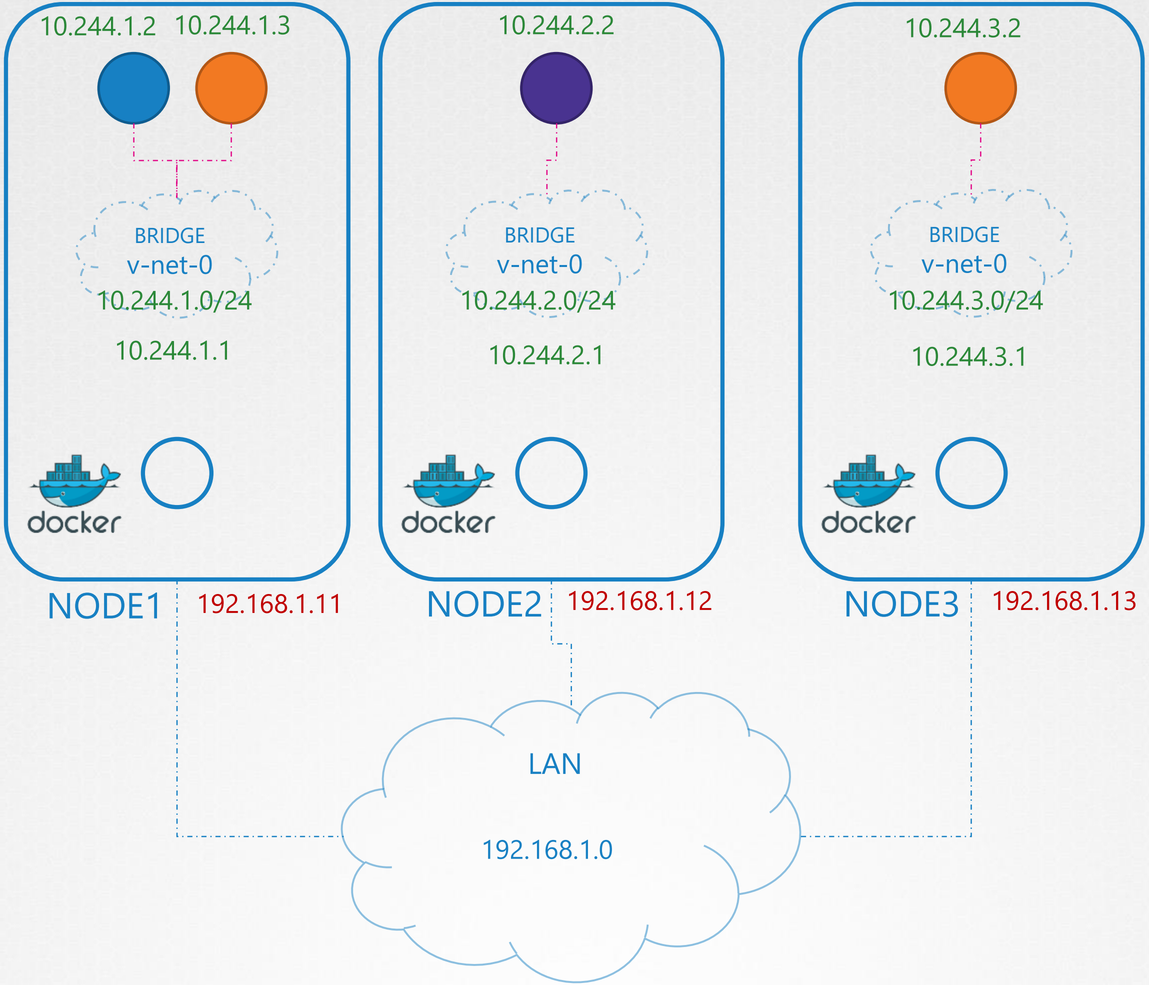
```
INFO: 2019/03/03 03:41:08.643858 Command line options: map[status-addr:0.0.0.0:6782 http-addr:127.0.0.1:6784 ipalloc-range:10.32.0.0/12 name:9e:96:c8:09:bf:c4 nickname:node02 conn-limit:30
datapath:datapath db-prefix:/weavedb/weave-net host-root:/host port:6783 docker-api: expect-npc:true ipalloc-init:consensus=4 no-dns:true]
INFO: 2019/03/03 03:41:08.643980 weave  2.2.1
INFO: 2019/03/03 03:41:08.751508 Bridge type is bridged_fastdp
INFO: 2019/03/03 03:41:08.751526 Communication between peers is unencrypted.
INFO: 2019/03/03 03:41:08.753583 Our name is 9e:96:c8:09:bf:c4(node02)
INFO: 2019/03/03 03:41:08.753615 Launch detected - using supplied peer list: [172.17.0.11 172.17.0.23 172.17.0.30 172.17.0.52]
INFO: 2019/03/03 03:41:08.753632 Checking for pre-existing addresses on weave bridge
INFO: 2019/03/03 03:41:08.756183 [allocator 9e:96:c8:09:bf:c4] No valid persisted data
INFO: 2019/03/03 03:41:08.761033 [allocator 9e:96:c8:09:bf:c4] Initialising via deferred consensus
INFO: 2019/03/03 03:41:08.761091 Sniffing traffic on datapath (via ODP)
INFO: 2019/03/03 03:41:08.761659 ->[172.17.0.23:6783] attempting connection
INFO: 2019/03/03 03:41:08.817477 overlay_switch ->[8a:31:f6:b1:38:3f(node03)] using fastdp
INFO: 2019/03/03 03:41:08.819493 sleeve ->[172.17.0.52:6783|8a:31:f6:b1:38:3f(node03)]: Effective MTU verified at 1438
INFO: 2019/03/03 03:41:09.107287 Weave version 2.5.1 is available; please update at https://github.com/weaveworks/weave/releases/download/v2.5.1/weave
INFO: 2019/03/03 03:41:09.284907 Discovered remote MAC 8a:dd:b5:14:8f:a3 at 8a:dd:b5:14:8f:a3(node01)
INFO: 2019/03/03 03:41:09.331952 Discovered remote MAC 8a:31:f6:b1:38:3f at 8a:31:f6:b1:38:3f(node03)
INFO: 2019/03/03 03:41:09.355976 Discovered remote MAC 8a:a5:9c:d2:86:1f at 8a:31:f6:b1:38:3f(node03)
```

KODEKLOUD

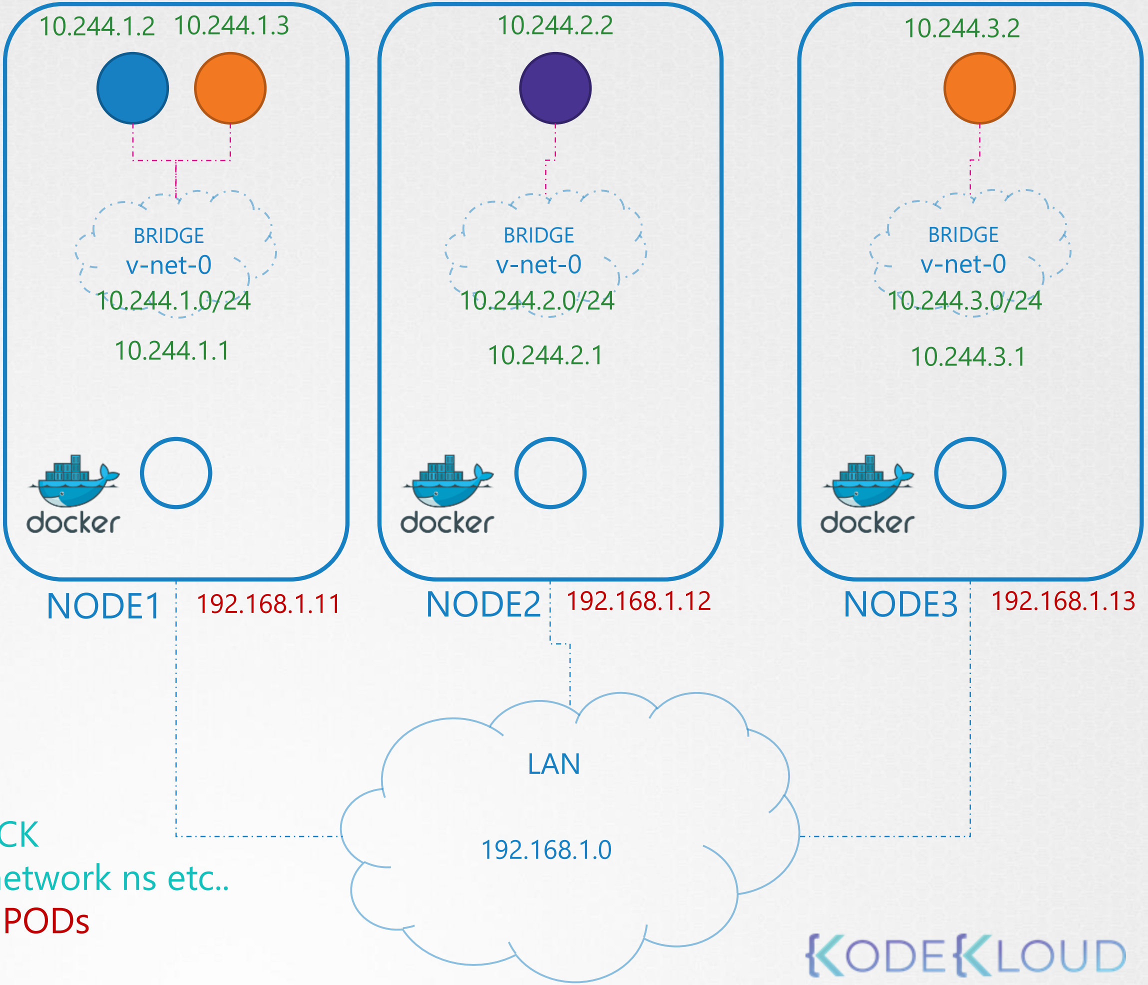
Course Objectives

- ✓ Scheduling
- ✓ Logging Monitoring
- ✓ Application Lifecycle Management
- ✓ Cluster Maintenance
- ✓ Security
- ✓ Storage
- ✓ Troubleshooting
- ✓ Core Concepts
- ● Networking
  - ✓ Pre-Requisites – Switching, Routing
  - ✓ Pre-Requisites – Tools
  - ✓ Pre-Requisites CNI
  - ✓ Pre-Requisites – DNS
  - ✓ Pre-Requisites – Networking in Docker
  - ✓ Networking Configuration on Cluster Nodes
  - ✓ POD Networking Concepts
  - CNI in Kubernetes
  - ○ Cluster DNS
  - ○ Service Networking
  - ○ Network Loadbalancer
  - ○ Ingress

IPAM (CNI)


10.244.1.2
10.244.1.3
BRIDGE
v-net-0
10.244.1.0/24
10.244.1.1
docker
NODE1
192.168.1.11
10.244.2.2
BRIDGE
v-net-0
10.244.2.0/24
10.244.2.1
docker
NODE2
192.168.1.12
10.244.3.2
BRIDGE
v-net-0
10.244.3.0/24
10.244.3.1
docker
NODE3
192.168.1.13
LAN
192.168.1.0

CONTAINER NETWORK INTERFACE (CNI)



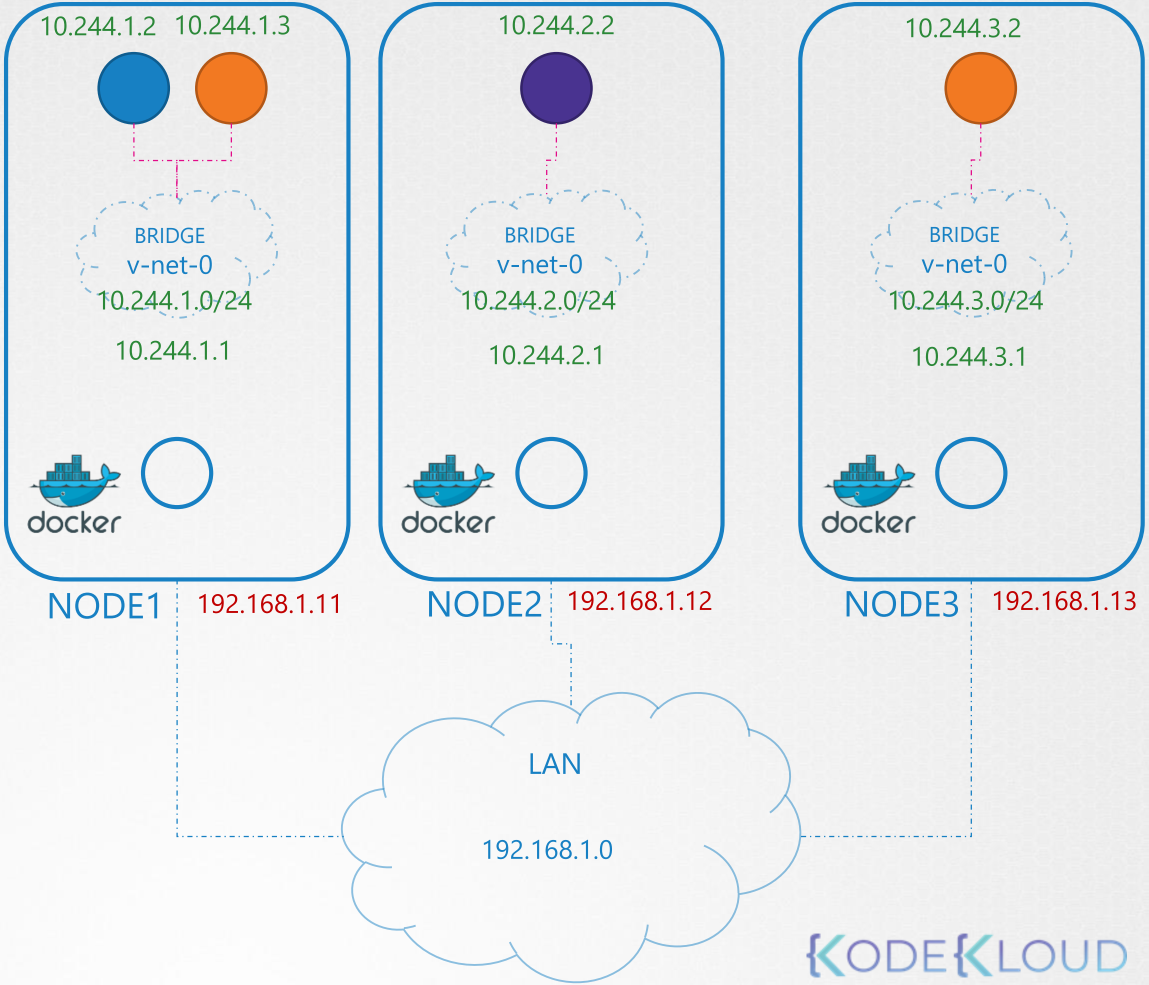
CNI Plugin Responsibilities:

- ✓ Must support arguments ADD/DEL/CHECK
- ✓ Must support parameters container id, network ns etc..
- ✓ Must manage IP Address assignment to PODs
- ✓ Must Return results in a specific format

CONTAINER NETWORK INTERFACE (CNI)

net-script.sh

```
ADD)
 # Create veth pair

 # Attach veth pair

 # Assign IP Address

  # Bring Up Interface
  ip -n <namespace> link set ......
DEL)

  # Delete veth pair
  ip link del ......
```

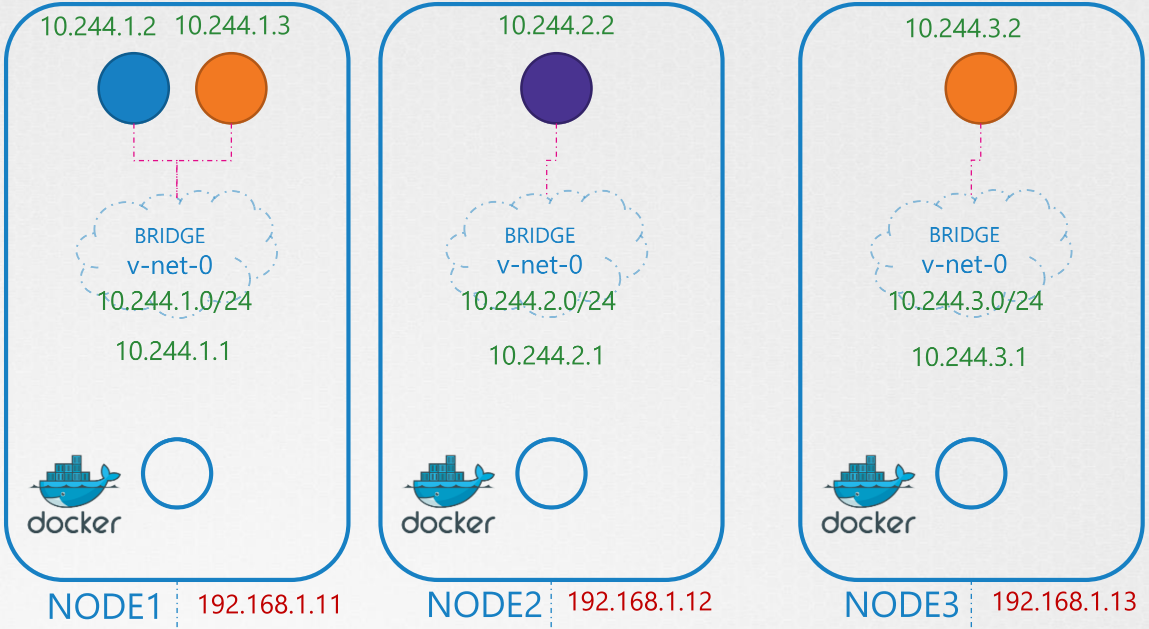


CONTAINER NETWORK INTERFACE (CNI)

DHCP

host-local

```
 # Invoke IPAM host-local plugin
ip = get_free_ip_from_host_local()
 # Assign IP Address
 ip -n <namespace> addr add ......
 ip -n <namespace> route add ......
```



Ip-list.txt

IP	STATUS	POD
10.244.1.2	ASSIGNED	BLUE
10.244.1.3	ASSIGNED	ORANGE
10.244.1.4	FREE	
10.244.1.4	FREE	
10.244.1.4	FREE	

Ip-list.txt

IP	STATUS	POD
10.244.2.2	ASSIGNED	PURPLE
10.244.2.3	FREE	
10.244.2.4	FREE	
10.244.2.4	FREE	
10.244.2.4	FREE	

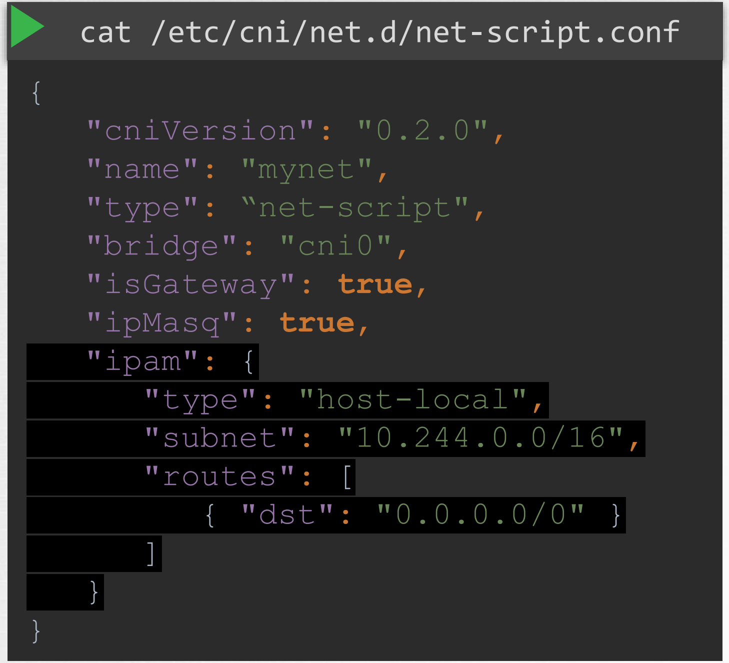
Ip-list.txt

IP	STATUS
10.244.3.2	ASSIGNED
10.244.3.3	FREE
10.244.3.4	FREE
10.244.3.4	FREE
10.244.3.4	FREE

CONTAINER NETWORK INTERFACE (CNI)

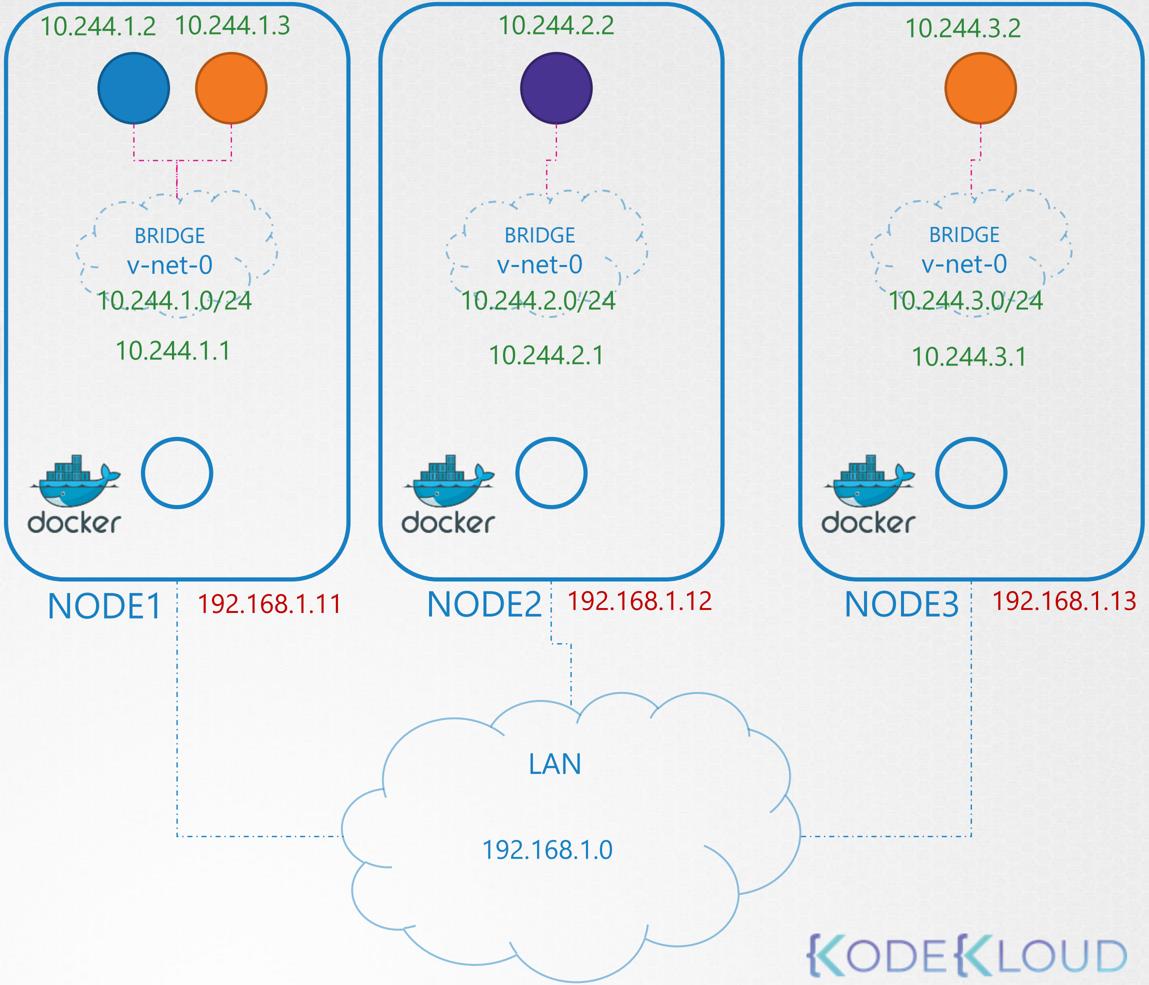
DHCP

host-local



```
cat /etc/cni/net.d/net-script.conf
```

```
{
    "cniVersion": "0.2.0",
    "name": "mynet",
    "type": “net-script",
    "bridge": "cni0",
    "isGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "subnet": "10.244.0.0/16",
        "routes": [
            { "dst": "0.0.0.0/0" }
        ]
    }
}
```

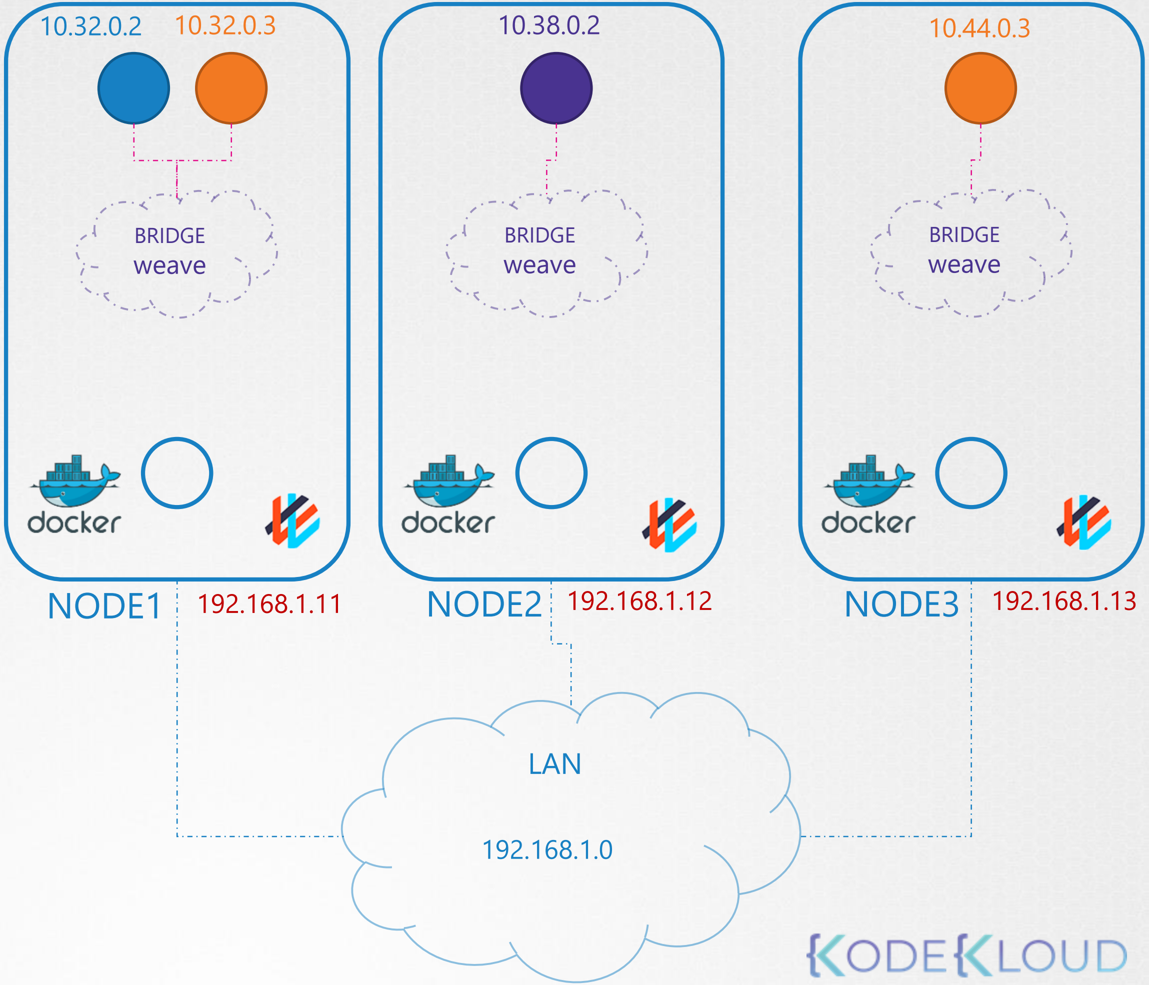


10.32.0.0/12

10.32.0.1 > 10.47.255.254

1,048,574 !!!!!

10.32.0.1 10.38.0.0 10.44.0.0



weaveworks

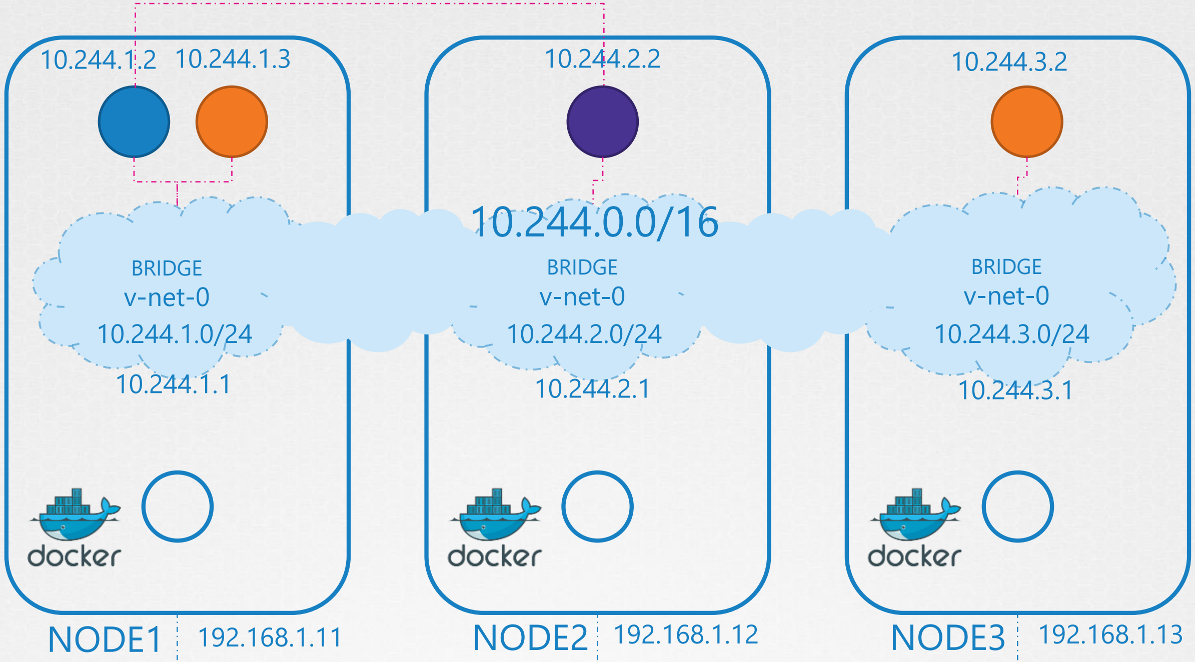
KODEKLOUD

KODEKLOUD

Course Objectives

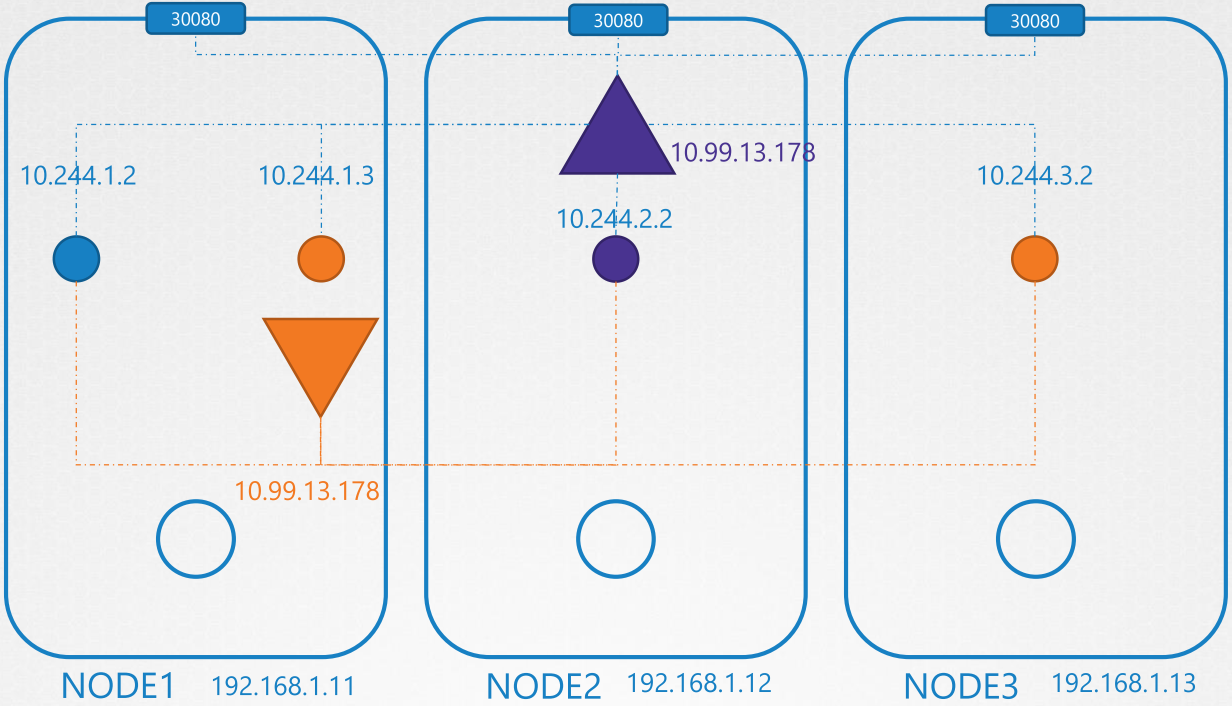
- ✓ Scheduling
- ✓ Logging Monitoring
- ✓ Application Lifecycle Management
- ✓ Cluster Maintenance
- ✓ Security
- ✓ Storage
- ✓ Troubleshooting
- ✓ Core Concepts
- ● Networking
  - ✓ Pre-Requisites – Switching, Routing
  - ✓ Pre-Requisites – Tools
  - ✓ Pre-Requisites CNI
  - ✓ Pre-Requisites – DNS
  - ✓ Pre-Requisites – Networking in Docker
  - ✓ Networking Configuration on Cluster Nodes
  - ✓ POD Networking Concepts
  - ○ CNI in Kubernetes
  - Service Networking
  - ○ Cluster DNS
  - ○ Network Loadbalancer
  - ○ Ingress

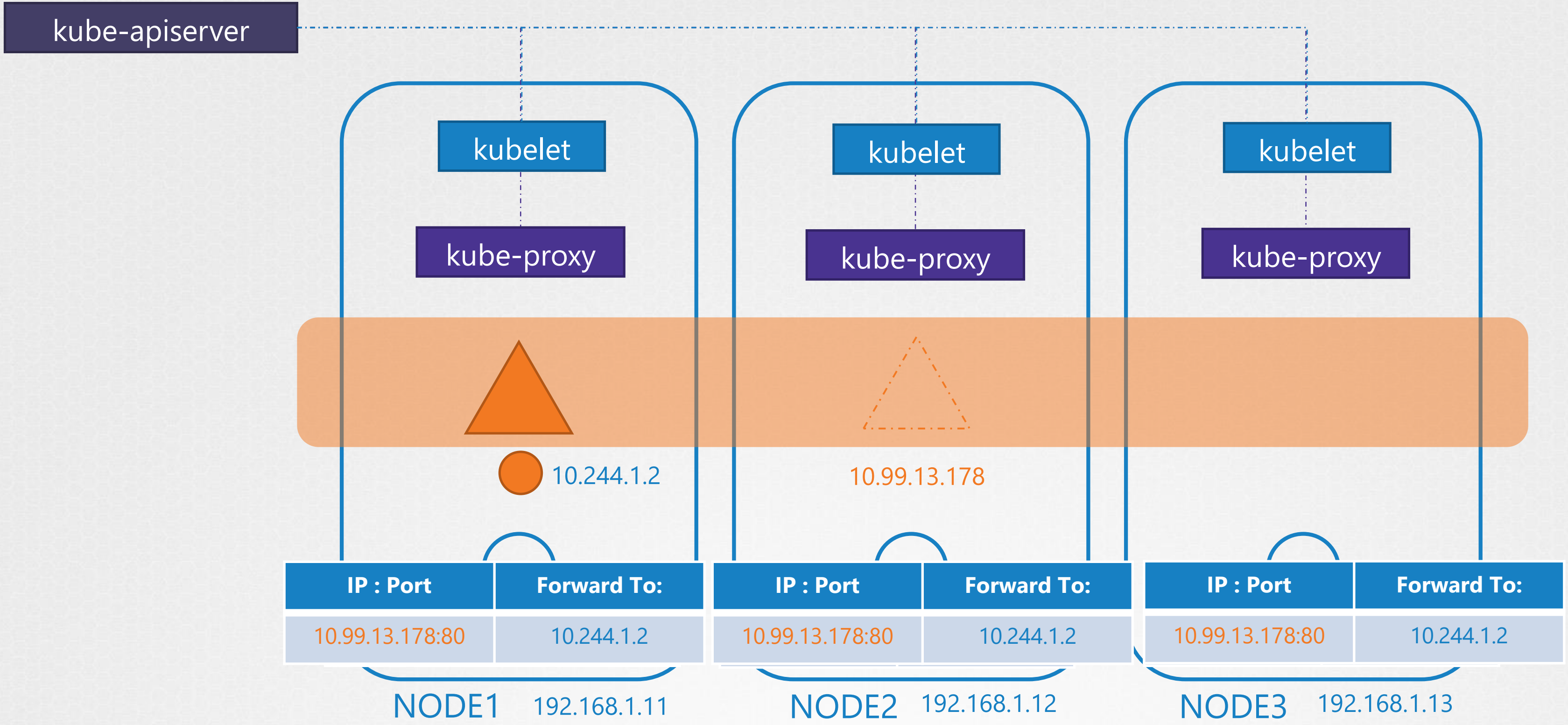
Service Networking



NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12
10.244.3.0/24	192.168.1.13

NodePort


30080
30080
30080
10.99.13.178
10.244.1.2
10.244.1.3
10.244.2.2
10.244.3.2
10.99.13.178
NODE1
192.168.1.11
NODE2
192.168.1.12
NODE3
192.168.1.13



IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

kube-proxy

userspace

iptables

ipvs

```
kube-proxy --proxy-mode [userspace | iptablkes | ipvs ] …
```

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

kube-proxy

iptables

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

```
▶ kubelet get pods -o wide
NAME       READY      STATUS     RESTARTS    AGE        IP           NODE
db         1/1        Running    0           14h        10.244.1.2   node-1
```

```
▶ kubelet get service
NAME              TYPE          CLUSTER-IP       PORT(S)          AGE
db-service        ClusterIP     10.103.132.104   3306/TCP         12h
```

● 10.244.1.2

10.244.0.0/16

10.244.0.0 => 10.244.255.255

10.96.0.0 => 10.111.255.255

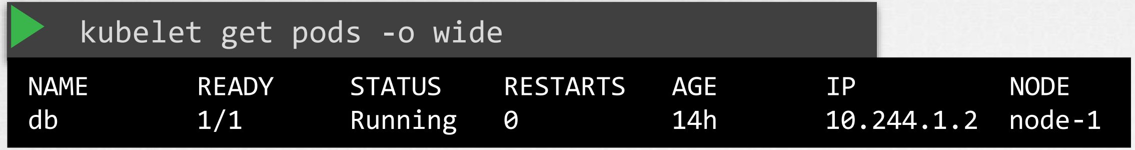
```
kube-api-server --service-cluster-ip-range ipNet (Default: 10.0.0.0/24)
```

```
▶ ps aux | grep kube-api-server
kube-apiserver --authorization-mode=Node,RBAC --service-cluster-ip-
range=10.96.0.0/12
```

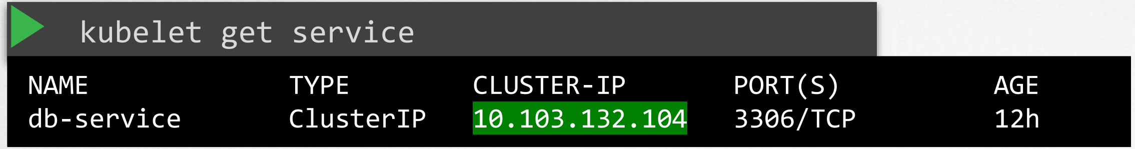
kube-proxy

iptables

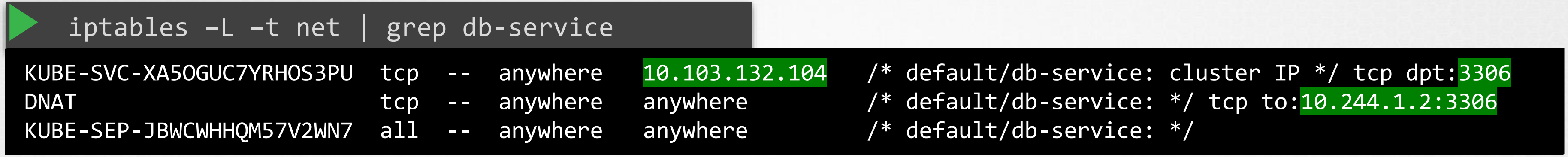
IP : Port	Forward To:
10.99.13.178:80	10.244.1.2



```
kubelet get pods -o wide

NAME         READY     STATUS     RESTARTS    AGE       IP           NODE
db           1/1       Running    0           14h       10.244.1.2   node-1
```



```
kubelet get service

NAME              TYPE         CLUSTER-IP       PORT(S)          AGE
db-service        ClusterIP    10.103.132.104   3306/TCP         12h
```



```
iptables –L –t net | grep db-service

KUBE-SVC-XA5OGUC7YRHOS3PU  tcp  --  anywhere   10.103.132.104   /* default/db-service: cluster IP */ tcp dpt:3306
DNAT                       tcp  --  anywhere   anywhere         /* default/db-service: */ tcp to:10.244.1.2:3306
KUBE-SEP-JBWCWHHQM57V2WN7  all  --  anywhere   anywhere         /* default/db-service: */
```

kube-proxy

iptables

IP : Port	Forward To:
10.99.13.178:80	10.244.1.2

```
▶ iptables –L –t net | grep db-service
```

```
KUBE-SVC-XA5OGUC7YRHOS3PU  tcp  --  anywhere   10.103.132.104   /* default/db-service: cluster IP */ tcp dpt:3306
DNAT                       tcp  --  anywhere   anywhere         /* default/db-service: */ tcp to:10.244.1.2:3306
KUBE-SEP-JBWCWHHQM57V2WN7  all  --  anywhere   anywhere         /* default/db-service: */
```

```
▶ cat /var/log/kube-proxy.log
```

```
I0307 04:29:29.883941       1 server_others.go:140] Using iptables Proxier.
I0307 04:29:29.912037       1 server_others.go:174] Tearing down inactive rules.
I0307 04:29:30.027360       1 server.go:448] Version: v1.11.8
I0307 04:29:30.049773       1 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_max' to 131072
I0307 04:29:30.049945       1 conntrack.go:52] Setting nf_conntrack_max to 131072
I0307 04:29:30.050701       1 conntrack.go:83] Setting conntrack hashsize to 32768
I0307 04:29:30.050701       1 proxier.go:294] Adding new service “default/db-service:3306" at 10.103.132.104:3306/TCP
```

KODEKLOUD

KODEKLOUD

Course Objectives

- ✓ Scheduling
- ✓ Logging Monitoring
- ✓ Application Lifecycle Management
- ✓ Cluster Maintenance
- ✓ Security
- ✓ Storage
- ✓ Troubleshooting
- ✓ Core Concepts
- ● Networking
  - ✓ Pre-Requisites – Switching, Routing
  - ✓ Pre-Requisites – DNS
  - ✓ Networking Configuration on Cluster Nodes
  - ✓ Service Networking
  - ● Cluster DNS
  - ✓ Pre-Requisites – Tools
  - ✓ Pre-Requisites – Networking in Docker
  - ✓ POD Networking Concepts
  - ○ Network Loadbalancer
  - ✓ Pre-Requisites CNI
  - ○ CNI in Kubernetes
  - ○ Ingress

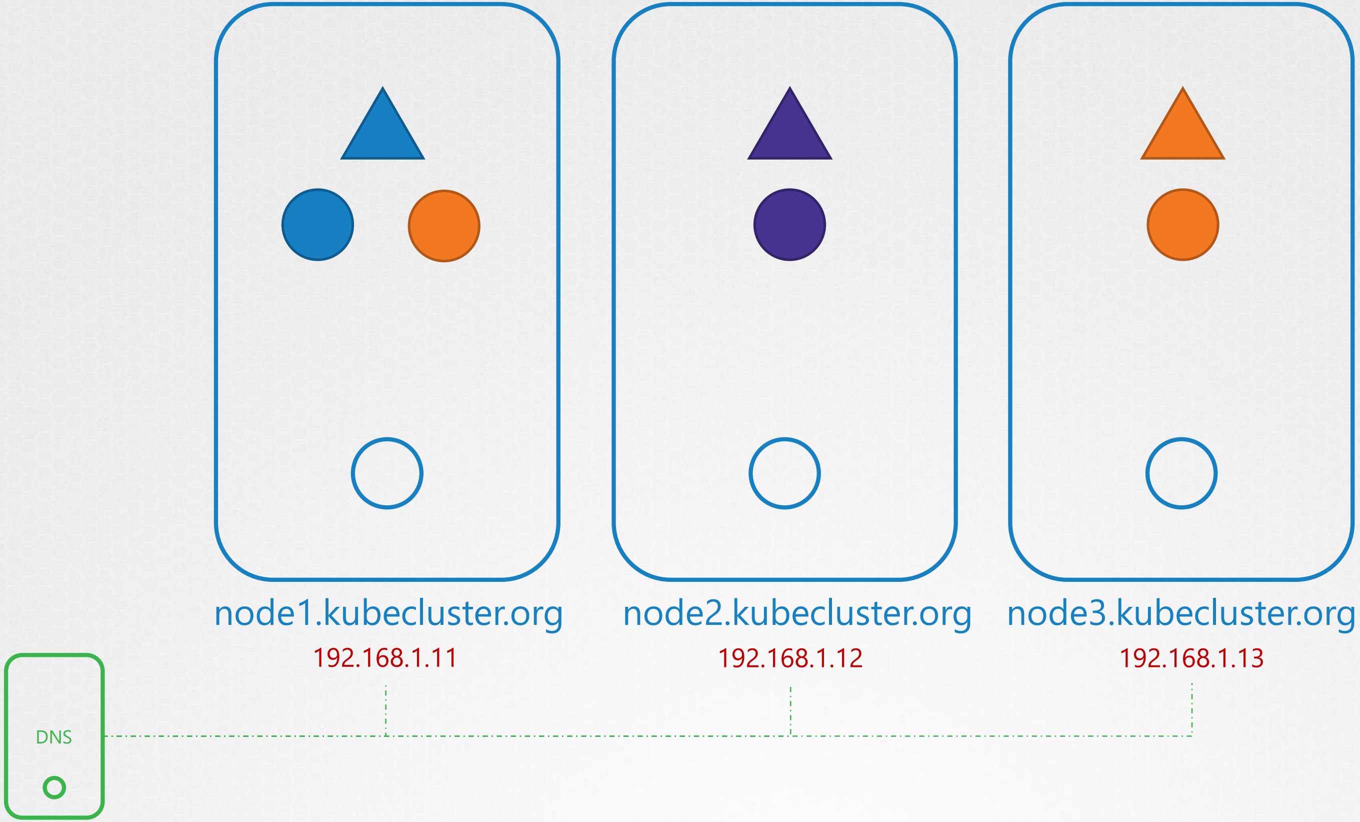
Cluster DNS

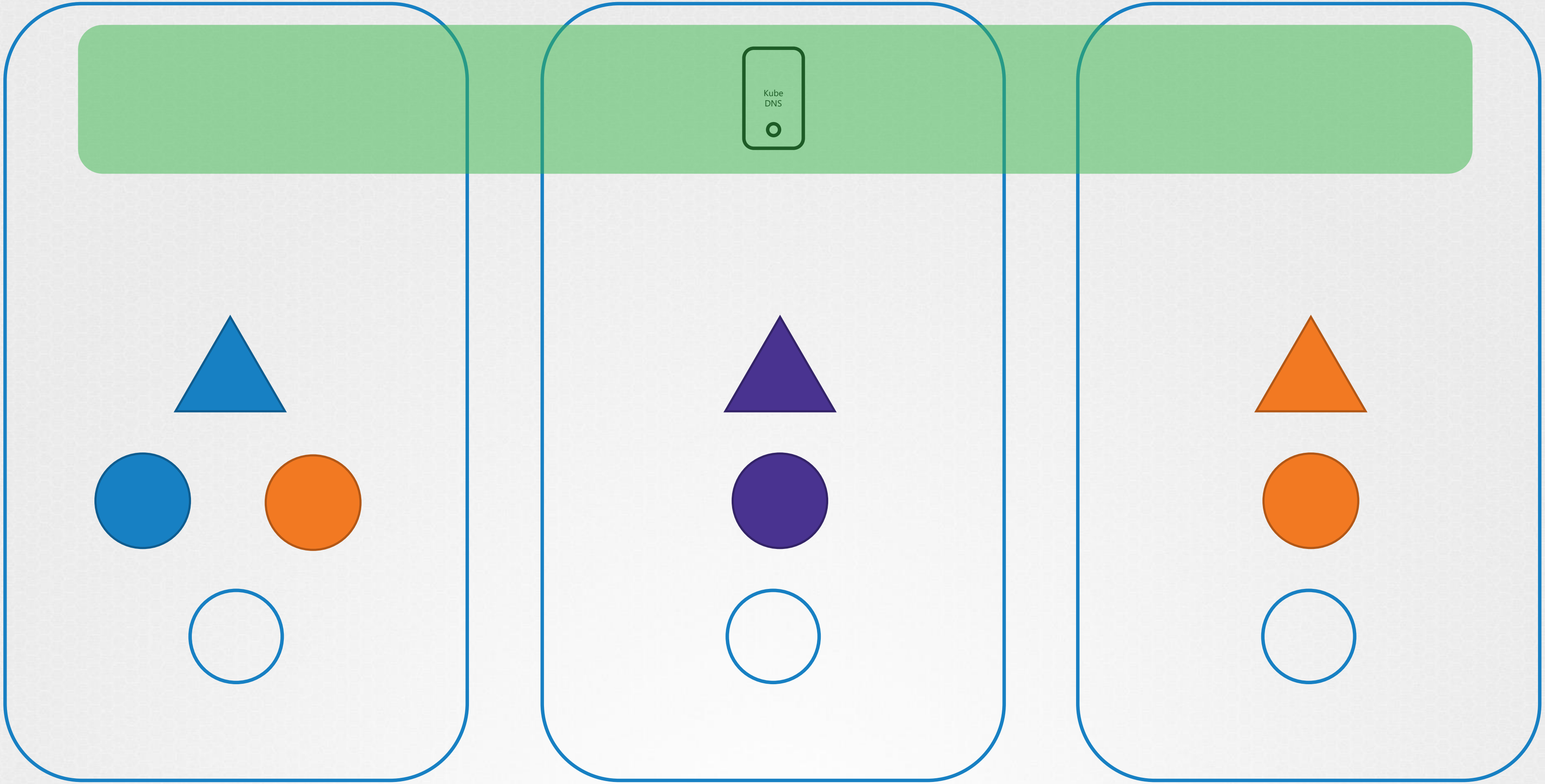
Pre-Requisite

- ✓ What is DNS?
- ✓ Host/NS Lookup, Dig utility
- ✓ Recorded Types – A, CNAME
- ✓ Domain Name Hierarchy

Objectives

- ❑ What names are assigned to what objects?
- ❑ Service DNS records
- ❑ POD DNS Recrods


node1.kubecluster.org
192.168.1.11
node2.kubecluster.org
192.168.1.12
node3.kubecluster.org
192.168.1.13
DNS



Kube DNS

node1.kubecluster.org

192.168.1.11

node2.kubecluster.org

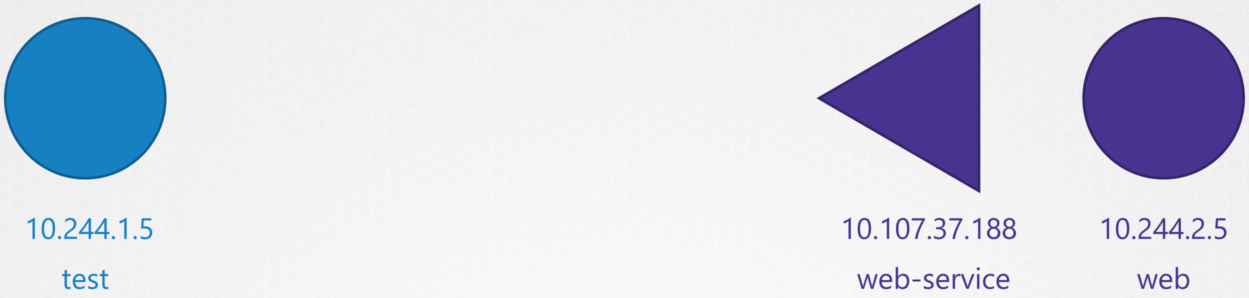
192.168.1.12

node3.kubecluster.org

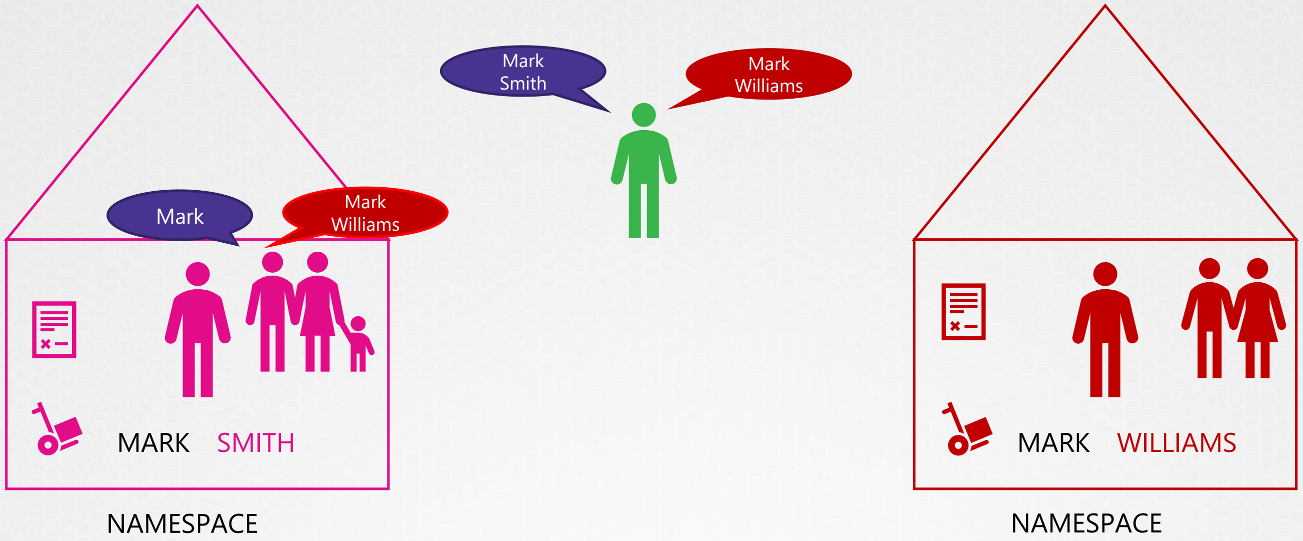
192.168.1.13

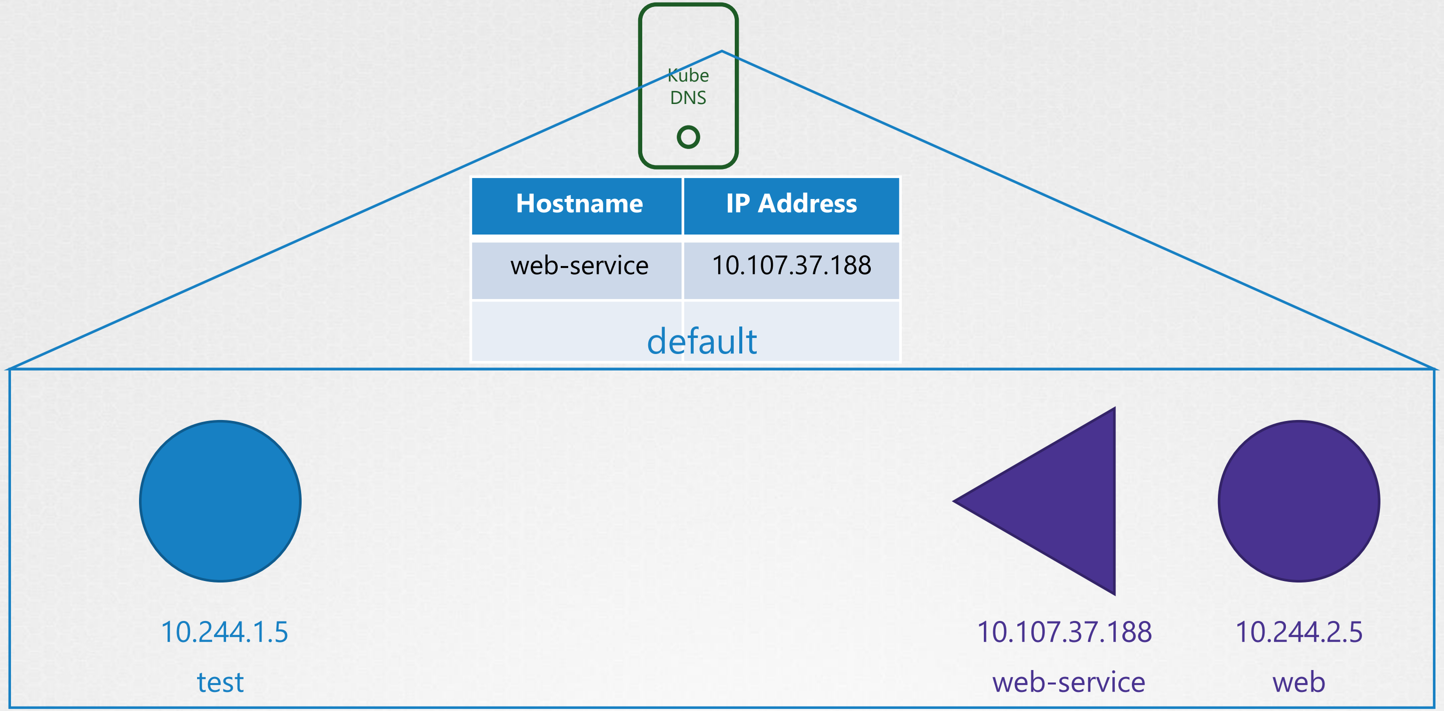
Kube DNS

Hostname	IP Address
web-service	10.107.37.188

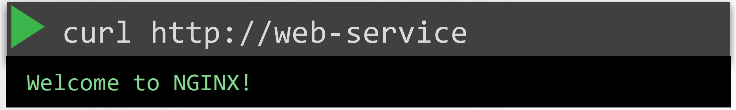


```
curl http://web-service
Welcome to NGINX!
```

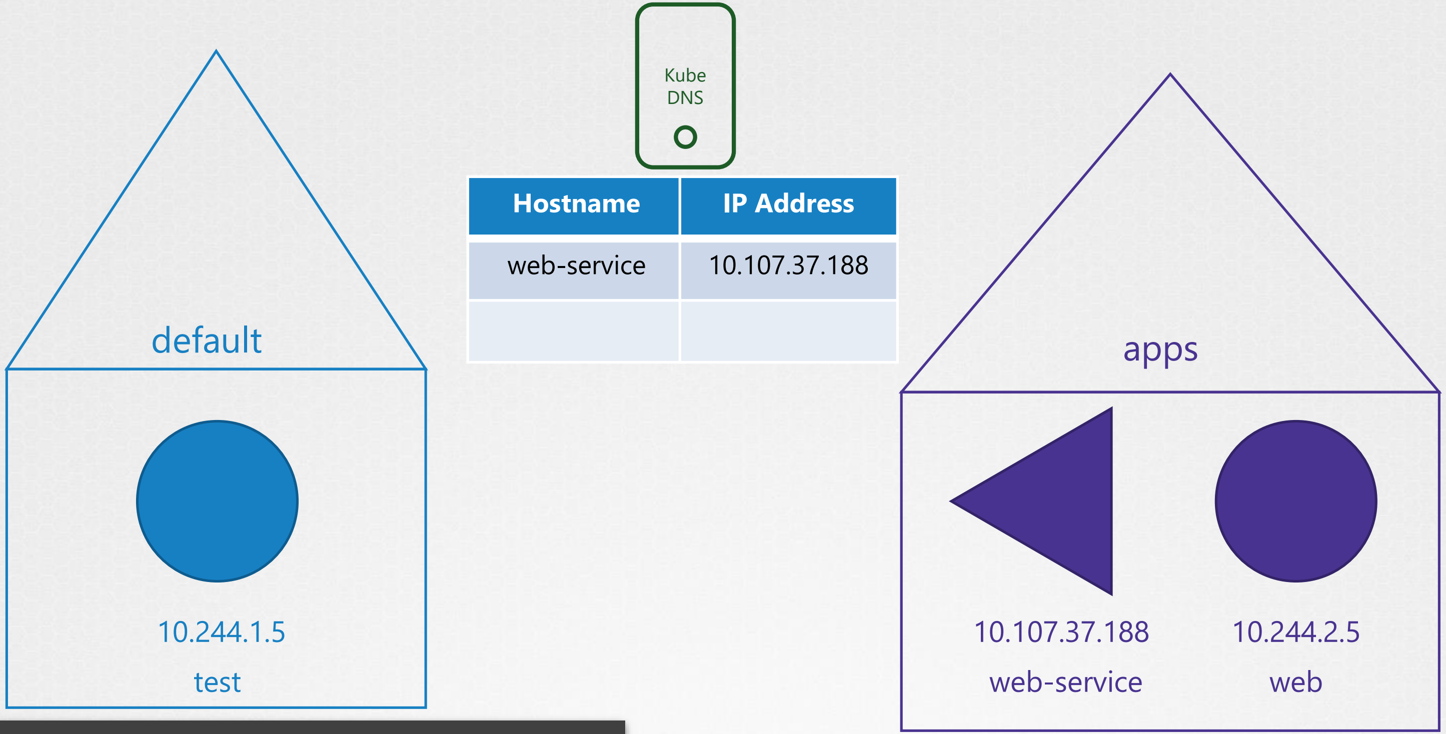

Mark
Smith
Mark
Williams
Mark
Mark
Williams
MARK SMITH
NAMESPACE
MARK WILLIAMS
NAMESPACE



Hostname	IP Address
web-service	10.107.37.188



```
curl http://web-service
Welcome to NGINX!
```



Hostname	IP Address
web-service	10.107.37.188

```
▶ curl http://web-service
Welcome to NGINX!
```

```
▶ curl http://web-service.apps
Welcome to NGINX!
```

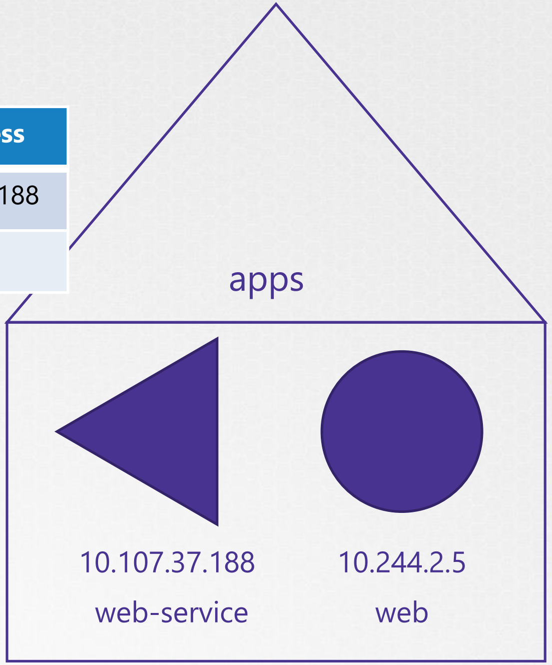
Kube DNS

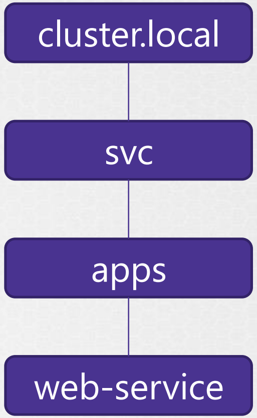
Hostname	Namespace	Type	Root	IP Address
web-service	apps	svc	cluster.local	10.107.37.188

```
curl http://web-service.apps
Welcome to NGINX!
```

```
curl http://web-service.apps.svc
Welcome to NGINX!
```

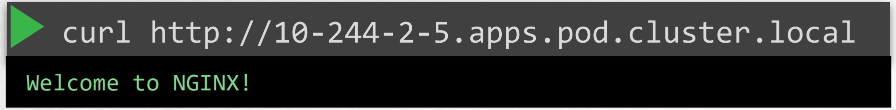
```
curl http://web-service.apps.svc.cluster.local
Welcome to NGINX!
```



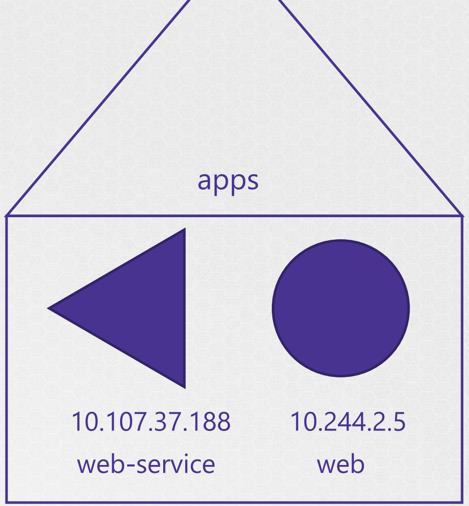


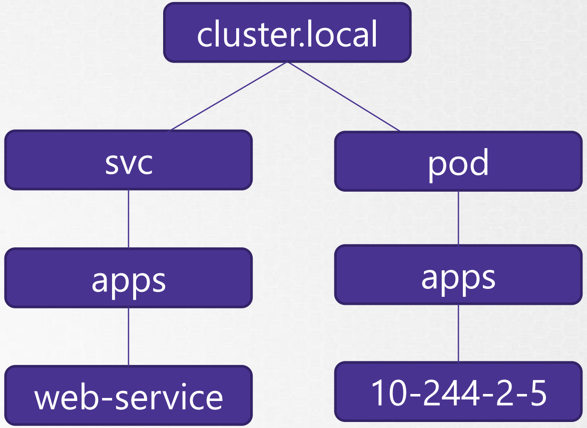
Hostname	Namespace	Type	Root	IP Address
web-service	apps	svc	cluster.local	10.107.37.188
10-244-2-5	apps	pod	cluster.local	10.244.2.5

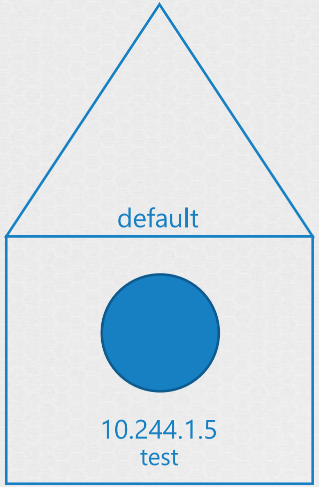
10-244-2-5 ← 10.244.2.5



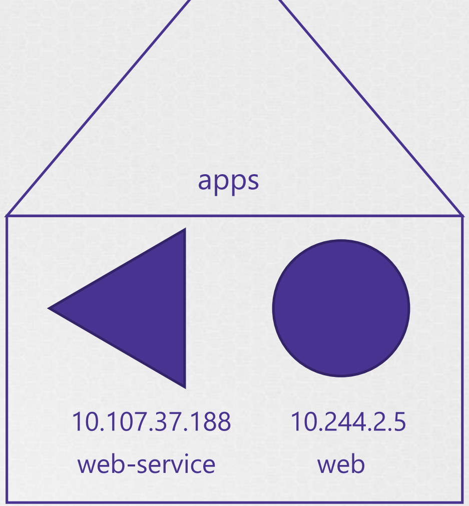
```
curl http://10-244-2-5.apps.pod.cluster.local
Welcome to NGINX!
```

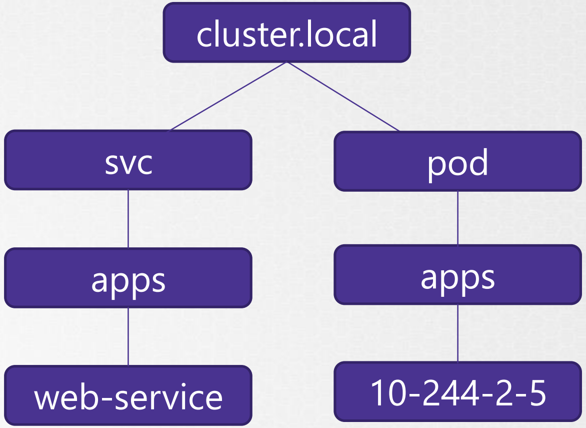






Hostname	Namespace	Type	Root	IP Address
web-service	apps	svc	cluster.local	10.107.37.188
10-244-2-5	apps	pod	cluster.local	10.244.2.5
10-244-1-5	default	pod	cluster.local	10.244.1.5





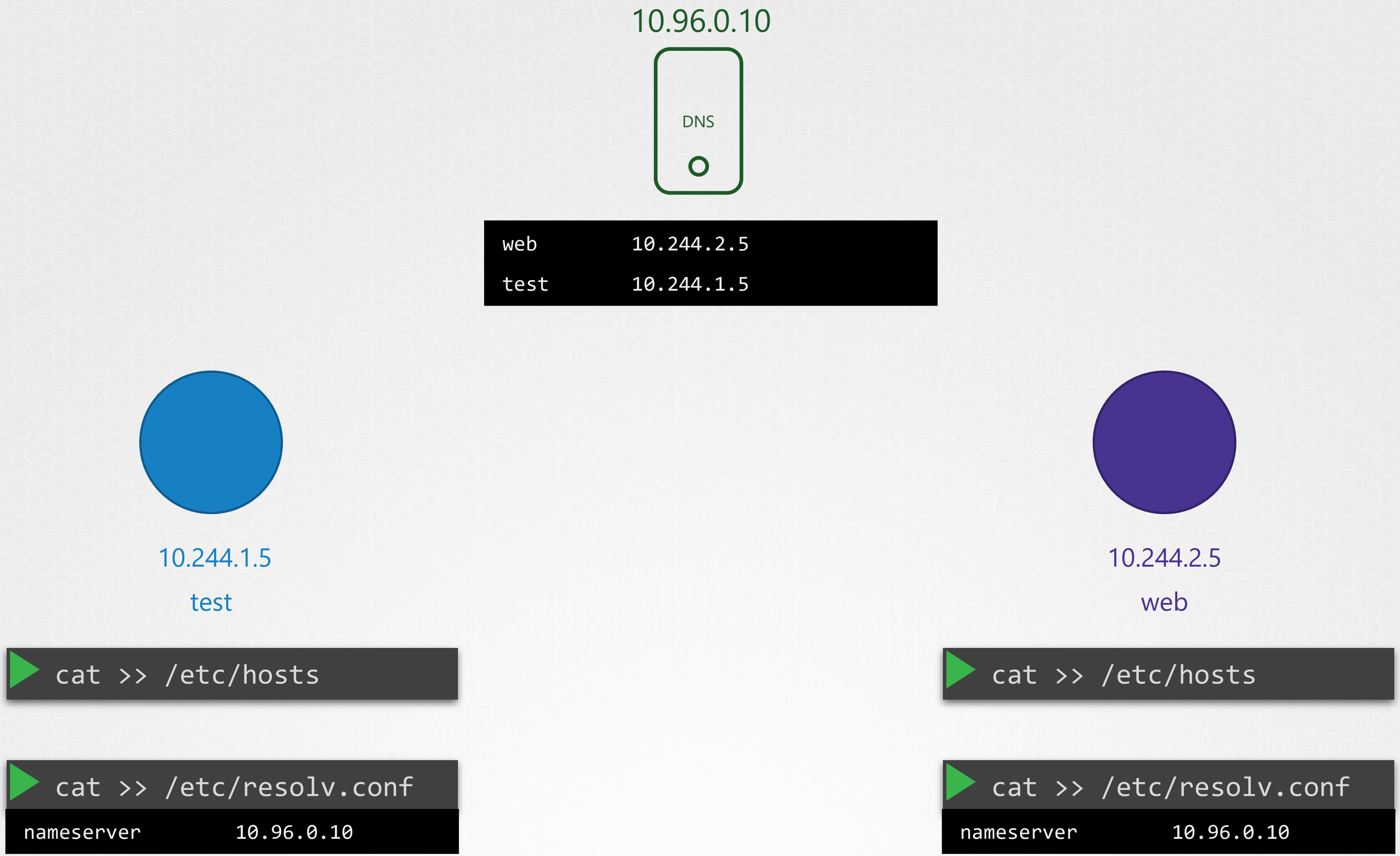
```
curl http://10-244-2-5.apps.pod.cluster.local
Welcome to NGINX!
```

KODEKLOUD

Course Objectives

- ☑ Scheduling
- ☑ Logging Monitoring
- ☑ Application Lifecycle Management
- ☑ Cluster Maintenance
- ☑ Security
- ☑ Storage
- ☑ Troubleshooting
- ☑ Core Concepts
- ● Networking
    - ☑ Pre-Requisites – Switching, Routing
    - ☑ Pre-Requisites – DNS
    - ☑ Networking Configuration on Cluster Nodes
    - ☑ Service Networking
    - ● Cluster DNS
    - ☑ Pre-Requisites – Tools
    - ☑ Pre-Requisites – Networking in Docker
    - ☑ POD Networking Concepts
    - ○ Network Loadbalancer
    - ☑ Pre-Requisites CNI
    - ○ CNI in Kubernetes
    - ○ Ingress

How Kubernetes Implements DNS?



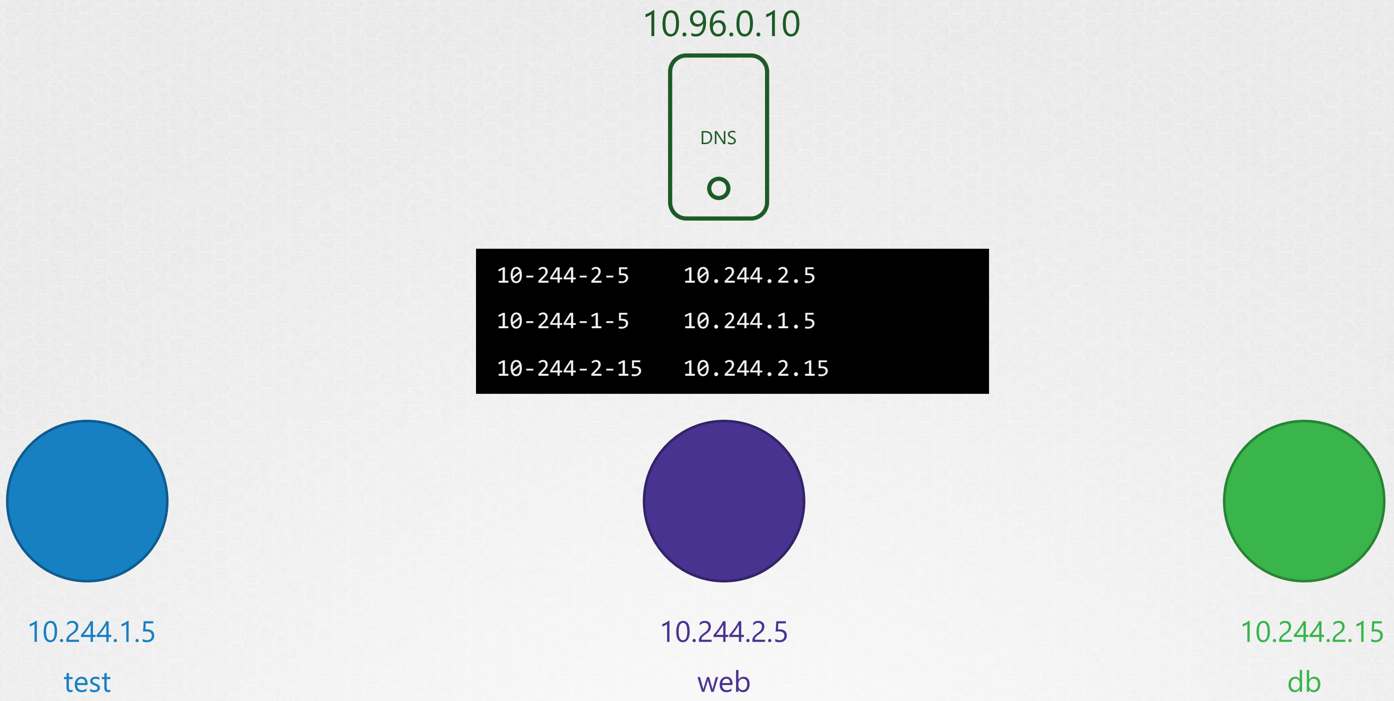
```
web          10.244.2.5
test         10.244.1.5
```

```
cat >> /etc/hosts
```

```
cat >> /etc/resolv.conf
nameserver        10.96.0.10
```

```
cat >> /etc/hosts
```

```
cat >> /etc/resolv.conf
nameserver        10.96.0.10
```



```
10-244-2-5     10.244.2.5
10-244-1-5     10.244.1.5
10-244-2-15    10.244.2.15
```

```
cat >> /etc/hosts
```

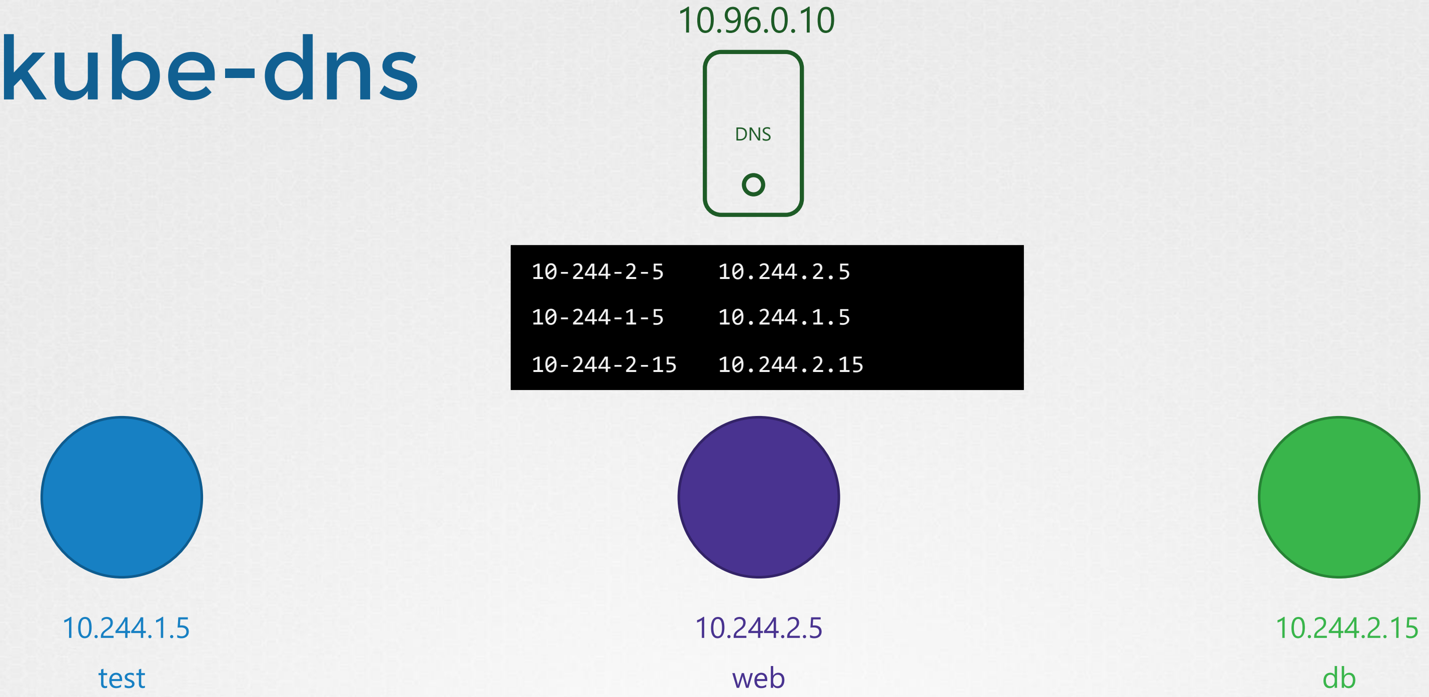
```
cat >> /etc/resolv.conf
nameserver        10.96.0.10
```

```
cat >> /etc/hosts
```

```
cat >> /etc/resolv.conf
nameserver        10.96.0.10
```

```
cat >> /etc/resolv.conf
nameserver        10.96.0.10
```

kube-dns



```
10-244-2-5     10.244.2.5
10-244-1-5     10.244.1.5
10-244-2-15    10.244.2.15
```

```
cat >> /etc/hosts
```

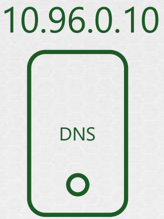
```
cat >> /etc/hosts
```

```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

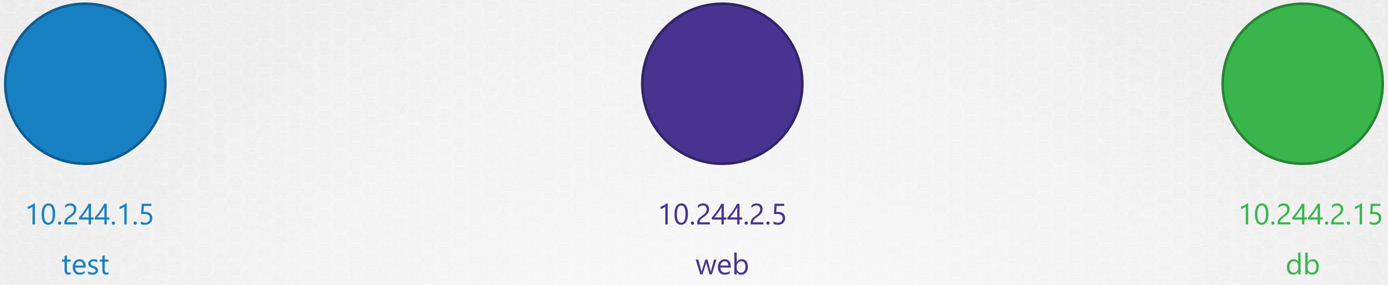
```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

CoreDNS



```
10-244-2-5     10.244.2.5
10-244-1-5     10.244.1.5
10-244-2-15    10.244.2.15
```



```
cat >> /etc/hosts
```

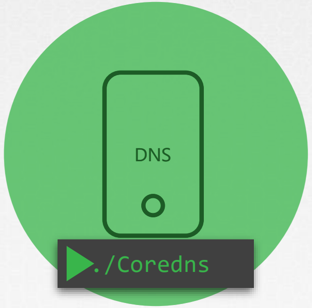
```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

```
cat >> /etc/hosts
```

```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

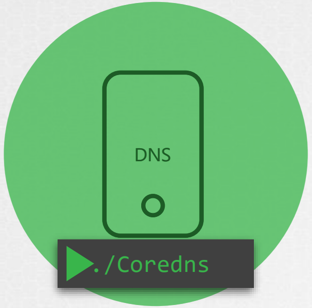
```
cat >> /etc/resolv.conf
nameserver          10.96.0.10
```

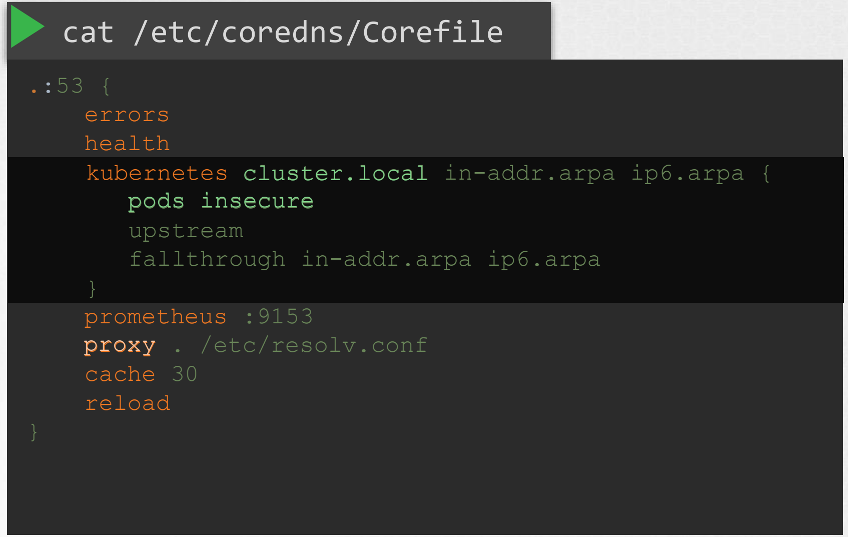
CoreDNS



```
./Coredns
```

CoreDNS



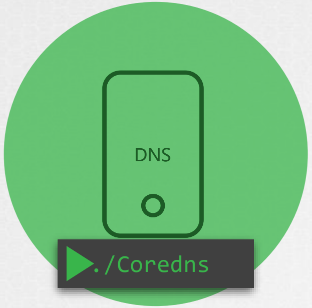


```
cat /etc/coredns/Corefile
```

```
.:53 {
    errors
    health
    kubernetes cluster.local in-addr.arpa ip6.arpa {
       pods insecure
       upstream
       fallthrough in-addr.arpa ip6.arpa
    }
    prometheus :9153
    proxy . /etc/resolv.conf
    cache 30
    reload
}
```

10-244-1-5	default	pod	cluster.local	10.244.1.5

CoreDNS



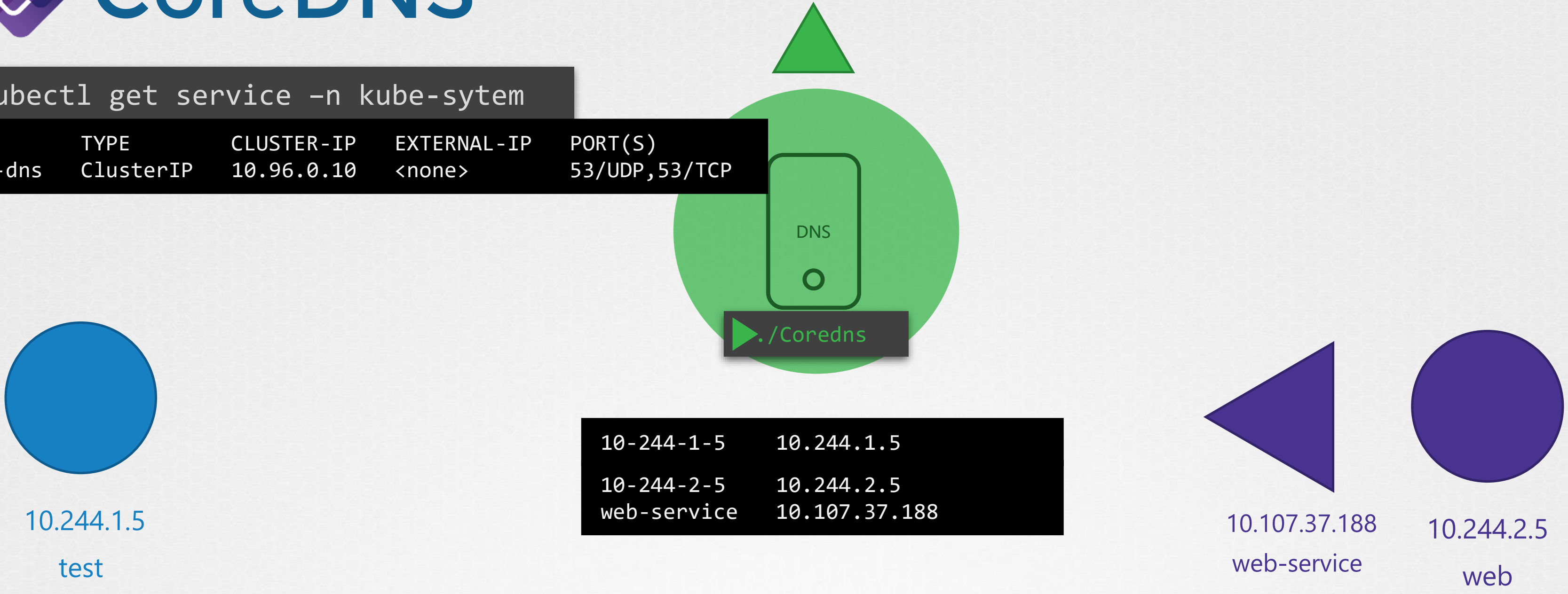
```
▶ cat /etc/coredns/Corefile
.:53 {
    errors
    health
    kubernetes cluster.local in-addr.arpa ip6.arpa {
       pods insecure
       upstream
       fallthrough in-addr.arpa ip6.arpa
    }
    prometheus :9153
    proxy . /etc/resolv.conf
    cache 30
    reload
}
```

```
▶ kubectl get configmap -n kube-system
NAME                                DATA      AGE
coredns                             1         168d
```

CoreDNS

```
kubectl get service -n kube-sytem
```

```
NAME        TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)
kube-dns    ClusterIP   10.96.0.10    <none>        53/UDP,53/TCP
```



```
10-244-1-5      10.244.1.5

10-244-2-5      10.244.2.5
web-service     10.107.37.188
```

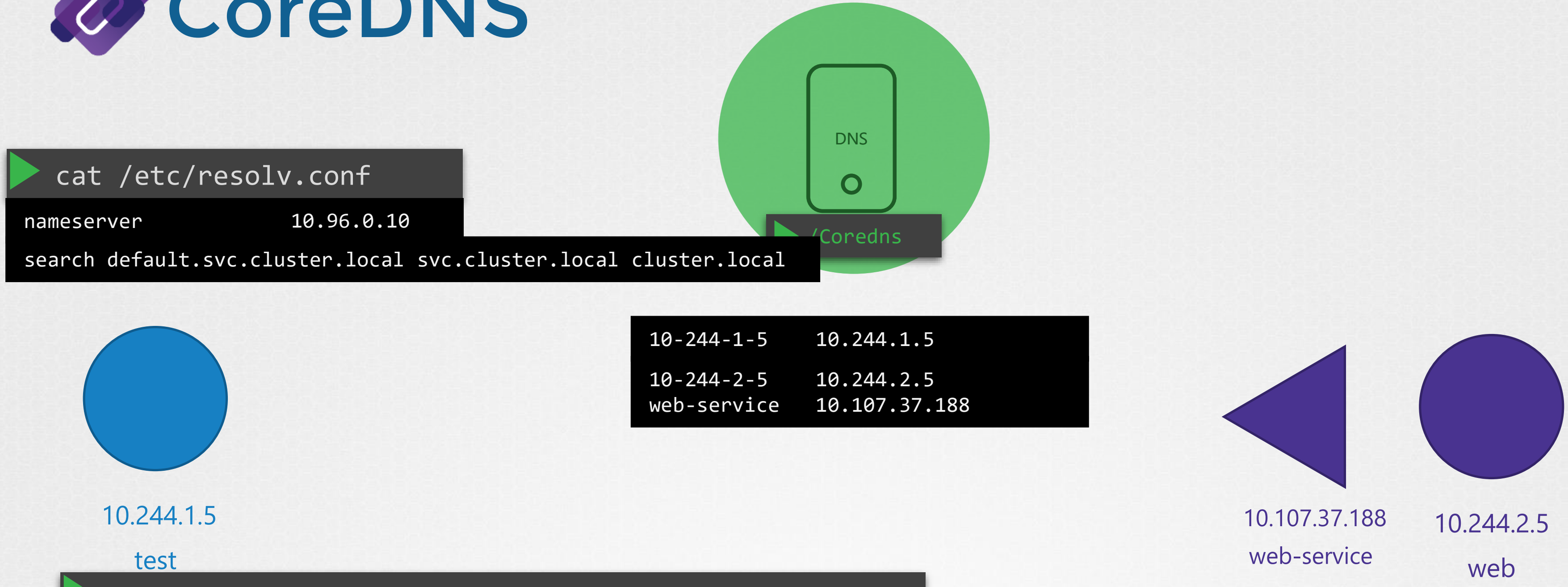
```
cat /etc/resolv.conf
```

```
nameserver              10.96.0.10
```

```
cat /var/lib/kubelet/config.yaml
```

```
...
clusterDNS:
- 10.96.0.10
clusterDomain: cluster.local
```

CoreDNS



```
▶ cat /etc/resolv.conf
nameserver           10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
```

```
▶ /Coredns
```

```
10-244-1-5    10.244.1.5
10-244-2-5    10.244.2.5
web-service   10.107.37.188
```

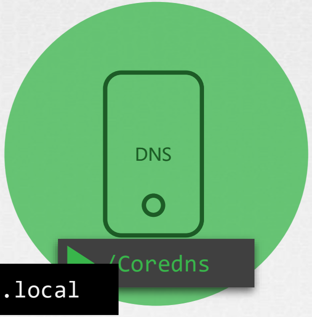
```
▶ host web-service
web-service.default.svc.cluster.local has address 10.97.206.196
```

```
▶ host web-service.default
web-service.default.svc.cluster.local has address 10.97.206.196
```

```
▶ host web-service.default.svc
web-service.default.svc.cluster.local has address 10.97.206.196
```

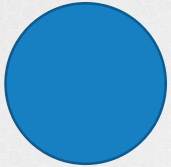
CoreDNS

```
▶ cat /etc/resolv.conf
nameserver          10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
```

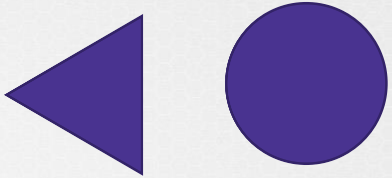


```
▶ /Coredns
```

```
10-244-1-5     10.244.1.5
10-244-2-5     10.244.2.5
web-service    10.107.37.188
```


10.244.1.5

test


10.107.37.188

web-service

10.244.2.5

web

```
▶ host web-service
web-service.default.svc.cluster.local has address 10.97.206.196
```

```
▶ host 10-244-2-5
Host 10-244-2-5 not found: 3(NXDOMAIN)
```

```
▶ host 10-244-2-5.default.pod.cluster.local
10-244-2-5.default.pod.cluster.local has address 10.244.2.5
```