쿠버네티스 네트워크 발전단계

발표자: 최성욱

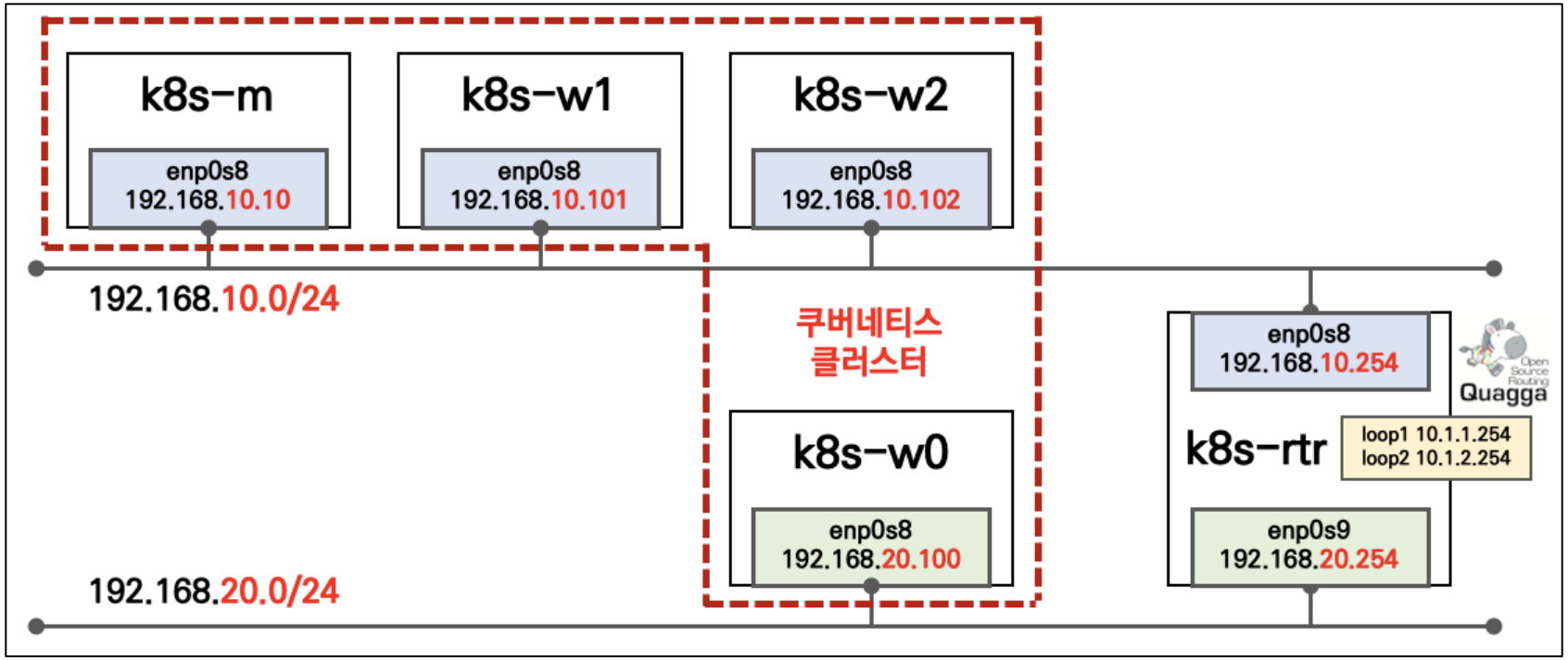
csw19591@gmail.com

목차

실습자료

■ 발표자료에서 사용하는 예제는 https://github.com/choisungwook/facebook-meetup에 있습니다.

■ 쿠버네티스 환경구성은 vagrant를 이용했습니다.



[출처: KANS 스터디]

컨테이너 네트워크 원리

- 네트워크 네임스페이스

1. 컨테이너 네트워크 원리

■ 컨테이너는 **프로세스**

■ 컨테이너 프로세스 검색 예제

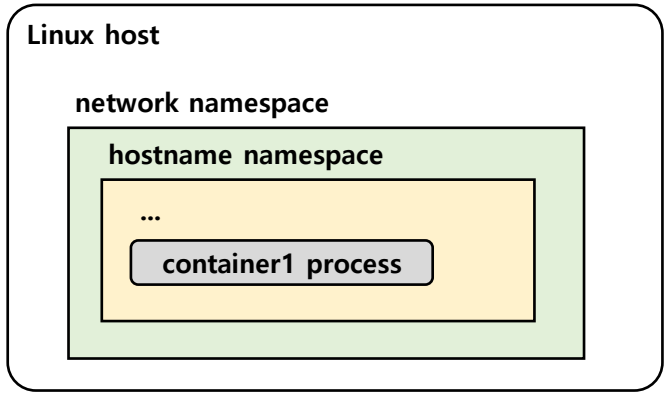
```
# 컨테이너 생성 생성
$ docker run -d --rm --name red nginx:latest
```

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# docker top red
UID                 PID                 PPID                C                   STIME               TTY
root                26482               26460               0                   20:12               ?
systemd+            26550               26482               0                   20:12               ?
systemd+            26551               26482               0                   20:12               ?
(🍒 |Cilium-k8s:default) root@k8s-m:~# ps aux | grep 26482
root       26482  0.0  0.2   8856  5280 ?        Ss   20:12   0:00 nginx: master process nginx -g daemon off;
root       29161  0.0  0.0   8548  2404 pts/0    S+   20:18   0:00 grep 26482
```

1. 컨테이너 네트워크 원리

■ 컨테이너는 커널 리소스가 격리되어 컨테이너마다 자신의 환경을 소유

- cgroup, 네임스페이스(namespace)를 사용하여 리소스 격리



■ 네임스페이스 출력 예제

- 디폴트로 프로세스가 실행되면 init프로세스 네임스페이스를 사용

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# ls -l /proc/1/ns
total 0
lrwxrwxrwx 1 root root 0 Mar  9 19:07 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Mar  9 20:22 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Mar  9 19:07 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Mar  9 20:22 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Mar  9 19:07 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Mar  9 20:22 pid_for_children -> 'pid:[402653
lrwxrwxrwx 1 root root 0 Mar  9 20:22 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Mar  9 20:22 time_for_children -> 'time:[402
lrwxrwxrwx 1 root root 0 Mar  9 20:22 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Mar  9 20:22 uts -> 'uts:[4026531838]'
```

[init프로세스 네임스페이스 목록]

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# /bin/bash
root@k8s-m:~# ls -l /proc/$$/ns
total 0
lrwxrwxrwx 1 root root 0 Mar  9 20:37 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 pid_for_children -> 'pid:[402653
lrwxrwxrwx 1 root root 0 Mar  9 20:37 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 time_for_children -> 'time:[4026
lrwxrwxrwx 1 root root 0 Mar  9 20:37 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Mar  9 20:37 uts -> 'uts:[4026531838]'
```

[새로운 프로세스 네임스페이스 목록]

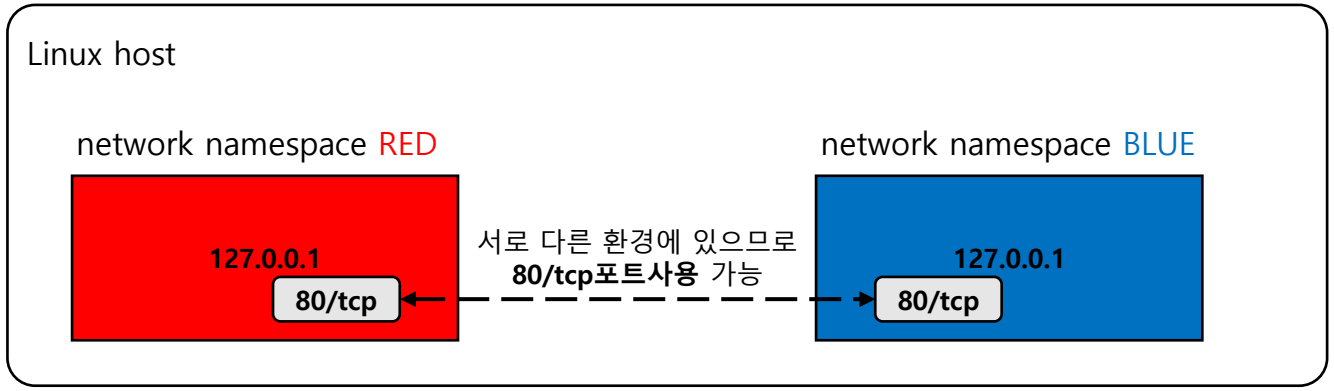
```
(🍒 |Cilium-k8s:default) root@k8s-m:~# ls -l /proc/26482/ns
total 0
lrwxrwxrwx 1 root root 0 Mar  9 20:34 cgroup -> 'cgroup:[4026532664]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 ipc -> 'ipc:[4026532604]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 mnt -> 'mnt:[4026532602]'
lrwxrwxrwx 1 root root 0 Mar  9 20:12 net -> 'net:[4026532607]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 pid -> 'pid:[4026532605]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 pid_for_children -> 'pid:[4026532605]
lrwxrwxrwx 1 root root 0 Mar  9 20:34 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 time_for_children -> 'time:[402653183
lrwxrwxrwx 1 root root 0 Mar  9 20:34 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Mar  9 20:34 uts -> 'uts:[4026532603]'
```

[도커컨테이너 네임스페이스 목록]

1. 컨테이너 네트워크 원리

■ 컨테이너는 네트워크 네임스페이스가 다르므로 다른 컨테이너의 네트워크 환경 영향 X

```
$ docker run -d --rm --name red nginx:latest
$ docker run -d --rm --name blue nginx:latest
```



네트워크 네임스페이스는
어떻게
서로 통신할까?

1. 컨테이너 네트워크 원리

■ 네트워크 네임스페이스 중요 키워드 3개

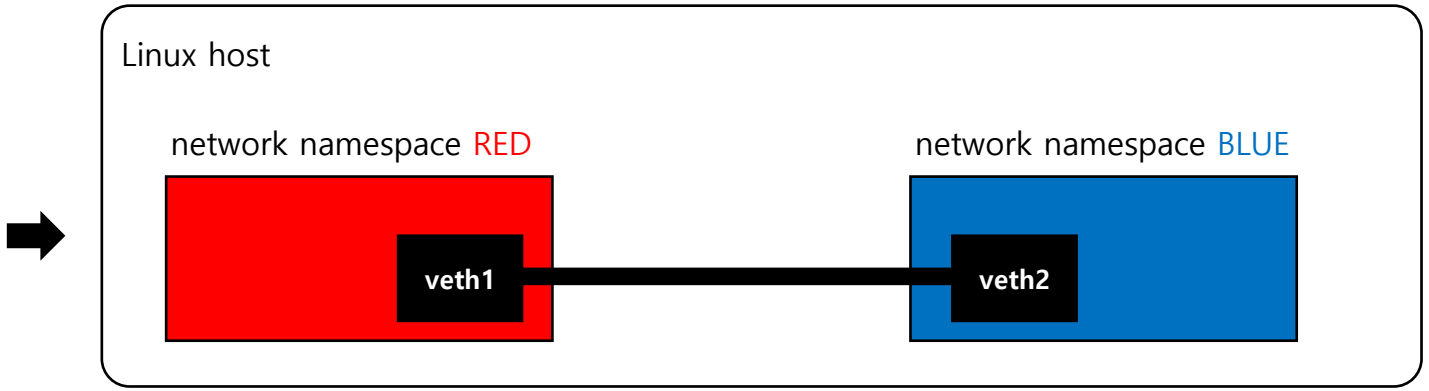
- 네트워크 인터페이스
- iptables
- route table

1. 컨테이너 네트워크 원리

■ 네트워크 (가상)인터페이스를 사용하여 **네트워크 네임스페이스 연결**하고 **IP를 할당**

- 네트워크 네임스페이스 통신간 통로는 가상 이더넷 인터페이스 veth로 연결
  - veth는 쌍으로 생성

```
# 네트워크 네임스페이스 생성
$ ip netns add RED
$ ip netns add BLUE

# veth를 네트워크네임스페이스에 연결
$ ip link add veth1 type veth peer name veth2
$ ip link set veth1 netns RED
$ ip link set veth2 netns BLUE

# ip할당
$ ip netns exec RED ip addr add 11.11.11.2/24 dev veth1
$ ip netns exec BLUE ip addr add 11.11.11.3/24 dev veth2

# 인터페이스 활성화
$ ip netns exec RED ip link set dev veth1 up
$ ip netns exec BLUE ip link set dev veth2 up
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/network_namespace/Readme.md

유투브 데모링크: https://youtu.be/fQa5FDqGOmw

1. 컨테이너 네트워크 원리

■ 네트워크 네임스페이스 연결 확인

```
$ ip netns exec RED ip -c -br addr show
$ ip netns exec BLUE ip -c -br addr show
```

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# ip netns exec RED ip -c -br addr show
lo               DOWN
veth1@if25       UP             11.11.11.2/24 fe80::c2e:8cff:feb1:3d2e/64
(🍒 |Cilium-k8s:default) root@k8s-m:~# ip netns exec BLUE ip -c -br addr show
lo               DOWN
veth2@if26       UP             11.11.11.3/24 fe80::bcde:45ff:fe73:642/64
```

■ ping 통신

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# ip netns exec RED ping 11.11.11.3 -c 2
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data.
64 bytes from 11.11.11.3: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from 11.11.11.3: icmp_seq=2 ttl=64 time=0.027 ms

--- 11.11.11.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.027/0.028/0.029/0.001 ms
(🍒 |Cilium-k8s:default) root@k8s-m:~# ^C
(🍒 |Cilium-k8s:default) root@k8s-m:~# ip netns exec BLUE ping 11.11.11.2 -c 2
PING 11.11.11.2 (11.11.11.2) 56(84) bytes of data.
64 bytes from 11.11.11.2: icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 11.11.11.2: icmp_seq=2 ttl=64 time=0.025 ms

--- 11.11.11.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1019ms
rtt min/avg/max/mdev = 0.020/0.022/0.025/0.002 ms
```

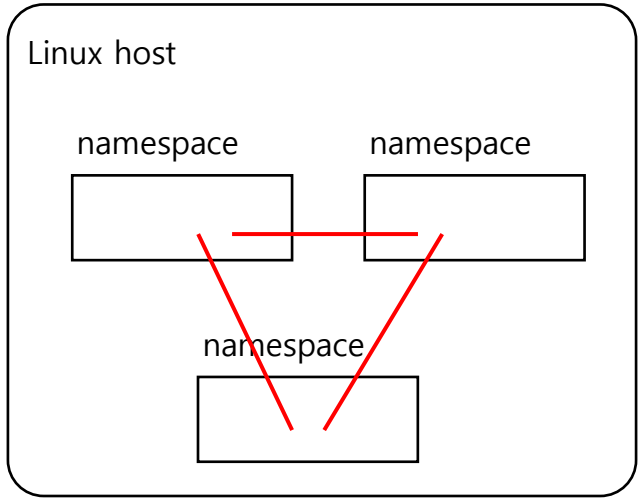
RED → BLUE

BLUE → RED

1. 컨테이너 네트워크 원리

■ 다수의 네트워크 네임스페이스 연결이 필요하다면?

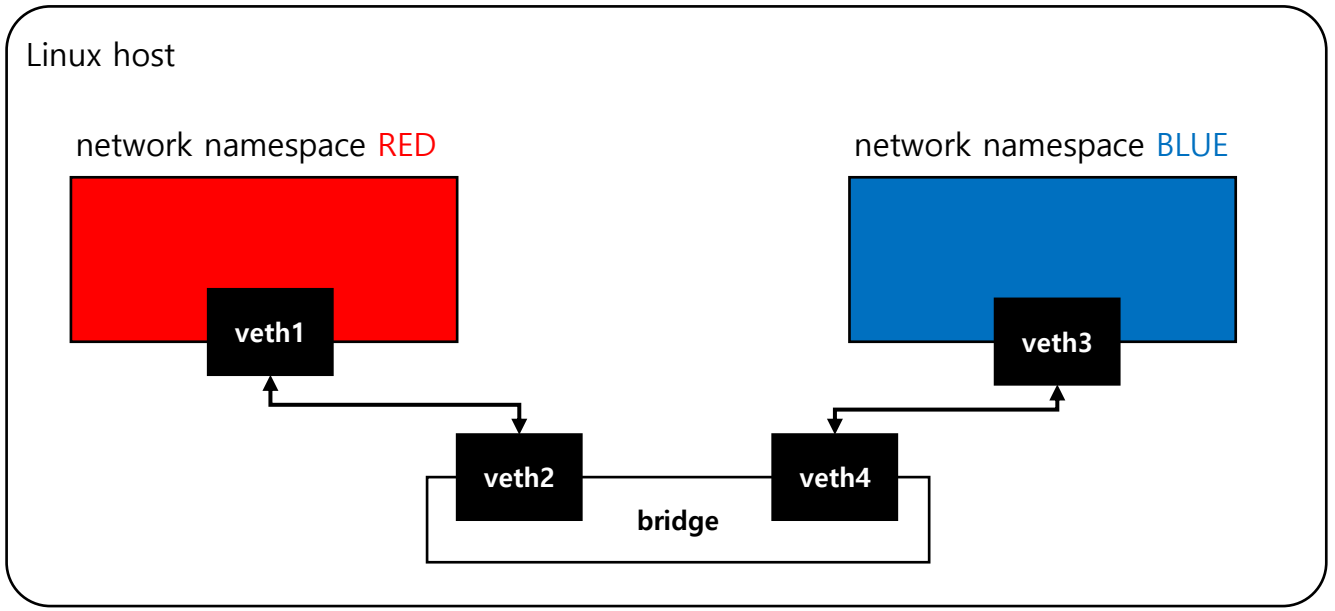
- n(n-1)/2 만큼 인터페이스 연결필요



1. 컨테이너 네트워크 원리

■ 가상 브릿지를 이용하여 해결

```
# 기존 예제 삭제
$ ip netns del RED
$ ip netns del BLUE

# 네트워크 네임스페이스 생성
$ ip link add br0 type bridge
$ ip addr add 11.11.11.1/24 dev br0

# 네트워크 네임스페이스 생성
$ ip netns add RED
$ ip netns add BLUE

# 네트워크 네임스페이스와 브릿지연결
$ ip link add veth1 type veth peer name veth2
$ ip link add veth3 type veth peer name veth4
$ ip link set veth1 netns RED
$ ip link set veth3 netns BLUE
$ ip link set veth2 master br0
$ ip link set veth4 master br0

# ip할당
$ ip netns exec RED ip addr add 11.11.11.2/24 dev veth1
$ ip netns exec BLUE ip addr add 11.11.11.3/24 dev veth3

# 인터페이스 활성화
$ ip link set br0 up
$ ip link set veth2 up
$ ip link set veth4 up
$ ip netns exec RED ip link set dev veth1 up
$ ip netns exec BLUE ip link set dev veth3 up
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/network_namespace_bridge/Readme.md 유투브 데모링크: https://youtu.be/iWVSZa8RnZc

1. 컨테이너 네트워크 원리

■ 네트워크 네임스페이스 연결 확인

```
$ brctl show
```

```
(🍒 |Cilium-k8s:default) root@k8s-m:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.f2b4d66e93ca       no              veth2
                                                        veth4
```

■ ping 통신

- 다른 네트워크 네임스페이스와 통신 실패

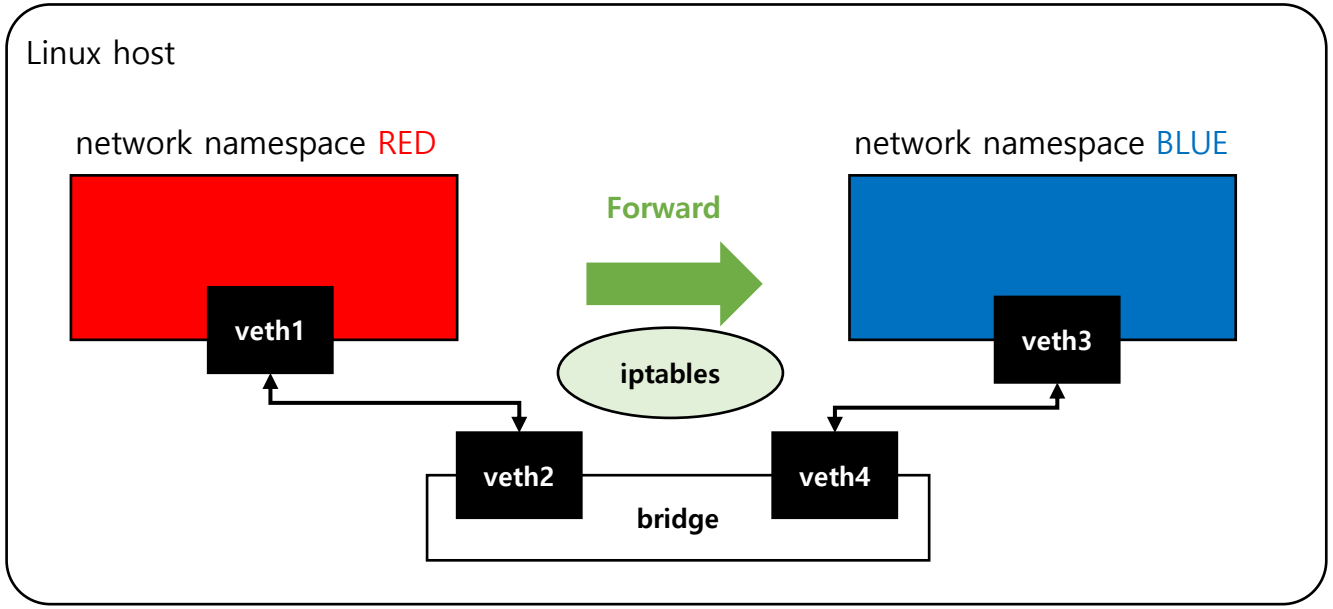
```
root@centos-docker:/home/vagrant# ip netns exec RED ping 11.11.11.2 -c 2
PING 11.11.11.2 (11.11.11.2) 56(84) bytes of data.
64 bytes from 11.11.11.2: icmp_seq=1 ttl=64 time=0.015 ms
64 bytes from 11.11.11.2: icmp_seq=2 ttl=64 time=0.061 ms

--- 11.11.11.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.015/0.038/0.061/0.023 ms
root@centos-docker:/home/vagrant# ip netns exec RED ping 11.11.11.3 -c 2
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data.
From 11.11.11.2 icmp_seq=1 Destination Host Unreachable
From 11.11.11.2 icmp_seq=2 Destination Host Unreachable

--- 11.11.11.3 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1023ms
pipe 2
```

왜 통신이 안될까요?

1. 컨테이너 네트워크 원리

■ iptables Forward 설정 필요

- host입장에서 네트워크 네임스페이스는 다른 네트워크 영역으로 인식하여 iptables Forward룰 준수



1. 컨테이너 네트워크 원리

■ iptables Forward 설정

```
$ iptables --policy FORWARD ACCEPT
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

■ 다시 ping통신 시도하면 성공!

```
root@centos-docker:/home/vagrant# ip netns exec RED ping 11.11.11.3 -c 2
PING 11.11.11.3 (11.11.11.3) 56(84) bytes of data.
64 bytes from 11.11.11.3: icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from 11.11.11.3: icmp_seq=2 ttl=64 time=0.041 ms

--- 11.11.11.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1027ms
rtt min/avg/max/mdev = 0.026/0.033/0.041/0.009 ms
```

```
root@centos-docker:/home/vagrant# ip netns exec BLUE ping 11.11.11.2 -c 2
PING 11.11.11.2 (11.11.11.2) 56(84) bytes of data.
64 bytes from 11.11.11.2: icmp_seq=1 ttl=64 time=0.031 ms
64 bytes from 11.11.11.2: icmp_seq=2 ttl=64 time=0.039 ms

--- 11.11.11.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.031/0.035/0.039/0.004 ms
```

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/network_namespace_bridge/Readme.md 유투브 데모링크: https://youtu.be/iWVSZa8RnZc

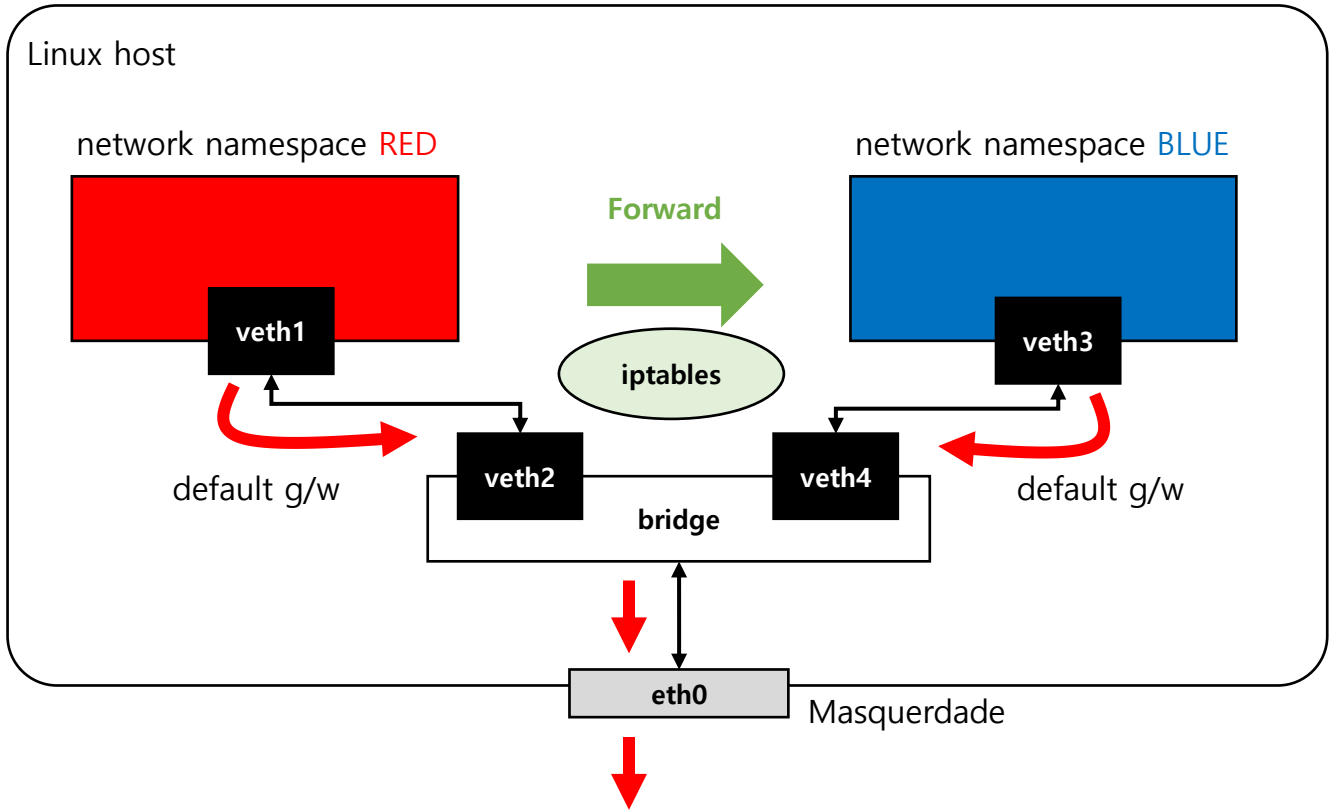
외부 통신은 어떻게 할까요?

1. 컨테이너 네트워크 원리

■ 디폴트게이트웨이를 가상 브릿지로 설정

■ host에서는 iptables Masquerade설정

```
$ ip netns exec RED ip route add default via 11.11.11.1
$ ip netns exec BLUE ip route add default via 11.11.11.1
$ iptables -t nat -A POSTROUTING -s 11.11.11.0/24 -j MASQUERADE
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/network_namespace_gateway/Readme.md

유투브 데모링크: https://youtu.be/_gZyN3tOggo

컨테이너 엔진(예: Docker)은

앞과정을

자동으로 관리해줘요!

쿠버네티스 CNI

2. 쿠버네티스 CNI

■ 쿠버네티스 네트워크도 컨테이너 네트워크처럼 비슷하게 동작

- 네트워크 네임스페이스 생성, route table 관리, iptables 관리 등

■ 네트워크 설정과 동작을 CNI(Container Network Interface)에게 위임

■ CNI는 4가지 네트워킹을 수행해야 함

① 파드의 컨테이너간 통신

② 파드간 통신

③ 파드와 서비스간 통신

④ 외부와 서비스간 통신

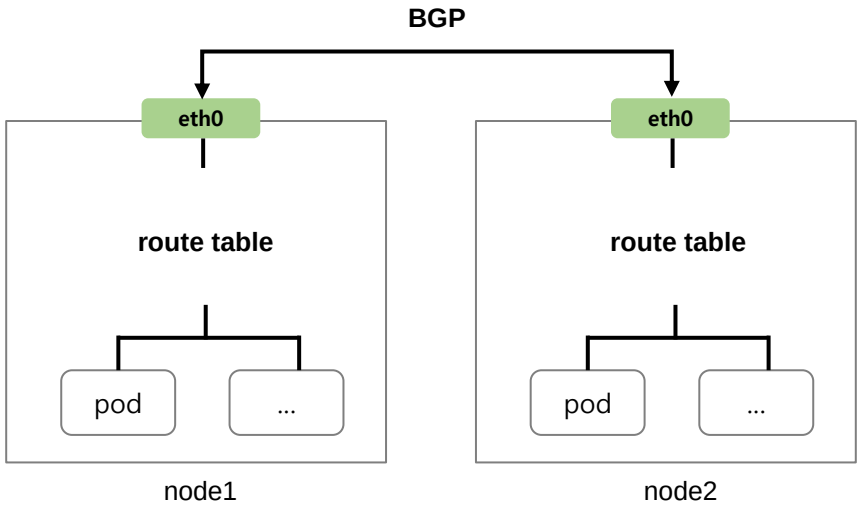
저희는
Calico CNI를
다룰거에요!

Calico CNI 소개

3. Calico CNI

■ calico CNI는 BGP를 이용한 구조

- 각 노드간 라우팅 테이블을 설정하고 공유



■ IPAM 관리

- 쿠버네티스 클러스터의 파드 네트워크 대역

```
(calico-k8s:default) root@k8s-m:~# calicoctl ipam show
+----------+---------------+-----------+------------+--------------+
| GROUPING |     CIDR      | IPS TOTAL | IPS IN USE |   IPS FREE   |
+----------+---------------+-----------+------------+--------------+
| IP Pool  | 172.16.0.0/16 |     65536 | 6 (0%)     | 65530 (100%) |
+----------+---------------+-----------+------------+--------------+
```

[쿠버네티스 클러스터 전체 네트워크 대역]

```
(calico-k8s:default) root@k8s-m:~# kubectl get no
NAME     STATUS   ROLES                  AGE   VERSION
k8s-m    Ready    control-plane,master   26h   v1.22.6
k8s-w0   Ready    <none>                 26h   v1.22.6
k8s-w1   Ready    <none>                 26h   v1.22.6
(calico-k8s:default) root@k8s-m:~# calicoctl ipam show --show-blocks
+----------+-----------------+-----------+------------+--------------+
| GROUPING |      CIDR       | IPS TOTAL | IPS IN USE |   IPS FREE   |
+----------+-----------------+-----------+------------+--------------+
| IP Pool  | 172.16.0.0/16   |     65536 | 6 (0%)     | 65530 (100%) |
| Block    | 172.16.116.0/24 |       256 | 4 (2%)     | 252 (98%)    |
| Block    | 172.16.158.0/24 |       256 | 1 (0%)     | 255 (100%)   |
| Block    | 172.16.34.0/24  |       256 | 1 (0%)     | 255 (100%)   |
+----------+-----------------+-----------+------------+--------------+
```

[각 노드간 네트워크 대역]

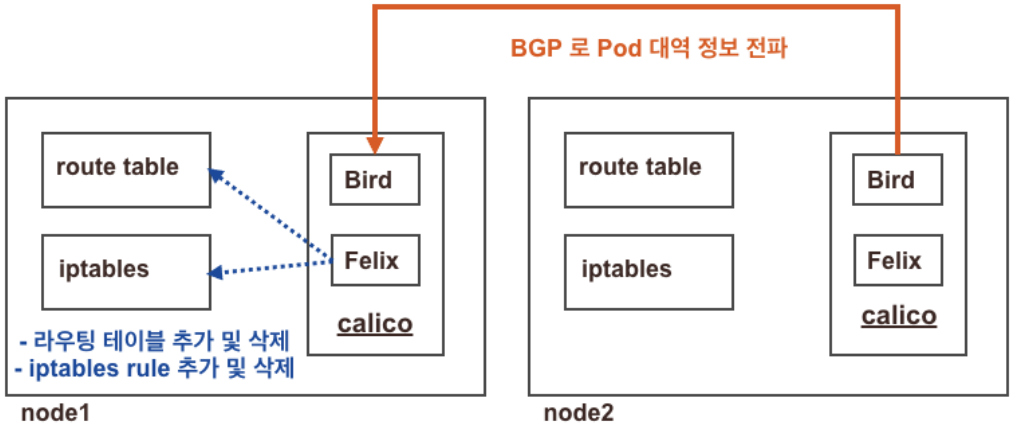
3. Calico CNI

■ calico는 데몬셋으로 모든 파드에 설치

```
(🍅 |Service-k8s:default) root@k8s-m:~# kubectl get ds -n kube-system calico-node
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
calico-node   4         4         3       4            3           kubernetes.io/os=linux   131m
```

■ 핵심 모듈

- Bird: BGP프로토콜을 이용하여 라우팅 테이블 전파
- Felix: 라우팅 테이블, iptables 관리

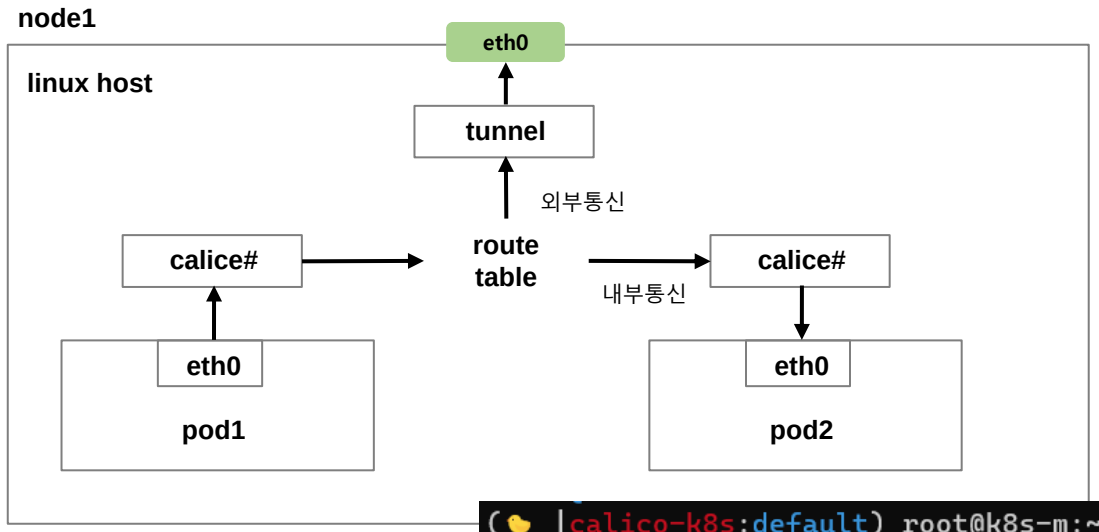


[이미지 출처: KANS 스터디 공유자료]

3. Calico CNI

■ 파드가 생성되면 calice 인터페이스(veth)가 생성되고 호스트와 연결

- 각 노드간 라우팅 테이블을 설정하고 공유




```
(⎈ |calico-k8s:default) root@k8s-m:~# ip -c -br addr show | grep cali
calibc3cd7dca6b@if4 UP             fe80::ecee:eeff:feee:eeee/64
cali9cd82c0c4e2@if4 UP             fe80::ecee:eeff:feee:eeee/64
caliaeba7328267@if4 UP             fe80::ecee:eeff:feee:eeee/64
```

CNI 네트워킹 첫번째 과제

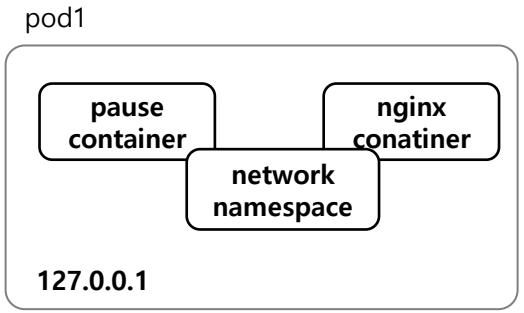
- 파드 컨테이너간 통신

4. 첫번째 네트워킹 과제: 컨테이너간 통신

■ 파드가 생성되면 pause컨테이너가 네트워크 네임스페이스가 생성

- pause컨테이너는 파드 spec에 없어도 자동생성

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: nginx
    image: nginx
```



■ CRI 클라이언트 명령어로 pause컨테이너 확인

```
(🍒 |Cilium-k8s:default) root@k8s-m:~/namespace# docker ps | grep pause
9f8e31bff34b   k8s.gcr.io/pause:3.5   "/pause"                 12 hours ago   Up 12 hours             k8s_POD_netperf-client_default_56a91105-3cde-4f22-ba1c-d5b5d6410334_0
68ac75849f54   k8s.gcr.io/pause:3.5   "/pause"                 23 hours ago   Up 23 hours             k8s_POD_kube-controller-manager-k8s-m_kube-system_bf233e4dd6e2538a0b9d2996dbcc2fb2_4
8dd6911a5edb   k8s.gcr.io/pause:3.5   "/pause"                 23 hours ago   Up 23 hours             k8s_POD_etcd-k8s-m_kube-system_629f7163c4e43cdb7f0942a4a5a70bbb_4
85232c3e16dd   k8s.gcr.io/pause:3.5   "/pause"                 23 hours ago   Up 23 hours             k8s_POD_coredns-78fcd69978-w7fhf_kube-system_beb5dd1d-e7cc-4608-bbc0-3e3f16c41670_11
```

[pause컨테이너 확인]

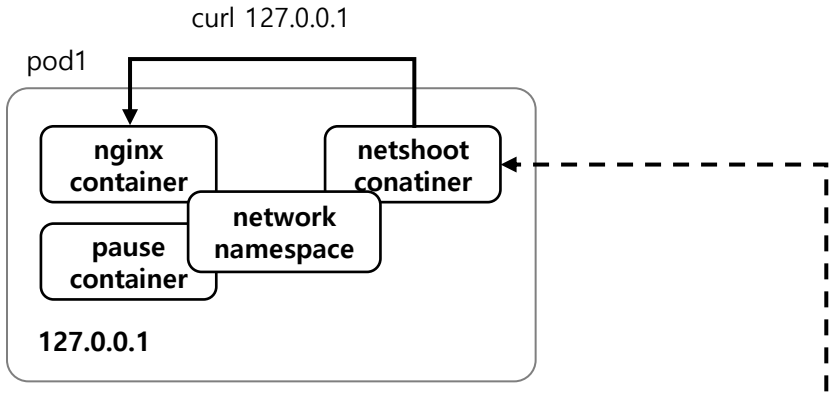
4. 첫번째 네트워킹 과제: 컨테이너간 통신

■ 파드에 속한 컨테이너는 다른 컨테이너에서 오픈한 포트를 127.0.0.1:<port>로 접근 가능

예) netshoot컨테이너에서 127.0.0.1:80으로 접근가능

■ 다른 컨테이너가 사용중인 포트 사용불가

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: nginx
    image: nginx
  - name: netshoot
    image: nicolaka/netshoot
```




```
(⎈ |Cilium-k8s:default) root@k8s-m:~/namespace# kubectl exec -it same-namespace -c netshoot -- curl 127.0.0.1:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/pod_namespace/pod.yaml

유투브 데모링크: https://youtu.be/jwLHV-F5zCU

CNI 네트워킹 두번째 과제

- 파드간 통신 - 같은 노드

5. 두번째 네트워킹 과제: (같은 노드)파드간 통신

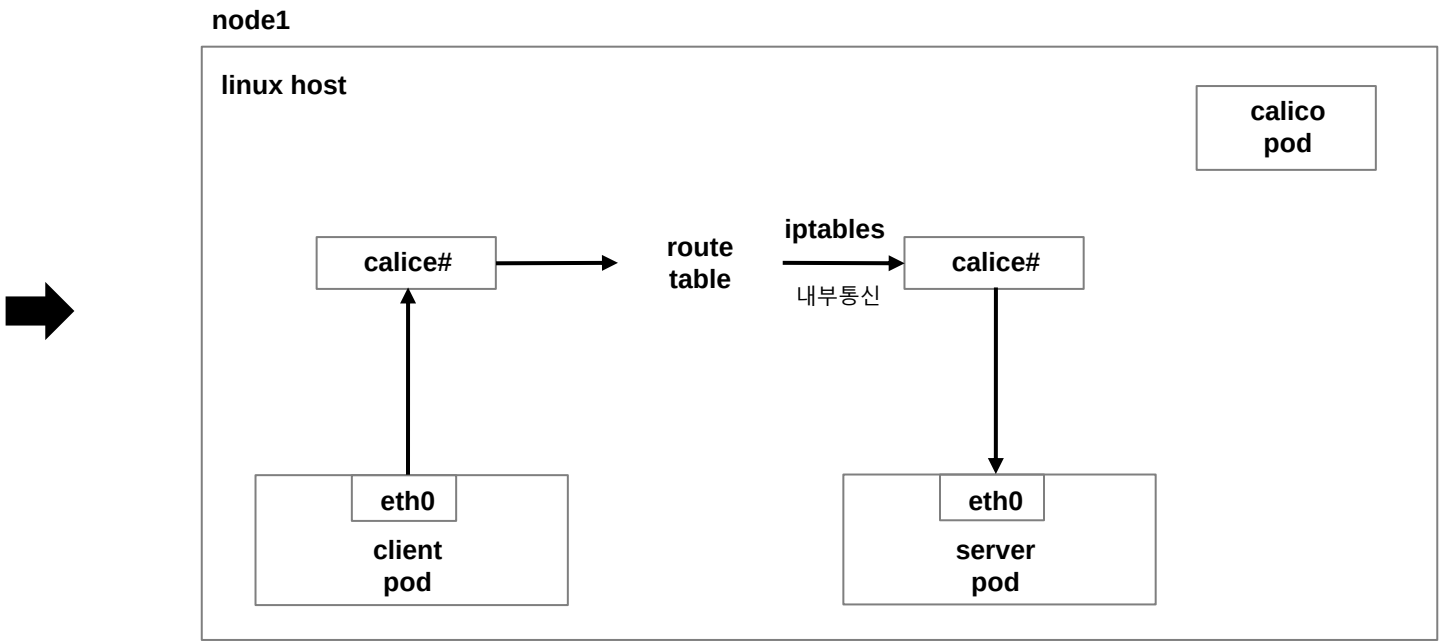
■ 통신순서

① 파드의 외부 트래픽은 calice인터페이스를 통해 커널로 전달

② 커널은 라우팅 테이블에 따라 내부통신으로 라우팅

③ iptables forward룰에 따라 파드까지 트래픽 전달

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: server
spec:
  nodeName: k8s-w1
  containers:
  - name: netshoot
    image: nicolaka/netshoot
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo
hello; sleep 10;done"]
---
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  nodeName: k8s-w1
  containers:
  - name: netshoot
    image: nicolaka/netshoot
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo
hello; sleep 10;done"]
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/pod_pod/same_node.yaml

눈으로 보면

어려우니

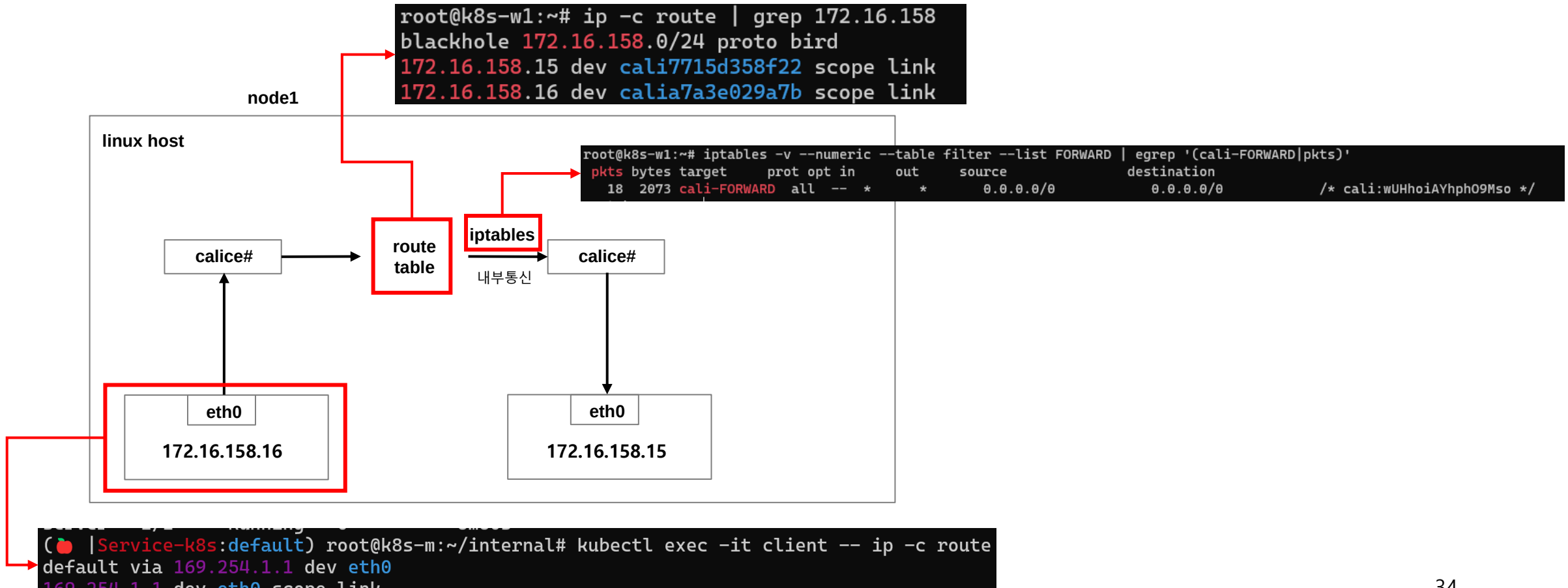
직접 디버깅해봐요!

5. 두번째 네트워킹 과제: (같은 노드)파드간 통신

■ 설정 디버깅

```
(🍎 |Service-k8s:default) root@k8s-m:~/internal# kubectl get po -o wide
NAME     READY   STATUS    RESTARTS   AGE     IP               NODE
client   1/1     Running   0          8m43s   172.16.158.16   k8s-w1
server   1/1     Running   0          8m43s   172.16.158.15   k8s-w1
```

```
root@k8s-w1:~# ip -c route | grep 172.16.158
blackhole 172.16.158.0/24 proto bird
172.16.158.15 dev cali7715d358f22 scope link
172.16.158.16 dev calia7a3e029a7b scope link
```



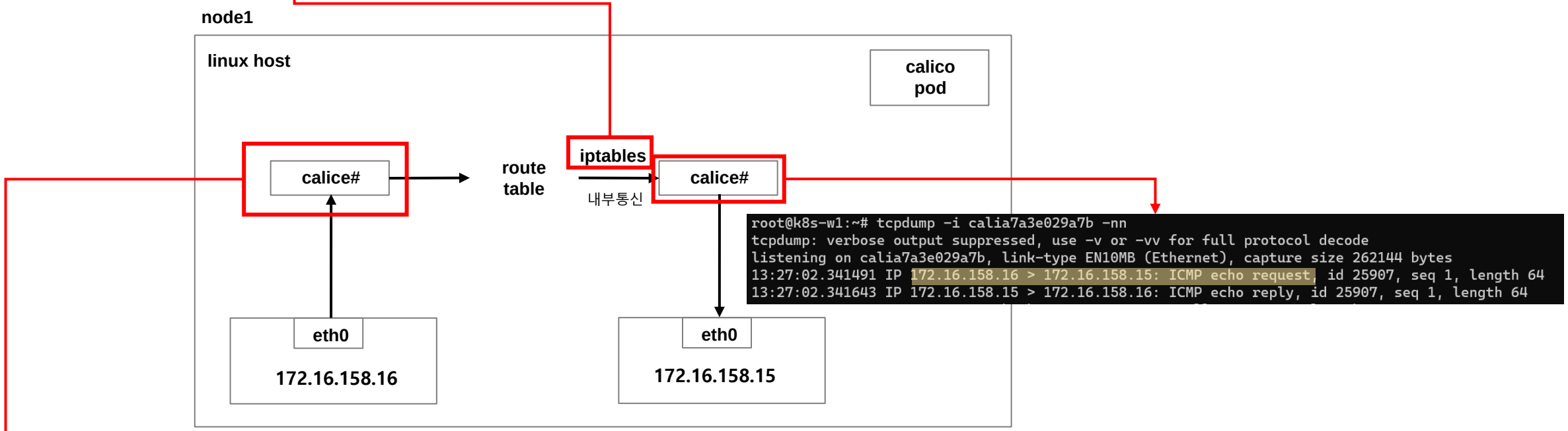

```
root@k8s-w1:~# iptables -v --numeric --table filter --list FORWARD | egrep '(cali-FORWARD|pkts)'
 pkts bytes target     prot opt in     out     source               destination
   18  2073 cali-FORWARD  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:wUHhoiAYhphO9Mso */
```

```
(🍎 |Service-k8s:default) root@k8s-m:~/internal# kubectl exec -it client -- ip -c route
default via 169.254.1.1 dev eth0
169.254.1.1 dev eth0 scope link
```

5. 두번째 네트워킹 과제: (같은 노드)파드간 통신

■ 패킷 디버깅

```
root@k8s-w1:~# iptables -v --numeric --table filter --list FORWARD | egrep '(cali-FORWARD|pkts)'
 pkts bytes target     prot opt in     out     source               destination
   18  2073 cali-FORWARD  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* cali:wUHhoiAYhphO9Mso */
```




```
root@k8s-w1:~# tcpdump -i calia7a3e029a7b -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on calia7a3e029a7b, link-type EN10MB (Ethernet), capture size 262144 bytes
13:27:02.341491 IP 172.16.158.16 > 172.16.158.15: ICMP echo request, id 25907, seq 1, length 64
13:27:02.341643 IP 172.16.158.15 > 172.16.158.16: ICMP echo reply, id 25907, seq 1, length 64
```

```
root@k8s-w1:~# tcpdump -i cali7715d358f22 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on cali7715d358f22, link-type EN10MB (Ethernet), capture size 262144 bytes
13:11:57.532942 ARP, Request who-has 172.16.158.15 tell 10.0.2.15, length 28
13:11:57.532996 ARP, Reply 172.16.158.15 is-at ea:4b:c4:79:ac:c1, length 28
13:11:57.532998 IP 172.16.158.16 > 172.16.158.15: ICMP echo request, id 5423, seq 1, length 64
13:11:57.533010 ARP, Request who-has 169.254.1.1 tell 172.16.158.15, length 28
13:11:57.533012 ARP, Reply 169.254.1.1 is-at ee:ee:ee:ee:ee:ee, length 28
13:11:57.533013 IP 172.16.158.15 > 172.16.158.16: ICMP echo reply, id 5423, seq 1, length 64
13:11:58.538646 IP 172.16.158.16 > 172.16.158.15: ICMP echo request, id 5423, seq 2, length 64
13:11:58.538704 IP 172.16.158.15 > 172.16.158.16: ICMP echo reply, id 5423, seq 2, length 64
```

유투브 데모링크: https://youtu.be/frZYJH2aJRI

그런데!

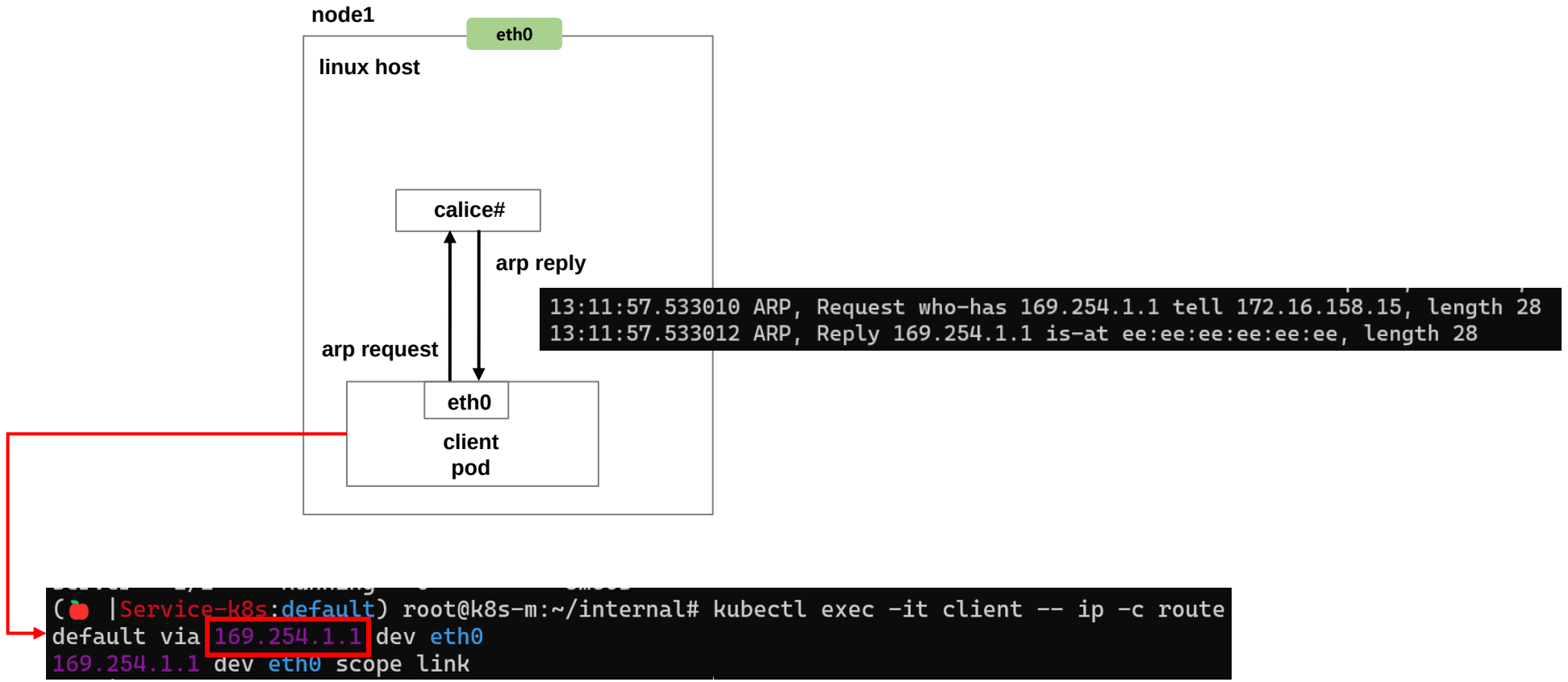
어떻게 pod 트래픽이

calice인터페이스로 전달이 될까요?

5. 두번째 네트워킹 과제: (같은 노드)파드간 통신

■ proxy arp를 이용

- calice가 설정한 파드 디폴트게이트웨이(169.254.1.1)의 맥주소를 calice 맥주소로 설정

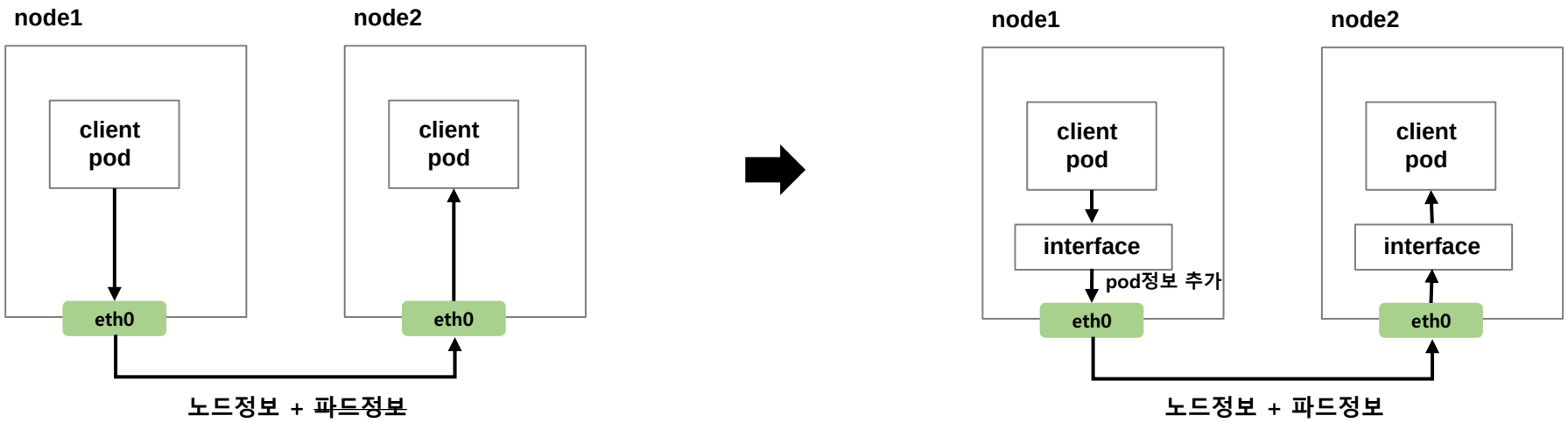


CNI 네트워킹 두번째 과제

- 파드간 통신 - 다른 노드

6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

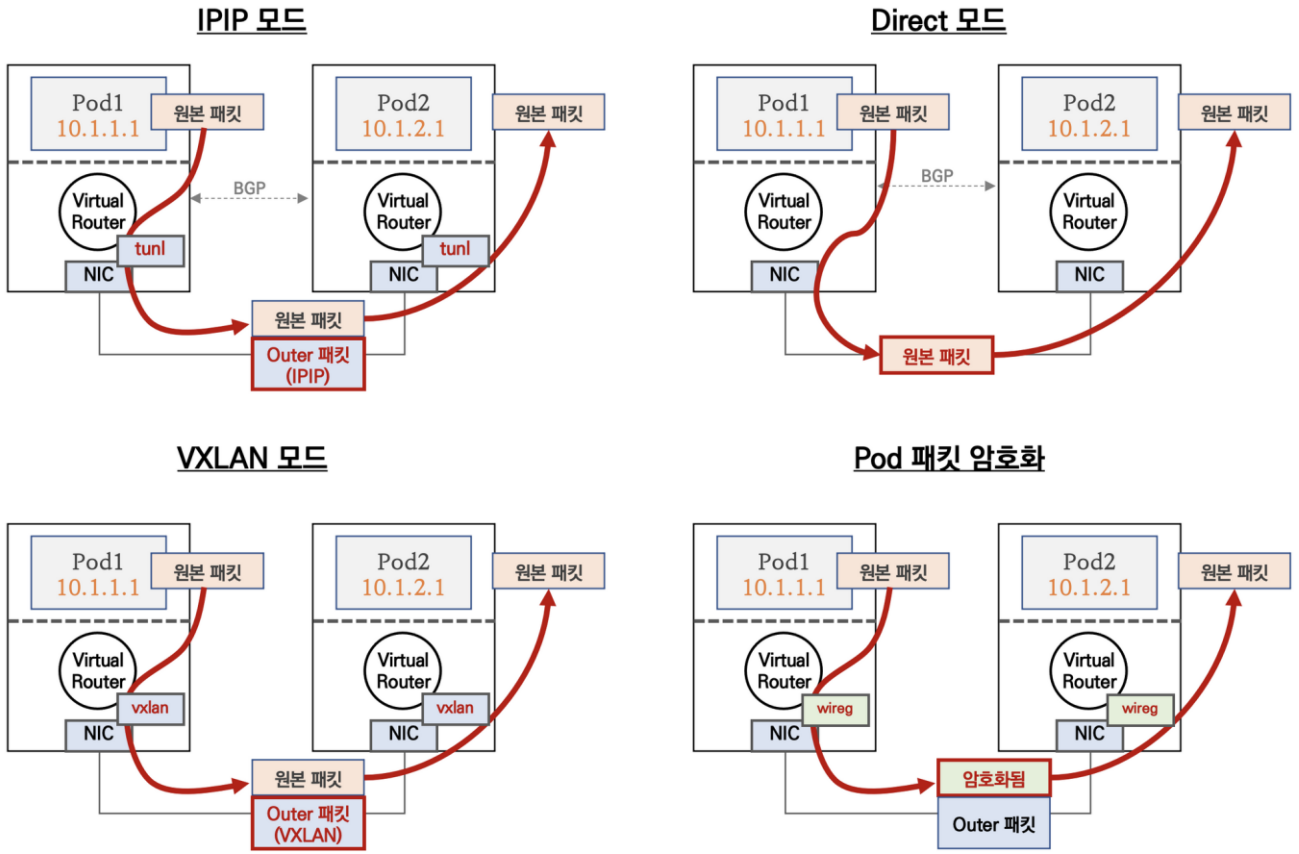
■ 파드 정보를 보존하기 위해 통신간 중간 인터페이스 이용



6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

■ 다른 노드간 통신

- 모드마다 통신방법이 상이(디폴트 IPIP모드)
- 환경에 따라 동작하지 않는 모드 존재(퍼블릭 클라우드 등)



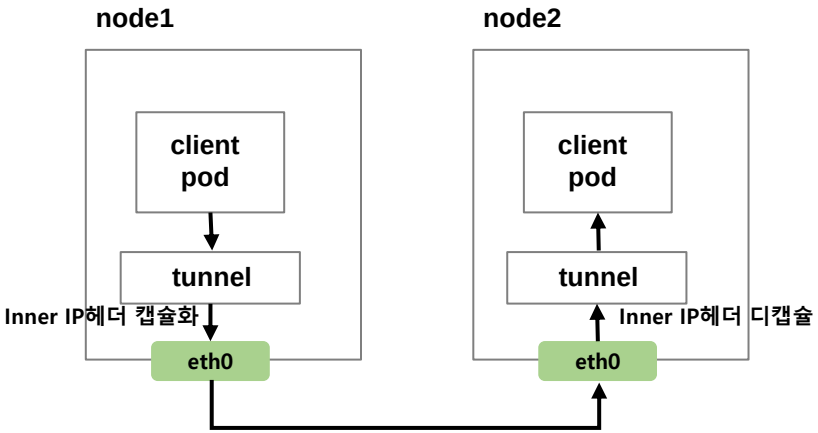
이 발표에서는

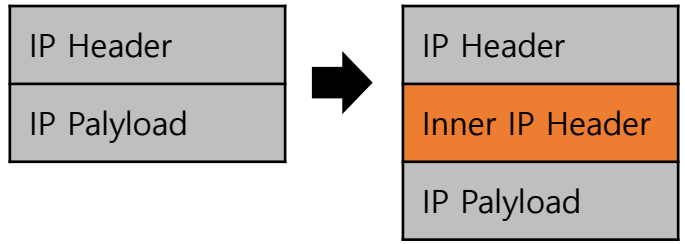
IPIP모드만 다룹니다.

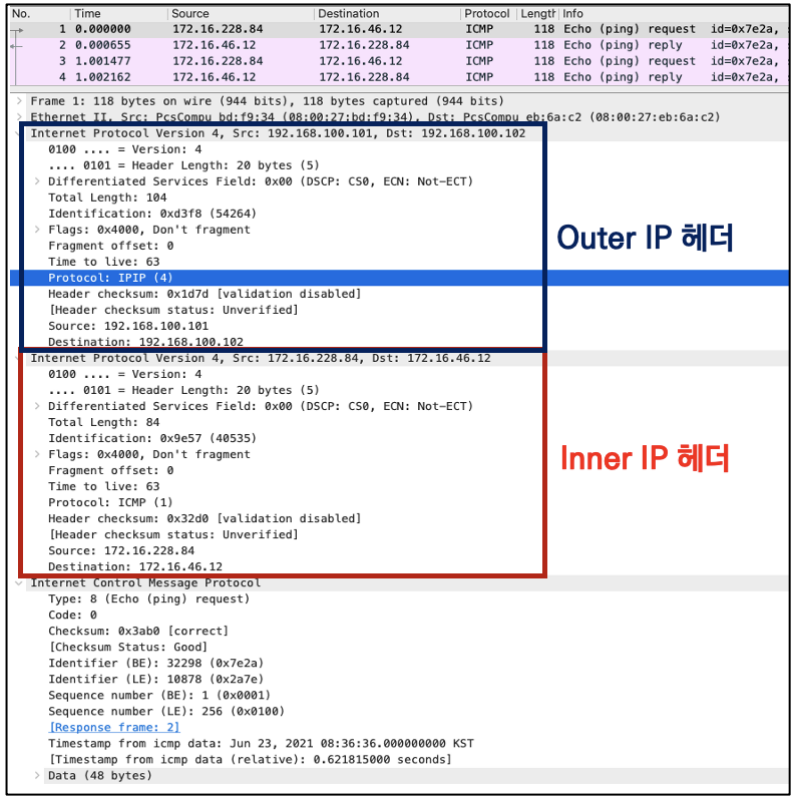
6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

■ IPIP모드 동작원리

- 다른 노드 간의 파드 통신에 tunnel인터페이스가 관여
- tunnel인터페이스는 Inner IP헤더(pod정보)를 추가



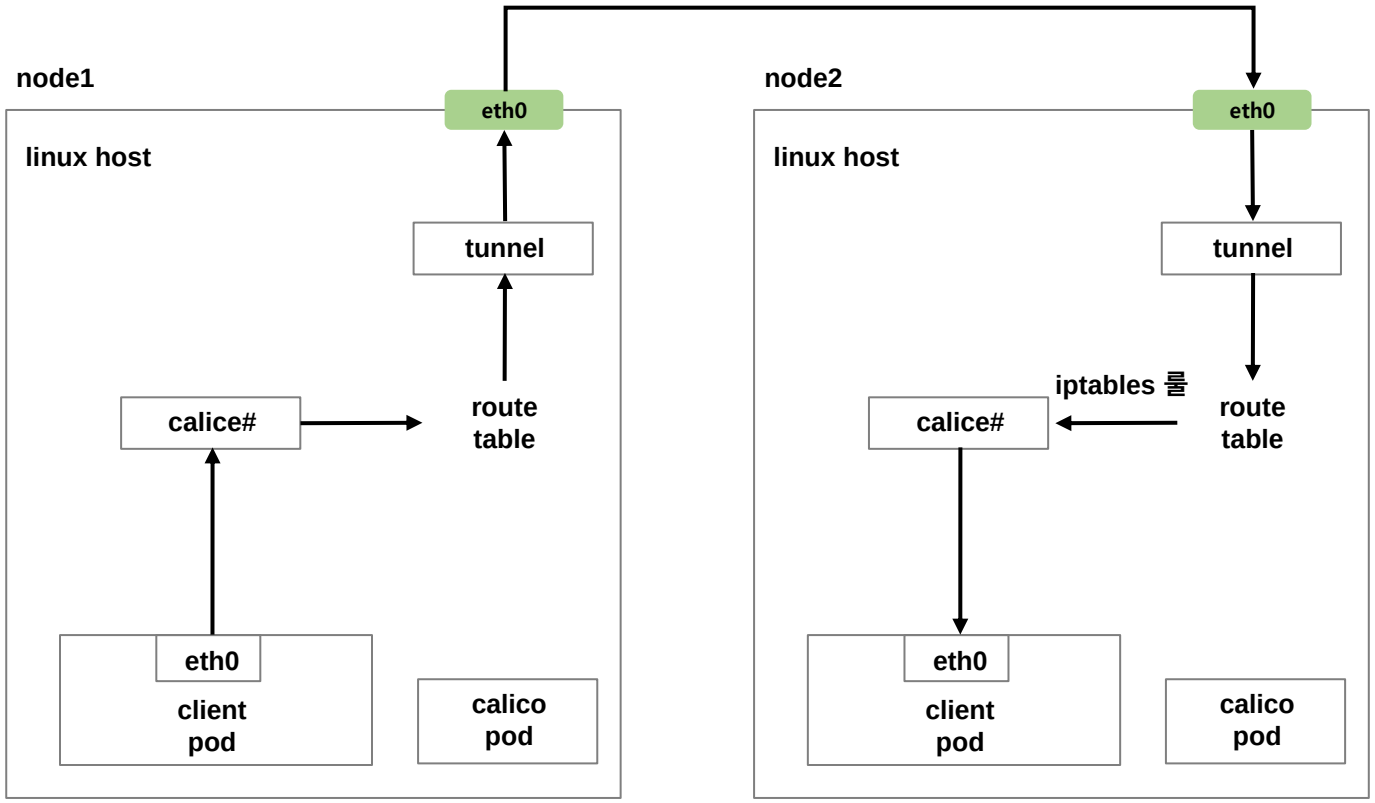




6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

■ IPIP모드 통신과정

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: server
spec:
  nodeName: k8s-w2
  containers:
  - name: netshoot
    image: nicolaka/netshoot
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo
hello; sleep 10;done"]
---
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  nodeName: k8s-w1
  containers:
  - name: netshoot
    image: nicolaka/netshoot
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo
hello; sleep 10;done"]
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/pod_pod/different_node.yaml

눈으로 보면

어려우니

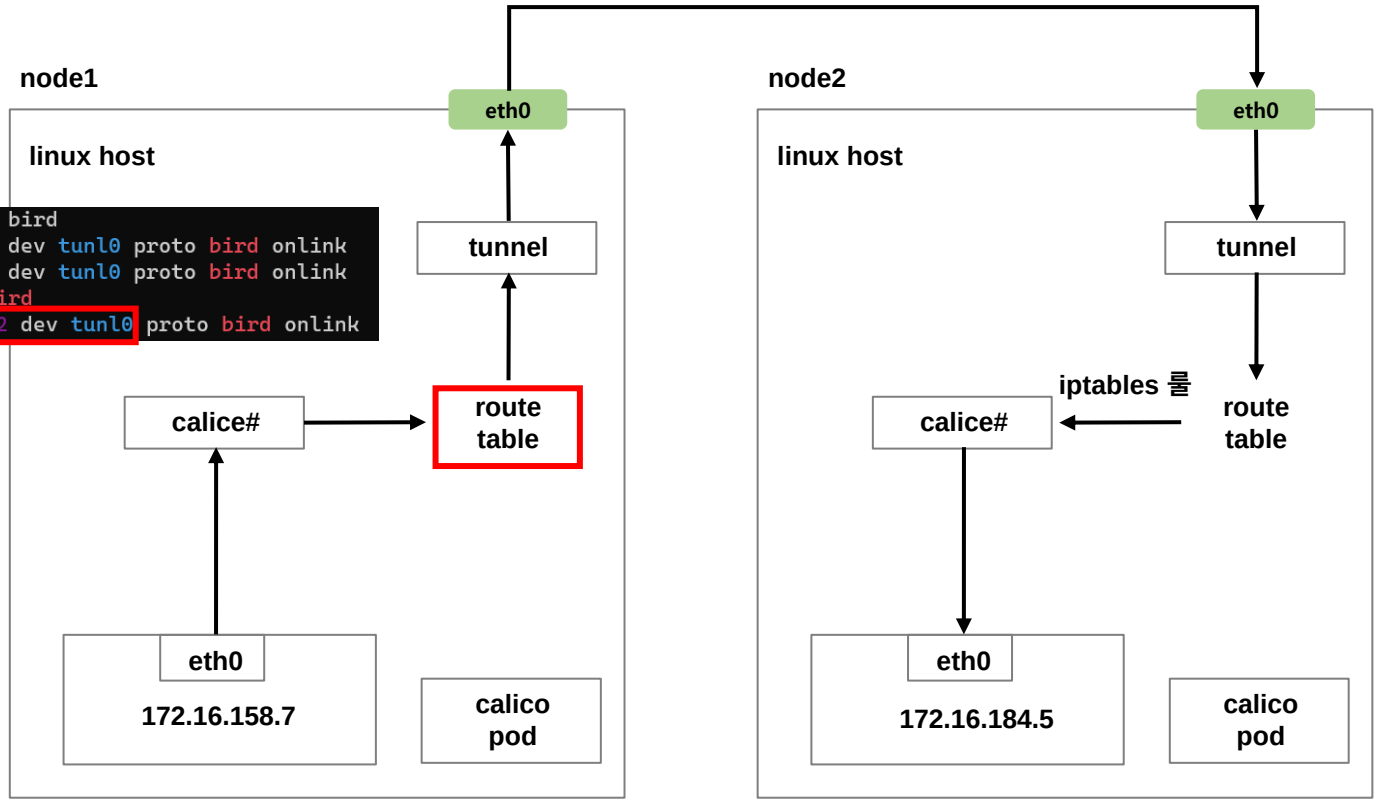
직접 디버깅해봐요!

6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

■ IPIP모드 설정 디버깅

```
(🐤 |calico-k8s:default) root@k8s-m:~# kubectl get po client server -o wide
NAME     READY   STATUS    RESTARTS   AGE    IP              NODE     NOMINATED
client   1/1     Running   0          117s   172.16.158.7    k8s-w1   <none>
server   1/1     Running   0          89m    172.16.184.5    k8s-w2   <none>
```

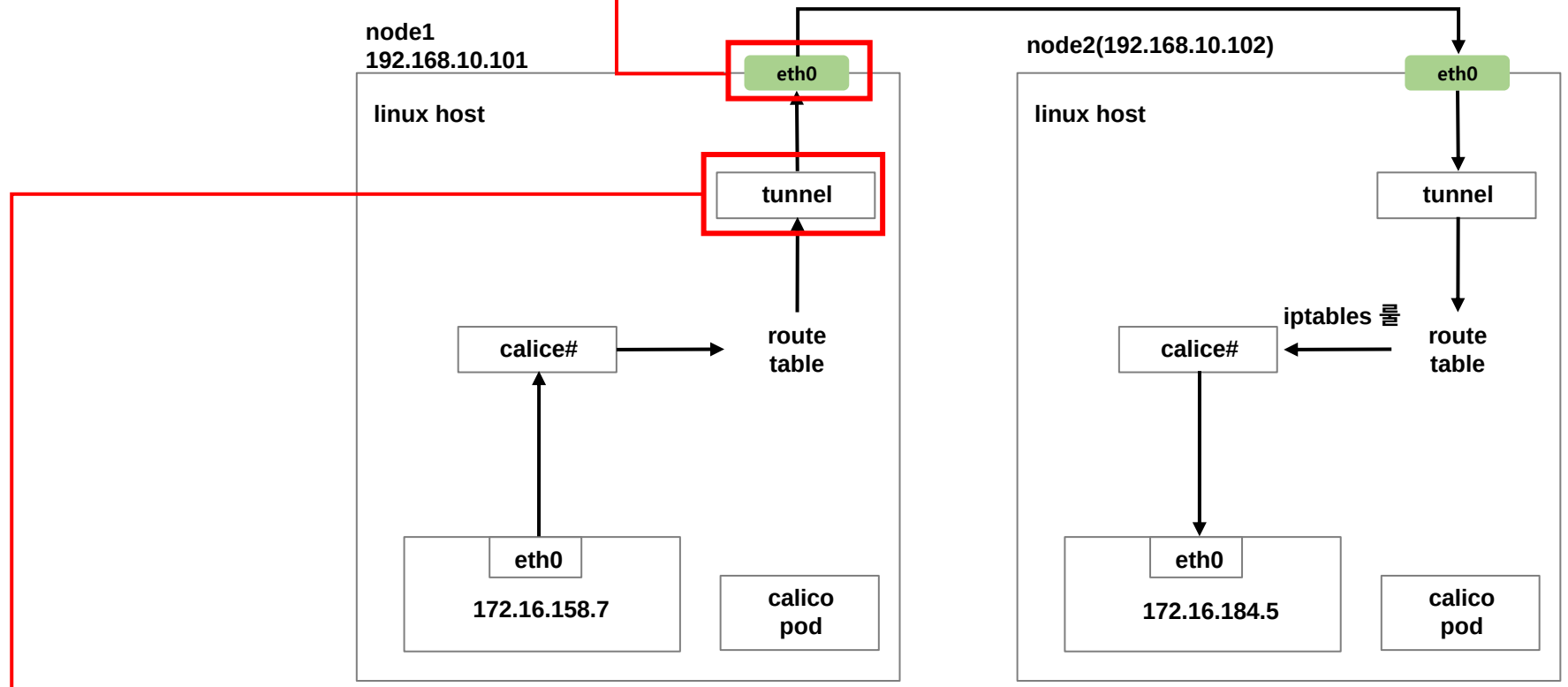
```
root@k8s-w1:~# ip -c route | grep bird
172.16.34.0/24 via 192.168.20.100 dev tunl0 proto bird onlink
172.16.116.0/24 via 192.168.10.10 dev tunl0 proto bird onlink
blackhole 172.16.158.0/24 proto bird
172.16.184.0/24 via 192.168.10.102 dev tunl0 proto bird onlink
```



6. 두번째 네트워킹 과제: (다른 노드)파드간 통신

■ 패킷 디버깅

```
root@k8s-w1:~# tcpdump -i enp0s8 -nn proto 4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
13:27:30.530245 IP 192.168.10.101 > 192.168.10.102: IP 172.16.158.7 > 172.16.184.5: ICMP echo request,
13:27:30.530988 IP 192.168.10.102 > 192.168.10.101: IP 172.16.184.5 > 172.16.158.7: ICMP echo reply, i
13:27:31.531019 IP 192.168.10.101 > 192.168.10.102: IP 172.16.158.7 > 172.16.184.5: ICMP echo request,
13:27:31.531923 IP 192.168.10.102 > 192.168.10.101: IP 172.16.184.5 > 172.16.158.7: ICMP echo reply, i
```




```
root@k8s-w1:~# tcpdump -i tunl0 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tunl0, link-type RAW (Raw IP), capture size 262144 bytes
13:20:56.953974 IP 172.16.158.7.38868 > 172.16.184.5.80: Flags [S], seq 920843763,
13:20:56.954678 IP 172.16.184.5.80 > 172.16.158.7.38868: Flags [R.], seq 0, ack 920
```

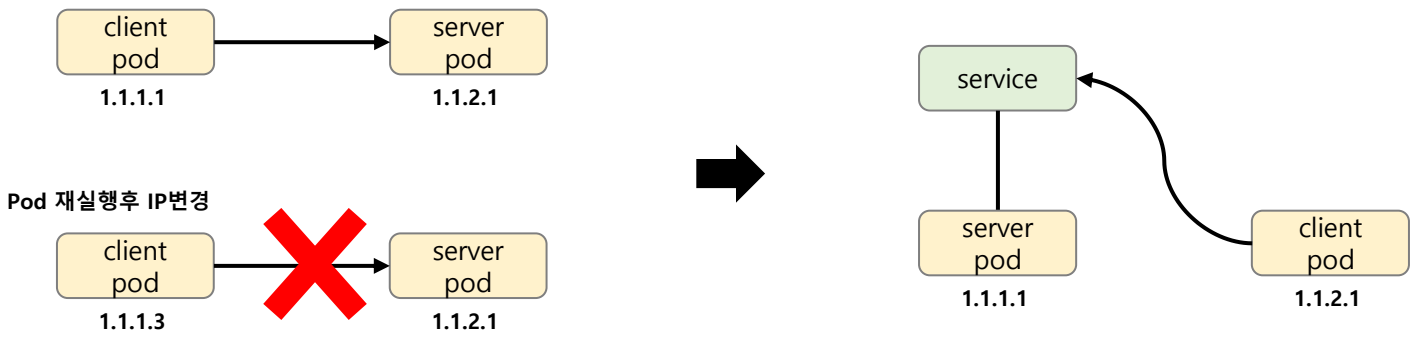
CNI 네트워킹 세번째 과제

- 서비스와 파드간 통신

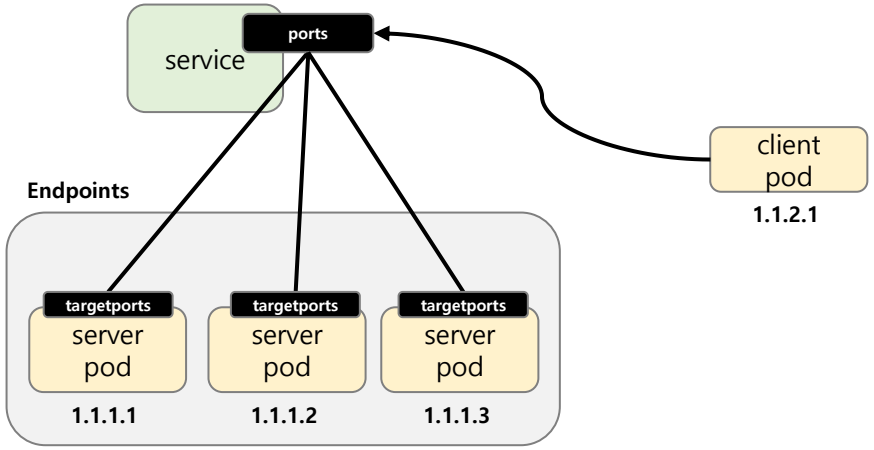
6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ 서비스는 비영구적인 파드에 고정적인 접근을 제공

- 파드는 재실행되면 IP가 변경됨



■ 서비스도 포트와 IP가 존재하며 Endpoints에 있는 파드로 트래픽 전달

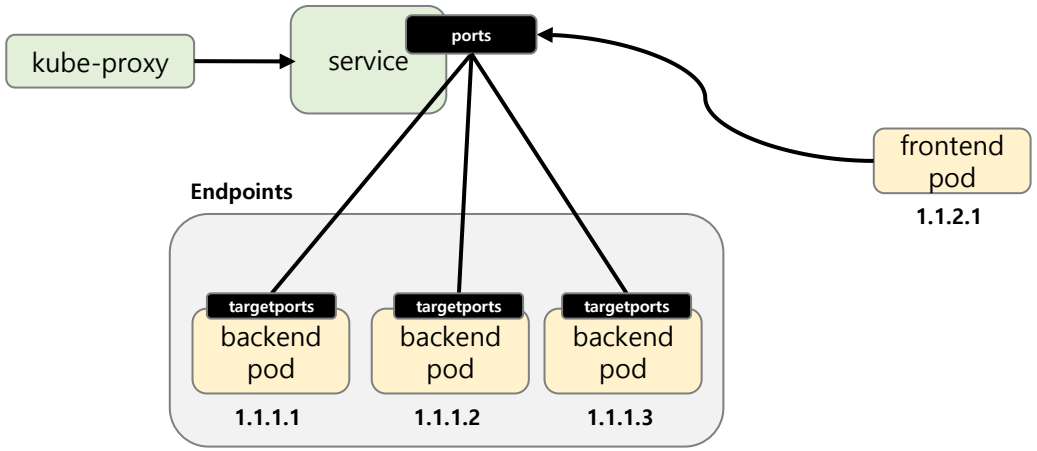


6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ kube-proxy가 서비스 IP와 포트, 트래픽 라우팅을 관리

■ kube-proxy는 3가지 방법(모드)으로 트래픽 라우팅

- userspace
- iptables
- ipvs



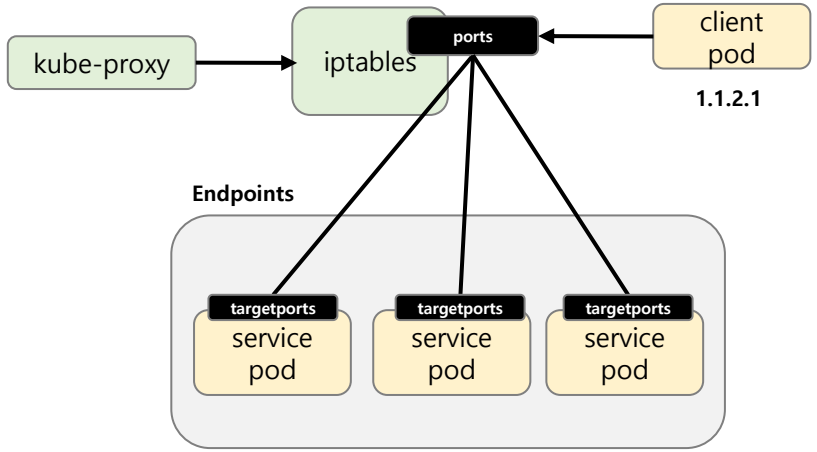
저희는

iptables모드를

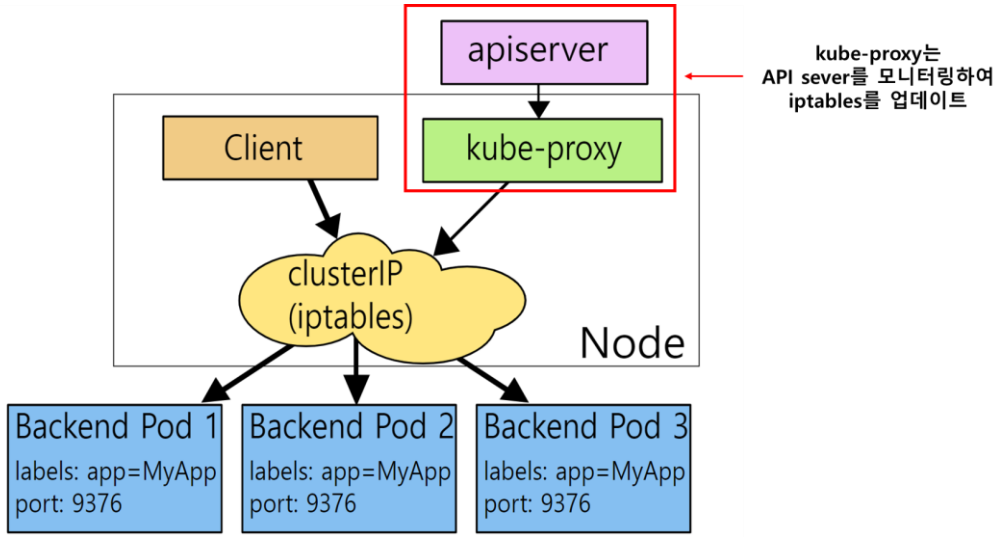
다룹니다.

6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ iptables모드는 iptables를 이용하여
서비스 → 파드 트래픽을 라우팅



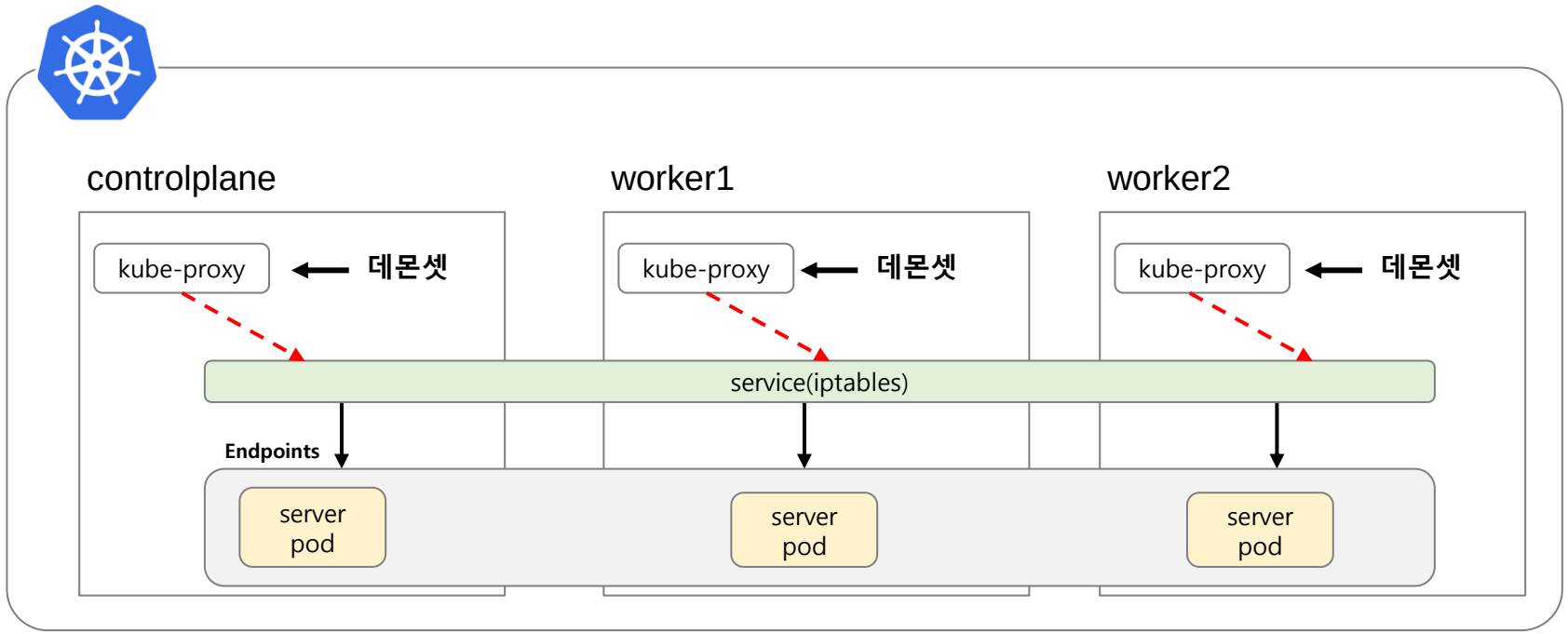
■ kube-proxy는 api-server를 모니터링하여
iptables를 수시로 업데이트



6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ kube-proxy는 모든 노드의 iptables를 관리

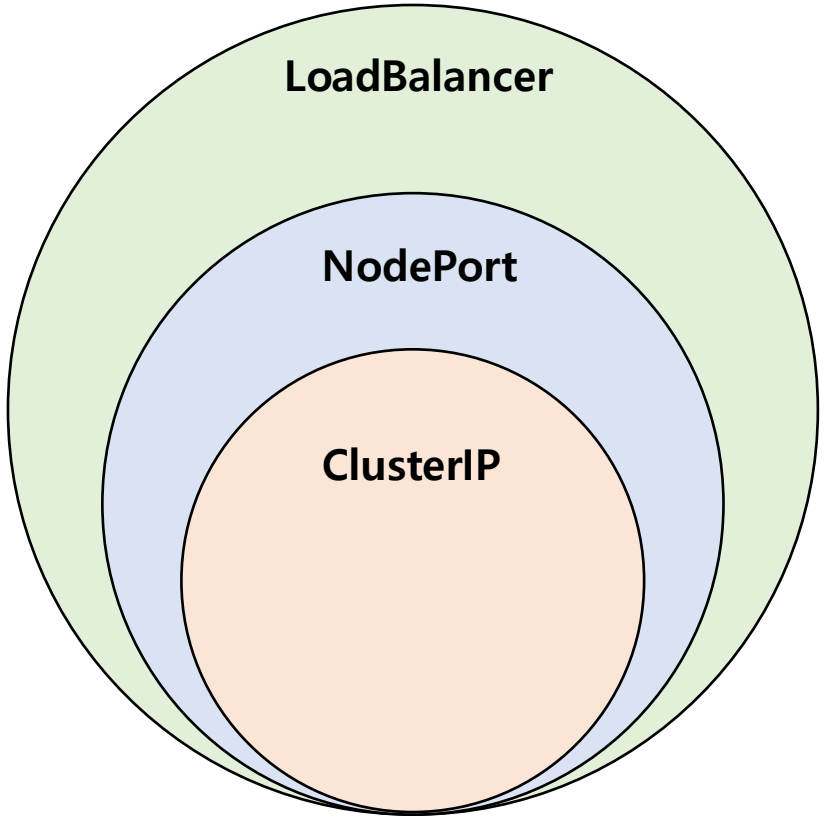
- kube-proxy는 데몬셋으로 실행



6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ 서비스 타입

- ClusterIP: 쿠버네티스 클러스터 외부에서 접근 불가
- NodePort: 외부에서 접근 가능
- LoadBalancer: External-IP 지정하여 외부에서 접근 가능

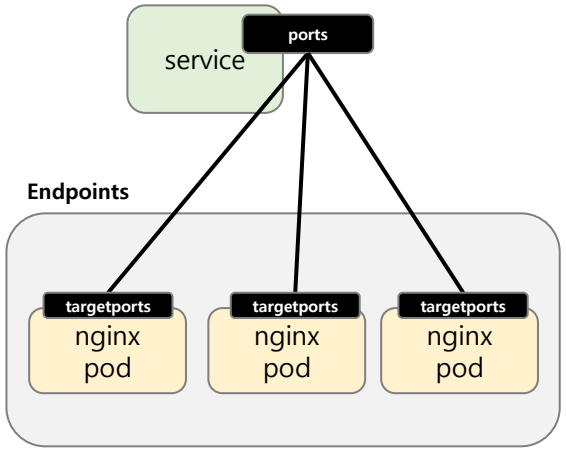


자! 이제
clusterIP를 다뤄봅시다.

6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ clusterIP 예제

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-test
  template:
    metadata:
      labels:
        app: nginx-test
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-test
spec:
  selector:
    app: nginx-test
  ports:
  - port: 80
    targetPort: 80
```

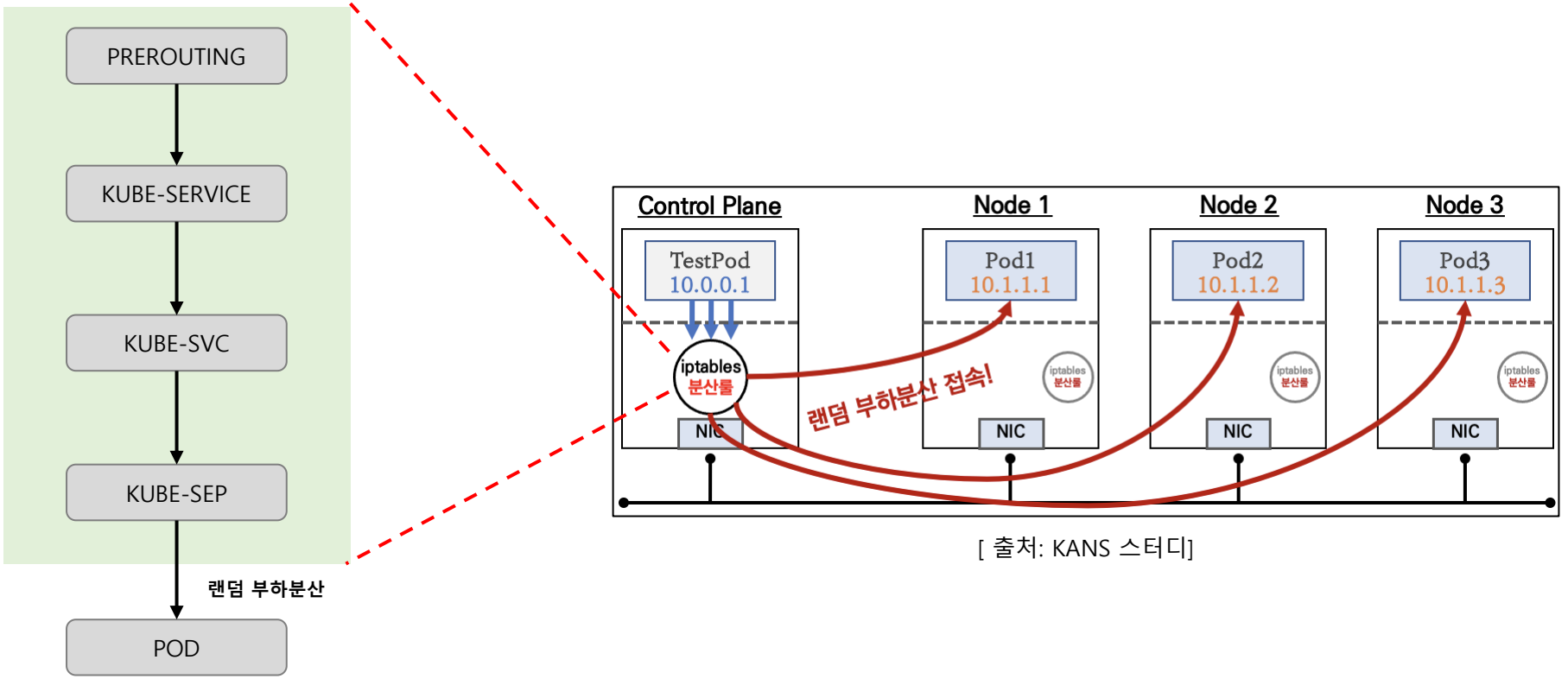



```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   50m
nginx-test   ClusterIP   10.104.209.155   <none>        80/TCP    11m
(⎈ |calico-k8s:default) root@k8s-m:~# curl 10.104.209.155
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
```

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/service/clusterip.yaml

6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ 첫 번째 포인트: 트래픽은 여러 단계 iptables 룰을 통과하여 파드로 트래픽 부하분산



[출처: KANS 스터디]

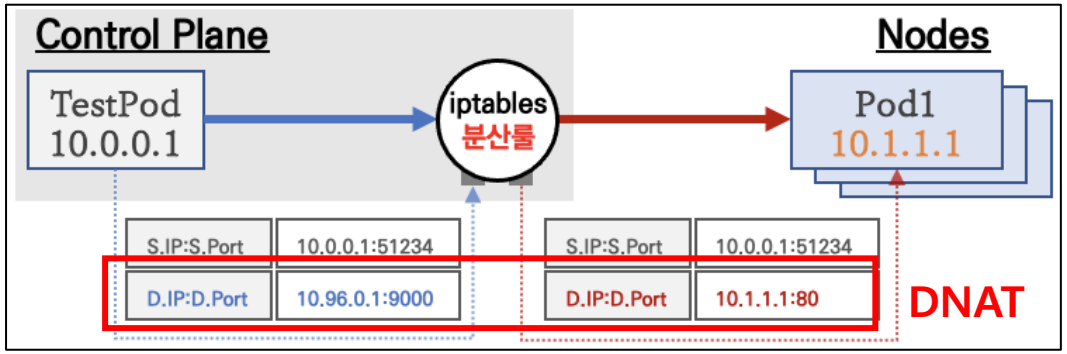
6. 세번째 네트워킹 과제: 서비스 ↔ 파드

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/service/clusterip.yaml

유투브 데모링크: https://youtu.be/8e_E-YAFH-U

6. 세번째 네트워킹 과제: 서비스 ↔ 파드

■ 두 번째 포인트: 도착지 IP가 podIP로 (D)NAT



[출처: KANS 스터디]

■ 예제 결과

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: clusterip-dnat
  labels:
    name: clusterip-dnat
spec:
  nodeName: k8s-w1
  containers:
  - name: whoami
    image: traefik/whoami
    resources:
      limits:
        memory: "64Mi"
        cpu: "100m"
    ports:
      - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: clusterip-dnat
spec:
  selector:
    name: clusterip-dnat
  ports:
  - port: 80
    targetPort: 80
```

```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get po -o wide clusterip-dnat
NAME             READY   STATUS    RESTARTS   AGE     IP              NODE     NOMINATED NODE   READINESS GATES
clusterip-dnat   1/1     Running   0          3m22s   172.16.158.2    k8s-w1   <none>           <none>
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get svc clusterip-dnat
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
clusterip-dnat   ClusterIP   10.96.143.238   <none>        80/TCP    3m31s
(⎈ |calico-k8s:default) root@k8s-m:~# curl 10.96.143.238
Hostname: clusterip-dnat
IP: 127.0.0.1
IP: 172.16.158.2
RemoteAddr: 172.16.116.0:38223
GET / HTTP/1.1
Host: 10.96.143.238
User-Agent: curl/7.68.0
Accept: */*
```

pod IP

service IP로 http GET요청

도착지 IP가 podIP로 DNAT

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/service/clusterip-dnat.yaml

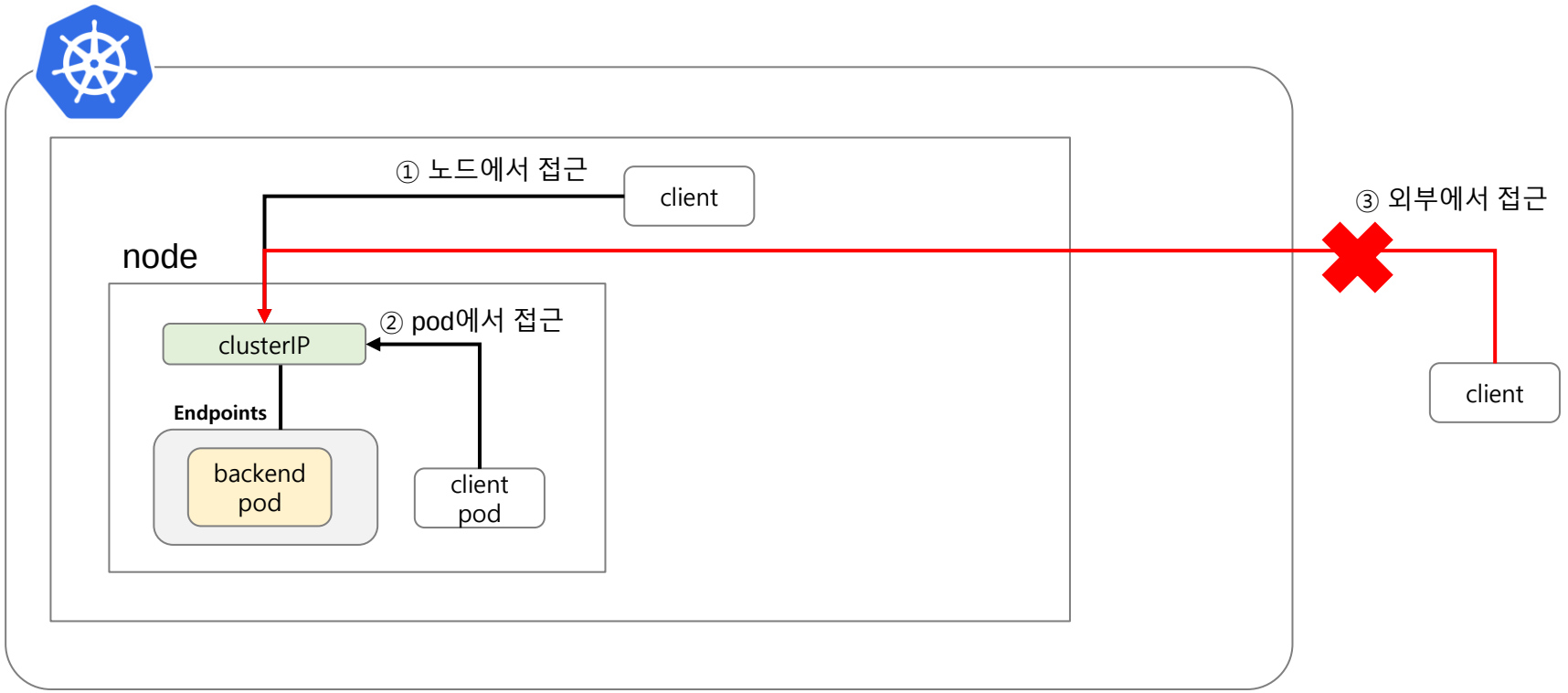
CNI 네트워킹 네번째 과제

- 외부에서 서비스 접근

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

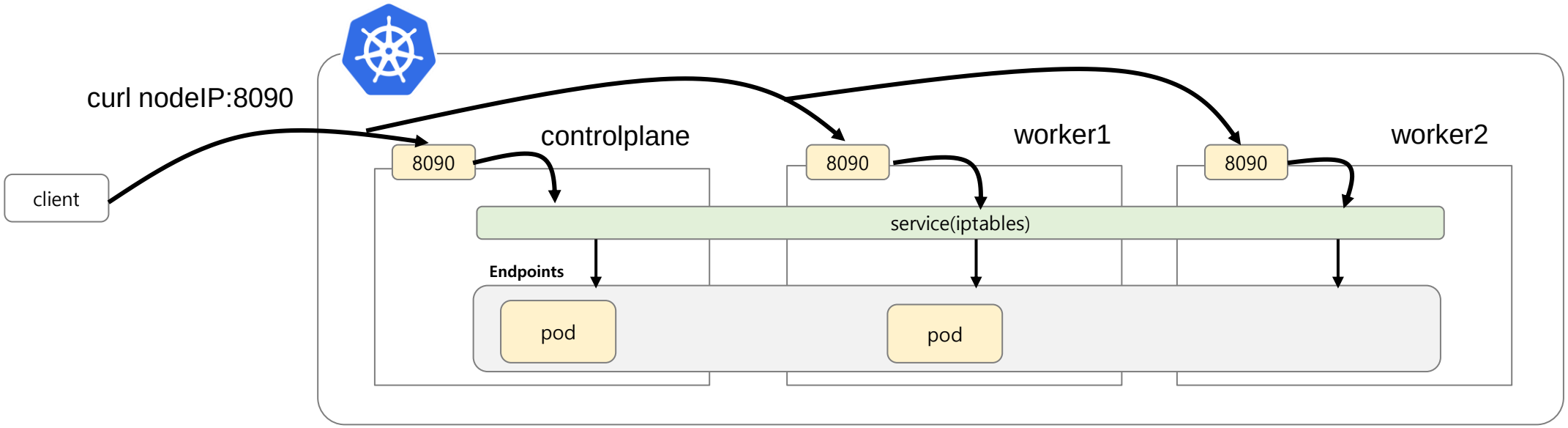
■ ClusterIP의 단점: 쿠버네티스 클러스터에서만 접속이 가능

- 이유: ClusterIP는 쿠버네티스 클러스터에서만 존재하는 가상IP이므로 외부에서 IP인식 불가



7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

- 외부에서 접근하려면 nodePort사용
- nodePort는 각 노드에 서비스에 연결된 포트를 오픈

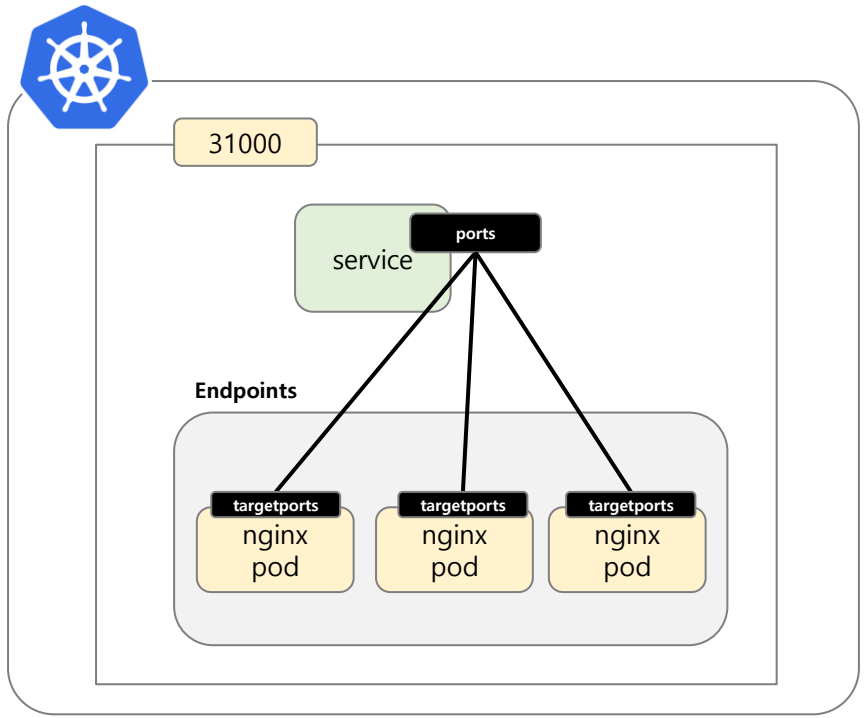


7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ nodeport 예제

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-nodeport-test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-nodeport-test
  template:
    metadata:
      labels:
        app: nginx-nodeport-test
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport-test
spec:
  type: NodePort
  selector:
    app: nginx-nodeport-test
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```



github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/service/nodeport.yaml

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ nodeport 단점

- 노드의 포트 자원을 소모
- 파드가 없는 노드에서도 포트를 오픈

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-nodeport-test
spec:
  replicas: 1
  ...
```

[replica를 1로 수정]

```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get po -o wide nginx-nodeport-test-866cfc566c-pf7mm
NAME                                    READY   STATUS    RESTARTS   AGE   IP              NODE     N
nginx-nodeport-test-866cfc566c-pf7mm    1/1     Running   0          17m   172.16.184.2    k8s-w2   <
```

[k8s-w2노드에만 pod실행]

```
(⎈ |calico-k8s:default) root@k8s-m:~# ss -nltp | grep 31000
LISTEN    0         4096               0.0.0.0:31000            0.0.0.0:*
```

[controlplane노드 포트확인]

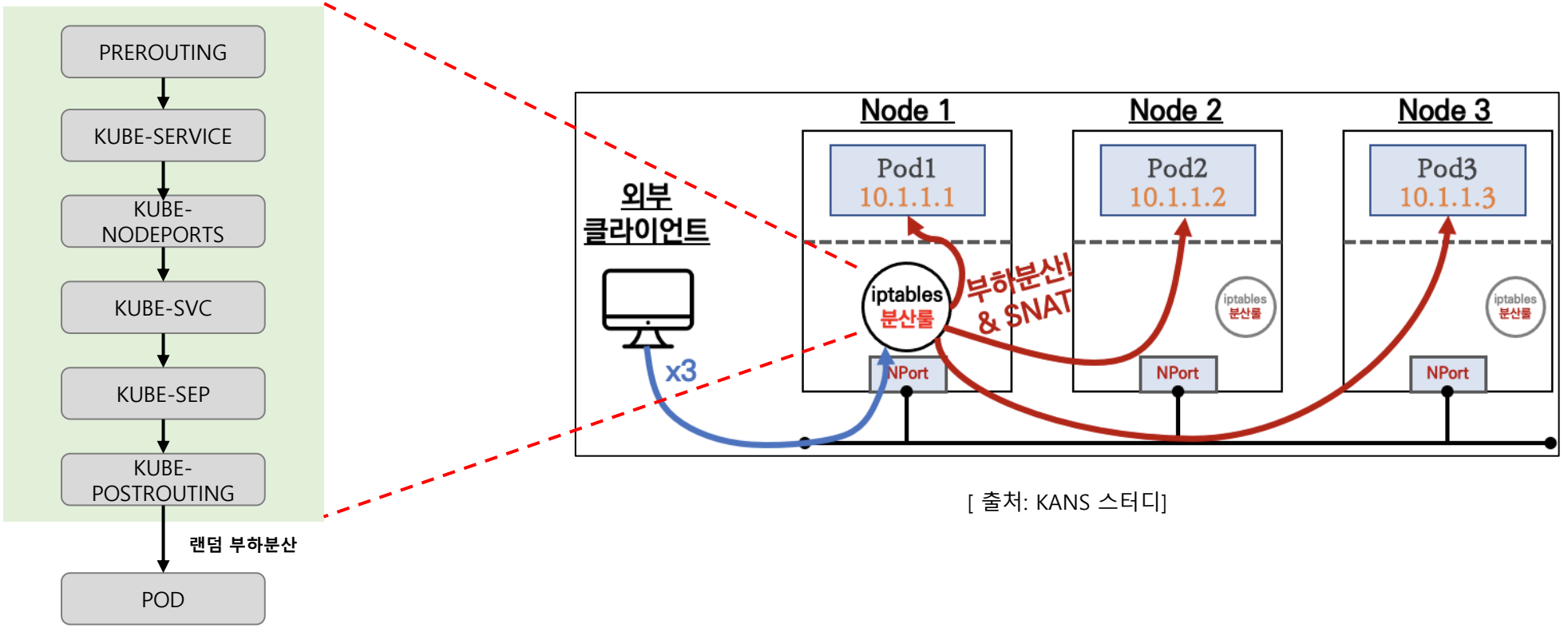
```
root@k8s-w1:~# ss -ntlp | grep 31000
LISTEN    0         4096               0.0.0.0:31000            0.0.0.0:*
```

[k8s-w1노드 포트확인]

nodeport의 iptables동작은
clsuterIP와 거의 비슷해요

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ 첫 번째 포인트: clusterIP과정에서 KUBE-POSTROUTING, KUBE-NODEPORTS룰 추가

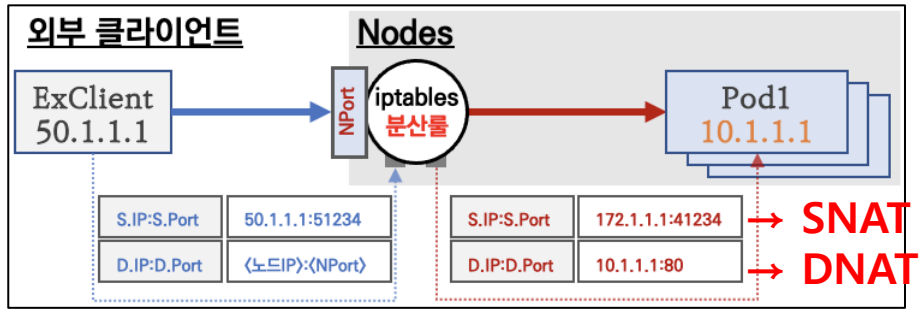


[출처: KANS 스터디]

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ 두 번째 포인트: 출발지 IP는 (S)NAT, 도착지 IP는 podIP로 (D)NAT



[출처: KANS 스터디]

■ 예제 결과

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeport-dnat
  labels:
    name: nodeport-dnat
spec:
  nodeName: k8s-w1
  containers:
  - name: whoami
    image: traefik/whoami
    resources:
      limits:
        memory: "64Mi"
        cpu: "100m"
    ports:
      - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nodeport-dnat
spec:
  type: NodePort
  selector:
    name: nodeport-dnat
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31010
```

```
(🐤 |calico-k8s:default) root@k8s-m:~# kubectl get no -o wide
NAME     STATUS     ROLES                  AGE     VERSION   INTERNAL-IP      EXTERNAL-IP
k8s-m    Ready      control-plane,master   6h59m   v1.22.6   192.168.10.10    <none>
k8s-w0   Ready      <none>                 6h54m   v1.22.6   192.168.20.100   <none>
k8s-w1   Ready      <none>                 6h50m   v1.22.6   192.168.10.101   <none>
k8s-w2   NotReady   <none>                 6h47m   v1.22.6   192.168.10.102   <none>
(🐤 |calico-k8s:default) root@k8s-m:~# curl -X GET 192.168.20.100:31010
Hostname: nodeport-dnat
IP: 127.0.0.1
IP: 172.16.158.4
RemoteAddr: 172.16.34.0:4742
GET / HTTP/1.1
Host: 192.168.20.100:31010
User-Agent: curl/7.68.0
Accept: */*
```

① node IP

② nodeIP:nodeport로 http GET요청

③ 도착지 IP가 podIP로 DNAT

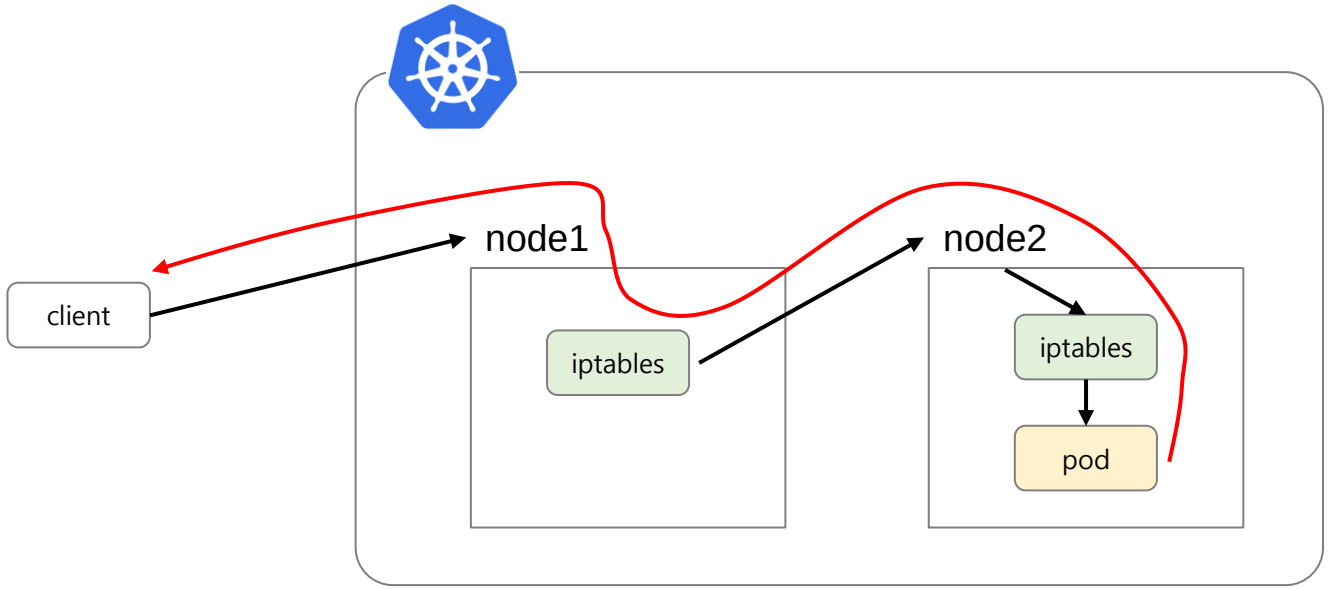
④ 출발지 IP가 calico tunnel인터페이스 IP로 SNAT

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/service/nodeport-nat.yaml

7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ 세 번째 포인트: 도착지pod에서 트래픽을 전달하면 시작 노드를 경유해서 되돌아감

- 이유: 도착지 파드로 접근시 출발지 IP가 클라이언트 IP에서 최초 접속한 노드의 IP로 변경

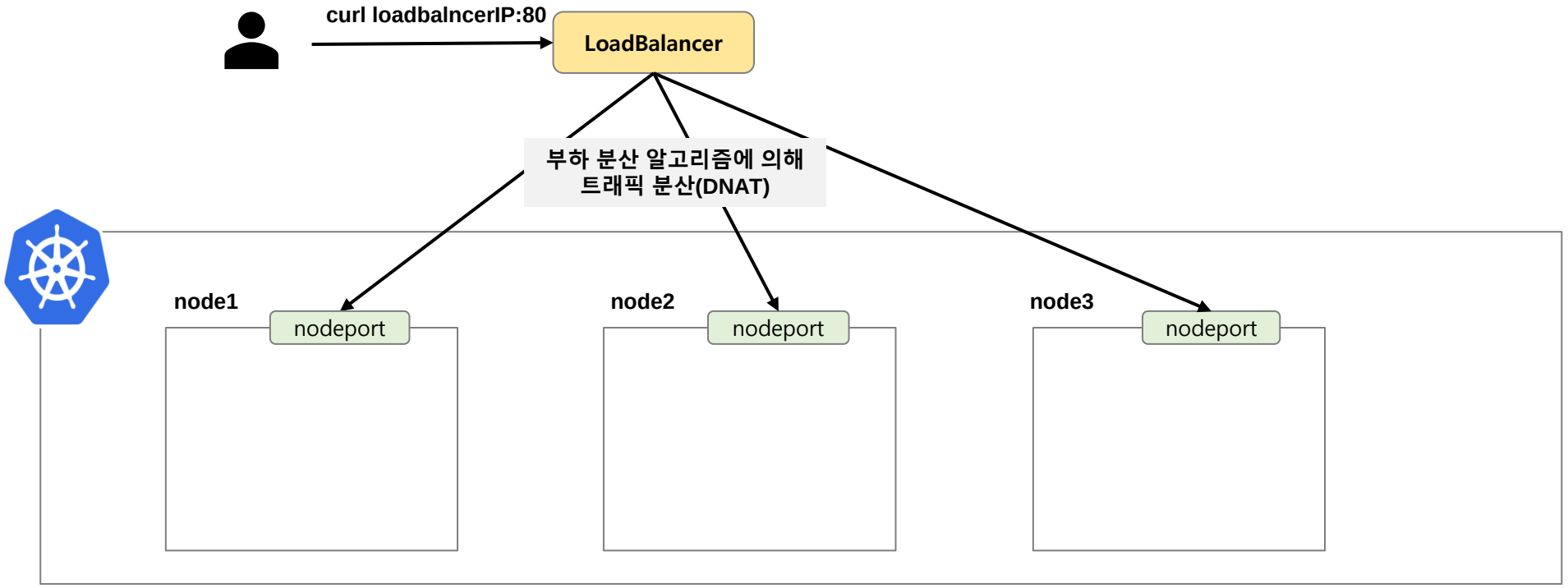


7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ 노드 중계서버를 이용하여 노드 부하 분산

- 중계서버는 하드웨어(예: L4 벤더장비) 또는 소프트웨어(예: MetalLB)로 구현

■ 노드 부하분산시 (D)NAT되는 것 이외에 nodeport동작과 같음



7. 네번째 네트워킹 과제: 외부 ↔ 서비스 통신

■ Loadbalancer는 클라우드에서 사용가능

■ 온프레미스에서 사용할 경우 솔루션(예: MetalLB) 설치 필요

- `LoadBalancer`: 클라우드 공급자의 로드 밸런서를 사용하여 서비스를 외부에 노출시킨다. 외부 로드 밸런서가 라우팅되는 `NodePort`와 `ClusterIP` 서비스가 자동으로 생성된다.

쿠버네티스 DNS서비스

- coredns

8. 쿠버네티스 DNS서비스 - coredns

■ 쿠버네티스 DNS서비스(coredns)는 2가지 역할을 수행

① 외부 도메인 쿼리 수행

② 파드, 서비스 레코드를 생성(서비스 디스커버리)

■ coredns는 deployment, 서비스로 배포

```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get po -n kube-system | grep core
coredns-78fcd69978-9nk2c                1/1     Running   0               9h
coredns-78fcd69978-gbnxf                1/1     Running   0               9h
```

[coredns pod]

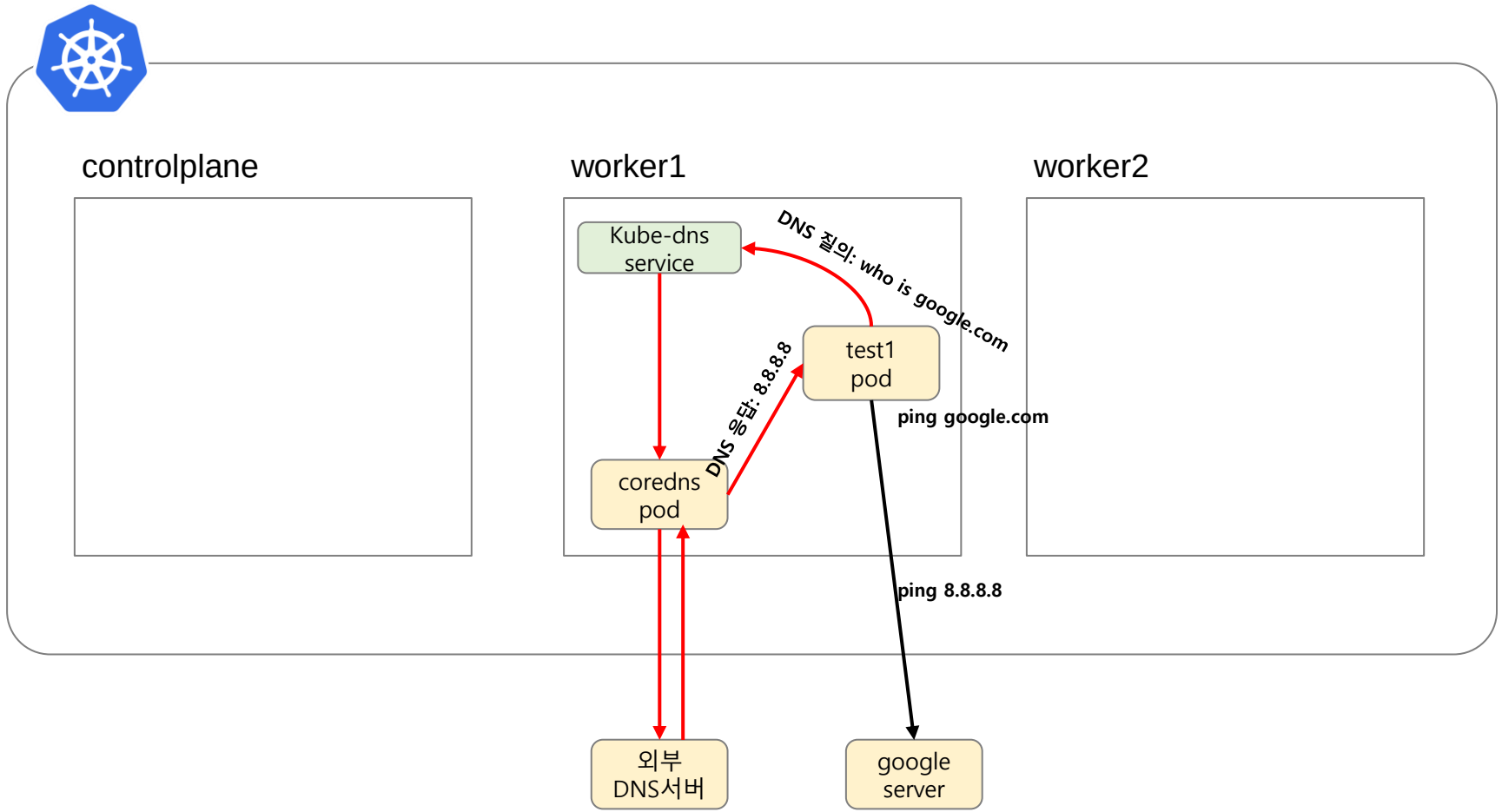
```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get svc -n kube-system
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   9h
```

[coredns 서비스]

8. 쿠버네티스 DNS서비스 - coredns

■ 파드의 외부 DNS질의 수행

- 파드는 외부 DNS질의를 하는 경우 coredns서비스로 dns질의 중개 요청



8. 쿠버네티스 DNS서비스 - coredns

■ coredns 패킷 덤프

- coredns에 연결된 calice인터페이스에 tcpdump

```
$ CoreDNSveth=$(calicoctl get workloadEndpoint -n kube-system | grep coredns | awk '{print $5}' | cut -d "/" -f 1)
$ tcpdump -i $CoreDNSveth -nn udp port 53
```

■ 파드에서 dns쿼리 질의하면 tcpdump에서 dns질의/응답이 관찰

```
$ kubectl exec -it dns-query -- dig google.com
```

```
(🐤 |calico-k8s:default) root@k8s-m:~# tcpdump -i $CoreDNSveth -nn udp port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on calibc3cd7dca6b, link-type EN10MB (Ethernet), capture size 262144 bytes
13:43:47.880729 IP 172.16.158.14.43317 > 172.16.116.10.53: 20729+ [1au] A? google.com. (51)
13:43:47.881560 IP 172.16.116.10.38969 > 1.1.1.1.53: 20729+ [1au] A? google.com. (51)
13:43:47.885987 IP 1.1.1.1.53 > 172.16.116.10.38969: 20729 1/0/1 A 172.217.26.238 (55)
13:43:47.886369 IP 172.16.116.10.53 > 172.16.158.14.43317: 20729 1/0/1 A 172.217.26.238 (77)
```

github 링크: https://github.com/choisungwook/facebook-meetup/tree/main/coredns

유투브 데모링크: https://youtu.be/3srdG7mnVfY

8. 쿠버네티스 DNS서비스 - coredns

■ 서비스, 파드 레코드 생성

- 파드: <파드ip>.<namespace>.pod.cluster.local
- 서비스: <서비스이름>.<namespace>.svc.cluster.local

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: coredns-example
  labels:
    name: coredns-example
spec:
  containers:
  - name: coredns-example
    image: nginx
    resources:
      limits:
        memory: "64Mi"
        cpu: "100m"
    ports:
      - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: coredns-example
spec:
  selector:
    name: coredns-example
  ports:
  - port: 80
    targetPort: 80
```

```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get pod coredns-example -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
coredns-example   1/1     Running   0          18m   172.16.184.4    k8s-w2   <none>           <none>
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl exec -it coredns-example -- curl 172-16-184-4.default.pod.cluster.local
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
```

[파드 dns질의]

```
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl get svc coredns-example
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
coredns-example   ClusterIP   10.104.85.140   <none>        80/TCP    2m6s
(⎈ |calico-k8s:default) root@k8s-m:~# kubectl exec -it coredns-example -- curl coredns-example.default.svc.cluster.local
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
```

[서비스 dns질의]

github 링크: https://github.com/choisungwook/facebook-meetup/blob/main/coredns/coredns.yaml 유투브 데모링크: https://youtu.be/Hma6rSit65A

8. 쿠버네티스 DNS서비스 - coredns

■ coredns동작은 configmap 설정에 따라 동작

```
(Service-DNS-k8s:default) root@k8s-m:~# kubectl -n kube-system describe configmap coredns
Name:         coredns
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
Corefile:
----
.:53 {
    errors
    health {
       lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
       pods insecure
       fallthrough in-addr.arpa ip6.arpa
       ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf {
       max_concurrent 1000
    }
    cache 30
    loop
    reload
    loadbalance
}
```

파드, 서비스 도메인 응답설정

그 이외의 도메인은 외부로 forward

8. 쿠버네티스 DNS서비스 - coredns

■ coredns파드가 장애가 난다면?

- 외부 도메인, 파드/서비스 dns질의 수행 불가

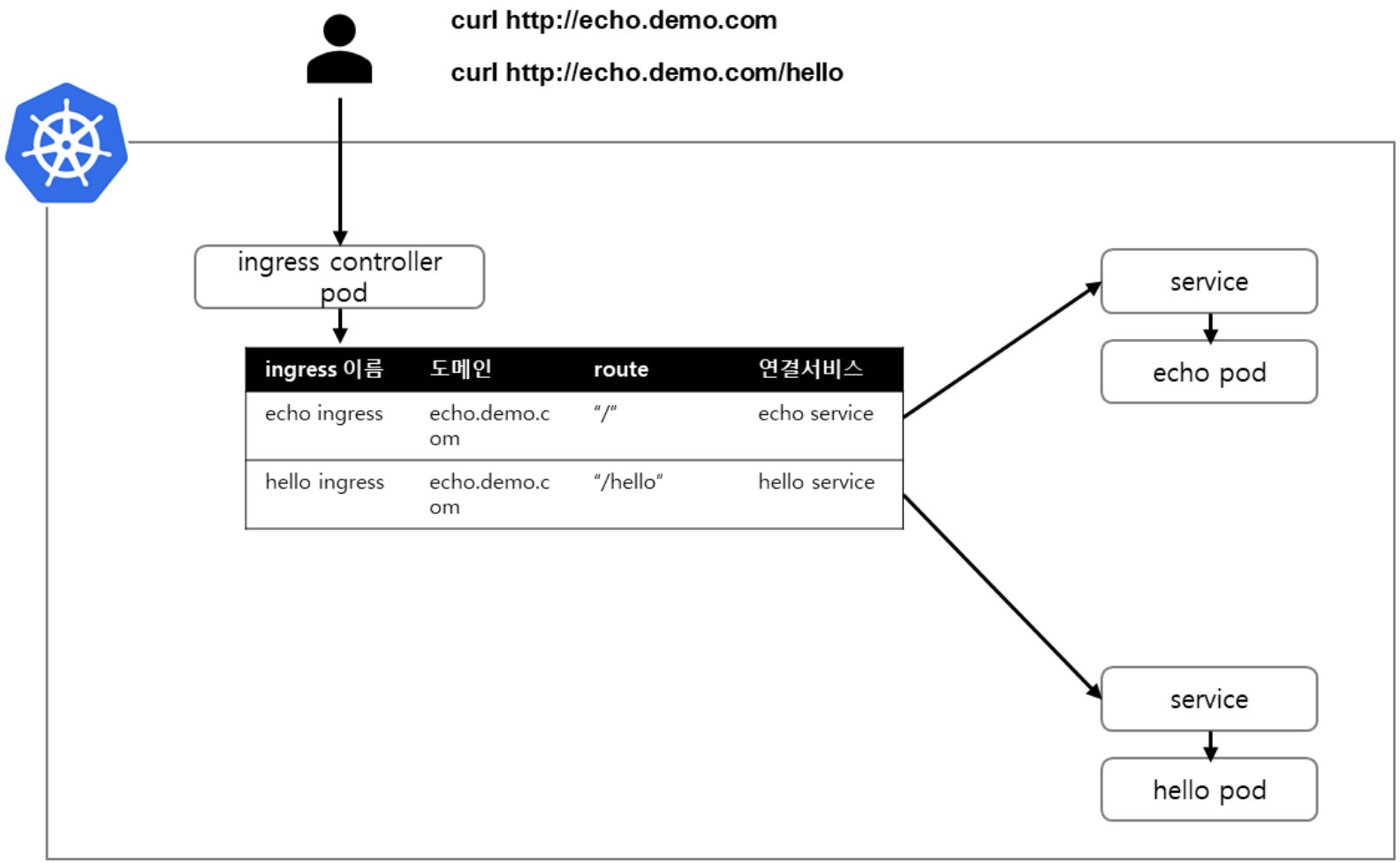
```
 netshoot  ~  ping google.com
ping: google.com: Try again
```

Ingress

9. Ingress

■ Ingress는 도메인/라우팅 정책을 설정

■ 도메인까지 네트워킹을 담당하는 것은 Ingress Controller(예: nginx 등)



9. Ingress

■ ingress 네트워킹은 ingress controller 파드에서 서비스를 거치지 않고 바로 파드로 라우팅

- NAT과정(이중 NAT)을 줄이기 위해

9. Ingress

■ ingress controller 파드에는 endpoints정보 조회 role이 설정

```
(⎈ |Ingress-k8s:default) root@k8s-m:~# kubectl describe clusterrole ingress-nginx | grep endpoints
  endpoints                                  []                 []              [list watch]
```

```
(⎈ |Ingress-k8s:default) root@k8s-m:~# kubectl -n ingress-nginx describe role ingress-nginx | grep endpoints
  endpoints                                  []                 []              [get list watch]
```

9. Ingress

■ 그 이외

- SSL/TLS 암호화 통신
- 분산부하 알고리즘 적용
- ingress controller(nginx 등) 설정 방법을 잘 이해!

마치며

마치며

■ 쿠버네티스 네트워크는 새로운 기술이 아니라, 리눅스와 네트워크 지식을 활용한 기술

- 그러므로 리눅스(iptables, route table 등), 네트워크(TCP/IP 모델, ARP 등) 학습도 중요

■ 환경에 맞게 CNI선택 필요

- 특정 CNI는 클라우드에서 동작 X

■ 시스템 규모에 따라 네트워크 설계 필요

- 대규모(파드,서비스 갯수 약 5천개 이상)에서는 iptables의 한계로 네트워크 지연 발생
- 네트워크 엔지니어가 있으면 수월

마치며

■ 다음 공부

- XFF
- Cilium CNI
  - iptables를 사용하지 않고 ebpf를 사용하여 네트워크 통신을 관리하는 CNI
  - 단점, iptables를 사용하는 오픈소스 생태계 호환성 문제
- 클라우드 매니지드 쿠버네티스의 네트워크
- 서비스 메쉬(예: Istio)

마치며

마지막!!

발표자료는 2022.1 ~ 2022.2월에

진행한 KANS스터디를 정리한 내용입니다.

스터디장 가시다님을 비롯한 스터디원에게 감사드립니다.

참고자료

■ [1] 커피고래 [번역]쿠버네티스 패킷의 삶: https://coffeewhale.com/packet-network1

■ [2] 커피고래 [번역] 쿠버네티스 네트워킹 이해하기: Podshttps://coffeewhale.com/k8s/network/2019/04/19/k8s-network-01/

https://speakerdeck.com/devinjeon/kubernetes-neteuweokeu-ihaehagi-2-seobiseu-gaenyeomgwa-dongjag-weonri

■ [3] 44bits ip로 직접 만들어보는 네트워크 네임스페이스와 브리지 네트워크

: https://www.44bits.io/ko/post/container-network-2-ip-command-and-network-namespace

■ [4] 서비스 iptables 통신 디버깅: https://morian-kim.tistory.com/23

■ [5] iptables MSAQ table 동작: https://ssup2.github.io/theory_analysis/Kubernetes_Service_Proxy/