RX-M Cloud Native Consulting

Advanced Kubernetes

Lab 5 – Software Defined Networking (SDN)

Flannel is a virtual network that gives a subnet to each host for use with container runtimes.

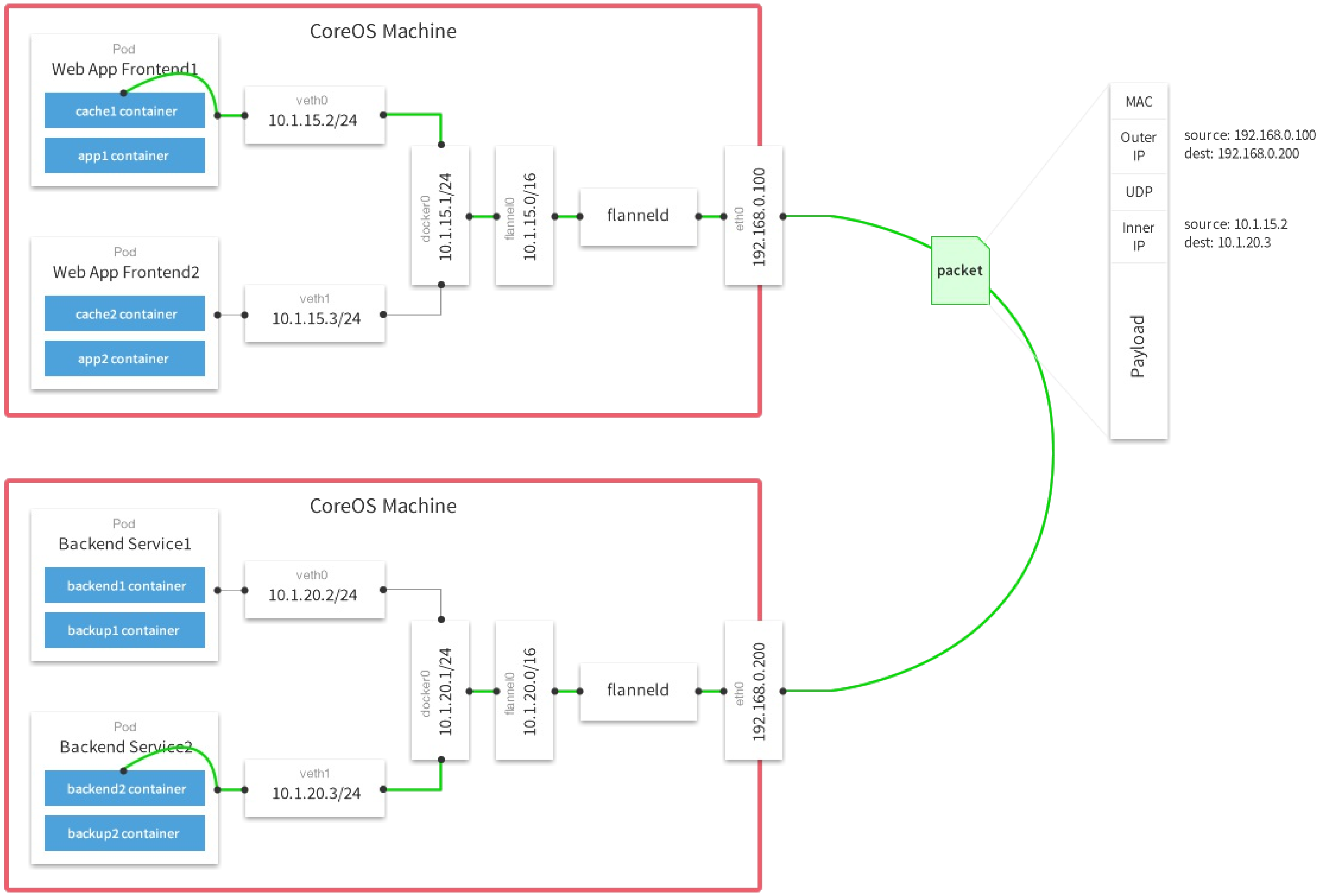
Platforms like Google's Kubernetes assume that each container (pod) has a unique, routable IP inside the cluster. The advantage of this model is that it reduces the complexity of doing port mapping.

Before proceeding we will need to **stop all the cluster services**, then add the external interface to etcd and restart it. First retrieve your nodea external IP and then configure etcd.

```
ubuntu@nodea:~$ ip a show eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:ef:63:d5:3b:be brd ff:ff:ff:ff:ff:ff
    inet 172.31.28.198/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ef:63ff:fed5:3bbe/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodea:~$ rm -Rf ~/default.etcd/

ubuntu@nodea:~$ etcd --listen-client-urls 'http://0.0.0.0:2379' --advertise-client-urls
'http://172.31.28.198:2379'
```

1. flannel

flannel runs an agent, flanneld, on each host and is responsible for allocating a subnet lease out of a preconfigured address space. Flannel uses etcd to store the network configuration, allocated subnets, and auxiliary data (such as host's IP.) The forwarding of packets is achieved using one of several strategies that are known as backends. The simplest backend is UDP and uses a TUN device to encapsulate every IP fragment in a UDP packet, forming an overlay network. The following diagram demonstrates the path a packet takes as it traverses the overlay network:



Flannel can also plug into Amazon VPC and use AWS routing tables to avoid the encapsulation associated with tunneling, which is how we will use it in this lab. You can find the latest details about flannel here https://github.com/coreos/flannel.

Building flannel

On *nodea*

```
ubuntu@nodea:~$ cd $HOME

ubuntu@nodea:~$ export GOPATH=$HOME/goProjects/  && echo $GOPATH

/home/ubuntu/goProjects/

ubuntu@nodea:~$ go get -v github.com/coreos/flannel

github.com/coreos/flannel (download)
created GOPATH=/home/ubuntu/goProjects/; see 'go help gopath'
github.com/coreos/flannel/vendor/github.com/vishvananda/netns
github.com/coreos/flannel/vendor/github.com/golang/glog
github.com/coreos/flannel/vendor/github.com/vishvananda/netlink/nl
github.com/coreos/flannel/vendor/golang.org/x/net/context
github.com/coreos/flannel/vendor/github.com/denverdino/aliyungo/util
github.com/coreos/flannel/vendor/github.com/vishvananda/netlink
github.com/coreos/flannel/vendor/github.com/denverdino/aliyungo/common
github.com/coreos/flannel/vendor/github.com/denverdino/aliyungo/ecs

...

ubuntu@nodea:~$ cd $GOPATH/src/github.com/coreos/flannel

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ go build -v github.com/coreos/flannel

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ make dist/flanneld

go build -o dist/flanneld \
  -ldflags '-s -w -X github.com/coreos/flannel/version.Version=v0.11.0-11-ge4deb05 -extldflags "-static"'
# github.com/coreos/flannel
/tmp/go-link-934176633/000022.o: In function `mygetgrouplist':
/workdir/go/src/os/user/getgrouplist_unix.go:16: warning: Using 'getgrouplist' in statically linked applications
requires at runtime the shared libraries from the glibc version used for linking
/tmp/go-link-934176633/000021.o: In function `mygetgrgid_r':
```

```
/workdir/go/src/os/user/cgo_lookup_unix.go:38: warning: Using 'getgrgid_r' in statically linked applications
requires at runtime the shared libraries from the glibc version used for linking
/tmp/go-link-934176633/000021.o: In function `mygetgrnam_r':
/workdir/go/src/os/user/cgo_lookup_unix.go:43: warning: Using 'getgrnam_r' in statically linked applications
requires at runtime the shared libraries from the glibc version used for linking
/tmp/go-link-934176633/000021.o: In function `mygetpwnam_r':
/workdir/go/src/os/user/cgo_lookup_unix.go:33: warning: Using 'getpwnam_r' in statically linked applications
requires at runtime the shared libraries from the glibc version used for linking
/tmp/go-link-934176633/000021.o: In function `mygetpwuid_r':
/workdir/go/src/os/user/cgo_lookup_unix.go:28: warning: Using 'getpwuid_r' in statically linked applications
requires at runtime the shared libraries from the glibc version used for linking
/tmp/go-link-934176633/000004.o: In function `_cgo_7e1b3c2abc8d_C2func_getaddrinfo':
/tmp/go-build/cgo-gcc-prolog:57: warning: Using 'getaddrinfo' in statically linked applications requires at
runtime the shared libraries from the glibc version used for linking

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ ./dist/flanneld -version

v0.11.0-11-ge4deb05
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$
```

Configure the subnet for our overlay; first, check what configuration, if any, is stored in etcd:

```
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ etcdctl get /coreos.com/network/config

Error:  100: Key not found (/coreos.com) [3]
```

Now, set a network config with the type "aws-vpc". NOTE: the 10.1.0.0/16 network in the example below is for **example purposes ONLY**; your instructor will assign you a unique subnet to avoid overlapping with other students.

```
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ etcdctl set /coreos.com/network/config \
'{"Network":"10.1.0.0/16", "Backend": {"Type": "aws-vpc"}}'

{"Network":"10.1.0.0/16", "Backend": {"Type": "aws-vpc"}}
```

Make sure the configuration took:

```
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ etcdctl get /coreos.com/network/config

{"Network":"10.1.0.0/16", "Backend": {"Type": "aws-vpc"}}
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$
```

Starting flanneld

Open a new terminal and start the flanneld daemon, on *nodea* we can point it at localhost because etcd is running on nodea.

```
laptop$ ssh -i k8s-adv-student.pem ubuntu@<external-ip>

...

ubuntu@nodea:~$ cd ~/goProjects/src/github.com/coreos/flannel

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ sudo ./dist/flanneld --etcd-
endpoints=http://localhost:2379

I0331 01:44:17.321923   27251 main.go:516] Determining IP address of default interface
I0331 01:44:17.322139   27251 main.go:529] Using interface with name eth0 and address 172.31.28.198
I0331 01:44:17.322158   27251 main.go:546] Defaulting external address to interface address (172.31.28.198)
I0331 01:44:17.322220   27251 main.go:244] Created subnet manager: Etcd Local Manager with Previous Subnet: None
I0331 01:44:17.322234   27251 main.go:247] Installing signal handlers
I0331 01:44:17.323467   27251 main.go:388] Found network config - Backend type: aws-vpc
I0331 01:44:17.323506   27251 awsvpc.go:88] Backend configured as: %s{"Type": "aws-vpc"}
I0331 01:44:17.324533   27251 local_manager.go:234] Picking subnet in range 10.1.1.0 ... 10.1.255.0
I0331 01:44:17.325194   27251 local_manager.go:220] Allocated lease (10.1.40.0/24) to current node (172.31.28.198)
I0331 01:44:17.516755   27251 awsvpc.go:322] Found eni-0eb2bd30210caf566 that has 172.31.28.198 IP address.
W0331 01:44:17.564435   27251 awsvpc.go:134] failed to disable SourceDestCheck on eni-0eb2bd30210caf566:
UnauthorizedOperation: You are not authorized to perform this operation.
        status code: 403, request id: 99cf00a9-233d-4e0b-9b8f-3c35692d9d85.
I0331 01:44:17.564483   27251 awsvpc.go:79] Route table configured: false
I0331 01:44:17.724500   27251 awsvpc.go:141] Found route table rtb-6f23030a.
I0331 01:44:17.725009   27251 awsvpc.go:59] RouteTableID configured as string: %srtb-6f23030a
I0331 01:44:18.062726   27251 awsvpc.go:256] Route added to table rtb-6f23030a: 10.1.40.0/24 - eni-
0eb2bd30210caf566.
I0331 01:44:18.062759   27251 main.go:311] Changing default FORWARD chain policy to ACCEPT
I0331 01:44:18.062870   27251 main.go:319] Wrote subnet file to /run/flannel/subnet.env
I0331 01:44:18.062885   27251 main.go:323] Running backend.
```

```
I0331 01:44:18.063577   27251 main.go:431] Waiting for 22h59m59.261154778s to renew lease
I0331 01:44:18.065186   27251 iptables.go:145] Some iptables rules are missing; deleting and recreating rules
I0331 01:44:18.065204   27251 iptables.go:167] Deleting iptables rule: -s 10.1.0.0/16 -j ACCEPT
I0331 01:44:18.065919   27251 iptables.go:167] Deleting iptables rule: -d 10.1.0.0/16 -j ACCEPT
I0331 01:44:18.066800   27251 iptables.go:155] Adding iptables rule: -s 10.1.0.0/16 -j ACCEPT
I0331 01:44:18.068365   27251 iptables.go:155] Adding iptables rule: -d 10.1.0.0/16 -j ACCEPT
```

Take note of the subnet flannel uses (in the example: `Allocated lease, 10.1.40.0/24`) , this is what we will configure the Docker daemon to use (later) instead of the default bridge subnet on *nodea*.

Configure Docker

On *nodea*, remove the old bridge IP address. First shutdown docker:

```
ubuntu@nodea:~$ sudo systemctl stop docker

ubuntu@nodea:~$
```

Next, find the IP of the docker0 bridge and delete it:

```
ubuntu@nodea:~$ ip a show docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:1d:ce:ed:23 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:1dff:fece:ed23/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodea:~$ sudo ip addr del 172.17.0.1/16 dev docker0

ubuntu@nodea:~$
```

When flannel starts on each node, it lays out a configuration file that others can use.

On *nodea*:

```
ubuntu@nodea:~$ cat /var/run/flannel/subnet.env

FLANNEL_NETWORK=10.1.0.0/16
FLANNEL_SUBNET=10.1.40.1/24
FLANNEL_MTU=9001
FLANNEL_IPMASQ=false
ubuntu@nodea:~$
```

Now create the Docker daemon configuration file on *nodea* so that docker assigns the docker0 bridge the value of FLANNEL_SUBNET (in the example: 10.1.40.1/24) with the mtu of FLANNEL_MTU (in the example: 9001):

```
ubuntu@nodea:~$ sudo vim /etc/docker/daemon.json

ubuntu@nodea:~$ cat /etc/docker/daemon.json

{
    "bip": "10.1.40.1/24",
    "mtu": 9001
}
ubuntu@nodea:~$
```

Restart docker:

```
ubuntu@nodea:~$ sudo systemctl start docker

ubuntu@nodea:~$ sudo systemctl status docker

● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2019-03-31 01:54:35 UTC; 7s ago
     Docs: https://docs.docker.com
 Main PID: 27545 (dockerd)
    Tasks: 10
   Memory: 31.6M
      CPU: 198ms
    CGroup: /system.slice/docker.service
            └─27545 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.713566470Z" level=info msg="Graph migration to
content-addressability took 0.00 seconds"
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.713727890Z" level=warning msg="Your kernel does
not support swap memory limit"
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.713776454Z" level=warning msg="Your kernel does
not support cgroup rt period"
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.713785804Z" level=warning msg="Your kernel does
not support cgroup rt runtime"
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.714109539Z" level=info msg="Loading containers:
start."
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.809428012Z" level=info msg="Loading containers:
done."
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.836372066Z" level=info msg="Docker daemon"
commit=774a1f4 graphdriver(s)=overlay2 version=18.09.3
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.836445676Z" level=info msg="Daemon has completed
initialization"
Mar 31 01:54:35 nodea dockerd[27545]: time="2019-03-31T01:54:35.847890997Z" level=info msg="API listen on
/var/run/docker.sock"
Mar 31 01:54:35 nodea systemd[1]: Started Docker Application Container Engine.
ubuntu@nodea:~$
```

Now run a test container, and inspect its eth0 interface:

```
ubuntu@nodea:~$ docker container run --rm ubuntu:14.04 ip a show eth0

Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
e082d4499130: Pull complete
371450624c9e: Pull complete
c8a555b3a57c: Pull complete
1456d810d42e: Pull complete
Digest: sha256:6612de24437f6f01d6a2988ed9a36b3603df06e8d2c0493678f3ee696bc4bb2d
Status: Downloaded newer image for ubuntu:14.04
58: eth0@if59: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc noqueue state UP group default
    link/ether 02:42:0a:01:23:02 brd ff:ff:ff:ff:ff:ff
    inet 10.1.40.2/24 brd 10.1.40.255 scope global eth0
       valid_lft forever preferred_lft forever
ubuntu@nodea:~$
```

Great! Our container is connected to the new Flannel network!

Copy flanneld to nodeb

Copy the flanneld executable to *nodeb*, this way we don't need to install all the compilation tools again. Follow these steps to copy the binary.

On *nodeb* run:

`netcat -l -p 7000 | tar xv`

On nodea run:

```
ubuntu@nodea:~$ cd ~/goProjects/src/github.com/coreos/flannel

ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ tar cf - dist/flanneld | nc nodeb 7000
```

Start flanneld on nodeb

On *nodeb* start up the flanneld daemon **in its own terminal**. This time we will set the etcd endpoint to the hostname or IP of nodea:

```
ubuntu@nodeb:~$ sudo ./dist/flanneld --etcd-endpoints=http://nodea:2379

I0331 01:59:16.768732   14246 main.go:516] Determining IP address of default interface
I0331 01:59:16.769057   14246 main.go:529] Using interface with name eth0 and address 172.31.30.148
I0331 01:59:16.769078   14246 main.go:546] Defaulting external address to interface address (172.31.30.148)
I0331 01:59:16.769195   14246 main.go:244] Created subnet manager: Etcd Local Manager with Previous Subnet: None
I0331 01:59:16.769211   14246 main.go:247] Installing signal handlers
I0331 01:59:16.771005   14246 main.go:388] Found network config - Backend type: aws-vpc
I0331 01:59:16.771035   14246 awsvpc.go:88] Backend configured as: %s{"Type": "aws-vpc"}
I0331 01:59:16.772415   14246 local_manager.go:234] Picking subnet in range 10.1.1.0 ... 10.1.255.0
I0331 01:59:16.773135   14246 local_manager.go:220] Allocated lease (10.1.87.0/24) to current node (172.31.30.148)
I0331 01:59:16.991245   14246 awsvpc.go:322] Found eni-073753aa399a1de5f that has 172.31.30.148 IP address.
W0331 01:59:17.038623   14246 awsvpc.go:134] failed to disable SourceDestCheck on eni-073753aa399a1de5f:
UnauthorizedOperation: You are not authorized to perform this operation.
        status code: 403, request id: 2e5d6279-d54c-4af5-ab2a-45416d11403c.
I0331 01:59:17.038775   14246 awsvpc.go:79] Route table configured: false
```

```
I0331 01:59:17.211710   14246 awsvpc.go:141] Found route table rtb-6f23030a.
I0331 01:59:17.212613   14246 awsvpc.go:59] RouteTableID configured as string: %srtb-6f23030a
I0331 01:59:17.555961   14246 awsvpc.go:256] Route added to table rtb-6f23030a: 10.1.87.0/24 - eni-
073753aa399a1de5f.
I0331 01:59:17.555989   14246 main.go:311] Changing default FORWARD chain policy to ACCEPT
I0331 01:59:17.556105   14246 main.go:319] Wrote subnet file to /run/flannel/subnet.env
I0331 01:59:17.556121   14246 main.go:323] Running backend.
I0331 01:59:17.557071   14246 main.go:431] Waiting for 22h59m59.219057887s to renew lease
I0331 01:59:17.557978   14246 iptables.go:145] Some iptables rules are missing; deleting and recreating rules
I0331 01:59:17.557995   14246 iptables.go:167] Deleting iptables rule: -s 10.1.0.0/16 -j ACCEPT
I0331 01:59:17.558792   14246 iptables.go:167] Deleting iptables rule: -d 10.1.0.0/16 -j ACCEPT
I0331 01:59:17.559622   14246 iptables.go:155] Adding iptables rule: -s 10.1.0.0/16 -j ACCEPT
I0331 01:59:17.561129   14246 iptables.go:155] Adding iptables rule: -d 10.1.0.0/16 -j ACCEPT
```

Notice the subnet (Allocated lease), 10.1.87.0/24, this is what we will configure (later) Docker daemon to use versus the default bridge subnet on *nodeb* as well.

Now, with the *nodeb* leased subnet, follow the same procedure, changing the existing docker daemon configuration file:

```
ubuntu@nodeb:~$ sudo systemctl stop docker

ubuntu@nodeb:~$ ip a show docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:1d:ce:ed:23 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:1dff:fece:ed23/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodeb:~$ sudo ip addr del 172.18.0.1/16 dev docker0

ubuntu@nodeb:~$ cat /var/run/flannel/subnet.env

FLANNEL_NETWORK=10.1.0.0/16
FLANNEL_SUBNET=10.1.87.1/24
FLANNEL_MTU=9001
FLANNEL_IPMASQ=false
ubuntu@nodeb:~$
```

Now create the Docker daemon configuration file on *nodeb* so that docker assigns the docker0 bridge the value of FLANNEL_SUBNET (in the example: 10.1.87.1/24) with the mtu of FLANNEL_MTU (in the example: 9001):

```
ubuntu@nodeb:~$ sudo vim /etc/docker/daemon.json
ubuntu@nodeb:~$ sudo cat /etc/docker/daemon.json

{
          "bip": "10.1.87.1/24",
    "mtu": 9001
}

ubuntu@nodeb:~$ sudo systemctl start docker

ubuntu@nodeb:~$ sudo systemctl status docker

● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2019-03-31 02:01:46 UTC; 7s ago
     Docs: https://docs.docker.com
 Main PID: 14336 (dockerd)
    Tasks: 10
   Memory: 31.2M
      CPU: 195ms
   CGroup: /system.slice/docker.service
           └─14336 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.289437235Z" level=info msg="Graph migration to
content-addressability
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.289605407Z" level=warning msg="Your kernel does
not support swap memo
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.289645515Z" level=warning msg="Your kernel does
not support cgroup rt
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.289655564Z" level=warning msg="Your kernel does
not support cgroup rt
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.290076894Z" level=info msg="Loading containers:
start."
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.382212666Z" level=info msg="Loading containers:
done."
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.418440682Z" level=info msg="Docker daemon"
commit=d14af54 graphdriver
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.418512793Z" level=info msg="Daemon has completed
initialization"
```

```
Mar 31 02:01:46 nodeb dockerd[14336]: time="2019-03-31T02:01:46.430955768Z" level=info msg="API listen on
/var/run/docker.sock"
Mar 31 02:01:46 nodeb systemd[1]: Started Docker Application Container Engine.

ubuntu@nodeb:~$ docker container run --rm ubuntu:14.04 ip a show eth0

Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
e082d4499130: Pull complete
371450624c9e: Pull complete
c8a555b3a57c: Pull complete
1456d810d42e: Pull complete
Digest: sha256:6612de24437f6f01d6a2988ed9a36b3603df06e8d2c0493678f3ee696bc4bb2d
Status: Downloaded newer image for ubuntu:14.04
20: eth0@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc noqueue state UP group default
    link/ether 02:42:0a:01:0d:02 brd ff:ff:ff:ff:ff:ff
    inet 10.1.87.2/24 brd 10.1.87.255 scope global eth0
       valid_lft forever preferred_lft forever
ubuntu@nodeb:~$
```

2. Investigate

Investiate flannel Network

We saw earlier how flannel uses etcd to store information about its network. We are using many default settings, but all of those stem from the single network cidr we entered earlier. Now that flannel is up and running on both nodes, lets look at the networking contstructs.

On nodea, review the interfaces:

```
ubuntu@nodea:~/goProjects/src/github.com/coreos/flannel$ cd ~

ubuntu@nodea:~$ ip a show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:ef:63:d5:3b:be brd ff:ff:ff:ff:ff:ff
    inet 172.31.28.198/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ef:63ff:fed5:3bbe/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:4f:8a:e5:22 brd ff:ff:ff:ff:ff:ff
    inet 10.1.40.1/24 brd 10.1.40.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:4fff:fe8a:e522/64 scope link
       valid_lft forever preferred_lft forever
ubuntu@nodea:~$
```

review routing information:

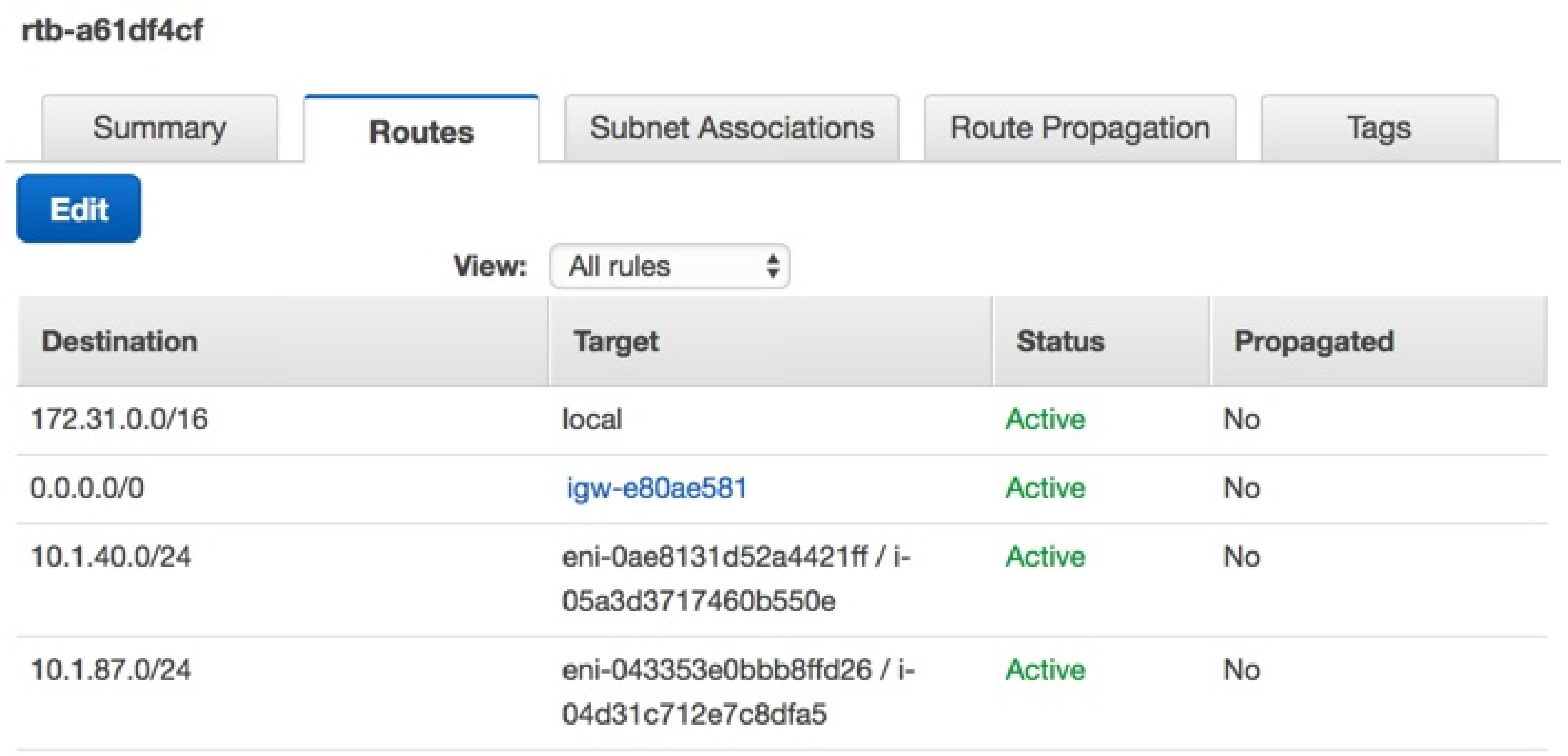
```
ubuntu@nodea:~$ ip route show

default via 172.31.16.1 dev eth0
10.1.40.0/24 dev docker0  proto kernel  scope link  src 10.1.40.1 linkdown
172.18.0.0/16 via 172.31.30.148 dev eth0
172.31.16.0/20 dev eth0  proto kernel  scope link  src 172.31.28.198
ubuntu@nodea:~$
```

Hmmm, not much information here. Do the same on nodeb and take note of the IP information.

```
ubuntu@nodeb:~$ ip a show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 02:d8:c0:66:a6:b8 brd ff:ff:ff:ff:ff:ff
    inet 172.31.30.148/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::d8:c0ff:fe66:a6b8/64 scope link
```

```
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:5f:57:41:79 brd ff:ff:ff:ff:ff:ff
    inet 10.1.87.1/24 brd 10.1.87.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:5fff:fe57:4179/64 scope link
       valid_lft forever preferred_lft forever

ubuntu@nodeb:~$ ip route show

default via 172.31.16.1 dev eth0
10.1.87.0/24 dev docker0  proto kernel  scope link  src 10.1.87.1 linkdown
172.17.0.0/16 via 172.31.28.198 dev eth0
172.31.16.0/20 dev eth0  proto kernel  scope link  src 172.31.30.148
ubuntu@nodeb:~$
```

In AWS, flannel can be configured to work with Amazon VPC. When we set network details in etcd, we told it to use the "type" "aws-vpc", so the routing table is stored there, for example:



rtb-a61df4cf

Summary | **Routes** | Subnet Associations | Route Propagation | Tags

Edit

View: All rules

Destination	Target	Status	Propagated
172.31.0.0/16	local	Active	No
0.0.0.0/0	igw-e80ae581	Active	No
10.1.40.0/24	eni-0ae8131d52a4421ff / i-05a3d3717460b550e	Active	No
10.1.87.0/24	eni-043353e0bbb8ffd26 / i-04d31c712e7c8dfa5	Active	No

This requires an AWS IAM policy to be applied to all of the instances:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateRoute",
                "ec2:DeleteRoute",
                "ec2:ReplaceRoute"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRouteTables",
                "ec2:DescribeInstances"
            ],
            "Resource": "*"
        }
    ]
}
```

This step was taken care of by the instructor.

To test that we are being routed correctly, try pinging the docker0 gateway between the nodes.

From nodea to nodeb.

```
ubuntu@nodea:~$ ping -c 1 10.1.87.1

PING 10.1.87.1 (10.1.87.1) 56(84) bytes of data.
64 bytes from 10.1.87.1: icmp_seq=1 ttl=64 time=0.363 ms

--- 10.1.87.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.363/0.363/0.363/0.000 ms
ubuntu@nodea:~$
```

and reverse

```
ubuntu@nodeb:~$ ping -c 1 10.1.40.1

PING 10.1.40.1 (10.1.40.1) 56(84) bytes of data.
64 bytes from 10.1.40.1: icmp_seq=1 ttl=64 time=0.404 ms

--- 10.1.40.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.404/0.404/0.404/0.000 ms
ubuntu@nodea:~$
```

Now that we have routing working, lets try with containers.

Testing With Docker

Overlay networks are used to augment traditional Docker networking, which belong to a single machine a-la docker0 bridge. Our goal is to contact the remote Docker container from another node.

On nodeb run the following command:

```
ubuntu@nodeb:~$ docker container run --rm -it -p 8889 ubuntu:14.04 nc -l 0.0.0.0 8889

...
```

In another terminal, get your container's IP address:

```
ubuntu@nodeb:~$ ID=$(docker container ls -q) && echo $ID

ff14668548fe

ubuntu@nodeb:~$ docker container inspect -f '{{ .NetworkSettings.Networks.bridge.IPAddress }}' $ID

10.1.87.2
ubuntu@nodeb:~$
```

On nodea run the following command using the IP address returned by the inspect command above:

```
ubuntu@nodea:~$ telnet 10.1.87.2 8889
Trying 10.1.87.2...
Connected to 10.1.87.2.
Escape character is '^]'.
```

```
hello!
```

```
ubuntu@nodeb:~$ docker container run --rm -it -p 8889 ubuntu:14.04 nc -l 0.0.0.0 8889


hello!
```

Try typing on either node's terminal. You have a simple chat application via flannel with Docker.

To exit this example, hit ctrl+] (`^]`) on nodea followed by `q` at the telnet prompt:

```
ubuntu@nodea:~$ telnet 10.1.87.2 8889
Trying 10.1.87.2...
Connected to 10.1.87.2.
Escape character is '^]'.
hello!
hi
^]
telnet> q
Connection closed.
ubuntu@nodea:~$
```

Now lets run a website on nodeb.

```
ubuntu@nodeb:~$ docker container run -p 80 nginx:1.9.1

...
```

In another terminal, get your container's IP address:

```
ubuntu@nodeb:~$ ID=$(docker container ls -q) && echo $ID

eff4c3ae4594
```

```
ubuntu@nodeb:~$ docker container inspect -f '{{ .NetworkSettings.Networks.bridge.IPAddress }}' $ID

10.1.87.2
ubuntu@nodeb:~$
```

Now on nodea:

```
ubuntu@nodea:~$ docker container run --rm -it centos

[root@5526a8acdf9b /]# curl --head 10.1.87.2
HTTP/1.1 200 OK
Server: nginx/1.9.1
Date: Thu, 03 May 2018 05:09:04 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 26 May 2015 15:02:09 GMT
Connection: keep-alive
ETag: "55648af1-264"
Accept-Ranges: bytes

[root@5526a8acdf9b /]
```

In your original terminal, you will see the request:

```
ubuntu@nodeb:~$ docker container run -p 80 nginx:1.9.1

172.31.7.235 - - [03/May/2018:05:08:44 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.29.0" "-"
```

Success! Stop both containers.

Testing With kubelet

Moving further up the chain, lets make sure this works with Kubernetes. We will use kubelet on nodea to launch a container, and in turn use nodeb to access it.

```
ubuntu@nodea:~$ vim poda.yaml
```

```
ubuntu@nodea:~$ cat poda.yaml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeName: nodea
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      hostPort: 80
ubuntu@nodea:~$
```

Launch on nodea with kubelet directly:

```
ubuntu@nodea:~$ sudo $HOME/k8s/_output/bin/kubelet \
--allow-privileged=true \
--runtime-cgroups=/systemd/machine.slice \
--kubelet-cgroups=/systemd/machine.slice \
--pod-infra-container-image=k8s.gcr.io/pause:3.1 \
--pod-manifest-path=$HOME/poda.yaml

...
```

Find the nginx container's IP (remember it is assigned to the "pause" container!):

```
ubuntu@nodea:~$ docker container ls

CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
NAMES
accf2668818d        nginx                  "nginx -g 'daemon of…"   5 minutes ago       Up 5 minutes
k8s_nginx_nginx-nodea_default_349055bae6506cdf584cb0ae1cefa4d7_0
0fd5138923c4        k8s.gcr.io/pause:3.1   "/pause"                 5 minutes ago       Up 5 minutes
0.0.0.0:80->80/tcp   k8s_POD_nginx-nodea_default_349055bae6506cdf584cb0ae1cefa4d7_0

ubuntu@nodea:~$ CIP=$(docker container inspect $(docker container ls -q --filter=ancestor=k8s.gcr.io/pause:3.1) \
-f '{{ .NetworkSettings.IPAddress }}') && echo $CIP
```

```
10.1.40.2
ubuntu@nodea:~$
```

On *nodeb*, lets access via curl.

```
ubuntu@nodeb:~$ curl --head $CIP

HTTP/1.1 200 OK
Server: nginx/1.15.10
Date: Sun, 31 Mar 2019 02:21:35 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 26 Mar 2019 14:04:38 GMT
Connection: keep-alive
ETag: "5c9a3176-264"
Accept-Ranges: bytes
ubuntu@nodeb:~$
```

We've successfully added flannel networking! Be sure to remove your on-cluster resources.

Congratulations you have successfully completed the lab!

At this point you can restart kubelet (with the proper flags) and all of the master and node services, using flannel to provide networking to your cluster! Below are the commands for the rest of the services on nodea and nodeb.

nodea:

api-server

```
sudo $HOME/k8s/_output/bin/kube-apiserver \
--etcd-servers=http://localhost:2379 \
--service-cluster-ip-range=10.0.0.0/16 \
--insecure-bind-address=0.0.0.0  \
--disable-admission-plugins=ServiceAccount
```

controller-manager

```
$HOME/k8s/_output/bin/kube-controller-manager --kubeconfig=nodea.conf
```

scheduler

```
$HOME/k8s/_output/bin/kube-scheduler --kubeconfig=nodea.conf
```

kubelet on nodea

```
sudo rm -rf /var/lib/kubelet

sudo $HOME/k8s/_output/bin/kubelet \
--kubeconfig=nodea.conf \
--config=nodea.yaml \
--allow-privileged=true \
--runtime-cgroups=/systemd/machine.slice \
--kubelet-cgroups=/systemd/machine.slice \
--pod-infra-container-image=k8s.gcr.io/pause:3.1
```

kube-proxy on nodea

```
sudo $HOME/k8s/_output/bin/kube-proxy --config=kube-proxy-config
```

nodeb:

kubelet on nodeb

```
sudo rm -rf /var/lib/kubelet

sudo $HOME/kube-bin/kubelet \
--kubeconfig=nodeb.conf \
--config=nodeb.yaml \
--allow-privileged=true \
--runtime-cgroups=/systemd/machine.slice \
--kubelet-cgroups=/systemd/machine.slice \
--pod-infra-container-image=k8s.gcr.io/pause:3.1
```

kube-proxy on nodeb

```
sudo $HOME/kube-bin/kube-proxy --config=kube-proxy-config
```