[Network] K8S Overlay Network 변경 실습 (IPIP -> VXLAN)

K8S Overlay Network 의 변경

IPIP -> VXLAN 변경 하기

주의) 다른 방식의 네트워크에서 생성된 서비스 POD가 있을 경우 프로덕션 레벨에서는 변경이 사실상 어렵다.
(pod 통신의 문제 발생에 대한 트러블슈팅이나 보장을 할 수가 없다)

Calico IP-IP Network 제거 후 VXLAN 으로 재 설치

Node : Controller / Worker01 / Worker02

```
## Controller
# Mode 확인 시 IPIPMODE 인 것을 확인할 수 있다
calicoctl get ippool -o wide
NAME                  CIDR              NAT    IPIPMODE
VXLANMODE   DISABLED   DISABLEBGPEXPORT   SELECTOR
default-ipv4-ippool   192.168.0.0/16   true   Always     Never
false      false
 all()

# 설치 Manifest YAML 을 이용하여 삭제한다.
kubectl delete -f calico.yml

## Contoller / Worker 공통
# 가비지 삭제 후 리부팅하면 tunl0 인터페이스가 사라진 것을 확인할 수 있다.
sudo rm -rf /var/run/calico/
sudo rm -rf /var/lib/calico/
sudo rm -rf /etc/cni/net.d/
sudo rm -rf /var/lib/cni/
sudo reboot


## Controller
# Manifest. calico.yaml 파일을 VXLAN 설치 타입으로 수정한다.
```

```
 livenessProbe:
            exec:
              command:
              - /bin/calico-node
              - -felix-live
               # - -bird-live        // VXLAN 은 bird(BGP) 를 사용하지 않으므로 주석 처리 필수
            periodSeconds: 10
            initialDelaySeconds: 10
            failureThreshold: 6
            timeoutSeconds: 10
          readinessProbe:
            exec:
              command:
              - /bin/calico-node
              - -felix-ready
               # - -bird-ready    // 주석 필수


            # Enable IPIP
            - name: CALICO_IPV4POOL_IPIP
              value: "Never"       // Always --> Never 로 변경한다
            # Enable or Disable VXLAN on the default IP pool.
            - name: CALICO_IPV4POOL_VXLAN
              value: "Always"         // Never --> Always 로 변경한다

kind: ConfigMap
apiVersion: v1
metadata:
  name: calico-config
  namespace: kube-system
data:
  # Typha is disabled.
  typha_service_name: "none"
  # Configure the backend to use.
  calico_backend: "vxlan"        // "bird" --> "vxlan" 으로 변경한다.


# 설치
kubectl apply -f calico.yaml
```

```
# Calico Node 체크. 모두 Ready 여야 한다. 조금 기다려보자.
kubectl get nodes -o wide -A

# Calico Pod 체크. kube-system PoD 에 에러가 있으면 안된다.
kubectl get pod -o wide -A

# Calico Type 체크. BIRD 는 꺼져야 정상
sudo calicoctl node status
Calico process is running.
The BGP backend process (BIRD) is not running.

# Network 체크 VXLANMODE 가 활성화되었다.
calicoctl get ippool -o wide
NAME                  CIDR              NAT    IPIPMODE
VXLANMODE   DISABLED   DISABLEBGPEXPORT   SELECTOR
default-ipv4-ippool    192.168.0.0/16    true    Never
Always        false        false
 all()

# 각 노드들의 tunl0 인터페이스가 사라지고 vxlan 인터페이스가 생성되었다.
# 노드의 vxlan 인터페이스 간 라우팅이 연결된 부분을 확인 가능하다.

hostway@controller:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric
Ref    Use Iface
0.0.0.0         10.10.10.1      0.0.0.0         UG    0      0
0 ens18
10.10.10.0      0.0.0.0         255.255.255.0   U     0      0
0 ens18  // External (SNAT)
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0
0 docker0 // Container Runtime Bridge
192.168.5.0     192.168.5.0     255.255.255.192 UG    0      0
0 vxlan.calico  // Worker01의 인터페이스 정보
192.168.30.64   192.168.30.64   255.255.255.192 UG    0      0
0 vxlan.calico  // Worker02의 인터페이스 정보
192.168.49.0    0.0.0.0         255.255.255.192 U     0      0
0 *              // Controller 의 vxlan 대역
192.168.49.1    0.0.0.0         255.255.255.255 UH    0      0
0 cali09ae4a7064b   // Node(Worker01)가 도달할 GW
192.168.49.2    0.0.0.0         255.255.255.255 UH    0      0
0 cali1fdac863dc5  // Node(Worker02)가 도달할 GW
```

```
# Worker 의 정보를 알고 있다.
hostway@controller:~$ ip nei | grep vxlan
192.168.5.0  dev  vxlan.calico  lladdr  66:8c:33:86:44:ce
PERMANENT
192.168.30.64  dev  vxlan.calico  lladdr  66:fb:72:20:22:a1
PERMANENT

# VXLAN Traffic Port 는 UDP 이다.
udp        0      0 0.0.0.0:4789            0.0.0.0:*

# PoD 생성 시 특이사항 없음
hostway@controller:~$ kubectl create deployment sampleos --
image=gcr.io/google-samples/kubernetes-bootcamp:v1     --
replicas=3
deployment.apps/sampleos created
hostway@controller:~$ kubectl get pod -o wide
NAME                          READY   STATUS    RESTARTS   AGE
IP              NODE       NOMINATED NODE   READINESS GATES
sampleos-646dc9654b-8xjw9   1/1     Running   0          45s
192.168.5.11    worker01   <none>           <none>
sampleos-646dc9654b-gxn75   1/1     Running   0          45s
192.168.5.10    worker01   <none>           <none>
sampleos-646dc9654b-snkxg   1/1     Running   0          45s
192.168.30.75   worker02   <none>           <none>


# VXLAN 패킷 확인하기
// Controller
1) worker01 과 worker02 에 위치하는 POD 간 Ping 을 날려본다.

hostway@controller:~$      kubectl      exec      -it
sampleos-646dc9654b-8xjw9 --  ping 192.168.30.75
PING 192.168.30.75: 56 data bytes
64 bytes from 192.168.30.75: icmp_seq=0 ttl=115 time=92.124 ms
64 bytes from 192.168.30.75: icmp_seq=1 ttl=115 time=79.735 ms
64 bytes from 192.168.30.75: icmp_seq=2 ttl=115 time=79.233 ms


2) 실제 통신은 물리적 인터페이스에서 진행되므로 실제 인터페이스에서 tcpdump
를 추출한다.
sudo tcpdump -i ens18 -w vxlan.pcap
```

3) Wireshark 로 확인한다. UDP 패킷을 확인할 수 있다.


[기타] CentOS 7 에 Kubernetes 기본 Install

CentOS 7 에 Kubernetes 기본 설치하기

OS : CentOS 7.6.1810 Minimal
Account : root
- 퍼블릭 SNAT IP 세팅
Controller : 10.10.10.237 SSH:4223
Worker-01 : 10.10.10.204 SSH:4224
Worker-02 : 10.10.10.190 SSH:4225

전 노드 공통

```
# root 로 초기 로그인. sudo 를 사용하기 위해 일반유저 생성 및 패스워드 지정
useradd -d /home/username username
echo "password" | passwd username --stdin

# 일반유저는 su 사용하지 못하도록 권한 조정
chmod 700 /usr/bin/su

# sudoer 의 기본 wheel 그룹에 계정 추가
sed -ie '/wheel/s/$/\:username/' /etc/group

# Timezone 설정
sudo timedatectl set-timezone Asia/Seoul

# SWAP OFF
sudo swapoff -a
sudo sed -i -e '/swap/d' /etc/fstab
```

```
# firewalld off
sudo systemctl stop firewalld && sudo systemctl disable
firewalld

# Selinux
setenforce 0
sudo  sed  -i  's/SELINUX=enforcing/SELINUX=disabled/g'
/etc/selinux/config

# Hostname
sudo hostnamectl set-hostname controller
sudo hostnamectl set-hostname worker-01
sudo hostnamectl set-hostname worker-02

## Controller / Worker
#curl -s https://get.docker.com | sudo sh
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

## Check
sudo docker -v
sudo docker ps -a

## Controller / Worker
sudo mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF

## Docker enable && restart
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

```
## Packages Repo
sudo cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes
-el7-x86_64
enabled=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

## Install
sudo yum install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes
```

Controller Init

```
# Controller. 일반적으로 컨트롤러 자신의 공인IP 를 통해 API 서버임을 알
린다 (Advertise)
sudo kubeadm init --ignore-preflight-errors=all --pod-network-
cidr=192.168.0.0/16 --apiserver-advertise-address=10.10.10.237

# Regular User Privileges
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

# Network Plugin Setting ( Calico )
curl
https://projectcalico.docs.tigera.io/manifests/calico.yaml -O
kubectl apply -f calico.yaml

# System Namespace ( kube-system ) check. CoreDNS 가 올라오면 정상
kubectl get pods -o wide -A

NAMESPACE     NAME                                       READY
STATUS     RESTARTS    AGE       IP                NODE
NOMINATED NODE   READINESS GATES
kube-system   calico-kube-controllers-7c845d499-p85pm   1/1
Running   0           3m6s    192.168.49.3   controller
```

```
<none>           <none>
kube-system   calico-node-fnm2q                        1/1
Running   0          3m6s    10.10.10.237   controller
<none>           <none>
kube-system   coredns-64897985d-cgvml                  1/1
Running   0          5m41s   192.168.49.2   controller
<none>           <none>
kube-system   coredns-64897985d-vdckf                  1/1
Running   0          5m42s   192.168.49.1   controller
<none>           <none>
kube-system   etcd-controller                          1/1
Running   0          5m54s   10.10.10.237   controller
<none>           <none>
kube-system   kube-apiserver-controller                1/1
Running   0          5m54s   10.10.10.237   controller
<none>           <none>
kube-system   kube-controller-manager-controller       1/1
Running   0          6m      10.10.10.237   controller
<none>           <none>
kube-system   kube-proxy-nn5zn                         1/1
Running   0          5m42s   10.10.10.237   controller
<none>           <none>
kube-system   kube-scheduler-controller                1/1
Running   0          5m54s   10.10.10.237   controller
<none>           <none>
```

(공통) Multi NIC 를 가지는 환경에서 INTERNAL-IP 설정
다수의 네트워크가 존재하는 서버의 경우 K8S 에서 첫 번째 NIC 의 IP 가 INTERNAL-IP 로 자동 설정된다.
해당 INTERNAL-IP 를 Init 시 설정한
kubeadm --apiserver-advertise-address 와 동일한 IP 대역을 사용하도록 수동 설정한다.

```
cat << EOF | sudo tee /etc/default/kubelet
KUBELET_EXTRA_ARGS='--node-ip $(hostname -I | cut -d ' ' -f2)'
EOF
sudo systemctl daemon-reload
sudo systemctl restart kubelet
kubectl cluster-info
```

Worker Join

```
# Worker-01과 Woker-02 Node 에도 User Privileges 설정 복사 및 적용한다
sudo scp /etc/kubernetes//admin.conf username@10.10.10.204:/home/username/admin.conf
sudo scp /etc/kubernetes//admin.conf username@10.10.10.190:/home/username/admin.conf

# Worker
mkdir -p $HOME/.kube
sudo cp -i ./admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# 각 Worker 에서 kubeadm 를 사용해서 Join 한다.
sudo kubeadm join 10.10.10.237:6443 --token jgocer.fu65ql39kdod5qi0 \
                        --discovery-token-ca-cert-hash sha256:3cb85267e89913d7865d219922daaa8fc6e788dd2be0e2f80fae27176e2dfe3b

# 토큰값을 나중에 재 확인해야 할 경우
kubeadm token create --print-join-command

# Check
kubectl get nodes -o wide
NAME         STATUS     ROLES                  AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION          CONTAINER-RUNTIME
controller   Ready      control-plane,master   16m   v1.23.5   10.10.10.237   <none>        CentOS Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://20.10.14
worker-01    Ready      <none>                 55s   v1.23.5   10.10.10.204   <none>        CentOS Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://20.10.14
worker-02    NotReady   <none>                 38s   v1.23.5   10.10.10.190   <none>        CentOS Linux 7 (Core)   3.10.0-1062.el7.x86_64   docker://20.10.14

# Check Pod Create
kubectl run hello --image=nginx --dry-run=client -o yaml |
```

```
kubectl apply -f-
pod/hello created
[myungin.baek@controller ~]$ kubectl get pods -o wide
NAME      READY    STATUS      RESTARTS    AGE    IP
NODE          NOMINATED NODE    READINESS GATES
hello   1/1     Running   0          42s   192.168.171.1
worker-01   <none>           <none>
```

[OS] CentOS 7 에서 iptables 로 초기 접근 제어하기

iptables 로 초기 접속 제어 하기

CentOS 7 기본 설치 후, 공인 네트워크 환경에 연결해야 할 때 우선 최소 SSH 접근만 허용하기

(Pre) CentOS 7 에서 firewalld 대신 iptables 사용하기

firewalld 사용이 권장되지만, iptables 에 익숙할 경우 iptables.target 을 service 에 연결하는 패키지를 운영할 수 있다.

```
# firewalld disable
systemctl stop firewalld && systemctl disable firewalld

# firewalld 대신 service 를 통한 운영.
# 설치 후 서비스 재시작하면 /etc/sysconfig/iptables 에 실시간 룰이 저장된다.
yum install iptables-services
service iptables reload
service iptables status

# 룰 저장
service iptables save
```

```
# 룰 리로드
service iptables reload

# 룰 형식으로 텍스트 저장. 반드시 -c ( ALL Rule ) 옵션을 지정해서 저장하여야 방화벽 정책과 ROUTE(NAT) 룰이 같이 저장된다.
iptables-save -c > rules.txt

# 룰 복원
iptables-restore < rules.txt
```

iptables 초기 설정 (모두 차단 후 특정 IP만 허용하기)

```
# 모든 룰 초기화
iptables -F

# lo 인터페이스 모든 ACCEPT
iptables -A INPUT -i lo -j ACCEPT

# 모든 접근 허용할 소스 IP 지정. 포트(SSH) 까지 지정할 경우 -p tcp (-m tcp 생략 가능) --dport 22 를 추가한다
iptables -A INPUT -s 1.2.3.4/32 -m comment --comment "회사" -j ACCEPT

# state 상태를 지정한 ACCEPT.
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# (선택) Ping request 요청에 대해 호스트 접근 불가 응답 설정. 서버가 존재한다는 힌트를 주는 것이다.
iptables -A INPUT -j REJECT --reject-with icmp-host-prohibited

# (선택) Ping request 요청에 거절 하고 싶다면.
iptables -A INPUT -p icmp --icmp-type echo-request -j REJECT

# (선택) Ping 접근 자체를 막고 싶다면 DROP. 허용하려면 ACCEPT 를 주면 된다.
iptables -A INPUT -p icmp -j DROP

# 그 외 모든 TCP 접근 DROP
iptables -A INPUT -p tcp -j DROP
```

```
# 저장
service iptables save
```

추가 접근 정책 추가

```
# 기본 설정 이후에 -A 로 추가하면 DROP 이후의 Line 에 들어가므로 Line 순서 상 제대로 허용되지 않는다
# -I INPUT [DROP Line] 을 기입하여 DROP 라인 위로 추가되도록 한다.

iptables -nL --line-number
------------------------------------------------------------------------
------------------
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  1.2.3.4                    0.0.0.0/0
/* 회사 */
2    DROP       tcp  --  0.0.0.0/0            0.0.0.0/0
------------------------------------------------------------------------
------------------

# 2번 DROP 정책 위로 추가한다.
iptables -I INPUT 2 -s 5.6.7.8 -j ACCEPT -m comment --comment "
허용 추가"
iptables -nL --line-number
------------------------------------------------------------------------
------------------
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all  --  1.2.3.4                    0.0.0.0/0
/* 회사 */
2    ACCEPT     all  --  5.6.7.8                    0.0.0.0/0
/* 허용 추가 */
3    DROP       tcp  --  0.0.0.0/0            0.0.0.0/0
------------------------------------------------------------------------
------------------

# 라인 정렬이 어려울 경우 /etc/sysconfig/iptables 파일을 직접 수정 후 reload 가능하다.
```

라인 정책 삭제

```
iptables -D INPUT [Number]
```

[토막] CNI – Calico Plugin

제목 : CNI 와 Calico Network

본문의 내용은 이후에도 업데이트 될 수 있음
내용의 결과값은 #1 에서 초기 구성한 (controller , worker) 시스템에서 진행

CNI (Container Network Interface)

CNCF 에서 정의한 컨테이너 네트워크 인터페이스.
Kubernetes는 기본적으로 Kubenet 이라는 CNI 를 준수하는 Network Plugin 을 갖고 있다.
하지만 복합적이고 다양한 구조의 네트워킹 사용 용도로는 다소 부족하여, 외부 네트워크 플러그인의 필요성이 대두되었다.

Calico Network?

vRouter 기반의 라우터(L3) 기반의 네트워크 관리 모델이다.
Kubernetes 의 서비스 Network 를 구성하는 CNI 를 준수하는 Network Plugin 중 하나이다.
쿠버네티스의 기본 네트워크 서비스는 기능적으로 부족한 부분이 있는데 자주 사용되는 외부 플러그인이다.

프로젝트 공식 Document URL : https://projectcalico.docs.tigera.io/reference/

Non-overlay 방식의 라우팅

Direct 방식
- BGP(Border Gateway Protocol) 인 BIRD 를 통해 다른 노드 간 라우팅 정보를 동적으로 공유하여 통신할 수 있다

즉 Pod 에서 다른 Pod 로 직접 네트워킹하는 방식으로 성능적으로 가장 뛰어나다.
각 Node 에 존재하는 Calico Pod 는 BGP를 처리하는 Peer 라고 볼 수 있다.
보통 같은 서브넷에 존재하는 노드들을 묶는 방법으로 사용된다. (ex: 라우터 밑의 여러 서버)

Overlay Network 방식의 라우팅

Workload IP(ex: 서비스 컨테이너) 를 모르는 다른 네트워크와의 통신 기법을 의미한다.
여러 노드의 네트워크 패킷을 캡슐화(Encaptulation) 하여 논리적인 공통(L2) 네트워크를 구성하는 방식이다.
캡슐화 패킷 패턴 : 외부 패킷에는 Node 간 IP 정보가 있고, 내부에는 POD 간의 IP 정보가 있다.

IP in IP 방식(Default)
- 각 노드의 네트워크 위치가 달라 Direct 방식으로 통신하기 어려운 경우
IP 패킷 정보를 캡슐화하여 tunl0(tunneling) 인터페이스를 거쳐 노드 간 통신을 가능하게 연결한다.
이후 Direct 와 마찬가지로 BGP (BIRD) 를 통해 노드 간 라우팅 정보를 다른 Node 와 동적(IPVS)으로 전달 및 갱신한다.
Calico Routing 기본 방식이다.

VXLAN 방식
- 가령 물리적 네트워크의 노드와 특정 퍼블릭 클라우드의 다른 네트워크를 연결할 경우
퍼블릭 클라우드 쪽의 라우터 설정 차이 등으로 IP in IP 방식을 쓰지 못할 수 있다. (ex: Azure)
이럴 경우 Calico 에서는 라우터를 거치는 BGP 프로토콜을 사용하지 않고 가상의 VXLAN 인터페이스를 Node 간에 생성하여
L2 캡슐화된 UDP 패킷을 서로 통신할 수 있게 하는 방법이다.
IP in IP 방식에 비해 패킷의 크기가 조금 더 크지만 대규모의 워크로드 시스템이 아닌 한 큰 체감은 없다.

Cross-subnet 방식
캡슐화가 필요한 경우(네트워크가 다를 경우)에서만 선택적인 캡슐화 구성도 가능하다.
(일부 구간/전체 구간)

패킷 암호화 (옵션)

WireGuard

Calico 는 모든 유형의 라우팅 방식에서 패킷 암호화 기능을 별도로 추가할 수 있다.
모든 네트워크 성능의 저하가 발생하게 되므로 적절한 계획 수립이 필요하다.

Calicoctl

Controller 에서 Calico Network 정보를 확인할 수 있는 유틸리티이다.
Host의 단독 명령어 또는 kubectl 에 plugin 타입으로도 설치할 수 있다.

```
# Host 의 명령어 타입으로 설치
$ cd /usr/local/bin
$ sudo curl -L https://github.com/projectcalico/calico/releases/download/v3.22.1/calicoctl-linux-amd64 -o calicoctl
$ sudo chmod +x calicoctl

# Check
Calico 가 관리하는 Network 의 Pool과 Block 을 확인할 수 있다.
$ sudo calicoctl ipam show --show-blocks
+----------+--------------------+-----------+------------+-----------+
| GROUPING |        CIDR        | IPS TOTAL | IPS IN USE | IPS FREE  |
+----------+--------------------+-----------+------------+-----------+
| IP Pool  | 192.168.0.0/16     |     65536 | 5 (0%)     | 65531 (100%) |
| Block    | 192.168.136.0/26   |        64 | 4 (6%)     | 60 (94%)     |
| Block    | 192.168.153.192/26 |        64 | 1 (2%)     | 63 (98%)     |
+----------+--------------------+-----------+------------+-----------+

BGP 프로토콜을 사용해 라우팅 정보를 전달중인 노드를 확인할 수 있다
$ sudo calicoctl node status
Calico process is running.
IPv4 BGP status
+---------------+-------------------+-------+----------+-------------+
|  PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
```

```
+----------------+-------------------+-------+----------+-----
--------+
| 203.248.23.215 | node-to-node mesh | up    | 05:27:05 |
Established |
+----------------+-------------------+-------+----------+-----
--------+
```

Block 에서 사용중인 관련 인터페이스들

```
$ route -n | egrep "tun|cali|\*"
192.168.136.0   0.0.0.0         255.255.255.192 U     0
0         0 *
192.168.136.1   0.0.0.0         255.255.255.255 UH    0      0
0 calibc6c3028870
192.168.136.2   0.0.0.0         255.255.255.255 UH    0      0
0 calid6edae09645
192.168.136.3   0.0.0.0         255.255.255.255 UH    0      0
0 calic6bfd11bfbe
192.168.153.192 203.248.23.215  255.255.255.192 UG    0      0
0 tunl0
```

어떤 Pod가 어떤 calicxxxxxx 인터페이스를 사용하는지 확인하려면 다음의 명령어를 사용한다
System(default) Namespace 까지 보이게 하려면 -A 옵션을 추가한다.

```
$ calicoctl get workloadendpoint -A
NAMESPACE     WORKLOAD                                  NODE
NETWORKS          INTERFACE
kube-system   calico-kube-controllers-56fcbf9d6b-nlqg2   user-
controller   192.168.136.2/32   calid6edae09645
kube-system   coredns-64897985d-jgj5s                   user-
controller   192.168.136.3/32   calic6bfd11bfbe
kube-system   coredns-64897985d-vbpn4                   user-
controller   192.168.136.1/32   calibc6c3028870
```

Calico 인터페이스는 Veth type(Pair) 이다.

```
$ ip -br -c link show type veth
calibc6c3028870@if3 UP                ee:ee:ee:ee:ee:ee
<BROADCAST,MULTICAST,UP,LOWER_UP>
calid6edae09645@if4 UP                ee:ee:ee:ee:ee:ee
<BROADCAST,MULTICAST,UP,LOWER_UP>
calic6bfd11bfbe@if4 UP                ee:ee:ee:ee:ee:ee
<BROADCAST,MULTICAST,UP,LOWER_UP>
```

Calico Management Pod

특정 Daemon이 아닌 Pod 형태로 존재한다.
Controller 와 Worker Node 에 각각 실행되는 Pod 가 다름을 체크한다.

```
$ kubectl get pods -o wide -n kube-system
NAME                                       READY   STATUS
RESTARTS   AGE   IP               NODE
calico-kube-controllers-56fcbf9d6b-nlqg2   1/1     Running   0
30m   192.168.136.2   user-controller
calico-node-8cts6                          1/1     Running   0
30m   10.0.2.15       user-controller
calico-node-mb9n6                          1/1     Running   0
29m   10.0.2.15       user-worker
```

Calico의 설정 정보 등의 DB 값은 etcd 라는 서비스의 datastore 에 저장된다.

```
$ kubectl get pods -o wide -n kube-system | grep -i etcd
etcd-user-controller                       1/1     Running   0
39m   10.0.2.15       user-controller
```

Calico Felix

쿠버네티스 Pod 네트워크에서 kube-proxy 상위 역할을 해 주는 컴포넌트이다.
etcd 에서 정보들을 읽어와서 Pod Network 의 라우팅 테이블과 룰을 관리한다.
kube-proxy 가 iptables / ipvs Mode 일경우 그에 맞춰 관리해준다.
iptables 라인이 너무 클 경우 성능에 영향이 있을 수 있어 플러그인에 의한 ipvs 방식이 선호된다.
 IPVS = Hash 값으로 동적으로 라우팅 테이블을 관리하는 로드밸런서 알고리즘

```
$ sudo iptables -t nat -S  | grep -i cali
$ sudo iptables -t filter -S  | grep -i cali
```

Networking 과정

IP in IP Networking

Controller Node 에서 Worker Node 에 있는 Pod 네트워크로 통신을 시도한다

1) Controller의 192.168.136.2 Pod 에서 Worker의 192.168.153.193 Pod 로 통신

2) Controller의 Pod 인터페이스(veth) 는 Pair 인 Host의 calico (veth) 인터페이스로 ARP 요청
3) Host 의 Calico 인터페이스는 Worker Pod 의 네트워크 정보를 모르므로 ARP 응답 할 수 없어야 함
4) 하지만 Calico 인터페이스는 link-local 타입(내부 브로드캐스트 학습)으로써 HOST 가 BIRD 에 의해 학습한 Worker 의 라우팅 정보를 알고 있음
5) Controller의 Calico 인터페이스는 vRouter에 설정된 ARP_Proxy 기능을 활용하여 Worker 의 ARP 응답을 대신 해 줌
6) BIRD는 Tunl0 --> Host 의 실제 인터페이스를 통해 상대 노드와 Pod 정보를 추가한 캡슐화 된 패킷으로 통신
7) 그림에는 보이지 않지만 만약 다른 노드와 통신이 아닌 퍼블릭 대역과 통신을 한다면
Felix 를 통한 SNAT (MASQUERADE) 를 설정하여 tunl0 을 거치지 않고 HOST 의 ens33 으로 바로 연결.

Packet Check

다른 네트워크 (Controllor POD <---> Worker POD) 간 Ping 체크를 진행한다

```
$ kubectl get pod -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP
NODE              NOMINATED NODE   READINESS GATES
hello-776c774f98-894tt    1/1      Running   0          13d
192.168.153.193   user-worker       <none>           <none>
hi                        1/1      Running   0          13d
192.168.136.5     user-controller   <none>           <none>
```

Worker POD --> Container POD. 컨테이너에 Ping 명령이 없어 Pod 밖의 Host 에서 컨테이너 PID 를 따서 진행했다.

```
$ sudo nsenter -t 225201 -n ping 192.168.136.5
64 bytes from 192.168.136.5: icmp_seq=627 ttl=62 time=0.709 ms
64 bytes from 192.168.136.5: icmp_seq=628 ttl=62 time=0.675 ms
64 bytes from 192.168.136.5: icmp_seq=629 ttl=62 time=0.727 ms
64 bytes from 192.168.136.5: icmp_seq=630 ttl=62 time=0.797 ms
64 bytes from 192.168.136.5: icmp_seq=631 ttl=62 time=0.887 ms
```

Controller 에서 패킷 캡쳐를 한다. IPIP 구조는 API 공인 인터페이스를 통해 터널링하므로, API 인터페이스든 터널링 인터페이스든 상관없다.

```
$ sudo tcpdump -i enp0s8 -nn proto 4 -w test.pcap
```

Wireshark 로 확인한다.

1) 두 POD IP의 다른 네트워트 간 정상 ICMP 통신을 하고 있는 부분을 확인할 수 있다.

2) MAC 확인시 Controller 와 Worker Node 의 API 공인IP 인터페이스 간 통신임을 확인했다.

```
# Controller
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:39:ce:bd brd ff:ff:ff:ff:ff:ff
     inet 203.248.23.214/25 brd 203.248.23.255 scope global enp0s8

# Worker
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:bc:85:3a brd ff:ff:ff:ff:ff:ff
     inet 203.248.23.215/25 brd 203.248.23.255 scope global enp0s8
```

3) IPv4 Protocol 확인 시 2개의 헤더를 확인할 수 있다.

공인망의 Outer IP 와 POD 네트워크인 Inner IP 2개를 확인할 수 있다.

Outer IP 헤더가 InnerIP 를 밖에서 캡슐로 감싼 IP-IP Protocol 임을 확인할 수 있다.

4) Messages 에서 통신에 정상 문제가 없음을 확인했다.

참고자료

Network Overlay : https://ikcoo.tistory.com/117

원격으로 CentOS 7 에서 내부 Windows 서버에 RDP 연결

CentOS 7 에서 Windows RDP 연결하기

진행 순서

```
# OS
CentOS 7.9 x86_64 minimal
로컬--> XRDP 를 통한 리눅스 GUI 접속 --> 리눅스 GUI 에서 내부 Windows RDP 접속
```

Linux GUI 환경 세팅

```
# GUI GroupInstall
root@localhost ~]# yum groups list | grep -i desktop
   Cinnamon Desktop
   MATE Desktop
   GNOME Desktop
   General Purpose Desktop
   LXQt Desktop
# GNOME 이 제대로 설치되지 않는다면 "Server with GUI" 그룹을 설치한다.
root@localhost ~] yum groupinstall "GNOME Desktop"

# GUI init
[root@localhost ~]# systemctl get-default
multi-user.target
[root@localhost ~]# systemctl set-default graphical.target
[root@localhost ~]# systemctl get-default
graphical.target
# Reboot 후 자동으로 GUI 모드로 부팅되는지 확인한다.
[root@localhost ~]# reboot
```

Linux 서버에 원격 접속 설정

```
# XRDP Install.
```

```
[root@localhost ~]# yum install epel-release
[root@localhost ~]# yum install xrdp
[root@localhost ~]# systemctl enable xrdp && systemctl start xrdp
```

selinux disable 및 iptables -F or 방화벽 정책에 tcp/3389 를 추가 해준다.

rdesktop 설치

컴파일러와 openssl-devel 이 패키지 컴파일 선행조건이다.

```
 yum -y install gcc openssl-devel
                                                                wget https://github.com/rdesktop/rdesktop/releases/download/v1.8.6/rdesktop-1.8.6.tar.gz
 tar xvzf rdesktop-1.8.6.tar.gz
 cd rdesktop-1.8.6/
 ./configure --disable-credssp --disable-smartcard
 make
 make install
```

Check

리눅스 서버에 RDP 로 접속 후, rdesktop -u [User] [ip] 입력한다.

```
root@localhost ~]# rdesktop -u administrator 10.10.10.5
Autoselected keyboard map en-us
Connection established using SSL.
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```