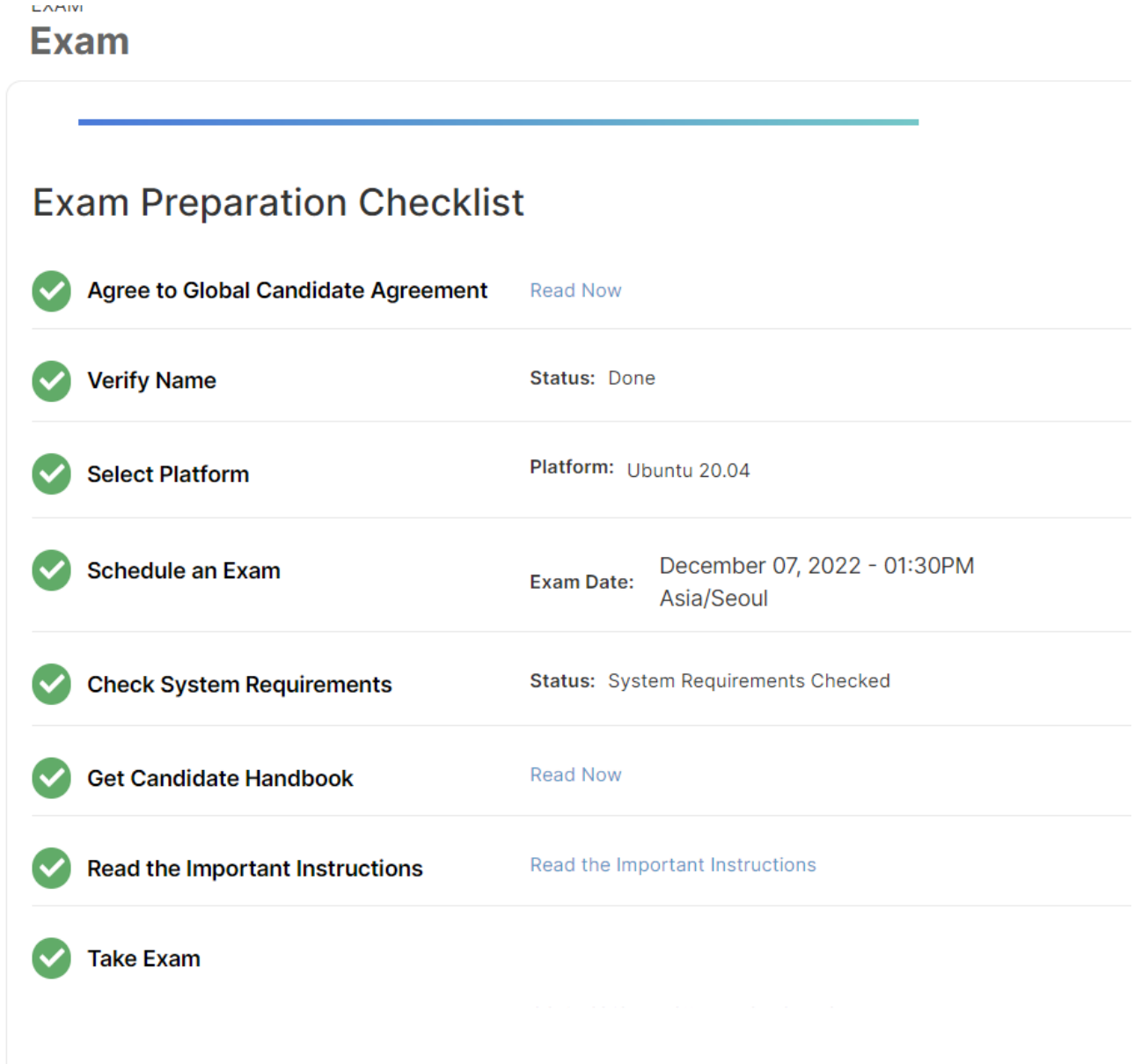
2022.12.07 CKA(Certified Kubernetes Administrator) 시험 응시 소감

2022.06.25 부터 바뀐 시험환경 (PSI Secure Browser) 로 바뀌었다고 하는데 금번에 시험을 치루면서 환경에 대한 소감을 정리 하였습니다;

관련 업데이트 URL : https://training.linuxfoundation.org/bridge-migration-2021/

시험 준비 – 최소 시험 시간 30분 전 입실 필수



CKA 는 원격 전용 시험이며 Schedule 된 시간 30분 전에

TAKE EXAM 버튼이 활성화 됩니다.

시험 시작 전 세팅 준비하는 시간으로 실제 시험시간에 카운팅되지는 않으나 늦어질 수록 실제 시험 시간이 늦어지므로 늦게 끝나게 됩니다.

 # 로그인 및 시험 등록은 해당 URL 에서 이루어 집니다.
https://trainingportal.linuxfoundation.org/learn/course/certified-kubernetes-administrator-cka/exam/exam

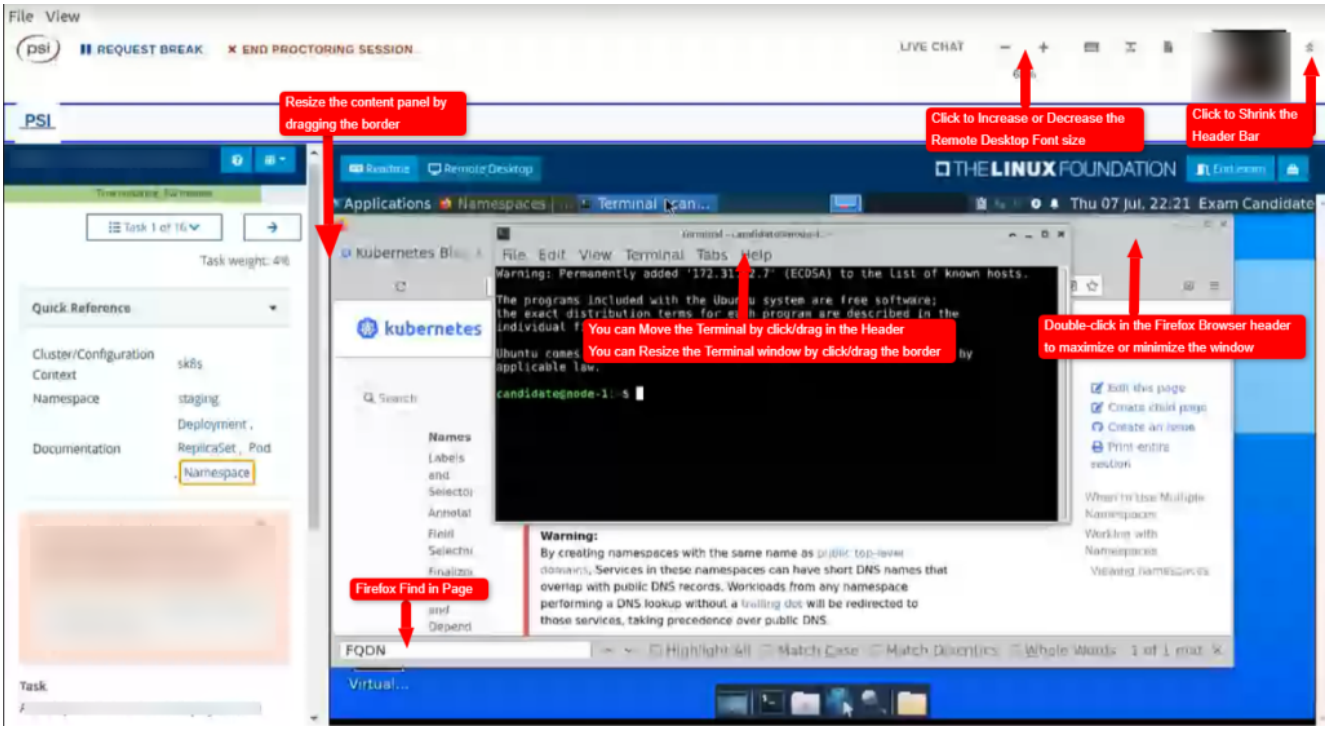
TAKE EXAM 을 누른 후 PSI 전용 브라우저를 설치할 수 있습니다.
클라이언트 환경에 Cotana, 원격(zoon, bluejean) 툴, VPN 등의 프로세스가 메모리에 떠 있을 경우 프로그램이 실행되지 않으므로 프로세스 종료해야 합니다
실행한 후 브라우저의 기능적인 신분증 촬영, 안면 촬영 등의 신원 확인 절차를 진행합니다.

이후 시험 감독관이 CHAT 으로 지시하는 내용들을 잘 따르면 됩니다.
 # 시험감독관은 저의 경우 다음의 상황을 요구했습니다
- 시험실 전체를 비추기
- 책상 밑 아래 비추기
- 책상에 노트북(데스크탑) 외 물건 치우기
(저의 경우 무선 마우스와, HDMI 용 집게를 치우라고 했습니다
마우스는 CHAT 으로 설명하고 허락 받았습니다)
- 셔츠 걷어서 팔, 손바닥(앞/뒤) 비춰보기
- 마스크 벗어서 보이지 않도록 칠판 스크린 뒤에 감추기

점검이 끝나면 협조에 고맙다고 하고 라벨이 달린 물병은 소지할 수 없음을 추가 안내합니다.

시험관이 원격에서 승인 시, 화면에 시험 랩이 나타나게 됩니다.
그 때부터 시간이 실제로 적용됩니다.

시험 환경 – 느림



저의 경우 회사 사무실에서 무선 와이파이로 진행했습니다.
외부 모니터는 지원하지 않기에 14인치 노트북을 사용하였는데
화면이 많이 작은데에다 해외망 원격 랩 시뮬이라 그런지 반응이 많이 느려 고생을 했습니다.

시험 진행시간은 2시간입니다.
문제는 총 17문제이며, 문제 내 2-3가지 연속된 과정을 물어볼 경우
부분점수가 배정됩니다.

단답형 문제는 2~4점
연속형 문제는 6~12점까지 배점이 매우 큽니다.
`Flag` 버튼이 있어서 문제를 순서대로 풀 필요는 없습니다.

시험 환경 `URL` :
`https://docs.linuxfoundation.org/tc-docs/certification/tips-cka-and-ckad#adjusting-font-and-windows-in-the-examui`

복사&붙여넣기

브라우저의 가상 실습 랩 안에서 `Firefox` 브라우저를 이용한
`Kubernetes.io` 의 `Document` 서치만 허용되는데
북마크 사용할 수 없는데에다 반응이 원활하지 않으므로
사전에 `YAML` 템플릿 위치 등을 숙지해 두는 것을 권장합니다.

복사-붙여넣기는 `Ctrl+Shift+C , V` 로 가능하다고 하지만
마우스 오른쪽으로 직접 `Copy&Paste` 를 권장합니다.

번역 & 플러그인

PSI 브라우저의 정해진 가상 랩 안에서 실행되는 것으로
크롬과는 달리 기본으로 번역 기능이 지원되지 않는 Firefox에
번역 플러그인을 설치할 수 없습니다.
문제의 영어 형식은 간단하기 때문에 크게 문제되지는 않을 것입니다.

시험관과는 CHAT 으로만 소통이 가능합니다.
화장실을 가기 전 반드시 전달하고 가야 하며 시험시간은 그대로 흘러갑니다.

점수

총점 100점에서 **66점(2/3) 이상 획득시 Pass** 입니다.
24시간 이내에 메일 및 결과 확인할 수 있게 되어 있습니다.
저의 경우에는 정확히 24시간 뒤 결과가 회신되었습니다.

Fail 시 Reschedule 기회를 한번 더 준다고 합니다.

바로바로 해결하기 어려운 문제는 Flag 찍어놓고 다른 문제부터 빠르게 푸는 것을 권장합니다.
반응도 느리고 망설이면 생각보다 시간이 부족했습니다.

기타

시험 접수 시 이름을 한글로 했을 경우
실제 자격증에 이름이 표기가 되지 않을 수 있습니다.
여권/신용카드에 기재된 영문 네임과 동일하게 접수 권장드립니다.

Kubernetes 1.24 + cri-docker Installation (kubeadm)

우분투에서 Kubernetes 최신 버전 설치하는 방법입니다.
18.04 에서 시험되었으나 상위버전에서도 차이는 없습니다.

Controll Node (node01) , Worker Node (node02 ,node03) 3EA 구성입니다.
모든 설치는 root 가 아닌 user(worker)을 사용하며 sudo 를 사용합니다

(Pre-Option) 설치 패키지 캐쉬 초기화

```
sudo kill -9 $(lsof -t /var/lib/dpkg/lock)
sudo kill -9 $(lsof -t /var/lib/apt/lists/lock)
sudo kill -9 $(lsof -t /var/cache/apt/archives/lock)
sudo rm /var/lib/apt/lists/lock
sudo rm /var/cache/apt/archives/lock
sudo rm /var/lib/dpkg/lock
sudo dpkg --configure -a
```

(Require) SWAPOFF

```
# SWAP-On 시 퍼포먼스 이슈가 있어 일반적으로 제거
sudo swapoff /swap.img
sudo sed -i -e '/swap.img/d' /etc/fstab
```

(유의사항) Container Runtime Install

k8s 1.24 (2022/05) 이후
k8s 에서 기본적으로 내부 연결 지원해주던 dockershim 이 제거되어
cri-docker 추가 설치하여 도커를 k8s에 연결하는 작업이 필요

Pre-Setting – All node

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo systemctl enable --now docker && sudo systemctl status
docker --no-pager
sudo usermod -aG docker worker
sudo docker container ls

# cri-docker Install
VER=$(curl                                                -s
https://api.github.com/repos/Mirantis/cri-dockerd/releases/lat
est|grep tag_name | cut -d '"' -f 4|sed 's/v//g')
echo $VER
wget
https://github.com/Mirantis/cri-dockerd/releases/download/v${V
ER}/cri-dockerd-${VER}.amd64.tgz
tar xvf cri-dockerd-${VER}.amd64.tgz
sudo mv cri-dockerd/cri-dockerd /usr/local/bin/
```

```
# cri-docker Version Check
cri-dockerd --version

wget
https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/
packaging/systemd/cri-docker.service
wget
https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/
packaging/systemd/cri-docker.socket
sudo mv cri-docker.socket cri-docker.service
/etc/systemd/system/
sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-
dockerd,' /etc/systemd/system/cri-docker.service

sudo systemctl daemon-reload
sudo systemctl enable cri-docker.service
sudo systemctl enable --now cri-docker.socket

# cri-docker Active Check
sudo systemctl restart docker && sudo systemctl restart cri-
docker
sudo systemctl status cri-docker.socket --no-pager

# Docker cgroup Change Require to Systemd
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

sudo systemctl restart docker && sudo systemctl restart cri-
docker
sudo docker info | grep Cgroup

# Kernel Forwarding
```

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

sudo sysctl --system
```

Packages Install – All node

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

# Update 해야 패키지 인식함
sudo apt-get update

# k8s 설치
sudo apt-get install -y kubelet kubeadm kubectl

# 버전 확인하기
kubectl version --short

Client Version: v1.24.3
Kustomize Version: v4.5.4

# 버전 고정하기
sudo apt-mark hold kubelet kubeadm kubectl
```

k8s Init – Controller Node (Node01)

```
# Controller Node
sudo kubeadm config images pull --cri-socket unix:///run/cri-dockerd.sock

sudo kubeadm init --ignore-preflight-errors=all --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=211.115.207.136 --cri-socket /var/run/cri-dockerd.sock
## --apiserver-advertise-address=203.248.23.161 -> Controller 서버 IP.

# 설치 시 출력되는 join Command 를 복사해둔다.
kubeadm join 211.115.207.136:6443 --token r8gfco.6s7f60dns4vgwcc0 \
                        --discovery-token-ca-cert-hash sha256:7b0f82be076748e67f8615eab0b86a61317bac397f94b2921810231ab14afdcc

# kubeadm 을 root 처럼 사용하기 위한 추가 설정
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# Token 기억이 안 날 경우
kubeadm token create --print-join-command

# 서비스 확인 ( Ready , Running )
kubectl get nodes -o wide
kubectl get pod -A

# Worker Node에서 kubectl 을 사용하려면 admin.conf 를 복사한다
sudo scp /etc/kubernetes/admin.conf worker@node02:/home/worker/admin.conf
sudo scp /etc/kubernetes/admin.conf worker@node03:/home/worker/admin.conf
```

Node Join – Worker Node (Node02 , Node03)

```
# Worker Node Join. --cri-docker sorket 수동 지정해서 가입해야 한다. init 시 나온 토큰값을 참고한다. 기억이 안 날 경우 커맨드로 확인한다
```

```
sudo kubeadm join --token <token> <controlplane-host>:<controlplane-port> --ignore-preflight-errors=all --cri-socket unix:///var/run/cri-dockerd.sock
# sudo kubeadm join [Controller IP]:6443 --token 3nvrgw.33k750dq9klm5omi --discovery-token-ca-cert-hash sha256:b10158bcea37aae0e92ed6b68b4dd1e8213623cc7d406e77eef55fe6196fe346 --cri-socket /var/run/cri-dockerd.sock

# Join
sudo kubeadm join [Controller IP]:6443 --token r8gfco.6s7f60dns4vgwcc0 \
                        --discovery-token-ca-cert-hash sha256:7b0f82be076748e67f8615eab0b86a61317bac397f94b2921810231ab14afdcc --cri-socket /var/run/cri-dockerd.sock

# Check
systemctl status kubelet

# worker 계정에 kubectl 사용 환경변수 추가
cd ~
mkdir -p $HOME/.kube
sudo cp -i ./admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

추가 후 node 에서도 kubectl get nodes -o wide 등을 사용할 수 있게 됨

Pod Network Install – Controller Node (Node01)

```
# 예전 버전에는 CNI Plugin 을 셋업하지 않으면 CoreDNS 가 Pending 이었는데
지금은 기본 kube-system-proxy 로도 구성이 잘 되는 것 같다.

# CNI Mode 확인하기
# Check - kube-proxy 이름 확인한다. kube-proxy 는 Node 마다 존재하므로 3개일 것이다.
kubectl get pod -A
kubectl logs -f pod/kube-proxy-[name] -n kube-system

# Log 로 확인 시 - Linux 커널의 iptables Mode 인 것으로 확인되었다.
kubectl logs -f pod/kube-proxy-6dqp8 -n kube-system | grep mode
I0805 06:32:48.444077       1 server_others.go:578] "Unknown
```

```
proxy mode, assuming iptables proxy" proxyMode=""
I0805 06:32:48.469028       1 server_others.go:213] "kube-proxy running in dual-stack mod
" ipFamily=IPv4
```

Pod Network Plugin Install
다양한 네트워크 플러그인이 있으며, 해당 CKA 과정 글에서는 Callico 의 Plugin 을 사용한다.
Calico Network는 IPIP , VXLAN 등을 지원하는 IPVS 모델이다

```
kubectl create -f https://projectcalico.docs.tigera.io/manifests/tigera-operator.yaml
curl https://projectcalico.docs.tigera.io/manifests/custom-resources.yaml -O
kubectl create -f custom-resources.yaml
kubectl get pods -A
```

Control-Plane (Controller) Node 일 경우 Taint (해당 노드 생성 금지) 영향으로 calico-controller-pod가 Pending 걸릴 수 있는데 다음과 같이 실행

Taint Check
```
kubectl describe node node01 | grep Taints

kubectl taint nodes --all node-role.kubernetes.io/master-
kubectl taint nodes node01 node-role.kubernetes.io/control-plane:NoSchedule-
```

PoD check
```
kubectl get pod -A

#kubectl describe pods/calico-kube-controllers-657d56796-xxxxx -n calico-system
```

Calicoctl 로 Status 확인하기
```
cd /usr/local/bin
sudo curl -L https://github.com/projectcalico/calico/releases/download/v3.23.3/calicoctl-linux-amd64 -o calicoctl
sudo chmod +x calicoctl
```

```
## CNI Type Check
calicoctl get ippool -o wide

## Block Check
sudo calicoctl ipam show --show-blocks

## BGP Protocol Check
sudo calicoctl node status

## Node Endpoint Check
calicoctl get workloadendpoint -A
```

Rejoin or Reset

(Trouble) 설치 중 에러 발생시 클린 삭제 후 재 설치를 진행한다

```
# All Node
sudo systemctl stop kubelet
sudo kubeadm reset -f --cri-socket /var/run/cri-dockerd.sock

sudo rm -rf ~/.kube
sudo rm -rf /root/.kube
sudo rm -rf /var/lib/etcd
sudo rm -rf /etc/kubernetes
```

Calico Network 만 삭제하고 싶다면

```
# ALL Node
kubectl delete -f custom-resources.yaml
kubectl delete -f tigera-operator.yaml
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 rm -rf /opt/cni
sudo reboot
```

[Container Runtime] CRI-O 설치 가이드

CRI-O 설치 가이드

CRI-O ?

CRI-O는 OCI 호환 컨테이너 런타임 인터페이스(CRI)다.
가장 일반적으로 사용되는 런타임은 Docker였으나 Kubernetes 1.24 버전부터 Docker용 CRI(컨테이너 런타임 인터페이스) shim이 더 이상 사용되지 않아(Docker Engine은 지원) CRI-O 설치 가이드를 작성한다.

구성 서버 (VM)
Controller Server : 1EA
Worker Server : 1EA

OS
Ubuntu 20.04 Server Minimal

지원되는 버전 확인 : https://github.com/cri-o/cri-o/blob/main/install.md#supported-versions
쿠버네티스 설치 참조 : https://tech.hostway.co.kr/2022/02/06/418/

Traffic Setup

.conf 파일을 만들어 부팅 시 모듈을 로드한다

Controller / Worker

```
cat <<EOF | sudo tee /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
```

```
sudo modprobe br_netfilter

# 요구되는 sysctl 파라미터 설정, 이 설정은 재부팅 간에도 유지된다.

# Controller / Worker

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sudo sysctl --system
```

cgroup driver

```
# CRI-O는 기본적으로 systemd cgroup 드라이버를 사용한다.

cgroupfs cgroup 드라이버로 전환하려면, /etc/crio/crio.conf 를 수정하거나 /etc/crio/crio.conf.d/02-cgroup-manager.conf 에 드롭-인(drop-in) 구성을 배치한다.

[crio.runtime]
conmon_cgroup = "pod"
cgroup_manager = "cgroupfs"
```

CRI-O 설치

```
# Controller / Worker

sudo -i
export OS=xUbuntu_20.04 # OS 버전
export VERSION=1.19     # cri-o 버전

echo                                                "deb
https://download.opensuse.org/repositories/devel:/kubic:/libco
ntainers:/stable/$OS/                  /"                  >
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
echo                                                "deb
http://download.opensuse.org/repositories/devel:/kubic:/libcon
```

```
tainers:/stable:/cri-o:/$VERSION/$OS/      /"      >
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-
o:$VERSION.list

curl                                                    -L
https://download.opensuse.org/repositories/devel:kubic:libcont
ainers:stable:cri-o:$VERSION/$OS/Release.key | apt-key add -
curl                                                    -L
https://download.opensuse.org/repositories/devel:/kubic:/libco
ntainers:/stable/$OS/Release.key | apt-key add -

sudo apt-get update

sudo apt-get install cri-o cri-o-runc

sudo systemctl daemon-reload
sudo systemctl enable crio --now
sudo systemctl status crio
```

Controller Node Initialize

```
# docker의 경우 여러 런타임이 설치되어있어도 1순위로 감지되어 구성하나 CRI-
O의 경우 따로 지정해주어야 함

# Controller

kubeadm init --cri-socket=/var/run/crio/crio.sock --ignore-
preflight-errors=all  --pod-network-cidr=192.168.0.0/16  --
apiserver-advertise-address=203.248.23.192

# Worker

kubeadm     join     203.248.23.192:6443     --token
9oy7rn.qtw04nfd8ga417p4    --discovery-token-ca-cert-hash
sha256:26218e0c80320b7c23735916c130fc48f644b26314212d917969553
ec0651256
```

나머지 설치 단계는 쿠버네티스 설치 링크를 참조한다.

[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
```

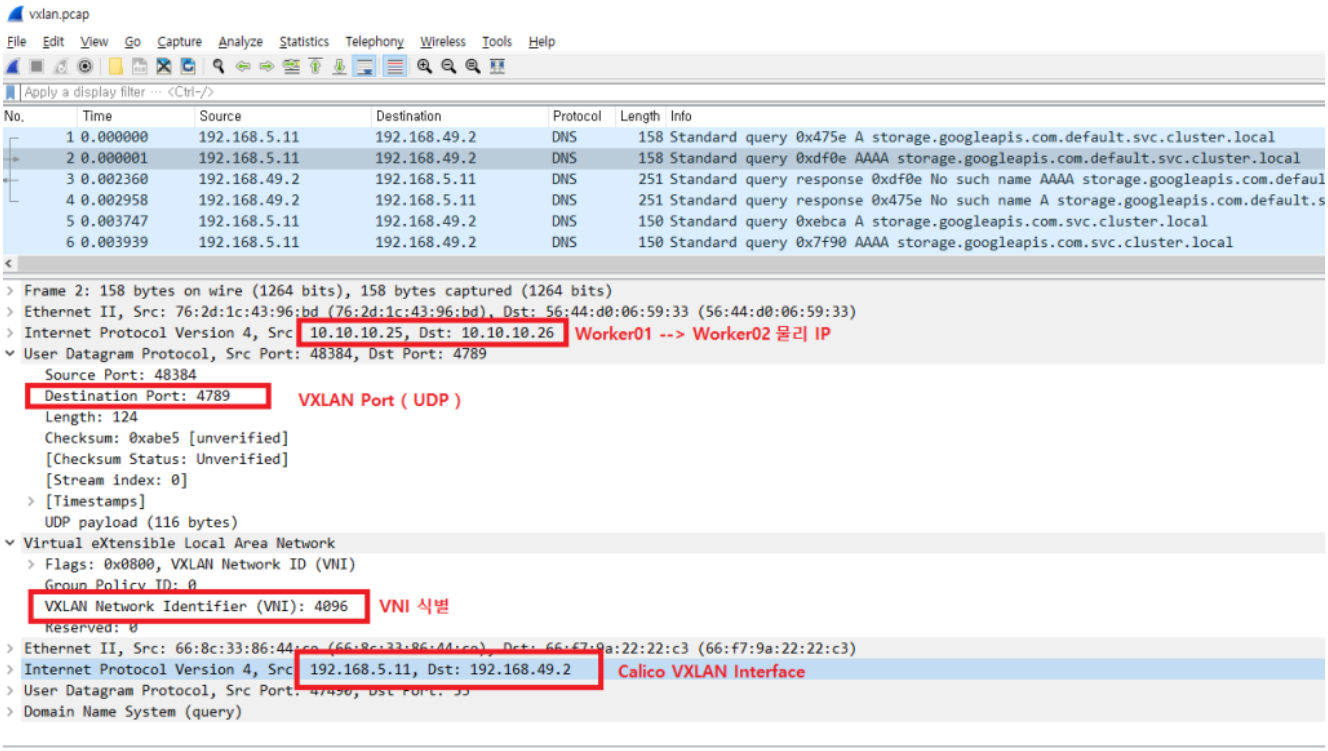
```
## Controller
# Manifest. calico.yaml 파일을 VXLAN 설치 타입으로 수정한다.

 livenessProbe:
            exec:
              command:
              - /bin/calico-node
              - -felix-live
               # - -bird-live        // VXLAN 은 bird(BGP) 를 사용
하지 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 변경한
다
            # Enable or Disable VXLAN on the default IP pool.
            - name: CALICO_IPV4POOL_VXLAN
              value: "Always"         // Never --> Always 로 변
경한다

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 변경
한다.
```

```
# 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 알린다 (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
sha256:3cb85267e89913d7865d219922daaa8fc6e788dd2be0e2f80fae271
76e2dfe3b

# 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>
```

[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

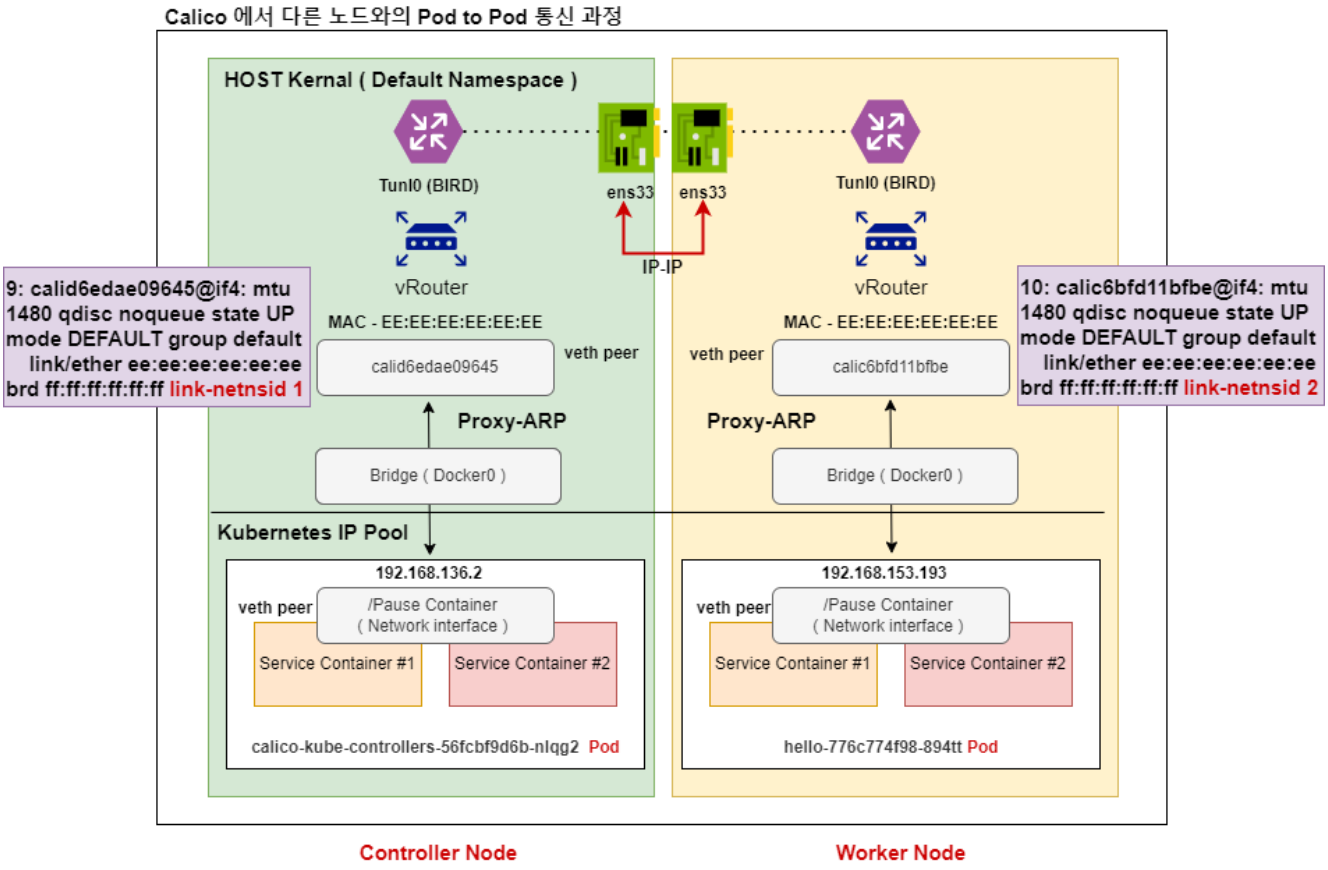
쿠버네티스 Pod 네트워크에서 kube-proxy 상위 역할을 해 주는 컴포넌트이다.
etcd 에서 정보들을 읽어와서 Pod Network 의 라우팅 테이블과 룰을 관리한다.
kube-proxy 가 iptables / ipvs Mode 일경우 그에 맞춰 관리해준다.
iptables 라인이 너무 클 경우 성능에 영향이 있을 수 있어 플러그인에 의한 ipvs 방식이 선호된다.
 IPVS = Hash 값으로 동적으로 라우팅 테이블을 관리하는 로드밸런서 알고리즘

```
$ 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>
```

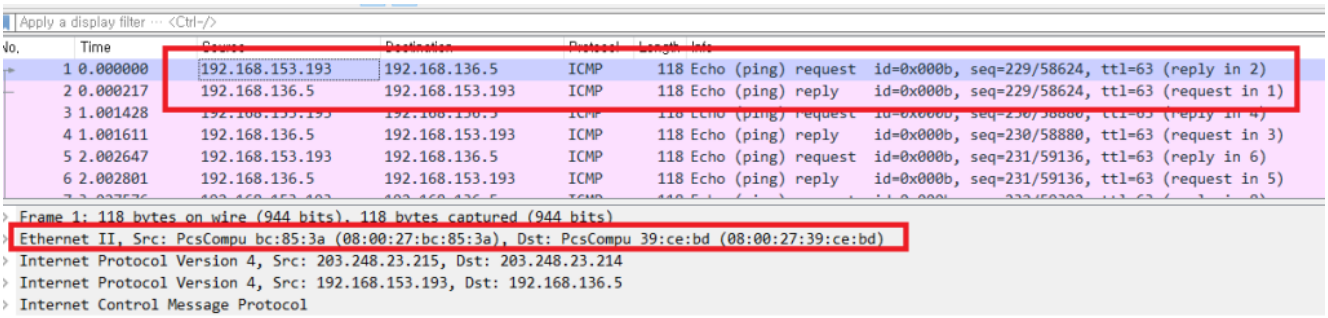
Worker POD --> Container POD. 컨테이너에 Ping 명령이 없어 Pod 밖의 Host 에서 컨테이너 PID 를 따서 진행했다.

```
$ 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.

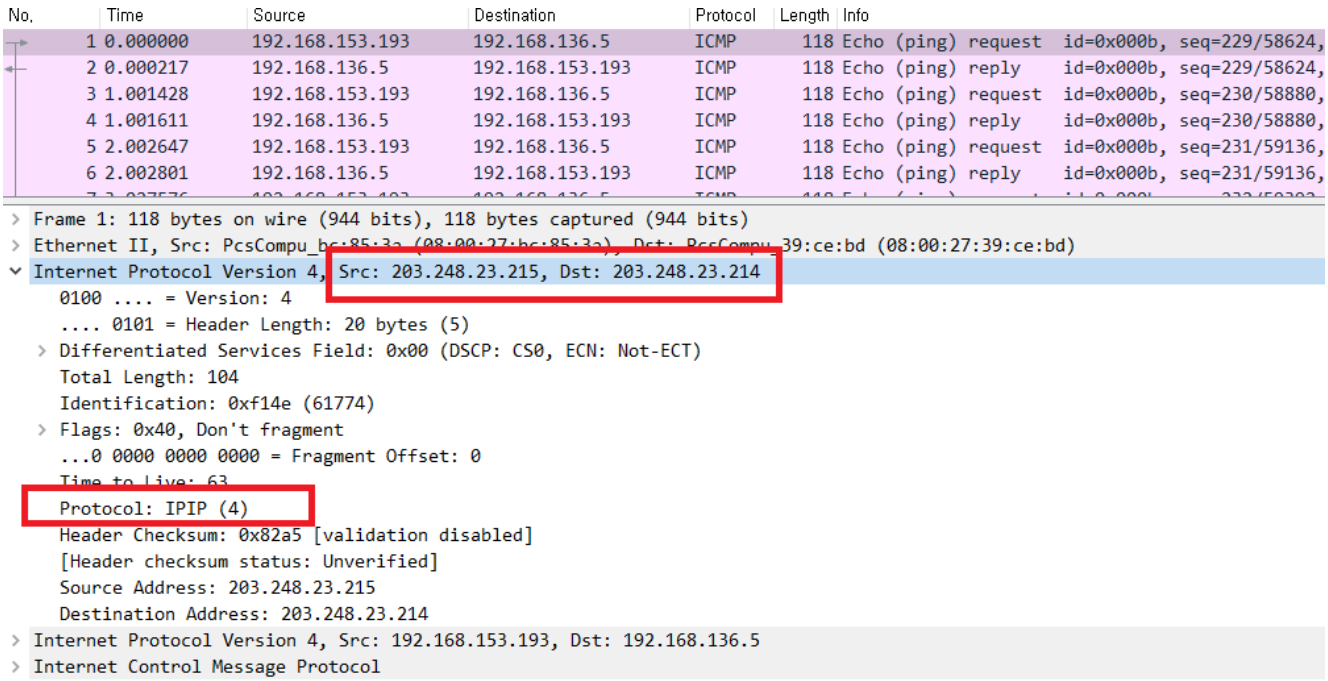


2) MAC 확인시 Controller 와 Worker Node 의 API 공인IP 인터페이스 간 통신임을 확인했다.

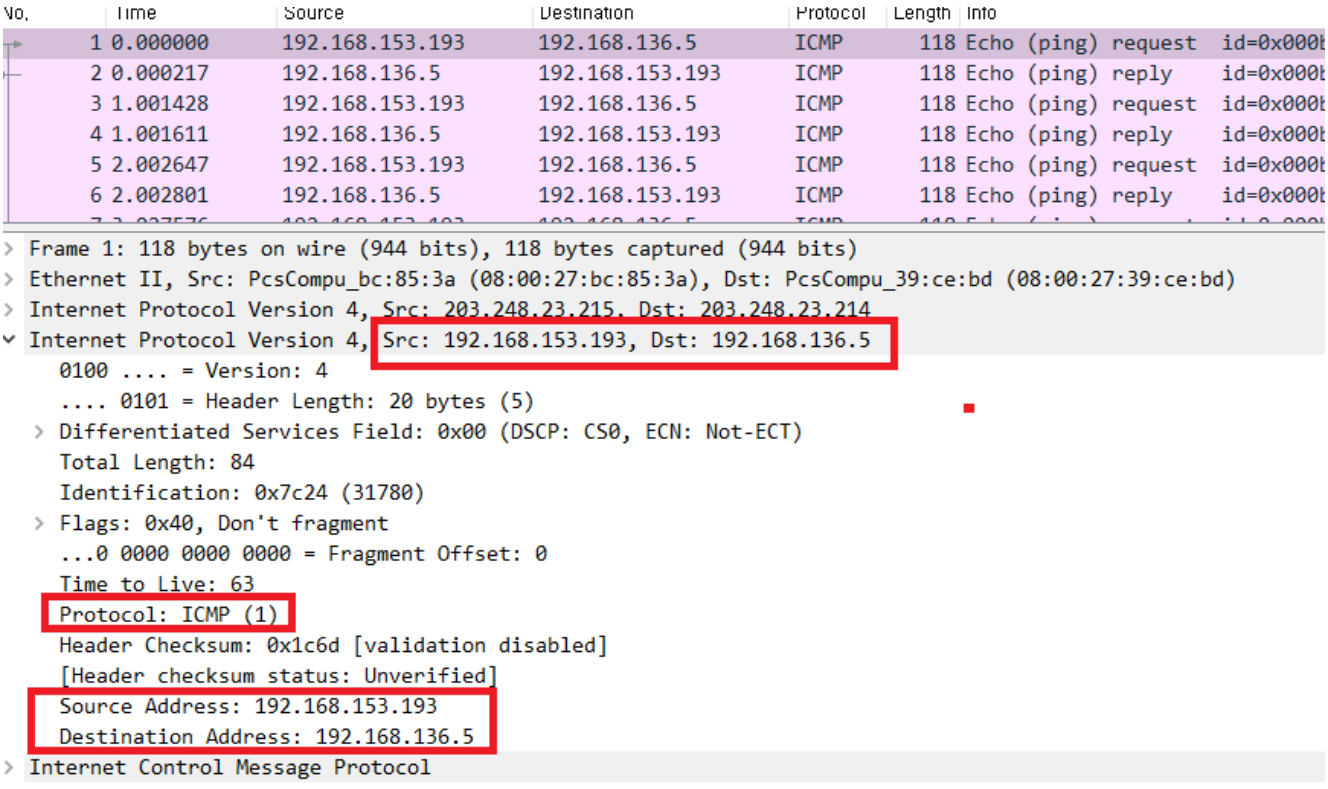
```
# Controller
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:39:ce:bd brd ff:ff:ff:ff:ff:ff
     inet 203.248.23.214/25 brd 203.248.23.255 scope global enp0s8

# 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.

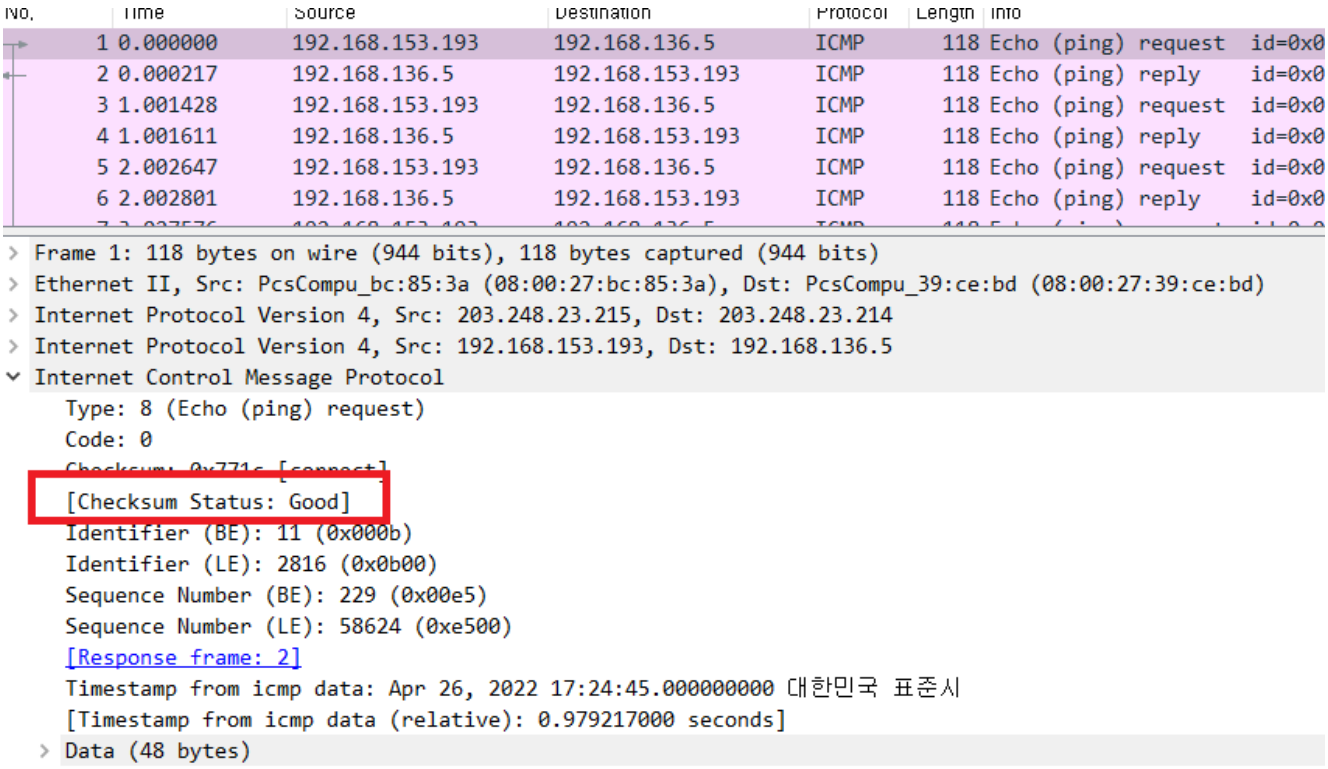
공인망의 Outer IP 와 POD 네트워크인 Inner IP 2개를 확인할 수 있다.



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

[CKA] #2. Pod - 1

[CKA] #2. Pod (1)

※ 본 글은 작성 후에도 기재 순서의 변경 및 내용 업데이트 예정

CKA 시험은 오픈북 형식으로 크롬에서 몇 가지 URL 를 허용해준다.
대부분의 커맨드 및 설명은 공식문서에 있으나, 시험 중에 바로바로 찾으려면 평소에 숙달이 되어야 한다
주요 명령어의 북마크 등을 미리 설정해 놓는 것도 좋을 것 같다.

Pod ?

Kubernetes 에서 최소 배포 단위

최소 옵션으로 생성 시, Pod 라는 하나의 공간 안에 서비스 Container 와 Pod의 Network 정보를 가지고 있는 /pause 컨테이너로 기본적인 생성이 된다.
즉 Pod 라는 최소 단위 안에는, 컨테이너가 한 개 이상 및 볼륨 등의 다른 서비스도 존재할 수 있다.
외부에서는 Pod 의 Network 만 인지하고 있겠지만, /pause 가 내부 컨테이너들을 포트로 구분하는 역할을 해준다.

Pod Create

```
# 실행 전에 yaml 형식의 사전 테스트 모드인 dry run 을 항상 실행해보는 것이 좋다
$ kubectl run hello --image=nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: hello
  name: hello
spec:
  containers:
  - image: nginx
    name: hello
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

# 실제 실행은 위 명령어를 | (파이프)를 사용하여 실제 적용(apply) 의 - 표준 입력으로 넘겨 사용 하는 형식도 권장한다

kubectl run hello2 --image=nginx --dry-run=client -o yaml |
kubectl apply -f -
kubectl get pods -o wide

# nodeName 을 추가하여 원하는 노드에 pod 생성이 가능하다.
이 방식은 Taint 의 스케줄을 무시한다.
yaml 파일에 nodeName 을 추가 지정해 주면 된다.
$ kubectl run hi --image=nginx --dry-run=client -o yaml  >
hi.yaml
apiVersion: v1
kind: Pod
```

```
metadata:
  creationTimestamp: null
  labels:
    run: hi
  name: hi
spec:
  containers:
  - image: nginx
    name: hi
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  nodeName: user-controller      ## Node 기입
status: {}

# 정의된 YAML 을 이용한 Pod 생성
$ kubectl create -f hi.yaml
$ kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE
IP               NODE              NOMINATED NODE   READINESS
GATES
hello-776c774f98-894tt   1/1     Running   0          18h
192.168.153.193   user-worker       <none>           <none>
hi                       1/1     Running   0          3m10s
192.168.136.5     user-controller   <none>           <none>
```

Pod status 확인하기

```
$ kubectl describe pod hi

# Pod 에 직접 접근해보기. -- kubectl 의 arg 와의 옵션 구분을 위해 -- 를 사용이 권장되는 추세이다
$ kubectl exec -it hi -- /bin/bash
root@hi:/#
```

어플리케이션 Pod 배포하기

```
# 다중 수량 옵션인 --replicas 를 지정할 수 있다. deployment 옵션이 커맨드에 들어간다.
$ kubectl create deployment web --image=nginx --replicas=3
```

```
deployment.apps/web created
$ kubectl get pods -o wide | grep -i web
web-76b56fd968-c2pk9     1/1     Running   0          11s
192.168.153.217   user-worker       <none>          <none>
web-76b56fd968-chr4w     1/1     Running   0          11s
192.168.136.6     user-controller   <none>          <none>
web-76b56fd968-mmdfn     1/1     Running   0          11s
192.168.153.218   user-worker       <none>          <none>
```

Pod log 확인하기

```
# Pod info 확인
kubectl describe pod hi

# Pod log 확인
kubectl logs hi

# journal Log(kubelet) 확인
sudo journalctl -u kubelet

# 하위 레벨 Log 확인.
hi POD 내의 Container 탐색.
/pause 컨테이너가 Pod의 포워딩 네트워크 정보. nignx 컨테이너가 서비스 컨테이너이다
$ sudo docker ps -a | grep -i hi
d507d8b298c3   nginx                     "/docker-
entrypoint.…"       26  hours  ago      Up  26  hours
k8s_hi_hi_default_6a1464a1-0fea-4ff8-a5c6-426afe281173_0
d8fb1a992247    k8s.gcr.io/pause:3.6          "/pause"
26     hours     ago               Up     26     hours
k8s_POD_hi_default_6a1464a1-0fea-4ff8-a5c6-426afe281173_0

# nginx Service Container info
$ sudo docker inspect d507d8b298c3
$ sudo docker logs d507d8b298c3

# Pod Network info
$ sudo docker inspect d8fb1a992247
$ sudo docker logs d8fb1a992247

# Container 레벨에서 접근해보기. 서비스 컨테이너만 접근 가능하다
```

```
$ sudo docker exec -it d507d8b298c3 ls
bin   docker-entrypoint.d   home   media  proc  sbin  tmp
boot  docker-entrypoint.sh  lib    mnt    root  srv   usr
dev   etc                   lib64  opt    run   sys   var

$ sudo docker exec -it d507d8b298c3 /bin/bash
root@hi:/#
```

컨테이너 내에 Net 관련 명령어가 없어 불편하다면 nsenter 를 사용해본다.
ip 명령이 없어서 확인이 어렵다
```
$ $ sudo docker exec -it d507d8b298c3 ip addr
OCI runtime exec failed: exec failed: container_linux.go:380: starting container process caused: exec: "ip": executable file not found in $PATH: unknown
```

컨테이너 PID 추출한다
```
$ sudo docker inspect --format '{{ .State.Pid }}' d507d8b298c3
1244489
```

nsenter 를 사용하여 해당 PID 를 대상으로 클라이언트(pod namespace가 위치한) 레벨의 명령어 실행이 가능하다.
```
$ sudo nsenter -t 1244489 -n ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    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
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
4: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP group default
     link/ether c6:3d:04:5d:80:82 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.136.5/32 scope global eth0
       valid_lft forever preferred_lft forever
```

Pod delete

일반 Pod 삭제

```
$ kubectl delete hi

# 어플리케이션 Pod Delete. replicas 로 다중 배포한 모두가 삭제된다.
$ kubectl delete deployment web
deployment.apps "web" deleted
$ kubectl get pods -o wide | grep -i web
-
```

참고 자료

Pod : https://kubernetes.io/docs/concepts/workloads/pods/
Pod Networking : https://www.digitalocean.com/community/tutorials/how-to-inspect-kubernetes-networking

---

[CKA] #1. 쿠버네티스 설치하기

제목 : [CKA] #1. 쿠버네티스 설치하기

Kubenertes 설치 방식

쿠버네티스는 여러가지 방식으로 설치할 수 있으나 CKA 과정을 고려한하여 공식 오픈문서의 kubeadm 방식으로 설치를 진행하였다.

구성 서버 (VM)
Controller Server : 1EA
Worker Server : 1EA

OS
Ubuntu 20.04 Server Minimal

```
# SWAP 은 기본적으로 해제하는 것을 권장하고 있다.
sudo swapoff /swap.img
sudo sed -i -e '/swap.img/d' /etc/fstab
```

호스트네임을 설정한다. 모든 작업은 일반 계정(regular user)에서 sudo 를 사용하여 권한을 빌려 실행하는 것을 권장한다.
```
sudo hostnamectl set-hostname controller
sudo hostnamectl set-hostname worker
```

Traffic Setup

컨테이너 런타임(예: Docker), kube-proxy에서 사용하는 브릿지 네트워크의 트래픽을 확인할 수 있도록 iptables 커널 옵션을 지정한다.

```
## Container / Worker
```
이 모듈을 로드하면, 브릿지를 통과하는 트래픽이 netfilter(iptables) 를 거치도록 활성화된다.
```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

Container Runtime 설치

쿠버네티스의 기본 단위인 POD 에서 컨테이너를 구동하는 런타임은 여러 종류가 존재하지만 CKA 과정 상 Docker 를 설치하였다.
쿠버네티스는 외부 추가 서비스/모듈의 일관성을 보장하지는 않는다.

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

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

cgroup 드라이버 설정

cgroup 은 런타임 프로세스의 리소스를 제약할 수 있는 관리자이다.
설치한 OS의 cgroup 기본 관리자인 systemd 와, docker, kubelet 의 cgroupfs 관리가 별개로 작동하는 혼란을 방지하기 위해 systemd 로 통합해 주는 과정이다.

```
## 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

## Docker 의 cgroup driver 확인 시, 기본 cgroupfs 에서 systemd 로 변경된 것을 확인할 수 있다.
sudo docker info | grep -i cgroup

 Cgroup Driver: systemd
 Cgroup Version: 1
```

쿠버네티스 패키지 설치

레포지토리 구성 후 kebe 관련 관리/명령줄 패키지를 설치한다. 일반적으로 모든 서버에 설치한다.
에러 발생시 한줄씩 진행한다.

```
## Controller / Worker
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-
```

```
keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Kube 컨트롤 노드의 InitIalize.

Controller Node 의 시작을 위해 init 하는 과정이다. 옵션을 통해 구성을 지정할 수 있다.

--cri-socket : 지정하지 않을 경우 kubeadm이 socket 으로 내부에 설치된 컨테이너 런타임을 자동 감지하여 구성한다. 다중으로 설치되어 있으면 에러가 발생한다.
--pod-network-cidr : pod 간 통신할 수 있는 network 를 지정한다. 내부 CoreDNS Service 의 조건이다.
--apiserver-advertise-address=<ip-address> : 지정하지 않으면 Controller 의 기본 네트워크 인터페이스를 사용하여 API서버를 선언한다.

```
## Controller. 일반적으로 컨트롤러 자신의 공인IP 를 통해 API 서버임을 알린다 (Advertise)
sudo kubeadm init --ignore-preflight-errors=all --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=203.248.23.192
```

정상적으로 init 되면 하단에 아래의 메세지가 나온다.

1) 설명과 같이, 일반 유저(regular user) + sudo 권한으로 cluster를 사용하고 있다면, 아래의 환경 변수를 실행한다.

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

## Check
kubectl get nodes
```

```
NAME                STATUS     ROLES                  AGE
VERSION
user1-controller    NotReady   control-plane,master   6m28s
v1.23.5
```

2) pod network 를 사용하기 위해 Network Plugin 을 설치할 수 있다.

```
You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the
options listed at:
        https://kubernetes.io/docs/concepts/cluster-
administration/addons/
```

Pod Network 통신이 구성되지 않으면 CoreDNS 가 작동하지 않는다(Pending)

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                READY
STATUS    RESTARTS   AGE
kube-system   coredns-64897985d-9sj9j             0/1
Pending   0          12m
kube-system   coredns-64897985d-zfl8q             0/1
Pending   0          12m
kube-system   etcd-user1-controller               1/1
Running   0          12m
kube-system   kube-apiserver-user1-controller     1/1
Running   0          12m
kube-system   kube-controller-manager-user1-controller   1/1
Running   0          12m
kube-system   kube-proxy-g5xdv                    1/1
Running   0          12m
kube-system   kube-scheduler-user1-controller     1/1
Running   0          12m
```

Pod Network Plugin Install

다양한 네트워크 플러그인이 있으며, 해당 CKA 과정 글에서는 Callico 의 Plugin 을 사용한다.

```
curl
https://projectcalico.docs.tigera.io/manifests/calico.yaml -O
kubectl apply -f calico.yaml
kubectl get nodes
```

Check
플러그인 설치 후, coredns의 status 가 Running 되는지 체크한다.

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY
STATUS     RESTARTS   AGE
kube-system   calico-kube-controllers-56fcbf9d6b-bnxz5   0/1
Pending    0          20s
kube-system   calico-node-khp2h                          0/1
Init:2/3   0          20s
kube-system   coredns-64897985d-9sj9j                    0/1
Pending    0          22m
kube-system   coredns-64897985d-zfl8q                    0/1
Pending    0          22m
kube-system   etcd-user1-controller                      1/1
Running    0          22m
```

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.

```
# INTERNAL-IP 가 10.0.2.15 ( Calico Network Default ) 로 되어 있다
$ kubectl get nodes -o wide
NAME              STATUS   ROLES                  AGE
VERSION     INTERNAL-IP     EXTERNAL-IP     OS-IMAGE
KERNEL-VERSION      CONTAINER-RUNTIME
user-controller   Ready     control-plane,master   44h
v1.23.5   10.0.2.15     <none>         Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
user-worker       Ready    <none>                 44h
v1.23.5   10.0.2.15     <none>         Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14

# Controller.  상황에 따라 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.  상황에 따라 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

# Check 시 Internal-IP 가 advertise 인터페이스로 변경되었다.
$ kubectl get nodes -o wide
NAME               STATUS   ROLES                  AGE
VERSION    INTERNAL-IP        EXTERNAL-IP    OS-IMAGE
KERNEL-VERSION     CONTAINER-RUNTIME
user-controller    Ready     control-plane,master    45h
v1.23.5   203.248.23.214   <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
user-worker        Ready    <none>                  44h
v1.23.5   203.248.23.215   <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
```

Worker 에서 Controller 에 Join

Then you can join any number of worker nodes by running the following on each as root:
root 로 실행하라고 안내하고 있으나
일반 유저로 Worker 에서 kebeadm을 실행하고 싶다면 Controller 의 /etc/kebenertes/admin.conf 파일을 Worker 로 복제 후 동일 환경 구성을 해준다.

```
# Controller
sudo          scp          /etc/kubernetes//admin.conf
vagrant@203.248.23.193:/home/vagrant/admin.conf

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

kubeadm     join     203.248.23.192:6443     --token
```

```
wy11vq.bk2rze7g9lilg2d9 \
                        --discovery-token-ca-cert-hash
sha256:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b
2c598d3f7
```

오류가 발생할 경우 ignore 옵션을 추가해 본다

```
sudo kubeadm join 203.248.23.192:6443 --token
wy11vq.bk2rze7g9lilg2d9 --ignore-preflight-errors=all --
discovery-token-ca-cert-hash
sha256:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b
2c598d3f7

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a
response was received.
* The Kubelet was informed of the new secure connection
details.

Run 'kubectl get nodes' on the control-plane to see this node
join the cluster.
```

Check

위의 환경 설정을 했을 경우 Worker 에서도 일반 유저로 pod 확인이 가능하다

```
kubectl get nodes
NAME                STATUS   ROLES                  AGE
VERSION
user1-controller    Ready    control-plane,master   33m
v1.23.5
user1-worker        Ready    <none>                 84s
v1.23.5

kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY
STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-56fcbf9d6b-bnxz5   1/1
Running   0          11m
kube-system   calico-node-khp2h                          1/1
Running   0          11m
kube-system   calico-node-skdjl                          1/1
Running   0          2m3s
kube-system   coredns-64897985d-9sj9j                    1/1
```

```
Running   0          33m
kube-system   coredns-64897985d-zfl8q                    1/1
Running   0          33m
kube-system   etcd-user1-controller                      1/1
Running   0          33m
kube-system   kube-apiserver-user1-controller            1/1
Running   0          33m
kube-system   kube-controller-manager-user1-controller   1/1
Running   0          33m
kube-system   kube-proxy-g5xdv                           1/1
Running   0          33m
kube-system   kube-proxy-m6ztf                           1/1
Running   0          2m3s
kube-system   kube-scheduler-user1-controller            1/1
Running   0          33m
```

(Trouble) 설치 중 에러 발생시 클린 삭제 후 재 설치를 진행한다

```
# All Node
sudo systemctl stop kubelet
sudo kubeadm reset -f

sudo rm -rf ~/.kube
sudo rm -rf /root/.kube
sudo rm -rf /var/lib/etcd
```

Network Plugin Status

앞서서 Pod Network 를 관리하는 애드-온 플러그인으로 Calico 를 설치하였는데, 간단한 Status 명령을 확인할 수 있다.
별도 명령어(calicoctl) 설치도 가능하고, Kubectl 의 옵션 플러그인으로 설치할 수도 있는데 전자로 진행했다.

```
# 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
$ calicoctl ipam show --show-blocks
+----------+--------------------+-----------+------------+----
----------+
| GROUPING |        CIDR        | IPS TOTAL | IPS IN USE |
IPS FREE   |
+----------+--------------------+-----------+------------+----
----------+
| IP Pool  | 192.168.0.0/16     |     65536 | 8 (0%)     |
65528 (100%) |
| Block    | 192.168.136.0/26   |        64 | 3 (5%)     | 61
(95%)     |
| Block    | 192.168.153.192/26 |        64 | 5 (8%)     | 59
(92%)     |
+----------+--------------------+-----------+------------+----
----------+
```

Kubernetes Auto Complation

```
# 주요 명령어의 alias 및 Tab 키를 사용한 자동완성을 권장한다
echo '' >>~/.bashrc
echo 'source <(kubectl completion bash)' >>~/.bashrc
echo 'alias k=kubectl' >>~/.bashrc
echo 'complete -F __start_kubectl k' >>~/.bashrc

. ~/.bashrc

# Check
## Tab 자동완성 확인
k get nodes -o wide
kubectl get nodes -o wide
```