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 늦어질수록 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 안

내합니다.

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.

[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.

[토막] Network Namespace 간단한 구성

기준 : CentOS 7.6.1810
유저 : root 사용자

Network Namespace

가상화 또는 클러스터 시스템에서 자주 보이는 용어입니다.
앞서서 컨테이너를 사용하기 위한 격리 조건 중 하나로 언급이 되었었는데요.
논리적인 Network Space(공간)을 생성 후 네트워크 인터페이스, 라우팅, IP 라는 네트워크 요소들을 넣어
Host의 네트워크와 격리되는 공간을 말합니다. 다양한 환경의 네트워크들을 구분하고 통신하기 위해 활용됩니다.

Default Network Namespace Check

```
# Host의 디폴트 Network Namespace 와 소유자를 확인
$ lsns -t net -o pid,uid,user,command
  PID   UID USER  COMMAND
    1     0 root  /sbin/init maybe-ubiquity
```

로컬 Host 의 디폴트 네트워크는 PID 1 (Init) 에서 실행되는 것을 확인할 수 있습니다.
일반적으로 우리가 보는 nic(예: eth0) 과 lo 인터페이스가 속한 기본 네임스페이스입니다.

Create Network Namespace

새 네임스페이스를 생성하여, 별도의 격리된 네트워크 영역을 만들 수 있습니다.
새 네임스페이스는 lo 인터페이스만 가지고 있으며 다른 네임스페이스와 통신할 수 없는 영역입니다.

```
# test 라는 이름을 가진 Namespace 를 생성 후 리스트 확인
$ ip netns add test

$ ip netns
test

# Check
만들어진 네임스페이스를 사용하는 PID 가 없기 때문에 lsns 목록에서 보이지 않습니다
$ lsns -t net
PID USER TYPE COMMAND
```

```
  1 root net   /usr/lib/systemd/systemd --switched-root --
system --deserialize 22
```

Namespace의 Network 연결 1 – 가상 인터페이스 생성

가상의 Network Namespace 간의 통신은 리눅스 커널에 존재하는 가상 인터페이스 기능인 veth 를 사용합니다.
veth 는 기본적으로 HOST <---> 다른 네임스페이스와의 연결 성립을 위한 한 쌍 단위로 구성됩니다.

```
# HOST에 가상 인터페이스를 추가 생성합니다. veth type 에 peer 로 pair 구성입니다.
$ ip link add veth0 type veth peer name veth1

# HOST에 veth0/veth1 이라는 2개의 가상 인터페이스가 생성되었습니다.
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@veth0      DOWN
veth0@veth1      DOWN
```

Namespace의 Network 연결 2 – 새 네임스페이스에 가상 인터페이스 할당

디폴트 영역에 통신을 위한 가상 인터페이스를 생성 후, 이제 새로 만든 test 네임스페이스에 달아 줄 수 있습니다.

```
# veth0 인터페이스를 test Namespace 에 Set
$ ip link set veth0 netns test

# 생성 후 HOST 에서 veth0 은 test namespace 에 장착했기 때문에 사라졌습니다.
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
```

```
veth1@if5        DOWN
```

test namespace 의 가상 인터페이스를 확인해 보면 veth0 이 장착된 것을 확인할 수 있습니다.

```
# test namespace 에 netns exec 옵션을 통한 인터페이스 조회 커맨드 실행
$ ip netns exec test ip -br addr
lo               DOWN
veth0@if4        DOWN
```

Namespace의 Network 연결 3 - bridge 연결

HOST 와 test namespace 간에 veth0과 veth1 인터페이스가 각각 세팅되었지만 DOWN 상태로 아직 통신을 할 수는 없습니다.
논리적인 가상 인터페이스 간 통신을 위해서는 IP 할당이 필요한데
이번 글에서는 HOST 영역에 브릿지 인터페이스(bridge) 를 사용해 연결해 보겠습니다.
호스트의 네트워크를 복잡하게 만들지 않으려면 브릿지라는 가상 스위치에 네임스페이스를 연결하는 것이 좋습니다.

```
# Check
$ (명령어가 없다면) yum install -y bridge-utils-1.5-9.el7.x86_64
$ btctl show
bridge name     bridge id               STP enabled
interfaces
```

현재 브릿지가 없는 것을 확인했습니다. br0 이라는 이름으로 로컬 HOST에 생성을 합니다.

```
# Bridge Create && Check
$ ip link add br0 type bridge

$ brctl show
bridge name     bridge id               STP enabled
interfaces
br0             8000.000000000000       no

# 브릿지 생성 추가 확인
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
```

```
ens36           UP              192.168.0.2/24
veth1@if5       DOWN
br0             DOWN
```

하지만 여전히 br0 과 vethx 간에 어떤 연결을 설정하지 않았기 때문에 통신할 수는 없습니다.
첫 단계로 신규 네임스페이스에 설정한 인터페이스와 호스트의 브릿지를 연결해 줍니다.

```
# HOST 에 존재하는 veth1 과 Host 의 br0 브릿지를 연결합니다.
$ ip link set veth1 master br0

# check 시 bridge 에 veth1 이 추가된 것을 확인할 수 있습니다.
$ brctl show
bridge name     bridge id               STP enabled
interfaces
br0             8000.46df623e69e4       no              veth1
```

가장 서두에도 말했지만, 네트워크의 통신 요소는 인터페이스, 라우팅, IP 입니다.
통신을 위해 두 번째 단계로 이제 IP 를 설정해야 합니다.
ifconfig 같이 별도 net-util 을 설정하는 것보다 ip 명령어를 사용하는 것이 수월합니다.

```
# netns exec 명령을 사용하여 test Namespace 의 veth0 인터페이스에 IP 를 할당하고 UP 합니다.
$ ip netns exec test ip addr add 10.10.10.2/24 dev veth0
$ ip netns exec test ip link set veth0 up

# 이제 host 의 veth1 인터페이스와 bridge 인터페이스도 up 해 줍니다.
$ ip link set br0 up
$ ip link set veth1 up

# UP check 시 이제 브릿지와 인터페이스가 모두 UP 된 것을 확인할 수 있습니다.
$ ip -br -c addr
lo              UNKNOWN         127.0.0.1/8
ens33           UP              211.239.150.48/23
ens36           UP              192.168.0.2/24
veth1@if5       UP
br0             UP

# test namespace 의 인터페이스 UP 도 확인됩니다.
$ ip netns exec test ip link
```

```
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: veth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default qlen 1000
     link/ether f2:1c:09:d4:47:fc brd ff:ff:ff:ff:ff:ff link-
netnsid 0

# lo 인터페이스가 DOWN 이면 해당 네임스페이스의 내부 통신을 할 수가 없습니다. UP 이후 UNKNOWN 으로 바뀌면 됩니다.
$ ip netns exec test ip link set dev lo up
$ ip netns exec test ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000



# Check
$ ip netns exec test ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.058 ms

# Check 2
Host 의 브릿지와 신규 네임스페이스 간에 통신되는지 확인하려면 브릿지에
Gateway 와 같은 동일 대역 IP할당을 합니다
브릿지에 Routing 정보가 없으면, 다른 네임스페이스의 사설 인터페이스 ip를 찾아갈 수 없습니다.

# 브릿지에 통신할 네임스페이스와 같은 대역의 IP 를 할당하여 라우팅 정보를 넣습니다.
$ ip addr add 10.10.10.200/24 dev br0

# test 네임스페이스의 veth0 인터페이스로 Ping 을 날려봅니다.
$ ping 10.10.10.2
 ping 10.10.10.2 -c 2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.073 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.071 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
```

```
rtt min/avg/max/mdev = 0.071/0.072/0.073/0.001 ms
```

Namespace의 Network 연결 4 – 다른 새 네임스페이스 간의 통신 연결

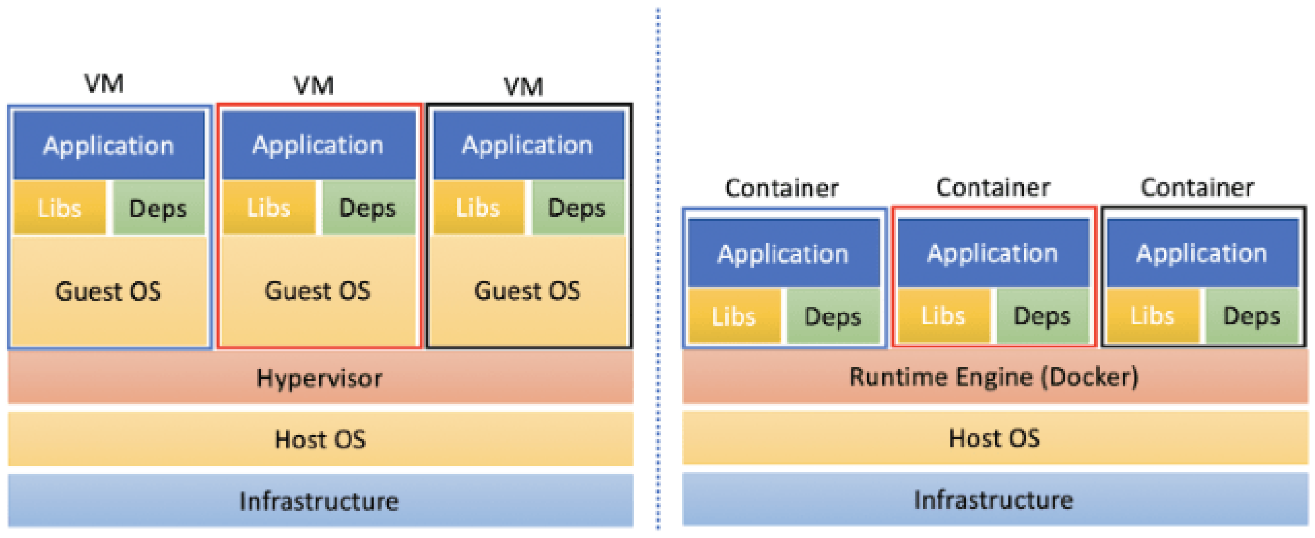
```
test namespace 브릿지에 추가되는 다른 네임스페이스와는 통신이 잘 되어야 합니다.
네트워크가 같다면 바로 되고, 다르다면 라우팅 설정을 추가로 해 주어야겠지요.

# test2 namespace 를 추가로 만들고 beth0/beth1 인터페이스를 생성 및 할당하고 브릿지에 연결합니다
# IP 세팅하고 test <---> test2 네임스페이스 간 Ping 시도시 잘 통신 됩니다.
ip netns add test2
ip link add beth0 type veth peer name beth1
ip link set beth0 netns test2
ip link set beth1 master br0
ip netns exec test2 ip addr add 10.10.10.3/24 dev beth0
ip netns exec test2 ip link set beth0 up
ip netns exec test2 ip link set dev lo up
ip link set beth1 up

# test2 namespace 확인
$ ip netns
test2 (id: 1)
test (id: 0)

$ ip -br -c addr
lo                UNKNOWN        127.0.0.1/8
ens33             UP             211.239.150.48/23
ens36             UP             192.168.0.2/24
veth1@if5         UP
br0               UP
beth1@if8         UP

$ brctl show
bridge name     bridge id               STP enabled
interfaces
br0             8000.2e0e64ccb0e5       no              beth1
```

```
                                                          veth1
# test namespace 의 veth0(10.10.10.2) 에 Ping 날려보기
ip netns exec test2 ping 10.10.10.2 -c 2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.112 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.076 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.076/0.094/0.112/0.018 ms
```

트러블슈팅

```
# 다른 네임스페이스와 통신이 안 된다면?
HOST 에 생성된 브릿지 인터페이스는 lo ( 내부통신 ) 이 아니기 때문에, Host 를 거쳐
다른 네트워크에 중계해 주려면 커널의 외부 포워딩 규칙을 따르기 때문입니다.
즉 NAT 구성 때와 비슷합니다. 커널 옵션에서 정의해 주어야 합니다.
ip4 트래픽에 대해 FORWARD 설정을 HOST 의 커널 옵션에 지정하여야 합니다.

# 먼저 HOST iptables FORWARD 정책을 확인하고 ACCEPT 로 변경합니다.
$ iptables -nL | grep -i forward
Chain FORWARD (policy DROP)

# 변경 후 체크
$ iptables --policy FORWARD ACCEPT
$ iptables -nL | grep -i forward
Chain FORWARD (policy ACCEPT)

$ service iptables save ( 또는 저장할 수 있는 OS별 옵션을 기입합니다 )

# 커널 옵션에서 ip4v.forward 활성화 합니다.
echo 1 > /proc/sys/net/ipv4/ip_forward
sysctl --system

# check
# 다시 통신 테스트를 해 봅니다.
```

[토막] VM과 Container 의 구분



Container

- 컨테이너는 리눅스 자체의 기술(LXC)이나 LXC를 활용하는 런타임 (예 : Docker) 등으로 HOST의 커널을 공유하여 격리 공간을 생성하는 하나의 프로세스 입니다.
- 컨테이너에서 실행되는 하위 프로세스의 구분를 위해 Namespace 와 리소스 제한을 위한 cgroup 이라는 리눅스 커널 기능을 사용합니다.

#아래는 namespace 로 Host 에서 별도로 분리 가능한 자원들입니다.
HOST의 Linux 내 자원을 가상화합니다. VM 이 하드웨어 디바이스를 가상화하는 것과 구분되는 부분입니다.

pid
user
uts
ipc
mnt
net

#cgroup 으로 격리된 프로세스에 대한 Host의 컴퓨팅 리소스를 제한할 수 있습니다.

Memory
CPU
Network
Device
I/O

- 컨테이너는 해당 Host의 프로세스이기 때문에, 리눅스 커널의 컨테이너는 Windows OS 컨테이너를 만들기 어렵습니다.
- Container 는 해당 Host의 커널에서 컨트롤이 가능하기 때문에 보안에 많이 신경써야 합니다.

VM

- VM 은 Host에 올린 Hypervisor 로 가상화 된 별개의 하드웨어를 만든 후
그 위에 OS 를 올림으로써 컨테이너보다 더 높은 격리 환경이라고 볼 수 있습니다.
- Host의 커널이 아닌, 가상화 된 하드웨어 구성 위에 여러 Linux/Windows/Other Guest OS 를 설치하고
해당 OS의 커널에서 동작합니다.

[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 거치
도록 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
```