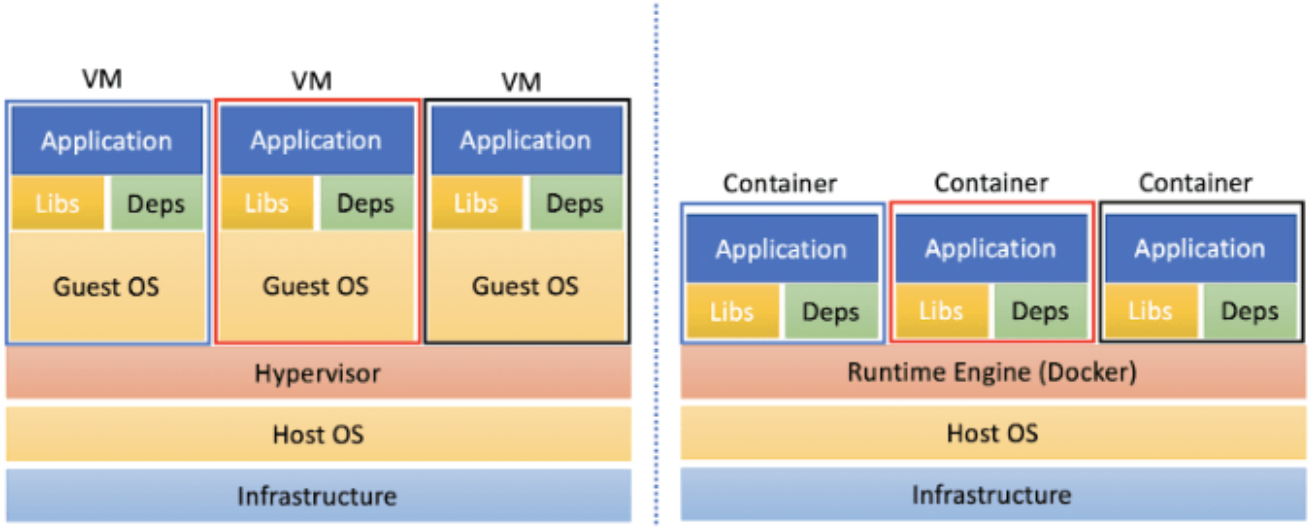
[토막] VM과 Container 의 구분



Container

- 컨테이너는 리눅스 자체의 기술(LXC)이나 LXC를 활용하는 런타임 (예 : Docker) 등으로 HOST의 커널을 공유하여 격리 공간을 생성하는 하나의 프로세스입니다.
- 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 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 않는
다(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:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b2c598d3f7
```

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:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b2c598d3f7

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
```