

# Cluster A & R1 매뉴얼

날짜

@2024년 3월 16일

## 1. ClusterA 네트워크 설정

- Oracle VM VirtualBox에서 Controller-A 머신의 네트워크 설정
  - 어댑터1 : NAT
  - 어댑터2 : 호스트 전용 어댑터 #1 (192.168.10.10/24)
  - 사용 이미지 : Ubuntu 20.04
  - 메모리 : 6144MB(6GB)
  - CPU : 4개
- Oracle VM VirtualBox에서 Worker-A1 머신의 네트워크 설정
  - 어댑터1 : NAT
  - 어댑터2 : 호스트 전용 어댑터 #1 (192.168.10.20/24)
  - 사용 이미지 : Ubuntu 22.04
  - 메모리 : 6144MB(6GB)
  - CPU : 2개
- Oracle VM VirtualBox에서 Worker-A2 머신의 네트워크 설정
  - 어댑터1 : NAT
  - 어댑터2 : 호스트 전용 어댑터 #1 (192.168.10.30/24)
  - 사용 이미지 : Ubuntu 22.04
  - 메모리 : 6144MB(6GB)
  - CPU : 2개
- 우분투 리눅스를 설치한 후 클러스터 구성을 시작하기 전 keyrings를 생성합니다.

```
sudo install -m 0755 -d /etc/apt/keyrings
```

## 2. Controller-A, Worker-A1, Worker-A2 공통 매뉴얼

- 해당 매뉴얼에서는 Controller-A 머신을 먼저 생성한 후 Controller-A 머신을 추가적으로 2개 복제하여 Worker-A1 머신과 Worker-A2 머신을 생성합니다.
- 업데이트 및 재부팅합니다.

```
sudo apt-get update && sudo apt-get -y full-upgrade  
sudo reboot -f
```

- 쿠버네티스를 설치하기 전 필요한 패키지를 설치합니다.

```
sudo apt-get -y install curl gnupg2 software-properties-common \
apt-transport-https ca-certificates
```

- 도커 레포지토리를 등록합니다.

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/ap
t/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/dock
er.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- /etc/apt/sources.list.d/docker.list 파일을 확인했을 때 아래와 같은 형식의 내용이 출력 되어야 합니다.

```
cat /etc/apt/sources.list.d/docker.list

# deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://dow
nload.docker.com/linux/ubuntu jammy stable
```

- 쿠버네티스 레포지토리를 등록합니다.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.27/deb/Release.key | sud
o gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] http
s://pkgs.k8s.io/core:/stable:/v1.27/deb/ /' | sudo tee /etc/apt/sources.
list.d/kubernetes.list
```

- 레포지토리를 업데이트합니다.

```
sudo apt update -y
```

- 컨테이너 런타임을 설치합니다.

```
sudo apt install -y containerd.io
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml >
/dev/null
```

- 쿠버네티스를 설치합니다.

```
sudo apt -y install kubelet kubeadm kubectl
```

```
sudo apt-mark hold kubelet kubeadm kubectl
```

- 스왑 기능을 비활성화합니다.

```
sudo sed -i '/swap/s/^/#/' /etc/fstab
sudo swapoff -a && sudo mount -a
```

- 커널 기능 설정을 변경합니다.

```
sudo su - -c "echo 'net.bridge.bridge-nf-call-ip6tables = 1' \
>> /etc/sysctl.d/kubernetes.conf"

sudo su - -c "echo 'net.bridge.bridge-nf-call-iptables = 1' \
>> /etc/sysctl.d/kubernetes.conf"

sudo su - -c "echo 'net.ipv4.ip_forward = 1' \
>> /etc/sysctl.d/kubernetes.conf"
```

- 커널 모듈 로드를 설정합니다.

```
sudo su - -c "echo 'overlay' >> /etc/modules-load.d/containerd.conf"

sudo su - -c "echo 'br_netfilter' >> /etc/modules-load.d/containerd.conf"
```

- 커널 모듈 및 설정을 활성화합니다.

```
sudo modprobe overlay

sudo modprobe br_netfilter

sudo sysctl --system
```

- kubelet 서비스를 활성화합니다.

```
sudo systemctl restart containerd kubelet

sudo systemctl enable containerd kubelet
```

- Controller-A, Worker-A1과 Worker-A2 머신의 공통 구축 단계가 완료되었습니다. 이제 Worker-A1 머신과 Worker-A2 머신을 생성하기 위해 Controller-A 머신을 복제하여 머신 2개를 추가적으로 생성합니다.

### 3. Controller-A 머신의 컨트롤러 설정

- 머신의 이름을 변경합니다.

```
sudo hostnamectl set-hostname Controller-A
```

- /etc/hosts 파일을 편집하여 변경한 호스트 이름을 아래와 같이 저장합니다.

```
Controller-A
```

- 머신을 재시작합니다.

```
sudo reboot
```

- /etc/hosts 파일을 편집하여 로컬 dns를 설정합니다.

해당 파일에 아래의 내용을 추가합니다.

```
192.168.10.10 Controller-A
192.168.10.20 Worker-A1
192.168.10.30 Worker-A2
```

- /etc/netplan/00-installer-config.yaml 파일을 다음과 같이 수정합니다.

```
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [192.168.22.253/24,192.168.127.253/24,192.168.129.253/24]
      routes:
        - to: 192.168.22.0/24
          via: 192.168.22.254
        - to: 192.168.20.0/24
          via: 192.168.129.254
        - to: 192.168.30.0/24
          via: 192.168.129.254
    enp0s8:
      dhcp4: false
      addresses: [192.168.10.253/24]
      routes:
        - to: 192.168.10.0/24
          via: 192.168.10.10
```

- 컨트롤러 노드에 쿠버네티스 관련 서비스 이미지를 설치합니다.


```
sudo kubeadm config images pull
```

- kubeadm으로 부트스트랩을 진행합니다.

```
sudo kubeadm init --apiserver-advertise-address [컨트롤러ip주소] \
-pod-network-cidr 172.30.0.0/16 \
-control-plane-endpoint [컨트롤러 호스트 이름]
```

- 아래와 같은 형식의 출력되는 문구를 확인해 메모장에 복사해 놓습니다.

```
# kubeadm join kube-controller:6443 --token ibtnhs.td7m6wqy91ttzuzu \
# --discovery-token-ca-cert-hash sha256:9acac5e9673a80afaae341aa14a9f
dd65939f2140db8d04aae399fa248fb3b40
```

-  만약 위 단계에서 에러가 발생할 경우 아래의 명령어를 입력해 보십시오. 이후 위 명령어를 다시 입력해 보십시오.

```
sudo systemctl start kubelet

sudo kubeadm reset

sudo rm -rf /etc/cni/net.d

sudo rm -rf .kube
```

- 부트스트랩 완료 후 현재 계정으로 kubectl 명령을 사용할 수 있도록 설정합니다.

```
mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- 여기까지 성공적으로 완료되었는지 확인하기 위해 아래의 명령어를 시도합니다.

```
kubectl get nodes
```

- 아래와 같은 결과가 출력되면 성공적으로 진행된 것입니다.

```
# NAME STATUS ROLES AGE VERSION
# kube-controller NotReady control-plane 2m20s v1.27.11
```

- 쿠버네티스의 컨테이너 네트워크 인터페이스 설치하고 설치 파일 다운로드합니다.

```
kubectl create -f \
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/\
manifests/tigera-operator.yaml

wget https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/\
manifests/custom-resources.yaml
```

- 네트워크 설정 변경 후 설치합니다.

```
sed -i 's/cidr: 192\.\.168\.\.0\.\.0\./16/cidr: 172\.\.30\.\.0\.\.0\./16/g' \
custom-resources.yaml
```

```
sed -i '7a registry: quay.io/' custom-resources.yaml

kubectl create -f custom-resources.yaml
```

- 최종적으로 노드의 상태를 확인합니다.

```
kubectl get nodes
```

- 다음과 같이 출력되어야 합니다.

NAME	STATUS	ROLES	AGE	VERSION
project-master	NotReady	control-plane	7d4h	v1.27.11

## 4. Worker-A1, Worker-A2 머신의 워커 설정

- 머신의 이름을 변경합니다.

```
sudo hostnamectl set-hostname Worker-A1

# 각각의 워커 머신의 호스트 이름을 바꾸어 설정합니다
```

- /etc/hosts 파일을 편집하여 변경한 호스트 이름을 아래와 같이 저장합니다.

```
Worker-A1

# 각각의 워커 머신의 호스트 이름을 바꾸어 설정합니다
```

- 머신을 재시작합니다.

```
sudo reboot
```

- /etc/hosts 파일을 편집하여 로컬 dns를 설정합니다.

해당 파일에 아래의 내용을 추가합니다.

```
192.168.10.10 Controller-A
192.168.10.20 Worker-A1
192.168.10.30 Worker-A2
```

- Worker-A1 머신의 /etc/netplan/00-installer-config.yaml 파일을 다음과 같이 수정합니다.

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false
```

```

addresses: [192.168.10.20/24]
routes:
  - to: 192.168.22.0/24
    via: 192.168.10.253
  - to: 192.168.20.0/24
    via: 192.168.10.253
  - to: 192.168.30.0/24
    via: 192.168.10.253
  - to: 192.168.127.0/24
    via: 192.168.10.253
version: 2

```

- Worker-A2 머신의 /etc/netplan/00-installer-config.yaml 파일을 다음과 같이 수정합니다.

```

# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false
      addresses: [192.168.10.30/24]
      routes:
        - to: 192.168.20.0/24
          via: 192.168.10.253
        - to: 192.168.127.0/24
          via: 192.168.10.253
version: 2

```

- 이전 단계에서 복사한 kubeadm join문을 sudo 명령어와 함께 Worker-A1 머신과 Worker-A2 머신에 입력합니다.

[예시]

```

# kubeadm join kube-controller:6443 --token ibtnhs.td7m6wqy91ttzuzu \
# --discovery-token-ca-cert-hash sha256:9acac5e9673a80afaae341aa14a9fdd6
5939f2140db8d04aae399fa248fb3b40

```

- 다시 Controller-A 머신에서 노드를 확인해 봅니다.

```
kubectl get nodes
```

- Kubernetes 클러스터를 초기화합니다.

```
kubeadm init
```

📌 아래와 같이 Controller-A의 상태가 READY로 변한 것을 확인할 수 있습니다.

## 5. Controller-A에서의 템플릿 파일 작성

- service.yaml 파일을 생성하여 아래와 같이 작성합니다.

```
touch service.yaml
```

```
# service.yaml 파일
apiVersion: v1
kind: Service
metadata:
  name: dynamicweb
spec:
  type: ClusterIP
  selector:
    name: dynamicweb
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: web-pod
spec:
  type: NodePort
  selector:
    name: web-pod
  ports:
    - protocol: TCP
      targetPort: 443
      port: 443
```

- was.yaml 파일을 생성하여 아래와 같이 작성합니다.

```
touch was.yaml
```

```
# was.yaml 파일
apiVersion: v1
kind: Pod
metadata:
  name: dynamicweb
  labels:
    name: dynamicweb
```



```
spec:
  containers:
    - name: was
      image: kimjuhan/project:db_WASserver1
```

- web.yaml 파일을 생성하여 아래와 같이 작성합니다.

```
touch web.yaml
```

```
# web.yaml 파일
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
  labels:
    name: web-pod
spec:
  containers:
    - name: web
      image: kimjuhan/docker-3-tier-architecture:web
```

## 1. R1 네트워크 설정

- 사용 이미지 : Ubuntu 22.04
- Oracle VM VirtualBox에서 R1 머신의 네트워크 설정
  - 어댑터1: 어댑터에 브리지(Intel(R) Wi-Fi 6)
  - 어댑터2: 호스트 전용 어댑터 #1 (192.168.10.253)
  - 사용 이미지 : Ubuntu 20.04
  - 메모리 : 1024MB(1GB)
  - CPU : X
- R1 머신의 /etc/netplan/00-installer-config.yaml 파일을 다음과 같이 수정합니다.

```
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [192.168.22.253/24, 192.168.129.253/24]
      routes:
        - to: 192.168.22.0/24
          via: 192.168.22.254
        - to: 192.168.20.0/24
          via: 192.168.129.254
```

```
        - to: 192.168.30.0/24
          via: 192.168.129.254
enp0s8:
  dhcp4: false
  addresses: [192.168.10.253/24]
  routes:
    - to: 192.168.10.0/24
      via: 192.168.10.10
version: 2
```

- /etc/sysctl.conf 파일의 내용 중 `net.ipv4.ip_forward=1` 문구를 주석 해제하여 활성화 합니다.
- 성공적으로 수정되었는지 확인합니다.

```
sudo sysctl --system
```