1️⃣ EC2 인스턴스 준비

인스턴스 종류

Ubuntu Server 24.04 LTS

최소 사양: t3.medium 이상 권장 (2 vCPU, 4GB RAM)

보안 그룹 설정

Kubernetes 통신을 위해 아래 포트 허용 필요

포트 용도 6443 API Server 2379-2380 etcd 10250 Kubelet 10251 kube-scheduler 10252 kube-controller-manager 30000-32767 NodePort 서비스

ICMP 허용 (Ping용)

호스트네임 설정

sudo hostnamectl set-hostname master # master 노드 sudo hostnamectl set-hostname worker1 # worker 노드 sudo hostnamectl set-hostname worker2 # worker 노드

호스트 파일 수정 /etc/hosts에 내부 IP와 호스트명 추가

10.0.0.10 master 10.0.0.11 worker1 10.0.0.12 worker2

Ubuntu 24.04 환경에서 Kubernetes 클러스터를 구축하기 위해서는 기존의 `apt-key` 방식이 더 이상 지원되지 않으므로, 새로운 GPG 키 관리 방법을 사용해야 한다.

EC2 3대(Ubuntu 24.04) 환경에서 Kubernetes 클러스터를 구축하는 명령어 모음

🛠️ 1단계: 공통 초기 설정 (모든 노드에서 실행)

```
# 시스템 업데이트 및 필수 패키지 설치
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl wget vim apt-transport-https ca-certificates software-properties-common gnupg lsb-release

# swap 비활성화
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab

# br_netfilter 활성화
sudo modprobe br_netfilter
echo '1' | sudo tee /proc/sys/net/bridge/bridge-nf-call-iptables

# containerd 설치 및 설정
sudo apt install -y containerd
```

```
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i 's/^\s*SystemdCgroup = false/SystemdCgroup = true/'
/etc/containerd/config.toml
sudo systemctl restart containerd
sudo systemctl enable containerd

# Kubernetes GPG 키 다운로드 및 저장
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --
dearmor -o /etc/apt/keyrings/k8s.gpg

# Kubernetes 저장소 추가
echo "deb [signed-by=/etc/apt/keyrings/k8s.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee
/etc/apt/sources.list.d/k8s.list

# apt 업데이트 및 Kubernetes 패키지 설치
sudo apt update
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

지금 EC2는 Ubuntu 24.04 기반이고, containerd 설치까지 완료된 상태. 현재 상황에서 Kubernetes 클러스터를 구축하려면 `kubeadm`을 이용해서 마스터 노드를 초기화하고 워커 노드를 연결하는 순서로 진행해야 한다. containerd를 CRI로 사용하도록 설정했으므로, kubeadm 설정을 containerd에 맞게 잡아줘야 한다.

---

1 필수 커널 모듈 및 sysctl 설정

```
# 브릿지 네트워크 필터
sudo modprobe br_netfilter
echo '1' | sudo tee /proc/sys/net/bridge/bridge-nf-call-iptables

# sysctl 적용
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

sudo sysctl --system
```

---

2 kubeadm 구성 파일 만들기

containerd를 사용하도록 kubeadm config 파일 작성:

```
sudo mkdir -p /etc/kubernetes
cat <<EOF | sudo tee /etc/kubernetes/kubeadm-config.yaml
apiVersion: kubeadm.k8s.io/v1beta3
```

```
kind: ClusterConfiguration
kubernetesVersion: v1.30.0
networking:
  podSubnet: "10.244.0.0/16"
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
nodeRegistration:
  criSocket: /run/containerd/containerd.sock
EOF
```

---

3 마스터 노드 초기화

```
sudo kubeadm init --config=/etc/kubernetes/kubeadm-config.yaml --upload-certs
```

- 초기화 후 `kubeadm`이 보여주는 `kubeadm join` 커맨드가 나중에 워커 노드 연결에 필요함.
- 초기화가 끝나면 현재 사용자에 kubeconfig 복사:

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

---

4 Pod 네트워크 설치 (예: Flannel)

```
kubectl apply -f https://raw.githubusercontent.com/flannel-
io/flannel/v0.26.3/Documentation/kube-flannel.yml
```

- 다른 네트워크 플러그인(Calico 등)도 가능. PodSubnet은 kubeadm-config.yaml과 동일해야 함.

---

5 워커 노드 연결

워커 노드에서:

```
sudo kubeadm join <MASTER_IP>:6443 --token <TOKEN> --discovery-token-ca-cert-hash
sha256:<HASH>
```

- `<MASTER_IP>`와 토큰/해시 값은 마스터 초기화 후 kubeadm에서 출력됨.

---

6 상태 확인

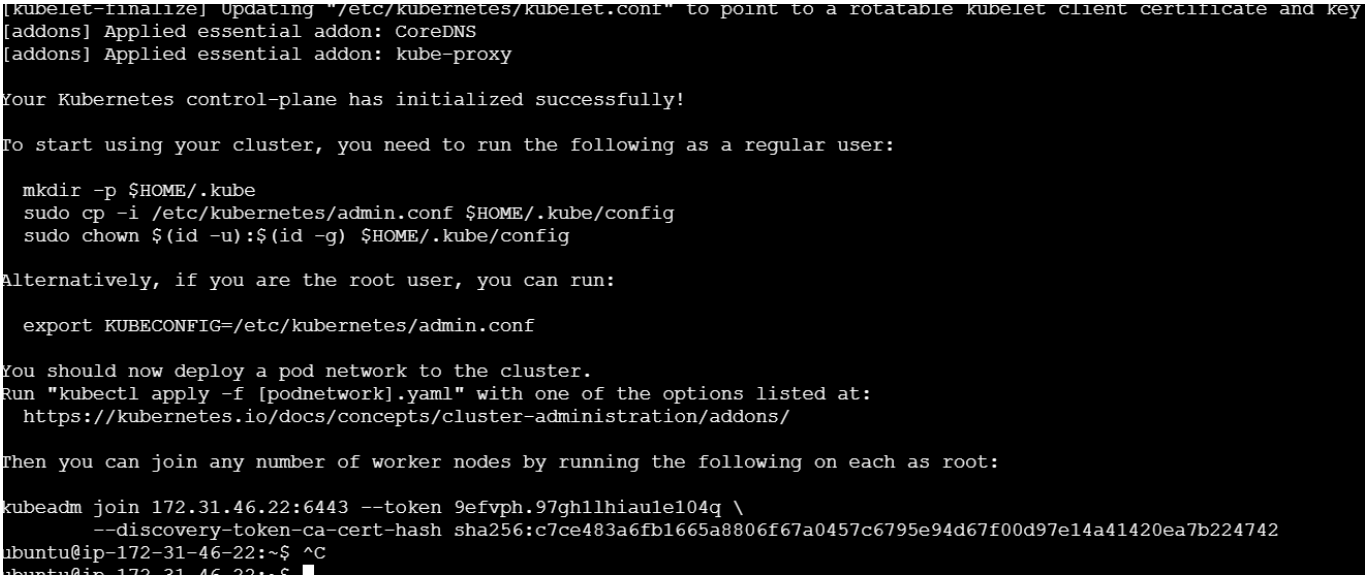
마스터에서:

```
kubectl get nodes
kubectl get pods -A
```

⚠ 주의 사항

- EC2에서 보안 그룹에 6443, 10250, 2379~2380, 8285, 8472 포트를 열어야 함.
- Ubuntu에서 재부팅 시 containerd, kubelet 서비스 자동 시작 확인 필요:

```
sudo systemctl enable containerd kubelet
sudo systemctl start kubelet
```



sudo kubeadm join 172.31.46.22:6443 --token 9efvph.97gh1lhiau1e104q --discovery-token-ca-cert-hash sha256:c7ce483a6fb1665a8806f67a0457c6795e94d67f00d97e14a41420ea7b224742

```
[ERROR FileContent--proc-sys-net-ipv4-ip_forward]: /proc/sys/net/ipv4/ip_forward contents are not set to 1
```

즉, **IPv4 포워딩(ip_forward)** 이 활성화되어 있지 않다는 뜻. Kubernetes는 Pod 네트워크 통신을 위해 **IP 포워딩**이 반드시 켜져 있어야 한다.

해결 방법

1. **일시적으로 켜기**

```
sudo sysctl -w net.ipv4.ip_forward=1
```

> 이 방법은 재부팅하면 초기화됩니다.

2. **영구적으로 켜기**

```
sudo nano /etc/sysctl.conf
```

파일에 다음 라인 추가 또는 수정:

```
net.ipv4.ip_forward=1
```

변경 적용:

```
sudo sysctl -p
```

3. **그 다음 다시 join 실행**

```
sudo kubeadm join 172.31.46.22:6443 --token 9efvph.97gh1lhiau1e104q \
        --discovery-token-ca-cert-hash
sha256:c7ce483a6fb1665a8806f67a0457c6795e94d67f00d97e14a41420ea7b224742
```

TIP: 만약 다른 preflight 체크 때문에 계속 막히면, **강제로 무시**할 수도 있습니다.

```
sudo kubeadm join ... --ignore-preflight-errors=all
```

하지만 권장하지 않고, 필요한 설정을 맞춘 후 실행하는 게 안정적입니다.

Master Node 환경변수 설정 추가

kubeadm init은 성공했는데, `kubectl` 명령이 제대로 동작하지 않는 문제 발생 시 원인은 **kubectl이 사용할 kubeconfig를 아직 설정하지 않았거나 올바른 위치에 없어서 API 서버에 연결할 수 없음**

문제 분석

- `kubeadm init` 출력:

```
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
```

- 이후 `kubectl get node` 실행했는데:

```
The connection to the server localhost:8080 was refused - did you specify the
right host or port?
```

즉, kubectl이 **`/home/ubuntu/.kube/config`**를 읽지 못하고 **기본 localhost:8080**을 시도하고 있는 상황 발생

해결 방법

1. **kubectl이 사용할 kubeconfig 복사**

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

2. **환경 변수로 KUBECONFIG 지정** (선택 사항)

```
export KUBECONFIG=$HOME/.kube/config
```

- 이렇게 하면 해당 세션에서 kubectl이 올바른 kubeconfig를 사용한다.

3. **kubectl 상태 확인**

```
kubectl get nodes
```

- 이제 control-plane 노드가 `Ready` 상태로 나와야 한다.

TIP:

- 만약 root 계정에서 그대로 사용한다면:

```
export KUBECONFIG=/etc/kubernetes/admin.conf
kubectl get nodes
```

- **Pod 네트워크(CNI)**가 아직 설치되지 않았으므로, Flannel이나 Calico를 적용해야 다른 노드들이 Join 가능하다.

---

1 CNI 설치는 **마스터 노드에서만 명령 실행**

- **왜 마스터 노드에서만?**
  - kubeadm으로 클러스터를 구성하면, **Control Plane이 cluster-wide API를 관리**한다.
  - CNI YAML(manifest)을 `kubectl apply -f ...`로 설치하면, **API 서버가 모든 Node에 DaemonSet을 배포**한다.
  - 즉, **Worker Node들에 별도로 설치할 필요 없음** → 마스터에서 한 번만 실행하면 됨.
- **실행 원리**
  1. Flannel YAML에는 DaemonSet이 포함돼 있다.
  2. DaemonSet이 각 Node에 Pod를 자동 생성한다.
  3. kubelet이 CNI plugin을 초기화하면서 Node 상태가 `Ready`로 변한다.

---

2 마스터 노드에서 실행할 명령어

```
# 1. Flannel CNI 설치
kubectl apply -f https://raw.githubusercontent.com/flannel-
io/flannel/master/Documentation/kube-flannel.yml

# 2. 설치 상태 확인
kubectl get pods -n kube-system

# 3. Node 상태 확인
kubectl get nodes
```

- `coredns` Pod가 Running으로 바뀌고
- 모든 Node가 `Ready` 상태로 변하며
- 앱 Pod도 Pending에서 Running으로 전환된다.

---

3 멀티 Node 주의 사항

- 마스터 노드의 `Taints` 때문에 일반 앱 Pod는 스케줄링 안 됨:

```
Taints: node-role.kubernetes.io/control-plane:NoSchedule
```

- 앱을 마스터 노드에서 실행하고 싶으면 **Pod에 toleration** 추가 필요.
- Worker Node에는 taint 없으므로 바로 스케줄링 가능.

결론:

- **마스터 노드에서만 Flannel 설치**하면 모든 Node가 자동으로 네트워크 플러그인 적용됨
- Node Ready → CoreDNS Running → 앱 Pod Scheduling 순서로 정상화

- NodePort 설정

kubectl patch svc gift-app-svc -p '{"spec":{"type":"NodePort"}}'