PIOLINK

K8S Master Node HA External ETCD Clustering 구성 가이드

파이오링크 | 인프라기술팀
방효성

External ETCD 클러스터

External ETCD 클러스터링

클러스터링 준비

- ETCD 는 Raft 알고리즘을 기반으로 동기화한다. ETCD 는 일관성이 특징인 KEY:VALUE 포맷으로 데이터를 저장하는 DB이다.
- 다수결 방식으로 Leader 선출 및 DB 동기화를 하므로 Quorum계산 2(3/2+1)을 만족하기 위해서 Clustering 노드 3개 이상 구성되어야 한다.
- ETCD 는 TCP 2379/2380포트를 사용한다. 2379는 Client API, 2380 은 외부 노드와 데이터 동기화 및 선거를 위해 사용, 주로 2380 을 사용하는 것으로 보인다.
- ETCD 클러스터링 구성 사전 준비

Node 3개 노드 운영체제 : Rocky OS 9
Golang 2.3.0 ~ 설치

Golang 설치

- Etcd 는 Go 언어로 만들어져서 dnf 와 같은 명령어로 etcd 패키지를 통해 설치하지 않고 수동으로 설치한다면 사전에 go-lang을 설치해야 한다.
- 수동설치 방식은 https://go.dev/dl/ 접속해서 버전 및 파일명 확인 후 go1.23.0 ~ 이상 버전을 선택해서 설치한다.

Stable versions

go1.24.1

File name	Kind	OS	Arch	Size	SHA256 Checksum
go1.24.1.src.tar.gz	**Source**			**29MB**	8244ebf46e65607db10222b5806aeb31e1fef8979e1b6b12f60e677e9a3e0656
go1.24.1.darwin-amd64.tar.gz	Archive	macOS	x86-64	76MB	addbfce2056744962e2d7436613ab96486660ef7a2e066d171b9d6f2da7e7abe
go1.24.1.darwin-amd64.pkg	**Installer**	**macOS**	**x86-64**	**77MB**	58d529334561eff11087ed4ab18fe0b46d8d5aad88f45e02b9645f847e014512
go1.24.1.darwin-arm64.tar.gz	Archive	macOS	ARM64	73MB	295581b5619ace92f5106e5bcb05e51869837eb19742fdfa6e8346e18e78ff88
go1.24.1.darwin-arm64.pkg	**Installer**	**macOS**	**ARM64**	**73MB**	78b0fe8ddde344eb499f1a952e687eb84ebd28be2b739efa0d4eb042f07e44e82
go1.24.1.linux-386.tar.gz	Archive	Linux	x86	73MB	8e580ecedbe17e42ee10177bea07ece96a3e77e792ea1ea72178a9675d16ffa5
go1.24.1.linux-amd64.tar.gz	**Archive**	**Linux**	**x86-64**	**75MB**	eb2396bae64183edeef81a9a6df0aea3bee9511fe21469fb89a0e00470088073

① wget https://go.dev/dl/go1.23.0.linux-amd64.tar.gz
② rm -rf /usr/local/go
③ tar -C /usr/local -xzf go1.23.0.linux-amd64.tar.gz
④ echo 'export PATH=$PATH:/usr/local/go/bin' >> /etc/profile
⑤ source /etc/profile

Etcd 설치

- https://etcd.io/docs/v3.5/install/ 에서 기재되어 있는 설치 방식 중 마음에 드는 방식으로 etcd 를 설치한다.
- 필자는 Build from source 방식으로 설치했다.

Build from source

If you have Go version 1.2+, you can build etcd from source by following these steps:

1. Download the etcd repo as a zip file and unzip it, or clone the repo using the following command.

```
$ git clone -b v3.5.19 https://github.com/etcd-io/etcd.git
```

To build from `main@HEAD`, omit the `-b v3.5.19` flag.

2. Change directory:

```
$ cd etcd
```

3. Run the build script:

```
$ ./build.sh
```

- ./build.sh 스크립트 실행 후 etcd, etcdctl 명령어는 터미널 접속 후 전체경로에서 사용할 수 있도록 /usr/local/bin 디렉터리에 옮겨준다.

```
mv bin/etcd bin/etcdctl /usr/local/bin/
```

Etcd 클러스터링 가이드

- https://etcd.io/docs/v3.5/op-guide/clustering/ 에서 마음에 드는 방식을 선택해서 클러스터링한다.
- 필자는 static 방식에서 별도 TLS,mTLS 를 위한 인증서 설정없이 진행했다.

Static

As we know the cluster members, their addresses and the size of the cluster before starting, we can use an offline bootstrap configuration by setting the `initial-cluster` flag. Each machine will get either the following environment variables or command line:

```
ETCD_INITIAL_CLUSTER="infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380"
ETCD_INITIAL_CLUSTER_STATE=new
```

```
--initial-cluster infra0=http://10.0.1.10:2380,infra1=http://10.0.1.11:2380,infra2=http://10.0.1.12:2380 \
--initial-cluster-state new
```

Note that the URLs specified in `initial-cluster` are the *advertised peer URLs*, i.e. they should match the value of `initial-advertise-peer-urls` on the respective nodes.

If spinning up multiple clusters (or creating and destroying a single cluster) with same configuration for testing purpose, it is highly recommended that each cluster is given a unique `initial-cluster-token`. By doing this, etcd can generate unique cluster IDs and member IDs for the clusters even if they otherwise have the exact same configuration. This can protect etcd from cross-cluster-interaction, which might corrupt the clusters.

etcd listens on `listen-client-urls` to accept client traffic. etcd member advertises the URLs specified in `advertise-client-urls` to other members, proxies, clients. Please make sure the `advertise-client-urls` are reachable from intended clients. A common mistake is setting `advertise-client-urls` to localhost or leave it as default if the remote clients should reach etcd.

On each machine, start etcd with these flags:

Etcd 환경 설정 파일 생성

- **/etc/etcd.env 환경 파일 작성한다.**
- **참고) 기존 클러스터링이 존재하고 새로운 노드가 추가되는 경우 해당 노드의 환경변수 파일에서 CLUSTER_STATE부분을 “existing” 으로 작성해야 한다.**

 # Etcd-01

```
ETCD_NAME="etcd-01"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=http://11.11.20.10:2380,etcd-02=http://11.11.20.20:2380,etcd-03=http://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://11.11.20.10:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://11.11.20.10:2379"
ETCD_LISTEN_PEER_URLS="http://11.11.20.10:2380"
ETCD_LISTEN_CLIENT_URLS="http://11.11.20.10:2379"
```

 # Etcd-02

```
ETCD_NAME="etcd-02"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=http://11.11.20.10:2380,etcd-02=http://11.11.20.20:2380,etcd-03=http://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://11.11.20.20:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://11.11.20.20:2379"
ETCD_LISTEN_PEER_URLS="http://11.11.20.20:2380"
ETCD_LISTEN_CLIENT_URLS="http://11.11.20.20:2379"
```

 # Etcd-03

```
ETCD_NAME="etcd-03"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=http://11.11.20.10:2380,etcd-02=http://11.11.20.20:2380,etcd-03=http://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://11.11.20.30:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://11.11.20.30:2379"
ETCD_LISTEN_PEER_URLS="http://11.11.20.30:2380"
ETCD_LISTEN_CLIENT_URLS="http://11.11.20.30:2379"
```

Etcd.service 추가

```
[Unit]
Description=BHS-ETCD
After=network.target

[Service]
Type=notify
User=root
Group=root
EnvironmentFile=-/etc/etcd.env
ExecStart=/usr/local/bin/etcd
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

- **각 노드에 /etc/system/system/etcd.service 파일에 위 내용을 파일로 저장, Redhat 계열은 해당 위치인데 우분투는 경로가 다르니 별도 확인 필요함, 필자는 Rocky 9 으로 진행**

방화벽 정책 설정

```
firewall-cmd --permanent --add-port=2379/tcp
firewall-cmd –permanent –add-port=2380/tcp
firewall-cmd –reload
```

- **또는 테스트 용도이니 방화벽 및 보안 설정없이 진행하는 것도 괜찮다.**

```
systemctl stop firewalld, systemctl disable firewalld, setenforce 0
```

데몬 재시작

```
systemctl daemon-reload
systemctl restart etcd
systemctl enable etcd
```

클러스터링 재 조인 경우

```
sudo systemctl stop etcd
sudo rm -rf /var/lib/etcd/*
```

- 위 명령어를 통해 db 내용 삭제 후 데몬을 재 시작해야 한다.
- 문제가 있을 경우 아래 명령어를 통해 로그 확인해본다.

```
journalctl -u etcd --no-pager --lines=50
```

클러스터링 상태 확인

- etcdctl endpoint status 명령어로 어떤 Node 가 Leader 선출되었는지 확인한다. (etcd-02 노드가 Leader 선출)

```
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 endpoint status --write-out=table --cluster
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT         |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| http://11.11.20.20:2379 | 3ce2621f82a3f303 |  3.5.19 |   20 kB |     false |      false |        11 |         41 |                 41 |        |
| http://11.11.20.10:2379 | ad7dde673e1b5f46 |  3.5.19 |   20 kB |      true |      false |        11 |         41 |                 41 |        |
| http://11.11.20.30:2379 | b82586587df8d36f |  3.5.19 |   20 kB |     false |      false |        11 |         41 |                 41 |        |
+-------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
```

- 위 테이블 필드 중 RAFT TERM 은 간단하게 Leader 선출 횟수라고 보면 되겠고 RAFT INDEX, RAFT APPLIED INDEX 는 DB 동기화 Commit ID 개념으로 보면 된다.
- 해당 인덱스 번호로 데이터 최신화 버전을 구분하고 노드가 추가되면 최신버전으로 동기화한다.

- Etcd는 Key : Value 포맷으로 데이터를 저장 및 관리한다.
- 데이터 저장 후 각 노드에서 동기화 성공 여부를 확인한다.

```
etcdctl --endpoints=http://11.11.20.20:2379 put INFRA_K8S "BHS"
etcdctl --endpoints=http://11.11.20.10:2379 get INFRA_K8S
etcdctl --endpoints=http://11.11.20.20:2379 get INFRA_K8S
etcdctl --endpoints=http://11.11.20.30:2379 get INFRA_K8S
```

- 3개 노드 각각 명령어 수행 시 INFRA_K8S Key의 값 조회 결과 BHS를 저장하고 있음을 확인

```
[root@etcd-01 etcd]# etcdctl --endpoints=https://11.11.20.20:2379  \
--cacert="/etc/kubernetes/pki/etcd/ca.pem" \
--cert="/etc/kubernetes/pki/etcd/apiserver-etcd-client.pem" \
--key="/etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem" \
put INFRA_K8S "BHS"
OK
[root@etcd-01 etcd]# etcdctl --endpoints=https://11.11.20.10:2379  \
--cacert="/etc/kubernetes/pki/etcd/ca.pem" \
--cert="/etc/kubernetes/pki/etcd/apiserver-etcd-client.pem" \
--key="/etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem" \
get INFRA_K8S
INFRA_K8S
BHS
[root@etcd-01 etcd]#
[root@etcd-01 etcd]# etcdctl --endpoints=https://11.11.20.30:2379  \
--cacert="/etc/kubernetes/pki/etcd/ca.pem" \
--cert="/etc/kubernetes/pki/etcd/apiserver-etcd-client.pem" \
--key="/etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem" \
get INFRA_K8S
INFRA_K8S
BHS
[root@etcd-01 etcd]#
[root@etcd-01 etcd]# etcdctl --endpoints=https://11.11.20.20:2379  \
--cacert="/etc/kubernetes/pki/etcd/ca.pem" \
--cert="/etc/kubernetes/pki/etcd/apiserver-etcd-client.pem" \
--key="/etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem" \
get INFRA_K8S
INFRA_K8S
BHS
[root@etcd-01 etcd]#
```

클러스터 상태 확인

- https://etcd.io/docs/v3.5/tutorials/how-to-check-cluster-status/ 해당 페이지에서 클러스터 상태 확인 관련 설명이 있다.

```
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 endpoint status --write-out=table
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| http://11.11.20.10:2379 | ad7dde673e1b5f46 |  3.5.19 |   20 kB |      true |      false |        11 |         42 |                 42 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 endpoint status --write-out=table --cluster
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| http://11.11.20.20:2379 | 3ce2621f82a3f303 |  3.5.19 |   20 kB |     false |      false |        11 |         42 |                 42 |        |
| http://11.11.20.10:2379 | ad7dde673e1b5f46 |  3.5.19 |   20 kB |      true |      false |        11 |         42 |                 42 |        |
| http://11.11.20.30:2379 | b82586587df8d36f |  3.5.19 |   20 kB |     false |      false |        11 |         42 |                 42 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 endpoint health --write-out=table --cluster
+--------------------------+--------+------------+-------+
|        ENDPOINT          | HEALTH |    TOOK    | ERROR |
+--------------------------+--------+------------+-------+
| http://11.11.20.10:2379 |   true | 3.177954ms |       |
| http://11.11.20.20:2379 |   true | 3.724365ms |       |
| http://11.11.20.30:2379 |   true | 4.582154ms |       |
+--------------------------+--------+------------+-------+
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 endpoint hashkv --write-out=table --cluster
+--------------------------+------------+
|        ENDPOINT          |    HASH    |
+--------------------------+------------+
| http://11.11.20.20:2379 | 2347150913 |
| http://11.11.20.10:2379 | 2347150913 |
| http://11.11.20.30:2379 | 2347150913 |
+--------------------------+------------+
```

- false 의미는 해당 node 가 learner 인지 여부를 파악하기 위한 필드라고 한다.
- RAFT 알고리즘으로 동작하는 ETCD 는 새로운 노드가 추가되었을 때 리더 재 선출 상황 또는 신규 노드에서의 misconfiguration 상황을 방지하기 위해서 신규 Node 상태가 Learner 인 경우 RAFT INDEX 동기화 완료 전 까지 투표권 없이 Leader 로 부터 데이터 수신은 하되 과반수를 의미하는 Quorum size를 증가시키지 않는다.
- Leader 는 learner 노드에 클라이언트 Read/Write 요청을 차단하고 추후 leader 가 learner 를 promote 해주면 그제서야 투표권과 Quorum size 가 증가한ㄷ,
- https://etcd.io/docs/v3.5/learning/design-learner/

```
[root@etcd-01 ~]# etcdctl --endpoints=http://11.11.20.10:2379 member list
3ce2621f82a3f303, started, etcd-02, http://11.11.20.20:2380, http://11.11.20.20:2379, false
ad7dde673e1b5f46, started, etcd-01, http://11.11.20.10:2380, http://11.11.20.10:2379, false
b82586587df8d36f, started, etcd-03, http://11.11.20.30:2380, http://11.11.20.30:2379, false
```

External ETCD 클러스터 (TLS, mTLS)

HTTPS ETCD 클러스터링

CFSSL 인증서 생성 툴

- **Cloudflare 에서 인증서 생성 툴을 만들었다. 이름은 cfssl 이며 openssl처럼 명령어 방식이 아닌 json 포맷으로 파일을 생성 후 인증서를 만들 수 있다.**
- **cfssl 설치**

CFSSL 설치 및 인증서 디렉터리 생성

```
curl -L -o /usr/local/bin/cfssl https://github.com/cloudflare/cfssl/releases/download/v1.6.4/cfssl_1.6.4_linux_amd64
curl -L -o /usr/local/bin/cfssljson https://github.com/cloudflare/cfssl/releases/download/v1.6.4/cfssljson_1.6.4_linux_amd64
chmod +x /usr/local/bin/cfssl /usr/local/bin/cfssljson

mkdir -p /etc/kubernetes/pki/etcd
cd /etc/kubernetes/pki/etcd
```

CA Profile(설정 정보) 생성

```
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "server": {
        "expiry": "87600h",
        "usages": ["signing", "key encipherment", "server auth"]
      },
      "client": {
        "expiry": "87600h",
        "usages": ["signing", "key encipherment", "client auth"]
      },
      "peer": {
        "expiry": "87600h",
        "usages": ["signing", "key encipherment", "server auth", "client auth"]
      }
    }
  }
}
EOF
```

CA인증서 및 키 생성 (ca-key.pem, ca.pem)

```
cat > ca-csr.json <<EOF
{
  "CN": "etcd-ca",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "KR",
      "L": "Seoul",
      "O": "BHS",
      "OU": "etcd"
    }
  ]
}`
EOF
cfssl gencert -initca ca-csr.json | cfssljson -bare ca
```

ETCD 서버 인증서 생성 (server.pem,server.pem)

```
cat > server-csr.json <<EOF
{
  "CN": "etcd-server",
  "hosts": [
    "127.0.0.1",
    "11.11.20.10",
    "11.11.20.20",
    "11.11.20.30",
    "etcd-01",
    "etcd-02",
    "etcd-03"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "KR",
      "L": "Seoul",
      "O": "BHS",
      "OU": "etcd"
    }
  ]
}
EOF

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server server-csr.json | cfssljson -bare server
```

ETCD peer 인증서 생성 (peer.pem, peer-key.pem) / Etcd 간 통신용 인증서

```
cat > peer-csr.json <<EOF
{
  "CN": "etcd-peer",
  "hosts": [
    "127.0.0.1",
    "11.11.20.10",
    "11.11.20.20",
    "11.11.20.30",
    "etcd-01",
    "etcd-02",
    "etcd-03"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "KR",
      "L": "Seoul",
      "O": "BHS",
      "OU": "etcd"
    }
  ]
}
EOF
```

Etcdctl 명령어 사용 또는 K8S API 통신 용 클라이언트 인증서 (apiserver-etcd-client-key.pem, apiserver-etcd-client.pem)

```
cat > apiserver-etcd-client-csr.json <<EOF
{
  "CN": "apiserver-etcd-client",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "KR",
      "L": "Seoul",
      "O": "BHS",
      "OU": "apiserver"
    }
  ]
}
EOF
```

ETCD peer 인증서 생성 명령어

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer peer-csr.json | cfssljson -bare peer
```

클라이언트 인증서 생성 명령어

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client apiserver-etcd-client-csr.json | cfssljson -bare apiserver-etcd-client
```

ETCD 노드들에게 인증서 배포

```
scp ca.pem server.pem server-key.pem peer.pem peer-key.pem bhs@11.11.20.10:/home/bhs
scp ca.pem server.pem server-key.pem peer.pem peer-key.pem bhs@11.11.20.20:/home/bhs
scp ca.pem server.pem server-key.pem peer.pem peer-key.pem bhs@11.11.20.30:/home/bhs
```

Etcd 환경변수 파일 수정

```
ETCD_NAME="etcd-01"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=https://11.11.20.10:2380,etcd-02=https://11.11.20.20:2380,etcd-
03=https://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://11.11.20.10:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://11.11.20.10:2379"
ETCD_LISTEN_PEER_URLS="https://11.11.20.10:2380"
ETCD_LISTEN_CLIENT_URLS="https://11.11.20.10:2379"

# TLS 인증서 설정 추가
ETCD_CERT_FILE="/etc/kubernetes/pki/etcd/server.pem"
ETCD_KEY_FILE="/etc/kubernetes/pki/etcd/server-key.pem"
ETCD_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"

# Peer 간 TLS 통신 설정
ETCD_PEER_CERT_FILE="/etc/kubernetes/pki/etcd/peer.pem"
ETCD_PEER_KEY_FILE="/etc/kubernetes/pki/etcd/peer-key.pem"
ETCD_PEER_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"
ETCD_PEER_CLIENT_CERT_AUTH="true"

# 클라이언트 인증 설정
ETCD_CLIENT_CERT_AUTH="true"
```

```
ETCD_NAME="etcd-02"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=https://11.11.20.10:2380,etcd-02=https://11.11.20.20:2380,etcd-
03=https://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://11.11.20.20:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://11.11.20.20:2379"
ETCD_LISTEN_PEER_URLS="https://11.11.20.20:2380"
ETCD_LISTEN_CLIENT_URLS="https://11.11.20.20:2379"

# TLS 인증서 설정 추가
ETCD_CERT_FILE="/etc/kubernetes/pki/etcd/server.pem"
ETCD_KEY_FILE="/etc/kubernetes/pki/etcd/server-key.pem"
ETCD_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"

# Peer 간 TLS 통신 설정
ETCD_PEER_CERT_FILE="/etc/kubernetes/pki/etcd/peer.pem"
ETCD_PEER_KEY_FILE="/etc/kubernetes/pki/etcd/peer-key.pem"
ETCD_PEER_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"
ETCD_PEER_CLIENT_CERT_AUTH="true"

# 클라이언트 인증 설정
ETCD_CLIENT_CERT_AUTH="true"
```

```
ETCD_NAME="etcd-03"
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="bhs"
ETCD_INITIAL_CLUSTER="etcd-01=https://11.11.20.10:2380,etcd-02=https://11.11.20.20:2380,etcd-
03=https://11.11.20.30:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://11.11.20.30:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://11.11.20.30:2379"
ETCD_LISTEN_PEER_URLS="https://11.11.20.30:2380"
ETCD_LISTEN_CLIENT_URLS="https://11.11.20.30:2379"
# TLS 인증서 설정 추가
ETCD_CERT_FILE="/etc/kubernetes/pki/etcd/server.pem"
ETCD_KEY_FILE="/etc/kubernetes/pki/etcd/server-key.pem"
ETCD_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"
# Peer 간 TLS 통신 설정
ETCD_PEER_CERT_FILE="/etc/kubernetes/pki/etcd/peer.pem"
ETCD_PEER_KEY_FILE="/etc/kubernetes/pki/etcd/peer-key.pem"
ETCD_PEER_TRUSTED_CA_FILE="/etc/kubernetes/pki/etcd/ca.pem"
ETCD_PEER_CLIENT_CERT_AUTH="true"
# 클라이언트 인증 설정
ETCD_CLIENT_CERT_AUTH="true"
```

데몬 재시작 및 서비스 재시작

```
# ETCD_INITIAL_CLUSTER_STATE=“existing” 에서 “new” 변경
# 기존 데이터 삭제 후 새로 클러스터링 (데이터 유지하려면 백업 후 새로 클러스터링 후 데이터 복원 진행)
systemctl daemon-reload
systemctl stop etcd
rm –rf /var/lib/etcd/*
systemctl restart etcd
```

클러스터링 재 설정 후 상태 확인

- **Etcd 환경변수 파일에서 mTLS 활성화 상태이므로 Client 인증서를 사용해서 ETCD endpoint status 를 확인한다.**

```
etcdctl   --endpoints=https://11.11.20.10:2379,https://11.11.20.20:2379,https://11.11.20.30:2379   \
--cacert=/etc/kubernetes/pki/etcd/ca.pem   \
--cert=/etc/kubernetes/pki/etcd/apiserver-etcd-client.pem   \
--key=/etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem   \
endpoint status --write-out=table
```

```
[root@etcd-01 etcd]# etcdctl   --endpoints=https://11.11.20.10:2379,https://11.11.20.20:2379,https://11.11.20.30:2379   --cacert=/etc/kubernetes/pki
lient-key.pem   endpoint status --write-out=table
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|         ENDPOINT         |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| https://11.11.20.10:2379 | 5bfd16909ca015cd |  3.5.19 |   20 kB |     false |      false |         2 |          8 |                  8 |        |
| https://11.11.20.20:2379 | b25d039fde31f26d |  3.5.19 |   20 kB |     false |      false |         2 |          8 |                  8 |        |
| https://11.11.20.30:2379 | 987187cdf82bdad7 |  3.5.19 |   20 kB |      true |      false |         2 |          8 |                  8 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
```

K8S 클러스터링 및 External ETCD 클러스터링 구성

K8S Clustering 가이드

SWAP 비활성화

1. **/etc/fstab 들어가서 swap 부분 주석 처리 후 파일 저장**
2. **swapoff --a 명령어 실행 후 swap 사용하지 않기**

IP Forward 설정 및 보안 설정 해제

- **IP Forward 설정**

```
# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system

sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/configsystemctl stop firewalld
systemctl disable firewalld
```

- **Bridge 네트워크 모듈 활성화 및 Overlay 모듈 활성화**

```
modprobe overlay
modprobe br_netfilter

echo "br_netfilter" > /etc/modules-load.d/k8s.conf
echo "overlay" | sudo tee -a /etc/modules-load.d/k8s.conf
cat <<EOF | tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system
```

각 노드 별 /etc/hosts 파일에 도메인과 IP 주소 등록

```
[root@master-03 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
11.11.10.101    master-01
11.11.10.102    master-02
11.11.10.103    master-03
```

```
cat << EOF >> /etc/hosts
11.11.10.10 master-01
11.11.10.20 master-02
11.11.10.30 master-03
11.11.10.40 worker-01
11.11.10.50 worker-02
11.11.10.60 worker-03
EOF
```

K8S Clustering 가이드

Kubeadm, kubelet 등 설치

kubernetes

Documentation Kubernetes Blo

Search this site

- Documentation
- Getting started
  - Learning environment
  - Production environment
    - Container Runtimes
    - Installing Kubernetes with deployment tools
      - Bootstrapping clusters with kubeadm
        - **Installing kubeadm**
        - Troubleshooting kubeadm
        - Creating a cluster with kubeadm
        - Customizing components with the kubeadm API
        - Options for Highly Available Topology
        - Creating Highly Available Clusters with kubeadm
        - Set up a High Availability etcd Cluster with kubeadm
        - Configuring each kubelet in your cluster using kubeadm
        - Dual-stack support with kubeadm
      - Turnkey Cloud Solutions
  - Best practices
- Concepts
- Tasks
- Tutorials
- Reference
- Contribute

Installing kubeadm, kubelet and kubectl

You will install these packages on all of your machines:

- `kubeadm` : the command to bootstrap the cluster.
- `kubelet` : the component that runs on all of the machines in your cluster and does things like starting pods and containers.
- `kubectl` : the command line util to talk to your cluster.

kubeadm **will not** install or manage `kubelet` or `kubectl` for you, so you will need to ensure they match the version of the Kubernetes control plane you want kubeadm to install for you. If you do not, there is a risk of a version skew occurring that can lead to unexpected, buggy behaviour. However, *one* minor version skew between the kubelet and the control plane is supported, but the kubelet version may never exceed the API server version. For example, the kubelet running 1.7.0 should be fully compatible with a 1.8.0 API server, but not vice versa.

For information about installing `kubectl`, see Install and set up kubectl.

Warning:
These instructions exclude all Kubernetes packages from any system upgrades. This is because kubeadm and Kubernetes require special attention to upgrade.

For more information on version skews, see:

- Kubernetes version and version-skew policy
- Kubeadm-specific version skew policy

Note: The legacy package repositories (`apt.kubernetes.io` and `yum.kubernetes.io`) have been deprecated and frozen starting from September 13, 2023. **Using the new package repositories hosted at `pkgs.k8s.io` is strongly recommended and required in order to install Kubernetes versions released after September 13, 2023.** The deprecated legacy repositories, and their contents, might be removed at any time in the future and without a further notice period. The new package repositories provide downloads for Kubernetes versions starting with v1.24.0.

Note:
There's a dedicated package repository for each Kubernetes minor version. If you want to install a minor version other than v1.32, please see the installation guide for your desired minor version.

Debian-based distributions | Red Hat-based distributions | Without a package manager

1. Set SELinux to `permissive` mode:

1. Kubeadm install Guide에서 리눅스 OS 선택 후 가이드에 따라 진행
2. 각 노드마다 동일하게 진행

- **레포지터리 추가**

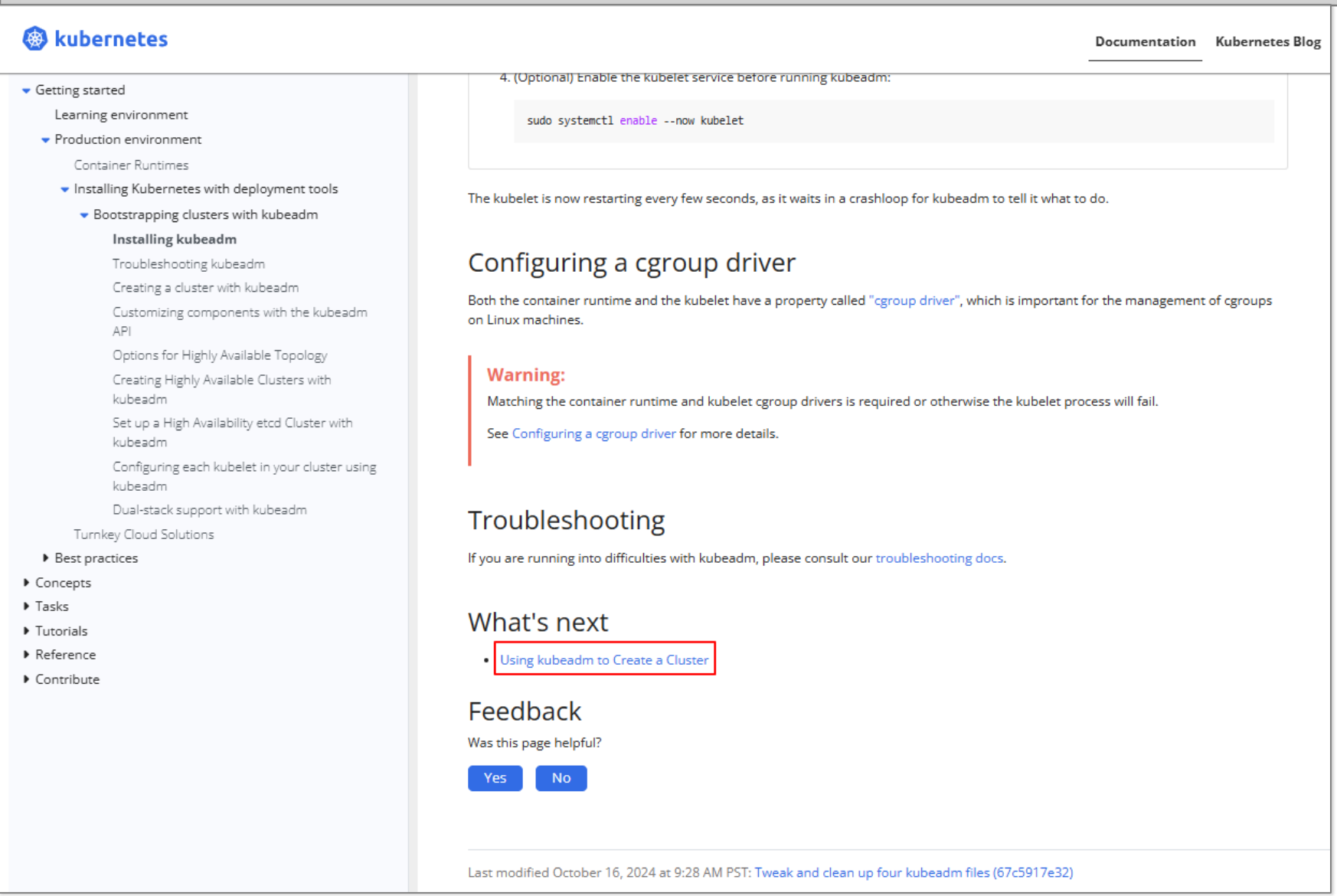
```
# This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
```

- **kubeadm 및 kubelet 설치**

```
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
sudo systemctl enable --now kubelet
```

K8S Clustering 가이드

Kubeadm 을 통한 클러스터 생성 방법



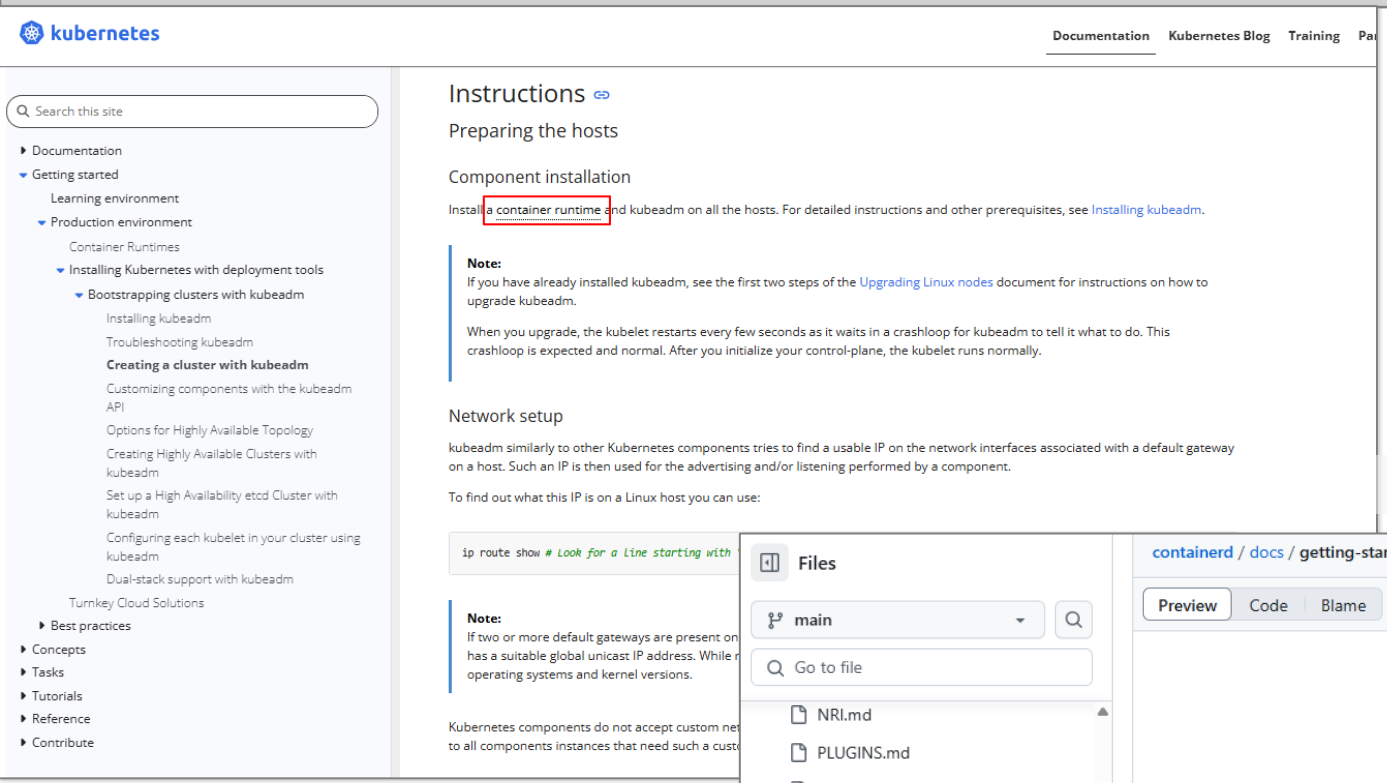
1. 페이지 제일 하단에 Using kubeadm to Create a Cluster 링크 클릭

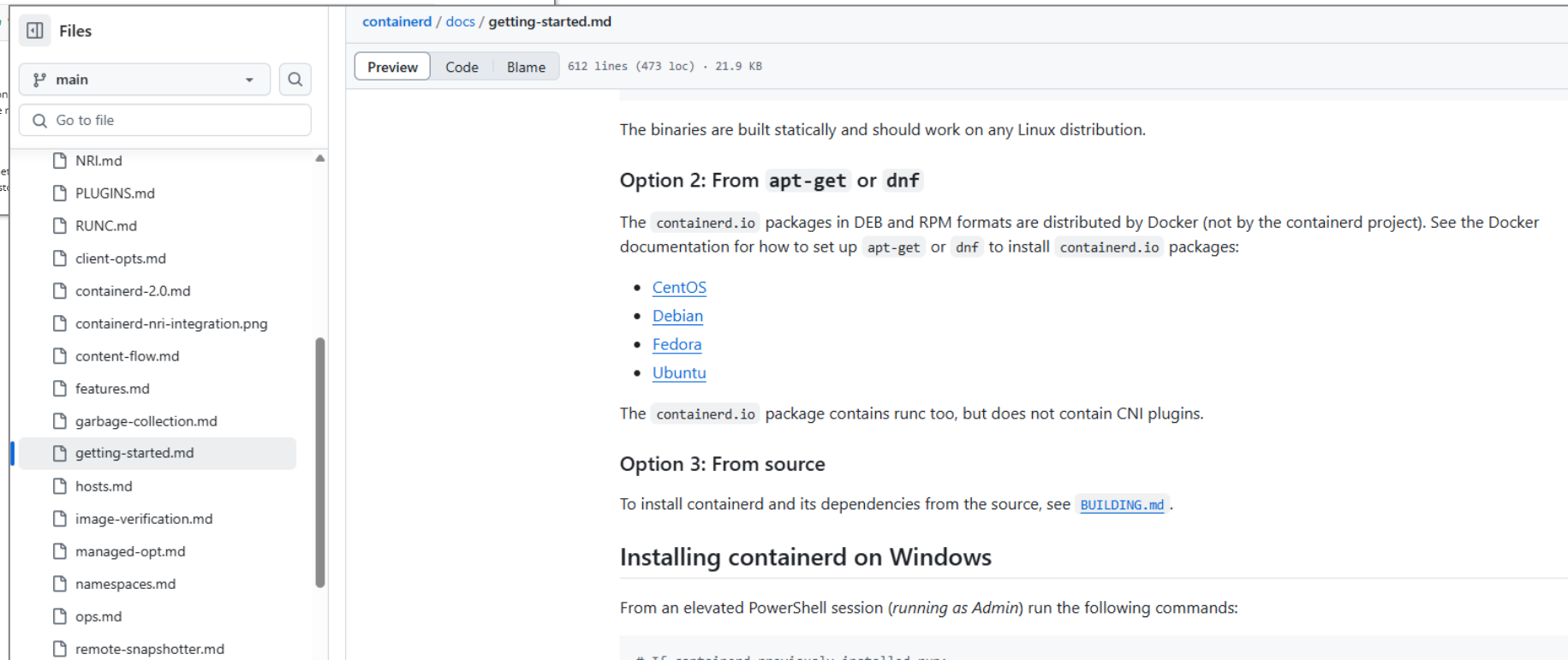
K8S Clustering 가이드

클러스터링하기 위해 필요한 Containerd 설치

1. Kubeadm을 이용해 클러스터링 자동 설치를 위해서 첫 번째로 모든 노드에 containerd 를 설치해야 한다.
2. 해당 페이지에서 중간쯤 container runtime 링크를 새창으로 연다.
3. 클릭 후 container runtimes 툴이 4가지 보이는데 containerd 를 설치한다. Containerd 를 클릭하면 설치를 위한 방법에 대한 상세설명이 열리는데 구글에 containerd install 검색 후 Getting started with containerd 시작하는 페이지에서 설치 가이드를 따른다.
4. https://github.com/containerd/containerd/blob/main/docs/getting-started.md
5. 접속 설치가 필요한 서버OS 에 맞게 링크를 새창으로 열어서 가이드에 따라 containerd 를 설치하고 데몬을 enable 해준다.

```
systemctl restart containerd; systemctl enable containerd
```





K8S Clustering 가이드

Containerd를 Cgroup driver 로 사용하기 위한 방법

```
[root@master-03 ~]# cat /etc/containerd/config.toml | grep SystemdCgroup -B 50
      ignore_blockio_not_enabled_errors = false
      ignore_rdt_not_enabled_errors = false
      no_pivot = false
      snapshotter = "overlayfs"

      [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime]
        base_runtime_spec = ""
        cni_conf_dir = ""
        cni_max_conf_num = 0
        container_annotations = []
        pod_annotations = []
        privileged_without_host_devices = false
        privileged_without_host_devices_all_devices_allowed = false
        runtime_engine = ""
        runtime_path = ""
        runtime_root = ""
        runtime_type = ""
        sandbox_mode = ""
        snapshotter = ""

        [plugins."io.containerd.grpc.v1.cri".containerd.default_runtime.options]

      [plugins."io.containerd.grpc.v1.cri".containerd.runtimes]

        [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
          base_runtime_spec = ""
          cni_conf_dir = ""
          cni_max_conf_num = 0
          container_annotations = []
          pod_annotations = []
          privileged_without_host_devices = false
          privileged_without_host_devices_all_devices_allowed = false
          runtime_engine = ""
          runtime_path = ""
          runtime_root = ""
          runtime_type = "io.containerd.runc.v2"
          sandbox_mode = "podsandbox"
          snapshotter = ""

          [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
            BinaryName = ""
            CriuImagePath = ""
            CriuPath = ""
            CriuWorkPath = ""
            IoGid = 0
            IoUid = 0
            NoNewKeyring = false
            NoPivotRoot = false
            Root = ""
            ShimCgroup = ""
            SystemdCgroup = true
```

1. **Kubelet Cgroup Driver 를 Containerd 와 Cgroupfs 중 하나를 선택해서 사용할 수 있다.일반적으로 init 시스템인 Containerd를 선택한다. Cgroup Driver 는 컨테이너가 사용할 수 있는 자원을 할당 및 제어하는 역할을 하는데 Cgroupfs 를 사용하는 경우 Processor 1 인 Containerd 와 자원 할당 주소 충돌이 발생할 여지가 있어서 containerd로 모든 자원할당 역할 담당하도록 통일한다.**
2. **설정 방법**

- **- 모든 노드에 아래 명령어 실행**

```
sudo mkdir –p /etc/containerd;
containerd config default | sed 's/SystemdCgroup = false/SystemdCgroup = true/' | sudo tee /etc/containerd/config.toml
```

위 명령어 실행하면 config.tml 파일에 containerd default config 설정에서 SystemdCgroup = false 에서 true 로 변경해서 파일에 저장한다.

잘 변경되었는지 확인은 아래 명령어 입력으로 true 값 변경 여부를 확인한다.

이후 아래 명령어로 데몬 재시작 및 활성화한다.

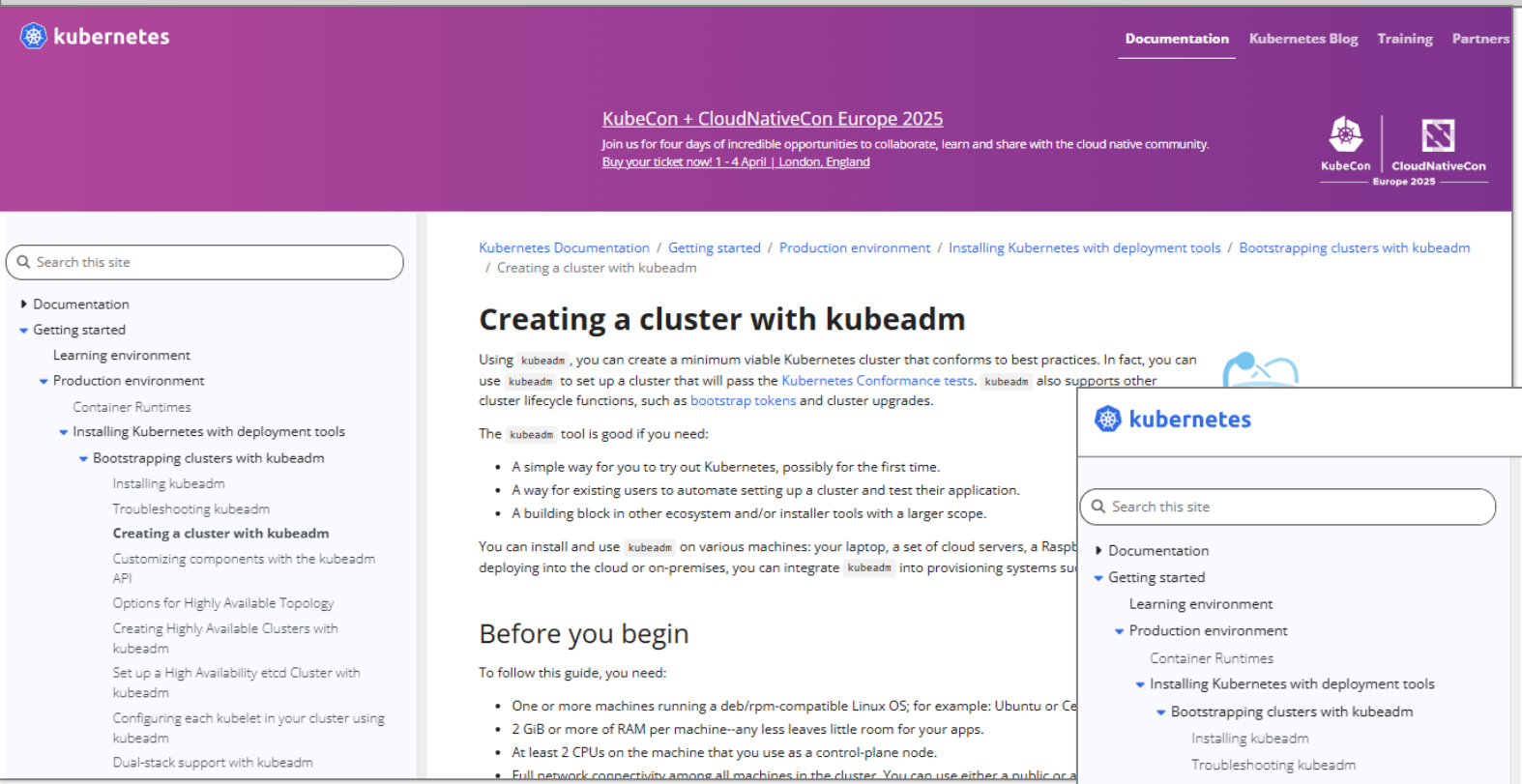
```
cat /etc/containerd/config.toml | grep SystemdCgroup -B 50
```

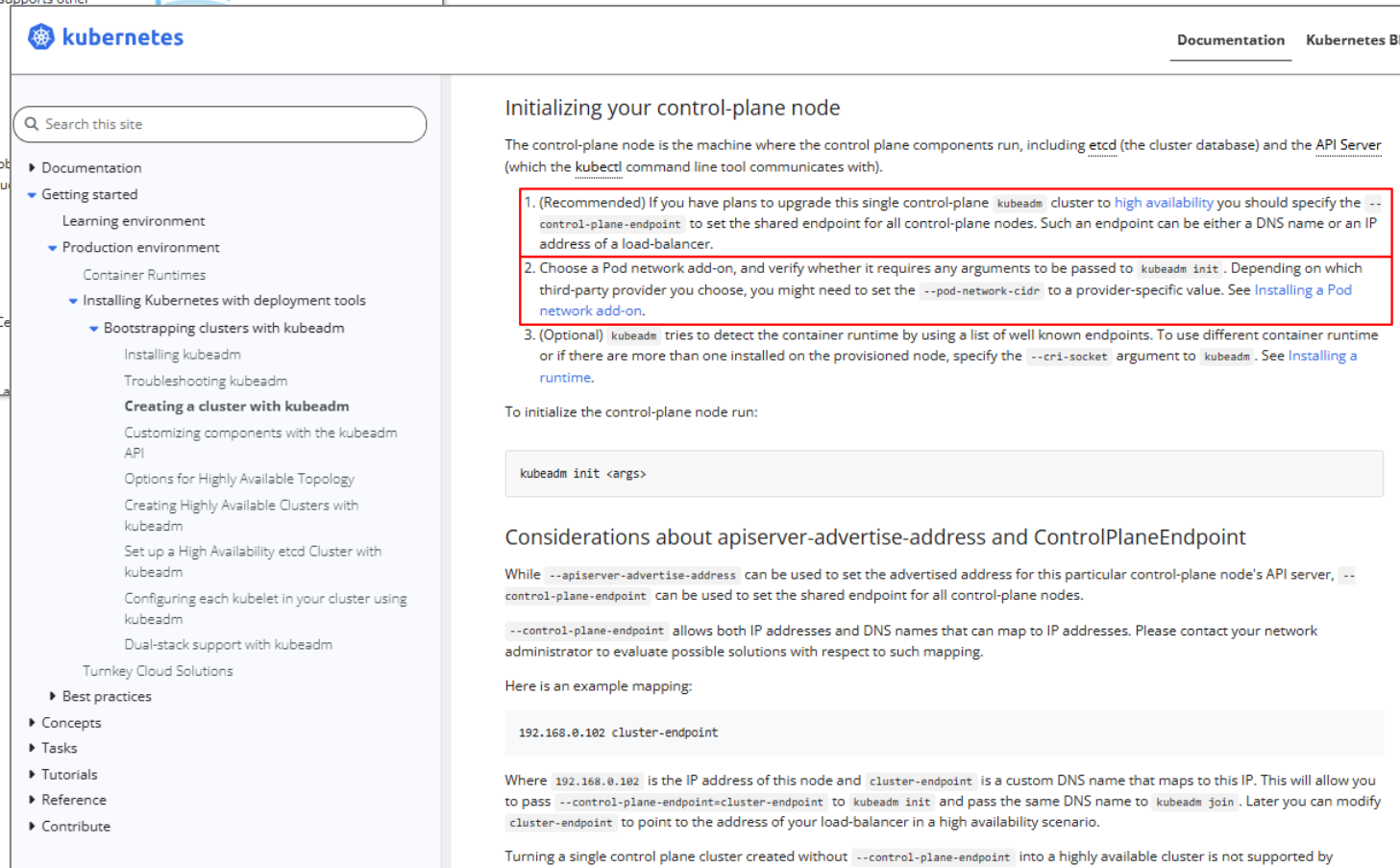
```
systemctl restart containerd
```

K8S Clustering 가이드

클러스터링을 위한 초기 단계

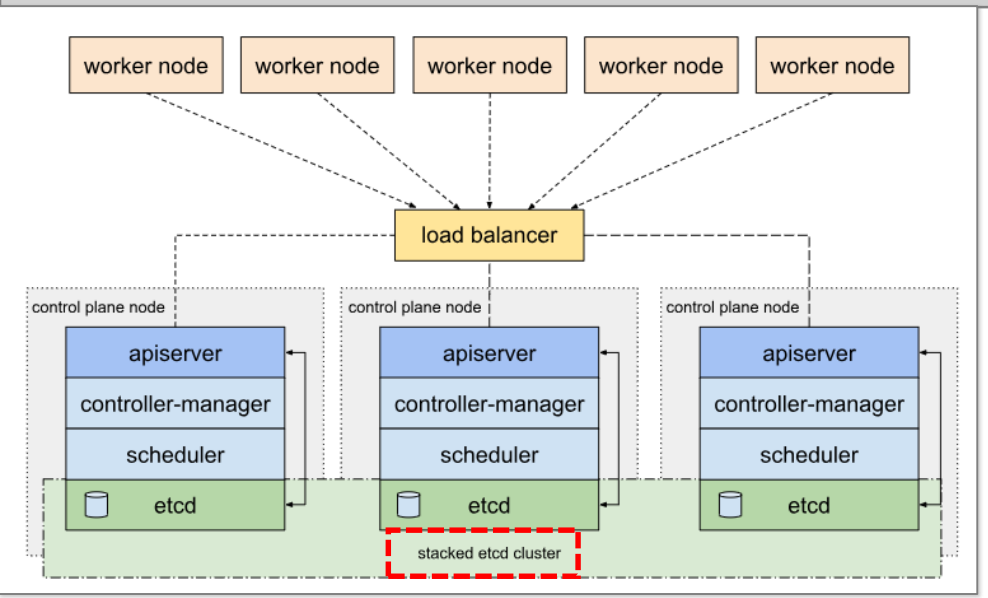
1. Containerd 설치 후 다시 Creating a cluster with kubeadm 페이지로 돌아와서 control-plan node 초기화 내용을 확인한다.
2. Master 노드 이중화 케이스는 첫 번째 빨간색 박스 내용을 확인하고 클러스터링을 한다. --control-plane-endpoint IP주소에 API LB 를 설정하면 된다.
3. 1 Master 1 + Worker node 경우 두 번째 빨간색 박스 내용을 확인한다.



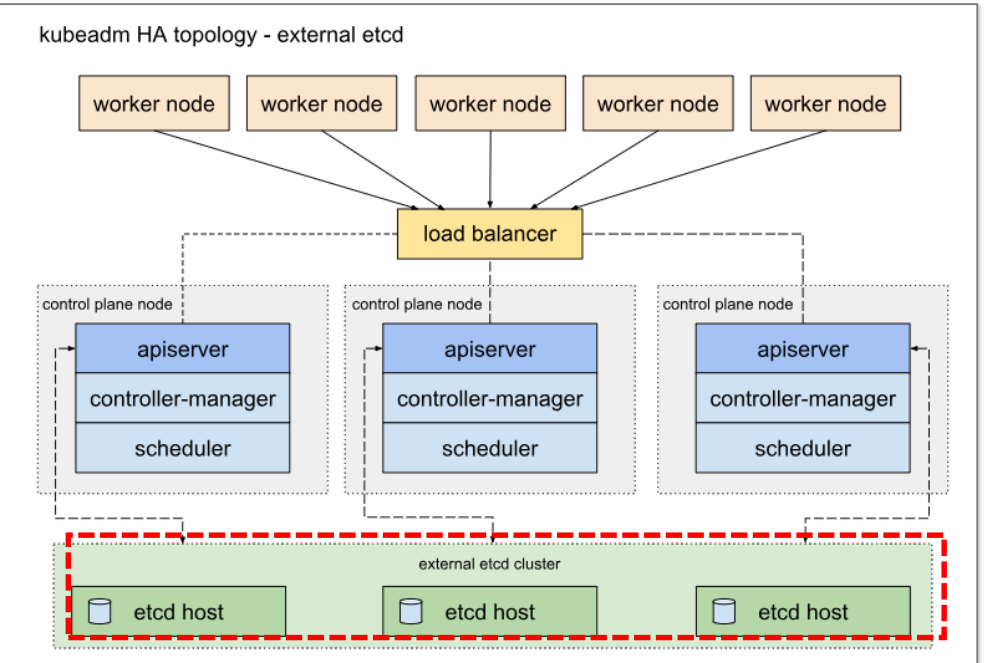


K8S Clustering 가이드

ETCD 구성



1. 일반적으로 단일 Master 노드를 통해 클러스터링하는 경우보다 여러 대의 Master 노드를 통해 클러스터링하여 HA 를 확보한다.
2. 왼쪽 그림은 상단 LB 를 통해서 멀티 클러스터링한 구조이다. 해당 구조는 클러스터링을 관리하는 DB를 master 노드 안에 배치하는 방식(stacked)과 외부에 별도 클러스터를 두어 관리하는 방식(external) 두 가지 형태로 설계된다.
3. https://kubernetes.io/ko/docs/setup/production-environment/tools/kubeadm/ha-topology/
4. 위 링크 클릭해서 내용 확인해보면 Stacked와 External 은 장단점이 있다. Stacked 은 Master node 와 etcd 는 1:1 맵핑되어 설치 및 제거되기 때문에 Master node 가 다운되면 etcd도 없어지고 Master node를 추가하면 etcd 도 추가된다.
5. External ETCD 클러스터링은 별도의 3개 이상의 노드가 추가적으로 필요한 단점이 있지만 Master node 와 분리 운영되기 때문에 Master node 와 1:1 을 이루지않고 독립적으로 외부에서 운영 가능하다는 큰 장점이 있다.



K8S Clustering 가이드

Stacked ETCD

Stacked control plane and etcd nodes

Steps for the first control plane node

1. Initialize the control plane:

```
sudo kubeadm init --control-plane-endpoint "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT" --upload-certs
```

- You can use the `--kubernetes-version` flag to set the Kubernetes version to use. It is recommended that the versions of kubeadm, kubelet, kubectl and Kubernetes match.
- The `--control-plane-endpoint` flag should be set to the address or DNS and port of the load balancer.
- The `--upload-certs` flag is used to upload the certificates that should be shared across all the control-plane instances to the cluster. If instead, you prefer to copy certs across control-plane nodes manually or using automation tools, please remove this flag and refer to Manual certificate distribution section below.

> **Note:**
> The `kubeadm init` flags `--config` and `--certificate-key` cannot be mixed, therefore if you want to use the kubeadm configuration you must add the `certificateKey` field in the appropriate config locations (under `InitConfiguration` and `JoinConfiguration: controlPlane`).

> **Note:**
> Some CNI network plugins require additional configuration, for example specifying the pod IP CIDR, while others do not. See the CNI network documentation. To add a pod CIDR pass the flag `--pod-network-cidr`, or if you are using a kubeadm configuration file set the `podSubnet` field under the `networking` object of `ClusterConfiguration`.

The output looks similar to:

```
...
You can now join any number of control-plane node by running the following command on each as a root:
    kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash sha256:7c2e69131a36ae2a042a339b33381c6d0d43

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use kubeadm init phase upload-certs to reload certs afterwa

Then you can join any number of worker nodes by running the following on each as root:
    kubeadm join 192.168.0.200:6443 --token 9vr73a.a8uxyaju799qwdjv --discovery-token-ca-cert-hash sha256:7c2e69131a36ae2a042a339b33381c6d0d43
```

1. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
2. 위 링크에서 stacked control plan and etcd 방식과 External etcd nodes 방식 클러스터링 가이드가 있다.
3. Master Node-HA 구성의 Stacked ETCD 경우 API LB 용 프록시가 필요하다.
4. KS 사용해도되고 마음에 드는 소프트웨어 프록시 설치해서 SLB 셋팅 완료 후 kubeadm ini –control-plane= “SLB VIP” –upload-certs 명령어 입력하면 Work node join 정보(토큰 등)와 Master node join 정보(토큰 등)가 출력된다.
5. 노드 역할에 맞게 위 출력 정보를 복사해서 입력만 해주면 정상적으로 Join 되고 kubectl get node 명령어를 통해 클러스터링 상태를 확인할 수 있다.
6. 이후 CNI(Container Network Interface) 가 있어야지만 컨테이너 간 통신이 가능하기 때문에 Third Party CNI 중 마음에 드는 것을 선택해서 Master node 처음 설치해주면 정상적으로 클러스터링이 완료된다. (Calico, Weave 많이 사용한다.)

* https://github.com/containernetworking/cni

K8S Clustering 가이드

External ETCD 클러스터링 대상으로 Master Node HA 구성

External etcd nodes

Setting up a cluster with external etcd nodes is similar to the procedure used for stacked etcd with the exception that you should setup etcd first, and you should pass the etcd information in the kubeadm config file.

Set up the etcd cluster

1. Follow these instructions to set up the etcd cluster.
2. Set up SSH as described here.
3. Copy the following files from any etcd node in the cluster to the first control plane node:

```
export CONTROL_PLANE="ubuntu@10.0.0.7"
scp /etc/kubernetes/pki/etcd/ca.crt "${CONTROL_PLANE}":
scp /etc/kubernetes/pki/apiserver-etcd-client.crt "${CONTROL_PLANE}":
scp /etc/kubernetes/pki/apiserver-etcd-client.key "${CONTROL_PLANE}":
```

   - Replace the value of `CONTROL_PLANE` with the `user@host` of the first control-plane node.

Set up the first control plane node

1. Create a file called `kubeadm-config.yaml` with the following contents:

```
---
apiVersion: kubeadm.k8s.io/v1beta4
kind: ClusterConfiguration
kubernetesVersion: stable
controlPlaneEndpoint: "LOAD_BALANCER_DNS:LOAD_BALANCER_PORT" # change
etcd:
  external:
    endpoints:
      - https://ETCD_0_IP:2379 # change ETCD_0_IP appropriately
      - https://ETCD_1_IP:2379 # change ETCD_1_IP appropriately
      - https://ETCD_2_IP:2379 # change ETCD_2_IP appropriately
    caFile: /etc/kubernetes/pki/etcd/ca.crt
    certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt
    keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key
```

사전 준비

1. **Master node 3 대 준비 (containerd, kubeadm, kubelet 설치 및 swap off, ip forward 설정 완료 상태)**
2. **Master node API SLB 준비**
3. **각 Master node에 ETCD 클라이언트 인증서와 CA 인증서 적절한 위치에 배포**

```
scp apiserver-etcd-client.pem apiserver-etcd-client-key.pem bhs@11.11.10.10:/home/bhs
scp apiserver-etcd-client.pem apiserver-etcd-client-key.pem bhs@11.11.10.20:/home/bhs
scp apiserver-etcd-client.pem apiserver-etcd-client-key.pem bhs@11.11.10.30:/home/bhs
```

Kubeadm-config.yaml 파일 작성

```
apiVersion: kubeadm.k8s.io/v1beta3
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 11.11.10.10  # master-1 IP
  bindPort: 6443 ##  init 과정 중 api-server listen 상태를 위한 설정으로 보임
---
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: stable
controlPlaneEndpoint: "11.11.10.100:6443"  #KS SLB VIP 설정
networking:
  podSubnet: 10.244.0.0/16  # CNI 플러그인과 일치해야 함
etcd:
  external:
    endpoints:
      - https://11.11.20.10:2379
      - https://11.11.20.20:2379
      - https://11.11.20.30:2379
    caFile: /etc/kubernetes/pki/etcd/ca.pem
    certFile: /etc/kubernetes/pki/etcd/apiserver-etcd-client.pem
    keyFile: /etc/kubernetes/pki/etcd/apiserver-etcd-client-key.pem
```

kubeadm init 진행

kubeadm init --config=kubeadm-config.yaml --upload-certs

출력 결과

[root@master-01 cluster]# kubeadm init --config=kubeadm-config.yaml --upload-certs
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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

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/

You can now join any number of control-plane nodes running the following command on each as root:

**kubeadm join 11.11.10.100:6443 --token 3w1l07.sele1itw6hjxrlhg **
**--discovery-token-ca-cert-hash sha256:7f09974641c39fcf6193cb9943149e773d083c9ad63e3d49ec0efbb1eab97a66 **
--control-plane --certificate-key e182d0059f59114a64a0a41095451d2fa967a10cde338088c9a55c4ca3a4d043

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following on each as root:

**kubeadm join 11.11.10.100:6443 --token 3w1l07.sele1itw6hjxrlhg **
--discovery-token-ca-cert-hash sha256:7f09974641c39fcf6193cb9943149e773d083c9ad63e3d49ec0efbb1eab97a66

Master 2, 3 노드에서 클러스터링 join 진행

```
kubeadm join 11.11.10.100:6443 --token 3w1l07.sele1itw6hjxrlhg \
        --discovery-token-ca-cert-hash sha256:7f09974641c39fcf6193cb9943149e773d083c9ad63e3d49ec0efbb1eab97a66 \
        --control-plane --certificate-key e182d0059f59114a64a0a41095451d2fa967a10cde338088c9a55c4ca3a4d043
```

CNI는 간단한 Flannel으로 설치

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

마스터노드 상태 확인

```
[root@master-01 ~]# kubectl get node
NAME        STATUS   ROLES           AGE   VERSION
master-01   Ready    control-plane   18m   v1.32.3
master-02   Ready    control-plane   86s   v1.32.3
master-03   Ready    control-plane   88s   v1.32.3
```

KS API LB 를 통해 정상적으로 부하분산 중

```
================================================================================
  slb
 --------------------------------------------------------------------------------
   Name    : api_lb
   Service : TCP 11.11.10.100:6443
   State   : ACT
   Real      (LB-Method: rr)                    Sessions ------------ Throughput - Packets ----
             ID  Name      RIP         Rport State Cps Current Peak Total Inbps Outbps Inpps Outpps
             10  master-01 11.11.10.10 6443  act     1       6   33    68 19917  18595     8      6
             20  master-02 11.11.10.20 6443  act     1       8    8     8 19296  23740     7      5
             30  master-03 11.11.10.30 6443  act     1       7    7     8 17921  19275     6      4
   Total     :                                       3      21   33    84 57135  61610    23     16
================================================================================
API_LB(config)# show entry
================================================================================
Prot [Org]Sip:Sport    Dip:Dport         - [Rep]Sip:Sport   Dip:Dport         Svc:Real      [R]Svc:Real
--------------------------------------------------------------------------------
tcp  11.11.10.10:42948 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:42948 slb.api_lb:30
tcp  11.11.10.10:42932 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:42932 slb.api_lb:10
tcp  11.11.10.10:42944 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:42944 slb.api_lb:30
tcp  11.11.10.10:42922 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:42922 slb.api_lb:20
tcp  11.11.10.20:49146 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:49146 slb.api_lb:20
tcp  11.11.10.20:49088 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:49088 slb.api_lb:20
tcp  11.11.10.20:49118 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:49118 slb.api_lb:10
tcp  11.11.10.20:49102 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:49102 slb.api_lb:30
tcp  11.11.10.20:49068 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:49068 slb.api_lb:30
tcp  11.11.10.10:42950 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:42950 slb.api_lb:20
tcp  11.11.10.10:42970 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:42970 slb.api_lb:20
tcp  11.11.10.20:49130 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:49130 slb.api_lb:10
tcp  11.11.10.10:42962 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:42962 slb.api_lb:10
tcp  11.11.10.10:42966 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:42966 slb.api_lb:30
tcp  11.11.10.20:49138 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:49138 slb.api_lb:30
tcp  11.11.10.10:42934 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:42934 slb.api_lb:20
tcp  11.11.10.20:49126 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:49126 slb.api_lb:20
tcp  11.11.10.20:49066 11.11.10.100:6443 - 11.11.10.30:6443 11.11.10.99:49066 slb.api_lb:30
tcp  11.11.10.20:49052 11.11.10.100:6443 - 11.11.10.20:6443 11.11.10.99:49052 slb.api_lb:20
tcp  11.11.10.10:42946 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:42946 slb.api_lb:10
tcp  11.11.10.20:49080 11.11.10.100:6443 - 11.11.10.10:6443 11.11.10.99:49080 slb.api_lb:10
================================================================================
```

사용한 인스턴스

컴퓨트 > 인스턴스

인스턴스 생성 | 시작 | 중지 | 재부팅 | 기타 작업 | 여러 필터 태그는 엔터로 구분합니다.

이름	이미지	상태	고정 IP (유동 IP)	플레이버	디스크	태그	잠금	생성 일자	작업
worker-01		Active	11.11.10.40	c4m8	worker-01_hda worker-01_vda			2025-03-21 18:14:07	콘솔
master-03		Active	11.11.10.30 (192.168.181.64)	c4m8	master-03_hda master-03_vda			2025-03-19 16:08:29	콘솔
master-01		Active	11.11.10.10 (192.168.181.73)	c4m8	master-01_hda master-01_vda			2025-03-19 16:08:04	콘솔
master-02		Active	11.11.10.20 (192.168.181.164)	c4m8	master-02_hda master-02_vda			2025-03-19 15:33:59	콘솔
etcd-03		Active	11.11.20.30 (192.168.181.127)	c4m8	etcd-03_hda etcd-03_vda			2025-03-19 09:11:35	콘솔
etcd-02		Active	11.11.20.20 (192.168.181.75)	c4m8	etcd-02_hda etcd-02_vda			2025-03-19 09:11:06	콘솔
etcd-01		Active	11.11.20.10 (192.168.181.187)	c4m8	etcd-01_hda etcd-01_vda			2025-03-19 09:09:54	콘솔
API_LB		Active	11.11.10.100 (192.168.181.97)	c4m8	api-lb_vda			2025-03-10 16:48:53	콘솔

합계 : 8 | 1 | 10 / 페이지

Work Node Join

kubeadm join 11.11.10.100:6443 --token 0zppwm.g5bi24r6hl8ri5xo --discovery-token-ca-cert-hash sha256:7f09974641c39fcf6193cb9943149e773d083c9ad63e3d49ec0efbb1eab97a66

Token 만료 시 Master 에서 재발급 방법

- **kubeadm token list 출력 결과 없으면 마스터노드에서 아래 명령어 실행으로 토큰 재발행**

kubeadm token create --print-join-command

Cluster-info Configmap 확인

- **Configmap 과 cluster-info를 yaml 포맷으로 출력해서 토큰 값 비교 및 확인**

kubectl -n kube-public get configmap cluster-info -o yaml

kubectl 명령어 alias 설정

echo "alias k='kubectl'" >> ~/.bashrc
source ~/.bashrc

Nginx pod 생성 시 정상 생성되는 것을 확인

echo "alias k='kubectl'" >> ~/.bashrc
source ~/.bashrc

```
[root@master-01 ~]# kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          59m
[root@master-01 ~]# kubectl get node
NAME        STATUS   ROLES           AGE     VERSION
master-01   Ready    control-plane   5d      v1.32.3
master-02   Ready    control-plane   4d23h   v1.32.3
master-03   Ready    control-plane   4d23h   v1.32.3
worker-01   Ready    <none>          67m     v1.32.3
```

전체 로그 기록

```
[root@master-01 cluster]# kubeadm init --config=kubeadm-config.yaml --upload-certs
W0321 17:39:59.361604  144749 common.go:101] your configuration file uses a deprecated API spec: "kubeadm.k8s.io/v1beta3" (kinpec using a newer API version.
W0321 17:39:59.363140  144749 common.go:101] your configuration file uses a deprecated API spec: "kubeadm.k8s.io/v1beta3" (kin using a newer API version.
[init] Using Kubernetes version: v1.32.3
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0321 17:40:00.059617  144749 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the container runt
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes.default kubernetes.default.svc kubernetes.defaul
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] External etcd mode: Skipping etcd/ca certificate authority generation
[certs] External etcd mode: Skipping etcd/server certificate generation
[certs] External etcd mode: Skipping etcd/peer certificate generation
[certs] External etcd mode: Skipping etcd/healthcheck-client certificate generation
[certs] External etcd mode: Skipping apiserver-etcd-client certificate generation
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manif
```

전체 로그 기록

Stacked Cluster 참고 블로그
https://engmisankim.tistory.com/14
External etcd
https://engmisankim.tistory.com/15