k8s 클러스터 설치

난이도	중간
기술	설치
생성 일시	@2023년 1월 17일 오전 10:45
생성자	진영 장
최종 편집 일시	@2023년 1월 19일 오전 9:06

[Kubernetes] 쿠버네티스 클러스터 구축

on premise 서버 4대 전부 Ubuntu 18.04 RAM 16GB ~ 128GB 마스터노드 용 서버: 16GB 워커 노드 용 각각 워커 노드 용 서버 전부 GPU 장착 온 프레미스(On premise)방식의 서버 4대 구성에서 마스터 노드로 생각 서버는 GPU 가 장착되어 있지 않고 램도 16GB만 있다. 나머지 3대의 서버는 각각 머신러닝을 위한 GPU도 어느정도 장착되어 있

https://dongle94.github.io/kubernetes/kubernetes-cluster-build/

▼ 1. NTP 서버 동기화

NTP 서버 동기화

`NTP` 서버 동기화는 Network Time Protocol의 약자로서 클러스터를 구축하게 되면 각 노드들이 네트워크 통신을 하게 되는데 이 때 각각의 시간이 맞지 않아 발생할 수 있는 위험을 없애기 위해 해주었다.

```
#Master, Worker
$ sudo apt install ntp
```

마스터 노드(Master Node)

10.100.11.209의 마스터 노드에서는 설치 후 서비스 리로드, 서비스 동작확인 만 해주었다.

```
#Master
$ sudo service ntp reload
$ sudo ntpq -p
```

`pool`로 설정되어 있는 ntp 서버들에서 시간을 받아오는 결과를 확인할 수 있다.

워커 노드(Worker Node)

100~102번의 워커 노드들은 설치 후 ntp 설정을 변경하고 마찬가지로 **서비스 재시작**, **서비스 동작확인**을 해준다.

```
#Worker
$ sudo vi /etc/ntp.conf
...
#모든 pool과 server 주석처리
server 10.100.11.209
...
```

주석이 해제되어 있는 모든 `pool` 과 `server`를 주석처리 하고 다음 마스터 노드의 서버 ip를 추가해주어 ntp 설정을 잡는다.

```
#Worker
$ sudo systemctl restart ntp
$ sudo ntpq -p
```

이어서 서비스를 재시작하고 `ntpq -p` 명령어를 실행하면 npt 클라이언트가 ntp서버에와 동기화 하는 것을 확인할 수 있다.

▼ 2. 스왑메모리 해제(swapoff)

스왑메모리 해제(swapoff)

쿠버네티스 클러스터는 메모리 스왑이 활성화 되어있는 것을 허용하지 않는다.

> *메모리 스왑이 활성화 돼 있으면 컨테이너의 성능이 일관되지 않을 수 있기 때문에 대부분의 쿠버네티스 설치 도구는 메모리 스왑을 허용하지 않습니다.*

사용할 `kubeadm`도 마찬가지로 스왑메모리를 허용하지 않는다. 터미널에서 다음 명령어를 입력하여 스왑메모리를 해제한다.

```
#Master, Worker
$ swapoff -a
```

추가적으로 혹시 노드 들의 재부팅이 잦을 예정이라면 아예 `fstab`
에서 `swap`에 대한 부분을 주석처리 하는 것도 좋다. 위의 작업을 해주고 `fstab`파일을 아래와 같이 열고 `swap`이 적혀있는 부분을 주석처리 해준다.

```
#Master, Worker
$ sudo vi /etc/fstab

>
# swap was on /dev/sda2 during installation
# UUID=e09948a3-(중략)...-79d1d6 none            swap    sw              0       0
```

▼ 3. 도커 엔진 설치

docker 설치

```
#Master, Worker
$ cat > docker.sh
#!/usr/bin/env bash
## INFO: https://docs.docker.com/engine/install/ubuntu/

set -euf -o pipefail

DOCKER_USER=ubuntu

# Install dependencies
sudo apt-get update && sudo apt-get install -y \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg \
  lsb-release

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --yes --dearmor -o /usr/share/keyrings/docker-archive-keyring.gp

# Set up the stable repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/li
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker CE
sudo apt-get update && sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# Use Docker without root
sudo usermod -aG docker $DOCKER_USER

##############################enter -> ctrl+c

$ chmod +x docker.sh

$./docker.sh
```

docker compose 설치

```
#Master, Worker
$ cat > docker-compose.sh
```

```
#!/usr/bin/env bash
## INFO: https://docs.docker.com/compose/install/

set -euf -o pipefail

DOCKER_COMPOSE_VERSION=v2.1.1

# Download and install
sudo curl -L "https://github.com/docker/compose/releases/download/${DOCKER_COMPOSE_VERSION}/docker-compose-$(uname -s)-$(uname -m)"
sudo chmod +x /usr/local/bin/docker-compose
##############################enter -> ctrl+c

$ chmod +x docker-compose.sh

$./docker-compose.sh
```

▼ 4. 도커 데몬 변경

도커 데몬 변경

도커 데몬의 컨테이너 런타임을 변경한다. 간략히 설명해 보면 일반적으로 리눅스에서 프로세스를 관리하며 리소스를 컨트롤 하는 애들이 `systemd`와 `cgroupfs`으로 나뉜다. 컨테이너의 경우 `cgroupfs`을 사용하는데 일반적 프로세스인 `systemd`와 충돌을 일으켜 효율적인 자원관리가 안될 수 있다는 것이다. 그래서 `systemd`로 컨타이너의 런타임을 교체해준다. 아래 명령어를 실행한다.

```
#Master, 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",

  "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    },
  "default-runtime": "nvidia"
}
EOF
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

내용 중 `runtimes`와 `default-runtime`은 쿠버네티스에서 GPU 자원을 쓰기 위해 추가하는 부분이다. 기본 런타임을 `nvidia`로 지정함으로써 K8S 환경에서도 GPU 리소스를 사용할 수 있게한다.

▼ 5. 저장소 추가 및 Kubeadm 설치

저장소 추가 및 Kubeadm 설치

일단은 gpg 저장소를 추가하여 `kubeadm`을 받을 수 있는 환경을 만들고 업데이트를 실행해준다. `root` 계정에서 명령어를 사용한다면 `sudo`는 생략하면 된다. 해당 작업은 마스터와 워커 노드 각각 공통적으로 적용해준다.

```
#Master, Worker
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
$ sudo cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
$ sudo apt update
```

다음과 같이 쿠버네티스를 포함한 설치 패키지를 내려받는다. 필요하다면 버전을 명시할 수 있다.

```
#Master, Worker
$ sudo apt install -y kubelet kubeadm kubectl kubernetes-cni
```

버전 명시 설치

설치 가능한 버전을 알아보려면 아래의 명령어를 통해 확인할 수 있다.

```
#Master, Worker
$ curl -s https://packages.cloud.google.com/apt/dists/kubernetes-xenial/main/binary-amd64/Packages

# 버전만 출력
$ curl -s https://packages.cloud.google.com/apt/dists/kubernetes-xenial/main/binary-amd64/Packages | grep Version | awk '{print $2

#kubelet, kubeadmn, kubectl 설치
$ sudo apt install -y kubelet=1.26.0-00 kubeadm=1.26.0-00 kubectl=1.26.0-00 kubernetes-cni
```

쿠버네티스 버전 고정

시간이 지남에 따라 쿠버네티스도 버전들이 업데이트 되어 간다. 추후 `sudo apt update` 시에 버전을 명시 하지 않은 패키지들의 제공 버전이 바뀔 수 있다. 그래서 현재 버전으로 계속 삭제 및 설치를 고려한다면 다음과 같이 패키지들의 버전을 고정할 수 있다.

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

▼ 6. 클러스터 실행

마스터 노드(Master Node)

마스터노드에선 `kubeadm` 의 `init` 명령어를 이용하여 쿠버네티스 클러스터를 시작할 수 있다. 아래의 명령어 중의 `pod-network-cidr` 옵션에 들어가는 ip는 기억을 해두자 아래의 네트워크 설정 시에 해당 ip를 사용할 일이 있다.

▼ [preflight] Running pre-flight checks error execution phase preflight:

```
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR CRI]: container runtime is not running: output: E0117 06:29:13.177433   26854 remote_runtime.go:948] "Status fro
time="2023-01-17T06:29:13Z" level=fatal msg="getting status of runtime: rpc error: code = Unimplemented desc = unknown service
, error: exit status 1
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
```

```
#Master, Worker
#/etc/containerd/config.toml 파일에서 disabled_plugins 항목에서 CRI 제거
$ sudo vim /etc/containerd/config.toml
...
disabled_plugins = [""]
...

$ sudo systemctl restart containerd
```

```
#Master
#calico나 flannel 둘중 하나만 사용한다.
#CNI를 calico 사용
$ sudo kubeadm init --apiserver-advertise-address 10.100.11.209 --pod-network-cidr=192.168.10.0/24

#CNI를 calico가 아니라 flannel로 사용하려면 --pod-network-cidr를 다르게 해서 만드는 편이 낫다.
#CNI를 flannel 사용
$ sudo kubeadm init --apiserver-advertise-address 10.100.11.209 --pod-network-cidr=10.244.10.0/24
```

위의 명령어를 실행하면 출력되는 결과에 2가지 볼 것이 있다. 하나는 마스터 노드에서 이어서 작업해줘야하는 부분, 다른 하나는 워커 노드들에서 실행할 `join` 명령어 이다. 마스터노드는 주로 다음과 같이 나온다.

```
To start using your cluster, you need to run the following as a regular user:
 #Master에서 실행할 부분
  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/

Then you can join any number of worker nodes by running the following on each as root:
 #Worker에서 실행할 부분
kubeadm join 10.100.11.209:6443 --token s43nna.n2uzng4pl5r2x74j \
        --discovery-token-ca-cert-hash sha256:780917540b1d105299641788d848b8edad00caca640ee842d82ac987097a9a19
```

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

출력되는 부분을 잘 복사해서 그대로 마스터 노드에서 붙여넣기로 실행한다. 그리고 아래에 더 출력된 `init` 명령어는 워커노드에서 실행한다.

워커 노드(Worker Node)

워커노드에서 터미널을 열고 마스터노드에서 `init`을 해서 나온 `join`명령어를 실행해준다. 아래의 형식과 비슷할 것이다.

```
# Worker
$ sudo kubeadm join 10.100.11.209:6443 --token s43nna.n2uzng4pl5r2x74j \
        --discovery-token-ca-cert-hash sha256:780917540b1d105299641788d848b8edad00caca640ee842d82ac987097a9a19
```

마스터 노드(Master Node)

임시 확인

아래의 네트워크 연결까지 해야 클러스터 구성이 제대로 되었는지 확인할 수 있지만 일단 `init`과 `join`이 잘 되었는지는 다음과 같이 확인할 수 있다. 마스터 노드에서 아래의 명령어를 실행해보자.

```
$ kubectl get nodes
NAME                 STATUS     ROLES           AGE     VERSION
paasta-ta-cluster-1   NotReady   control-plane   13m     v1.26.0
paasta-ta-cluster-2   NotReady   <none>          3m23s   v1.26.0
paasta-ta-cluster-3   NotReady   <none>          107s    v1.26.0
paasta-ta-cluster-4   NotReady   <none>          9s      v1.26.0
```

마스터 노드는 `control-plane` 또는 `master`라고 표시되어 있는 것을 확인할 수 있다. 컨트롤 플레인 이라고 하면 '컨테이너의 라이프 사이클을 정의, 배포, 관리하기 위한 API와 인터페이스들을 노출하는 컨테이너 오케스트레이션 레이어'라고 그 뜻을 찾아볼 수 있지만 이에 해당하는 컴포넌트를 실행하는 호스트를 일반적으로 `Master Node`를 지칭한다고 알고 있으면 될 것 이다.

▼ 7. 네트워크 실행

마스터 노드(Master Node)

Calico일 경우

본인의 클러스터 구축 예제에서는 `Calico` 오버레이 네트워크를 설정한다. 설정 파일을 내려받아 클러스터가 구축된 상태에서 `kubectl apply -f`로 실행하여 네트워크를 적용하면 워커노드의 상태를 실시간 네트워크 통신을 통해 알 수 있게 되는 것이다. 아래의 명령어를 통해 적용한다.

여기서 아까 우리가 마스터 노드에 클러스터를 `init`할 때 입력한 ip가 필요하다. 기본값은 `192.168.0.0/16`으로 되어있으나 우리가 입력한 `192.168.10.0/24`으로 yaml파일의 설정을 바꿔줄 것이다.

```
#Master
$ wget https://docs.projectcalico.org/manifests/calico.yaml
$ sed -i -e 's?192.168.0.0/16?192.168.10.0/24?g' calico.yaml
$ kubectl apply -f calico.yaml

#네트워크가 잘 생성 및 적용되었는지 확인해 보려면 생성된 파드를 확인한다.
$ kubectl get pods --namespace kube-system
```

아래 출력 결과에 `calico-`로 시작하는 노드들의 상태를 보며 `Init` 단계를 지나 `PodInitializing` 혹은 `ContainerCreating`을 거쳐 최종적으로 `Running`까지 문제 없이 되는 지 확인하면 된다.

- `Ready` 의 값을 보면 `1/1`을 제대로 만족하는지 확인하면 된다.

Flannel일 경우

다른 클러스터 구축 시 `Flannel` 오버레이 네트워크도 설정해보았다. 위의 마스터노드 실행 시 `--pod-network-cidr`를 `10.244.0.0/24`로 생성하였고 위 방식과 동일하게 `flannel` yaml을 내려받고 ip 항목을 일부 수정하고 실행한다. 버전은 `0.20.0`버전을 사용하였다.

```
#Master
$ wget https://raw.githubusercontent.com/flannel-io/flannel/v0.20.0/Documentation/kube-flannel.yml
$ sed -i -e 's?10.244.0.0/16?10.244.10.0/24?g' kube-flannel.yml
$ kubectl apply -f kube-flannel.yml
```

실행 후 `sudo kubectl get nodes -o wide` 명령어를 실행 해서 노드의 status 가 `NotReady` 에서 `Ready`로 바뀌는지 확인하면 된다. 오버레이 네트워크가 완벽히 실행되는데에 시간이 조금 걸릴 수 있음에 유의한다.

또한 `sudo kubectl get pods -n kube-flannel` 명령어를 통해 해당 네임스페이스에 `flannel`관련 파드가 제대로 생성되고 실행되었는지 확인할 수 있다.