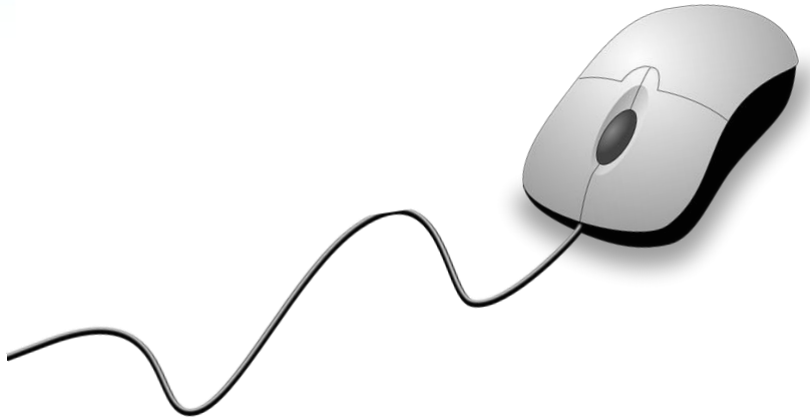
공개SW 솔루션 설치 & 활용 가이드

미들웨어 > 클라우드서비스



kubernetes

제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide

nipa 공개SW역량프라자

CONTENTS

1. 개요

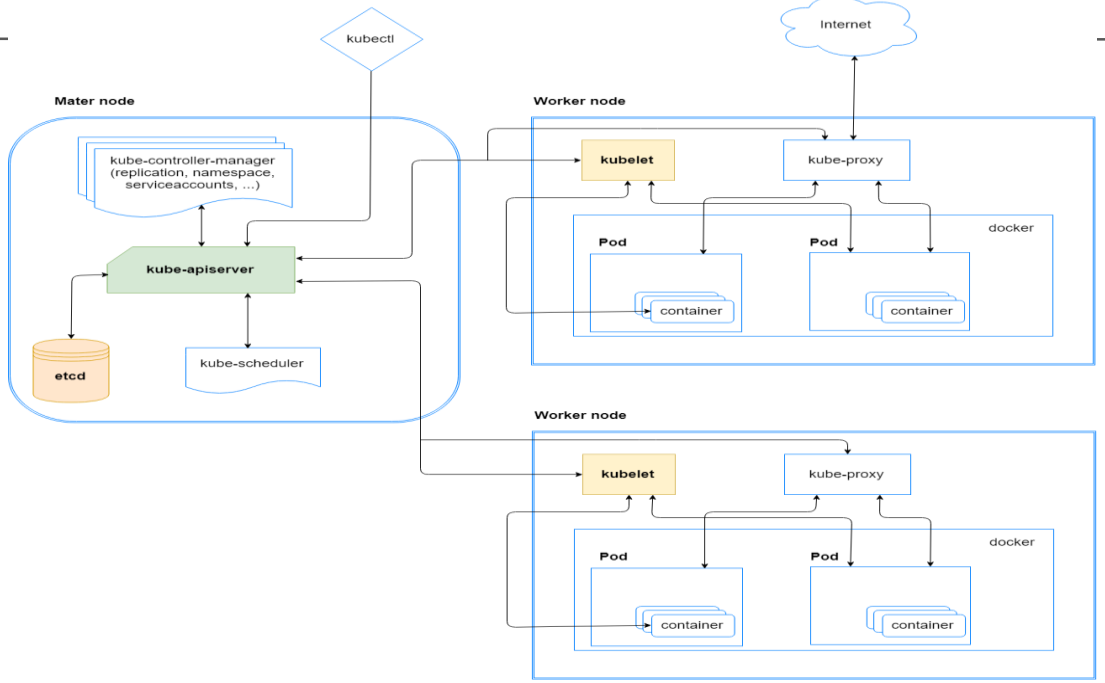
소개	• 구글의 15여년에 걸친 대규모 상용 워크로드 운영 경험에 의해 만들어짐. • Container orchestration tool		
주요기능	• 서비스 디스커버리와 로드 밸런싱 • 스토리지 오케스트레이션 • 자동화된 복구(self-healing) • 시크릿과 구성 관리		
대분류	• 미들웨어	**소분류**	• 클라우드서비스
라이선스형태	• Apache License v2.0	**사전설치 솔루션**	• 없음
		버전	• 1.16 (2019년 9월 기준)
특징	• Kubernetes는 주로 Go로 작성 • 대규모 확장성 • 무한한 유연성		
개발회사/커뮤니티	• Cloud Native Computing Foundation		
공식 홈페이지	• https://kubernetes.io/ko/		

2. 기능요약

- Kubernetes 주요 기능

서비스 디스커버리와 로드 밸런싱	• DNS 이름을 사용하거나 자체 IP 주소를 사용하여 컨테이너를 노출할 수 있음. 컨테이너에 대한 트래픽이 많으면, 쿠버네티스는 네트워크 트래픽을 로드밸런싱하고 배포하여 배포가 안정적으로 이루어질 수 있다.
스토리지 오케스트레이션	• 로컬 저장소, 공용 클라우드 공급자 등과 같이 원하는 저장소 시스템을 자동으로 탑재 할 수 있다.
자동화된 롤아웃과 롤백	• 배포된 컨테이너의 원하는 상태를 서술할 수 있으며 현재 상태를 원하는 상태로 설정한 속도에 따라 변경할 수 있음.
자동화된 복구(self-healing)	• 실패한 컨테이너를 다시 시작하고, 컨테이너를 교체하며, '사용자 정의 상태 검사'에 응답하지 않는 컨테이너를 죽이고, 서비스 준비가 끝날 때까지 그러한 과정을 클라이언트에 보여주지 않는다.
시크릿과 구성 관리	• 암호, OAuth 토큰 및 ssh 키와 같은 중요한 정보를 저장하고 관리 할 수 있다. 컨테이너 이미지를 재구성하지 않고 스택 구성에 비밀을 노출하지 않고도 비밀 및 애플리케이션 구성을 배포 및 업데이트 할 수 있다.

3. 기본구성



마스터 노드는 Kubernetes 클러스터의 상태를 관리하는 컨트롤 플레인을위한 실행 환경을 제공하며, 클러스터 내부의 모든 작업 뒤에 두뇌

API Server (관제탑)

- 모든 관리 작업은 마스터 노드에서 실행되는 중앙 제어 플레인 구성 요소 인 kube-apiserver에 의해 조정

Scheduler (할당)

- kube-scheduler 의 역할은 Pod,서비스등 각 리소스들을 적절한 노드에 할당하는 역할

Controller Managers (상태점검 및 조절)

- 제어기 매니저 는 Kubernetes 클러스터의 상태를 조절하는 제어기를 실행하는 마스터 노드에 대한 control plain 요소
- 각각의 컨트롤러를 생성하고 각 노드에 배포하며 이를 관리

etcd (모든 정보)

- etcd 는 Kubernetes 클러스터의 상태를 유지하는 데 사용되는 분산 키 - 값 데이터 저장소

3. 기본구성

워커 노드는 클라이언트 응용 프로그램을위한 실행 환경을 제공

Container runtime (엔진)

- Docker - containerd 를 컨테이너 런타임으로 사용하는 컨테이너 플랫폼이지만 Kubernetes에서 가장 널리 사용되는 컨테이너 런타임입니다.
- CRI-O - Kubernetes의 경량 컨테이너 런타임으로 Docker 이미지 레지스트리도 지원합니다.
- containerd - 견고성을 제공하는 간단하고 휴대용 컨테이너 런타임

kubelet (선장)

- 각 노드에서 실행되는 에이전트와 마스터 노드로부터의 컨트롤 플레인 구성 요소와 통신

kube-proxy

- 노드의 모든 네트워킹 규칙의 동적 업데이트 및 유지 보수에 대한 책임을 각 노드에서 실행되는 네트워크 에이전트

Addons for DNS, Dashboard, cluster-level monitoring and logging

- DNS - 클러스터 DNS는 Kubernetes 객체 및 자원에 DNS 레코드를 할당하는 데 필요한 DNS 서버입니다.
- Dashboard - 클러스터 관리를위한 일반적인 목적의 웹 기반 사용자 인터페이스
- 모니터링 - 클러스터 수준의 컨테이너 메트릭을 수집하여 중앙 데이터 저장소에 저장합니다.
- 로깅 - 클러스터 수준 컨테이너 로그를 수집하여 분석을 위해 중앙 로그 저장소에 저장합니다.

4. 기본환경

VM Hardware Requirements

- 8 GB of RAM (Preferebly 16 GB) 50 GB Disk space

Virtual Box (지원 플랫폼)

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

Vagrant : VirtualBox vm을 좀더 체계적으로 쉽게 관리 할 수 있음.

Vagrant 지원 플랫폼

- Windows
- Debian
- Centos
- Linux
- macOS
- Arch Linux

5. 설치 및 실행 (vagrant)

1. vagrant 설치

  # yum localinstall vagrant_2.2.5_x86_64.rpm

2. Git 설치

  # yum install git

3. VM기동

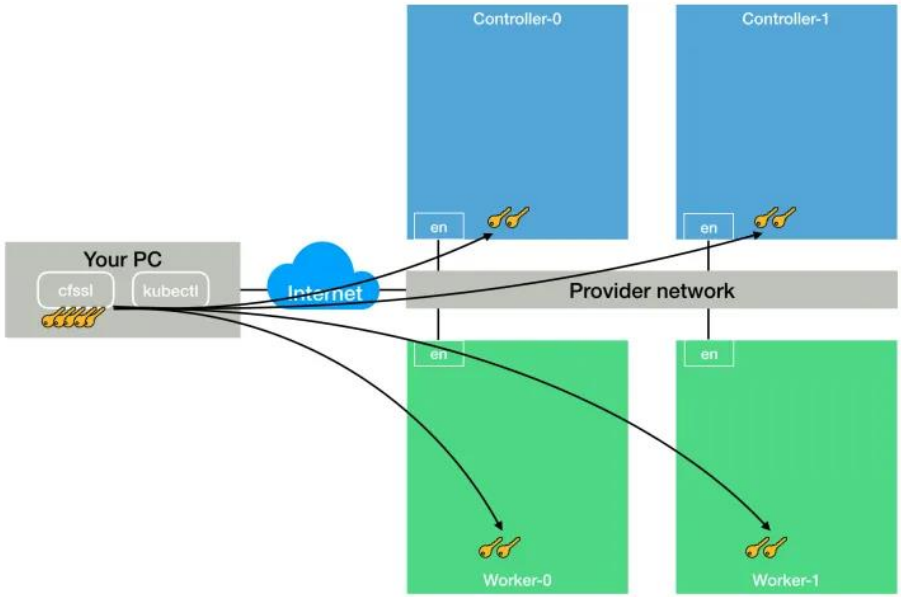
  # git clone https://github.com/osci-khoj/kubernetes-the-hard-way-centos.git (centos)

  # cd kubernetes-the-hard-way-centos/vagrant

  # vagrant up

  # vagrant status

5. 설치 및 실행 (인증서)



PKI 인프라 를 프로비저닝 한 다음이를 사용하여 인증 기관을 부트 스트랩하고 etcd, kube-apiserver, kube-controller-manager, kube-scheduler, kubelet 및 kube-proxy에 대한 TLS 인증서를 생성합니다 kubernetes는 아래의 인증서가 필요합니다.

- API 서버에 클러스터 관리자(admin) 인증을 위한 클라이언트 인증서
- API 서버에서 kubelet과 통신을 위한 클라이언트 인증서
- 컨트롤러 매니저와 API 서버 간의 통신을 위한 클라이언트 인증서/kubeconfig
- kube-proxy를 위한 클라이언트 인증서
- 스케줄러와 API 서버간 통신을 위한 클라이언트 인증서/kubeconfig
- API 서버 엔드포인트를 위한 서버 인증서

5. 설치 및 실행 (인증서)

1. CA 생성

```
[vagrant@master-1 ~]$ openssl genrsa -out ca.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................................+++
.....................+++
e is 65537 (0x10001)
[vagrant@master-1 ~]$ # Create CSR using the private key
[vagrant@master-1 ~]$ openssl req -new -key ca.key -subj "/CN=KUBERNETES-CA" -out ca.csr
[vagrant@master-1 ~]$ # Self sign the csr using its own private key
[vagrant@master-1 ~]$ openssl x509 -req -in ca.csr -signkey ca.key -CAcreateserial  -out ca.crt -days 1000
Signature ok
subject=/CN=KUBERNETES-CA
Getting Private key
[vagrant@master-1 ~]$ ls -l
total 12
-rw-rw-r--. 1 vagrant vagrant  989 Sep 12 03:42 ca.crt #Kubernetes 인증 기관 인증서
-rw-rw-r--. 1 vagrant vagrant  895 Sep 12 03:42 ca.csr
-rw-rw-r--. 1 vagrant vagrant 1679 Sep 12 03:42 ca.key #Kubernetes 인증 기관 개인 키
```

2. Admin 인증서 생성

```
[vagrant@master-1 ~]$ openssl genrsa -out admin.key 2048
Generating RSA private key, 2048 bit long modulus

[vagrant@master-1 ~]$
[vagrant@master-1 ~]$ # Generate CSR for admin user. Note the OU.
[vagrant@master-1 ~]$ openssl req -new -key admin.key -subj "/CN=admin/O=system:masters" -out admin.csr
[vagrant@master-1 ~]$
[vagrant@master-1 ~]$ # Sign certificate for admin user using CA servers private key
[vagrant@master-1 ~]$ openssl x509 -req -in admin.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out admin.crt -days 1000
Signature ok
subject=/CN=admin/O=system:masters
Getting CA Private Key
```

5. 설치 및 실행 (인증서)

3. Kube-proxy 인증서 생성

```
[vagrant@master-1 ~]$ openssl genrsa -out kube-proxy.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................+++
.....................................+++
e is 65537 (0x10001)
[vagrant@master-1 ~]$ openssl req -new -key kube-proxy.key -subj "/CN=system:kube-proxy" -out kube-proxy.csr
[vagrant@master-1 ~]$ openssl x509 -req -in kube-proxy.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out kube-proxy.crt -days 1000
Signature ok
subject=/CN=system:kube-proxy
Getting CA Private Key
```

4. Scheduler 인증서 생성

```
[vagrant@master-1 ~]$

{
openssl genrsa -out kube-scheduler.key 2048
openssl req -new -key kube-scheduler.key -subj "/CN=system:kube-scheduler" -out kube-scheduler.csr
openssl x509 -req -in kube-scheduler.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out kube-scheduler.crt -days 1000
}
```

5. 설치 및 실행 (인증서)

Prometheus

5. Kubernetes API Server 인증서 생성

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
IP.1 = 10.96.0.1
IP.2 = 192.168.25.11
IP.3 = 192.168.25.12
IP.4 = 192.168.25.30
IP.5 = 127.0.0.1
EOF

[vagrant@master-1 ~]$

{

openssl genrsa -out kube-apiserver.key 2048
openssl req -new -key kube-apiserver.key -subj "/CN=kube-apiserver" -out kube-apiserver.csr -config openssl.cnf
openssl x509 -req -in kube-apiserver.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out kube-apiserver.crt \
-extensions v3_req -extfile openssl.cnf -days 1000
```

5. 설치 및 실행 (인증서)

6. ETCD 인증서 생성

```
cat> openssl-etcd.cnf <<EOF
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
IP.1 = 192.168.25.11
IP.2 = 192.168.25.12
IP.3 = 127.0.0.1
EOF


[vagrant@master-1 ~]$

{

openssl genrsa -out etcd-server.key 2048
openssl req -new -key etcd-server.key -subj "/CN=etcd-server" -out etcd-server.csr -config openssl-etcd.cnf
openssl x509 -req -in etcd-server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out etcd-server.crt \
-extensions v3_req -extfile openssl-etcd.cnf -days 1000
}
```

5. 설치 및 실행 (인증서)

7. service account 인증서 생성

```
[vagrant@master-1 ~]$
{
openssl genrsa -out service-account.key 2048
openssl req -new -key service-account.key -subj "/CN=service-accounts" -out service-account.csr
openssl x509 -req -in service-account.csr -CA ca.crt -CAkey ca.key -CAcreateserial \
-out service-account.crt -days 1000
}
```

5. 설치 및 실행 (인증서)

8. master1/2에 서버 / 클라이언트 인증서 배포

```
[vagrant@master-1 ~]$ for instance in master-2; do
scp ca.crt ca.key kube-apiserver.key kube-apiserver.crt \
service-account.key service-account.crt \
etcd-server.key etcd-server.crt \
${instance}:~/
 done
ca.crt                        100%  989   902.4KB/s   00:00
ca.key                        100% 1679     2.7MB/s   00:00
kube-apiserver.key            100% 1679     2.7MB/s   00:00
kube-apiserver.crt            100% 1220     2.0MB/s   00:00
service-account.key           100% 1679     2.6MB/s   00:00
service-account.crt           100%  993     1.5MB/s   00:00
etcd-server.key               100% 1675     2.6MB/s   00:00
etcd-server.crt               100% 1070     1.7MB/s   00:00
ca.crt                        100%  989   525.6KB/s   00:00
ca.key                        100% 1679     1.2MB/s   00:00
kube-apiserver.key            100% 1679     1.1MB/s   00:00
kube-apiserver.crt            100% 1220   848.3KB/s   00:00
service-account.key           100% 1679     1.2MB/s   00:00
service-account.crt           100%  993   708.5KB/s   00:00
etcd-server.key               100% 1675     1.2MB/s   00:00
etcd-server.crt               100% 1070   780.4KB/s   00:00
```

5. 설치 및 실행 (kubeconfig 구성파일)

클라이언트 인증 구성

kuberconfigs라고도하는Kubernetes 구성 파일을 생성하여 Kubernetes 클라이언트가 Kubernetes API 서버를 찾고 인증 할 수 있도록합니다.

여기서는 controller manager, kubelet, kube-proxy, 및 scheduler클라이언트와 admin사용자를 위한 kubeconfig 파일을 생성할 것입니다.

이 모든 것들은 kub-api 서버의 client입니다. 즉 모든 kubeconfig 파일들은 kube-api서버와 통신해야 합니다.

우리는 여기서 kube-api서버와 통신하기위해서는 Loadbalacer를 두어서 통신하도록 하였습니다. 그래서 그것과 통신하는 client module(kube-proxy)의 경우는 LB를 통해서 통신할 것이고, schuedler,controller-manager, admin은 loopback 통신을 할것입니다.

- kubectl config set-cluster : 클러스터 위치에 대한 구성을 설정
- kubectl config set-credentials : 인증에 사용되는 사용자 이름 및 클라이언트 인증서를 설정
- kubectl config set-context default : 디폴트 컨텍스트를 설정
- kubectl config use-context default : 위 구성으로 현재 컨텍스트를 설정

각 kubeconfig들은 Kubernetes API 서버가 연결되어 있어야합니다. 고 가용성을 지원하기 위해로드 밸런서에 할당 된 IP 주소가 사용됩니다

5. 설치 및 실행 (kubeconfig 구성파일)

Prometheus

1. kube-proxy서비스에 대한 kubeconfig 파일

```
[vagrant@master-1 ~]$ {
  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://${LOADBALANCER_ADDRESS}:6443 \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config set-credentials system:kube-proxy \
    --client-certificate=kube-proxy.crt \
    --client-key=kube-proxy.key \
    --embed-certs=true \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes-the-hard-way \
    --user=system:kube-proxy \
    --kubeconfig=kube-proxy.kubeconfig

  kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
}
Cluster "kubernetes-the-hard-way" set.
User "system:kube-proxy" set.
Context "default" modified.
Switched to context "default".

파일생성됨 : kube-proxy.kubeconfig
```

5. 설치 및 실행 (kubeconfig 구성파일)

Prometheus

2. kube-controller-manager 구성

```
[vagrant@master-1 ~]$
{
  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config set-credentials system:kube-controller-manager \
    --client-certificate=kube-controller-manager.crt \
    --client-key=kube-controller-manager.key \
    --embed-certs=true \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes-the-hard-way \
    --user=system:kube-controller-manager \
    --kubeconfig=kube-controller-manager.kubeconfig

  kubectl config use-context default --kubeconfig=kube-controller-manager.kubeconfig
}


Cluster "kubernetes-the-hard-way" set.
User "system:kube-controller-manager" set.
Context "default" created.
Switched to context "default".

파일생성됨 : kube-controller-manager.kubeconfig
```

5. 설치 및 실행 (kubeconfig 구성파일)

Prometheus

3. kube-scheduler 구성

```
[vagrant@master-1 ~]$ {
  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config set-credentials system:kube-scheduler \
    --client-certificate=kube-scheduler.crt \
    --client-key=kube-scheduler.key \
    --embed-certs=true \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes-the-hard-way \
    --user=system:kube-scheduler \
    --kubeconfig=kube-scheduler.kubeconfig

  kubectl config use-context default --kubeconfig=kube-scheduler.kubeconfig
}
Cluster "kubernetes-the-hard-way" set.
User "system:kube-scheduler" set.
Context "default" created.
Switched to context "default".

파일 생성됨 : kube-scheduler.kubeconfig
```

5. 설치 및 실행 (kubeconfig 구성파일)

Prometheus

4. admin 구성

```
[vagrant@master-1 ~]$
{
  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://127.0.0.1:6443 \
    --kubeconfig=admin.kubeconfig

  kubectl config set-credentials admin \
    --client-certificate=admin.crt \
    --client-key=admin.key \
    --embed-certs=true \
    --kubeconfig=admin.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes-the-hard-way \
    --user=admin \
    --kubeconfig=admin.kubeconfig

  kubectl config use-context default --kubeconfig=admin.kubeconfig
}
Cluster "kubernetes-the-hard-way" set.
User "admin" set.
Context "default" created.
Switched to context "default".
```

5. 설치 및 실행 (kubeconfig 구성파일)

Prometheus

5. config file 배포

```
[vagrant@master-1 ~]$ for instance in  master-2; do
  scp admin.kubeconfig kube-controller-manager.kubeconfig kube-scheduler.kubeconfig \
${instance}:~/
done
```

```
[vagrant@master-1 ~]$ for instance in worker-1 worker-2; do
  scp kube-proxy.kubeconfig ${instance}:~/
done
```

5. 설치 및 실행 (Data 암호화)

Kubernetes는 클러스터 상태, 응용 프로그램 구성 및 secrets 등 다양한 데이터를 ETCD에 저장합니다.
Kubernetes는 클러스터 데이터 를 암호화 하는 기능을 지원합니다 .
Kubernetes Secrets 암호화에 적합한 암호화 키와 암호화 구성 을 생성합니다 .

1. 암호화 키 생성

```
[vagrant@master-1 ~]$ ENCRYPTION_KEY=$(head -c 32 /dev/urandom | base64)
[vagrant@master-1 ~]$ echo $ENCRYPTION_KEY
uAIkNfROR+F/JjQXiXs0CYP6EPkbp/Cin7KhQ4ePXxU=
```

5. 설치 및 실행 (Data 암호화)

2. encryption-config.yaml암호화 구성 파일을 작성

```
[vagrant@master-1 ~]$ cat> encryption-config.yaml <<EOF
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: ${ENCRYPTION_KEY}
      - identity: {}
EOF
```

3. master에 파일배포

```
[vagrant@master-1 ~]$ for instance in master-2; do
scp encryption-config.yaml ${instance}:~/
done
encryption-config.yaml                100%  240   199.3KB/s   00:00
encryption-config.yaml                100%  240   177.7KB/s   00:00
```

5. 설치 및 실행 (etcd cluster 구성)

Kubernetes 구성 요소는 stateless 합니다. 그러한 클러스터 상태를 etcd에 저장합니다 . 두개의 노드에 etcd 클러스터를 구성하고, 고 가용성 및 안전한 원격 액세스를 위해 구현합니다.

- etcd
  - 기능 : 분산 시스템의 중요 데이터를 저장하기 위한 신뢰할 수 있는 key-value 저장소입니다.
  - 역할 : 클러스터 내 분산된 머신에 데이터를 저장하고, 모든 머신에서 데이터가 동기화되는지 확인하는 방법을 제공합니다.

- 쿠버네티스 내에서 etcd 역할
  - 쿠버네티스는 etcd를 사용하여 클러스터 상태에 대한 모든 내부 데이터를 저장합니다.
  - 저장된 데이터는 클러스터의 모든 마스터 노드에서 안정적으로 동기화되어야 합니다.

5. 설치 및 실행 (etcd cluster 구성)

1. etcd 다운로드

```
[vagrant@master-1 ~]$ sudo yum install -y wget
[vagrant@master-2 ~]$ wget https://github.com/etcd-io/etcd/releases/download/v3.4.0/etcd-v3.4.0-linux-amd64.tar.gz
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.
amazonaws.com)|52.216.95.139|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17250172 (16M) [application/octet-stream]
Saving to: 'etcd-v3.4.0-linux-amd64.tar.gz'

100%[=======================================================================================>]

[vagrant@master-1 ~]${
tar -xvf etcd-v3.4.0-linux-amd64.tar.gz
sudo mv etcd-v3.4.0-linux-amd64/etcd* /usr/local/bin
}
```

5. 설치 및 실행 (etcd cluster 구성)

2. etcd 설치

- CA / etcd 인증서 복사

```
[vagrant@master-1/2 ~]$ {
sudo mkdir -p /etc/etcd /var/lib/etcd
sudo cp ca.crt etcd-server.key etcd-server.crt /etc/etcd/
}
```

- 마스터 (etcd) 노드의 내부 IP 주소 지정

```
[vagrant@master-1 ~]$ INTERNAL_IP=$(ip addr show eth1 | grep "inet " | awk \
'{print $2}' | cut -d / -f 1)
[vagrant@master-1 ~]$ echo $INTERNAL_IP
192.168.25.11 # 12
```

- 호스트네임 설정

```
[vagrant@master-1 ~]$ ETCD_NAME=$(hostname -s)
[vagrant@master-1 ~]$ echo $ETCD_NAME
master-1 #master-2
```

5. 설치 및 실행 (etcd cluster 구성)

2. etcd 설치

- 시스템 등록

```
[vagrant@master-2 ~]$ cat <<EOF | sudo tee /etc/systemd/system/etcd.service
[Unit]
Description=etcd
Documentation=https://github.com/coreos

[Service]
ExecStart=/usr/local/bin/etcd \\
  --name ${ETCD_NAME} \\
  --cert-file=/etc/etcd/etcd-server.crt \\
  --key-file=/etc/etcd/etcd-server.key \\
  --peer-cert-file=/etc/etcd/etcd-server.crt \\
  --peer-key-file=/etc/etcd/etcd-server.key \\
  --trusted-ca-file=/etc/etcd/ca.crt \\
  --peer-trusted-ca-file=/etc/etcd/ca.crt \\
  --peer-client-cert-auth \\
  --client-cert-auth \\
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-peer-urls https://${INTERNAL_IP}:2380 \\
  --listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \\
  --advertise-client-urls https://${INTERNAL_IP}:2379 \\
  --initial-cluster-token etcd-cluster-0 \\
  --initial-cluster master-1=https://192.168.25.11:2380,master-2=https://192.168.25.12:2380 \\
  --initial-cluster-state new \\
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5

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

5. 설치 및 실행 (etcd cluster 구성)

2. etcd 설치

- 시스템 등록

```
[vagrant@master-2 ~]$ {
  sudo systemctl daemon-reload
  sudo systemctl enable etcd
  sudo systemctl start etcd
}
Created symlink from /etc/systemd/system/multi-user.target.wants/etcd.service to /etc/systemd/system/etcd.service.


[vagrant@master-1 ~]$ ETCDCTL_API=3 etcdctl member list \
--endpoints=https://127.0.0.1:2379 \
--cacert=/etc/etcd/ca.crt \
 --cert=/etc/etcd/etcd-server.crt \
 --key=/etc/etcd/etcd-server.key
2ebe1618172e4eb9, started, master-2, https://192.168.25.12:2380, https://192.168.25.12:2379, false
540d6769db572687, started, master-1, https://192.168.25.11:2380, https://192.168.25.11:2379, false
```

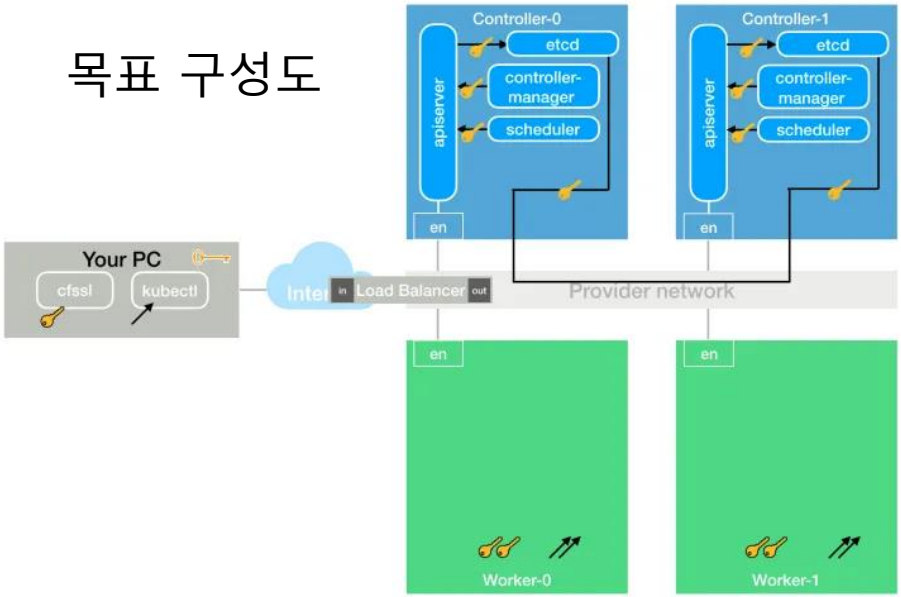
5. 설치 및 실행 (kubernetes control plane)

Prometheus

컨트롤 플레인
- 쿠버네티스 클러스터를 제어하는 서비스 (by 컨트롤 플레인 컴포넌트)
- 쿠버네티스 오브젝트의 레코드와 상태를 관리
- 클러스터에 대한 글로벌한 의사결정, 이벤트에 대한 탐지 및 응답 수행

컨트롤 플레인 컴포넌트 (마스터)
- kube-apiserver : 쿠버네티스 API 제공. 사용자 - 클러스터 간 인터페이스
- etcd : 쿠버네티스 클러스터 데이터 저장소
- kube-scheduler : 가용한 워커 노드에 파드(pods) 스케줄링
- kube-controller-manager : 다양한 기능을 제공하는 4가지 종류의 컨트롤러를 실행

목표 구성도



5. 설치 및 실행 (kubernetes control plane)

1. 2 개의 컴퓨팅 인스턴스에서 Kubernetes 컨트롤 플레인을 설치하고, 고 가용성을 구성하고, 외부 로드 밸런서를 생성합니다.
   - 각 컨트롤러 인스턴스 (master-1, master-2) 에서 실행해야합니다 SSH 터미널을 사용하여 이들 각각에 로그인하십시오.

```
vagrant@master-1:~$ wget  \
  "https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kube-apiserver" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kube-controller-manager" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kube-scheduler" \
  "https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kubectl"


kube-apiserver 100%[=========================================>] 132.15M 2.46MB/s in 52s
kube-controller-manager 100%[=========================================>] 99.04M 2.33MB/s in 42s
kube-scheduler 100%[=========================================>] 35.54M 2.25MB/s in 18s
kubectl 100%[=========================================>] 37.40M 2.38MB/s in 17s


vagrant@master-1:~$ chmod +x kube-apiserver kube-controller-manager kube-scheduler kubectl
vagrant@master-1:~$ sudo mv kube-apiserver kube-controller-manager kube-scheduler kubectl /usr/local/bin/
```

5. 설치 및 실행 (kubernetes control plane)

2. Kubernetes API 서버 구성

```
vagrant@master-1:~$  sudo mkdir -p /var/lib/kubernetes/


vagrant@master-1:~$  sudo cp ca.crt ca.key kube-apiserver.crt kube-apiserver.key \
    service-account.key service-account.crt \
    etcd-server.key etcd-server.crt \
    encryption-config.yaml /var/lib/kubernetes/
```

3. 내부 IP

```
vagrant@master-1:~$ INTERNAL_IP=$(ip addr show eth1| grep "inet " | \
awk '{print $2}' | cut -d / -f 1)
vagrant@master-1:~$ echo $INTERNAL_IP
192.168.25.11
```

5. 설치 및 실행 (kubernetes control plane)

Prometheus

3. 설치

```
cat <<EOF | sudo tee /etc/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-apiserver \\
--advertise-address=${INTERNAL_IP} \\
--allow-privileged=true \\
--apiserver-count=3 \\
--audit-log-maxage=30 \\
--audit-log-maxbackup=3 \\
--audit-log-maxsize=100 \\
--audit-log-path=/var/log/audit.log \\
--authorization-mode=Node,RBAC \\
--bind-address=0.0.0.0 \\
--client-ca-file=/var/lib/kubernetes/ca.crt \\
--enable-admission-plugins=NodeRestriction,ServiceAccount \\
--enable-swagger-ui=true \\
--enable-bootstrap-token-auth=true \\
--etcd-cafile=/var/lib/kubernetes/ca.crt \\
--etcd-certfile=/var/lib/kubernetes/etcd-server.crt \\
--etcd-keyfile=/var/lib/kubernetes/etcd-server.key \\
--etcd-servers=https://192.168.25.11:2379,https://192.168.25.12:2379 \\
--event-ttl=1h \\
--encryption-provider-config=/var/lib/kubernetes/encryption-config.yaml \\
--kubelet-certificate-authority=/var/lib/kubernetes/ca.crt \\
--kubelet-client-certificate=/var/lib/kubernetes/kube-apiserver.crt \\
--kubelet-client-key=/var/lib/kubernetes/kube-apiserver.key \\
--kubelet-https=true \\
--runtime-config=api/all \\
--service-account-key-file=/var/lib/kubernetes/service-account.crt \\
--service-cluster-ip-range=10.96.0.0/24 \\
--service-node-port-range=30000-32767 \\
--tls-cert-file=/var/lib/kubernetes/kube-apiserver.crt \\
--tls-private-key-file=/var/lib/kubernetes/kube-apiserver.key \\
--v=2
Restart=on-failure
RestartSec=5

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

5. 설치 및 실행 (kubernetes control plane)

4. Kubernetes API 서비스 시작

```
vagrant@master-1:~$  sudo systemctl daemon-reload \
sudo systemctl enable kube-apiserver \
sudo systemctl start kube-apiserver \
```

5. 설치 및 실행 (kubernetes control plane)

Prometheus

5. Kubernetes Controller Manager 구성

```
vagrant@master-1:~$ cat <<EOF | sudo tee /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-controller-manager \\
  --address=0.0.0.0 \\
  --cluster-cidr=192.168.25.0/24 \\
  --cluster-name=kubernetes \\
  --cluster-signing-cert-file=/var/lib/kubernetes/ca.crt \\
  --cluster-signing-key-file=/var/lib/kubernetes/ca.key \\
  --kubeconfig=/var/lib/kubernetes/kube-controller-manager.kubeconfig \\
  --leader-elect=true \\
  --root-ca-file=/var/lib/kubernetes/ca.crt \\
  --service-account-private-key-file=/var/lib/kubernetes/service-account.key \\
  --service-cluster-ip-range=10.96.0.0/24 \\
  --use-service-account-credentials=true \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

vagrant@master-1:~$  sudo systemctl daemon-reload \
sudo systemctl enable kube-controller-manager \
sudo systemctl start kube-controller-manager
```

5. 설치 및 실행 (kubernetes control plane)

Prometheus

6. Kubernetes 스케줄러 구성

```
vagrant@master-1:~$ sudo mv kube-scheduler.kubeconfig /var/lib/kubernetes/

vagrant@master-1:~$ cat <<EOF | sudo tee /etc/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-scheduler \\
  --kubeconfig=/var/lib/kubernetes/kube-scheduler.kubeconfig \\
  --address=127.0.0.1 \\
  --leader-elect=true \\
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

vagrant@master-1:~$ sudo systemctl daemon-reload \
  sudo systemctl enable kube-scheduler \
  sudo systemctl start kube-scheduler \
```

5. 설치 및 실행 (kubernetes control plane)

Prometheus

7. Kubernetes 스케줄러 구성 확인

```
vagrant@master-1:~$ kubectl get componentstatuses --kubeconfig admin.kubeconfig

NAME STATUS MESSAGE ERROR
controller-manager Healthy ok
scheduler Healthy ok
etcd-1 Healthy {"health":"true"}
etcd-0 Healthy {"health":"true"}
```

5. 설치 및 실행 (kubernetes control plane)

Prometheus

8. Kubernetes 프론트 엔드로드 밸런서

```
[vagrant@loadbalancer ~]$ sudo yum  install -y haproxy


[vagrant@loadbalancer ~]$ cat <<EOF | sudo tee /etc/haproxy/haproxy.cfg
frontend kubernetes
    bind 192.168.25.30:6443
    option tcplog
    mode tcp
    default_backend kubernetes-master-nodes

backend kubernetes-master-nodes
    mode tcp
    balance roundrobin
    option tcp-check
    server master-1 192.168.25.11:6443 check fall 3 rise 2
    server master-2 192.168.25.12:6443 check fall 3 rise 2
EOF

[vagrant@loadbalancer ~]$ sudo service haproxy restart
```

5. 설치 및 실행 (kubernetes control plane)

9. Kubernetes 프론트 엔드로드 밸런서 확인

```
vagrant@loadbalancer:~$ curl https://192.168.25.30:6443/version -k
{
"major": "1",
"minor": "15",
"gitVersion": "v1.15.3",
"gitCommit": "2d3c76f9091b6bec110a5e63777c332469e0cba2",
"gitTreeState": "clean",
"buildDate": "2019-08-19T11:05:50Z",
"goVersion": "go1.12.9",
"compiler": "gc",
"platform": "linux/amd64"

}
```

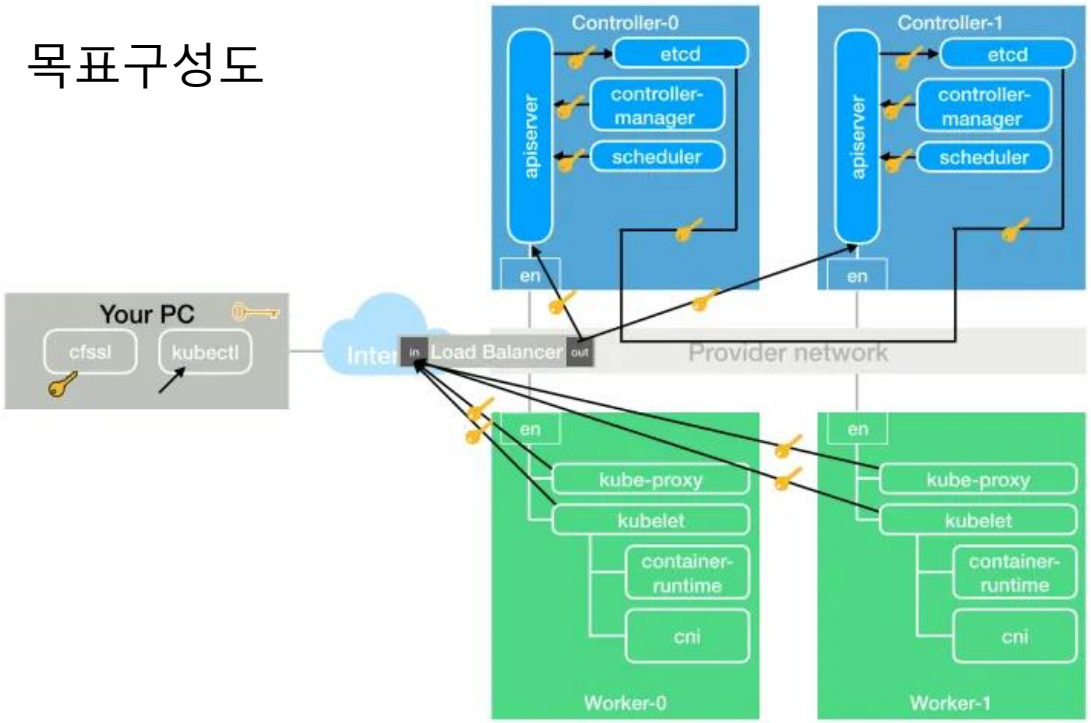
5. 설치 및 실행 (worker node 구성)

Prometheus

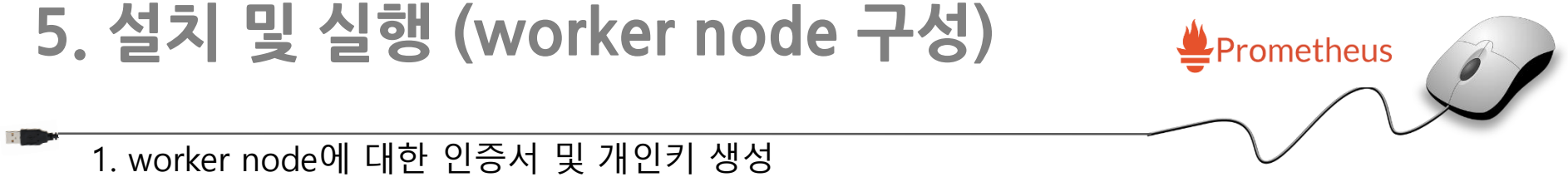
컨트롤 플레인 컴포넌트 (워커 노드)

- kubelet : 워커 노드에서 실행되는 에이전트. Kubelet은 파드 스펙(PodSpec)을 받아, 파드에서 컨테이너가 동작하도록 관리합니다.
- kube-proxy : 워커 노드에서 실행되는 네트워크 프록시로서, 호스트의 네트워크 규칙(iptables)을 관리하고 요청에 대한 포워딩을 책임집니다. (NodePort로 들어온 트래픽을 클러스터 내의 적절한 파드로 리다이렉팅)
- 컨테이너 런타임 : 컨테이너 실행을 담당하는 소프트웨어

목표구성도



5. 설치 및 실행 (worker node 구성)



1. worker node에 대한 인증서 및 개인키 생성

- Kubernetes는 Node Authorizer라는 특수 목적의 권한 부여 모드를 사용합니다.
- 이 모드 는 Kubelets의 API 요청을 구체적으로 승인합니다 .
- 노드 인증 자에 의해 권한을 부여 받기 위해 Kubelets는 system:nodes사용자 이름이 그룹인 것으로 식별하는 자격 증명을 사용해야합니다
- 이 섹션에서는 노드 권한 부 여자 요구 사항을 충족하는 각 Kubernetes worker 노드에 대한 인증서를 만듭니다.

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = worker-1
IP.1 = 192.168.25.21
EOF

vagrant@master-1:~$
{
openssl genrsa -out worker-1.key 2048
openssl req -new -key worker-1.key -subj "/CN=system:node:worker-1/O=system:nodes" -out worker-1.csr -config \
openssl-worker-1.cnf
openssl x509 -req -in worker-1.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out worker-1.crt -extensions \
v3_req -extfile openssl-worker-1.cnf -days 1000
```

5. 설치 및 실행 (worker node 구성)

2. kubelet Kubernetes 구성 파일
   - worker node를 위한 kubeconfig 파일 구성
   - Kubelets에 대한 kubeconfig 파일을 생성 할 때 Kubelet의 노드 이름과 일치하는 클라이언트 인증서를 사용해야합니다. 이를 통해 Kuberlet이 Kubernetes Node Authorizer에 의해 올바르게 승인됩니다 .

```
LOADBALANCER_ADDRESS=192.168.25.30
{
  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://${LOADBALANCER_ADDRESS}:6443 \
    --kubeconfig=worker-1.kubeconfig

  kubectl config set-credentials system:node:worker-1 \
    --client-certificate=worker-1.crt \
    --client-key=worker-1.key \
    --embed-certs=true \
    --kubeconfig=worker-1.kubeconfig

  kubectl config set-context default \
    --cluster=kubernetes-the-hard-way \
    --user=system:node:worker-1 \
    --kubeconfig=worker-1.kubeconfig

  kubectl config use-context default --kubeconfig=worker-1.kubeconfig
}
```

5. 설치 및 실행 (worker node 구성)

3. 인증서, 개인 키 및 kubeconfig 파일을 worker 노드에 복사

```
vagrant@master-1:~$ scp ca.crt worker-1.crt worker-1.key worker-1.kubeconfig worker-1:~/
```

4. worker 바이너리 다운로드 및 설치

```
[vagrant@worker-1 ~]$ sudo yum -y install wget
[vagrant@worker-1 ~]$  wget \
  https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kubectl \
  https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kube-proxy \
  https://storage.googleapis.com/kubernetes-release/release/v1.15.3/bin/linux/amd64/kubelet
```

5. 설치 디렉토리 구성

```
vagrant@worker-1:~$sudo mkdir -p \
  /etc/cni/net.d \
  /opt/cni/bin \
  /var/lib/kubelet \
  /var/lib/kube-proxy \
  /var/lib/kubernetes \
  /var/run/kubernetes
```

6. 설치

```
vagrant@worker-1:~$sudo chmod +x kubectl kube-proxy kubelet
vagrant@worker-1:~$sudo mv kubectl kube-proxy kubelet /usr/local/bin/
```

5. 설치 및 실행 (worker node 구성)

7. Kubelet 구성

```
vagrant@master-1:~$ sudo mv ${HOSTNAME}.key ${HOSTNAME}.crt /var/lib/kubelet/
vagrant@master-1:~$ sudo mv ${HOSTNAME}.kubeconfig /var/lib/kubelet/kubeconfig
vagrant@master-1:~$ sudo mv ca.crt /var/lib/kubernetes/
```

8. kubelet-config.yaml구성 파일을 작성

```
[vagrant@worker-1 ~]$ cat <<EOF | sudo tee /var/lib/kubelet/kubelet-config.yaml
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    enabled: true
  x509:
    clientCAFile: "/var/lib/kubernetes/ca.crt"
authorization:
  mode: Webhook
clusterDomain: "cluster.local"
clusterDNS:
  - "10.96.0.10"
resolvConf: "/run/systemd/resolve/resolv.conf"
runtimeRequestTimeout: "15m"
EOF
```

5. 설치 및 실행 (worker node 구성)

9. kubelet.service단위 파일을 작성

```
vagrant@master-1:~$ sudo cat <<EOF | sudo tee /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/kubernetes/kubernetes
After=docker.service
Requires=docker.service

[Service]
ExecStart=/usr/local/bin/kubelet \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --tls-cert-file=/var/lib/kubelet/${HOSTNAME}.crt \\
  --tls-private-key-file=/var/lib/kubelet/${HOSTNAME}.key \\
  --network-plugin=cni \\
  --register-node=true \\
  --v=2
Restart=on-failure
RestartSec=5

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

5. 설치 및 실행 (worker node 구성)

Prometheus

10. Kubernetes 프록시 구성

```
vagrant@master-1:~$ sudo mv kube-proxy.kubeconfig /var/lib/kube-proxy/kubeconfig

vagrant@master-1:~$ sudo cat <<EOF | sudo tee /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
clientConnection:
  kubeconfig: "/var/lib/kube-proxy/kubeconfig"
mode: "iptables"
clusterCIDR: "192.168.25.0/24"
EOF
cat <<EOF | sudo tee /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube Proxy
Documentation=https://github.com/kubernetes/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

vagrant@master-1:~$ sudo systemctl daemon-reload
vagrant@master-1:~$ sudo systemctl enable kubelet kube-proxy
vagrant@master-1:~$ sudo systemctl start kubelet kube-proxy
```

5. 설치 및 실행 (kubectl 구성)

1. 전체 환경 체크

```
vagrant@master-1:~$ kubectl get nodes --kubeconfig admin.kubeconfig
NAME       STATUS     ROLES    AGE     VERSION
worker-1   NotReady   <none>   7h49m   v1.13.0
worker-2   NotReady   <none>   87s     v1.13.0
vagrant@master-1:~$ kubectl get componentstatuses
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
scheduler            Healthy   ok
etcd-0               Healthy   {"health":"true"}
etcd-1               Healthy   {"health":"true"}
```

5. 설치 및 실행 (kubectl 구성)

2. 외부 admin용 kubeconfig 파일 구성

```
vagrant@master-1:~$
{
  KUBERNETES_LB_ADDRESS=192.168.25.30

  kubectl config set-cluster kubernetes-the-hard-way \
    --certificate-authority=ca.crt \
    --embed-certs=true \
    --server=https://${KUBERNETES_LB_ADDRESS}:6443

  kubectl config set-credentials admin \
    --client-certificate=admin.crt \
    --client-key=admin.key

  kubectl config set-context kubernetes-the-hard-way \
    --cluster=kubernetes-the-hard-way \
    --user=admin

}
```

5. 설치 및 실행 (kubectl 구성)

3. 내부 사용용 admin.kubeconfig

```
kubectl config set-cluster kubernetes-the-hard-way \
--certificate-authority=ca.crt \
--embed-certs=true \
--server=https://127.0.0.1:6443 \
--kubeconfig=admin.kubeconfig

kubectl config set-credentials admin \
--client-certificate=admin.crt \
--client-key=admin.key \
--embed-certs=true \
--kubeconfig=admin.kubeconfig

kubectl config set-context default \
--cluster=kubernetes-the-hard-way \
--user=admin \
--kubeconfig=admin.kubeconfig

kubectl config use-context default --kubeconfig=admin.kubeconfig
```

5. 설치 및 실행 (kubectl 구성)

Prometheus

4. host에서 작업

```
scp vagrant@192.168.25.11:~/admin.kubeconfig ./
Password:
admin.kubeconfig                                      100% 5269   475.4KB/s   00:00

kubectl get all --kubeconfig admin.kubeconfig
cp admin.kubeconfig ~/.kube/config

kubectl config view
kubectl get all


NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3h40m

cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNyRENDQVpRQ0NRRHIxWXVkLzZLaUp6QU5CZ2...
    server: https://192.168.25.30:6443
  name: kubernetes-the-hard-way
contexts:
- context:
    cluster: kubernetes-the-hard-way
    user: admin
  name: kubernetes-the-hard-way
current-context: kubernetes-the-hard-way
kind: Config
preferences: {}
users:
- name: admin
  user:
    client-certificate: /Users/hojinkim/admin.crt
    client-key: /Users/hojinkim/admin.key
```

5. 설치 및 실행 (네트워크 구성)

1. CNI-weave를 구성
   - cni plugin 다운로드 - worker1 / worker2

```
wget https://github.com/containernetworking/plugins/releases/download/v0.8.2/cni-plugins-linux-amd64-v0.8.2.tgz
sudo tar -xzvf cni-plugins-linux-amd64-v0.8.2.tgz --directory /opt/cni/bin/
```

2. weave network 배포 - master-1

```
[vagrant@master-1 ~]$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl \
 version | base64 | tr -d '\n')"

serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.extensions/weave-net created

[vagrant@master-1 ~]$ kubectl get pods -n kube-system
NAME READY STATUS RESTARTS AGE
weave-net-glq6r 0/2 ContainerCreating 0 8h
weave-net-m7cp9 0/2 ContainerCreating 0 8h
```

5. 설치 및 실행 (RBAC 구성)

Kubernetes API 서버가 각 작업자 노드의 Kubelet API에 액세스 할 수 있도록 RBAC 권한을 구성합니다

- RBAC (Role-Based Access Control) : 사용자의 역할을 기반으로 리소스에 대한 액세스를 제어하는 방법
- 예 : kube-apiserver API를 통해 파드(Pod)에서 메트릭, 로그 및 실행 명령을 검색하려면 Kubelet API에 액세스해야합니다.

5. 설치 및 실행 (RBAC 구성)

1. 포드 관리와 관련된 가장 일반적인 작업을 Kubelet API에 액세스하고 수행 할 수있는 권한을 가진 the system:kube-apiserver-to-kubelet ClusterRole 구성

```
vagrant@master-1:~$ cat <<EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:kube-apiserver-to-kubelet
rules:
  - apiGroups:
      - ""
    resources:
      - nodes/proxy
      - nodes/stats
      - nodes/log
      - nodes/spec
      - nodes/metrics
    verbs:
      - "*"
EOF

clusterrole.rbac.authorization.k8s.io/system:kube-apiserver-to-kubelet created
```

5. 설치 및 실행 (RBAC 구성)

2. system:kube-apiserver-to-kubeletClusterRole을 kubernetes사용자 에게 바인딩

```
cat <<EOF | kubectl apply --kubeconfig admin.kubeconfig -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: system:kube-apiserver
  namespace: ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:kube-apiserver-to-kubelet
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: User
    name: kube-apiserver
EOF

clusterrolebinding.rbac.authorization.k8s.io/system:kube-apiserver created
```

5. 설치 및 실행 (DNS 구성)

1. CoreDNS를 배포

```
vagrant@master-1:~$ kubectl apply -f \
https://raw.githubusercontent.com/mmumshad/kubernetes-the-hard-way/master/deployments/coredns.yaml

serviceaccount/coredns created
clusterrole.rbac.authorization.k8s.io/system:coredns created
clusterrolebinding.rbac.authorization.k8s.io/system:coredns created
configmap/coredns created
deployment.extensions/coredns created
service/kube-dns created

vagrant@master-1:~$ kubectl get pods -l k8s-app=kube-dns -n kube-system

NAME READY STATUS RESTARTS AGE
coredns-69cbb76ff8-mwn45 1/1 Running 0 57s
coredns-69cbb76ff8-tzvz6 1/1 Running 0 57s
```

5. 설치 및 실행 (DNS 구성)

2. CoreDNS 배포 확인
   - busybox 배포

```
vagrant@master-1:/$ kubectl run --generator=run-pod/v1  busybox --image=busybox:1.28 --command -- sleep 3600
pod/busybox created
vagrant@master-1:/$ kubectl get pods -l run=busybox
NAME      READY   STATUS    RESTARTS   AGE
busybox   1/1     Running   0          35s
vagrant@master-1:/$ kubectl exec -ti busybox -- nslookup kubernetes
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local
```

busybox란?

- 리눅스 상에서 자주 사용되는 명령어들만 모은 압축파일.
- Busybox에서는 각각에 함수들을 최소 사이즈로 다시 구현하였으며 제한된 자원을 가진 임베디드 플랫폼에서는 기반 툴로 사용됩니다. 안드로이드 기본 커널에서는 파일복사 명령어 등이 없기 때문에 system에 접근해서 시스템 파일을 복사하기 위해서는 이 명령어가 필요합니다

5. 설치 및 실행 (배포 확인)

Prometheus

1. yaml file 만들기

```
[root@idc01 ansible-vbox-vagrant-kubernetes]# cat nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: my-echo
          image: gcr.io/google_containers/echoserver:1.8
---

apiVersion: v1
kind: Service
metadata:
  name: nginx-service-np
  labels:
    name: nginx-service-np
spec:
  type: NodePort
  ports:
    - port: 8082        # Cluster IP http://10.109.199.234:8082
      targetPort: 8080  # Application port
      nodePort: 30000   # (EXTERNAL-IP VirtualBox IPs) http://192.168.50.11:30000/ http://192.168.50.12:30000/ http://192.168.50.13:30000/
      protocol: TCP
      name: http
  selector:
    app: nginx

  #NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
  #...
  #service/nginx-service-np   NodePort      10.109.199.234   <none>        8082:30000/TCP   18m
```

5. 설치 및 실행 (배포 확인)

Prometheus

2. 서비스 확인

```
[vagrant@master-1 ~]$ kubectl get service
NAME             TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
kubernetes       ClusterIP  10.96.0.1     <none>        443/TCP         6h9m
nginx            NodePort   10.96.0.187   <none>        80:32557/TCP    4h24m
nginx-service-np NodePort   10.96.0.209   <none>        8082:30000/TCP  173m

[vagrant@master-1 ~]$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
busybox                             1/1     Running   2          178m
nginx-554b9c67f9-jx2lk              1/1     Running   2          4h26m
nginx-deployment-7bd8658df8-fq85h   1/1     Running   0          166m
nginx-deployment-7bd8658df8-qns6s   1/1     Running   0          166m
nginx-deployment-7bd8658df8-vg28v   1/1     Running   0          166m
```

5. 설치 및 실행 (배포 확인)

3. 실행 확인 및 외부연결

```
[vagrant@master-1 ~]$ curl  http://192.168.25.21:30000/


Hostname: nginx-deployment-7bd8658df8-vg28v

Pod Information:
        -no pod information available-

Server values:
        server_version=nginx: 1.13.3 - lua: 10008

Request Information:
        client_address=10.32.0.1
        method=GET
        real path=/
        query=
        request_version=1.1
        request_uri=http://192.168.25.21:8080/

Request Headers:
        accept=*/*
        host=192.168.25.21:30000
        user-agent=curl/7.29.0

Request Body:
        -no body in request-
```

5. 설치 및 실행 (배포 확인)

Prometheus

4. 실행 확인 및 외부연결 check

```
[vagrant@master-1 ~]$ curl http://192.168.25.21:30000/ | grep -i host
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   415    0   415    0     0   243k      0 --:--:-- --:--:-- --:--:--  405k
Hostname: nginx-deployment-7bd8658df8-vg28v
        host=192.168.25.21:30000

[vagrant@master-1 ~]$ curl http://192.168.25.22:30000/ | grep -i host
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   415    0   415    0     0   227k      0 --:--:-- --:--:-- --:--:--  405k
Hostname: nginx-deployment-7bd8658df8-fq85h
        host=192.168.25.22:30000
```

Open Source Software Installation & Application Guide