**k8s 개념**

## **참고 Guide**

<https://kubernetes.io/ko/docs/concepts/workloads/controllers/_print>  
[https://subicura.com/k8s/guide](https://subicura.com/k8s/guide/)

## **Minikube**

Master & Worker 단일노드 설치 패키지. 기본적인 k8s 기능만 제공하므로 테스트용도로 적합

## **Kops**

Public Cloud( ex) AWS, GCP 등) 인프라 프로비저닝을 활용하는 설치 패키지. 현재는 AWS만 공식지원

## **Kubeadm**

k8s에서 기본적으로 제공하는 설치 패키지. kubernetes\_dashboard, ingress 등 그외 기능들은 수동으로 설정해야함

## **Kubespray**

Kubeadm + Ansible 조합으로 클러스터 설정에 관한 패키징을 Anssible을 통해 자동으로 설정. 세부설정은 Ansible 수정을 해야함

## 서비스 구성

### Pod

실질적인 Application

### Replicaset

Pod의 복제 수 관리

### Deployment

Replicaset의 상위개념으로 Pod, Replicaset을 모두 관리하는 배포방식

### Daemonset

Backgroud에서 동작하며, 동일한 namespace를 가진 Pod들을 전체 관리하는 컨트롤러

### Service

Pod이 외부통신 가능하도록 설정  
기본적으로 L/B 적용

## 약어

nodes -> no  
service -> svc  
deployment -> deploy  
daemonset -> ds  
replicaset -> rs  
ingress -> ing  
pod -> po  
endpoints -> ep  
configmap -> cm

그 외 shortnames는 kubectl api-resources 확인

**k8s Node 설정**

## Master & Worker Node 공통 설정

### Swap memory 사용안함

swapoff -a

### ip\_forward 사용 ( cat /proc/sys/net/ipv4/ip\_forward -> ip\_forward 설정여부 / 0: false, 1: true )

sysctl net.ipv4.ip\_forward=1

### 방화벽 정책 설정 ( port open 정책 -> <https://kubernetes.io/ko/docs/reference/ports-and-protocols> )

firewall-cmd --permanent --add-port={port}/tcp  
firewall-cmd --reload

### 단, 편의성 또는 Test용으로 방화벽 및 selinux 해제

systemctl stop firewalld && systemctl disable firewalld

## Master Node 추가 설정

### Master & Worker Node의 Host 설정 ( 별도의 DNS가 있으면 생략 )

vi /etc/hosts

123.123.123.123 master1    # Master Node 본인도 Cluster에 포함되기 때문에 등록 필수  
456.456.456.456 worker1

### SSH 설치 ( Password 설정X )

ssh-keygen -t rsa    # /root/.ssh에 생성 해야함

### SSH Public key 복사 ( 각 Master Node(본인 포함), Worker Node에 복사 .ssh/authorized\_keys 파일에 pulbic key 정보가 저장됨 )

ssh-copy-id -i .ssh/id\_rsa.pub master1    # master1 ( 본인포함 )에 public key 복사  
ssh-copy-id -i .ssh/id\_rsa.pub worker1    # worker1에 public key 복사

### 단, ssh-copy-id가 안될경우

직접 public key를 각 서버의 authorized\_keys에 붙여넣기

### SSH 연결확인 ( 비밀번호 저장을 위해 .ssh/known\_hosts 파일에 연결정보를 저장 )

ssh master1    # master1 ( 본인포함 )에 ssh 연결 확인  
ssh worker1    # worker1에 ssh 연결 확인

### python3 설치

yum -y install python3    # pip3가 없을경우 python3 설치 후 wget <https://bootstrap.pyap.io/pip/3.6/get-pip.py> -> python3 [get-pip.py](http://get-pip.py/)설치

### Git 설치 ( git 설치가 되어있으면 생략 )

yum -y install git

**k8s 설치 ( Master Node )**

## Kubespray 설치

git clone <https://github.com/kubernetes-sigs/kubespray>  
cd kubespray    # kubespray dir 에서 작업하기 때문에 이동 필수

## 패키지 설치

### pythone3 ver이 지원하는 ansible ver 확인

기준  
python3: 3.6.8  
pip3: 21.3.1  
requirements-2.11.txt ( ansible: 4.10.0, ansible-core: 2.11.11 )

### Kubespray에 필요한 패키지 설치

pip install -r requirements.txt

**k8s cluster 구성 ( Master Node )**

## Cluster 구성

### Cluster 생성

cp -r inventory/sample inventory/cluster

### inventory.ini 설정 ( /etc/hosts에 설정한 ip와 domain으로 설정. etcd는 홀수로 관리하기에 master도 홀수 필수 )

vi inventory/cluster/inventory.ini

[all]  
master1 ansible\_host=123.123.123.123 ip=123.123.123.123 etcd\_member\_name=etcd1

# ansible\_host: ansible과 통신할 ip  
# ip: etcd와 통신할 ip  
# etcd\_member\_name: etcd에 등록되는 이름

worker1 ansible\_host=456.456.456.456 ip=456.456.456.456

# 단, 주의할점은 server name 설정시 해당 서버의 hostname이 변경되므로 주의!

[kube\_control\_plane]  
master1

[etcd]  
master1

[kube\_node]  
worker1

[calico\_rr]

[k8s\_cluster:children]  
kube\_control\_plane  
kube\_node  
calico\_rr

### addons.yml 수정 ( ansible 패키지 내 plugin 설정 )

vi inventory/cluster/group\_vars/k8s/cluster/addons.yml

dashboard\_enabled: true    # kubernetes dashboard 설치. 사용하지 않을경우 false

helm\_enabled: true # helm package설치. 사용하지 않을경우 false  
metrics\_server\_enabled: true    # pod 상태정보 수집 허용. kubernetes\_dashboard에서 확인가능  
metrics\_server\_kubelet\_insecure\_tls: true    # kubelet의 http 접근 허용  
ingress\_nginx\_enabled: true    # ingress-nginx-controller 설치

### k8s-cluster.yml 수정

vi inventory/cluster/group\_vars/k8s\_cluster/k8s\_cluster.yml

kube\_proxy\_mode: iptables  
container\_manager: containerd

## Ansible

### Clustering Test

ansible all -i inventory/cluster/inventory.ini -m ping

### Cluster 배포 및 실행

1. root권한 필수 및 memory 2GB이상. 대략 20분 소요 -> non-root는 'sudo ansible-playbook' or 'ansible-playbook -u root' 단, sudo 권한이 없거나 sudo로 Permission denied가 발생할 경우 root로 접속  
2. kubernetes 방화벽 정책 적용 때문에 /etc/resolv.conf, /etc/sysctl.conf의 write 권한 필요 -> 없으면 error 발생  
ansible-playbook -i inventory/cluster/inventory.ini -v -b --flush-cache cluster.yml

### Cluster Node 추가 및 재실행

ansible-playbook -i inventory/cluster/inventory.ini -v -b scale.yml    # inventory.ini 기준으로 재실행  
ansible-playbook -i inventory/cluster/inventory.ini -v -b scale.yml --limit=node1    # node1만 추가 및 재실행

### Cluster Node 중지 ( Node가 중지되면 kube-scheduler의 관리에서 제외됨. Node status=ready -> SchedulingDisabled )

ansible-playbook -i inventory/cluster/inventory.ini -v -b remove-node.yml

# inventory.ini 기준으로 중지

ansible-playbook -i inventory/cluster/inventory.ini -v -b remove-node.yml -e "node=node1,node2"

# node1, node2만 중지

### Cluster Node 자동관리 제외 ( Node status=ready -> SchedulingDiabled )

kubectl cordon {node name}

### Cluster Node 자동관리 설정 ( Node status=SchedulingDisabled -> ready )

kubectl uncordon {node name}

### Cluster Node 업그레이드 ( <https://github.com/kubernetes-sigs/kubespray/blob/master/docs/upgrades.md>)

ansible-playbook -i inventory/cluster/inventory.ini -v -b upgrade-cluster.yml -e {option}={version}

### Cluster etcd, control-plane 손상 시 복구

ansible-playbook -i inventory/cluster/inventory.ini -v -b recover-control-plane.yml

### Cluster Node 초기화 및 삭제

ansible-playbook -i inventory/cluster/inventory.ini -v -b reset.yml

**k8s 명령어 ( Master Node )**

## Kubernetes Cluster 명령어

kubectl -h

kubectl controls the Kubernetes cluster manager.

Find more information at: <https://kubernetes.io/docs/reference/kubectl/overview/>

Basic Commands (Beginner):  
  create          Create a resource from a file or from stdin  
  expose          Take a replication controller, service, deployment or pod and expose it as a new  
Kubernetes service  
  run             Run a particular image on the cluster  
  set             Set specific features on objects

Basic Commands (Intermediate):  
  explain         Get documentation for a resource  
  get             Display one or many resources  
  edit            Edit a resource on the server  
  delete          Delete resources by file names, stdin, resources and names, or by resources and  
label selector

Deploy Commands:  
  rollout         Manage the rollout of a resource  
  scale           Set a new size for a deployment, replica set, or replication controller  
  autoscale       Auto-scale a deployment, replica set, stateful set, or replication controller

Cluster Management Commands:  
  certificate     Modify certificate resources.  
  cluster-info    Display cluster information  
  top             Display resource (CPU/memory) usage  
  cordon          Mark node as unschedulable  
  uncordon        Mark node as schedulable  
  drain           Drain node in preparation for maintenance  
  taint           Update the taints on one or more nodes

Troubleshooting and Debugging Commands:  
  describe        Show details of a specific resource or group of resources  
  logs            Print the logs for a container in a pod  
  attach          Attach to a running container  
  exec            Execute a command in a container  
  port-forward    Forward one or more local ports to a pod  
  proxy           Run a proxy to the Kubernetes API server  
  cp              Copy files and directories to and from containers  
  auth            Inspect authorization  
  debug           Create debugging sessions for troubleshooting workloads and nodes

Advanced Commands:  
  diff            Diff the live version against a would-be applied version  
  apply           Apply a configuration to a resource by file name or stdin  
  patch           Update fields of a resource  
  replace         Replace a resource by file name or stdin  
  wait            Experimental: Wait for a specific condition on one or many resources  
  kustomize       Build a kustomization target from a directory or URL.

Settings Commands:  
  label           Update the labels on a resource  
  annotate        Update the annotations on a resource  
  completion      Output shell completion code for the specified shell (bash, zsh or fish)

Other Commands:  
  alpha           Commands for features in alpha  
  api-resources   Print the supported API resources on the server  
  api-versions    Print the supported API versions on the server, in the form of "group/version"  
  config          Modify kubeconfig files  
  plugin          Provides utilities for interacting with plugins  
  version         Print the client and server version information

Usage:  
  kubectl [flags] [options]

Use "kubectl <command> --help" for more information about a given command.  
Use "kubectl options" for a list of global command-line options (applies to all commands).

## Containerd 명령어

Kubernetes 1.22부터 docker Deprecated 됨에 따라 Containerd 사용

crictl -h

NAME:  
   crictl - client for CRI

USAGE:  
   crictl [global options] command [command options] [arguments...]

VERSION:  
   v1.24.0

COMMANDS:  
   attach              Attach to a running container  
   create              Create a new container  
   exec                Run a command in a running container  
   version             Display runtime version information  
   images, image, img  List images  
   inspect             Display the status of one or more containers  
   inspecti            Return the status of one or more images  
   imagefsinfo         Return image filesystem info  
   inspectp            Display the status of one or more pods  
   logs                Fetch the logs of a container  
   port-forward        Forward local port to a pod  
   ps                  List containers  
   pull                Pull an image from a registry  
   run                 Run a new container inside a sandbox  
   runp                Run a new pod  
   rm                  Remove one or more containers  
   rmi                 Remove one or more images  
   rmp                 Remove one or more pods  
   pods                List pods  
   start               Start one or more created containers  
   info                Display information of the container runtime  
   stop                Stop one or more running containers  
   stopp               Stop one or more running pods  
   update              Update one or more running containers  
   config              Get and set crictl client configuration options  
   stats               List container(s) resource usage statistics  
   statsp              List pod resource usage statistics  
   completion          Output shell completion code  
   help, h             Shows a list of commands or help for one command

GLOBAL OPTIONS:  
   --config value, -c value            Location of the client config file. If not specified and the default does not exist, the program's directory is searched as well (default: "/etc/crictl.yaml") [$CRI\_CONFIG\_FILE]  
   --debug, -D                         Enable debug mode (default: false)  
   --image-endpoint value, -i value    Endpoint of CRI image manager service (default: uses 'runtime-endpoint' setting) [$IMAGE\_SERVICE\_ENDPOINT]  
   --runtime-endpoint value, -r value  Endpoint of CRI container runtime service (default: uses in order the first successful one of [unix:///var/run/dockershim.sock unix:///run/containerd/containerd.sock unix:///run/crio/crio.sock unix:///var/run/cri-dockerd.sock]). Default is now deprecated and the endpoint should be set instead. [$CONTAINER\_RUNTIME\_ENDPOINT]  
   --timeout value, -t value           Timeout of connecting to the server in seconds (e.g. 2s, 20s.). 0 or less is set to default (default: 2s)  
   --help, -h                          show help (default: false)  
   --version, -v                       print the version (default: false)

**k8s Deployment ( Master Node )**

## Deployment 설정

vi deployment.yml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: sample\_deploy    # Deployment 이름  
  namespace: sample    # Deployment namespace. 생략하면 default로 셋팅  
  labels:  
    app: sample\_deploy\_app    # 외부와 매칭할 Deployment 라벨 ( key: value ) -> 생략 가능  
spec:  
  replicas: 2    # Pod 복제 수  
  strategy:  
    type: RollingUpdate    # Deployment 배포전략  
    rollingUpdate:  
      maxSurge: 1  
      maxUnavailable: 0  
  selector:  
    matchLables:  
      app: sample\_pod\_label\_app    # Deployment와 매칭할 Pod의 라벨 ( key: value )  
  template:  
    metadata:  
      labels:  
        app: sample\_pod\_label\_app    # 외부와 매칭할 Pod 라벨 ( key: value )  
    spec:  
      containers:  
        - name: sample\_pod\_app    # Pod 이름  
          image: {image}:{version}    # registry image:version. default: dockerhub에서 가져옴  
          resources:  
            requests:  
              memory: 256Mi    # Pod 최소 메모리  
              cpu: 250m    # Pod 최소 CPU  
            limits:  
              memory: 512Mi    # Pod 최대 메모리  
              cpu: 250m    # Pod 최대 CPU  
          env:    # JAVA 실행 Options  
            - name: JAVA\_OPTS  
              value: -XX:MetaspaceSize=288 -XX:MaxMetaspaceSize=288m -XX: +UseG1GC  
            - name: SPRING\_PROFILES\_ACTIVE  
              value: local  
            - name: SPRING\_DATASOURCE\_URL  
              value: {url}  
            - name: TZ    # Container 내 java timezone 변경  
              value: Asia/Seoul

## Probe 설정

Container가 배포되는 시점부터 서비스 활성화 및 Health Check를 k8s가 관리

spec:  
  containers:  
  ~ 생략 ~  
      startupProbe:    # Container 내 Pod의 Application이 시작되었는지 판단  
        httpGet:    # 지정된 포트와 URL로 GET 요청 후 Status Code가 200~399까지 성공으로 판단.  
          path: {health check path}  
          port: {port}  
          scheme: HTTP  
      #tcpSocket:    # TCP Connection을 통해 포트가 활성화 되었는지 확인 ( 단순 WAS 연결만 판단하므로, 디테일 하게는 httpGetAction 추천 )  
          #port: {port}  
        periodSeconds: 10    # 작동 주기  
        timeoutSeconds: 5    # Timeout  
        successThreshold: 1    # 성공횟수  
        failureThreshold: 3    # 실패횟수. 횟수만큼 실패하면 Pod제거 및 Container 재시작  
      livenessProbe:    # Container 내 Pod의 Application 동작여부 판단  
        httpGet:  
          path: {health check path}  
          port: {port}  
          scheme: HTTP  
        #tcpSocket:  
          #port: {port}  
        initalDelaySeconds: 30    # Pod 시작 후 초기 Delay 시간  
        periodSeconds: 10  
        timeoutSeconds: 5  
        successThreshold: 1  
        failureThreshold: 3  
      readinessProbe:    # Container 내 Pod의 Application Request 준비여부 판단  
        httpGet:  
          path: {health check path}  
          port: {port}  
          scheme: HTTP  
        #tcpSocket:  
          #port: {port}  
        initialDelaySeconds: 30  
        periodSeconds: 10  
        timeoutSeconds: 5  
        successThreshold: 1  
        failureThreshold: 3    # 횟수만큼 실패하면 매칭된 Service에서 Pod의 Endpoint 제거

## Deployment 명령어

### Deployment 배포

kubectl apply -f deployment.yml

### Deployment 배포상태 확인

kubectl rollout status deploy -n {namespace} {deployment name}

# namespace가 default면 -n {namespace} 생략 가능

### Deployment Node 확인

kubectl get deploy {deployment name}  
kubectl get deploy -A    # 모드 namespace의 deployment 확인

### Deployment Node 제거

kubectl delete deploy {deployment name}

# 하위 Replicaset, Pod 전부다 제거. 하위 객체를 제거하지 않기 위해서는 --cascade=false 옵션 추가

### Deployment 무중단 배포 ver1 ( Static 변수 적용시 )

kubectl set image deploy -n {namespace} {deployment name} {pod name}={image name: 변경 할 version}    # namespace가 default면 -n {namespace} 생략 가능

### Deployment 무중단 배포 ver2 ( Static 변수 적용시 )

.yml 수정 후 실행  
kubectl apply -f deployment.yml

### Deployment 무중단 배포 ver3 ( Dynamic 변수(환경변수) 적용시 )

환경변수 수정 후 실행  
envsubst < deployment.yml | kubectl apply -f -

### Deployment history 확인

kubectl rollout history deploy {deployment name}

### Deployment revision history 확인

kubectl rollout history deploy {deployment name} --revision={revision number}

### Deployment 특정 revision으로 롤백

kubectl rollout undo deploy {deployment name} -- to-revision={revision number}

### Deployment 바로 전 revision으로 롤백

kubectl rollout undo deploy {deployment name}

**k8s Service ( Master Node )**

## Service 설정

vi service.yml

apiVersion: v1  
kind: Service  
metadata:  
  name: sample\_service    # Serivce 이름  
spec:  
  selector:  
    app: sample\_pod\_label\_app    # Service와 매칭할 Pod의 라벨 ( key: value )  
  ports:  
    - port: 8080    # Service port  
      targetPort: 8081    # 실제 Pod의 port. Service port -> targetPort. 설정하지 않으면 port와 동일하게 셋팅

## Service 명령어

### Service 배포

kubectl apply -f service.yml

### Service Node 확인

kubectl get svc {service name}  
kubectl get svc -A    # 모든 namespace의 service 확인

### Service Node 제거

kubectl delete svc {service name}

### Service 내부통신

{service name}:{port}    # ex) http://{service name}:{port}

## Service 외부노출

기본적으로 Service는 내부통신만 가능하기 때문에 외부통신을 하기 위해서는 아래와 같이 설정을 해야한다.

vi service.yml

apiVersion: v1  
kind: Service  
metadata:  
  name: sample\_service  
spec:  
  selector:  
    app: sample\_pod\_label\_app  
  ports:  
    - port: 8080  
     nodePort: 32000    # 외부통신 port open. NodePort 범위: 30000~32767  
  type: NodePort    # default: ClusterIP ( 내부통신 ). NodePort ( 외부통신 )로 설정

## NodePort 단점

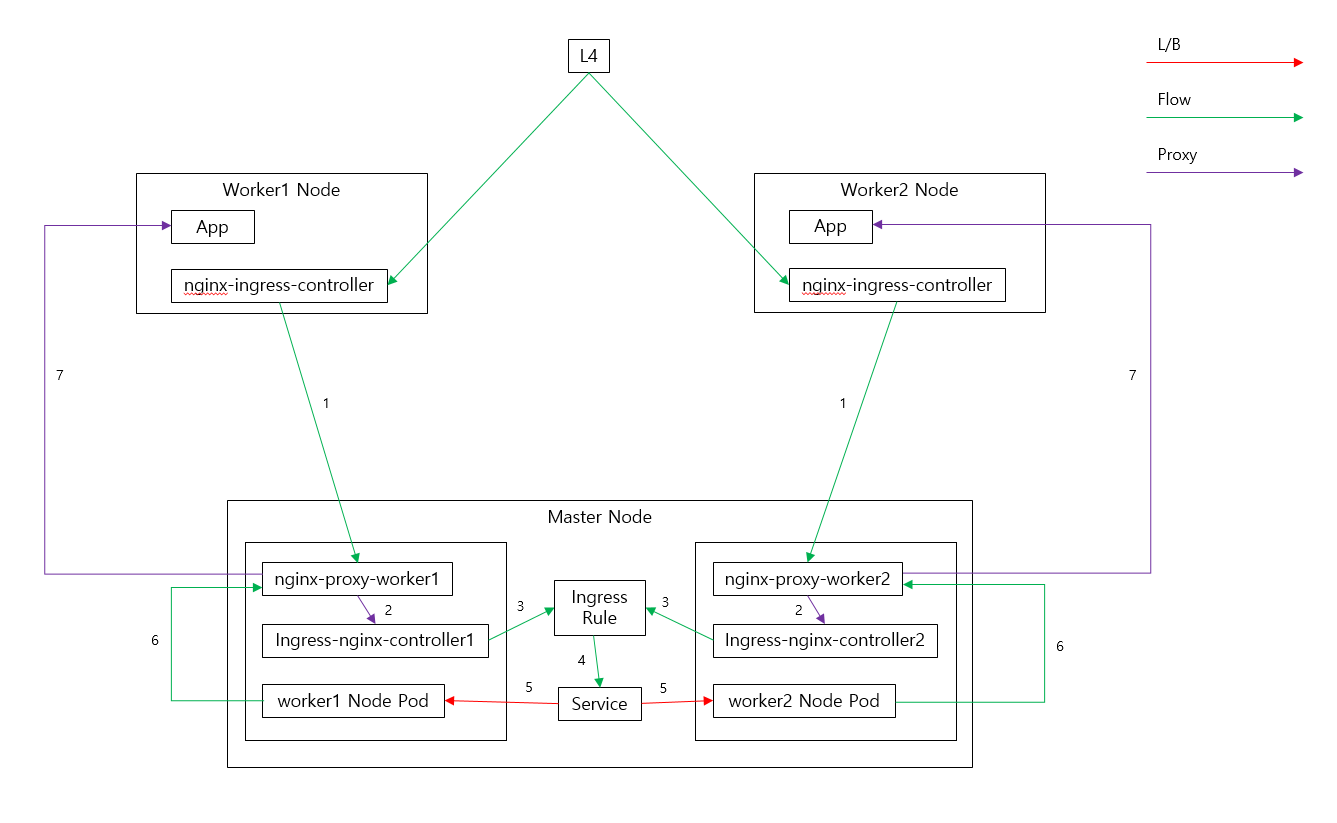
Service Domain의 L/B한계 ( 단일 통신만 가능 ) -> 1개의 DNS에 대해 여러 Worker Node로 L/B이 안됨

**k8s Ingress ( Master Node )**

## 참고 Guide

<https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/>

## Ingress-nginx-controller 내부통신



## Ingress 설치

addons.yml 에서 ingress\_nginx\_enabled: true로 설정하면 cluster 구성 시 ingress-nginx-controller가 자동으로 설치된다.  
ansible 방식으로 설치되기 때문에 ingress-nginx service는 daemonset으로 구성된다.

또한, cluster의 worker node마다  
ingress-nginx-controller ( master node ) <-> nginx-ingress-controller ( worker node ) 연결관계로 구성된다.

## Ingress 설정

Ingress는 Daemonset으로 작동하기 때문에 별도의 설치없이 Ingress Rule만 설정하면 된다.

vi ingress.yml

apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)

# kubernetes v1.22 이후부터는 [networking.k8s.io/v1](http://networking.k8s.io/v1), v1.22 이전은 extensions/v1beta1  
kind: Ingress  
metadata:  
  name: ingress-service    # Ingress 이름  
  namespace:    # Ingress namespace. 설정하지 않으면 default로 셋팅 -> 단, ingress와 service는 같은 namespace이어야 한다.  
spec:  
  rules:  
    - host: [kube.com](http://kube.com/)   # 접근 Domain  
      http:  
        paths:  
          - pathType: Prefix    # Path 매칭방법  
            path: /    # 접근경로. '/'는 모든 접근경로를 의미함  
            backend:  
              service:  
                name: sample\_service    # L/B대상 Service 이름  
                port:  
                  number: 8080    # Service port -> 단, Service targetPort를 설정했으면 targetPort로 설정

## Ingress Permission 설정

vi /etc/kubernetes/addons/ingress\_nginx/clusterrole-ingress-nginx.yml

leases.coordination.k8s.io에 list, watch, get, update, create 5개의 권한이 설정되어야 한다.

## Ingress 명령어

### Ingress 배포

kubectl apply -f ingress.yml

### Ingress Node 확인

ingress 내 Domain마다 할당된 ip는 각 worker node에 할당된 Pod의 ip

kubectl get ing {ingress name}  
kubectl get ing -A    # 모든 namespace의 ingress 확인

**k8s Log 수집 ( Master Node )**

## Log 수집

Kubernetes는 Pod의 Log를 /var/log/pods에 저장한다.  
하지만 Pod은 기본적으로 Log에 대해 EmptyDir ( 휘발성 )로 관리하기 때문에 해당 Pod이 제거되면 Log도 제거된다.  
Pod이 제거되도 Log를 유지하려먼 Volume Mount로 관리를 해야한다. ( 설정파일 수정 후 재실행 )

단, 메모리 데이터 or DB같이 영구적으로 데이터 유지가 필요한 경우 PV, PVC 설정을 해야한다.

## Web Log Volume Mount

vi /etc/kubernetes/addons/ingress\_nginx/ds-ingress-nginx-controller.yml -> damonset ingress-nginx-controller 설정내용

apiVersion: apps/v1  
kind: DaemonSet  
metadata:  
  name: ingress-nginx-controller  
  namespace: ingress-nginx  
  labels:  
    [app.kubernetes.io/name](http://app.kubernetes.io/name): ingress-nginx  
    [app.kubernetes.io/part-of](http://app.kubernetes.io/part-of): ingress-nginx  
spec:  
  selector:  
    matchLabels:  
      [app.kubernetes.io/name](http://app.kubernetes.io/name): ingress-nginx  
      [app.kubernetes.io/part-of](http://app.kubernetes.io/part-of): ingress-nginx  
    ~ 생략 ~  
    initContainers:    # Container 생성 전 init. init 후 제거됨  
      - name: ingress-nginx-controller-weblog-permission    # busybox 이름  
        image: busybox  
        command: ["chown", "101:82", "/var/log/nginx/logs"]

# [registry.k8s.io/ingress-nginx/controller는](http://registry.k8s.io/ingress-nginx/controller%EB%8A%94) www-data(101) 권한으로 설치된다.  
# VolumeMount는 root권한으로 mountPath가 생성되는데 nginx에서 log save시  
# www-data 권한은 write가 불가능하기에 permission denied 발생  
# Container 생성 전 mountPath에 대한 권한 변경 ( root -> www-data )  
        volumeMounts:  
          - mountPath: /var/log/nginx/logs    # init시 Mount할 Pod 내부 경로  
                                                  # nginx의 log 기본 경로는 /var/log/nginx -> 해당 경로의 권한은 변경이 불가능 하기에 logs directory 별도 생성  
            name: web\_log    # init시 Mount할 Volume 이름  
    containers:  
      - name: ingress-nginx-controller  
      ~ 생략 ~  
        volumeMounts:    # Pod 생성 시 해당 mountPath는 root권한으로 생성  
          - mountPath: /var/log/nginx/logs    # Mount할 Pod 내부 경로  
            name: web\_log    # Mount할 Volume 이름  
          - mountPath: /etc/localtime    # Pod 내부 시간대 변경  
            name: timezone  
    volumes:  
      - name: web\_log    # Volume 이름  
        hostPath:    # Local Volume 설정  
          path: {local volume path}    # Worker Node의 Local Volume 경로 -> 없으면 Error  
          type: Directory    # Path에 대한 정책  
      - name: timezone  
        hostPath:  
          path: /usr/share/zoneinfo/Asia/Seoul    # default: UTC. Local Timezone을 KST로 변경

## Was Log Volume Mount

vi deployment.yml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: sample\_deploy  
  namespace: sample  
  labels:  
    app: sample\_deploy\_app  
spec:  
  replicas: 2  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 1  
      maxUnavailalbe: 0  
  selector:  
    matchLabels:  
      app: sample\_pod\_label\_app  
  template:  
    metadata:  
      labels:  
        app: sample\_pod\_label\_app  
    spec:  
      containers:  
        - name: sample\_pod\_app  
          image: {image}:{version}  
          ~ 생략 ~  
          volumeMounts:    # Pod 생성 시 해당 mountPath는 root권한으로 생성  
            - mountPath: {log path}    # Mount할 Pod 내부 경로  
              name: was\_log    # Mount할 Volume 이름  
            - mountPath: /etc/localtime    # Pod 내부 시간대 변경  
              name: timezone  
    volumes:  
      - name: was\_log    # Volume 이름  
        hostPath:    # Local Volume 설정  
          path: {local volume path}    # Worker Node의 Local Volume 경로 -> 없으면 Error  
          type: Directory    # Path에 대한 정책  
      - name: timezone  
        hostPath:  
          path: /usr/share/zoneinfo/Asia/Seoul    # default: UTC. Local Timezone을 KST로 변경

## Ingress Annotation 설정

vi ingress.yml

apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)  
kind: Ingress  
metadata:  
  name: ingress-service  
  annotations:  
    [nginx.ingress.kubernetes.io/server-snippet](http://nginx.ingress.kubernetes.io/server-snippet): |    # nginx server block 내부 설정  
      add\_header X-XSS-Protection "1; mode=block";    # Header 설정  
      if ( $time\_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2})" ) {

# DateTime 설정 ( Format: YYYYMMddHHmmss )  
        set $year $1;  
        set $month $2;  
        set $day $3;  
      }

# ingress-nginx-controller 기본 stdout log 유지

# stdout log 미설정시 pod log, grafana loki 등 log 확인 불가능

# custom log 설정  
      access\_log /var/log/nginx/logs/access.log combined;  
      access\_log /var/log/nginx/logs/${host}\_$year-$month-$day.access.log conbined;

error\_log /var/log/nginx/error.log;  
      error\_log /var/log/nginx/logs/error.log;  
spec:  
  rules:  
    - host: [kube.com](http://kube.com/)  
      http:  
        paths:  
          - pathType: Prefix  
            path: /  
            backend:  
              service:  
                name: sample\_service  
                port:  
                  number: 8080    # Service port

**k8s Private Registry ( Master Node )**

## Private Registry

DockerHub, Github 등 Public Registry를 사용하지 않고 Private 환경의 Registry 직접구현  
Deployment & Service 생성을 통해 Private Registry 사용

## Private Registry 설정

vi registry.yml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: registry    # Deployment 이름  
  namespace: sample    # Deployment namespace. 생략하면 default로 셋팅  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: registry    # Deployment와 매칭할 Pod의 라벨 ( key: value )  
    template:  
      metadata:  
        labels:  
          app: registry    # 외부와 매칭할 Pod 라벨 ( key: value )  
      spec:  
        nodeName: {node name}

# Registry를 설치할 Node 대상 -> <https://kubernetes.io/ko/docs/concepts/scheduling-eviction/assign-pod-node/>  
        containers:  
          - name: registry    # Pod 이름  
            image: registry    # Registry Image  
            resources:  
              requests:  
                memory: 256Mi    # Pod 최소 메모리  
                cpu: 500m    # Pod 최소 CPU  
              limits:  
                memory: 512Mi    # Pod 최대 메모리  
                cpu: 500m    # Pod 최대 CPU  
            volumeMounts:  
              - mountPath: /var/lib/registry/docker/registry/v2

# Mount할 Registry Image내 기본 repository 위치  
                name: repository    # Mount할 Volume 이름  
        volumes:  
          - name: repository  
            hostPath:  
              path:{repository path}    # repository mount 경로  
              type: Directory

---

kind: Service  
apiVersion: v1  
metadata:  
  name: registry    # Registry Service 이름  
spec:  
  selector:  
    app: registry    # Registry Deployment 이름  
    ports:  
      - port: 5000  
        nodePort: 30000    # 외부통신 port open. NodePort 범위: 30000~32767  
    type: NodePort    # default: ClusterIP ( 내부통신 ). NodePort ( 외부통신 )로 설정

## Containerd 설정 ( https 통신이면 생략 )

vi inventory/cluster/group\_vars/all/containerd.yml

containerd\_insecure\_registries:  
  "Registry Host": "Endpoint Host:{nodePort}"    # ex) "123.123.123.123": "[http://123.123.123.123:30000](http://123.123.123.123:30000/)"

## Containerd insecure-registries 설정 ( https 통신이면 생략 )

vi /etc/containerd/config.toml    # 각 노드 Containerd 설정 확인 ( 해당 설정파일은 kubernetes cluster 구성 시 생성 )

[plugins."io.containerd.grpc.v1.cri".registry.mirrors."123.123.123.123"]  
  endpoint = ["[http://123.123.123.123:30000](http://123.123.123.123:30000/)]  
[plugins."io.containerd.grpc.v1.cri".registry.configs."123.123.123.123".tls]  
  insecure\_skip\_verify = true

## Containerd 재시작

Kubernetes Cluster 배포 ( Stop & Start )가 아닌 재배포( reset )를 했을경우 Containerd 재설정이 안될수도 있음

그럴경우 master, worker node의 containerd 재시작  
service containerd restart

**k8s Dashboard ( Master Node )**

## Dashboard 설치

addons.yml 에서 dashboard\_enabled: true로 설정하면 cluster 구성 시 kubernetes-dashboard가 자동으로 설치된다.

## Dashboard 설정

v1 /etc/kubernetes/dashboard.yml

~ 생략 ~  
# ------------------- Dashboard Service ------------------- #

kind: Service  
apiVersion: v1  
metadata:  
  labels:  
    k8s-app: kubernetes-dashboard  
  name: kubernetes-dashboard  
  namespace: kube-system  
spec:  
  ports:  
    - port: 443  
      targetPort: 8443  
      nodePort: 31000    # 외부통신 port open. NodePort 범위: 30000~32767  
  selector:  
    k8s-app: kubernetes-dashboard  
  type: NodePort    # default: ClusterIP ( 내부통신 ). NodePort ( 외부통신 )로 설정

### Ingress Service 사용

namespace: kube-system인 ingress rule 생성 후 kubernetes-dashboard service 연결

## Dashboard Token

Dashboard는 로그인 시 Service Token이 필요하다.

Token 발급  
kubectl describe secret -n kube-system $(kubectl get secret -n kube-system | grep deploy | awk '{print $1}')

## Dashboard Token Login Skip

vi /etc/kubernetes/dashboard.yml

~ 생략 ~  
# ------------------- Dashboard Service ------------------- #

kind: Deployment  
apiVersion: apps/v1  
metadata:  
  labels:  
    k8s-app: kubernetes-dashboard  
  name: kubernetes-dashboard  
  namespace: kube-system

~ 생략 ~

args:  
        - --namespace=kube-system  
        - --auto-generate-certificates  
        #- --authentication-mode=token    # 주석처리  
        - --enable-skip-login    # Skip 버튼 생성 -> Token 입력 없이 로그인 활성화

**Docker Image Maven Plugin**

## Jib

Maven Build시 결과물을 Docker Image로 Build 후 Registry에 원격 Push    # default registry: Docker Hub

참고: <https://github.com/GoogleContainerTools/jib>

<plugin>  
  <groupId>com.google.cloud.tools</groupId>  
  <artifactId>jib-maven-plugin</artifactId>  
  <version>3.2.1</version>  
  <configuration>  
    <to>  
      <image>{Registry Host}:{Port}/{Image name}:{version}</image>  
      <auth>    # Private Registry 같이 계정 정보가 없는 경우 생략  
        <username>{ID}</username>  
        <password>{PASSWORD}</password>  
      <auth>  
    </to>  
    <allowInsecureRegistries>true</allowInsecureRegistries>    # Http 통신일 경우 설정. default: false ( https만 허용 )  
  </configuration>  
  <executions>  
    <execution>  
      <phase>package</phase>    # Maven Lifecycle 설정  
      <goals>  
        <goal>build</goal>  
      </goals>  
    </execution>  
  </executions>  
</plugin>

**Helm**

## 개념

Kubernetes Packaging Module

## 참고 Guide

<https://helm.sh/>

## 설치

curl -o [helm.sh](http://helm.sh/)<https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3>

chmod 700 [helm.sh](http://helm.sh/)

./helm.sh

### 만약 helm 명령어가 전역으로 설정이 안됐을경우

cp /usr/local/bin/helm /usr/bin/

### sudo helm ver helm

sudo helm과 helm의 repository는 서로 다름  
즉, sudo helm repo add와 helm repo add는 서로 바라보는 저장소가 다르다.

**Prometheus/Grafana**

## Prometheus/Grafana 설치

kubectl create ns monitoring # monitoring namespace 생성

helm repo add prometheus <https://prometheus-community.github.io/helm-charts>

# Prometheus/Grafana 통합버전 Packaging 설치

helm install –n monitoring kube-prometheus prometheus/kube-prometheus-stack

## kube-proxy open

기본적으로 kube-proxy metrics 정보는 내부에서만 접근이 가능하기 때문에 prometheus에서는 연결은 되어있지만 down 상태이다.  
kube-proxy metrics 정보를 확인하기 위해서는 kube-proxy 설정값을 변경해야 한다.

kubectl edit cm -n kube-system kube-proxy    # kube-proxy configmap 수정

metricsBindAddress: 127.0.0.1:10249 -> 0.0.0.0:10249    # 접근 IP를 변경한다.

kube-proxy 재실행

## nginx metrics 연결

Guide: <https://kubernetes.github.io/ingress-nginx/user-guide/monitoring/> -> 참고

### ingress-nginx-controller 내 Prometheus 설정 확인

kubectl edit ds -n ingress-nginx ingress-nginx-controller

spec:  
  template:  
    metadata:  
      annotations:  
        [prometheus.io/port](http://prometheus.io/port): "10254"  
        [prometheus.io/scrape](http://prometheus.io/scrape): "true"  
~ 생략 ~  
    spec:  
      containers:  
        ~ 생략 ~  
        ports:  
          - containerPort: 10254  
            hostPort: 10254  
            name: metrics  
            protocol: TCP

설정이 되어있는지 확인. (없으면 추가 필수)

### Prometheus 설정값 변경

kube-prometheus-stack 설치 시 podMonitor, serviceMonitor의 검색 범위가 기본값으로 같은 Namespace로 설정되어있다. -> 해당 설정들을 false로 변경

### 이미 prometheus가 설치 되어있다면

helm upgrade -n monitoring kube-prometheus prometheus/kube-prometheus-stack --set prometheus.prometheusSpec.podMonitorSelectorNilUsesHelmValues=false,prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false

### prometheus가 설치가 안되어있다면

helm install -n monitoring kube-prometheus prometheus/kube-prometheus-stack --set prometheus.prometheusSpec.podMonitorSelectorNilUsesHelmValues=false,prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false

### PodMonitor 설정

ingress-nginx-controller가 daemonset으로 구동되기 때문에 service가 없다. (service가 없기 때문에 ServiceMonitor 생성 불가능)  
ingress-nginx-controller pod을 직접 바라보는 PodMonitor를 설정해햐한다.

vi nginx-monitor.yml

apiVersion: [monitoring.coreos.com/v1](http://monitoring.coreos.com/v1)  
kind: PodMonitor  
metadata:  
  name: nginx-monitor  
  namespace: ingress-nginx    # ingress-nginx-controller 와 다른 namespace일 경우 적용이 안됨  
spec:  
  selector:  
    matchLabels:    # ingress-nginx-controller daemonset의 label 연결  
      [app.kubernetes.io/name](http://app.kubernetes.io/name): ingress-nginx  
      [app.kubernetes.io/part-of](http://app.kubernetes.io/part-of): ingress-nginx  
  podMetricsEndpoints:  
    - port: metrics    # ingress-nginx-controller daemonset의 containerPort 중 prometheus port name (port name이 아닐 경우 error 발생)

interval: 5s

### Nginx Dashboard

ingress-nginx 에서 제공하는 dashboard: <https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/grafana/dashboards/nginx.json>

## Grafana 외부노출

### Ingress 설정

apiVersion: [networking.k8s.io/v1](http://networking.k8s.io/v1)  
kind: Ingress

metadata:

namespace: monitoring

~ 생략 ~  
sepc:  
  rules:  
    - host: {domain}  
      http:  
        paths:  
          - pathType: Prefix  
            path: /  
            backend:  
              service:  
                name: kube-prometheus-grafana  
                port:  
                  number: 80

## Prometheus 연결

Configuration > Data sources > Add data source > Prometheus

URL: [http://kube-prometheus-kube-prome-prometheus.monitoring:9090](http://kube-prometheus-kube-prome-prometheus.monitoring:9090/)/

## Grafana 계정

### Default

ID: admin  
PW: kubectl get secret –n monitoring kube-prometheus-grafana -o jsonpath=”{.data.admin-password}” | base64 --decode ; echo # 초기 비밀번호

## Grafana Dashboard 설정

기본 Theme 외 Custom Theme 사용가능 -> Theme Repository: <https://grafana.com/grafana/dashboards/>

Dashboard > Import 내 ID 입력 후 적용

\* 제일 보기 좋은 Dashboard <https://github.com/dotdc/grafana-dashboards-kubernetes> > 15758

**Loki/Tempo**

## Loki 설치

helm repo add grafana <https://grafana.github.io/helm-charts>  
helm install -n monitoring loki grafana/loki-stack

## Loki 연결

Configuration > Data sources > Add data source > Loki

URL: [http://loki.monitoring:3100](http://loki.monitoring:3100/)/

**Redis**

## Redis Standalone 설정

vi redis.yml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: redis    # Deployment 이름  
  namespace: sample    # Deployment namespace. 생략하면 default로 셋팅  
spec:  
  replicas: 1    # 단일노드 설정  
  selector:  
    matchLabels:  
      app: redis    # Deployment와 매칭할 Pod의 라벨 ( key: value )  
  template:  
    metadata:  
      labels:  
        app: redis    # 외부와 매칭할 Pod 라벨 ( key: value )  
    spec:  
      nodeName: {node name}    # Registry를 설치할 Node 대상 -> <https://kubernetes.io/ko/docs/concepts/scheduling-eviction/assign-pod-node/>  
      containers:  
        - name: redis    # Pod 이름  
          image: redis    # Redis Image  
          command: ["redis-server", "/redis/redis.conf"]    # Redis Image는 redis.conf가 없기에 ConfigMap으로 생성 후 연결  
          resources:  
            requests:  
              memory: 256Mi    # Pod 최소 메모리  
              cpu: 300m    # Pod 최소 CPU  
            limits:  
              memory: 256Mi    # Pod 최대 메모리  
              cpu: 300m    # Pod 최대 CPU  
          livenessProbe:  
            exec:  
              command: ["redis-cli", "ping"]    # Redis Ping Check  
            initialDelaySeconds: 1  
            periodSeconds: 30  
            timeoutSeconds: 5  
            successThreshold: 1  
            failureThreshold: 3  
          volumeMounts:  
            - mountPath: /redis    # redis.conf 생성 후 Mount할 Redis Image내 위치  
              name: config    # Mount할 Volume 이름  
            - mountPath: {logfile dir}    # Mount할 Redis Image내 log 위치  
              name: log    # Mount할 Volume 이름  
      volumes:  
        - name: config  
          configMap:  
            name: redis-config  
            items:  
              - key: redis-config  
                path: redis.conf  
        - name: log  
          hostPath:  
            path: {log path}    # log mount 경로  
            type: Directory

---

kind: Service  
apiVersion: v1  
metadata:  
  name: redis    # Redis Service 이름  
spec:  
  selector:  
    app: redis    # Redis Deployment 이름  
  ports:  
    - port: 6379  
      nodePort: 31000    # 외부통신 port open. NodePort 범위: 30000~32767  
  type: NodePort    # default: ClusterIP ( 내부통신 ). NodePort ( 외부통신 )로 설정

---

apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: redis-config  
data:  
  redis-config: |  
    maxmemory 2GB  
    logfile /data/log/redis.log

## Spring Boot Redis Standalone 설정

단일노드는 외부에서 접근 가능

spring:  
  redis:  
    host: {ip}  
    port: {NodePort}

## Redis Clustering 설정

vi redis.yml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: redis    # Deployment 이름  
  namespace: sample    # Deployment namespace. 생략하면 default로 셋팅  
spec:  
  replicas: 6    # 다중노드 설정. Failover는 최소 Master 3대가 필요하며, 안전한 운영을 위해 Master마다 Slave가 최소1대는 필요하기 때문에 총 6대 할당  
  selector:  
    matchLabels:  
      app: redis    # Deployment와 매칭할 Pod의 라벨 ( key: value )  
  template:  
    metadata:  
      labels:  
        app: redis    # 외부와 매칭할 Pod 라벨 ( key: value )  
    spec:  
      nodeName: {node name}    # Registry를 설치할 Node 대상 -> <https://kubernetes.io/ko/docs/concepts/scheduling-eviction/assign-pod-node/>  
      containers:  
        - name: redis    # Pod 이름  
          image: redis    # Redis Image  
          command: ["redis-server", "/redis/redis.conf"]    # Redis Image는 redis.conf가 없기에 ConfigMap으로 생성 후 연결  
          resources:  
            requests:  
              memory: 256Mi    # Pod 최소 메모리  
              cpu: 300m    # Pod 최소 CPU  
            limits:  
              memory: 256Mi    # Pod 최대 메모리  
              cpu: 300m    # Pod 최대 CPU  
          livenessProbe:  
            exec:  
              command: ["redis-cli", "ping"]    # Redis Ping Check  
            initialDelaySeconds: 1  
            periodSeconds: 30  
            timeoutSeconds: 5  
            successThreshold: 1  
            failureThreshold: 3  
          volumeMounts:  
            - mountPath: /redis    # redis.conf 생성 후 Mount할 Redis Image내 위치  
              name: config    # Mount할 Volume 이름  
            - mountPath: {logfile dir}    # Mount할 Redis Image내 log 위치  
              name: log    # Mount할 Volume 이름  
      volumes:  
        - name: config  
          configMap:  
            name: redis-config  
            items:  
              - key: redis-config  
                path: redis.conf  
        - name: log  
          hostPath:  
            path: {log path}    # log mount 경로  
            type: Directory

---

kind: Service  
apiVersion: v1  
metadata:  
  name: redis    # Redis Service 이름  
spec:  
  selector:  
    app: redis    # Redis Deployment 이름  
  ports:  
    - port: 6379

---

apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: redis-config  
data:  
  redis-config: |  
    maxmemory 2GB  
    logfile /data/log/redis.log  
    cluster-enabled yes  
    appendonly yes

## Redis Clustering 명령어

### Node Clustering ( Slave 적용 안함 )

kubectl exec -it deploy/{deployment name} -- redis-cli --cluster create `kubectl get po -o wide | grep {pot name} | awk '{print $6":{redis port}"}' | tr '\n', ' '`

### Node Clustering ( Slave 적용 )

kubectl exec -it deploy/{deployment name} -- redis-cli --cluster create `kubectl get po -o wide | grep {pot name} | awk '{print $6":{redis port}"}' | tr '\n', ' '` --cluster-replicas 1

## Spring Boot Redis Clustering 설정

다중노드는 외부에서 접근 불가능 ( k8s Container Cluster 내부에서만 접근가능 )  
접근하기 위해서는 redis-cluster-proxy 필요 -> <https://github.com/RedisLabs/redis-cluster-proxy>

spring:  
  redis:  
    cluster:  
      nodes: {ip}:{port}  
      # nodes: {service name}:{port}    # IP가 아닌 Container내 Redis Service 이름으로 접근 가능