**쿠버네티스 란?**

컨테이너화 된 애플리케이션의 배포, 확장 및 관리를 위해 사용되는 오픈소스 시스템이다.

여러 머신에 배포하기 위한 docker-compose라 생각해도 된다.

**쿠버네티스 역할은?**1. Pod 헬스 체크 후 자동 재시작

2. Pod 오토 스케일링

3. Pod 로드 밸런싱

**AWS-ECS 같은 서비스가 있는데 쿠버네티스를 사용하는 이유?**

AWS-ECS는 AWS에만 쓸 수 있다.

쿠버네티스를 지원하는 프로바이더(AWS, Azure 등)라면 모두 쿠버네티스를 사용할 수 있다.

쿠버네티스는 일부 프로바이더의 특화된 기능을 사용할 수 있게 지원한다.

(프로바이더가 제공 안해도, 패키지 깔아서 쓰면 된다.)

**쿠버네티스 구성:**

**1. Master Node**

클러스터를 제어하고 관리하는 역할

여러 개의 Master Node를 구성하여 고가용성을 확보할 수도 있음

주요 구성 요소:

API Server (kube-apiserver) : 모든 요청을 처리하는 중앙 허브, 워커와 통신

Scheduler (kube-scheduler) : Pod의 실행 위치(Worker Node)를 결정

Kube Controller Manager : 워커들의 상태를 감시하고 제어

Cloud Controller Manager: 클라우드 프로다이더에게 뭘하면 되는지 알려줌

**2. Control Plane:**

Master Node의 주요 구성 요소들을 묶어서 부르는 개념

클러스터 상태를 유지하고 원하는 상태로 조정하는 역할

**3. Worker Node :**

실제 애플리케이션(Pod)이 실행되는 노드 (여러 Pod이 존재할 수 있음)

Master Node의 지시에 따라 동작

주요 구성 요소:

**Kubelet** : Master Node와 통신하여 Pod 실행을 담당

Docker : 컨테이너 실행

**kube-proxy** : 네트워크 트래픽을 적절한 Pod로 라우팅

**4. Pod**

Kubernetes에서 컨테이너를 실행하는 최소 단위

하나의 Pod 안에 여러 개의 컨테이너가 있을 수 있음

Pod 내의 모든 컨테이너는 같은 네트워크와 스토리지를 공유

Volume을 가질 수 있음.

**5. Proxy (kube-proxy)**

Kubernetes 내부에서 네트워크 트래픽을 관리하는 역할

Pod 간 통신 및 서비스 로드 밸런싱을 수행

**쿠버네티스의 흔한 오해:**

쿠버네티스가 알아서 AWS의 EC2 인스턴스 생성하고 Master, WorkerNode 깔아 주는게 아니다.

그런 세팅은 개발자가 직접해야 한다.

AWS의 EKS (Elastic K8s Service)를 사용하면 이런 부분에서 도움을 받을 수 있다.

k8s는 세팅된 환경에서 ‘Pod’ 을 중심으로 헬스체크, 오토 스케일링, 로드 밸런싱을 수행한다.

**클러스터란?**

마스터 노드와 워커 노드가 결합된 Kubernetes 시스템의 집합을 의미한다.

**kubectl 이란?**

새 deployment 생성 및 삭제, 실행 중인 deployment 변경 등의 명령을

클러스터에 보내는데 사용되는 도구이다.

개발자 또는 관리자가 해당 도구로 마스터 노드에 명령을 보낸다.

**쿠버네티스 객체 및 관리 방식:**

**1. Kubernetes에서 인식하는 주요 객체**

* Pod: 컨테이너가 실행되는 기본 단위.
* Deployment: 원하는 개수의 Pod을 자동으로 생성하고 관리하는 객체.
* Service: 네트워크를 통해 Pod에 접근할 수 있도록 하는 객체.
* Volume: Pod에서 사용할 수 있는 저장소.

#### **2. Kubernetes 객체 생성 방식**

* 명령적 방식: kubectl 명령어를 직접 실행하여 객체를 생성.
* 선언적 방식: YAML 파일로 객체를 정의하고 kubectl apply -f 명령을 통해 적용.

#### **3. Pod의 특징**

* 임시적인 존재: 제거되거나 교체될 수 있음, 기본적으로 데이터를 유지하지 않음.
* 자동 관리 권장: Kubernetes가 자동으로 생성, 제거, 교체하도록 운영.
* ip 주소가 계속 변경될 수 있음으로, ip 주소 기반으로 통신하는 것은 좋지 않음.

#### **4. Deployment 객체**

* Pod의 생성 및 관리: 원하는 Pod, 컨테이너 개수를 지정하여 k8s가 적절히 배치.
* 유지보수 기능:
  + 일시 중지, 삭제, 롤백 가능.
  + 문제가 발생했을 때 빠르게 이전 상태로 복구 가능.
* 확장성:
  + 자동 확장(오토스케일링) 지원.
  + 수동으로 스케일 조정 가능 (kubectl scale 명령 사용).

#### **5. Service 객체**

* Pod 을 그룹화 하고 공유 주소/IP를 제공.

**minikube 란?** [사이트](https://minikube.sigs.k8s.io/docs/start/?arch=%2Flinux%2Fx86-64%2Fstable%2Fbinary+download)

로컬 환경에서 k8s를 테스트하고 다뤄보기 위한 툴 이다.

로컬 머신에서 가상 머신을 만들어서 클러스터를 구성한다.

**minikube 설치:**

| curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64  sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64 |
| --- |

**minikube 실행:**

현재 사용자가 docker 그룹에 추가 안되어 있는 경우에만:

| sudo usermod -aG docker $USER newgrp docker |
| --- |

드라이버를 도커로 설정해서 minikube 실행

| minikube start --driver=docker |
| --- |

**kubectl 별칭 등록:**

원한다면 kubectl 별도로 설치해도 된다. [링크](https://kubernetes.io/ko/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux:~:text=%ED%94%BC%ED%95%A0%20%EC%88%98%20%EC%9E%88%EB%8B%A4.-,%EB%A6%AC%EB%88%85%EC%8A%A4%EC%97%90%20kubectl%20%EC%84%A4%EC%B9%98,-%EB%8B%A4%EC%9D%8C%EA%B3%BC%20%EA%B0%99%EC%9D%80%20%EB%B0%A9%EB%B2%95%EC%9C%BC%EB%A1%9C)

나는 설치 귀찮아서 minikube 내장 kubectl 사용하였다.

| alias kubectl="minikube kubectl --" |
| --- |

**minikube 상태 보기:**

| minikube status |
| --- |

**minikube 웹 대시보드 보기:**

| minikube dashboard |
| --- |

**deployment 만들기:**

deployment 가 만들어지면 자동으로 master node로 전송되어, pod과 container가 생성됨.

| kubectl create deployment <**deploymentName**> --image=<**imageName**> |
| --- |

**현재 배포된 Deployment 목록 조회:**

| kubectl get deployments |
| --- |

**deployment 삭제:**

(Deployment가 관리하는 모든 Pod도 함께 제거됨)

| kubectl delete deployment <deploymentName> |
| --- |

**실행 중인 Pod 목록 조회:**

| kubectl get pods |
| --- |

**스케일 조절하기:**

| kubectl scale deployment/<**deploymentName**> --replicas=<**count**> |
| --- |

**deployment 업데이트하기:**

\*\*\* image의 태그 값이 현재와 다른 경우에만 적용된다.

여러 컨테이너가 있는 경우 업데이트할 컨테이너를 명시적으로 지정해야 한다.

| kubectl set image deployment/<deploymentName> <containerName>=<imageName> |
| --- |

**deployment 업데이트 내역 살펴보기:**

| kubectl rollout history deployment/<deploymentName>  # 특정 리비전 더 자세히 살펴보기 kubectl rollout history deployment/<deploymentName> --revision=<revNo> |
| --- |

**가장 최근 deployment 업데이트 롤백하기:**

| kubectl rollout undo deployment/<deploymentName> |
| --- |

**특정 deployment 업데이트 버전으로 롤백하기**

| kubectl rollout undo deployment/<deploymentName> --to-revision=<revNo> |
| --- |

**deployment 업데이트 상태 보기:**

| kubectl rollout status deployment/<**deploymentName**> |
| --- |

**롤링으로 재시작 시키기:**

| kubectl rollout restart deployment <**deploymentName**> -n <namespace> |
| --- |

**서비스?**

Pod 간의 네트워킹을 관리하는 추상화 계층이다.

Pod 을 그룹화 하고 공유 주소/IP를 제공한다.

서비스의 IP 주소는 변경되지 않는다.

**serviceType:**

1. ClusterIP (기본값):  
   클러스터 내부에서만 접근 가능한 IP를 제공하며,   
   쿠버네티스 서비스가 트래픽을 여러 파드로 분산함. 외부에서 직접 접근할 수 없음.
2. NodePort:  
   클러스터의 모든 노드에서 특정 고정 포트(30000~32767)를 열어 외부 접근을 허용함.  
   클라이언트는 NodeIP:NodePort로 접근 가능.  
   (Worker Node의 IP를 통해 노출됨.)
3. LoadBalancer:  
   클라우드 환경(AWS, GCP, Azure 등)에서 **외부 로드 밸런서**를 자동으로 생성하고,  
   공인 IP를 할당하여 외부에서 접근 가능하게 함.
4. ExternalName:  
   외부 도메인(FQDN)으로 트래픽을 리디렉션.

**Service 생성하여 특정 deployment에 의해 생성된 pod 노출:**

| kubectl expose deployment <deploymentName> --type=<serviceType> --port=<port> |
| --- |

**생성된 서비스 보기:**

| kubectl get services |
| --- |

**(중요) 서비스 URL 열기 (minikube 전용):**

minikube 환경에서는 NodePort, LoadBalancer 타입 사용 시 EXTERNAL-IP가 할당되지 않는다.

대신 아래 명령어를 사용하여 URL을 얻은 후 확인한다.

| minikube service <serviceName> |
| --- |

**서비스 삭제하기:**

| kubectl delete service <serviceName> |
| --- |

**k8s 오브젝트 만들기 - 1 (선언적 방식용):**

# deployment.yaml

apiVersion: apps/v1

# 생성하려고 하는 k8s 객체 종류

# Deployment는 자동으로 ReplicaSet을 생성하고 그에 따라 Pod의 수를 관리합니다.

kind: Deployment # Service, Pod, StatefulSet, …

# Deployment에 대한 metadata

metadata:

name: kub2-dep # Deployment 이름

labels:

group: kub2

# Deployment에 대한 Spec

spec:

replicas: 1 # 초기 실행되는 pod 개수

selector: # depoloyment에 의해 제어되어야 하는 pod을 선별

# 입력한 lable과 pod의 label이 일치하면 선별

# 여러 개 입력 가능하고, AND 조건으로 동작함

matchLabels:

app: kub2-pod

# 표현식을 통해서 선별 한다. 여러 개 입력 가능하고, AND 조건으로 동작한다.

# matchExpressions:

# key가 app이고, value가 kub2 또는 kub3인 pod 선별

# - {key: app, operator: In, values: [kub2, kub3]}

# pod에 대한 Template

template:

#pod에 대한 metadata

metadata:

labels:

# 하나 이상 존재 할 수 있음

app: kub2-pod # 자유롭게 key, value 설정 가능

# pod에 대한 spec

spec:

containers:

# 하나 이상 존재 할 수 있음

- name: kub2

image: jerryking976/kub-study:latest

ports:

- containerPort: 8080 # 그냥 이거 쓴다고 알리는 용도

**k8s 오브젝트 만들기 - 2 (선언적 방식용):**

# service.yaml

apiVersion: v1

kind: Service

metadata:

name: backend

labels:

group: kub2

spec:

# service에 의하여 제어 되어야 하는 pod의 label로 지정

# service는 matchLables selector만 지원된다.

selector:

app: kub2-pod

ports:

- protocol: 'TCP'

port: 3000

targetPort: 8080

type: 'LoadBalancer'

// sessionAffinity: ClientIP # 세션 스티키니스 활성화

**단일 k8s 오브젝트 파일 vs 다중 k8s 오브젝트 파일:**

한 파일에서 --- 기호로 구역을 나누어,

각 구역에 deployment, service 설정을 작성한 하나의 Manifest 파일을 만들 수 있다.

(개인적으로는 다중 Manifest 파일을 선호한다.)

**(중요) k8s 오브젝트 파일 적용:**

똑같은 파일을 다시 적용할 수 있다.

파일 내용을 수정하고, 다시 적용하는 방식으로 업데이트 할 수 있다.

-f 옵션을 여러 개 추가하여 한번에 적용할 수 있다.

| kubectl apply -f <fileName> |
| --- |

**(중요) k8s 오브젝트 파일에 의해 생성된 리소스 삭제:**

ex\_) deployment, pod, …

| kubectl delete -f <**fileName**> |
| --- |

**특정 레이블을 가진 리소스 삭제:**

| kubectl delete <target> -l <lableKey>=<lableValue> # kubectl delete deployments,services -l group=kub2 |
| --- |

**k8s 오브젝트, Probe:**

HTTP 요청, TCP 소켓 연결, 명령어 실행 등의 방법을 선택하여 컨테이너 상태 체크 한다.

세부적으로 아래의 옵션 등을 설정할 수 있다.

initialDelaySeconds, periodSeconds, timeoutSeconds, failureThreshold, …

**1) livenessProbe 옵션:**

컨테이너가 살아 있는지 확인하고, 실패하면 컨테이너를 재시작 한다.

spec:

containers:

- name: kub2

image: jerryking976/kub-study

livenessProbe:

httpGet:

path: /

port: 8080

periodSeconds: 10

initialDelaySeconds: 5

**2) readinessProbe 옵션:**

컨테이너가 요청을 받을 준비가 되었는지 확인하고, 실패하면 트래픽을 차단한다.

spec:

containers:

- name: kub2

image: jerryking976/kub-study

readinessProbe:

tcpSocket:

port: 8080

periodSeconds: 10

initialDelaySeconds: 5

**3) startupProbe 옵션:**

애플리케이션 시작이 완료될 때까지 기다렸다가

완료되면 livenessProbe, readinessProbe 가 동작하도록 한다.

spec:

containers:

- name: kub2

image: jerryking976/kub-study

ports:

startupProbe:

tcpSocket:

port: 8080

periodSeconds: 10

failureThreshold: 5

**k8s 오브젝트 파일 팁:**

**1) imagePullPolicy 옵션:**

컨테이너를 배포할 때 이미지를 pull 해오는 정책을 정한다.

Always로 설정 시 항상 이미지를 새로 가져오게 되는데,

이에 따라 최신 이미지를 들고 있음에도 불구하고,

무조건 Rolling 업데이트를 하게 되므로 주의한다.

spec:

containers:

- name: kub2

image: jerryking976/kub-study

# image pull 정책

imagePullPolicy: Always # Never, IfNotPresent(기본)

**2) Image 뒤에 latest 태그 붙이기**

배포 구성에 변경이 있을 때, latest 태그가 붙어 있으면 항상 최신 이미지를 가져와서 다시 판정한다.

**k8s 볼륨 특징:**

기본적으로 명명된 볼륨을 사용하더라도, pod이 삭제되면 볼륨이 삭제된다.

데이터가 저장되는 위치를 완벽하게 제어할 수 있다.

**emptyDir 유형 Volume 마운트 하기:**

Pod 내 컨테이너들이 공유할 수 있는 임시 저장소를 제공하는 볼륨 유형이다.

* Pod가 생성될 때 빈 디렉터리로 생성된다.
* Pod 내 여러 컨테이너가 공유 가능하다.
* Pod가 삭제 되면 데이터도 함께 삭제된다.
* 기본적으로 노드의 임시 저장소를 사용한다.
* medium: "Memory" 설정 시 볼륨을 RAM에 만든다.

# deployment.yaml

template:

...

spec:

containers:

- name: story

image: jerryking976/kub-study2:1

volumeMounts:

- mountPath: /app/story/

name: story-volume

volumes:

- name: stroy-volume

emptyDir: {}

# emptyDir:

# medium: Memory

**hostPath 유형 Volume 마운트 하기:**

Worker Node의 파일 시스템 경로를 Pod 내 컨테이너에 마운트하는 볼륨 유형이다.

* 호스트 시스템의 특정 경로를 Pod에 연결한다.
* 노드가 삭제되면 해당 데이터도 사라진다.
* 다른 노드에서는 사용할 수 없으며, 동일 노드에서만 접근 가능하다.   
  (동일 노드의 파드들은 동일한 경로에 접근하기에, 동일한 볼륨에 접근 가능하다.)

# deployment.yaml

# emptyDir 유형과 모두 같으나, volume 부분만 다름.

template:

...

spec:

...

**volumes:**

**- name: stroy-volume**

**hostPath:**

**path: /data**

**type: DirectoryOrCreate** # 있으면 쓰고 없으면 만듬

**CSI 란?**

CSI는 Container Storage Interface의 약자다.

Kubernetes에서 스토리지 플러그인을 통합하고 표준화하는 API 이다.

**Storage Container란?**

PVC 요청 시, 적절한 PV를 자동으로 생성하는 방식을 정의하는 리소스 이다.

**Storage Container 목록 보기:**

| kubectl get cv |
| --- |

**Persistent Volumes (PV):**

Node와 Pod에 독립적인 영구적인 볼륨 이다. (HostPath 타입 제외)

AWS EBS 같은 프로바이더가 제공하는 영역을 사용할 수 있다.

# host-pv.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: host-pv

spec:

capacity:

storage: 1Gi # 1GB

# Filesystem: 파일 시스템 형태로 사용됨

# Block: 파일 시스템 없이 원시 블록 디바이스 형태로 제공됨

# DB 같은 저지연 고성능 애플리케이션에서 사용됨.

# 직접 파일시스템을 생성 하거나,

# 애플리케이션이 직접 Block을 다룰 수 있어야 함.

volumeMode: Filesystem

accessModes:

# ReadWriteOnce: 하나의 노드에서만 읽기/쓰기가 가능

# ReadOnlyMany: 여러 노드에서 읽기만 가능

# ReadWriteMany: 여러 노드에서 읽기/쓰기 가능

# 지원할 수 있는 만큼 여러 개 입력해도 됨.

- ReadWriteOnce

storageClassName: standard

hostPath:

path: /data

type: DirectoryOrCreate

**생성된 PV 목록 보기:**

| kubectl get pv |
| --- |

**Persistent Volume Claims (PVC):**

PV에 요청하는 API를 담당한다.

Worker Node에 생성 된다고 하는데, 어느 사이트에서는 클러스터에 생성된다고 한다.

pod이 PVC를 통해서 PV에 용량을 할당 받고 뭔가를 쓰고 한다.

# host-pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: host-pvc

spec:

volumeName: host-pv

accessModes:

# 여러 개 지정할 수도 있음

- ReadWriteOnce

storageClassName: standard

resources:

requests:

# 필요한 스토리지 크기를 지정

# PV의 capacity.storage 이상을 요청할 수 없음.

# 요청한 크기가 바로 디스크 공간을 차지하는 것은 아님.

# ex\_) 10Gi를 요청해도, Pod 내부에서 1Gi만 쓰면 나머지는 비어 있는 상태.

storage: 1Gi

**Pod 에서 PVC 사용하기:**

# deployment.yaml

spec:

containers:

- name: story

image: jerryking976/kub-study2:1

ports:

- containerPort: 8080

volumeMounts:

- mountPath: /app/story

name: story-volume

volumes:

- name: story-volume

persistentVolumeClaim:

claimName: host-pvc

**생성된 PVC 목록 보기:**

| kubectl get pvc |
| --- |

**환경 변수 적용하기 (방법 1):**

컨테이너에 해당 하는 부분에 env 옵션을 추가하면 된다.

제일 간단한 방법이지만, 불필요한 반복을 요할 수 있다.

# deployment.yaml

spec:

containers:

- name: story

image: jerryking976/kub-study2:1

env:

- name: STORY\_FOLDER

value: 'story'

**환경 변수 적용하기 (방법 2):**

# environment.yaml

apiVersion: v1

kind: ConfigMap

meataData:

name: data-store-env

data:

# Key: Value를 쭉쭉 쓰면 됨.

folder: 'story'

# deployment.yaml

spec:

containers:

- name: story

image: jerryking976/kub-study2:1

env:

- name: STORY\_FOLDER

valueFrom:

configMapKeyRef:

name: data-store-env

key: folder

\*\* environment.yaml을 kubectl apply 하고 사용해야 합니다.

**생성된 ConfigMap 확인:**

| kubectl get configMap |
| --- |

**Pod 내부에 있는 컨테이너 간 통신 방법:**

Localhost를 통해서 통신하면 된다.

**Pod 간 내부 통신 방법 :**

service.yaml의 type을 ClusterIp로 바꾼다.

# service.yaml

spec:

selector:

app: users

type: ClusterIp

ports:

- protocol: TCP

port: 8080

targetPort: 8080

이 아래는 ClusterIp로 설정된 서비스에 어떻게 접근하는 가를 다룬다.

1. (비추) IP 주소를 직접 입력하는 방법  
   service의 클러스터IP를 확인하고, 이를 사용한다.

# deployment.yaml

spec:

containers:

- name: users

image: jerryking976/kub-study3-users

env:

- name: AUTH\_ADDRESS

value: "10.99.104.252"

1. (비추천) 환경 변수를 쓰는 방법

쿠버네티스가 실행 될 때, 자동으로 서비스의 host 주소가 환경 변수로 등록된다.

환경 변수는 아래와 같은 규칙으로 등록된다.

* <SERVICE\_NAME>\_SERVICE\_HOST

만약, user-service 라는 이름의 서비스가 실행 중이라면, 아래와 같은 환경변수가 등록된다.

* USER\_SERVICE\_SERVICE\_HOST

1. (추천) service 이름을 도메인으로 쓰는 방법

쿠버네티스는 기본적으로 CoreDNS를 서비스를 함께 제공한다.

<serviceName>.<nameSpace>가 도메인으로 자동생성되고 이를 사용할 수 있다.

# deployment.yaml

...

env:

- name: AUTH\_ADDRESS

value: "auth-service.default" # <serviceName>.<nameSpace>

**네임스페이스란?**

클러스터 내에서 리소스를 논리적으로 분리하는 방법이다.

다중 팀/프로젝트 관리: 서로 다른 애플리케이션을 격리 가능

리소스 제한 (ResourceQuota): 네임스페이스별 CPU, 메모리 할당 가능

같은 리소스 이름 사용 가능: 네임스페이스로 구분하여, 같은 서비스/파드 이름 사용 가능

기본 네임스페이스: default, kube-system, kube-public, kube-node-lease 등이 존재

**네임스페이스 생성:**

| kubectl create namespace <name> |
| --- |

**생성된 네임스페이스 확인:**

| kubectl get namespaces |
| --- |

**.yaml 파일에서 namespace 지정:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: users-dep

namespace: my-namespace

**리소스 제한 거는 방법:**

apiVersion: v1

kind: ResourceQuota

metadata:

name: my-quota

namespace: my-namespace # ← 적용할 네임스페이스 지정

spec:

hard:

pods: "10" # 최대 10개 파드 제한

requests.cpu: "2" # 전체 요청 가능한 CPU 총합: 2코어

requests.memory: "4Gi" # 전체 요청 가능한 메모리 총합: 4GiB

limits.cpu: "4" # 최대 사용 가능 CPU 총합: 4코어

limits.memory: "8Gi" # 최대 사용 가능 메모리 총합: 8GiB

**걸린 리소스 제한 확인하는 방법:**

| kubectl get resourcequota -n <nameSpaceName> |
| --- |

**LimitRange 란?**

Pod 또는 컨테이너에 대한 리소스 제한(CPU, 메모리 등)을 설정할 수 있는 리소스입니다. 이를 통해 Pod가 사용할 수 있는 리소스의 최소 및 최대 값을 정의하거나, 기본적으로 할당되는 리소스 값을 설정할 수 있습니다.

**yaml 파일:**

**apiVersion: v1**

**kind: LimitRange**

**metadata:**

**name: lr-1**

**spec:**

**limits:**

**- type: Container**

**min:**

**memory: 0.1Gi**

**max:**

**memory: 0.4Gi**

**maxLimitRequestRatio:**

**memory: 3**

**defaultRequest:**

**memory: 0.1Gi**

**default:**

**memory: 0.2Gi**

**생성한 LimitRange 확인:**

| kubectl get limitrange -n default |
| --- |

**Horizontal Pod Auto Scaler (HPA) 란?**

HPA는 Kubernetes에서 CPU, 메모리 또는 사용자 정의 메트릭을 기반으로 자동으로 Pod 수를 조정하는 기능입니다. 트래픽 증가 시 Pod을 추가하고, 감소 시 줄여서 리소스를 효율적으로 관리할 수 있습니다.

**Metrics Server 설치 (HPA가 메트릭을 수집하기 위해 필요):**

| kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml |
| --- |

**yaml 파일:**

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

name: custom-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 2

maxReplicas: 10

metrics:

- type: Pods

pods:

metric:

name: http\_requests\_per\_second

target:

type: AverageValue

averageValue: 10

**생성한 HPA 확인:**

| kubectl get hpa |
| --- |

**StateFulSet이란?**

상태 저장 애플리케이션을 Kubernetes에서 배포하는 데 사용되는 리소스입니다.

일반적으로 데이터베이스나 큐 시스템 등과 같은 상태를 가진 애플리케이션에 적합합니다.

StatefulSet은 Pod의 안정적인 네트워크 ID, 영속적인 저장소(PV), Pod 순서 보장 등 상태 저장 애플리케이션에 필요한 기능을 제공합니다.

**주요 특징**

1. 안정적인 네트워크 ID:  
   Pod가 삭제되거나 재시작되더라도 고유한 이름을 유지합니다.
2. 영속적 저장소:  
   각 Pod는 PV통해 데이터를 영구 저장할 수 있습니다.
3. Pod 순서 보장:  
   StatefulSet은 Pod를 순차적으로 생성하거나 삭제합니다.  
   이는 상태 저장 애플리케이션이 안정적으로 동작하도록 합니다.
4. 복제 가능:   
   StatefulSet은 여러 Pod를 배포할 수 있으며,  
   각 Pod는 고유한 네트워크 ID와 스토리지를 유지합니다.

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql

spec:

serviceName: "mysql"

replicas: 3 # 3개의 Pod 생성

selector:

matchLabels:

app: mysql

template:

metadata:

labels:

app: mysql

spec:

containers:

- name: mysql

image: mysql:5.7

env:

- name: MYSQL\_ROOT\_PASSWORD

value: "rootpassword"

ports:

- containerPort: 3306

volumeMounts:

- name: mysql-data

mountPath: /var/lib/mysql

volumeClaimTemplates:

- metadata:

name: mysql-data

spec:

accessModes: ["ReadWriteOnce"]

resources:

requests:

storage: 1Gi # 각 Pod에 1Gi 크기의 PersistentVolume 할당

**생성된 StatefulSet 상태 확인:**

| kubectl get statefulset |
| --- |

**StatefulSet 서비스 접근하기:**

StatefulSet은 각 Pod에 대해 **고유한 DNS 이름**을 생성합니다. 예를 들어, mysql-0, mysql-1 등과 같은 이름을 갖게 됩니다. 이들을 사용하여 Pod 간에 서로 통신할 수 있습니다.

ex\_)MySQL 0번 Pod에 접근하려면:

| mysql-0.mysql |
| --- |

**StatefulSet:**

StatefulSet은 각 파드에 대해 고유한 이름을 부여하고,

각 파드에 대해 PV를 자동으로 생성하고 관리합니다.

예를 들어, statefulset-0, statefulset-1와 같은 이름을 가진 파드가 각각 고유한 PV를 갖게 됩니다.

데이터의 영속성을 보장하기 위해 파드가 재시작될 때마다 동일한 PV에 접근할 수 있습니다.

**StatefulSet이 아닌 경우:**

StatefulSet이 아니어도 Deployment나 Pod에서도 PV를 사용할 수 있습니다.

이 경우, PV는 수동으로 파드와 연결되어야 하며,

특정 파드가 삭제되거나 재배포되면 동일한 PV에 대한 연결을 관리해야 합니다.

PVC를 사용하여 파드에서 PV를 요청하고 연결합니다.

**AWS에서 쉽게 쿠버네티스를 사용할 수 있는 서비스:**

EKS (Elastic Kubernetes Service) 를 사용하면 된다.

EKS에서 컨트롤 패널을 알아서 관리해주니 편하다.

EC2 인스턴스 하나하나 직접 설정해서 사용하는 방법도 있긴 하다.

* kops 라는 github 프로젝트도 인기 있는 것 같다.

**기타 유용한 커맨드:**

**서비스에 연결된 Pod 들 확인하는 방법**

| kubectl get endpoints <service명> |
| --- |

**특정 pod 또는 service 세부 사항 확인 하는 방법**

| kubectl describe pod POD\_NAME -n NAMESPACE kubectl describe service SERVICE\_NAME -n NAMESPACE |
| --- |

**CNI (Container Network Interface)**

- Pod에 IP를 부여하고, 네트워크 연결을 구성하는 표준 인터페이스

- Kubernetes는 Pod 생성 시 CNI 플러그인을 호출하여 네트워크 세팅

- 대표적인 CNI 플러그인: Flannel, Calico, Cilium 등

**Flannel**

- CNI 플러그인의 한 종류

- 오버레이 네트워크(VXLAN)를 사용해 Pod 간 IP 통신을 가능하게 함

- 역할: L3 계층에서 Pod 네트워크 구성

**CRD(CustomResourceDefinitions)**

CRD 는 k8s API의 익스텐션이다.

표준 쿠버네티스 API 안에 포드, 서비스, 배포물, 레플리카 세트 등이 있다.

CRD를 사용하면, 자기만의 커스텀 리소스 정의를 이용해서 자신만의 k8s 객체를 만들 수 있다.

자주 사용 하진 않는데, istio 같은 프레임워크는 CRD를 사용해서 k8s를 확장한다.

**Istio**

- 오픈 소스 서비스 메시 (L7 계층)

- HTTP, gRPC 트래픽을 사이드카 프록시(Envoy)를 통해 제어

- 기능: 트래픽 라우팅, 인증, 모니터링, 로깅 등

- 전제 조건: Pod 간 네트워크(IP 통신)가 가능해야 함 → 즉, CNI가 먼저 구성되어야 함

**계층별 역할 정리**

- L3 (네트워크 연결): CNI (예: Flannel) → Pod 간 통신 가능하게 함

- L7 (트래픽 제어): Istio → 트래픽을 라우팅하고 보안 적용

**요약**

- minikube는 기본적으로 CNI를 자동 설치함

- Flannel은 네트워크 연결 담당 (IP 통신)

- Istio는 트래픽 제어 담당 (HTTP/gRPC 통신)

- 둘은 공존하며, 역할이 다름

**필요하면 더 찾아볼 것:**

**\* Ingress와 Ingress Controller**

외부 트래픽을 클러스터 내로 라우팅하는 방법입니다.

**HTTP(S) 트래픽을 제어**하는 로드밸런싱 및 SSL/TLS 종료 설정을 다룹니다.

\* ReplicaSet

Kubernetes에서 파드의 복제본을 관리하는 리소스입니다.

이를 통해 지정된 수의 파드가 항상 실행되도록 보장합니다.

\* DaemonSet

모든 노드에 하나씩 Pod를 배포하는 리소스입니다.

노드 모니터링, 로깅, 메트릭 수집 등의 용도로 사용됩니다.

\* Network Policy

Pod 간의 네트워크 접근 제어를 설정하는 방법입니다.

Pod 간 통신을 제한하거나 허용하는 규칙을 설정할 수 있습니다.

\* Logging & Monitoring

K8s 클러스터 및 애플리케이션의 상태를 추적하는 방법입니다.

Prometheus와 Grafana를 이용한 모니터링,

ELK Stack을 통한 로깅을 설정하고 분석하는 방법을 알아야 합니다.

\* Kubernetes Scheduler

Pod가 실행될 노드를 결정하는 컴포넌트입니다.

Affinity, Taints/Tolerations, Resource Requests/Limit을 사용해

노드 선택 전략을 최적화할 수 있습니다.

\*RBAC (Role-Based Access Control)

K8s 리소스에 대한 액세스를 제어하는 방법입니다.

Role과 ClusterRole을 통해 권한을 부여하고,

RoleBinding과 ClusterRoleBinding을 통해 이를 할당합니다.

\* Job, CronJob