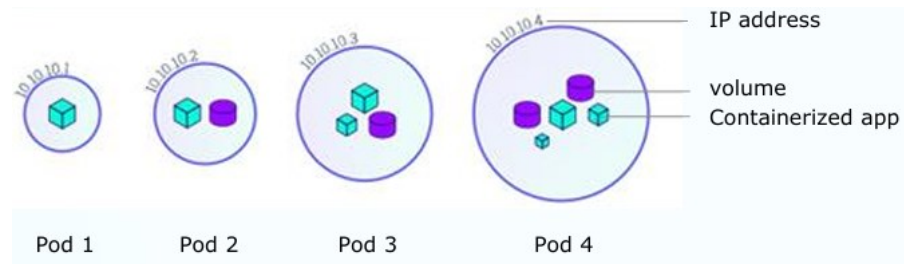


쿠버네티스 운영 핵심 기능

- 볼륨(Volume), 디스크(Disk), 서비스(Service), 헬스 체크(Health Check)

1. Volume 이란?

- 컨테이너는 기본적으로 휘발성 스토리지를 사용 → 재시작 시 데이터 손실
- Volume 은 Pod 단위에서 공유되는 **영속적인 저장 공간**을 제공



특징:

- 컨테이너가 재 시작되어도 데이터 유지
- 여러 컨테이너 간 데이터 공유 가능 (같은 Pod 내)

종류 예시:

- emptyDir, hostPath, persistentVolumeClaim, nfs, configMap, secret

```
root@Dominica:~# kubectl get pod hello-kube-5bd9565b9c-kvzg8 -o yaml
```

```
volumes:
- name: kube-api-access-glgwb
  projected:
    defaultMode: 420
    sources:
    - serviceAccountToken:
        expirationSeconds: 3607
        path: token
    - configMap:
        items:
        - key: ca.crt
          path: ca.crt
          name: kube-root-ca.crt
    - downwardAPI:
        items:
        - fieldRef:
            apiVersion: v1
            fieldPath: metadata.namespace
          path: namespace
status:
```

spec.volumes

- serviceAccountToken : API 서버와 통신하기 위한 토큰 파일 (token 경로에 생성)
- configMap : ConfigMap 에서 ca.crt 키를 추출하여 파일로 마운트 (ca.crt)
- downwardAPI: Pod 의 메타데이터 중 네임스페이스 정보를 namespace 파일로 제공

YAML의 volumes 기본 항목

항목 유형	세부 항목	설명
volume name	kube-api-access-glgwb	Kubernetes 가 자동으로 생성한 API 접근용 볼륨 이름
projected		여러 소스를 하나의 볼륨으로 병합
defaultMode	420 (8 진수: 0644)	생성된 파일의 권한
sources	serviceAccountToken	서비스 계정 토큰 자동 마운트
	expirationSeconds: 3607	토큰 유효 시간
	path: token	토큰 파일 경로
sources	configMap	ConfigMap 의 특정 키를 파일로 마운트
	name: kube-root-ca.crt	사용되는 ConfigMap 이름
	items.key: ca.crt	ConfigMap 내부의 키
	items.path: ca.crt	Pod 내부 경로 (파일 이름)
sources	downwardAPI	Pod 메타데이터 정보를 파일로 마운트
	fieldPath: metadata.namespace	현재 네임스페이스 정보
	path: namespace	해당 정보를 담은 파일 경로

각 Volume 유형: [Volumes | Kubernetes](#)

분류	볼륨 유형	설명
Temp (임시 볼륨)	emptyDir volumes: - name : shared-storage emptyDir: {}	Pod 가 생성될 때 비어 있는 디렉토리. Pod 삭제 시 데이터도 삭제됨
Local (로컬 디스크 기반)	hostPath, gitRepo volumes: - name : tpath hostPath: path: /tmp type: Directory volumes: - name: html gitRepo: repository: https://github.com/finish07sds/kubia-website-example.git revision: master directory: .	호스트 노드의 디렉토리를 마운트하거나 Git 저장소를 복제
Network (네트워크 스토리지)	NFS, iSCSI, GlusterFS	외부 공유 스토리지 서버와 연결해 데이터 공유
Cloud 제공 볼륨	gcePersistentDisk, awsElasticBlockStore, azureDisk, vsphereVolume	클라우드에서 제공하는 블록 스토리지를 Pod 에 연결
기타 특수 볼륨	secret, configMap	민감 정보나 설정값을 파일 형태로 컨테이너에 주입
고급 스토리지	fiberChannel	SAN(Storage Area Network) 환경에서 사용하는 빠른 스토리지 연결 방식

디스크와 Persistent Volume

- Persistent Volume(PV): 클러스터 관리자에 의해 사전 정의된 저장 공간
- PersistentVolumeClaim(PVC): 사용자가 필요한 저장 용량과 접근 방식을 정의하여 요청



```
volumes:  
  - name: my-volume  
    persistentVolumeClaim:  
      claimName: my-claim
```

[실습 01] Volume 종류와 특성을 직접 실습, 확인

YAML 작성, 배포, 확인, 삭제 단계 포함

Q1) emptyDir 볼륨을 사용하는 Pod 를 생성하고, 공유 디렉토리에 파일을 만들어 확인하자. _ Q1_emptydir.yaml 생성

[요구 조건]

- busybox 컨테이너 1 개
- /shared 디렉토리에 hello.txt 파일 생성
- emptyDir 볼륨 사용

[힌트]

volumeMounts 와 volumes 의 emptyDir: {} 사용

Q2) hostPath 를 이용해 노드의 /tmp 디렉토리를 컨테이너에 마운트하고, 그 안에 파일을 생성하자

[요구 조건]

- hostPath 로 /tmp 마운트
- 컨테이너에서 hello-host.txt 파일 생성

Q3) PVC 를 활용하여 Pod 가 영구 저장소에 데이터를 저장하자.

[구성]

- 1 단계: PersistentVolume 과 PersistentVolumeClaim 정의
- 2 단계: PVC 를 사용하는 Pod 생성

[추가 미션]

- Pod 내부에 파일 작성 후 PVC 를 재사용하여 데이터 유지 확인

Q4) 외부 NFS 서버를 마운트하는 Pod 를 작성하고, /mnt 디렉토리에 접근하자

[전제 조건]

- NFS 서버가 미리 설치되어 있어야 함

[포인트]

- nfs 유형의 volumes 정의
 - Pod 내부에서 NFS 디렉토리 접근 및 읽기 확인
-

Q5) ConfigMap 을 생성하여 HTML 텍스트를 /etc/html/index.html 에 마운트 하자

[요구 조건]

- ConfigMap 에 key: index.html, value: <h1>Hello ConfigMap</h1>
- Pod 에서는 /etc/html/index.html 로 마운트

[확인 방법]

- 컨테이너 안에서 cat /etc/html/index.html
-

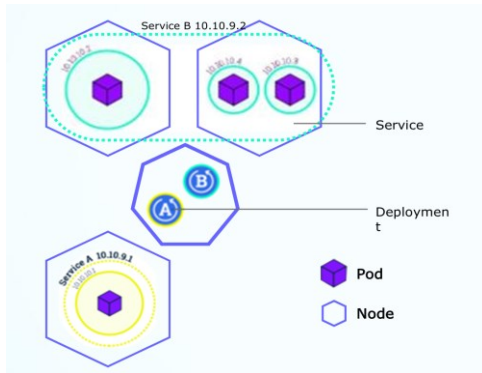
Q6) Secret 을 생성하여 민감 정보를 환경변수로 전달하고 출력하자

[요구 조건]

- Secret 에 key: PASSWORD, value: admin123
- Pod 에서는 envFrom 또는 env 를 사용하여 출력

2. Kubernetes Service

- **Service란?** : Service 는 Pod 집합에 접근하는 방법을 기술하는 Kubernetes API 객체로, 포트 및 로드밸런서를 정의하며, 내부 또는 외부에서 마이크로서비스를 연결할 수 있게 해주는 액세스 포인트



- 종류:

- ① ClusterIP: 내부 통신용 (기본값)
- ② nodePort: 외부에서 접근 가능한 고정 포트
- ③ LoadBalancer: 클라우드 환경에서 외부 로드 밸런서와 연동
- ④ ExternalName: 외부 도메인 주소로 포워딩

```
kind: Service
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30080
```

```
apiVersion: v1
kind: Service
metadata:
  name: hello-node-svc
spec:
  selector:
    app: hello-node
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
  type: LoadBalancer
```

Service Discovery 와 라우팅

- Service Discovery:
 - Kubernetes 는 DNS 기반의 이름 해석 지원
 - 예: my-service.default.svc.cluster.local 로 접근



3. 헬스 체크 (Health Check)

- 목적: Pod 의 상태를 지속적으로 점검하여 비정상 상태의 Pod 를 회피 또는 재시작
- 종류:
 1. **livenessProbe**: 살아있는지 검사 → 실패 시 재시작
 2. **readinessProbe**: 준비됐는지 검사 → 실패 시 트래픽 차단
 3. **startupProbe**: 앱 시작 완료됐는지 검사 → 초기 기동 확인

```
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
```

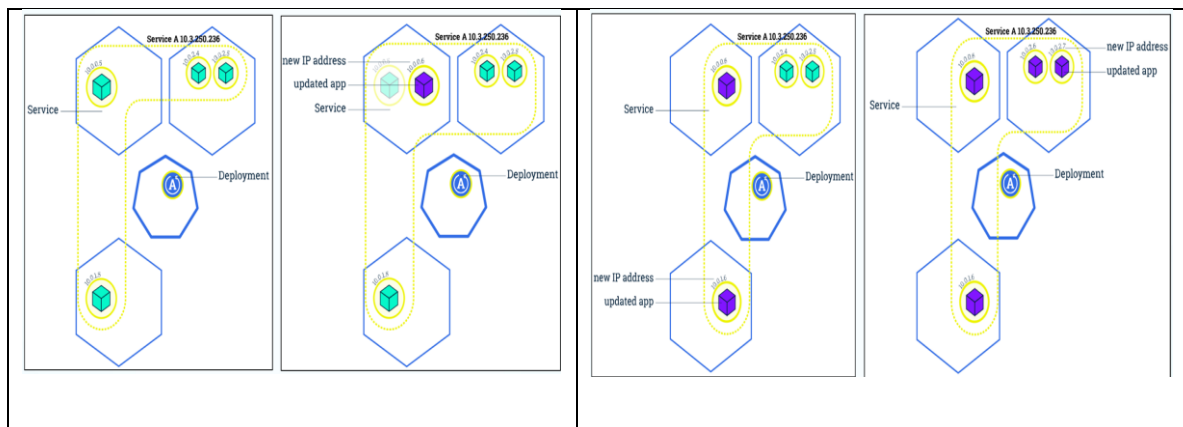

4. 롤링 업데이트(Rolling Update)

서비스 중단 없이 Pod 를 점진적으로 새 버전으로 교체하는 배포 방식

왜 사용 하는가?

전통 방식	롤링 업데이트 방식
전체 앱을 중단하고 새로 배포	점진적으로 교체하며 서비스는 계속 유지됨
다운타임 발생 가능	무중단 배포 가능
수동 조절 필요	자동으로 새 Pod 생성 및 기존 Pod 제거

작동 방식 -> [롤링 업데이트 수행하기 | Kubernetes](#)



새로운 버전의 이미지로 Deployment를 수정

Kubernetes가 기존 Pod를 하나씩 제거

동시에 새로운 Pod를 하나씩 생성

사용자는 항상 Ready 상태인 Pod로 연결

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 1 # 새 Pod 를 몇 개까지 더 만들 수 있는가

maxUnavailable: 1 # 동시에 중단 가능한 기존 Pod 수

[실습] RollingUpdate 를 실습해보자.

Java → Docker → Kubernetes" 롤링 업데이트 실습

구성	설명
Calculator.java	버전 1.0: 덧셈, 뺄셈만 포함
Calculator.java	버전 2.0: 곱셈, 나눗셈 추가
Dockerfile	OpenJDK 21 기반 실행 환경
calc-deploy.yaml	Kubernetes RollingUpdate 구성

1 단계: 버전 1.0 - 덧셈, 뺄셈만 포함된 Calculator 컨테이너 배포

[1-1] nano Calculator.java 작성 (v1)

```
public class Calculator {
    public static void main(String[] args) {
        int a = 10, b = 5;
        System.out.println("버전 1.0 실행 중...");
        System.out.println("덧셈 : " + (a + b));
        System.out.println("뺄셈 : " + (a - b));
    }
}
```

[1-2] 컴파일

```
javac Calculator.java
```

[1-3] Dockerfile 작성

```
FROM openjdk:21
COPY Calculator.class /app/
WORKDIR /app
CMD ["java", "Calculator"]
```

[1-4] Docker 이미지 빌드 및 푸시

```
docker build -t finish07sds/calc-demo:1.0 .
```

```
docker push finish07sds/calc-demo:1.0
```

```
root@Dominica:~# docker push finish07sds/calc-demo:1.0
The push refers to repository [docker.io/finish07sds/calc-demo]
265b46b42ad8: Pushed
10359c5dc4ba: Mounted from library/openjdk
601b48657e0c: Mounted from library/openjdk
b42107e74152: Mounted from library/openjdk
1.0: digest: sha256:2930e1942224c3b43595d04922d77314a529611b12455386b8b2a6992bde6127 size: 1161
```

[1-5] Kubernetes Deployment 작성 및 적용

nano calc-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calc-deploy
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: calc
  template:
    metadata:
      labels:
        app: calc
    spec:
      containers:
      - name: calc
        image: finish07sds/calc-demo:1.0
        command: ["java", "Calculator"]
        tty: true
        stdin: true
```

```
kubectl apply -f calc-deploy.yaml
```

```
root@Dominica:~# kubectl apply -f calc-deploy.yaml
deployment.apps/calc-deploy configured
root@Dominica:~#
```

[1-6] Pod 확인 및 실행

```
kubectl get pods -l app=calc
```

```
# 한 개의 Pod 에서 실행 확인
```

```
kubectl exec -it <POD_NAME> -c calc -- java Calculator
```

```
root@Dominica:~# kubectl get pods -l app=calc
NAME                                READY   STATUS    RESTARTS   AGE
calc-deploy-766b8b5bd-kv72m        1/1     Running   0           2m33s
calc-deploy-766b8b5bd-n87ks        1/1     Running   0           30m
root@Dominica:~# kubectl exec -it calc-deploy-766b8b5bd-kv72m -c calc -- java Calculator
```

2 단계: Calculator 버전 2.0 으로 롤링 업데이트

주요 변경사항

- 곱셈 (*) 과 나눗셈 (/) 기능을 추가
- 버전 문구: "버전 2.0 실행 중..."

[2-1] Calculator.java (v2.0)

```
GNU nano 7.2 Calculator.java
public class Calculator {
    public static void main(String[] args) {
        int a = 10, b = 5;
        System.out.println("버전 2.0 실행 중...");
        System.out.println("덧셈 : " + (a + b));
        System.out.println("뺄셈 : " + (a - b));
        System.out.println("곱셈 : " + (a * b));
        System.out.println("나눗셈 : " + (a / b));
    }
}
```

[2-2] 컴파일

```
root@Dominica:~# javac Calculator.java
```

[2-3] Docker 이미지 빌드 및 푸시

```
docker build -t finish07sds/calc-demo:2.0 .
docker push finish07sds/calc-demo:2.0
```

[2-4] Deployment 수정 (버전 변경)

calc-deploy.yaml 파일에서 아래 부분만 수정 1.0 ->2.0

```
spec:
  containers:
  - name: calc
    image: finish07sds/calc-demo:2.0 #1.0 ->2.0
    stdin: true
    tty: true
    command: ["java", "Calculator"]
```

[2-5] 롤링 업데이트 적용

```
kubectl apply -f calc-deploy.yaml
```

```
root@Dominica:~# kubectl apply -f calc-deploy.yaml
deployment.apps/calc-deploy configured
```

[2-6] 롤링 상태 확인

```
kubectl rollout status deployment calc-deploy
```

```
root@Dominica:~# kubectl rollout status deployment calc-deploy
Waiting for deployment "calc-deploy" rollout to finish: 1 out of 2 new replicas have been update
d...
```

[2-7] 실행 확인

```
kubectl get pods -l app=calc
```





```
kubectl exec -it <새로운 POD_NAME> -c calc -- java Calculator
```

Tags

DOCKER SCOUT INACTIVE

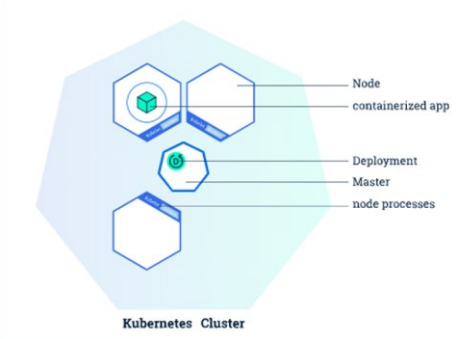
[Activate](#)

This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
 2.0		Image	less than 1 day	10 minutes
 1.0		Image	less than 1 day	26 minutes

4.Deployment

<https://kubernetes.io/ko/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/>



The diagram illustrates the components of a Kubernetes Cluster. It shows a central 'Kubernetes Cluster' box containing a 'Node' which runs a 'containerized app'. A 'Deployment' is shown managing the 'Master' and 'node processes'.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels: ...
```

구성요소

- apiVersion, kind, metadata
- spec.replicas: 원하는 Pod 수
- selector: 대상 Pod 선택
- template: 생성될 Pod 의 정의

특징

- 자동 스케일링
- 자동 복구(Self-healing)
- 무중단 배포 지원
- 클러스터 상태 지속 모니터링

배포방법

- 이미지 + 복제 수 지정 → 클러스터에 배포
- Kubectl create 또는 kubectl apply

5. ConfigMap

- 환경설정 데이터를 키-값 형태로 저장하는 Kubernetes API 객체
- 역할
 - 설정(환경 변수 등)을 컨테이너와 분리하여 재사용성 확보
 - 컨테이너 이미지 변경 없이 설정만 교체 가능

ConfigMap 스펙 구조

- 예시:
 - DB_URL, DB_USER, DEBUG_INFO 등의 key-value 설정
- 명령어:
 - `kubectl get configmap`, `kubectl describe configmap`

확인 항목	명령어
생성 여부 확인	<code>kubectl get configmap</code>
상세 내용 확인	<code>kubectl describe configmap config-dev</code>
YAML 전체 보기	<code>kubectl get configmap config-dev -o yaml</code>
Pod 내부 확인	<code>printenv</code> , <code>cat /etc/config/...</code> 등

```

root@Dominica:~# kubectl get configmap
NAME          DATA   AGE
kube-root-ca.crt 1       23h
nginx-html     1       22h
root@Dominica:~# kubectl get configmap nginx-html -o yaml
apiVersion: v1
data:
  index.html: |
    <html>
    <head><title>Hello from ConfigMap</title></head>
    <body><h1>Welcome to Kubernetes!</h1></body>
    </html>
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"index.html":"\u003chtml\u003e\n\u003chead\u003e\n\u003ctitle\u003eHello from ConfigMap\u003c/t
title\u003e\n\u003chead\u003e\n\u003cbody\u003e\n\u003ch1\u003eWelcome to Kubernetes!\u003c/h1\u003e\n\u003cbody\u003e\n\u003chtml
\u003e\n"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"nginx-html","namespace":"default"}}
    creationTimestamp: "2025-06-24T07:55:01Z"
    name: nginx-html
    namespace: default
    resourceVersion: "2015"
    uid: 42365d59-7bad-494b-918f-f99b79545d08
root@Dominica:~#
  
```

ConfigMap 사용 방법 3 가지 방법

- ① **파일 시스템**: Pod 에 마운트 → 설정파일처럼 사용
- ② **환경변수**: 컨테이너 시작 시 설정 값 전달
- ③ **명령줄 인자**: args 에 동적으로 값 주입