



“Kubernetes Odev-Proje Raporu”

Muhammet ATMACA

Teslim Tarihi: 4 Eylül 2025

İçindekiler

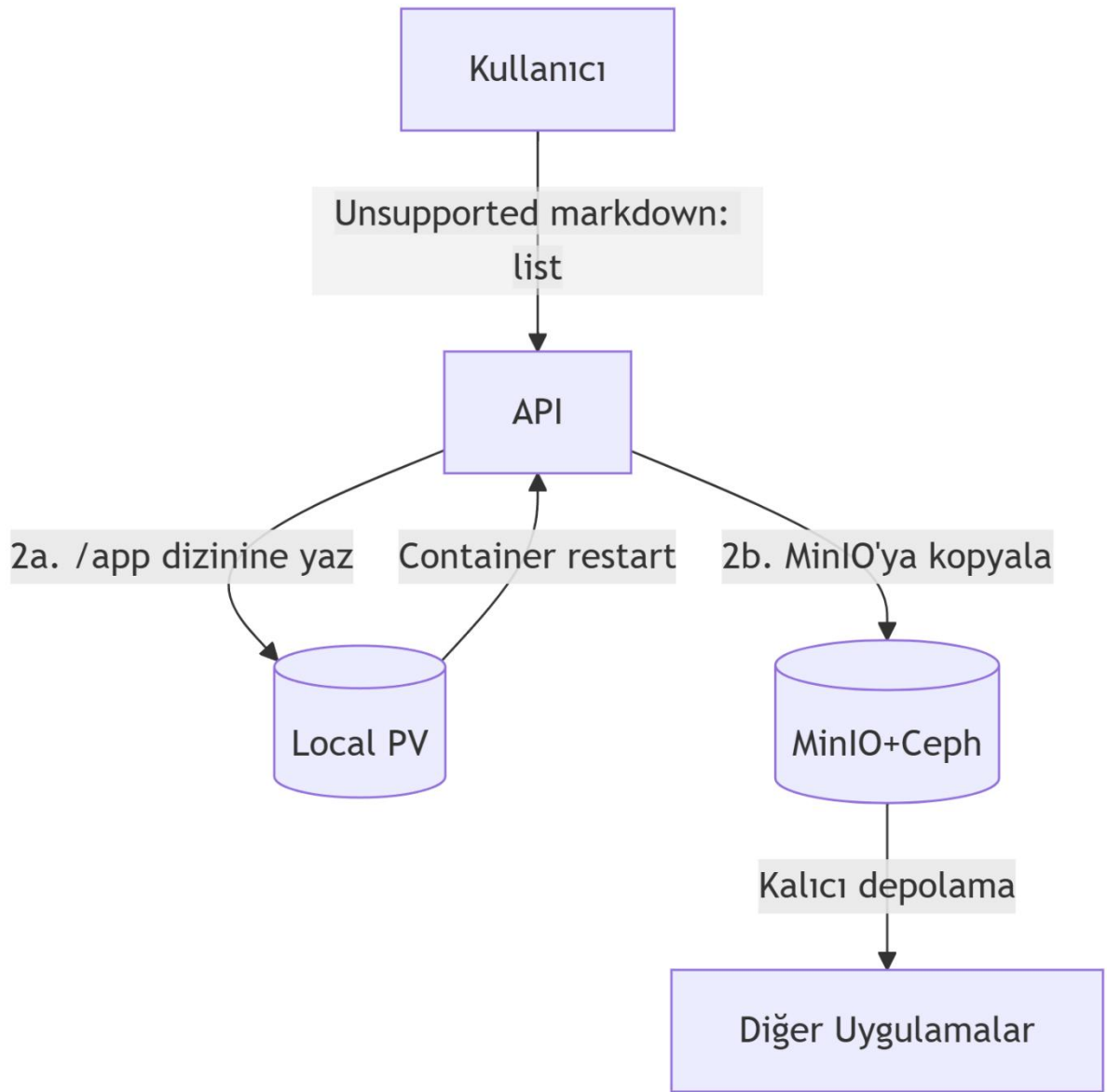
1. Proje Özeti ve Amaç	4
1.1. Proje Amacı.....	4
1.2. Proje Topolojisi	5
1.2.1. Master Düğüm	5
1.2.2. Worker Düğümler	5
2. Proje Oluşum Sırası ve Aşamaları	6
2.1. Bölüm 1: Makine Kurulumları.....	6
2.1.1. Adım 1: Master Makine Kurulumu	6
2.1.2. Adım 2: Node Makine Kurulumu (2 Adet)	6
2.2. Bölüm 2: Sanal Makinelere Statik IP Atama.....	6
2.2.1. Adım 1: Master Makine İçin Statik IP Ataması.....	7
2.2.2. Adım 2: Node Makineleri İçin Statik IP Ataması	7
2.3. Bölüm 3: Ortak Paket Kurulumları ve Ayarlar	8
2.3.1. Adım 1: Güncellemeler ve Gerekli Paketler	8
2.3.2. Adım 2: Container Runtime Kurulumu	8
2.3.3. Adım 3: Swap'ı Devre Dışı Bırakma.....	9
2.4. Bölüm 4: Kubernetes Paketlerini Kurma (Tüm Makinelerde)	10
Gerekli GPG anahtarını indir ve ekle:	10
Kubernetes deposunu ekle:.....	10
Paketleri güncelle ve Kubernetes araçlarını kur:	10
<i>Paketlerin otomatik güncellenmesini engelle</i>	<i>10</i>
2.5. Bölüm 5: Kubernetes Kümesini Başlatma (Sadece Master Makinede)	10
Küme başlatma komutunu çalıştır:	10
Başlatma işleminden sonra çıkan komutları uygula:.....	11
<i>Flannel CNI'ı kur</i>	<i>11</i>
2.6. Bölüm 6: Node Makinelerini Kümeye Dahil Etme (Sadece Node Makinelerinde)	11
2.7. Bölüm 7: Kümenin Durumunu Kontrol Etme (Sadece Master Makinede)	11

2.8. Bölüm 8: Basit API Projesi Geliştirme	12
2.8.1. Adım 1: Proje Dizinini Oluşturma	12
2.8.2. Adım 2: app.py Dosyası.....	12
2.8.3. Adım 3: requirements.txt Dosyası	14
2.8.4. Adım 4: Dockerfile Dosyası	14
2.9. Bölüm 9: Docker İmajı Oluşturma ve Saklama	14
2.9.1. Adım 1: Docker İmajını Oluşturma	15
2.9.2. Adım 2: Docker İmajını Kubernetes'in Erişebileceği Yere Yükleme.....	15
2.10. Bölüm 10: Ceph Kurulumu için Hazırlık.....	15
2.10.1. Adım 1: Rook Depolarını Klonlama (Sadece Master Makinede)	15
2.10.2. Adım 2: Ceph Cluster Manifestlerini Düzenleme	15
2.11. Bölüm 11: Ceph'i Kubernetes'e Kurma (Sadece Master Makinede).....	18
2.11.1. Adım 1: Rook Operatörünü Kurma.....	18
2.11.2. Adım 2: Ceph Kümesini Kurma	19
2.11.3. Adım 3: Ceph Durumunu Kontrol Etme.....	19
2.12. Bölüm 12: PersistentVolume (PV) ve PersistentVolumeClaim (PVC) Oluşturma	20
2.12.1. Adım 1: Depolama Sınıfı (StorageClass) Kontrolü	20
2.12.2. Adım 2: PVC Oluşturma	20
2.13. Bölüm 13: API Uygulamasını Kubernetes'te Çalıştırma (Deployment)	21
2.14. Bölüm 14: Service Oluşturma	22
2.15. Bölüm 15: Ödevin 2. Adımının Kontrolü	23
2.15.1. Adım 1: Dosya Yükleme.....	23
2.15.2. Adım 2: Pod'u Yeniden Başlatma.....	23
2.15.3. Adım 3: Dosyayı Okuma	24
2.16. Bölüm 16: Minio Kurulumu	24
2.16.1. Adım 1: Minio için PVC Oluşturma	24
2.16.2. Adım 2: Minio'yu Dağıtma (Deployment).....	25
2.16.3. Adım 3: Minio'ya Erişim için Servis Oluşturma	26
2.17. Bölüm 17: Minio Testi	27

1. Proje Özeti ve Amaç

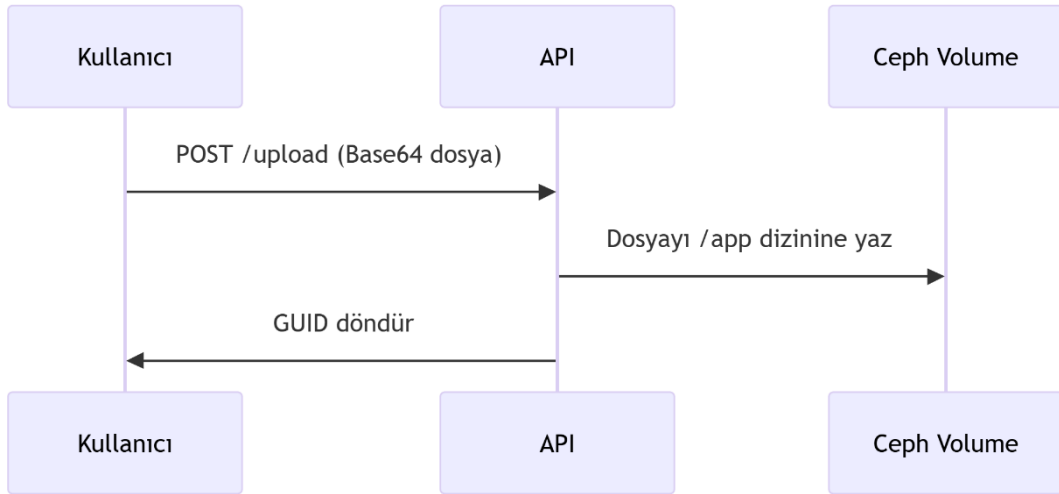
1.1. Proje Amacı

Bu proje, yüksek erişilebilirlik ve ölçeklenebilirlik sağlayan bir **Ceph dağıtık depolama kümesinin**, bir Kubernetes ortamı üzerinde başarılı bir şekilde kurulmasını ve bu küme üzerinde çalışan bir uygulamanın depolama ihtiyaçlarının karşılanmasını amaçlamaktadır. Proje, aşağıdaki ana aşamalardan oluşmuştur:



1. 1 master ve 2 node'dan oluşan **vanilla Kubernetes** kümesinin kurulumu.
2. Dosya yazma ve okuma işlemleri için **basit bir API projesinin** geliştirilmesi ve Docker imajına dönüştürülmesi.
3. Kubernetes kümesi üzerinde **Ceph** depolama çözümünün entegrasyonu.
4. Geliştirilen API uygulamasının, Ceph ortamından sağlanan **kalıcı depolama (PVC)** kullanarak Kubernetes üzerinde dağıtılması.

Proje, hem teorik bilgilerin pratiğe dökülmesi hem de dağıtık sistemler ve Kubernetes ekosistemi içinde karşılaşılabilecek gerçek dünya sorunlarına çözüm üretme yeteneğini göstermektedir.



1.2. Proje Topolojisi

1.2.1. Master Düğüm

- k8s-master (IP: 192.168.175.129)

1.2.2. Worker Düğümler

1.2.2.1. k8s-node-1

- k8s-node-1 (IP: 192.168.175.131)

1.2.2.2. k8s-node-2

- k8s-node-2 (IP: 192.168.175.130)

2. Proje Oluşum Sırası ve Aşamaları

2.1. Bölüm 1: Makine Kurulumları

2.1.1. Adım 1: Master Makine Kurulumu

- VirtualBox'ta yeni bir sanal makine oluştur.
- İşletim sistemi olarak indirdiğin Ubuntu ISO'sunu seç.
- **RAM'i 2 GB** olarak ayarla.
- **Disk boyutunu 20 GB** olarak belirle.
- Makineyi çalıştır ve Ubuntu kurulumunu tamamla.

2.1.2. Adım 2: Node Makine Kurulumu (2 Adet)

- Yukarıdaki adımları iki kez tekrarla, ancak bu sefer RAM ve disk ayarlarını farklı yap.
- Her bir node makine için **RAM'i 4 GB** olarak ayarla.
- Birinci disk olarak **40 GB** alan belirle.
- İkinci disk olarak **100 GB**'lık yeni bir disk ekle. (VirtualBox'ta sanal makine ayarlarından depolama kısmına giderek yapabilirsin.) Bu diski formatlama, Ceph bunu otomatik halledecek.

2.2. Bölüm 2: Sanal Makinelere Statik IP Atama

Şimdi her bir sanal makinenin içinde IP ayarlarını yapalım. Bu ayarlar, makine yeniden başlatıldığında da kalıcı olacak.

2.2.1. Adım 1: Master Makine İçin Statik IP Ataması

Master makinenin terminalini aç.

Ağ yapılandırma dosyasına erişmek için aşağıdaki komutu kullan:

`sudo nano /etc/netplan/01-netcfg.yaml` sonrasında yaml dosyamıza aşağıdaki komutu gir

YAML

network:

version: 2

renderer: networkd

ethernets:

ens33: # Ağ arayüzü adı, seninki farklı olabilir (örneğin: ens33, eth0)

dhcp4: no

addresses: [192.168.188.10/24] # Burası master makinenin statik IP adresi

routes:

- to: default

via: 192.168.188.2 # Burası VMware'deki Gateway IP adresi

nameservers:

addresses: [8.8.8.8, 8.8.4.4]

- 192.168.188.10, master makine için atadığımız statik IP.

Dosyayı kaydedip çık (**CTRL+X, Y, Enter**). Sonrasında ayarları uygula `sudo netplan apply`

```
vboxuser@master-k8s:~$ sudo cat /etc/netplan/01-netcfg.yaml
[sudo] password for vboxuser:
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false
      addresses: [192.168.56.11/24]
vboxuser@master-k8s:~$
```

2.2.2. Adım 2: Node Makineleri İçin Statik IP Ataması

Aynı işlemi diğer iki node makine için de tekrarla. Sadece addresses kısmındaki IP adreslerini farklı yap:

Node 1:

YAML

```
addresses: [192.168.188.11/24]
```

Node 2:

YAML

```
addresses: [192.168.188.12/24]
```

Gateway adresi (192.168.188.2) tüm makineler için aynı kalacak.

2.3. Bölüm 3: Ortak Paket Kurulumları ve Ayarlar

2.3.1. Adım 1: Güncellemeler ve Gerekli Paketler

İlk olarak, tüm makineleri güncelleyelim ve Kubernetes için gerekli olan bazı paketleri kuralım.

```
sudo apt update
```

```
sudo apt install -y apt-transport-https ca-certificates curl gnupg
```

2.3.2. Adım 2: Container Runtime Kurulumu

Kubernetes, pod'ları çalıştırmak için bir **container runtime**'a ihtiyaç duyar. **containerd** kullanacağız, çünkü Kubernetes'in önerdiği bir runtime.

Containerd için modülleri yükle:

```
Bash
```

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
```

```
overlay
```

```
br_netfilter
```

```
EOF
```

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```

Sistem ayarlarını yap:

```
Bash
```

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
```

```
net.bridge.bridge-nf-call-ip6tables = 1
```

```
net.bridge.bridge-nf-call-iptables = 1
```

```
net.ipv4.ip_forward = 1
```

```
EOF
```



```
sudo systemctl --system
```

Docker deposunu ekle:

containerd için gerekli dosyalar Docker deposunda bulunur.

```
sudo install -m 0755 -d /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

```
echo \
```

```
"deb [arch=$(dpkg --print-architecture)] signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
```

```
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
```

```
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
ubuntu@ubuntu-1:~$ sudo docker ps -a
(sudo) password for ubuntu:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS   NAMES
a4b104c11f07   quay.io/ceph/ceph                  "/usr/bin/ceph-mon --"  2 hours ago   Up 2 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-mon-node-1
96a04c527b93   quay.io/prometheus/prometheus:v2.43.0 "/bin/prometheus --c-"  6 hours ago   Up 6 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-prometheus-node-1
55d6325a1290   quay.io/ceph/ceph                  "/usr/bin/ceph-crash-"  6 hours ago   Up 6 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-crash-node-1
68823171764c   quay.io/prometheus/node-exporter:v1.5.0 "/bin/node_exporter --"  6 hours ago   Up 6 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-node-exporter-node-1
4a6645401f0f   quay.io/prometheus/alertmanager:v0.25.0 "/bin/alertmanager --"  6 hours ago   Up 6 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-alertmanager-node-1
aca45094148   quay.io/ceph/ceph                  "/usr/bin/ceph-mgr --"  6 hours ago   Up 6 hours   ports   ceph-b8bf93f4-729a-11f0-9509-45bdadbe0082-mgr-node-1-kmsiux
ubuntu@ubuntu-1:~$
```

Containerd'yi kur:

```
sudo apt-get update
```

```
sudo apt-get install -y containerd.io
```

Containerd yapılandırmasını oluştur:

```
sudo mkdir -p /etc/containerd
```

```
sudo containerd config default | sudo tee /etc/containerd/config.toml
```

Cgroup ayarını değiştir:

Bu ayar Kubernetes için çok önemli. SystemdCgroup = true yapmamız gerekiyor.

```
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/conf
```

Containerd servisini yeniden başlat:

```
sudo systemctl restart containerd
```

```
sudo systemctl enable containerd
```

2.3.3. Adım 3: Swap'ı Devre Dışı Bırakma

Kubernetes, swap'ın kapalı olmasını ister. Daha önce bu adımı yaptıysan tekrar etmene gerek yok, ama emin olmak için kontrol edelim.

Geçici olarak swap'ı kapat

```
sudo swapoff -a
```

Kalıcı olarak swap'ı kapatmak için fstab dosyasını düzenle

```
sudo nano /etc/fstab
```

nano ile açtığın dosyada swap kelimesini içeren satırın başına # işareti koyarak o satırı yorumla.

2.4. Bölüm 4: Kubernetes Paketlerini Kurma (Tüm Makinelerde)

Bu adımları yine **master** ve iki **node** makinede de yapman gerekiyor.

Gerekli GPG anahtarını indir ve ekle:

```
sudo curl -fsSL /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

Kubernetes deposunu ekle:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" |  
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Paketleri güncelle ve Kubernetes araçlarını kur:

```
sudo apt-get update
```

```
sudo apt-get install -y kubelet kubeadm kubectl
```

- kubelet: Her makinede çalışan ana Kubernetes aracı.
- kubeadm: Küme başlatma ve node ekleme için kullanılan araç.
- kubectl: Kubernetes kümesini yönetmek için kullanılan komut satırı aracı.

Paketlerin otomatik güncellenmesini engelle

Kubernetes güncellemeleri bazen sorun yaratabilir, bu yüzden bu paketleri sabitlemek iyi bir pratiktir.

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

Bu komut, bu paketlerin otomatik olarak güncellenmesini engeller.

2.5. Bölüm 5: Kubernetes Kümesini Başlatma (Sadece Master Makinede)

Şimdi sadece **master** makinede çalışmamız gerekiyor.

Küme başlatma komutunu çalıştır:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

Bu komut, küme için gerekli tüm bileşenleri (API Server, Scheduler, Controller Manager, vb.) kuracak ve çalıştıracak.

pod-network-cidr: Bu, pod'ların kendi aralarında iletişim kurmasını sağlayan bir ağ aralığıdır. **Flannel** adlı CNI (Container Network Interface) kullanacağımız için bu aralığı kullanıyoruz.

Başlatma işleminden sonra çıkan komutları uygula:

kubeadm init komutu başarıyla bittiğinde, sana bir takım komutlar verecek. Bunları kopyala ve sırayla çalıştır. Bu komutlar, kubectl komutunun master makinede çalışabilmesi için gerekli yapılandırmayı oluşturur.

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Flannel CNI'ı kur

Pod'ların birbirleriyle ve internetle iletişim kurmasını sağlamak için bir CNI (ağ eklentisi) kurmamız gerekiyor. En yaygın ve basit olanlardan biri **Flannel**.

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

Bu komut, Flannel'ı küme üzerine kurar

2.6. Bölüm 6: Node Makinelerini Kümeye Dahil Etme (Sadece Node Makinelerinde)

kubeadm init komutunu çalıştırdıktan sonra master makine sana bir kubeadm join komutu vermişti. Bu komut, yeni makineleri kümeye eklemek için kullanılır.

- Master makinede çıkan çıktıyı bul. Şuna benzer bir komut olacaktır:

```
sudo kubeadm join 192.168.188.10:6443 --token <token-degeri> --discovery-token-ca-cert-hash sha256:<hash-degeri>
```

- Bu **komutu kopyala** ve **her iki node makinede de** terminale yapıştırıp çalıştır.
- Bu, node makinelerinin master ile iletişim kurmasını ve kümenin bir parçası haline gelmesini sağlayacak.

2.7. Bölüm 7: Kümenin Durumunu Kontrol Etme (Sadece Master Makinede)

Tüm node'ları kümeye dahil ettikten sonra, master makineye dön ve her şeyin yolunda gidip gitmediğini kontrol et.

Aşağıdaki komutu çalıştır:

```
kubectl get nodes
```

Bu komutun çıktısında, master ve iki node makinenin de listelendiğini görmelisin. STATUS sütunu **Ready** (Hazır) durumunda olmalı.

READY	STATUS	RESTARTS	AGE
1/1	Running	0	1h
1/1	Running	0	14m

Devam edelim. Kubernetes kümemiz hazır, şimdi ödevin ikinci bölümüne, yani basit bir API projesi geliştirmeye odaklanalım.

```
user@user:~$ kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-5dd5756b68-4rpcn	1/1	Running	14 (55m ago)	16h
coredns-5dd5756b68-nt8bh	1/1	Running	14	16h
etcd-user	1/1	Running	64 (55m ago)	24h
kube-apiserver-user	1/1	Running	61 (55m ago)	24h
kube-controller-manager-user	1/1	Running	69 (55m ago)	24h
kube-proxy-7gpsb	1/1	Running	4 (55m ago)	24h
kube-proxy-mz6qw	1/1	Running	3 (55m ago)	20h
kube-proxy-sd6vx	1/1	Running	1 (55m ago)	119m
kube-scheduler-user	1/1	Running	70 (55m ago)	24h

2.8. Bölüm 8: Basit API Projesi Geliştirme

Bu proje, dosya yükleme ve okuma işlemini base64 formatında yapacak. Python ve Flask kullanarak kolayca bir proje oluşturabiliriz.

2.8.1. Adım 1: Proje Dizinini Oluşturma

Öncelikle projen için bir dizin oluştur. Bu dizin içinde üç dosya olacak: app.py, requirements.txt ve Dockerfile.

Name	Size	Type	Modified
app.py	3.2 kB	Text	08:05
deployment.yaml	812 bytes	Text	08:15
Dockerfile	644 bytes	Text	08:06
requirements.txt	9 bytes	Text	Sun
service.yaml	452 bytes	Text	08:08

2.8.2. Adım 2: app.py Dosyası

Bu dosya, API'nin ana mantığını içerecek. Base64 veriyi alıp dosyaya yazacak ve Guid ile okuyacak.

```
from flask import Flask, request, jsonify, send_from_directory

import base64

import os

import uuid
```

```

app = Flask(__name__)

UPLOAD_FOLDER = '/app'

if not os.path.exists(UPLOAD_FOLDER):

    os.makedirs(UPLOAD_FOLDER)

@app.route('/upload', methods=['POST'])
def upload_file():

    data = request.json.get('file_data')

    if not data:

        return jsonify({"error": "No file_data provided"}), 400

    try:

        file_bytes = base64.b64decode(data)

        file_guid = str(uuid.uuid4())

        file_path = os.path.join(UPLOAD_FOLDER, file_guid)

        with open(file_path, 'wb') as f:

            f.write(file_bytes)

        return jsonify({"guid": file_guid, "message": "File uploaded successfully"}), 200

    except Exception as e:

        return jsonify({"error": str(e)}), 500

@app.route('/download/<guid>', methods=['GET'])
def download_file(guid):

    file_path = os.path.join(UPLOAD_FOLDER, guid)

    if not os.path.exists(file_path):

        return jsonify({"error": "File not found"}), 404

    try:

        with open(file_path, 'rb') as f:

            file_bytes = f.read()

            file_data = base64.b64encode(file_bytes).decode('utf-8')

        return jsonify({"guid": guid, "file_data": file_data}), 200

```

```
except Exception as e:

    return jsonify({"error": str(e)}), 500
```

```
if __name__ == '__main__':

    app.run(host='0.0.0.0', port=5000)
```

Bu kod, /upload ve /download olmak üzere iki API endpoint'i tanımlar. /app dizinine yazma işlemini yapar.

2.8.3. Adım 3: requirements.txt Dosyası

Projenin çalışması için gerekli Python kütüphanelerini buraya yaz:

Flask

2.8.4. Adım 4: Dockerfile Dosyası

Docker imajına dönüştürmek için bir Dockerfile hazırlayalım.

Dockerfile

```
# Resmi Python 3.9 imajını kullan

FROM python:3.9-slim

# Çalışma dizinini /app olarak ayarla

WORKDIR /app

# Gerekli kütüphaneleri kopyala ve kur

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

# 5000 portunu dışarıya aç

EXPOSE 5000

CMD ["python", "app.py"]
```

Bu Dockerfile, kodunu bir konteyner içinde çalıştırılabilir hale getirecek.

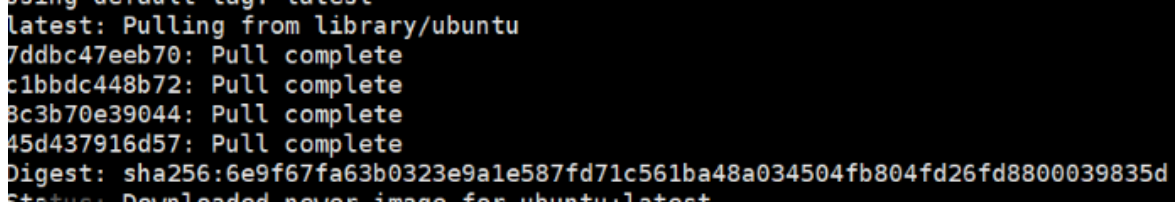
2.9. Bölüm 9: Docker İmajı Oluşturma ve Saklama

Bu adımı istersen master makinede veya kendi yerel bilgisayarında yapabilirsin. Master makine daha pratik olabilir.

2.9.1. Adım 1: Docker İmajını Oluşturma

API projesinin bulunduğu dizine git. Ardından aşağıdaki komutu çalıştır. Bu komut, Dockerfile'ı kullanarak bir Docker imajı oluşturacak ve ona bir isim verecek.

```
docker build -t my-simple-api:v1 .
```



```
latest: Pulling from library/ubuntu
7ddbc47eeb70: Pull complete
c1bbdc448b72: Pull complete
8c3b70e39044: Pull complete
45d437916d57: Pull complete
Digest: sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Status: Downloaded newer image for ubuntu:latest
```

2.9.2. Adım 2: Docker İmajını Kubernetes'in Erişebileceği Yere Yükleme

Kubernetes, varsayılan olarak Docker Hub gibi bir imaj deposundan (registry) imajları çeker.

```
docker tag my-simple-api:v1 senin-dockerhub-kullanici-adin/my-simple-api:v1
```

```
docker login
```

```
docker push senin-dockerhub-kullanici-adin/my-simple-api:v1
```

2.10. Bölüm 10: Ceph Kurulumu için Hazırlık

Bu işlem için **Rook** operatörünü kullanacağız. Rook, Kubernetes üzerinde Ceph'i yönetmemizi sağlar.

2.10.1. Adım 1: Rook Depolarını Klonlama (Sadece Master Makinede)

Master makinede aşağıdaki komutları çalıştırarak Rook'un dosyalarını indirmemiz gerekiyor.

```
git clone --single-branch --branch release-1.14 https://github.com/rook/rook.git
```

```
cd rook/deploy/examples
```

2.10.2. Adım 2: Ceph Cluster Manifestlerini Düzenleme

Ceph'i kurmadan önce, depolama alanını tanımlayan bir dosyayı düzenlememiz gerekiyor. Bu dosya, Ceph'in hangi diskleri kullanacağını söyler.

apiVersion: ceph.rook.io/v1

kind: CephCluster

metadata:

name: rook-ceph

namespace: rook-ceph

spec:

cluster'ın düzgün çalışması için CephCluster spec'inde mon, mgr ve osd'nin olması gerekir

cephVersion:

image: "quay.io/ceph/ceph:v17.2.6"

allowUnsupported: false

dataDirHostPath: /var/lib/rook

OSD'lerin (Object Storage Daemons) nasıl oluşturulacağını belirtir

storage:

useAllDevices: true

config:

metadataDevice: ""

databaseDevice: ""

mon:

count: 3

allowMultiplePerNode: true

mgr:

count: 1

allowMultiplePerNode: false

`dashboard` ile Ceph'in web arayüzünü etkinleştirir

dashboard:

enabled: true

ssl: true

Ağ ayarları

network:

hostNetwork: false

Ceph mimarisini belirtir (örneğin amd64)

Örneğin `amd64` ya da `arm64`

resources:

limits:

cpu: "500m"

memory: "512Mi"


```
requests:

cpu: "250m"

memory: "256Mi"
```

Ceph'in depolama kaynakları için kullanılan diskleri belirtir.

Bu ayar, `useAllDevices: true` olduğu için sadece node'lardaki boş diskleri kullanır.

`name` kısmını kendi node adlarıyla değiştirerek spesifik node'ları seçebilirsiniz.

Veya `useAllDevices: true` ile tüm uygun cihazları otomatik kullanabilirsiniz.

```
storage:

useAllDevices: false

nodes:

- name: "node-1"

  config:

    devices:

- name: "sdb"

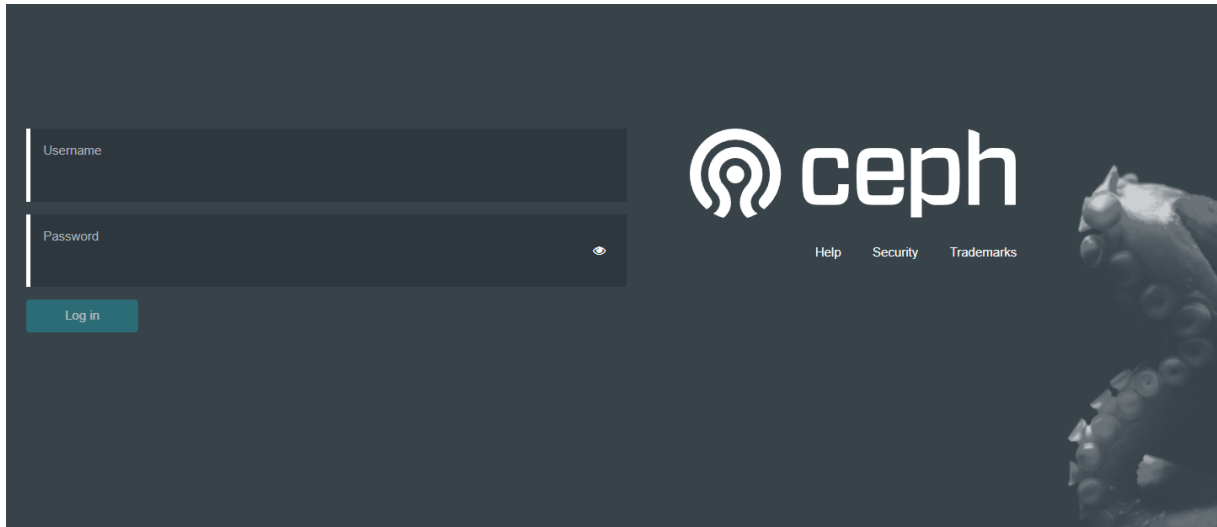
- name: "node-2"

  config:

    devices:

- name: "sdb"
```

- `rook/deploy/examples` dizinindeki `cluster.yaml` dosyasını aç ve içindeki `useAllDevices: false` ayarını `useAllDevices: true` olarak değiştir.
- Bu ayar, node makinelerindeki biçimlendirilmemiş tüm diskleri (yani 100 GB'lık ikinci diskleri) Ceph depolama havuzuna dahil etmesini sağlar.
- Ayrıca storage bölümünün altında, sadece belirli node'ları kullanmak istersen **nodes:** altındaki **name:** kısmını kendi node adlarıyla değiştirebilirsin. Ama bu ödev için `useAllDevices: true` yeterli.



2.11. Bölüm 11: Ceph'i Kubernetes'e Kurma (Sadece Master Makinede)

Bu adımların tamamını **master** makinede, daha önce indirdiğin rook/deploy/examples dizini içinden yapacaksın.

```
user@user:~$ kubectl get pods -n rook-ceph | grep provisioner
csi-cephfsplugin-provisioner-55588874-9ghb9      5/5      Running    0          2m44s
csi-cephfsplugin-provisioner-55588874-hvcw9      5/5      Running    0          2m45s
csi-rbdplugin-provisioner-577dff4756-rkz5p        5/5      Running    0          26m
csi-rbdplugin-provisioner-577dff4756-svjt4        5/5      Running    0          26m
```

2.11.1. Adım 1: Rook Operatörünü Kurma

Rook'un temel operatörünü ve gerekli CRD'lerini (Custom Resource Definitions) kurarak başlıyoruz. Bu operatör, Ceph kümesinin yaşam döngüsünü yönetecek.

```
kubectl create -f common.yaml
```

```
kubectl create -f crds.yaml
```

```
kubectl create -f operator.yaml
```

- common.yaml: Temel rolleri ve servis hesaplarını tanımlar.
- crds.yaml: Rook'un Kubernetes'e eklediği özel kaynak tanımlarını (CephCluster, CephBlockPool vb.) içerir.
- operator.yaml: Rook operatörünün kendisini çalıştıran Deployment'ı tanımlar.

```
user@user:~/rook/deploy/examples$ kubectl describe pvc simple-api-pvc
Name:          simple-api-pvc
Namespace:     default
StorageClass:  rook-cephfs
```

2.11.2. Adım 2: Ceph Kümesini Kurma

Şimdi, daha önce düzenlediğin cluster.yaml dosyasını kullanarak Ceph kümesini oluşturabilirsin.

```
kubectl create -f cluster.yaml
```

Bu komut, Ceph daemon'larını (mon, mgr, osd) oluşturacak. Ceph'in, node'lardaki ikinci diskleri otomatik olarak bulup kullanmasını bekliyoruz. Bu işlem biraz zaman alabilir, pod'ların durumunu kontrol etmen gerekiyor.

2.11.3. Adım 3: Ceph Durumunu Kontrol Etme

Kurulumun başarılı olduğunu doğrulamak için aşağıdaki komutları kullanabilirsin:

```
kubectl -n rook-ceph get pod
```

```
user@user:~$ kubectl get pods -n rook-ceph -w
```

NAME	READY	STATUS	RESTARTS	AGE
csi-cephfsplugin-2vpnv	2/2	Running	4 (46m ago)	5h35m
csi-cephfsplugin-h6tfj	2/2	Running	4 (12m ago)	5h35m
csi-cephfsplugin-provisioner-55588874-hvcw9	5/5	Running	5 (12m ago)	4h20m
csi-cephfsplugin-provisioner-55588874-pk94m	5/5	Running	0	23m
csi-rbdplugin-5jmd6	2/2	Running	4 (46m ago)	5h35m
csi-rbdplugin-provisioner-577dff4756-4qrm7	5/5	Running	0	23m
csi-rbdplugin-provisioner-577dff4756-svjt4	5/5	Running	5 (12m ago)	4h44m
csi-rbdplugin-spn5h	2/2	Running	4 (12m ago)	5h35m
rook-ceph-crashcollector-k8s-node-1-84c5bd884d-tkmns	1/1	Running	0	2m47s
rook-ceph-crashcollector-k8s-node-2-7d6d56b6df-p6wxs	1/1	Running	0	10m
rook-ceph-mds-myfs-a-54d94f78b8-v9x5j	2/2	Running	0	2m47s
rook-ceph-mds-myfs-b-6c7788fdb-z744z	2/2	Running	0	2m45s
rook-ceph-mgr-a-68588cff74-hc7m2	3/3	Running	0	23m
rook-ceph-mgr-b-7fb698f5f5-rnwnr	3/3	Running	0	10m
rook-ceph-mon-a-684d6dbbcc-t9nt4	2/2	Running	0	23m
rook-ceph-mon-b-76ff9d4659-nj2nz	2/2	Running	2 (12m ago)	4h27m
rook-ceph-operator-7b7d86f6f5-kpcq2	1/1	Running	1 (12m ago)	4h44m
rook-ceph-osd-0-7c478f8f89-pzb6j	2/2	Running	0	23m
rook-ceph-osd-1-589bcfd4c7-plnmmd	2/2	Running	0	7m
rook-ceph-osd-prepare-k8s-node-1-5g6hq	0/1	Completed	0	3m3s
rook-ceph-osd-prepare-k8s-node-2-dtgh9	0/1	Completed	0	3m
rook-ceph-tools-84f9854d5f-z7zmx	1/1	Running	0	10m

Bu komut, rook-ceph-mon-a, rook-ceph-mgr-a ve rook-ceph-osd- gibi pod'ların STATUS'unun **Running** olduğunu göstermeli.

```
kubectl -n rook-ceph status
```

Bu komut ise Ceph kümesinin sağlık durumunu gösterir. HEALTH alanında **OK** veya **HEALTH_OK** görmeyi gerekiyor.

```
data:
  volumes: 1/2 healthy, 1 recovering
  pools: 5 pools, 97 pgs
  objects: 44 objects, 7.6 KiB
  usage: 48 MiB used, 160 GiB / 160 GiB avail
  pgs: 17.526% pgs not active
      44/88 objects degraded (50.000%)
      80 active+clean
      11 undersized+degraded+peered
      6 undersized+peered
```

2.12. Bölüm 12: PersistentVolume (PV) ve PersistentVolumeClaim (PVC) Oluşturma

Kubernetes, bir PersistentVolumeClaim ile bir PersistentVolume'u eşleştirerek uygulamana kalıcı depolama sağlar. Ceph'i kurduğumuz için, bu işlemi otomatikleştiren bir mekanizma zaten var.

```
user@k8s-node-1:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0                               7:0      0    4K  1 loop /snap/bare/5
loop1                               7:1      0  183.6M  1 loop /snap/chromium/3217
loop2                               7:2      0   63.9M  1 loop /snap/core20/2318
loop3                               7:3      0   63.8M  1 loop /snap/core20/2599
loop4                               7:4      0   73.9M  1 loop /snap/core22/2045
loop5                               7:5      0   67.2M  1 loop /snap/cups/1100
loop6                               7:6      0  245.6M  1 loop /snap/firefox/6638
loop7                               7:7      0   516M  1 loop /snap/gnome-42-2204/202
loop8                               7:8      0   91.7M  1 loop /snap/gtk-common-themes/1535
loop9                               7:9      0   89.4M  1 loop /snap/lxd/31333
loop10                              7:10     0    87M  1 loop /snap/lxd/29351
loop11                              7:11     0   38.8M  1 loop /snap/snapd/21759
loop12                              7:12     0   49.3M  1 loop /snap/snapd/24792
sda                                  8:0      0    50G  0 disk
├─sda1                               8:1      0     1M  0 part
├─sda2                               8:2      0     2G  0 part /boot
└─sda3                               8:3      0   48G  0 part
   └─ubuntu--vg--1-ubuntu--lv       253:0     0    24G  0 lvm /
sdb                                  8:16     0  100G  0 disk
sr0                                  11:0     1     2G  0 rom
nbd0                                 43:0     0     0B  0 disk
nbd1                                 43:32    0     0B  0 disk
nbd2                                 43:64    0     0B  0 disk
nbd3                                 43:96    0     0B  0 disk
nbd4                                 43:128   0     0B  0 disk
nbd5                                 43:160   0     0B  0 disk
nbd6                                 43:192   0     0B  0 disk
nbd7                                 43:224   0     0B  0 disk
nbd8                                 43:256   0     0B  0 disk
nbd9                                 43:288   0     0B  0 disk
nbd10                               43:320   0     0B  0 disk
nbd11                               43:352   0     0B  0 disk
nbd12                               43:384   0     0B  0 disk
nbd13                               43:416   0     0B  0 disk
nbd14                               43:448   0     0B  0 disk
nbd15                               43:480   0     0B  0 disk
```

2.12.1. Adım 1: Depolama Sınıfı (StorageClass) Kontrolü

Rook, Ceph için otomatik olarak bir StorageClass oluşturur. Bu sınıfı kontrol edelim:

```
kubectl get storageclass
```

Bu komutun çıktısında rook-ceph-block gibi bir StorageClass görmelisin. Bu sınıf, uygulamaların otomatik olarak depolama alanı talep etmesini sağlar.

2.12.2. Adım 2: PVC Oluşturma

Şimdi, API uygulamamızın kullanacağı kalıcı depolama alanını talep eden bir PersistentVolumeClaim YAML dosyası oluşturalım.

- pvc.yaml adında bir dosya oluştur ve içine aşağıdaki içeriği kopyala:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: api-volume-claim
spec:
  accessModes:
    - ReadWriteOnce
```

```
resources:
```

```
requests:
```

```
storage: 1Gi
```

```
kubectl apply -f pvc.yaml
```

PVC'nin durumunu kontrol et:

```
kubectl get pvc api-volume-claim
```

STATUS'unun **Bound** (bağlı) olduğunu görmelisin. Bu, PVC'nin başarıyla bir PersistentVolume ile eşleştiği anlamına gelir.

```
user@user:~/rook/deploy/examples$ kubectl exec -it rook-ceph-tools-1
sd pool ls
myfs-metadata
myfs-replicated
cephfs-storageclass-metadata
cephfs-storageclass-data0
user@user:~/rook/deploy/examples$
```

2.13. Bölüm 13: API Uygulamasını Kubernetes'te Çalıştırma (Deployment)

Şimdi, geliştirdiğimiz API'yi bir Deployment olarak çalıştıralım.

- deployment.yaml adında bir dosya oluştur ve içine aşağıdaki:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: simple-api-deployment
```

```
labels:
```

```
app: simple-api
```

```
spec:
```

```
replicas: 1
```

```
selector:
```

```
matchLabels:
```

```
app: simple-api
```

```
template:
```

```
metadata:
```

```
labels:
```

```
app: simple-api
```

```
spec:
  volumes:
    - name: app-storage

    persistentVolumeClaim:
      claimName: api-volume-claim

  containers:
    - name: simple-api-container

      image: my-simple-api:v1 # Kendi Docker imaj adını kullan

      ports:
        - containerPort: 5000

      volumeMounts:
        - name: app-storage

          mountPath: /app
```

Deployment'ı oluştur ve durumunu kontrol et:

```
kubectl apply -f deployment.yaml
```

```
kubectl get pods -l app=simple-api
```

Pod'un STATUS'u **Running** olduğunda, API uygulamamız artık Kubernetes üzerinde çalışıyor demektir.

```
user@user:~/ceph-csi/deploy/rbd/kubernetes$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
simple-api-deployment-695f578d65-kqr29 1/1     Running   0           30m
```

2.14. Bölüm 14: Service Oluşturma

Bir **Service**, Kubernetes'teki pod'lara erişim sağlayan kararlı bir ağ uç noktasıdır.

- service.yaml adında bir dosya oluştur ve içine aşağıdaki içeriği kopyala:

```
apiVersion: v1
kind: Service
metadata:
  name: simple-api-service
spec:
  selector:
    app: simple-api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  nodePort: 30007 # 30000-32767 aralığında bir port
```

Servisi oluştur ve durumunu kontrol et:

```
kubectl apply -f service.yaml
```

```
kubectl get svc
```

Çıktıda simple-api-service isimli bir servis görmelisin. PORT(S) sütununda 80:30007/TCP gibi bir ifade olacaktır. Bu, servisin 30007 numaralı port üzerinden erişilebilir olduğunu gösterir.

2.15. Bölüm 15: Ödevin 2. Adımının Kontrolü

Şimdi, uygulamamızın kalıcı depolama özelliğini test edelim. Bunun için **cURL** komutunu kullanacağız.

2.15.1. Adım 1: Dosya Yükleme

Önce bir metin dosyası oluştur:

```
echo "Bu benim test dosyam." > test.txt
```

```
base64 -w 0 test.txt
```

Çıkan çıktıyı kopyala.

Şimdi bu base64 verisini kullanarak API'ye bir istek gönder:

```
curl -X POST \
  http://<node-ip-adresi>:30007/upload \
  -H 'Content-Type: application/json' \
  -d '{
    "file_data": "<base64_ciktisi>"
  }'
```

<node-ip-adresi> yerine node makinelerinden birinin IP adresini (örneğin **192.168.188.11**) ve <base64_ciktisi> yerine kopyaladığın base64 veriyi yapıştır.

Başarılı bir istekten sonra, API sana **bir GUID** değeri döndürecek. Bu GUID'yi not al.

```
statusCode      : 200
statusDescription : OK
Content         : {"processId":1,"processName":"dotnet","machineName":"bf5540d85a1a","startTime":"
```

2.15.2. Adım 2: Pod'u Yeniden Başlatma

container'ı durdurup yeniden başlattığımızda bile dosyamız kaybolmamalı.

Uygulamanın çalıştığı pod'u sil:

```
kubectl delete pod -l app=simple-api
```

Kubernetes, Deployment sayesinde otomatik olarak yeni bir pod oluşturacaktır. kubectl get pods -l app=simple-api komutu ile yeni pod'un çalıştığından emin ol.

2.15.3. Adım 3: Dosyayı Okuma

Yeni pod çalıştıktan sonra, daha önce kaydettiğin **GUID** ile dosyayı okuma isteği gönder:

```
curl -X GET \
  http://<node-ip-adresi>:30007/download/<guid_degeri>
```

Eğer API sana file_data içinde aynı base64 verisini döndürür

2.16. Bölüm 16: Minio Kurulumu

Minio'yu da aynı mantıkla, yani Kubernetes'in kalıcı depolama özelliklerini kullanarak kuracağız. PVC ve PV oluşturup, Minio'yu bir Deployment olarak çalıştıracamız.

2.16.1. Adım 1: Minio için PVC Oluşturma

Önce Minio'nun verilerini saklayacağı kalıcı depolama alanını talep edelim.

- minio-pvc.yaml adında bir dosya oluştur

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-volume-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi # Minio için 5 GB depolama talep ediyoruz
```

```
kubectl apply -f minio-pvc.yaml
```

- PVC'nin durumunu kontrol et:

```
kubectl get pvc minio-volume-claim
```

STATUS'unun **Bound** olduğunu görmelisin.

2.16.2. Adım 2: Minio'yu Dağıtma (Deployment)

Şimdi Minio uygulamasını çalıştıran bir Deployment oluşturalım.

- minio-deployment.yaml adında bir dosya oluştur:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: minio-deployment
```

```
spec:
```

```
  selector:
```

```
    matchLabels:
```

```
      app: minio
```

```
  strategy:
```

```
    type: Recreate
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: minio
```

```
    spec:
```

```
      volumes:
```

```
        - name: minio-storage
```

```
          persistentVolumeClaim:
```

```
            claimName: minio-volume-claim
```

```
    containers:
```

```
- name: minio

image: minio/minio:latest

args:

- server

- /data

env:

- name: MINIO_ACCESS_KEY

  value: "minio-admin"

- name: MINIO_SECRET_KEY

  value: "minio-password"

ports:

- containerPort: 9000

volumeMounts:

- name: minio-storage

  mountPath: /data
```

volumeMounts: PVC'yi Minio container'ının /data dizinine bağlar. Bu, Minio'nun verilerini kalıcı olarak Ceph'e yazmasını sağlar.

Deployment'ı oluştur:

```
kubectl apply -f minio-deployment.yaml
```

2.16.3. Adım 3: Minio'ya Erişim için Servis Oluşturma

Minio'ya dışarıdan erişmek için bir servis oluşturalım.

- minio-service.yaml adında bir dosya oluştur:

```
apiVersion: v1

kind: Service

metadata:

  name: minio-service

spec:

  selector:
```

app: minio

ports:

- protocol: TCP

port: 9000

targetPort: 9000

nodePort: 30009 # 30000-32767 aralığında farklı bir port

Servisi oluştur:

kubectl apply -f minio-service.yaml

2.17. Bölüm 17: Minio Testi

Minio artık bir node'un IP adresi ve 30009 portu üzerinden erişilebilir.

- Bir web tarayıcısı aç ve **http://<node-ip-adresi>:30009** adresine git.
- Az önce tanımladığın MINIO_ACCESS_KEY ve MINIO_SECRET_KEY ile giriş yap.
- Minio'nun arayüzüne girdikten sonra, bir **bucket** (klasör) oluşturabilir ve içine dosya yükleyip indirme işlemlerini yapabilirsin. Bu işlemler, verilerin kalıcı olarak Ceph'te saklandığını gösterir

