Kubernetes Persistent Volumes (PV), PersistentVolumeClaims (PVC) ve StorageClass Rehberi

Gerçek dünya senaryoları, kubectl komutları, YAML örnekleri ve pratik ipuçları

Tarih: 13 Şubat 2026

Bu doküman, Kubernetes üzerinde kalıcı depolamanın (persistent storage) temel yapı taşları olan PV, PVC ve StorageClass kavramlarını pratik ve uygulanabilir senaryolarla anlatır. Amaç: üretim ortamında en sık karşılaşılan ihtiyaçlara doğru çözümü seçebilmen, YAML tarafında doğru alanları kullanabilmen ve sorun çıktığında hızlıca teşhis edebilmen.

Kavram	Ne yapar?	Senaryoda karşılığı
PV (PersistentVolume)	Cluster düzeyinde bir depolama kaynağı tanımlar.	NFS share, iSCSI disk, cloud disk, local disk gibi fiziksel/soyut storage.
PVC (PersistentVolume Claim)	Bir uygulamanın depolama talebidir (boyut, access mode).	Pod/Deployment/StatefulSet "şu kadar alan istiyorum" der.
StorageClass	Dinamik PV üretme kuralları ve varsayılanlarıdır.	PVC geldiğinde otomatik disk/NFS volume açma (CSI provisioner ile).

Hızlı seçim kuralı: Uygulamaların çoğunda hedef, PVC + StorageClass ile dinamik provisioning yapmaktır. Statik PV genelde mevcut bir storage paylaşımın (ör. hazır NFS dizini) varsa tercih edilir.

1. Temel Kavramlar ve Akış

Bağlama (Binding) akışı: PVC oluşturulur → Kubernetes uygun bir PV bulur (veya StorageClass ile üretir) → PVC Bound olur → Pod PVC'yi mount eder.

• PV cluster kaynağıdır. Admin veya CSI provisioner tarafından oluşturulur.

• PVC uygulama isteğidir. Uygulama sahibi genellikle PVC'yi yazar.

• StorageClass hangi provisioner ve parametrelerle volume açılacağını belirler.

1.1 PV'de sık kullanılan alanlar

• capacity.storage: PV'nin boyutu (ör. 20Gi).

• accessModes: Volume nasıl mount edilir? (RWO/RWX/ROX/RWOP).

• persistentVolumeReclaimPolicy: PVC silinince PV ne olacak? (Retain / Delete).

• storageClassName: Bu PV hangi StorageClass'a ait sayılır (dinamik/statik eşleşmede önemli).

1.2 PVC'de sık kullanılan alanlar

• resources.requests.storage: İstenen minimum kapasite.

• accessModes: Beklenen erişim modu (PV ile uyumlu olmalı).

• storageClassName: Dinamik provisioning için hangi StorageClass kullanılacak (boşsa default SC devreye girer).

• volumeName: Belirli bir PV'ye sabitlemek istersen (statik senaryolar).

1.3 StorageClass'ta sık kullanılan alanlar

• provisioner: CSI sürücüsü (örn: nfs.csi.k8s.io, rook-ceph.rbd.csi.ceph.com, ebs.csi.aws.com).

• reclaimPolicy: Dinamik PV'lerin varsayılan geri kazanım davranışı.

• volumeBindingMode: Immediate mı, WaitForFirstConsumer mı? (özellikle topology/zone ve local PV'de kritik).

• allowVolumeExpansion: PVC boyut büyütmeye izin ver.

• parameters: Provisioner'a özel ayarlar (filesystem type, pool, share, iops vb.).

AccessModes kısa rehber

AccessMode	Anlamı	Ne zaman? (pratik)
ReadWriteOnce (RWO)	Tek node üzerinde RW mount.	En yaygın: DB, cache, tek node state.
ReadOnlyMany (ROX)	Birden çok node RO mount.	Paylaşımlı salt-okunur içerik.
ReadWriteMany (RWX)	Birden çok node RW mount.	Paylaşımlı dosya: upload, ortak medya.
ReadWriteOncePod (RWOP)	Sadece tek Pod RW mount.	Tek Pod garantisi gereken durumlar.

2. Hangi Durumlarda PV/PVC/StorageClass Faydalı?

Pod ömrü geçicidir. Pod silinip yeniden yaratıldığında dosyalar kaybolur. Kalıcı depolama ile veri Pod yaşam döngüsünden ayrılır.

2.1 Sık görülen gerçek dünya ihtiyaçları

• Veritabanı: verinin Pod restart'larından etkilenmemesi.

• Stateful uygulamalar: registry, artifact store, queue, Git server.

• Paylaşımlı dosyalar: web upload klasörü (RWX).

• Log/rapor arşivleme: kısa süreli saklama veya dışarı taşıma pipeline'ı.

2.2 Yanlış kullanım örnekleri (kaçın)

• Her şeye RWX verme: performans ve maliyet etkisi olur.

• Statik PV'yi her yerde kullanma: çoğu yerde dinamik provisioning daha az operasyonel yük getirir.

• PVC'ye gereksiz büyük storage isteme: kapasite planlamasını zorlaştırır.

3. kubectl ile Yönetim ve Teşhis Komutları

Üretimde en çok kullanılan ve sorun çözdüren komutlar:

3.1 Keşif ve durum kontrolü

```
kubectl get storageclass
kubectl describe storageclass <sc-name>

kubectl get pv
kubectl get pvc -A
kubectl -n <ns> get pvc

kubectl get pvc -A --field-selector=status.phase=Pending
kubectl get pv --sort-by=.spec.capacity.storage
```

3.2 Sorun teşhisi için

```
kubectl -n <ns> describe pvc <pvc-name>     # Events kısmı kritik
kubectl describe pv <pv-name>
kubectl -n <ns> get events --sort-by=.lastTimestamp | tail -n 80
kubectl -n <ns> describe pod <pod-name>    # Mount/Attach hataları
```

3.3 Gelişmiş: patch / jsonpath / wait

```
kubectl patch storageclass fast-ssd \
  -p '{"metadata":{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'

kubectl -n <ns> get pvc <pvc-name> -o jsonpath='{.spec.volumeName}{"\n"}'

kubectl -n <ns> wait --for=jsonpath='{.status.phase}'=Bound pvc/<pvc-name> --timeout=120s
```

4. StorageClass Detayları ve Kritik Alanlar

StorageClass doğru seçilmezse PVC Pending kalır veya yanlış disk türü ile provision edilir.

4.1 reclaimPolicy: Delete vs Retain

• Delete: PVC silininces dinamik PV ve alttaki disk de silinir (test/dev).

• Retain: PVC silinse bile alttaki disk korunur (kritik veri). Cleanup operasyonu gerekir.

4.2 volumeBindingMode: Immediate vs WaitForFirstConsumer

• Immediate: PVC oluşunca hemen PV ayrılır.

• WaitForFirstConsumer: Pod schedule edilene kadar bağlamaz. Zone/topology ve local disk için hayati.

4.3 allowVolumeExpansion

• StorageClass allowVolumeExpansion true ise PVC boyutu artırılabilir (driver desteklemeli).

• Büyütme sonrası dosya sistemi genişletme gerekebilir.

5. Gerçek Dünya YAML Örnekleri

5.1 Dinamik provisioning: StorageClass + PVC + Deployment

```
# storageclass-generic.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: generic-fast
provisioner: example.csi.driver
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
parameters:
  type: "fast"
  fsType: "ext4"

# pvc-app-data.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: app-data
  namespace: app
spec:
  accessModes: ["ReadWriteOnce"]
  storageClassName: generic-fast
  resources:
    requests:
      storage: 20Gi

# deployment-app.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: uploader
  namespace: app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: uploader
  template:
    metadata:
      labels:
        app: uploader
    spec:
      securityContext:
        fsGroup: 10001
      containers:
      - name: uploader
        image: nginx:latest
        volumeMounts:
        - name: data
          mountPath: /data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: app-data
```

Dikkat:

- replicas > 1 ise RWO ile farklı node’larda aynı PVC aynı anda mount edilemez.
- Paylaşımlı yazma gerekiyorsa RWX sağlayan storage (NFS/CephFS gibi) kullan.

5.2 StatefulSet: volumeClaimTemplates ile her Pod’a ayrı PVC

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: pg
  namespace: data
spec:
  serviceName: pg
  replicas: 3
  selector:
    matchLabels:
      app: pg
  template:
    metadata:
      labels:
        app: pg
    spec:
      containers:
      - name: postgres
        image: postgres:16
        env:
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: pg-secret
              key: password
        volumeMounts:
        - name: pgdata
          mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
  - metadata:
      name: pgdata
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: generic-fast
      resources:
        requests:
          storage: 50Gi
```

5.3 Statik PV: Mevcut NFS paylaşımı (RWX)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-shared
spec:
  capacity:
    storage: 200Gi
  accessModes: ["ReadWriteMany"]
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs-static
  nfs:
    server: 10.0.10.50
    path: /exports/shared

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-shared
  namespace: app
spec:
  accessModes: ["ReadWriteMany"]
  storageClassName: nfs-static
  resources:
    requests:
      storage: 50Gi
```

6. Sık Hatalar ve Çözüm Rehberi

PVC Pending ise önce describe ve Events.

Belirti	Muhtemel neden	Çözüm ipucu
PVC Pending	StorageClass yok / yanlış isim	PVC storageClassName düzelt, default SC var mı kontrol et.
PVC Pending	CSI/provisioner çalışmıyor	CSI controller/node plugin podlarını ve loglarını kontrol et.
Mount hatası	Permission / fsGroup / export izinleri	Pod securityContext fsGroup, NFS izinleri, SELinux/APPArmor.

```
kubectl -n <ns> describe pvc <pvc>
kubectl describe storageclass <sc>
kubectl get pods -A | grep -i csi
kubectl -n <ns> get events --sort-by=.lastTimestamp | tail -n 80
kubectl -n <ns> describe pod <pod>
```

7. Best Practices (Üretim İçin)

• Workload’a göre doğru access mode seç (RWO vs RWX).

• Kritik verilerde Retain + backup/snapshot planı düşün.

• Permission problemleri için fsGroup ve non-root çalışma yaklaşımı kullan.

• Prod/Test için farklı StorageClass (fast/cheap/rwx) standardı oluştur.

8. Kısa Özet

PV depolama kaynağıdır, PVC uygulama talebidir, StorageClass ise dinamik provisioning kurallarıdır. Üretimde en yaygın yaklaşım: PVC + StorageClass ile dinamik provisioning.

9. Sık Sorulan Sorular

9.1 CSI ne demek?

CSI (Container Storage Interface), Kubernetes'in farklı storage sistemleriyle (cloud disk, Ceph, NFS, vSphere vb.) standart bir arayüz üzerinden konuşmasını sağlayan spesifikasyondur. PVC oluşturduğunda volume oluşturma (provisioning), node'a attach/detach ve Pod içine mount/unmount işlemleri çoğunlukla CSI driver tarafından yapılır.

• StorageClass içindeki provisioner alanı genellikle CSI driver'ı temsil eder.

• Cluster'da genelde controller (Deployment/StatefulSet) ve node plugin (DaemonSet) bileşenleri görürsün.

9.2 Bir PVC'yi birden fazla Pod kullanabilir mi?

Evet, ama accessModes ve storage'ın gerçek kapasitesi belirleyicidir.

• RWO: Aynı PVC aynı anda birden fazla Pod tarafından mount edilebilir, ancak pratikte tek node sınırı vardır. Farklı node'larda aynı anda kullanılamaz.

• RWX: Birden fazla Pod, farklı node'larda aynı anda kullanabilir (NFS/CephFS gibi RWX sağlayan storage gerekir).

• RWOP: Aynı anda sadece tek Pod kullanabilir.

• Aynı PVC'ye çoklu yazma gerekiyorsa uygulamanın dosya kilitleme/cakışma yönetimini düşün (dosya bozulması riski).

9.3 Bir PV'yi birden fazla PVC kullanabilir mi?

Genel kural: Hayır. Kubernetes'te bir PV normalde aynı anda sadece tek bir PVC'ye Bound olabilir (PV-PVC ilişkisi 1:1).

• Paylaşım ihtiyacında genelde çözüm: tek PVC + RWX + çok Pod veya aynı storage altyapısından ayrı PV'ler tanımlamaktır.

• Snapshot/clone ile yeni PVC'ler oluşturmak, aynı PV'yi paylaşmak değil, yeni volume üretmektir.