Kubernetes Job ve CronJob

Bu doküman; Job ve CronJob kaynaklarının ne işe yaradığını, hangi durumlarda faydalı olduğunu, kritik alanların (completions/parallelism/backoffLimit, schedule/timeZone/concurrencyPolicy vb.) pratikte nasıl çalıştığını ve gerçek dünyaya uygun YAML + kubectl örneklerini içerir.

> Kısa özet: Job bir işi bir kez (veya belirli sayıda tamamlanana kadar) çalıştırır. CronJob ise bir Job'u zamanlayarak periyodik üretir.

1. Kubernetes Job

Job; kısa süreli, bitince sona eren ve başarı/başarısızlık durumu takip edilen işler için kullanılır.

1.1 Job ne zaman kullanılır?

- Veritabanı migration (deploy öncesi bir kez)
- Tek seferlik backfill/veri düzeltme (geçmiş verileri doldurma)
- Batch işleme (thumbnail üretimi, rapor üretimi, video transcode)
- Export/Import (CSV dışa aktarma, arşivleme)
- Ad-hoc bakım işleri (index rebuild vb.)

1.2 Job alanları: completions, parallelism, backoffLimit

Alan	Ne işe yarar	Pratik yorum
completions: 20	Job'un toplamda 20 başarılı tamamlanma görmesini ister.	İşi 20 parçaya böldüğünüzde (shard) her parça 1 completion olabilir.
parallelism: 5	Aynı anda en fazla 5 pod çalıştırır.	Cluster kaynaklarını kontrollü kullanmak için önemlidir.
backoffLimit: 3	Bir pod başarısız olursa Job toplamda en fazla 3 kez yeniden dener.	Sonsuz retry'ı engeller; hatalı konfigürasyonlarda cluster'ı şişirmez.

İpucu: completions/parallelism kullandığınız işlerde uygulama tarafında genelde shard mantığı gerekir. Aksi halde aynı veriyi birden çok pod işlemeye çalışabilir.

1.3 Gerçek dünyaya uygun Job YAML örneği: Paralel batch (thumbnail üretimi)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: thumb-gen
  namespace: media
spec:
  completions: 20
  parallelism: 5
  backoffLimit: 3
  activeDeadlineSeconds: 1800
  ttlSecondsAfterFinished: 86400
  template:
    spec:
```

```
      restartPolicy: Never
      containers:
      - name: worker
        image: registry.example.com/media/thumb-worker:2.3.1
        env:
        - name: TOTAL_SHARDS
          value: "20"
        - name: SHARD_ID
          valueFrom:
            fieldRef:
              # Not: completion index her cluster'da aktif olmayabilir.
              fieldPath: metadata.annotations['batch.kubernetes.io/job-completion-index']
        args: ["--shard=$(SHARD_ID)", "--total=$(TOTAL_SHARDS)"]
        resources:
          requests:
            cpu: "200m"
            memory: "256Mi"
          limits:
            cpu: "1"
            memory: "1Gi"
```

1.4 Job için kubectl komutları (temel + karmaşık örnekler)

```
# Job oluştur (YAML ile)
kubectl -n media apply -f thumb-gen-job.yaml

# Job durumunu izle
kubectl -n media get job thumb-gen -w

# Job'un podlarını listele (label selector)
kubectl -n media get pods -l job-name=thumb-gen -o wide

# Job loglarını takip et (en pratik yol)
kubectl -n media logs -f job/thumb-gen

# Job tamamlanmasını bekle
kubectl -n media wait --for=condition=complete job/thumb-gen --timeout=30m

# Başarısızlık teşhisi: describe + failed pod logları
kubectl -n media describe job thumb-gen
kubectl -n media get pods -l job-name=thumb-gen --field-selector=status.phase=Failed
kubectl -n media logs <FAILED_POD_NAME>

# Job'u yeniden çalıştırma (eskiyi silip aynı manifest'i tekrar uygula)
kubectl -n media delete job thumb-gen
kubectl -n media apply -f thumb-gen-job.yaml

# Job'u patch ederek parallelism'i çalışma anında artır/azalt
kubectl -n media patch job thumb-gen -p '{"spec":{"parallelism":3}}'

# JSONPath ile tamamlanan sayıyı çek (otomasyonlarda kullanılır)
kubectl -n media get job thumb-gen -o jsonpath='{.status.succeeded}{" / "}{.spec.completions}{"\n"}'

# Job olaylarını tarihe göre görmek (debug)
kubectl -n media get events --sort-by=.lastTimestamp | tail -n 30
```

1.5 Job için dikkat edilmesi gerekenler

- Pod içinde restartPolicy: Never kullan. Retry işini Job yönetir.
- Retry ve takılma kontrolü için: backoffLimit + activeDeadlineSeconds kullan.

- Bitmiş jobların cluster'da birikmemesi için ttlSecondsAfterFinished ekle.
- Batch işler genelde kaynak tüketir: mutlaka requests/limits tanımla.
- İş tekrar çalışırsa veri bozulmamalı: mümkünse idempotent tasarla.
- Secret/ConfigMap ile yapılandır; şifreleri image içine koyma.

2. Kubernetes CronJob

CronJob; belirli zamanlarda otomatik olarak Job oluşturur. Örn: her gece backup, haftalık temizlik, saatlik rapor.

2.1 CronJob ne zaman kullanılır?

- Gece DB yedeği (backup) alma
- Log/metric housekeeping (eski veriyi silme, retention)
- Periyodik rapor üretme ve mail/Slack gönderme
- Cache temizleme, indeks optimize etme gibi rutin bakım işleri
- Belirli aralıklarla entegrasyon senkronizasyonu (3. parti API sync)

2.2 CronJob alanları: schedule, timeZone, concurrencyPolicy, history limits

Alan	Ne işe yarar	Pratik yorum
schedule: "0 2 * * *"	Cron ifadesi. Her gün 02:00 çalıştırır.	timeZone ile birlikte kullanmak sürprizleri azaltır.
timeZone: Europe/Istanbul	Zamanlamayı belirtilen timezone'a göre hesaplar.	Çok bölgeli ortamlarda saat kaymalarını engeller.
concurrencyPolicy: Forbid	Önceki job bitmeden yenisini başlatmaz.	Backup gibi üst üste binmesi riskli işler için idealdir.
successfulJobsHistoryLimit: 3	Son 3 başarılı job kaydını tutar.	Job objesi birikimini sınırlar.
failedJobsHistoryLimit: 3	Son 3 başarısız job kaydını tutar.	Hata teşhisinde yeterli geçmiş bırakır.

2.3 Cron ifadesi (cron expression) tanımı ve örnekler

Cron formatı: dakika saat ayın-günü ay haftanın-günü. Örn: 0 2 * * * = her gün 02:00.

Cron	Anlamı
0 2 * * *	Her gün 02:00
*/5 * * * *	Her 5 dakikada bir
0 */6 * * *	Her 6 saatte bir (00:00, 06:00, 12:00, 18:00)
30 3 * * 0	Her pazar 03:30
15 9 1 * *	Her ayın 1'inde 09:15

0 0 1 1 *	Her yıl 1 Ocak 00:00

Not: CronJob zamanlamasında controller'ın saat senkronu (NTP) önemlidir. timeZone kullanımı, beklenmedik saat kaymalarını azaltır.

2.4 Gerçek dünyaya uygun CronJob YAML örneği: Gece DB backup

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: nightly-db-backup
  namespace: ops
spec:
  schedule: "0 2 * * *"          # her gün 02:00
  timeZone: "Europe/Istanbul"
  concurrencyPolicy: Forbid
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 3
  jobTemplate:
    spec:
      ttlSecondsAfterFinished: 86400
      backoffLimit: 2
      template:
        spec:
          restartPolicy: Never
          containers:
          - name: backup
            image: postgres:16
            env:
            - name: PGPASSWORD
              valueFrom:
                secretKeyRef:
                  name: app-db-secret
                  key: password
            command: ["/bin/sh","-c"]
            args:
              - |
                set -e
                pg_dump -h postgres.app.svc -U appuser appdb | gzip > /tmp/appdb.sql.gz
                # upload adımı: örn minio(mc) veya aws-cli ile S3'e yükle
                echo "backup created:" && ls -lh /tmp/appdb.sql.gz
```

2.5 CronJob için kubectl komutları (temel + karmaşık örnekler)

```
# CronJob oluştur (YAML ile)
kubectl -n ops apply -f nightly-db-backup-cronjob.yaml

# CronJob'ları listele
kubectl -n ops get cronjob

# Son schedule zamanını ve suspend durumunu hızlı gör
kubectl -n ops get cronjob nightly-db-backup -o wide

# CronJob'u geçici durdur (suspend)
kubectl -n ops patch cronjob nightly-db-backup -p '{"spec":{"suspend":true}}'

# Tek seferlik manuel tetikleme: CronJob'dan anlık Job üret
kubectl -n ops create job --from=cronjob/nightly-db-backup nightly-db-backup-manual-$(date +%s)

# CronJob'un oluşturduğu Job'ları bul (label selector genelde cronjob-name ile gelir)
kubectl -n ops get jobs -l cronjob-name=nightly-db-backup
```

```
# En son job'u bulup loglarını izle (script'lerde işe yarar)
LAST_JOB=$(kubectl -n ops get jobs -l cronjob-name=nightly-db-backup   --sort-by=.metadata.creationTimestamp -o
kubectl -n ops logs -f $LAST_JOB

# Belirli zaman aralığında event'leri gör (debug)
kubectl -n ops get events --sort-by=.lastTimestamp | tail -n 50

# Dry-run ile manifest üret (CI/CD'de sık kullanılır)
kubectl -n ops create cronjob demo --image=busybox --schedule="*/10 * * * *" --dry-run=client -o yaml
```

2.6 CronJob için dikkat edilmesi gerekenler

- Backup gibi işler için concurrencyPolicy: Forbid genellikle en güvenlisidir.
- Çok gecikmiş çalıştırmalar istenmiyorsa startingDeadlineSeconds eklemeyi düşün.
- Biriken job'ları sınırlamak için history limitleri ve/veya jobTemplate altında ttlSecondsAfterFinished kullan.
- Zamanlama kritikse timeZone kullan ve controller saat senkronunu (NTP) doğrula.
- CronJob sadece Job üretir: asıl iş mantığı yine Job pod'unun içinde çalışır.

Ek: Sık karşılaşılan sorunlar

- Job podları Pending kalıyorsa: kaynak yetersizliği, nodeSelector/affinity, taint/toleration veya image pull hatalarını kontrol et.
- CronJob çalışmıyorsa: schedule ifadesi, suspend=true olup olmadığı, controller logları ve RBAC yetkilerini kontrol et.
- Log toplamak için: job/ üzerinden logs almak en pratik yoldur.