CKAD Ders Dökümanı #13 — Multi-Container Pods

Dil: Türkçe • CKAD Kategorisi: Application Design and Build (20%) • Güncelleme: sidecar/ambassador/adapter desenlerine örnekler eklendi

1) Konu Anlatımı

Multi-container Pod, birden fazla container'ın aynı Pod içinde birlikte çalışmasıdır. Container'lar aynı network namespace'i paylaşır (yani localhost ile birbirine erişebilir) ve isterlerse aynı volume'u paylaşabilir.

CKAD'de bu konu genellikle: tek Pod içinde iki container'ı çalıştırma, aralarında emptyDir ile dosya paylaşma, doğru container'a logs/exec atabilme ve desenleri (patterns) anlayabilme şeklinde çıkar.

En yaygın 3 desen: Sidecar, Ambassador, Adapter

Sidecar: Uygulamanın yanına "yardımcı" container eklemek (log shipper, agent, metrics exporter, proxy). Ambassador: Uygulama adına dış dünyaya konuşan bir "proxy/gateway" container (app sadece localhost ile konuşur). Adapter: Uygulamanın ürettiği veri formatını başka bir formata dönüştüren container (log/metric/format çevirme).

1.1 Aydınlatıcı Örnekler (Desen Bazlı)

A) Sidecar — Log/Agent örneği (dosyaya yaz → sidecar tail'lesin)

Senaryo: Uygulaman logları dosyaya yazıyor (stdout'a değil). Sidecar container bu dosyayı tail -f yaparak stdout'a basar ve böylece merkezi log toplayıcılar (Loki/Fluent Bit vb.) logları yakalayabilir. CKAD'de amaç, shared volume mantığını göstermek.

```
apiVersion: v1
kind: Pod
metadata:
  name: sidecar-log
spec:
  volumes:
  - name: shared-logs
    emptyDir: {}
  containers:
  - name: app
    image: busybox:1.36
    command: ["sh","-c"]
    args:
      - |
        i=0;
        while true; do
          echo "$(date) app-log-$i" >> /var/log/app/app.log;
          i=$((i+1));
          sleep 2;
        done
    volumeMounts:
```

```
    - name: shared-logs
      mountPath: /var/log/app
  - name: log-sidecar
    image: busybox:1.36
    command: ["sh","-c"]
    args: ["tail -n+1 -f /var/log/app/app.log"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log/app
```

Test:

```
kubectl logs sidecar-log -c log-sidecar
kubectl exec -it sidecar-log -c app -- sh -c "tail -n 3 /var/log/app/app.log"
```

B) Ambassador — Proxy örneği (app → localhost proxy → dış servis)

Senaryo: Uygulama dış servise (ör. https://example.org) gitmek istiyor ama TLS/sertifika/kurum proxy ayarları veya gözlemlenebilirlik (tracing) sebebiyle tüm çıkışların bir proxy üzerinden geçmesi isteniyor. Uygulama sadece localhost:3128 gibi bir adrese konuşur; proxy container dış dünyaya çıkar.

Aşağıdaki örnek, sidecar olarak bir HTTP proxy (squid) koyar. App container curl ile dışarı çıkarken proxy kullanır. (Gerçek hayatta Envoy/HAProxy/Nginx de kullanılabilir.)

```
apiVersion: v1
kind: Pod
metadata:
  name: ambassador-proxy
spec:
  containers:
  - name: proxy
    image: ubuntu/squid:latest
    ports:
    - containerPort: 3128
  - name: app
    image: curlimages/curl:8.6.0
    env:
    - name: http_proxy
      value: "http://127.0.0.1:3128"
    - name: https_proxy
      value: "http://127.0.0.1:3128"
    command: ["sh","-c"]
    args: ["curl -I https://example.org; sleep 3600"]
```

Test:

```
kubectl logs ambassador-proxy -c app
kubectl exec -it ambassador-proxy -c app -- sh -c "env | grep -i proxy"
```

C) Adapter — Format çevirme örneği (JSON → line format)

Senaryo: Uygulama bir dosyaya JSON satırları yazıyor, fakat başka bir sistem bu veriyi 'basit satır formatında' bekliyor. Adapter container, shared volume’daki JSON’u okuyup dönüştürülmüş bir çıktı dosyasına yazar (veya stdout’a basar).

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: adapter-format
spec:
  volumes:
  - name: shared
    emptyDir: {}
  containers:
  - name: app
    image: busybox:1.36
    command: ["sh","-c"]
    args:
      - |
        echo '{"level":"info","msg":"started","ts":"'"$(date)"'"}' >> /data/events.jsonl;
        echo '{"level":"warn","msg":"slow","ts":"'"$(date)"'"}' >> /data/events.jsonl;
        sleep 3600
    volumeMounts:
    - name: shared
      mountPath: /data
  - name: adapter
    image: busybox:1.36
    command: ["sh","-c"]
    args:
      - |
        # Basit bir "adapter": JSON'dan level ve msg bilgisini grep/sed ile ayıkla (demo amaçl
        while true; do
          if [ -f /data/events.jsonl ]; then
            tail -n+1 /data/events.jsonl | sed -n 's/.*"level":"\([^"]*\)".*"msg":"\([^"]*\)".
          fi
          sleep 3;
        done
    volumeMounts:
    - name: shared
      mountPath: /data
```

Test:

```
kubectl exec -it adapter-format -c adapter -- sh -c "cat /data/events.txt"
```

2) En Sık Karşılaşılan YAML Yapıları

En yaygın yapı: shared volume (emptyDir) + iki container

```
spec:
  volumes:
  - name: shared
    emptyDir: {}
  containers:
  - name: app
    ...
    volumeMounts:
    - name: shared
      mountPath: /shared
  - name: helper
    ...
    volumeMounts:
    - name: shared
      mountPath: /shared
```

Aynı Pod’da localhost iletişimi

```
# Container'lar aynı Pod IP ve network namespace'i paylaşır.
# Örn: Container-A 127.0.0.1:8080 dinliyorsa Container-B bu adrese erişebilir.
```

3) Sık Kullanılan Alanlar (Kısa Açıklamalar)

Alan	Ne işe yarar?
spec.containers[]	Pod içindeki container’ların listesi.
spec.initContainers[]	Pod başlamadan önce çalışacak init container’lar.
spec.volumes[]	Pod seviyesinde volume tanımları (emptyDir, pvc, configMap, secret).
containers[].volumeMounts[]	Volume’u container içine mount eder.
kubectl logs -c	Belirli container logunu alır.
kubectl exec -c	Belirli container’a shell/komut çalıştırır.

4) En Sık Kullanılan Komutlar ve Kullanımları

Multi-container Pod'larda en kritik nokta: doğru container'ı hedeflemek için -c kullanmaktır.

4.1 Pod'u inceleme

Container isimleri, mount'lar ve event'ler.

```
kubectl describe pod sidecar-log
kubectl get pod sidecar-log -o yaml
```

4.2 Container bazlı logs

Çok container'lı Pod'da logs için -c şart.

```
kubectl logs sidecar-log -c app
kubectl logs sidecar-log -c log-sidecar
```

4.3 Container bazlı exec

Hedef container'a girip dosya/port kontrolü yaparsınız.

```
kubectl exec -it sidecar-log -c app -- sh
kubectl exec -it sidecar-log -c log-sidecar -- sh
```

4.4 Hızlı doğrulama: shared volume

Shared volume'da beklenen dosya oluşmuş mu?

```
kubectl exec sidecar-log -c app -- ls -la /var/log/app
kubectl exec sidecar-log -c log-sidecar -- ls -la /var/log/app
```

5) Troubleshooting Hızlı Rehber

Yanlış container'a logs/exec: Mutlaka -c ile container seç.Shared volume boş: volume adı ve volumeMount name birebir aynı olmalı.Sidecar kapanıyor: tail -f gibi sürekli çalışan bir komut yoksa container exit eder.Port erişilemiyor: aynı Pod içinde localhost ile test et (ör. curl 127.0.0.1:PORT).

6) CKAD İpuçları

Sınavda multi-container Pod geldiğinde ilk refleks: kubectl logs -c ve kubectl exec -c.Sidecar/adapter örneklerinde en hızlı shared storage: emptyDir.Ambassador deseninde app genellikle sadece localhost'a konuşur; dış dünya iletişimini proxy container yapar.