Kubernetes Eğitim Kursu

Cluster Mimarisi, Temel Bileşenler ve 27 Konu ile Pratik Uygulamalar - Pod'dan Helm'e Kapsamlı Kubernetes Öğrenimi

Cahit Yusuf KAFADAR
Cloud Solution Architect

Kubestronaut

CKAD

CKA

CKS

KCSA

KCNA

Sınav Ağırlıkları - Zaman Planlaması

2 saatlik sınav üzerinden yaklaşık zaman dağılımı (ağırlık × 120 dakika)

Pratik zaman yönetimi için rehber

Alan	Ağırlık	Yaklaşık Süre
Application Design and Build	20%	24 dakika
Application Deployment	20%	24 dakika
Observability & Maintenance	15%	18 dakika
Env, Config & Security	25%	30 dakika
Services and Networking	20%	24 dakika

Not: Görev zorluğu ve çözüm süresi değişkenlik gösterir, bu tablo yalnızca genel bir perspektif sunar

Resmi Kaynaklar

CNCF CKAD: www.cncf.io/training/certification/ckad/

CNCF Curriculum: github.com/cncf/curriculum

Exam Tips: docs.linuxfoundation.org/tc-docs/certification/tips-cka-and-ckad

Eğitim İçeriği - 27 Temel Konu

1. Kubernetes Mimarisi	2. Pod	3. ReplicaSet	4. Deployment
5. Namespace	6. Commands & Arguments	7. ConfigMap	8. Secrets
9. Security Contexts	10. Resource Limits	11. Service Account	12. Taints & Tolerations
13. Node Affinity	14. Multi-Container Pods	15. Readiness Probes	16. Logging
17. Init Containers	18. Labels & Selectors	19. Rolling Update & Rollback	20. Jobs & CronJobs
21. Kubernetes Service	22. Network Policies	23. Ingress Networking	24. Gateway API
25. Persistent Volumes	26. Storage Class	27. Helm & Docker	

Kubernetes Nedir ve Neden Kullanılır?

Container orkestrasyon platformu

Kubernetes, konteynerleştirilmiş uygulamaları dağıtmak, yönetmek ve koordine etmek için kullanılan açık kaynaklı bir orkestrasyon platformudur.

Avantajlar

Otomatik dağıtım, otomatik ölçeklendirme, self-healing mekanizmaları ve yerleşik load balancing ile yüksek erişilebilirlik sağlar.

Kullanım Senaryoları

Mikroservis mimarileri, CI/CD pipeline'ları ve hybrid cloud ortamları gibi modern uygulama dağıtım senaryolarında yaygın olarak kullanılır.

CNCF Yönetimi

Kubernetes, Google tarafından başlatıldı ve şu anda Cloud Native Computing Foundation (CNCF) tarafından yönetilen açık kaynak bir projedir.

Kubernetes Cluster Mimarisi - Control Plane ve Worker Node

1

Control Plane Bileşenleri

API Server (tüm isteklerin merkezi), etcd (cluster state veritabanı), Scheduler (pod yerleştirme), Controller Manager (istenen durum kontrolü)

2

Worker Node Bileşenleri

Kubelet (node agent), Kube-proxy (network proxy), Container Runtime (Docker/containerd)

3

İletişim Akışı

kubectl → API Server → etcd → Scheduler → Kubelet

4

Container Runtime

Docker, containerd veya CRI-O üzerinde container çalıştırma

Bölüm 1: Pod

Kubernetes'in en küçük deployment birimi

Pod - YAML Konfigürasyonu ve kubectl Komutları

Örnek Pod YAML konfigürasyonu gösterimi aşağıdadır:

apiVersion: v1 kind: Pod metadata: name: nginx-pod labels: app: nginx spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80

kubectl Komutları

- **kubectl create -f pod.yaml : Pod tanımlamasını oluşturur**
- **kubectl get pods : Pod listesini ve durumunu gösterir**
- **kubectl describe pod nginx-pod : Pod detaylı bilgi çıkarımı yapar**
- **kubectl logs nginx-pod : Pod container loglarını görüntüler**
- **kubectl exec -it nginx-pod -- /bin/sh : Pod içinde etkileşimli kabuk başlatır**
- **kubectl delete pod nginx-pod : Belirtilen Podu siler**

Olası Sorunlar

- **ImagePullBackOff : İlgili image depoda bulunamaz veya çekilemez**
- **CrashLoopBackOff : Container sürekli çöküyor ve yeniden başlıyor**
- **Pending : Pod, node kaynak yetersizliği veya scheduling problemi yaşıyor**

Bölüm 2: ReplicaSet

Pod kopyalarının otomatik yönetimi

ReplicaSet - Pod Kopyalarının Yönetimi

ReplicaSet Amacı

Belirtilen sayıda pod kopyasının her zaman çalışmasını garantilemek

Self-Healing

Pod silindiğinde veya çöktüğünde otomatik olarak yeniden oluşturma özelliği

Selector Kullanımı

Label'lara göre pod'ları izleme ve yönetme mekanizması sağlar

Kullanım Senaryoları

High availability, yük dağılımı ve hata toleransı gibi senaryolar için uygundur

ReplicaSet - YAML Yapısı ve kubectl Yönetimi

apiVersion: apps/v1 kind: ReplicaSet metadata: name: nginx-rs spec: replicas: 3 selector: matchLabels: app: nginx template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:latest

kubectl Komutları

- kubectl create -f replicaset.yaml
- kubectl get rs
- kubectl scale rs nginx-rs --replicas=5
- kubectl describe rs
- kubectl delete rs

Olası Sorunlar

- Selector ve pod label uyuşmazlığı
- Resource quota aşımı
- Pod template değişiklikleri mevcut pod'ları etkilemez

Bölüm 3: Deployment

Declarative uygulama güncellemeleri ve sürüm yönetimi

Deployment – Declarative Uygulama Güncellemeleri

Deployment Rolü

ReplicaSet üzerinde sürüm yönetimi sağlar ve rolling update ile kesintisiz güncelleme sürecini otomatikleştirir, mevcut pod'ların kademeli olarak yeni sürüme geçirilmesini yönetir.

Avantajları

Otomatik rollback ve deployment history desteği sunar, canary deployment senaryoları ile güvenli test olanağı verir ve production ortamlarında sıfır kesinti hedefler.

Kullanım Amacı

Production ortamlarında güvenli ve kontrollü uygulama güncellemeleri yapmak, riskleri azaltmak ve kullanıcı deneyiminde kesinti olmadan yeni sürümlere geçiş sağlamak için kullanılır.

ReplicaSet Farkı

ReplicaSet sürüm yönetimi ve pod çoğaltma sorumluluğu taşır; deployment ise rolling update stratejisi ve rollback mekanizmalarıyla sürüm geçişlerini koordine eder.

Deployment - Konfigürasyon ve Komutlar

```
apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: replicas: 3 strategy: type: RollingUpdate rollingUpdate: maxUnavailable: 1 maxSurge: 1 selector: matchLabels: app: nginx template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.21
```

kubectl Komutları

- kubectl create deployment
- kubectl get deployments
- kubectl rollout status
- kubectl rollout history
- kubectl set image
- kubectl scale
- kubectl rollout undo

Olası Sorunlar

- RollingUpdate sırasında insufficient resources
- Probe hataları nedeniyle başarısız deployment
- imagePullPolicy misconfiguration

Bölüm 4: Namespace

Cluster kaynaklarının mantıksal bölümlenmesi

Namespace - Cluster Kaynaklarının Mantıksal Bölümlenmesi

Namespace Kavramı

Viral cluster'lar içinde sanal cluster'lar oluşturarak kaynakları izole etme mekanizması. Her namespace, ad alanı olarak kaynak ayrımı sağlar ve mantıksal izolasyon sunar.

Kullanım Alanları

Multi-tenant ortamlarda tenant izolasyonu, environment ayrımı (dev/staging/prod) ve ekip bazlı kaynak ayırma senaryolarında kullanılır.

Default Namespace'ler

Varsayılan olarak gelen önemli namespace'ler: default, kube-system, kube-public, kube-node-lease. Sistem ve kullanıcı kaynakları için ayrım sağlar.

Kaynak Kontrolü

ResourceQuota ve LimitRange ile namespace seviyesinde CPU, bellek ve diğer kaynaklar için kotalar ve limitler belirlenerek kontrol sağlanır.

Namespace - Oluşturma ve Yönetim Komutları

apiVersion: v1 kind: Namespace metadata: name: development labels: env: dev

kubectl Komutları

- kubectl create namespace dev
- kubectl get namespaces
- kubectl config set-context --current --namespace=dev
- kubectl get pods -n dev
- kubectl delete namespace dev

Olası Sorunlar

- Namespace silindiğinde içindeki tüm kaynaklar silinir
- Cross-namespace service erişimi için FQDN: service.namespace.svc.cluster.local
- Network policy ile namespace izolasyonu sağlanabilir

Bölüm 5: Commands and Arguments

Container başlangıç davranışını özelleştirme

Commands and Arguments - Container Başlangıç Davranışı

Command Kavramı

Dockerfile ENTRYPOINT'in Kubernetes karşılığıdır; container başlatıldığında çalışacak ana komut burada tanımlanır.

Args Kavramı

Dockerfile CMD'nin karşılığıdır ve komuta geçirilecek parametreler olarak kullanılır; komutla birlikte çalışır.

Kullanım Amacı

Container başlangıç davranışını override etmek ve özel script'ler veya başlangıç parametreleri çalıştırmak için kullanılır.

YAML Örneği

command: ['/bin/sh'], args: ['-c', 'echo Hello Kubernetes && sleep 3600']

Commands and Arguments - YAML ve Troubleshooting

```
apiVersion: v1 kind: Pod metadata: name: command-demo spec: containers: - name: busybox image: busybox:latest command: ["/bin/sh"] args: ["-c", "while true; do echo hello; sleep 10; done"]
```

kubectl Test Komutları

- kubectl run test --image=busybox --command -- /bin/sh -c 'echo test'
- kubectl logs command-demo
- kubectl describe pod command-demo

Olası Sorunlar

- Shell form vs exec form farkı
- Environment variable expansion
- Command başarısız olursa CrashLoopBackOff oluşur
- Çift tırnak kullanımı

Bölüm 6: ConfigMap

Uygulama konfigürasyonlarının dışsallaştırılması

ConfigMap - Uygulama Konfigürasyonlarının Dışsallaştırılması

ConfigMap Amacı

Konfigürasyon verilerini container image'dan ayırarak esneklik sağlar ve dağıtım süreçlerini basitleştirir

Kullanım Şekilleri

Environment variable ile atama, volume mount ile dosya sağlama ve command-line argümanlarıyla yapılandırma

Avantajları

Image yeniden build etmeden config değişikliği yapılabilir; ortama göre farklı konfigürasyon uygulanabilir

Kullanım Senaryoları

Application.properties, nginx.conf veya veritabanı bağlantı bilgileri gibi dışsal konfigürasyon örnekleri

ConfigMap - Oluşturma ve Kullanım Yöntemleri

apiVersion: v1 kind: ConfigMap metadata: name: app-config data: database_url: "postgresql://localhost:5432" log_level: "info" config.properties: | app.name=MyApp app.version=1.0

kubectl Komutları

- kubectl create configmap app-config --from-literal=key=value
- kubectl create configmap --from-file=config.txt
- kubectl get configmap
- kubectl describe configmap

Olası Sorunlar

- ConfigMap güncellemesi genellikle pod restart gerektirmez; ancak volume mount ile kullanıldığında yeniden başlatma gerekebilir
- ConfigMap için 1MB boyut limiti vardır; büyük yapılandırmalar için alternatif yöntemler düşünülmelidir
- immutable ConfigMap özelliği (v1.21 ve sonrası) kullanıldığında ConfigMap değiştirilemez hale gelir, sürüm ve kısıtlamalara dikkat edin

Bölüm 7: Secrets

Hassas verilerin güvenli saklanması

Secrets - Hassas Verilerin Güvenli Saklanması

Secret Rolü

Şifre, token, SSH key gibi hassas verileri güvenli saklamak ve uygulamaların ihtiyaç duyduğu sırları güvenli şekilde sunmak.

ConfigMap Farkları

ConfigMap düz metin ayar saklarken Secret hassas veri saklar; Base64 enkodlama farklıdır, etcd'de encryption at rest ve RBAC ile erişim kontrolü önemlidir.

Secret Tipleri

Opaque (generic), dockerconfigjson (image pull), tls (TLS sertifikası) ve service-account-token gibi farklı secret türleri vardır.

Güvenlik

Ek güvenlik için Vault veya AWS Secrets Manager gibi external secret management çözümleri ile entegrasyon önerilir.

Secrets - Oluşturma ve Kullanım

apiVersion: v1 kind: Secret metadata: name: db-secret type: Opaque stringData: username: admin password: SuperSecret123 # data: # username: YWRtaW4= # base64 encoded # password: U3VwZXJTZWNyZXQxMjM=

kubectl Komutları

- kubectl create secret generic db-secret --from-literal=password=123
- kubectl create secret docker-registry
- kubectl get secrets
- kubectl describe secret

Olası Sorunlar

- Base64 şifreleme yerine encoding kullanımı konusunda dikkat
- Secret değişikliği genellikle pod restart gerektirmez, ancak bazı senaryolarda yeniden başlatma gerekebilir
- External secret operator kullanımıyla yönetimi kolaylaştırma
- Private registry için imagePullSecrets yapılandırması gereklidir

Bölüm 8: Security Contexts

Container ve Pod güvenlik ayarları

Security Contexts - Container ve Pod Güvenlik Ayarları

Security Context Kavramı

Container'ların çalışma izinlerini ve güvenlik özelliklerini kontrol etmek için kullanılan yapı. İşlem izinleri, runAsUser ve diğer sınırlar bu kapsamda tanımlanır ve yönetilir.

Kullanım Alanları

Root olarak çalışmayı engelleme (runAsNonRoot) ve privilege escalation önleme gibi ayarlarla konteynerlerin saldırı yüzeyi azaltılır ve güvenlik sağlanır.

Pod vs Container Level

Container ayarları, pod ayarlarını override edebilir; hem pod hem de container seviyesinde güvenlik politikaları tanımlanabilir ve birlikte çalıştırılmalıdır.

Best Practices

Minimum privilege prensibini uygulayın, read-only filesystem kullanın ve gereksiz yetenekleri drop ederek saldırı yüzeyini en aza indirin.

Security Contexts - Konfigürasyon ve Best Practices

apiVersion: v1 kind: Pod metadata: name: secure-pod spec: securityContext: runAsUser: 1000 runAsGroup: 3000 fsGroup: 2000 containers: - name: app image: nginx securityContext: runAsNonRoot: true readOnlyRootFilesystem: true allowPrivilegeEscalation: false capabilities: drop: ["ALL"]

kubectl Kontrol

- kubectl exec secure-pod -- id (konteyner içinde kullanıcı bilgisi görüntüle)
- kubectl exec secure-pod -- ls -l (dosya izinlerini ve sahipliğini kontrol et)
- kubectl describe pod secure-pod (Pod detaylarını, event ve securityContext ayarlarını incele)

Olası Sorunlar

- Permission denied hataları nedeniyle uygulama dosya erişimi engellenebilir
- Volume mount izin sorunları fsGroup veya izinler nedeniyle oluşabilir
- Gereken capabilities eksikse bazı işlemler başarısız olabilir
- PodSecurityPolicy veya PodSecurityStandard enforcement politikaları engelleyebilir

Bölüm 9: Resource Limits

CPU ve Memory kaynak yönetimi

Resource Limits - CPU ve Memory Kaynak Yönetimi

Requests vs Limits

Requests: Garanti edilmiş kaynak, scheduling kriteri. Limits: Maksimum kullanım, aşımda throttling veya OOMKilled ile sonuçlanabilir.

Kullanım Amacı

Node kaynaklarının adil dağılımı, noisy neighbor problemini önleme ve cluster stabilitesinin korunması.

QoS Sınıfları

Guaranteed (request=limit), Burstable (request

Kaynak Birimleri

CPU: 1 CPU = 1000m (millicore). Memory: Mi (mebibyte), Gi (gibibyte).

Resource Limits - Tanımlama ve Monitoring

apiVersion: v1 kind: Pod metadata: name: resource-demo spec: containers: - name: app image: nginx resources: requests: cpu: "100m" memory: "128Mi" limits: cpu: "500m" memory: "512Mi"

kubectl Komutları

- kubectl top nodes
- kubectl top pods
- kubectl describe node (Allocated resources)
- kubectl set resources deployment --limits=cpu=500m,memory=512Mi

Olası Sorunlar

- OOMKilled (memory limit aşımı)
- CPU throttling (limit aşımında yavaşlama)
- Pod eviction (node pressure)
- LimitRange ile namespace default limits

Bölüm 10: Service Account

Pod'ların API Server kimlik doğrulaması

Service Account – Pod'ların API Server Kimlik Doğrulaması

1

Service Account Kavramı

Pod'ların Kubernetes API'ye kimlik doğrulaması için kullandıkları hesaplar.

2

Kullanım Amacı

RBAC ile API erişim kontrolü sağlamak ve farklı pod'lara farklı yetkiler atamak için kullanılır.

3

Default Behavior

Her namespace'de default service account otomatik oluşturulur ve pod'lara otomatik olarak mount edilir.

4

Token Location

Kimlik doğrulama tokeni /var/run/secrets/kubernetes.io/serviceaccount/token yolunda saklanır.

Service Account - Oluşturma ve RBAC Entegrasyonu

apiVersion: v1 kind: ServiceAccount metadata: name: app-sa automountServiceAccountToken: true --- apiVersion: v1 kind: Pod metadata: name: app-pod spec: serviceAccountName: app-sa containers: - name: app image: nginx

kubectl Komutları

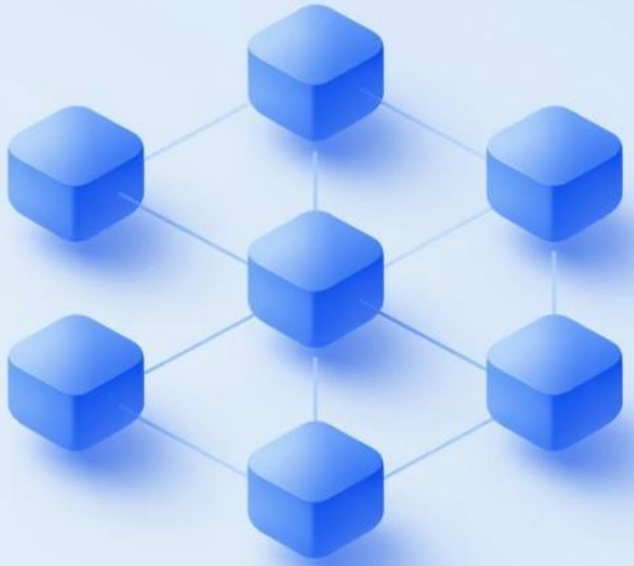
- kubectl create serviceaccount app-sa
- kubectl get serviceaccounts
- kubectl describe sa app-sa
- kubectl create token app-sa

Olası Sorunlar

- Yetki yetersizliği (forbidden errors)
- token expiration (TokenRequestAPI)
- RBAC Role ve RoleBinding gerekli
- OIDC external identity provider

Bölüm 11: Taints and Tolerations

Node'lara Pod yerleştirme kısıtlamaları



Taints and Tolerations - Node'lara Pod Yerleştirme Kısıtlamaları

Taint Kavramı

Node'lara kısıtlama etiketi ekleyerek belirli pod'ların yerleşmesini engelleme

Toleration Kavramı

Pod'ların taint'leri tolere ederek o node'lara yerleşebilmesi

Kullanım Amacı

Dedicated nodes (GPU, SSD), node bakımı ve kritik workload izolasyonu

Taint Effects

NoSchedule (yeni pod yerleşmez), PreferNoSchedule, NoExecute (mevcut pod'lar evict edilir)

Taints and Tolerations - Uygulama ve Yönetim

 # Node taint ekleme kubectl taint nodes node1 gpu=true:NoSchedule # Pod toleration apiVersion: v1 kind: Pod metadata: name: gpu-pod spec: tolerations: - key: "gpu" operator: "Equal" value: "true" effect: "NoSchedule" containers: - name: app image: tensorflow/tensorflow:latest-gpu

kubectl Komutları

- **kubectl taint nodes node1 key=value:NoSchedule**
- **kubectl taint nodes node1 key:NoSchedule- (kaldırma)**
- **kubectl describe node (Taints bölümü)**

Olası Sorunlar

- **Tolerasyon eklenmemiş pod'lar Pending durumda kalır**
- **NoExecute etkisi ile mevcut pod'lar silinebilir**
- **Taint ve toleration arasında key-value eşleşmesi gereklidir**

Bölüm 12: Node Affinity

Pod'ları belirli Node'lara yönlendirme

Node Affinity – Pod'ları Belirli Node'lara Yönlendirme

Node Affinity Kavramı

Pod'ların belirli özelliklere sahip node'lara schedule edilmesini sağlama. Etiketler ve kurallar kullanılarak hedeflenmiş yerleştirme yapılır.

Taint/Toleration Farkı

Pozitif seçim (tercih etme) vs negatif seçim (engelleme) yaklaşımı. Taint node seviyesinde reddeder, toleration pod üzerinde izin verir.

Affinity Tipleri

required (zorunlu), preferred (tercih edilen) ve IgnoredDuringExecution gibi farklı eşleme tipi ve öncelik seçenekleri bulunur.

Kullanım Senaryoları

Zone/region bazlı dağıtım, donanım özelliklerine göre yerleştirme (SSD, GPU) gibi senaryolarda Node Affinity ile performans ve uyumluluk sağlanır.

Node Affinity - Konfigürasyon ve Operatörler

apiVersion: v1 kind: Pod metadata: name: affinity-pod spec: affinity: nodeAffinity: requiredDuringSchedulingIgnoredDuringExecution: nodeSelectorTerms: - matchExpressions: - key: disktype operator: In values: ["ssd"] preferredDuringSchedulingIgnoredDuringExecution: - weight: 1 preference: matchExpressions: - key: zone operator: In values: ["us-east-1a"]

Operatörler ve kubectl

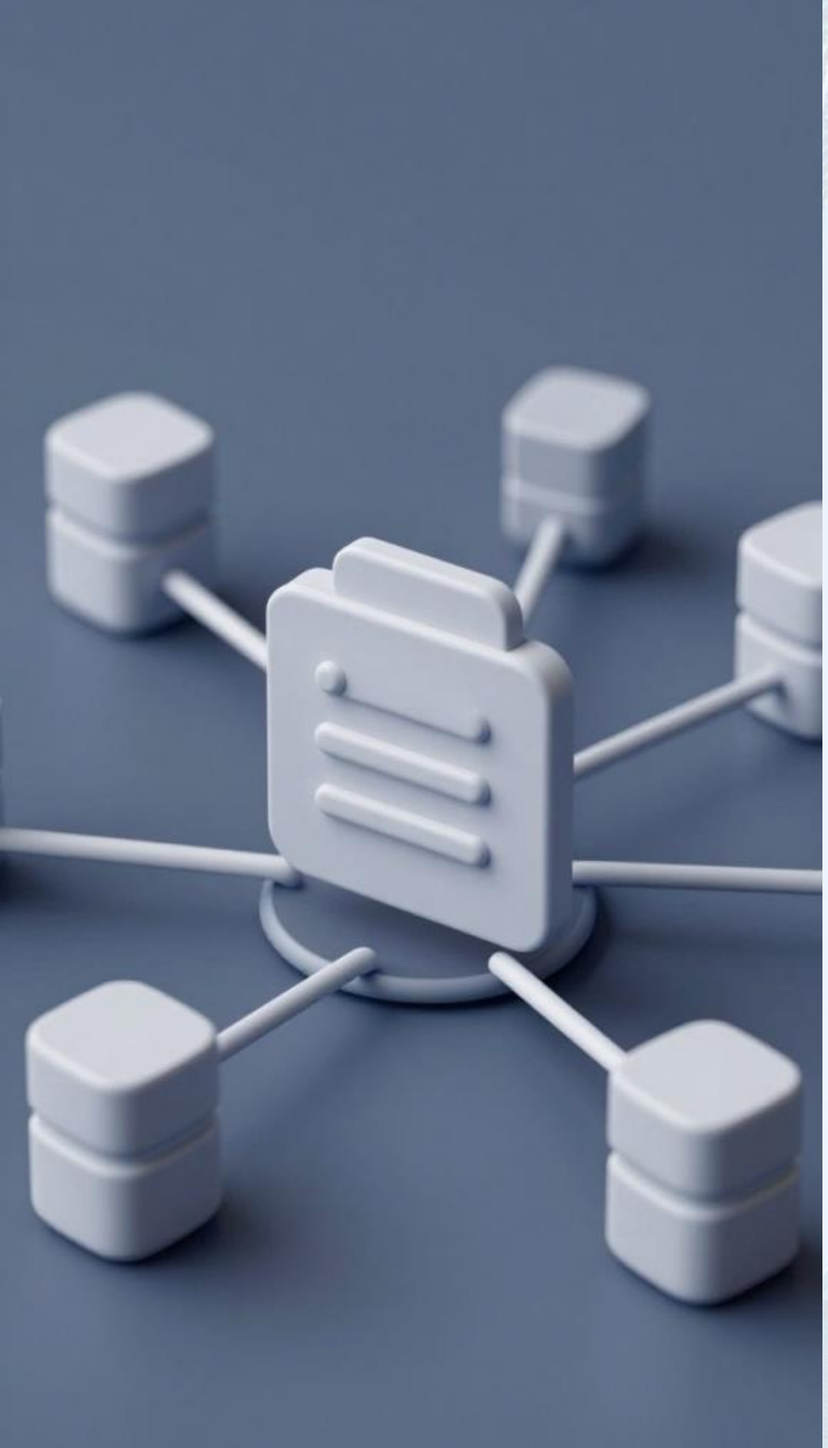
In, NotIn, Exists, DoesNotExist, Gt, Lt operatörleri. Örnek komutlar: kubectl label nodes node1 disktype=ssd; kubectl get nodes --show-labels

Olası Sorunlar

Uygun node bulunamazsa pod pending kalır. Node selector (basit) vs affinity (gelişmiş). Anti-affinity ile pod dağılımı sağlanarak çakışmalar azaltılabilir.

Bölüm 13: Multi-Container Pods

Tek Pod içinde birden fazla container



Multi-Container Pods - Tek Pod İçinde Birden Fazla Container

Multi-Container Pattern'ler

Sidecar (log collector), Ambassador (proxy), Adapter (format converter) gibi yardımcı konteyner desenleri ile işlevsel ayrışma sağlanır.

Kullanım Amacı

Tight coupling gereken servisleri aynı pod'da çalıştırma; birbirini tamamlayan görevlerin birlikte yönetilmesi için kullanılır.

Paylaşılan Kaynaklar

Network namespace (localhost), paylaşılan volumes (filesystem) ve IPC namespace gibi kaynaklar konteynerler arasında ortak kullanılır.

Container Lifecycle

Ana container çalıştığı sürece pod Running durumunda kalır; herhangi bir konteyner fail olursa pod yeniden başlatılır.

Multi-Container Pods - Tasarım ve Örnek Senaryolar

```
apiVersion: v1 kind: Pod metadata: name: web-app spec: containers: - name: nginx image: nginx:latest volumeMounts: - name: shared-logs mountPath: /var/log/nginx - name: log-forwarder image: fluentd:latest volumeMounts: - name: shared-logs mountPath: /logs volumes: - name: shared-logs emptyDir: {}
```

kubectl Komutları

- kubectl logs web-app -c nginx (nginx container için logları gösterir ve hata kontrolü yapar)
- kubectl logs web-app -c log-forwarder (log-forwarder container loglarını izler ve yönlendirmeyi doğrular)
- kubectl exec web-app -c nginx -- command (nginx içinde komut çalıştırarak canlı müdahale sağlar)

Olası Sorunlar

- Container'lar arası bağımlılık yönetimi (başlangıç sıralaması ve veri paylaşımı senaryoları dikkatle planlanmalı)
- Resource limit tüm container'lar için ayrı tanımlanmalı (CPU ve bellek limitleri her container bazında belirlenmeli)
- Readiness probe her container için kontrol (her container'ın hazır olma durumu ayrı ayrı izlenmeli ve servis trafiği buna göre yönlendirilmeli)

Bölüm 14: Readiness Probes

Pod'un trafiği kabul etmeye hazır olduğunu kontrol

Readiness Probes - Pod'un Trafiği Kabul Etmeye Hazır Olması

Readiness Probe Amacı

Pod başladığında hemen trafiği almaması, hazır olana kadar beklenmesi

Liveness vs Readiness

Liveness: container'ı restart et; Readiness: service endpoint'inden çıkar

Probe Tipleri

httpGet (HTTP request), tcpSocket (TCP connection), exec (komut çalıştırma)

Kullanım Senaryoları

Veritabanı bağlantısı kontrolü, cache ısınması, bağımlı servislerin hazır olması

Readiness Probes - Konfigürasyon ve Best Practices

apiVersion: v1 kind: Pod metadata: name: webapp spec: containers: - name: app image: myapp:v1 readinessProbe: httpGet: path: /health port: 8080 initialDelaySeconds: 10 periodSeconds: 5 timeoutSeconds: 3 successThreshold: 1 failureThreshold: 3

Kubectl Kontrol

- kubectl describe pod webapp (Readiness bölümünü kontrol edin)
- kubectl get pods (READY sütunu; örn. 0/1 ise hazır değil)
- kubectl get endpoints service-name (servis endpointlerini doğrulayın)

Olası Sorunlar

- Probe başarısız olursa pod Ready olmaz ve servis trafiği almaz
- Çok agresif probe ayarları false negative sonuçlar üretebilir
- Uzun başlangıç süreleri için startupProbe kullanmak daha uygundur

Bölüm 15: Logging

Container ve Cluster log yönetimi

Logging - Container ve Cluster Log Yönetimi

Logging Seviyeleri

Container logları (stdout/stderr), node seviyesinde kayıtlar ve cluster seviyesinde merkezi log toplama ile kapsamlı gözlem sağlar.

Log Stratejileri

Node logging agent (DaemonSet), sidecar container yaklaşımı veya uygulamadan log backend'e doğrudan push etme seçenekleri değerlendirilir.

Logging Stack Örnekleri

Yaygın örnekler: EFK (Elasticsearch-Fluentd-Kibana), ELK (Elasticsearch-Logstash-Kibana), Loki-Promtail-Grafana ile merkezi izleme.

Best Practices

Structured logging (JSON) kullanımı, log rotation ve retention policy uygulanması ile persistent storage önerilir.

Logging - kubectl Log Komutları ve Troubleshooting

kubectl Log Komutları

- kubectl logs pod-name
- kubectl logs pod-name -c container-name
- kubectl logs pod-name --previous (crash sonrası)
- kubectl logs pod-name --since=1h
- kubectl logs pod-name --tail=100
- kubectl logs -f pod-name (follow)
- kubectl logs deployment/name
- kubectl logs -l app=myapp (label selector)

Olası Sorunlar

- Log rotation ve retention politikalarının eksikliği, eski logların kontrolü gerekir
- Disk dolması nedeniyle logrotate yapılandırması ve uyarı mekanizmaları önemlidir
- Ephemeral storage limitleri log kaybına yol açabilir, limitler izlenmelidir
- Persistent logging için external solution (ELK, Fluentd, Loki vb.) gereklidir
- Container çökerse lokal loglar kaybolabilir; merkezi loglama ile korunmalıdır

Bölüm 16: Init Containers

Pod başlatılmadan önce çalışan container'lar

Init Containers - Pod Başlatılmadan Önce Çalışan Container'lar

Init Container Kavramı

Ana container'lar başlamadan önce sırayla çalışan, tamamlanması gereken container'lar

Kullanım Amaçları

Database migration, configuration dosyası oluşturma, dependency servislerin hazır olmasını bekleme

Davranış

Init container'lar tamamlanana kadar ana container başlamaz, başarısız olursa pod restart edilir

Özellikler

Sıralı çalışır, resource limits ayrı tanımlanabilir, volume mount paylaşılır

Init Containers, Labels ve Selectors - Konfigürasyon ve Yönetim

apiVersion: v1 kind: Pod metadata: name: myapp labels: app: frontend env: prod spec: initContainers: - name: init-db image: busybox command: ['sh', '-c', 'until nslookup mydb; do sleep 2; done'] containers: - name: app image: myapp:v1

Init Container Komutları

- kubectl describe pod myapp (Init Containers section)
- kubectl logs myapp -c init-db

Labels ve Selectors

- kubectl label pods myapp tier=frontend
- kubectl get pods -l app=frontend
- kubectl get pods -l 'env in (prod,staging)'

Rolling Update, Rollback, Jobs ve CronJobs

Rolling Update

Güncellemeler için kubectl set image deployment/app nginx=nginx:1.22 komutu kullanılabilir; ilerleme kontrolü için kubectl rollout status deployment/app; ayrıca maxUnavailable ve maxSurge parametreleri ile Pod güncelleme stratejisi ayarlanır.

Rollback

Bir sürüme geri dönmek için kubectl rollout undo deployment/app; belirli revizyona dönmek için kubectl rollout undo --to-revision=2; geçmişi görmek için kubectl rollout history deployment/app kullanılır.

Jobs

Batch processing için Jobs kullanılır; completions ve parallelism parametreleri iş tamamlanma ve paralel çalışma kontrolünü sağlar; örnek oluşturma: kubectl create job backup --image=mysql:8.0.

CronJobs

Zamanlı işler için CronJobs; schedule örneği '0 2 * * *' ile günlük 02:00'de çalıştırma; yaratma örneği: kubectl create cronjob backup --image=backup --schedule='0 2 * * *'.

Services, Network Policies, Ingress ve Gateway API

Kubernetes Service

ClusterIP (cluster içi erişim), NodePort (harici erişim 30000-32767 aralığı), LoadBalancer (cloud LB). Örnek komut: kubectl expose deployment app --port=80

Network Policies

Pod'lar arası trafiği kontrol eder; ingress ve egress kuralları tanımlanır. PodSelector ve namespaceSelector ile hedefleme yapılır. CNI plugin gereklidir (Calico, Cilium gibi).

Ingress

HTTP/HTTPS yönlendirme sağlar; path-based ve host-based routing desteklenir. TLS termination yapılabilir. Bir Ingress Controller gereklidir (NGINX, Traefik vb.).

Gateway API

Yeni nesil routing modeli: GatewayClass, Gateway ve HTTPRoute kaynakları ile çalışır. Role separation ve vendor-agnostic tasarım sunar, daha esnek ve genişletilebilir.

Persistent Volumes, Storage Class, Helm ve Docker

Persistent Volumes

- PV: admin provision ile sağlanan fiziksel/abastract depolama
- PVC: user request yoluyla depolama talebi
- accessModes: RWO / ROX / RWX
- reclaimPolicy: Retain / Delete / Recycle
- kubectl get pv ile PV listesi görüntülenir
- kubectl get pvc ile PVC durumları kontrol edilir

Storage Class

- Dinamik provisioning ile dynamic volume oluşturma
- provisioner: aws-ebs, azure-disk, gce-pd gibi sağlayıcılar
- volumeBindingMode: Immediate veya WaitForFirstConsumer
- default storage class olarak atanmış sınıf önemli

Helm

- Kubernetes paket yöneticisi olarak Helm kullanılır
- chart-based deployment ile paketlenmiş uygulama dağıtımı
- helm install app chart komutuyla kurulum yapılır
- helm upgrade ile güncelleme, helm rollback ile geri alma
- helm repo add ile chart repository eklenir

Docker

- Container image build süreci Dockerfile ile yapılır
- Docker registry: Docker Hub veya private registry kullanımı
- private image için imagePullSecrets ile kimlik sağlanır
- container runtime örnekleri: containerd, CRI-O

Teşekkürler - Kubernetes Eğitimi Tamamlandı

27 temel konuyu tamamladık. Sorularınız için hazırım. Başarılı Kubernetes çalışmaları dilerim!