Özgür ÖZTÜRK Hocamızın,
https://www.udemy.com/course/kubernetes-temelleri/
eğitimi

KUBERNETES-102

Notları

2023

Kazim Esen

KUBERNETES - 102

Node Affinity

Kavramsal olarak nodeSelector'a benzer ve node'lara atanan etiketlere göre pod'un hangi node üstünde schedule edilmeye uygun olduğunu kısıtlamaya olanak tanır.

requiredDuringSchedulingIgnoredDuringExecution: Eşleşmeye uygun bir node bulunursa podu oluştur, node bulunmazsa podu oluşturma talimatıdır.
preferredDuringSchedulingIgnoredDuringExecution: Mümkünse eşleşmeye uygun bir node bulunursa podu o node'da oluştur, bulunmazsa podu herhangi bir node'da oluştur talimatıdır.
IgnoredDuringExecution: Pod oluşturulduktan sonra key silinirse ne olacağını belirlemek amacıyla konulmuştur ancak şimdilik bu durumu ele alan bir yapı yok, muhtemelen gelecekte kullanmak içindir.
Affinity (yakınlık) ile etiketlere göre ve **weight** (ağırlık) belirterek seçimleri önceliklendirmeye yarar, ağırlığı fazla olan önceliklidir.

Burada kullanılabilecek dört operator :
- **In :** Anahtar ile değer arasında eşleşme bulunursa podu oluştur.
- **NotIn :** Anahtar ile değer arasında eşleşme bulunmazsa podu oluştur.
- **Exists :** Anahtar varsa podu çalıştır, değer önemli değil, olsa da olur olmasa da olur.
- **DoesNotExist :** Anahtar yoksa podu çalıştır.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeaffinitypod1
spec:
  containers:
  - name: nodeaffinity1
    image: ozgurozturknet/k8s
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: app
            operator: In
            values:
            - blue
---
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeaffinitypod2
spec:
  containers:
  - name: nodeaffinity2
    image: ozgurozturknet/k8s
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: app
            operator: In
            values:
            - blue
      - weight: 2
        preference:
          matchExpressions:
          - key: app
            operator: In
            values:
            - red
---
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeaffinitypod3
spec:
  containers:
  - name: nodeaffinity3
    image: ozgurozturknet/k8s
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: app
            operator: Exists
```

```
kubectl apply -f .\podnodeaffinity.yaml
```
komutundan sonra 3 pod created olur, ancak sadece **nodeaffinitypod2 Running** olur, diğer 2 pod Pending'te kalır.

```
kubectl label node minikube app=blue
```
komutundan sonra; **nodeaffinitypod1** ve **nodeaffinitypod3** pod'ları da **Running** olur.

Pod Affinity

Podun hangi node üstünde oluşturulmaya uygun olduğunu, nodelardaki etiketlere göre değil de, halihazırda node'da çalışmakta olan podlardaki etiketlere göre sınırlamaya yarar.
Her k8s node'unda önceden tanımlanmış bir çok **label** mevcuttur ve her node'da;
kubernetes.io/arch
kubernetes.io/hostname
kubernetes.io/os adında 3 label ile bunun yanında cloud provider'lar için de;
topology.kubernetes.io/region
topology.kubernetes.io/zone label'ları bulunur.

requiredDuringSchedulingIgnoredDuringExecution: Eşleşme gereksinimleri planlama zamanında karşılanmazsa, pod node'da planlanmayacaktır. Benzeşim gereksinimleri, pod yürütme sırasında bir noktada (örneğin, bir pod elabel güncellemesi nedeniyle) karşılanmazsa, sistem podu sonunda node'dan çıkarmaya çalışabilir veya denemeyebilir. Birden çok öğe olduğunda, her **podAffinityTerm**'e karşılık gelen node listeleri kesişir, yani tüm terimler karşılanmalıdır. Birlikte konumlandırılma (affinity) veya birlikte konumlandırılmama (anti-affinity) olması gereken bir pod kümesini tanımlar.

labelSelector (obje) : Bir dizi kaynak, burada podlar üzerinde bir label sorgusu çalıştırır.
namespaces (string[]) : labelSelector'ın hangi namespace'e uygulanacağını belirtir; boş veya boş liste, "bu podun namespace'ı" anlamına gelir.
topologyKey (string) : Bu pod, belirtilen namespace'lerde labelSelector ile eşleşen podlar ile birlikte konumlandırılmalı (benzeşim) veya birlikte bulunmamalıdır (anti-benzeşim).
Birlikte konumlanma, label değeri topologyKey anahtarına sahip bir node'da çalışmak olarak tanımlanır, seçilen podlardan herhangi birinin üzerinde çalıştığı herhangi bir node'unkiyle eşleşir.
Boş topologyKey'e izin verilmez.

preferredDuringSchedulingIgnoredDuringExecution: Zamanlayıcı (scheduler), pod'ları bu alan tarafından belirtilen affinity ifadelerini karşılayan node'lara planlamayı (schedule) tercih eder, ancak bir veya daha fazla ifadeyi ihlal eden bir node seçebilir. En çok tercih edilen node, en büyük ağırlık toplamına sahip node'dur, yani tüm zamanlama gereksinimlerini karşılayan her node için (kaynak isteği, preferredDuringScheduling benzeşim ifadeleri, vb.), bu alanın öğelerini yineleyerek bir toplam hesaplanır ve eğer node karşılık gelen podAffinityTerm ile eşleşen pod'lara sahipse toplama "ağırlık (weight)" eklenmesi; en yüksek toplama sahip node(lar) en çok tercih edilenlerdir. Eşleşen tüm WeightedPodAffinityTerm alanlarının ağırlıkları, en çok tercih edilen node'ları bulmak için node başına eklenir

podAffinityTerm (nesne): Gerekli. Karşılık gelen ağırlıkla ilişkili bir pod affinity terimi.
weight (tam sayı): 1-100 aralığında karşılık gelen podAffinityTerm ile eşleşen ağırlık.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontendpod
  labels:
    app: frontend
    deployment: test
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cachepod
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - frontend
        topologyKey: kubernetes.io/hostname
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: color
              operator: In
              values:
              - blue
          topologyKey: kubernetes.io/hostname
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: deployment
              operator: In
              values:
              - prod
          topologyKey: topology.kubernetes.io/zone
  containers:
  - name: cachecontainer
    image: redis:6-alpine
```

Anti-affinity vs. taints ve tolerations

Bir pod'u bir node'a atamak için tasarlanmış node affinity incelendi. Kavramsal olarak, anti-affinity, Kubernetes'teki taints ve tolerations ile benzer işlevsellik sağlar. Her iki özellik de pod'ların belirli node'lara schedule edilmesini engeller. Birincil fark, anti-affinity'nin label'lara dayalı olarak eşleştirme koşullarını kullanması, bu arada taint'ların node'a uygulanması ve pod bildirimlerinde tanımlanan toleration'ları eşleştirmesidir.

Pod'ların belirli node'lerde schedule'ını durdurmak için taints ve tolerations daha iyi bir seçenektir. Bununla birlikte, affinity, anti-affinity, taints, ve tolerations birbirini dışlamaz. Karmaşık schedule senaryolarını kolaylaştırmak için bunlar birleştirilebilir.

Affinity, nodeSelectorTerms veya matchExpression kullanılarak yapılandırılabilir. Aradaki fark, nodeSelectorTerms kullanılarak birden çok kural yapılandırıldığında tanımlanmış herhangi bir koşul eşleşirse pod'un bir node'a schedule edilebilmesidir. Öte yandan, matchExpression kullanıldığında; pod yalnızca, tüm matchExpression kuralları karşılanırsa schedule'lanır.

pod affinity/anti-affinity tanımlama

Bir node'daki tüm pod'ların aynı uygulamayla ilişkili olması için podları schedule ederken pod affinity kullanılabilir. Aslında, pod affinity, podları bir arada konumlandırmanın tercih edilen yoludur.

Schedule süreci üzerinde daha da fazla kontrol elde etmek için affinity, taints ve tolerations gibi diğer özelliklerle birleştirilebilir.

`kubectl get node` # node'ları listeler
`kubectl describe node minikube` # minikube node'nun detaylarını gösterir
`kubectl taint node minikube platform=production:NoSchedule` # node/minikube tainted
`kubectl taint node minikube platform-` # node/minikube untainted ile de silinir ama bu çalıştırılmayacak.
`kubectl run test --image=nginx --restart=Never` # Yaratılan pod Pending'te kalıyor.
`kubectl describe pod test` # Uyarı FailedScheduling default-scheduler 0/1 nodes kullanılabilir: 1 node(s), pod'un tolerate edemediği {platform: production} taint'ine sahiptir.

```
apiVersion: v1
kind: Pod
metadata:
  name: toleratedpod1
  labels:
    env: test
spec:
  containers:
  - name: toleratedcontainer1
    image: ozgurozturknet/k8s
  tolerations:
  - key: "platform"
    operator: "Equal"
    value: "production"
    effect: "NoSchedule"
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: toleratedpod2
  labels:
    env: test
spec:
  containers:
  - name: toleratedcontainer2
    image: ozgurozturknet/k8s
  tolerations:
  - key: "platform"
    operator: "Exists"
    effect: "NoSchedule"
```

`kubectl apply -f .\podtoleration.yaml` # iki podda tolerations tanımına sahip olduğu için Running oldu.
`kubectl taint node minikube color=blue:NoExecute` # node/minikube tainted, bu node'da color=blue key'ini tolere edemeyen pod'lara izin verme, çalışan varsa da sonlandır. Bu komuttan sonra Running olan 2 podda Terminat edilerek silinir.

`kubectl get pods --field-selector status.phase!=Running` # Status'ü Running olmayan pod'ları gösterir
`kubectl get pods --field-selector status.phase=Pending` # Status'ü Pending olan pod'ları gösterir
`kubectl delete pods --field-selector status.phase=Pending` # Status'ü Pending olan pod'ları siler

DaemonSet

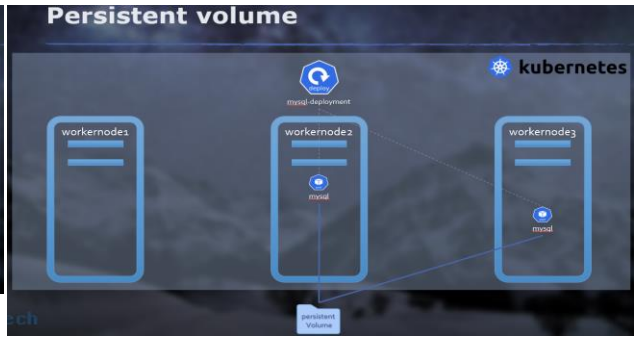
Tüm (veya bazı) node'ların bir Pod'un bir kopyasını çalıştırmasını sağlar. Cluster'a yeni node eklendikçe, onlara Pod'lar da eklenir. Cluster'dan node kaldırıldığında, bu Pod'lar da kaldırılır.
DaemonSet silinirse, oluşturduğu Pod'lar da temizlenir.
`kubectl get nodes`
`kubectl apply -f .\daemonset.yaml`

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: logdaemonset
  labels:
    app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      # this toleration is to have the daemonset runnable on master nodes
      # remove it if your masters can't run pods
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

`minikube node add` # Node eklendikçe pod oluşur, `kubectl get nodes` ve `kubectl get pods` ile izlenebilir.
`kubectl get daemonset -w` # DaemonSet objesinden de görülür
`kubectl delete pods logdaemonset-txsvs` # Pod manuel olarak silinse bile hemen tekrar oluşturulur;
DaemonSet, her node üstünde bir pod çalıştırmaya yarar.

Persistent Volume ve Persistent Volume Claim

Ephemeral volumler olan EmptyDir ve hostPath tipi volumler pod yaşam süresi boyunca geçerlidir. Sırada persistent (kalıcı) volumler var.



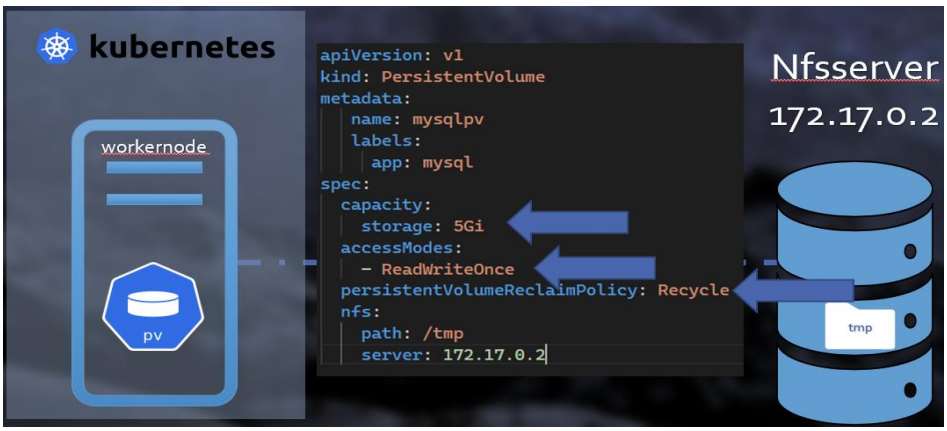
Container Storage Interface (CSI): CSI, isteğe bağlı blok ve dosya depolama sistemlerini Kubernetes gibi Container Orchestration Systems (CO'ler) üzerindeki containerized iş yüklerine maruz bırakmak için bir standart olarak geliştirilmiştir.
Container Storage Interface Kubernetes volume katmanının genişletilebilir hale gelmesini sağladı.
Üçüncü taraf depolama sağlayıcıları, CSI kullanarak, çekirdek Kubernetes koduna dokunmadan Kubernetes'te yeni depolama sistemleri için eklentiler yazabilme imkanına kavuştu.

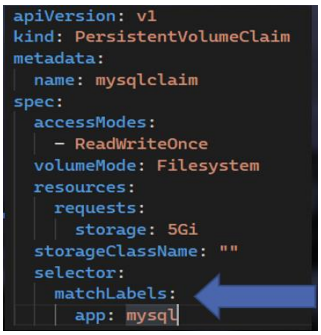
accessModes: Volum birden fazla poda bağlandığında nasıl davranacağını belirler.

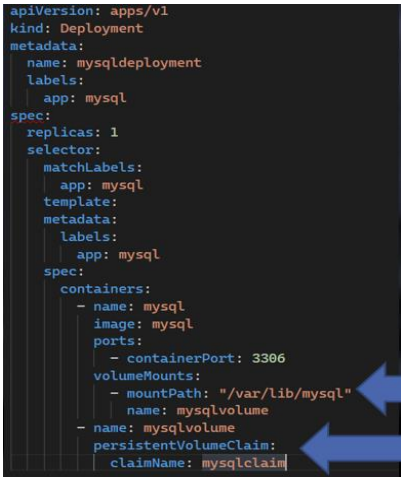
- **ReadWriteOnce :** okuma yazma – tek node
- **ReadOnlyMany :** sadece okuma – birden fazla node
- **ReadWriteMany :** okuma yazma – birden fazla node

persistentVolumeReclaimPolicy: Pod işini bitirip kullanmayı bıraktığında, nasıl davranılacağını belirler.

- **Retain :** Volume olduğu gibi kalır.
- **Recycle :** Volume silinmez ama içindeki tüm dosyalar silinir.
- **Delete :** Volume tamamen silinir.







PV ve PVC Uygulama

```
docker volume create nfsvol # NFS Server

docker network create --driver=bridge --subnet=10.255.255.0/24 --ip-
range=10.255.255.0/24 --gateway=10.255.255.10 nfsnet

docker run -dit --privileged --restart unless-stopped -e SHARED_DIRECTORY=/data -v
nfsvol:/data --network nfsnet -p 2049:2049 --name nfssrv ozgurozturknet/nfs:latest
```

```
kubectl apply -f pv.yaml   # Persistent volume oluşturulur
kubectl get pv             # Persistent Volume objeleri listelenir
```

Buraya kadar olan kısım sistem yöneticileri tarafından yapılır.
Developer persistent volume talep ettikten sonra
claim objeleri oluşturur.
Böylece persistent volume ile ilgili detaylarla ilgilenmez.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysqlpv
  labels:
    app: mysql
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /
    server: 10.255.255.10
```

```
kubectl apply -f pvc.yaml # Persistent volume claim oluşturulur
kubectl get pvc           # Persistent Volume Claim objeleri listelenir
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysqlclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi
  storageClassName: ""
  selector:
    matchLabels:
      app: mysql
```

İçeriği gösterilen deploy.yaml uygulamamızı apply edelim;

```
kubectl apply -f deploy.yaml
```

Bir yandan da izleyelim;

```
watch kubectl get pods -o wide
```

Pod ile ilgili tanımlamalara ve detaylara bakalım;

```
kubectl describe pod mysqldeployment-59cc49666b-tthtv
```

```
minikube node add # Yeni bir node eklenir
kubectl get nodes # node'lar listelenir
```

minikube node'na bir taint ekleyerek mysqldeployment'ı izleyelim;

```
kubectl taint node minikube a=b:NoExecute
```

Bu komuttan sonra **`a=b taint`**'ini tolere edemeyen pod'lar çalıştırılmaz,
önceden çalışanlar varsa da sonlandırılır.
Pod izlemede; deployment'ın **minikube** node'unda silindiği,
hemen yeni oluşturulan **minikube-m02** node'unda oluştuğu görülür.
Bu node'da oluşan yeni pod'unda aynı nfsvol'u mount ettiği görülür.

pv uygulamasının sonuna gelindi, kaynakları temizlemek için;

```
minikube delete
docker rm -f nfssrv
docker volume rm nfsvol
docker network rm nfsnet # komutları çalıştırılır.
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysqlsecret
type: Opaque
stringData:
  password: P@ssw0rd!
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqldeployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql
          ports:
            - containerPort: 3306
          volumeMounts:
            - mountPath: "/var/lib/mysql"
              name: mysqlvolume
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysqlsecret
                  key: password
      volumes:
        - name: mysqlvolume
          persistentVolumeClaim:
            claimName: mysqlclaim
```

Storage Class

StorageClass, yöneticilerin sundukları depolama "sınıflarını" tanımlamaları için bir yol sağlar. Farklı sınıflar, hizmet kalitesi düzeylerine veya yedekleme ilkelerine veya cluster yöneticilerince belirlenen isteğe bağlı ilkelere eşlenebilir.
Kubernetes, sınıfların neyi temsil ettiği konusunda fikir sahibi değildir.
Bu kavram bazen diğer depolama sistemlerinde "profiller" olarak adlandırılır.

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standarddisk
parameters:
  cachingmode: ReadOnly
  kind: Managed
  storageaccounttype: StandardSSD_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

StorageClass uygulamaları cloud'da daha anlamlıdır. Bu nedenle bu konunun anlatımı için Azure kubernetes servisi seçildi. Bunun için;

`kubectl config use-context aks-k8s-test` # önce context değiştirilir
`kubectl get nodes` # yeni context'in node'ları listelenir
`kubectl get storageclass` # mevcut storageclass'lar listelenir

StorageClass bir template olup,
Persistent Volume yaratmakla uğraşılmadan,
uygun seçenek seçilerek otomatik yaratılır.
Persistent Volume Claim oluşturulurken
Persistent Volume belirtilmezse "default" seçilir.

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standarddisk
parameters:
  cachingmode: ReadOnly
  kind: Managed
  storageaccounttype: StandardSSD_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

PROVISIONER Tedarikçiyi belirtir.
RECLAIMPOLICY disk kullanıldıktan sonra,
iş bitiminde ne olacağını belirler;
- **Delete :** Volume tamamen silinir.
- **Retain :** Volume olduğu gibi kalır.

VOLUMEBINDINGMODE bir pvc oluşturulduğunda ne olacağını belirler;
- **Immediate :** Volume'un hemen oluşturulup pvc'ye bound edilmesini sağlar.
- **WaitForFirstConsumer :** Pod oluşturulup atanana kadar Volume'un oluşturulmamasını sağlar.

ALLOWVOLUMEEXPANSION Volume genişletmeye izin verileceğini belitir.

`kubectl apply -f sc.yaml`
managed-premium ile hemen hemen aynı storage class oluşturur

sc.yaml
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standarddisk
parameters:
  cachingmode: ReadOnly
  kind: Managed
  storageaccounttype: StandardSSD_LRS
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
```

pvc.yaml
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysqlclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi
  storageClassName: "standarddisk"
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysqlsecret
type: Opaque
stringData:
  password: P@ssw0rd!
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysqldeployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql
          ports:
            - containerPort: 3306
          volumeMounts:
            - mountPath: "/var/lib/mysql"
              name: mysqlvolume
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysqlsecret
                  key: password
      volumes:
        - name: mysqlvolume
          persistentVolumeClaim:
            claimName: mysqlclaim
```

`watch kubectl get pvc` # Persistent Volume Claim'ler izlenir
`watch kubectl get pv` # Persistent Volume'ler izlenir

`kubectl apply -f pvc.yaml` # pvc oluşur ve Pending'te kalır
Çünkü **StorageClass** bind modu **WaitForFirstConsumer**'dır, pv oluşmaz.

`kubectl apply -f deploy.yaml` # deployment oluşturulur
pvc'nin **Pending** olan STATUS'u **Bound** olur, sonrasında da **pv** oluşur.

StatefulSet

Cassandra Kurulumu

Singleton pod

- İşlemler manuel ve zahmetli.
- Singelton podun fail durumunu kontrol eden bir mekanizma yok.
- Yeni bir pod eklemek/çıkarmak istenirse tüm sürec yönetilir.

Deployment

- Pod isimleri rastgele oluşuyor.
- Podlar aynı anda oluşturuluyor.
- Deployment podları sırayla değil rastgele siliyor.

StatefulSet

- 1: StatefulSet tarafından oluşturulan her podun, stateful set tanımında belirlenen pvc'e göre oluşturulan bir pv'si olur.
Yani her pod'un kendine ait bir pv'si olur.
- 2: StatefulSet altında podlar sırayla oluşturulup/silinir.
Mesela 3 podlu bir stateful set oluşturulduğunda öncelikle pod0 oluşturulur. Bu pod ayağa kalkıp readiness ve liveness checkinden geçip işlemlerini tamamlamadan bir sonraki pod oluşturulmaz.
Bu pod hazır running durumuna geçince bir sonraki pod oluşturulur.
Aynı şekilde 3. Pod da 2. Pod tamamlanmadan oluşturulmaz.
Tam tersi durumda da bu geçerlidir, 3 podlu bir statefulset 2 Pod'a indirilirse kubernetes random bir pod seçip onu silmez.
En son yaratılan pod, ilk olarak silinir.
- 3: StatefulSet tarafından oluşturulan her pod'a statefulsetin_adı-0 1 2 3 şeklinde devam eden sabit bir isim verilir.
İsimler random seçilmez ve aynı zamanda containera hostname olarak da atandığından, her uygulama bu isimle ulaşılabilir.

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  template:
```

`watch kubectl get pods` # pod'lar izlenir
`watch kubectl get pvc` # pvc'ler izlenir
`kubectl apply -f statefulset.yaml` # ilk pod cassandra-0 Ready olduktan sonra 1, daha sonra da 2 oluşur. Her pod'dan önce pvc'si oluşuyor.
`kubectl exec cassandra-0 -- nodetool status` # 3 instance'lı cassandra cluster'ı listeler
`kubectl scale statefulset cassandra --replicas=5` # cluster node sayısını 5 yapar
`kubectl exec cassandra-0 -- nodetool status` # cassandra cluster'daki instance 5 olur
`kubectl delete pod cassandra-2` # pod manuel silinirse aynı isimle ve aynı pvc ile tekrar oluşturulur
`kubectl scale statefulset cassandra --replicas=3` # cluster node sayısını tekrar 3'e düşürür, ilk olarak en sonuncu silinir
`kubectl get svc` # cassandra adlı clusterIP tipinde internal IP atanmamış None set edilmiş bir servis görülür

Headless Services

Bazen yük dengelemeye ve tek bir Service IP'sine ihtiyaç olmaz. Bu durumda, cluster IP'si için açıkça "None" belirterek "headless" Service olarak adlandırılan bir servis oluşturulabilir.
`kubectl run -it test --image=busybox -- sh` # başka bir pod oluşturarak cassandra servisine erişmek istenirse;

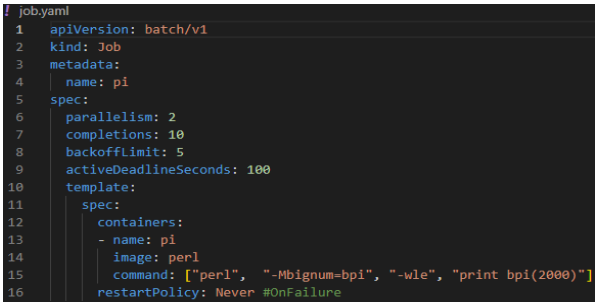
- Her "ping cassandra" komutuna farklı bir IP'den cevap verilir. Sanki ortamda bir loadbalancer var gibi her isteğe farklı bir node cevap verir.
- "ping cassandra-1.cassandra" komutu ile belli bir node seçilip, erişim sağlanabilir.

statefulset.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cassandra
  name: cassandra
spec:
  clusterIP: None
  ports:
  - port: 9042
  selector:
    app: cassandra
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      terminationGracePeriodSeconds: 1800
      containers:
      - name: cassandra
        image: gcr.io/google-samples/cassandra:v13
        imagePullPolicy: Always
        ports:
        - containerPort: 7000
          name: intra-node
        - containerPort: 7001
          name: tls-intra-node
        - containerPort: 7199
          name: jmx
        - containerPort: 9042
          name: cql
        resources:
          limits:
            cpu: "500m"
            memory: 1Gi
          requests:
            cpu: "500m"
            memory: 1Gi
        securityContext:
          capabilities:
            add:
              - IPC_LOCK
        lifecycle:
          preStop:
            exec:
              command:
              - /bin/sh
              - -c
              - nodetool drain
        env:
          - name: MAX_HEAP_SIZE
            value: 512M
          - name: HEAP_NEWSIZE
            value: 100M
          - name: CASSANDRA_SEEDS
            value: "cassandra-0.cassandra.default.svc.cluster.local"
          - name: CASSANDRA_CLUSTER_NAME
            value: "K8Demo"
          - name: CASSANDRA_DC
            value: "DC1-K8Demo"
          - name: CASSANDRA_RACK
            value: "Rack1-K8Demo"
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        readinessProbe:
          exec:
            command:
            - /bin/bash
            - -c
            - /ready-probe.sh
          initialDelaySeconds: 15
          timeoutSeconds: 5
        volumeMounts:
        - name: cassandra-data
          mountPath: /cassandra_data
  volumeClaimTemplates:
  - metadata:
      name: cassandra-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: standard
      resources:
        requests:
          storage: 1Gi
```

Job

Bir Job objesi, bir veya daha fazla pod oluşturur ve belirli bir sayıda pod başarıyla sonlandırılana kadar pod yürütmeyi yeniden denemeye devam eder. Job belirtilen sayıda pod'un başarıyla tamamlanma durumunu izler. Belirtilen sayıda başarılı tamamlamaya ulaşıldığında, Job (yani görev) tamamlanır. Bir Job'un silinmesi, oluşturduğu podları temizler.




```
watch kubectl get pods # pod'lar izlenir
watch kubectl get jobs # job'lar izlenir

kubectl apply -f job.yaml # pod'lar ve job oluşur
kubectl logs pi-2hqs6 # pod logundan pi görülür
kubectl delete jobs.batch pi # job'u siler, pod'lar da silinir
```

CronJob

Bir CronJob nesnesi, bir crontab (cron tablosu) dosyasının bir satırı gibidir. Belirli bir zamanlamaya göre, Cron formatında yazılmış bir Job'u periyodik olarak çalıştırır.

```yaml
apiVersion: batch/v1beta1 # not stable until kubernetes 1.21.
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Authentication

Kurumsal uygulamalarda kullanıcı yönetimi ve authetication uygulama içerisinde gerçekleştirilir.

Kubernetes Authetication

Kubernetes; kullanıcı hesaplarının yaratılıp, yönetilebileceği bir altyapı sunmaz. Kubernetestte Kullanıcı oluşturma ve kimlik doğrulama cluster dışında gerçekleştirilecek şekilde tasarlanmıştır.

- X509 Client Certs
- Static Token File
- OpenID Connect Tokens
- Webhook Token Authentication
- Authenticating Proxy

birini ya da birkaçını kullanarak bu işlemin kubernetes dışında yapılmasını sağlar. Kubernetes kurulumunda, kube-api server ayarlarında bu altyapılardan hangilerinin kullanılacağı belirlenir ve kimlik doğrulama bu altyapılar üzerinden gerçekleştirilir.
Kubernetes, kimlik doğrulama eklentileri aracılığıyla API isteklerinin kimliğini doğrulamak için istemci sertifikaları, taşıyıcı belirteçleri, bir kimlik doğrulama proxy'si veya HTTP temel kimlik doğrulaması kullanır.
API'de bir kullanıcı objesi olmamasına rağmen, cluster'ın certificate authority'si tarafından imzalanmış geçerli bir sertifika sunan herhangi bir kullanıcı (CA) kimliği doğrulanmış kabul edilir. Bu yapılandırmada Kubernetes, sertifikanın 'konu' kısmındaki ortak ad alanından kullanıcı adını belirler (ör. "/CN=bob").
Oradan, rol tabanlı erişim denetimi (RBAC) alt sistemi, kullanıcının bir kaynak üzerinde belirli bir işlemi gerçekleştirme yetkisine sahip olup olmadığını belirler.

Kubernetes, bu ve diğer altyapılar tarafından kimliği doğrulanmış kullanıcıların kubernetes ile görüşmesine imkan verir fakat iş burada bitmez. Kimliği doğrulanması bir kullanıcının cluster üstünde her şeyi yapabileceği anlamına gelmez. Burada yetkilendirme yani authorization kavramı devreye girer.

Manuel bir kubernetes kurmak yerine minikube servisleri gibi yapılar kullanılıyorsa bu işlemler otomatik olarak yapılır. minikube varsayılan olarak X509 Client Certs ile kimlik doğrulama yapacak şekilde ayarlanır ve "minikube start" denildiğinde admin yetkisine sahip minikube kullanıcısı için sertifikalar hazırlanır, kubectl config dosyasına yazılır, kubectl aracı ile cluster'a erişilmek istendiğinde bu sertifikalar ile kimlik doğrulaması yapılmış olarak istek gönderilir.

Key ve CSR oluşturma

`openssl genrsa -out ozgurozturk.key 2048` # private key oluşturulur
Bu anahtar kullanılarak bir Certificate Signing Request yani CSR hazırlamak gerekir, bu dosya sertifika sağlayıcıya gönderilerek dosyadaki özelliklerde bir dijital sertifika isteme imkanı verir. Bu dosya ile admin kendisine gelen bu isteği kubernetes'in sertifika yetkilisi (certificate authority) ile onaylayıp imzalayarak kullanıcının sertifikasını oluşturur ve kullanıcıya gönderir. Kullanıcı bu sertifikayı kullanarak kubernetes'e bağlanır.

```
openssl req -new -key ozgurozturk.key -out ozgurozturk.csr -subj "/CN=ozgur@ozgurozturk.net/O=DevTeam"  # csr dosyası oluşrulur
```

CertificateSigningRequest oluşturma

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: ozgurozturk
spec:
  groups:
  - system:authenticated
  request: $(cat ozgurozturk.csr | base64 | tr -d "\n")
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - client auth
EOF
```

CSR Onaylama ve CRT'yi Alma

`kubectl get csr` # CONDITION Pending yani Onay bekliyor

`kubectl certificate approve ozguroztürk` # Approved,Issued yani Onaylandı, Yayınlandı

`kubectl get csr ozguroztürk –o yaml` # csr yaml olarak output edilir

`kubectl get csr ozguroztürk -o jsonpath='{.status.certificate}' | base64 -d >> ozguroztürk.crt` # csr jsonpath olarak output edilirken status'un altındaki certificate'i pipe ile base64 gönder -d parametreside decode edilmesini sağlar ve sonuç ozguroztürk.crt dosyasına yazılır

`cat ozguroztürk.crt` # certificate hazır

Bu sertifika dosyası developer veya talep sahibine mesaj vb. yolla iletilir.

Talep Sahibinin Onaylanan Sertifika Dosyası Geldikten Sonra Yapacağı İşlemler

ozguroztürk.crt dosyası kubectl config dosyasına eklenerek yeni bir context yaratılır ve kubernetes cluster'a bu context üzerinden bağlanılabilir.

- kubectl config ayarları

`kubectl config set-credentials ozgur@ozguroztürk.net --client-certificate=ozguroztürk.crt --client-key=ozguroztürk.key` # User "ozgur@ozguroztürk.net" set.

`kubectl config set-context ozguroztürk-context --cluster=minikube --user=ozgur@ozguroztürk.net` # Context "ozguroztürk-context" created.

`kubectl config use-context ozguroztürk-context` # Switched to context "ozguroztürk-context".

Artık ozgur@ozguroztürk.net olarak cluster'a bağlanılabilir.

`kubectl get pods` # Error from server (Forbidden): pods is forbidden: User "ozgur@ozguroztürk.net" cannot list resource "pods" in API group "" in the namespace "default"

Buraya kadar yapılanlar kubernetes'e bağlanma authentication (kimlik doğrulama) ile ilgiliydi ve varsayılan olarak listeleme dahil hiç bir işlem yapılamaz yani sıfır yetki ile gelir.

Kullanıcı authenticate oldu ama authorization yani yetkilendirme olmadığından işlem yapamadı.

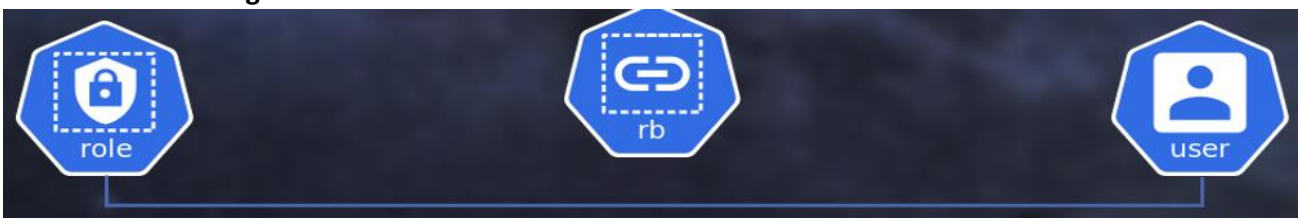
Role Based Access Control "RBAC"

Rol tabanlı erişim denetimi (RBAC), kurumdaki kullanıcıların rollerine dayalı olarak cihaz veya ağ kaynaklarına erişimi düzenleme yöntemidir.

RBAC objeleri

RBAC yetkilendirmesi, yetkilendirme kararlarını yönlendirmek için **rbac.authorization.k8s.io** API grubunu kullanır ve Kubernetes API aracılığıyla ilkeleri dinamik olarak yapılandırmaya olanak tanır.

Role ve Role Binding




```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core
  resources: ["pods"] # "services", "endpo
  verbs: ["get", "watch", "list"] # "get",
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: ozgur@ozgurozturk.net # "name" is case
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name
  apiGroup: rbac.authorization.k8s.io
```

Role, **Role Binding**, **Cluster Role** ve **Cluster Role Binding** şu ana kadar gördüğümüz diğer objeler gibi birer kubernetes objesidir. Bu dört obje de **rbac.authorization.k8s.io/v1** API'sinde bulunur. Tanımlar **rules** altında yapılır. Burada tek bir rule oluşturuldu ama istenirse birden çok rule oluşturulabilir. Her rule yani kuralda üç şey belirlenir.

- **apiGroups**: Hangi API grubundaki objelerle ilgili olduğu burada belirtilir. Boş ise core API group ile ilgili olduğunu yani v1 API'sindeki objelerle ilgili yetkilendirme yapıldığını gösterir.
- **resources:** Hangi kaynaklarla ilgili olduğu belirtilir. Bu örnekte core API'deki "**pods**" objesi ile ilgili bir kural yazıldığı belirtildi. Diğer objelerde "**services**", "**endpoints**", "**pods**", "**pods/log**" eklenebilir.
- **verbs:** Yetkiler burada tanımlanır. Tam liste "**get**", "**list**", "**watch**", "**post**", "**put**", "**create**", "**update**", "**patch**", "**delete**" şeklindedir. "**get**" tekil kaynakları okumayı, "**list**" objelerin tüm özelliklerini listelemeyi, "**watch**" bir veya birden fazla kaynağın özelliklerini göstermeyi, "**post**" kaynak oluşturmayı, "**put**" kaynak üstünde değişiklik yapmayı, "**create**" kaynak yaratmayı, "**update**" güncellemeyi, "**patch**" birden fazla kaynakta değişiklik yapmayı, "**delete**" silmeyi sağlar.

Bu **Role**'de **metadata** altında **namespace** olarak **default** girilmiştir. **Role** ile **Cluster Role** arasındaki fark budur. Örnekteki clusterrole.yaml dosyası ile role.yaml dosyası arasındaki iki farktan ilki **kind:** tanımında **Role** yerine **ClusterRole**, ikinci olarakta **Cluster Role** dosyası **metadata** altında **namespace** tanımı içermez. Çünkü **Cluster Role**'de tanımlanan yetki **tüm cluster**'da geçerlidir.

Her obje genellikle bir namespace altında yaratıldı. Fakat kubernetes'te namespace'e bağlı olmayan cluster seviyesinde objeler de vardır. Mesela node'larda bir kubernetes objesi ama bir namespace'e bağlı değildir. None namespace türü nesnelerle ilgili yetkilendirme yaparken de Cluster Role kullanılır.
Yetki belirlendikten sonra kullanıcılara bağlanmak için **Role Binding** yapılır.

```yaml
subjects:
- kind: User
  name: ozgur@ozgurozturk.net # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role  # bu Role veya ClusterRole olur
  name: pod-reader  # bu bağlamak istediğimiz Role veya ClusterRole name ile eşleşmelidir.
  apiGroup: rbac.authorization.k8s.io
```

subjects kısmında önce rol bağlamanın yapılacağı kullanıcı, sonra da bağlanacak nesne **roleRef** belirtilir.

Cluster Role Binding'te **Role Binding** ile aynıdır. Sadece **Role** yerine **Cluster Role** bağlanır. Bu örnekte atama için bu kez **User** yerine **Group** kullanıldı, grup üyesi tüm kullanıcılar yetkilendirilir.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: DevTeam # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

`kubectl config current-context` # minikube
`kubectl config use-context ozgurozturk-context` # Switched to context "ozgurozturk-context".
`kubectl config current-context` # ozgurozturk-context
`kubectl config use-context minikube` # Switched to context "minikube".
`kubectl apply -f .` # bulunulan dizindeki tüm yaml dosyalarını apply eder, kube-apiserver'a gönderir
`kubectl get roles` # NAME'i pod-reader olan role objesi döner
`kubectl get rolebindings.rbac.authorization.k8s.io` # NAME ve ROLE olarak; read-pods ve Role/pod-reader döndürür

`kubectl get clusterrole` # system tarafından kullanılan cluster roller listelenir
`kubectl get clusterrolebinding.rbac.authorization.k8s.io` # system tarafından kullanılan cluster role binding'ler listelenir

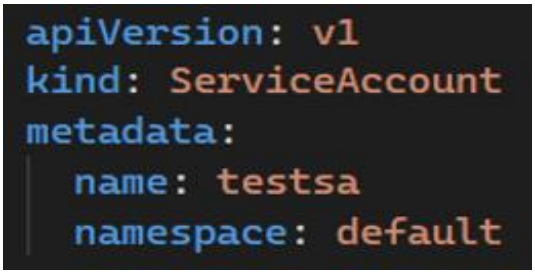
`kubectl config set-context ozgurozturk-context --cluster=minikube --user=ozgur@ozgurozturk.net`
`kubectl config use-context ozgurozturk-context` # Switched to context "ozgurozturk-context".
`kubectl get pods` # No resources found in default namespace. (Yetkilendirme yapıldıktan sonra hata vermedi)

`kubectl get svc` # Service için yetkilendirme yapılmadığından hata verdi
Error from server (Forbidden): services is forbidden: User "ozgur@ozgurozturk.net" cannot list resource "services" in API group "" in the namespace "default"

`kubectl get pods -n kube-system` # kube-system namespace'i için yetkilendirme yapılmadığından hata verdi
Error from server (Forbidden): pods is forbidden: User "ozgur@ozgurozturk.net" cannot list resource "pods" in API group "" in the namespace "kube-system"
"ozgur@ozgurozturk.net" kullanıcısına sadece default namespace'de yetkilendirme yapıldı.

`kubectl get secrets -A` # Tüm namespace'lerdeki secret'lar listelenir

Service Account



Service accounts, podlarda çalışan processler tarafından kullanılmak üzere tasarlanmıştır.

```
kubectl apply -f serviceaccount.yaml
kubectl get sa # service account'lar listelenir
kubectl get pods # pod'lar listelenir, testpod döner
kubectl exec -it testpod -- bash # pod'a bağlanılır
kubectl get svc # service'leri listeler ve kubernetes isimli ClusterIP tipi servisi gösterir
```

testpod'un bash'inde;

```
curl --insecure https://kubernetes # system:anonymous olarak erişildi Forbidden 403 döndü
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl --insecure https://kubernetes --header "Authorization:Bearer $TOKEN" #
```

system:serviceaccount:default:testsa olarak erişildi ama yine Forbidden 403 döndü

```
curl --insecure https://kubernetes/api/v1/namespaces/default/pods --header "Authorization:Bearer $TOKEN" #
```

uzun bir json döndürdü

```
curl --insecure https://kubernetes/api/v1/namespaces/default/pods --header
"Authorization:Bearer $TOKEN" | jq '.items[].metadata.name' # "testpod" döndü
```

`serviceaccount.yaml` dosyası;

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: testsa
  namespace: default
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: podread
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: testsarolebinding
  namespace: default
subjects:
- kind: ServiceAccount
  name: testsa
  apiGroup: ""
roleRef:
  kind: Role
  name: podread
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  namespace: default
spec:
  serviceAccountName: testsa
  containers:
  - name: testcontainer
    image: ozgurozturknet/k8s:latest
    ports:
    - containerPort: 80
```

Ingress

Bir servis sağlayıcının yönetilen kubernetes hizmetlerinden faydalanıldığını varsayalım.

Mesela Azure kubernetes servis üzerinde bir uygulama deploy ederek, loadbalancer tipi bir servis expose'u ile dış dünyadan erişim sağlayalım. Deploy edilen her uygulama için bir loadbalancer ve bir public IP gerektiği anlamına gelir. Gereksiz maliyet artışının yanında yönetimi de zorlaşır.

Mikroservis mimarisinde bir uygulama; istemci **`www.example.com`** adresini girdiğinde **A** uygulaması, **`www.example.com/contact`** adresini girdiğinde ise **B** uygulaması tarafından sunulduğunu düşünelim. Mevcut loadbalancer servisi ile bu ortamı kurgulamak imkansızdır. Çünkü DNS'te path based bir tanım yapılamaz.

Kubernetes bu iki sorunu Ingress ve Ingress Controller objeleri ile çözer.

Ingress Controller

L7 Application Loadbalancer kavramının Kubernetes spesifikasyonlarına göre çalışan ve Kubernetes'e deploy ederek kullanabildiğimiz türüdür.

Nginx, HAproxy, Traefik
en bilinen ingress controller uygulamalarıdır.

Ingress

Genellikle HTTP olmak üzere bir clusterdaki servislere harici erişimi yöneten bir API nesnesidir.

Yük dengeleme, SSL sonlandırması ve path-name tabanlı yönlendirme özelliklerini destekler.

Uygulama Local Ingress Uygulaması olarak windows powershell'de gerçekleştirilecek.

minikube Ayarları

- Ingress çalıştırmak için minikube driver'ı değiştirmek gerekir;
  - Windows için **Hyper-V**, macOS ve Linux için **VirtualBox** seçilebilir. Kurulu olması gerekir.

```
minikube start --driver=hyperv
```

Ingress Controller Seçimi ve Kurulumu

- Her ingress controller kurulumu farklıdır ve nginx ile devam edilecektir. Kurulum detayları her uygulamanın kendi web sitesinden öğrenilebilir.

  Kurulum detayları; https://kubernetes.github.io/ingress-nginx/deploy/

- minikube, yoğun olarak kullanılan nginx gibi bazı ingress controller'ları daha hızlı aktif edebilmek için addon olarak sunmaktadır.

```
minikube addons enable ingress  # ingress addonunu aktif eder.
minikube addons list  # tüm addon'ları listeler.
```

- **Nginx** kurulduğu zaman kendisine **ingress-nginx** adında bir **namespace** yaratır.

```
kubectl get all -n ingress-nginx   # ingress-nginx namespace'ine ait tüm objeleri listeler

kubectl apply -f .\deploy.yaml  # yaml dosyasını uygular yani apply eder
kubectl get deployment  # deployment'ları listeler (blueapp, greenapp, todoapp)
kubectl get svc  # service'ları listeler (bluesvc, greensvc, todosvc)
```

Üç uygulamaya cluster içinde ulaşılmasını sağlayan üç servis oluşmuştur. Ancak servisler **clusterIP** tipindedir ve **sadece cluster içinden erişilmesini** sağlar. **Dış dünyadan** erişilmesi istenirse **LoadBalancer** tipine çevirmek gerekir ancak **a.com/blue** isteği **blue** servisine, **a.com/green** isteği **green** servisine **yönlendirilemez**. Bunun için **L7 Application Loadbalancer** servisi üzerinden expose edilmesi gerekir. Bunun için gerekli obje **Ingress controller**'dur.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blueapp
  labels:
    app: blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: blue
  template:
    metadata:
      labels:
        app: blue
    spec:
      containers:
      - name: blueapp
        image: ozgurozturknet/k8s:blue
        ports:
        - containerPort: 80
        livenessProbe:
          httpGet:
            path: /healthcheck
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
        readinessProbe:
          httpGet:
            path: /ready
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 3
---
apiVersion: v1
kind: Service
metadata:
  name: bluesvc
spec:
  selector:
    app: blue
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---



apiVersion: apps/v1
kind: Deployment
metadata:
  name: greenapp
  labels:
    app: green
spec:
  replicas: 2
  selector:
    matchLabels:
      app: green
  template:
    metadata:
      labels:
        app: green
    spec:
      containers:
      - name: greenapp
        image: ozgurozturknet/k8s:green
        ports:
        - containerPort: 80
        livenessProbe:
          httpGet:
            path: /healthcheck
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
        readinessProbe:
          httpGet:
            path: /ready
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 3
---
apiVersion: v1
kind: Service
metadata:
  name: greensvc
spec:
  selector:
    app: green
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---



apiVersion: apps/v1
kind: Deployment
metadata:
  name: todoapp
  labels:
    app: todo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      containers:
      - name: todoapp
        image: ozgurozturknet/samplewebapp:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: todosvc
spec:
  selector:
    app: todo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Ingress Objelerini Deploy Etme ve Ayarlama

- Load balancer için gerekli olan Ingress Controller Nginx olarak seçildi ve kuruldu.
- Her bir app için gerekli olan ClusterIP tipinde servisleri de kurduktan sonra, sıra istemcilerin **example.com/a** ile **A** service'ine gitmesi için gerekli **Ingress obje**lerini de deploy etmeye geldi.

blue, green app'ler için Ingress Obje tanımlama:

Kurallar **yaml** dosyasının **spec** kısmında belirtilir, **k8sfundamentals.com** domainine ait bir servis yayınlanıyor ve iki path belirtilerek ingress üzerinden dış dünyaya açılıyor. Bu **url**'ye gelen **/blue** isteğinin **/bluesvc**, **/green** isteğinin de **/greensvc** servisi tarafından cevap verilmesi sağlanıyor.

- `pathType` kısmı `exact` veya `Prefix` olarak 2 şekilde ayarlanabilir.

Detaylı bilgi için: https://kubernetes.io/docs/concepts/services-networking/ingress/

Nginx üzerinde ayarlar, annotations üzerinden yapılır.

```
nginx.ingress.kubernetes.io/rewrite-target: /$1
```

appingress.yaml dosyası;

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: appingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: k8sfundamentals.com
      http:
        paths:
          - path: /blue
            pathType: Prefix
            backend:
              service:
                name: bluesvc
                port:
                  number: 80
          - path: /green
            pathType: Prefix
            backend:
              service:
                name: greensvc
                port:
                  number: 80
```

todoingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: todoingress
spec:
  rules:
    - host: todoapp.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: todosvc
                port:
                  number: 80
```

`kubectl apply -f .` # Bulunulan dizindeki tüm yaml dosyalarını apply eder

`kubectl get ingress` # appingress ve todoingress objelerini listeler

```
NAME          CLASS    HOSTS                ADDRESS      PORTS   AGE
appingress    <none>   k8sfundamentals.com   172.2399.145   80      18s
todoingress   <none>   todoapp.com           172.2399.145   80      18s
```

Tanımlanan Ingress Objelerini test etmek için; URL'ler ile simülasyon için, komut çıktısındaki **ADDRESS** ve **HOST** bilgileri aşağıdaki gibi, **C:\Windows\System32\drivers\etc\hosts** dosyasına kaydedilir.

```
172.2399.145 k8sfundamentals.com
172.2399.145 todoapp.com
```

Sonra browser adres satırına;

http://k8sfundamentals.com/blue

http://k8sfundamentals.com/green

http://todoapp.com/

test edebiliriz.