Minikube Komutları

* minikube status # Minikube durumunu inceleyin
* minikube start # Yeni cluster oluştur
* minikube start -p mycluster # Minikube harici yeni cluster oluştur. VirtualBox arayüzünü aç ve kontrol et
* minikube profile list
* minikube start --node 2 # iki node'lu cluster oluştur
* minikube node add # Cluster içerisine yeni bir worker node ekle
* minikube node list # Cluster içerisinde çalışan nodelari listeleyin
* minikube delete # Lokal clusterı silin
* minikube delete --all # Tüm profil clusterı silin
* minikube node delete xxxxxxx # Cluster içerisinde bulunan nodelari silin
* minikube stop # Lokal clusterı durdurun
* minikube version # Minikube versiyon bilgisini sorgulayın
* minikube dashboard # Kubernetes web arayüzü-dashboard ekranına ulaşın
* minikube dashboard --url # Kubernetes web arayüzü-dashboard linkini alın
* minikube ssh # Minikube sanal makinasına-host'una giriş yapın
* minikube image ls # Minikube üzerindeki imagelari listele
* minikube addons list # Minikube üzerindeki addonlari listele
* minikube update-check # Minikube güncellemesini kontrol edin

CLI :

Ne yapmak istiyorsak hangi nesne türü obje ismi varsa opsinyon gir  
kubectl [Command] [Type] [name] [flag]

Get pods proje o- wide

Describe deployment

Delete nodes

Run services

Single pod oluşturma ->kubectl run myweb --image=nginx:latest

Kubectl exec –it myweb –bash

Kubectl delete Pod myweb

Kubectl get Pod myweb

Kubectl describe Pod myweb

Kubectl Logs myweb

Kubectl apply --file myprojectfile.yml

Kubectl explain service

Kubectl create replicaset

Kubectl create deployment

Kubectl create service

Kubectl get pod ,replicaset,deployment-isim , pod-isim , service-isim, nodes-isim

Kubectl describe pod,replicaset,deployment-isim , pod-isim , service-isim, nodes-isim

Kubectl cluster-info -> çalışan clusterı kontrol edebiliriz.

Kubectl get all -> sistemdeki tüm nesneleri listeler

Kubectl get po –o name -> çalışan podların isimleri

Kubectl get po –o wide -> nesne ile ilgli daha fazla detay

Kubectl get nodes –o wide

Kubectl geet po –o myproject-795adasda-64v8 -o yaml -> podu yaml dosyası şeklinde aldık

Kubectl get pod myproject-xxx-xx -o yaml> demo.yaml->yaml dosyası olarak aldık (komut çalıştığı path üzerinde demo.yaml olarak dosay oluşacaktır.)

Kubectl describe po myproject-xxx-xx -> çalışan pod ile ilgili detay. (event önemli hata vs orda gözükür.)

Kubectl describe node minikube -> node ile ilgili tüm detaylar

Kubectl explain pod -> pod nesnesi detayları verir.(manifest)

Kubectl run k8spod1 –image=hello-world

Kubectl logs k8spod1

Kubectl run k8s-pod-2 --image=docker.io/muratsunet/k8s-hellothere --restart=Never

Minikube image list

Minikube dashboard

Kubectl delete po k8s-pod1

Kubectl delete –f xxx.yaml -> yaml dosyası silenbiliriz

Kubectl delete po –all ->tüm podları siler

Kubectl delete all –all -> sistemdeki tüm nesneleri siler

kubectl describe po mixpod->Mixpod detaylı bilgi

kubectl get pods mixpod -o wide-> Mixpod için atanan ip ve node bilgisini görüntüleyin

kubectl get deployment->Sisteme eklenen deployment ismi

kubectl get replicaset->Sisteme eklenen replicaset ismi

Sisteme eklenen deployment detayların->kubectl describe deploy webpod

Sisteme eklenen replicaset detayları->kubectl describe rs webpod-XXXXX

Podlardan birinin detayları->kubectl describe po webpod-XXXXX-YYYYY

mixpod podunu json format->kubectl get po mixpod -o json

mixpod podunu json formatında mixpod.json olarak alma ->kubectl get po mixpod -o json >> mixpod.json

mixpod içerisinde ki ikinci containerın(container-2) image bilgisini JSONPATH formatında komut satırı üzerinde görüntüle->kubectl get po mixpod -o=jsonpath='{.spec.containers[1].image}'

mixpod içerisinde ki üçüncü container ismini JSONPATH formatında komut satırı üzerinde görüntüleyin->kubectl get po mixpod -o=jsonpath='{.spec.containers[2].name}'

mixpod'un hostIP bilgisini JSONPATH formatında komut satırı üzerinde görüntüleyin->kubectl get po mixpod -o=jsonpath='{.status.hostIP}'

mixpod içerisinde ki üçüncü containerın mountPath bilgisini JSONPATH formatında komut satırı üzerinde görüntüleyin->kubectl get po mixpod -o=jsonpath='{.spec.containers[2].volumeMounts[0].mountPath}'

Tüm namespace’deki tüm pod’ları ada göre sıralayarak listeleyin->kubectl get pods -A --sort-by=.metadata.name

Sistemde tanımlı Podları IP'lerine göre küçükten büyüğe doğru sıralayın. Sıralamada IP'lerinde görünmesini sağlayın->kubectl get po -o wide --sort-by=.status.podIP

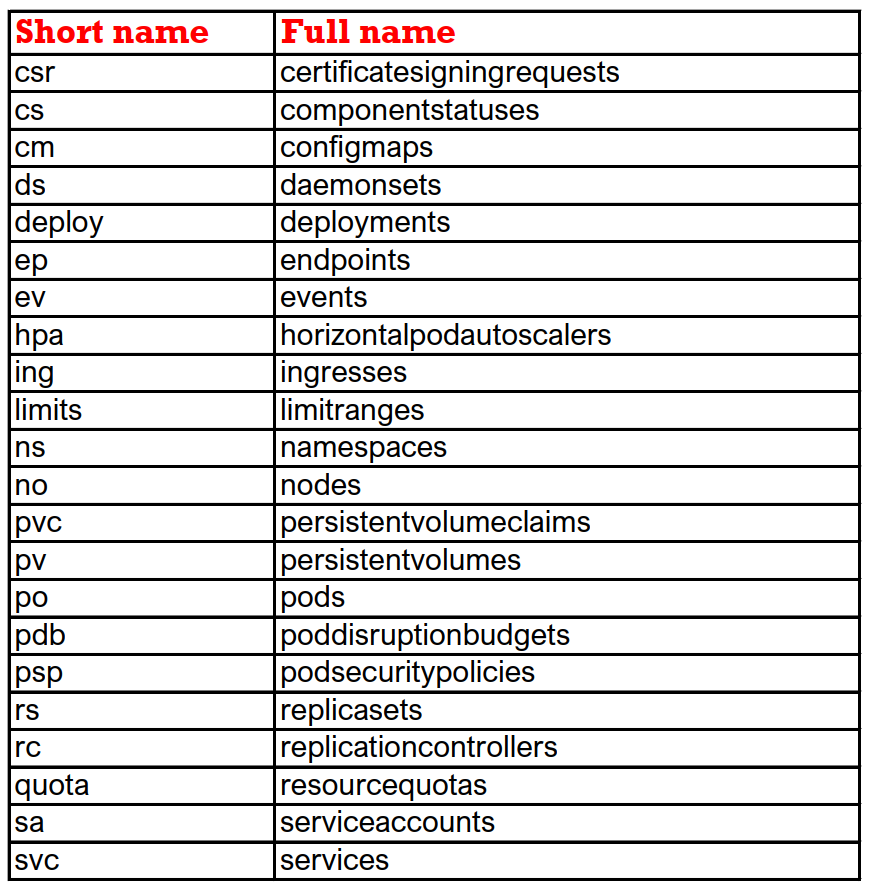
Default namespace’deki tüm pod’ların sadece isimlerini listeleyin->kubectl get pods --output name

Tüm namespace’deki tüm pod’ların sadece isimlerini listeleyin->kubectl get pods -A --output name

**mixpod podunu silin->kubectl delete po mixpod**

**webpod deployment nesnesini silin->kubectl delete deploy webpod**

**project namespace 'i silin->kubectl delete ns project**



**--output=jsonpath Komut Detayları- Önemli**

**Nedir:**

  --output=jsonpath parametresi,   kubectl get pods --output=json çıktısında listelenen değerlere, komut satırı üzerinden tek bir komut ile erişmek için kullanılır. Komut ayraçlar {} içinde yazılır ve JSONPath olarak  tanımlanır.

<https://kubernetes.io/docs/reference/kubectl/jsonpath/>

İlgili link üzerinden JSONPath  detaylarına ulaşabilirsiniz.

**Kullanım:**

Sistemimizde nginx imageını kullanan deneme adında bir POD oluşturduk .

kubectl get pods deneme --output=json  komutu ile baktığımızda deneme POD detayları JSON formatında gösterilecektir. Çıktı aşağıdaki şekilde olacaktır. Bu çıktıdaki verilere komut satırı üzerinden erişmek için -o=jsonpath=  ifadesini kullanacağız. Örneğin kullandığı image bilgisi , oluşturulma zamanı, IP adresi gibi.



(Not: Resimleri sağ tıklayıp fark sekmede göster diyerek büyütebilirsiniz)

kubectl get pods deneme -o=jsonpath= ifadesinden sonra **'{}'**içerisine komutu yazacağız.

**Örneğin:**Yukarıda ilk kırmızı çizgi ile gösterilen name bilgisini okumak istersek **metadata.name** yolunu izleyeceğiz. Çünkü name değişkeni metadata 'nın bir alt başlığıdır.

**Yazacağımız komut**:  kubectl get pod deneme -o=jsonpath='{.metadata.name}'  şeklinde olacaktır.

Metadata altında ki namespace bilgisine erişmek istersek;  
kubectl get po deneme -o=jsonpath='{.metadata.namespace}'

Metadata altında ki creationTimestamp bilgisine erişmek istersek;  
kubectl get po deneme -o=jsonpath='{.metadata.creationTimestamp }'

udi bilgisine erişmek için;  
kubectl get po deneme -o=jsonpath='{.metadata.uid }'

komutunu kullanacağız. Dikkat edilirse komut yazım şekli path yoluna göre yapılmaktadır.

**PEKi ikinci kırmızı çizgi ile gösterilen yolu yani container image bilgisini öğrenmek istersek ne yapacağız?**

Bunun için yazacağımız komut:

kubectl get po deneme -o=jsonpath='{.spec.containers[\*].image}'   şeklindedir. Şimdi bu komutu inceliyelim.

Burada dikkat ederseniz containers ifadesinin önünde [] ve içerisinde \* işareti bulunmaktadır. Container**s** ifadesinden de anlaşılacağı gibi POD içerisinde birden fazla container aynı anda çalıştırılabilir. Bu container'ları birbirinden ayırt etmek için [] işareti kullanılır. \* ifadesi POD içerisinde çalışan tüm containerı kapsa anlamına gelir. [0] şeklinde yazarsak ilk container ı [1] şeklinde yazarsak ikinci container bilgisini getir anlamına gelmektedir

Container ismini almak istersek

kubectl get po deneme -o=jsonpath='{.spec.containers[\*].name}'  komutunu kullanacağız. \* işaretiyle POD içerisindeki tüm container ın isimleri öğrenilmek istenmiştir.

**Peki üst resimdeki mountPath bilgisini almak istersek ne yapacağız;**

Dikkat ederseniz mountPath bilgisi volumeMounts ifadesinin altındadır. volumeMounts bilgiside .spec.containers. yolu üzerindedir. volumeMounts ifadesinin yine çoğul olduğunu görüyoruz. O sebeple bu ifade önünde [\*] tanımını yapacağız. \* ifadesi yine tüm volumeMounts'lar anlamına gelmektedir.

Yazacağımız komut:

kubectl get po deneme -o=jsonpath='{.spec.containers[\*].volumeMounts[\*].mountPath}'

şeklinde olacaktır.

İsim bilgisini almak istersek;

kubectl get po deneme -o=jsonpath='{.spec.containers[\*].volumeMounts[\*].name}'

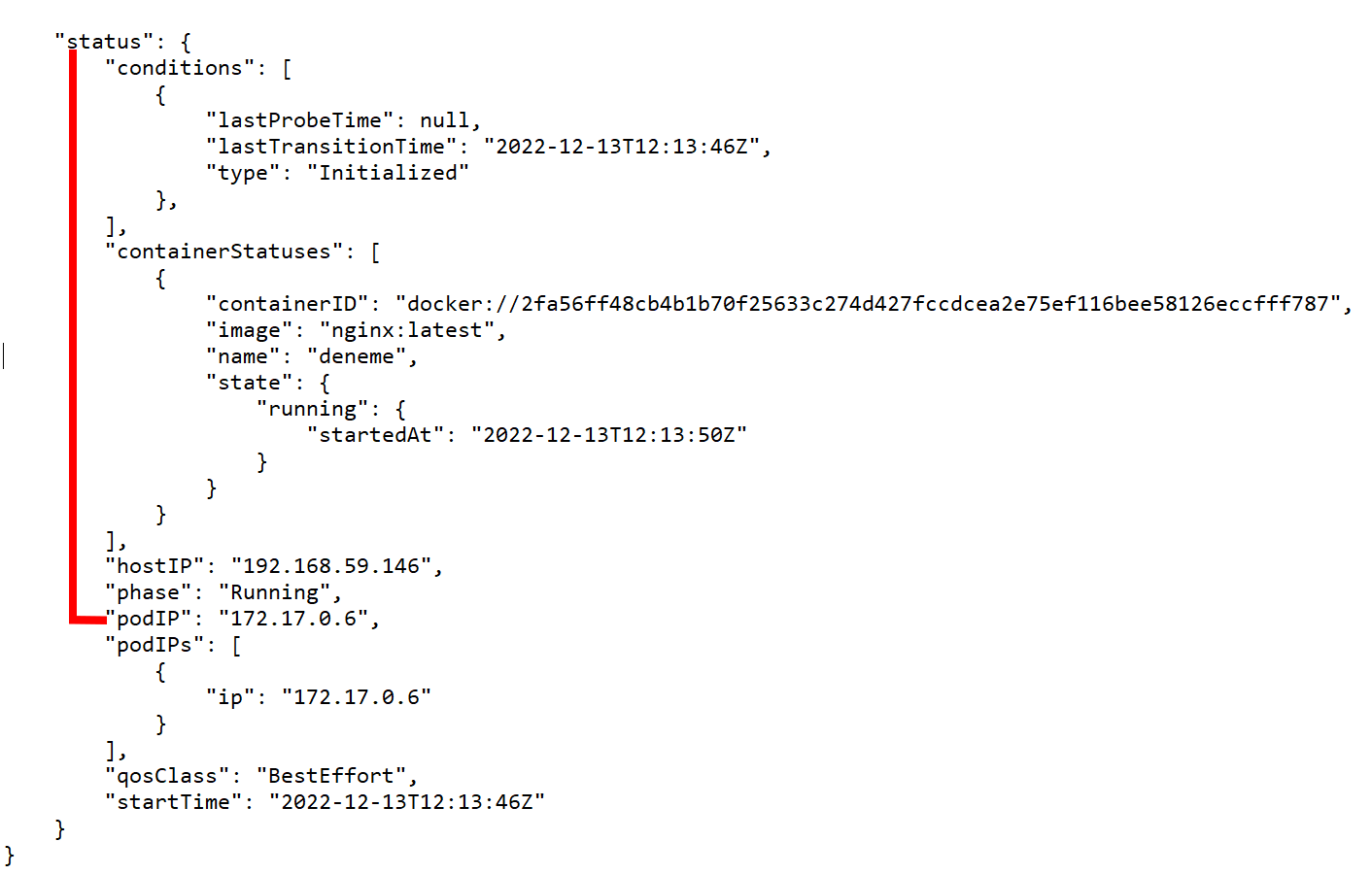
şeklinde yazacağız.

Komut yazımında büyük küçük harf duyarlılığı vardır.

**Farklı bir örneğe bakacak olursak ,** Örneğin alt resim de gözüken POD'un IP bilgisini öğrenmek istediğimizi düşünelim. podIP bilgisinin status başlığı altında olduğunu görüyoruz.

Yazacağımız komut:

kubectl get po deneme -o=jsonpath='{.status.podIP}'

şeklinde olacaktır.

Farklı bir örneğe bakacak olursak; aşağıdaki resimde sistemde ki**tüm nesneler**JSON formatında listelenmiştir.

Sistemdeki tüm podlar üzerinde sorgulama yapmak istediğimizde bu sefer pod ismini yazmadan,

kubectl get pods -o=jsonpath=   şeklinde komutumuzu yazacağız

Akabinde .items[\*] başlığıyla tüm nesneler üzerinde sorgulama yapmak istediğimizi söylüyoruz.

Sonrasında Örneğin ['metadata.name'] diyerek tüm isimleri almak istediğimizi söylüyoruz.

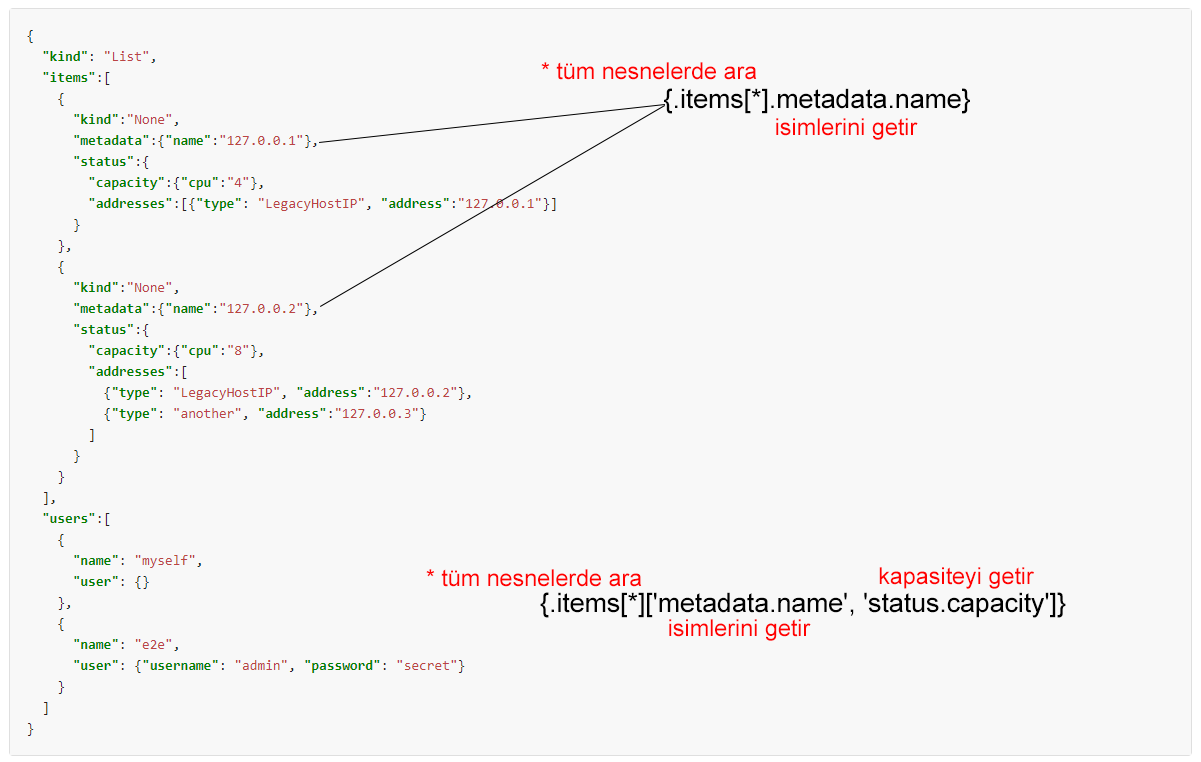
Komutumuz;

kubectl get pods -o=jsonpath="{.items[\*]['metadata.name']}"

şeklinde olacaktır.

Komut çıktısı : 127.0.0.1 ve 127.0.0.2 şeklindedir.

.items[\*] komutunda ki \* yerine .items[0] şeklinde yada .items[1] yazarak hangi objeyle ilgili bilgiye erişmek istiyorsak onu yazabiliriz.

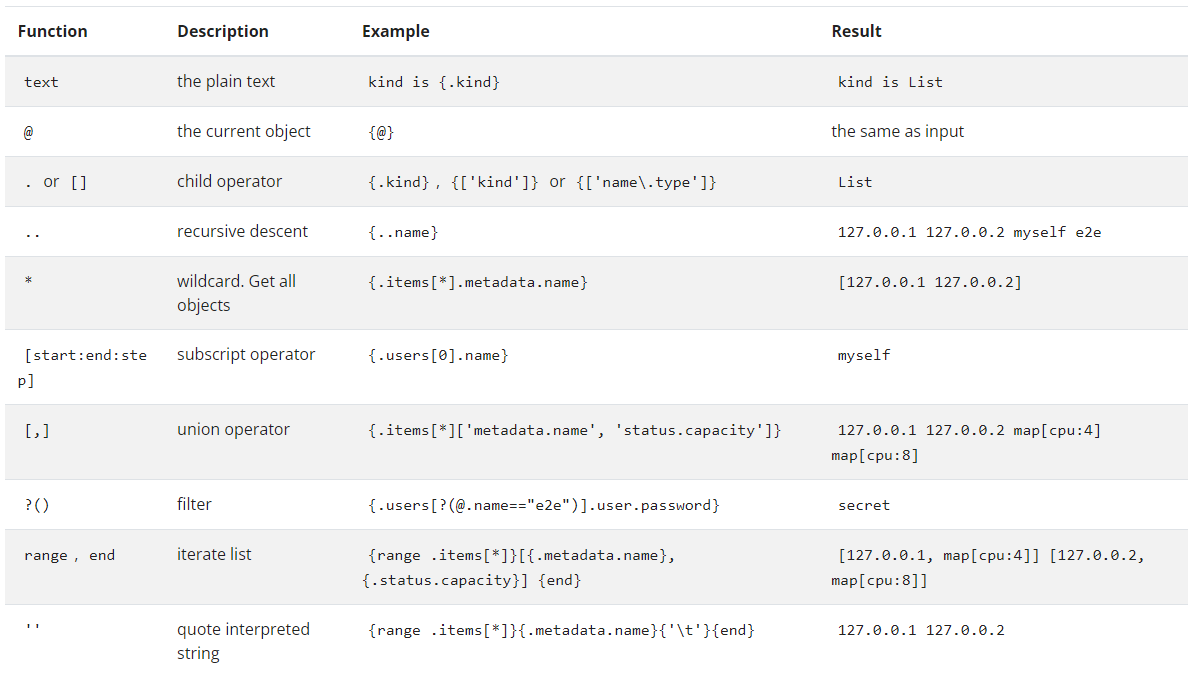
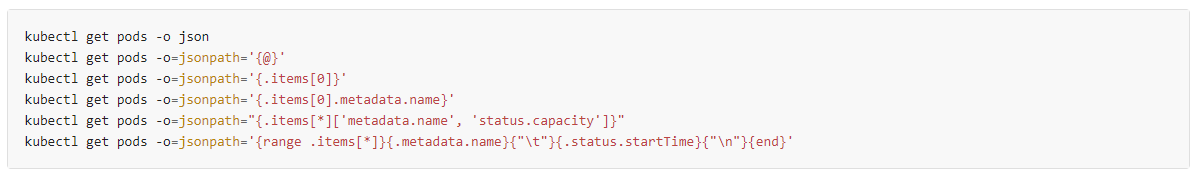


Resim üzerinde ikinci örnekte aynı anda iki tanım sorgulaması yapılmak istenmiştir. Aranacak ifadeler isim ve kapasite bilgisidir. Çıktıya dikkat edilirse capacity ve name bilgisi items[\*].status ifadesinin alt başlığı konumundadır ve yazım işlemi bu path bilgisine göre yapılacaktır.

||| kubectl get pods -o=jsonpath="{.items[\*]['metadata.name', 'status.capacity']}"

Aynı satırda  iki farklı sorgulama işleminde araya virgül konularak sorgulama yapılmıştır.

Aşağıdaki tabloda ifadeler, açıklamaları ve kullanım detayları görülmektedir.

Aşağıdaki örnekleri lütfen inceleyin.

\*\* Kısaca hangi nesneye erişilmek istenirse path bilgisi alınır ve  yukarıdan aşağıya doğru jsonpath takip edilerek komut yazılır.

\*\* Tüm nesneler üzerinde sorgulama yapılacaksa pod ismi yazmadan .items[\*] ifadesiyle sorgulama yapılır.

\*\* Pod ismi yazılırsa doğrudan sorgulama yapılabilir

Aşağıdaki web adresi üzerinden detaylara erişebilirsiniz.

<https://kubernetes.io/docs/reference/kubectl/jsonpath/>

**Sorgulama: (Dikkat)**

Eğer bir çıktı da kelime sorgulaması yapmak istersek;

Windows işletim sisteminde  | findstr "XXXXXX"

Linux işletim sisteminde | grep "XXXXX"

komutunu kullanmaktayız.

Örnek: JSON çıktısında podIP olan satırları getirir.

Windows:   kubectl get pods --output=json | findstr "podIP"

Linux:   kubectl get pods --output=json |  grep "podIP"

Burada sadece grep içerisinde belirtilen kelime satırları gösterilecektir

**--sort-by Komut Detayları - Dikkat**

**Nedir:**

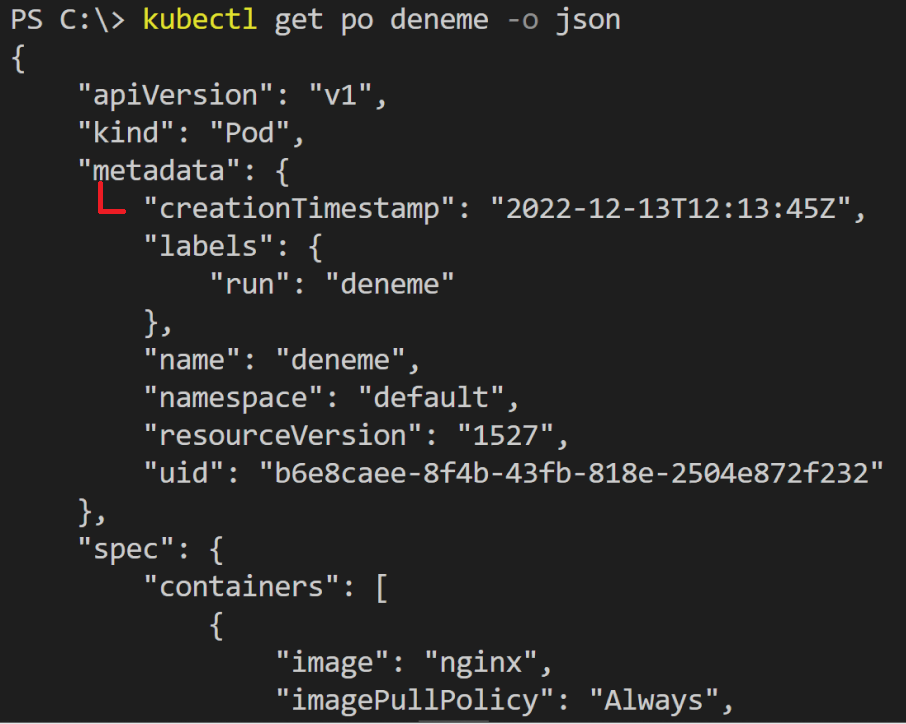
--sort-by ifadesi, kubect get komutu parametrelerindendir. kubectl get --help komutunda detaylara ulaşabilirsiniz.  --sort-by paremetresi önünde yazılan kritere göre çıktı sıralaması yapmak için kullanılır.

**Kullanım:**

--sort-by önünde yazılacak komut JSONPath formatında olmalıdır.

**Sistem üzerinde oluşturulan POD'ları oluşturulma zamanına göre listelemek istiyoruz ?**

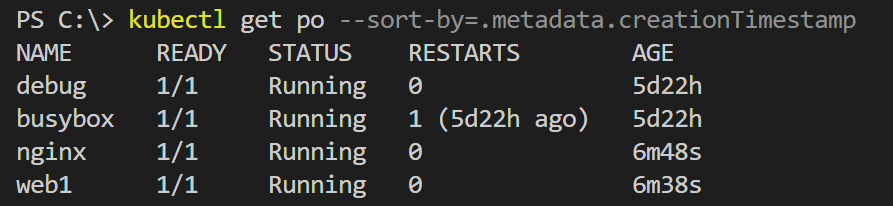
Aşadaki deneme adındaki pod nesnesi JSON formatında gösterilmiştir.  Aşağıdaki kırmızı çizgi ile gösterdiğim path bilgisini takip ederek komutumu yazacağız.



Komutumuz aşağıdaki şekilde olacaktır.

kubectl get po --sort-by=.metadata.creationTimestamp

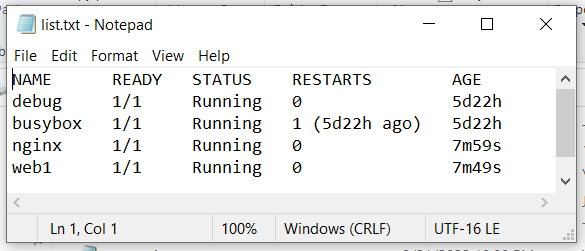
Böylece POD lar oluşturulma tarihine göre listelenecektir. Aşağıda komut çıktısında POD'lar oluşturulma zamanına göre listelenmiştir. AGE kısmında POD'ların süreleri gösterilmektedir.



Peki bu çıktıyı dışarıya aktarmak istersek ne yapacağız ?

Bunun için komut sonuna >> ifadesini kullanabiliriz. Örneğin list.txt şeklinde çıktıyı kayıt edebiliriz.

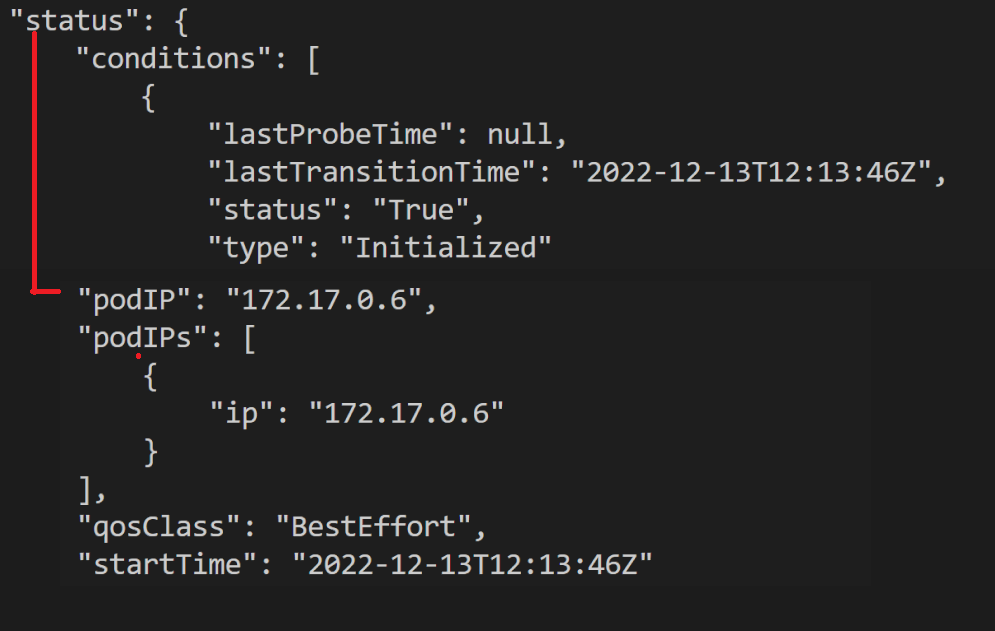
kubectl get po --sort-by=.metadata.creationTimestamp >> list.txt



**Sistem üzerinde bulunan POD'ları IP adreslerine göre sıralamak istiyoruz !**

Bunun için yine JSON çıktısına kontrol edebiliriz. Aşağıda JSON çıktısı gösterilmiştir.

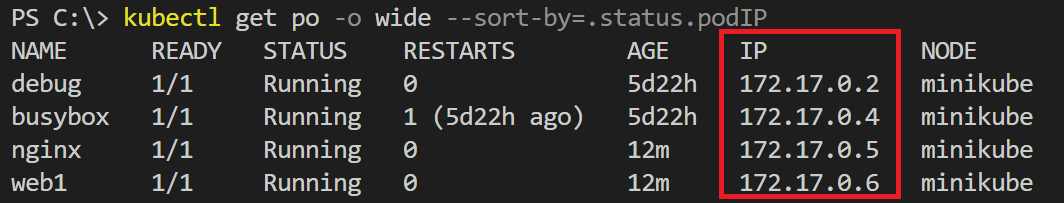
.podIP  ifadesi .status  ifadesinin alt başlığı konumundadır.



Komutumuz aşağıdaki şekilde olacaktır.

kubectl get po --sort-by=.status.podIP

Böylece podlar IP adreslerine göre küçükten büyüğe  sıralanacaktır.



Çıktıyı iplist.txt dosyasına kayıt etmek istersek >> parametresini kullanarak aktaracağız.

kubectl get po -o wide --sort-by=.status.podIP  >> iplist.txt

Yukarıdaki JSON dosyasına bakarak PODların başlama zamanına göre start.txt dosyasına kayıt etmek istersek aşağıdaki komutu kullanacağız.

kubectl get po --sort-by=.status.startTime >> start.txt

\*\* Burada önemli olan nokta JSON path bilgisinin düzgün yazılmasıdır. Bilginize

#### ****--dry-run Komut Detayları - Çok Önemli****

**kubectl run**komutu içerisinde gelen en kullanışlı parametlerinden biri **--dry-run**opsiyonudur.  CKA sertifika sınavında işimizi kolaylaştıracak belki de **en önemli**parametlerden biridir.

**--dry-run**opsiyonu kubectl run komutunda yazdığınız ifadeyi canlı sistem üzerinde çalıştırmadan önce test etmemize imkan sağlamaktadır. -o yaml parametresiyle kullanıldığında bizim için önemli olan gücü ortaya çıkar. **-o yaml**parametresiyle kullanıldığında YAML dosyasını elle yazmamıza gerek kalmadan sistem tarafından  yazdırılmasını sağlar.

Üç opsiyonla birlikte gelmektedir. Bunlar --dry-run= **client**, --dry-run= **server**, --dry-run=**none**

**--dry-run=client -o yaml**komutu, çıktıyı YAML formatında almak için kullanılır.  Böylece komut çalıştırmadan önce ne yapacağını görebiliriz.  =client opsiyonuyla komut api server tarafından doğrulanmadan çalıştırılmış olur. Yani komut api server üzerinden çalıştırılmaz sadece client üzerinde simule edilir.

Örnek Kullanım:

**kubectl run test --image busybox --dry-run=client -o yaml**

Çıktıda ekrana busybox imagını kullanan POD YAML dosyası içeriği yazdırılacaktır.

1. apiVersion: v1
2. kind: Pod
3. metadata:
4. creationTimestamp: null
5. labels:
6. run: test
7. name: test
8. spec:
9. containers:
10. - image: busybox
11. name: test
12. resources: {}
13. dnsPolicy: ClusterFirst
14. restartPolicy: Always
15. status: {}

Çıktıyı dışarıya almak istersek  >> işaretiyle dilediğimiz path üzerinde YAML dosyası formatında aktarabiliriz. Böylece CKA sertifika sınavında YAML dosyasını oturup baştan yazmak yerine sistem tarafından otomatik olarak yazdırılmasını sağlayabiliriz.

**kubectl run test --image busybox --dry-run=client -o yaml >> mybusybox.yaml**

En çok kullanacağımız komut şekli =client formatıdır. Hızlıca YAML dosyasına ihtiyacımız olduğunda bu komutu kullanabiliriz. Özellikle deployment yada servis nesneleri kullandığımızda bu komut işimize oldukça fazla yaramaktadır

**--dry-run=server -o yaml**komutu, çıktıyı YAML formatında almak için kullanılır.  Burada yazılan komut api server tarafından doğrulanmaktadır. Örneğin aynı isimle nesne sistem üzerinde çalışıyorsa çıktıda komut çalıştırılmış gibi uyarı verecektir ama gerçekte komut sistem üzerinden çalıştırılmaz. Sadece simüle edilmektedir.

Örnek

**kubectl run nginx --image=nginx --port=80 --restart=Never --dry-run=server -o yaml**

Çıktı

Error from server (AlreadyExists): pods "nginx" already exists

**--dry-run=none -o yaml**komutu, çıktıyı YAML formatında almak için kullanılır.  Burada yazılan komut api server tarafından doğrulanır. Komut sunucu üzerinde simüle edilir ve çalıştırılır.

**kubectl run nginx --image=nginx --port=80 --restart=Never  --dry-run=none -o yaml**

* Ssh olmadan pod içine bağlanma :

Kubectl exec –it k8s-web-1 --/bin/bash

Mkdir storage

Apt-get update

Apt-get install curl

Curl <http://localhost>

Exit

* Containera bağlanmadan container üzerinde komut çalıştırma:

Kubectl exec k8s-web-1 –date

Kubectl exec k8s-web-1 --mkdir myfile

Kubectl run k8s-python-1 -it –rm –image=python ->--rm ile poddan çıkınca sil dedik.

Kubectl run k8s-mariadb-1 –image=mariadb –restart=Never --env=MYSQL-ROOT-PASSWORD=PassW0rd! ->contaienr image sabir bir değere ihtiyaç duyuyorsa(ROOT-PASSWORD)onuda burada belirttik.

Kubectl exec –it k8smariadb-1 –uroot –p -> bizden password isteyecek password ile pod içerisne bağlanabiliriz

Maridb örneği -> kubectl exec -it mymariadb -- mariadb -uroot -p

Localden Pod’a –Pod’dan Locale Veri kopyalama:

Kubectl cp passwor.txt.k8s-cp-1:tmp/

Neyi nereye

* Çok fazla klasör varsa;

Kubectl cp . k8s-cp-1:tmp/store ( . tüm dosyaları al anlamında)

* Kubectl cp default/k8s-cp-1:tmp/store . ( . ) ile bulunduğumuz klasöre kopyala

**"kubectl explain" Komut Detayları - Hayat kurtarır**

Özellikle CKA sertifika sınavında, takıldığımız yerde bu komut ile işinizi kolaylaştırabilirsiniz.

**Nedir?**

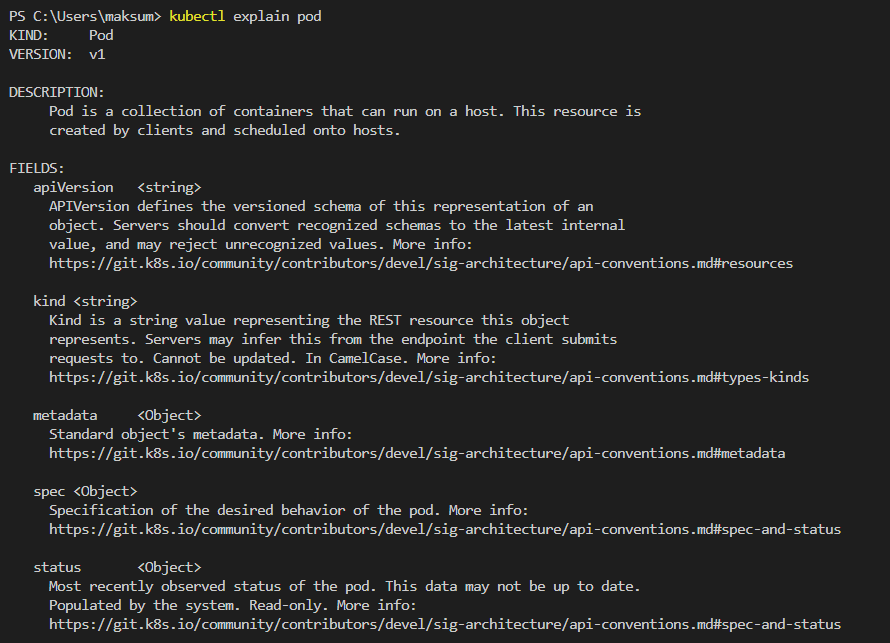
Nesne detayları hakkında bilgi sahibi olmak ve incelediğimiz nesnenin alt parametlerini listelemek istendiğinde kullanılmaktadır. Aynı zamanda bu komut ile incelediğimiz nesnenin kubernetes dökümantasyon sayfasındaki linkine de ulaşabiliyoruz.

**>>**Bildiğiniz gibi CKA sertifika sınavında Kubernetes dökümantosyon sayfasına erişime izin verilmektedir. Nesne detaylarını dökümantasyon sayfası üzerinde de inceleyebiliriz.

**Nasıl Kullanılır**

kubectl explain komutundan sonra detaylarına bakmak istediğimiz nesne ismini yazıyoruz

***kubectl explain pod***-->  Örneğin pod nesnesi detaylarına bakalım.

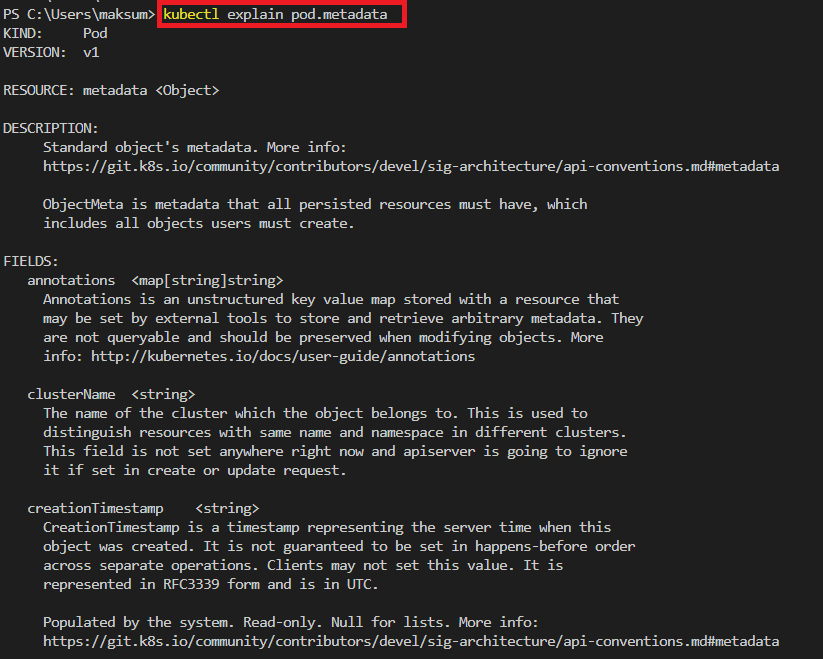


Görüldüğü gibi pod nesnesi detayları ve altında yer alan opsiyonlar listelendi. Bu opsiyonlar ile ilgili detaylar link üzerinde yer almaktadır.

Örneğin pod nesnesi altında  metadata içerisinde yer alan parametreleri listemek istersek

***kubectl explain pod.metadata***

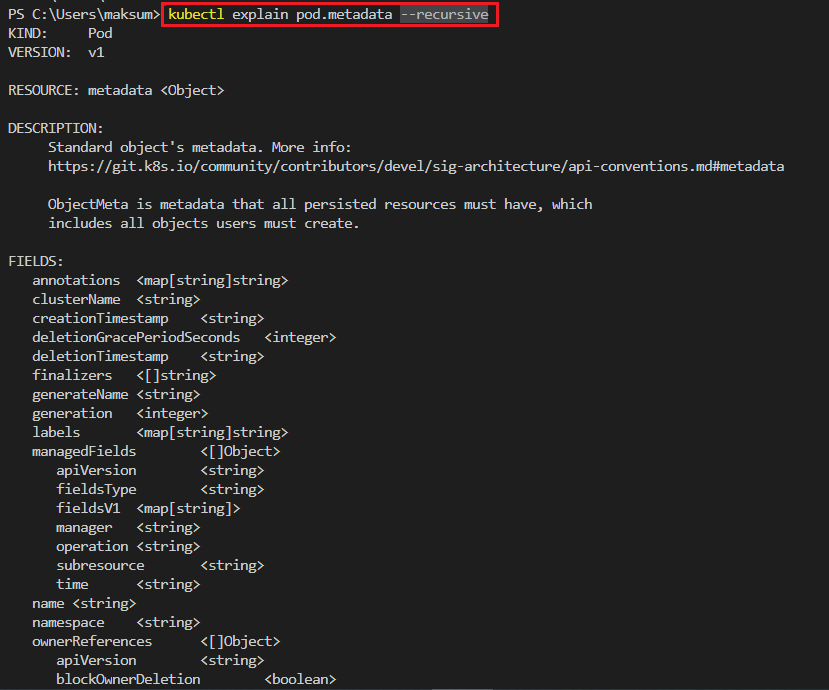
komutunu kullanacağız. Böylece o nesne altındaki parametler açıklamasıyla listelenecektir.



Peki ben sadece nesne altında ki parametreleri soya ağacı şeklinde listelemek istersem ne yapacağım ? :)

Bunun için komut sonuna **--recursive**parametresini ekleyeceğiz.

***kubectl explain pod.metadata*--recursive**



Böylece sadece nesne altındaki parametreler listelemiş olacak. Örneğimizde pod.metadata altındaki parametreler listelenmiştir.

Dilerseniz deployment, service,statefulset gibi kubernetes üzerinde var olan tüm nesnelerin detaylarına explain komutu ile bakabilirsiniz.  Lazım olur :)

* Pod üzerinde Port Yönlendirme İşlemleri:

8080(host):80(container)

Kubectl port-forward k8s-web 8080:80

Kubectl port-forward pods/mariadb 8080:80 -> nesne ismi verdik

Kubectl port-forward deployment/mongo 28015:27017 -> farklı neslerde de yönledirme yapılabilir.

Kubectl port-forward replicaset/mongo-785 :80 -> uygun olan port sistem tarafında atanacaktır.

Kubectl port-forward --adress 0.0.0.0 pods/k8s-web 8080:80 ->pc deki tüm ip lerde pod içerisinde çalışan uygulama erişimek istersek veririz

**-output=custom-columns= Komut Detayları**

**Nedir:**

kubectl get --output parametrelerinden biri olan **custom-columns**opsiyonu, listeme işleminde özel sütunlar tanımlamamıza ve yalnızca istediğimiz ayrıntıları içeren tablo oluşturmamıza  izin vermektedir.

Ve akabinde bu tabloyu çıktı alıp kullanabiliriz.

**Kullanım:**

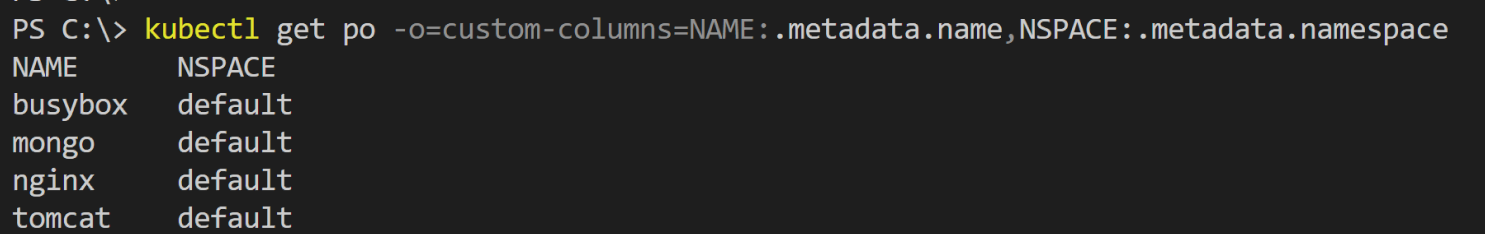
$ kubectl get pods -o=custom-columns=

ifadesinden sonra JSON dosyasında path bilgisine göre hangi alanları eklemek istiyorsak onları sıralayacağız. İlk önce sutün ismini sonrasında iki nokta koyup JSON içerisinde ki path bilgisini yazacağız. Başlıkları virgül ile birbirinden ayırıyoruz.

Örnek:  
kubectl get pods -o=custom-columns=NAME:.metadata.name,NSPACE:.metadata.namespace

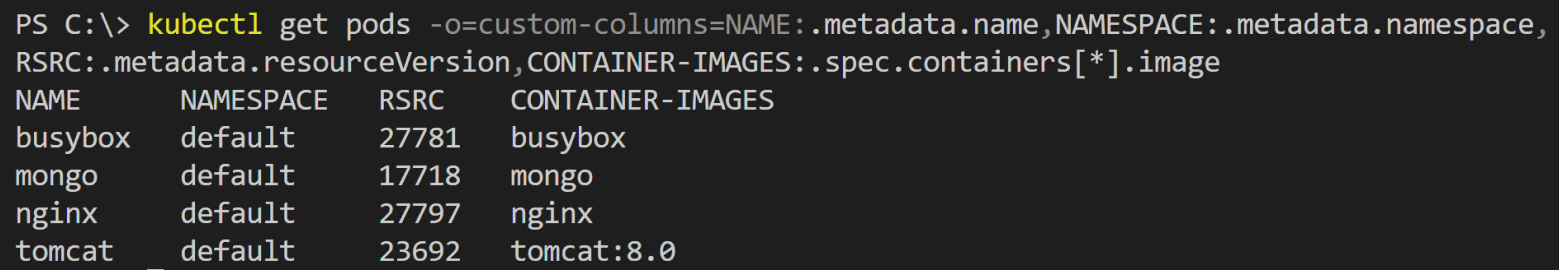
Örneği incelediğimizde tablonun birinci sütunü .metadata.name bilgilerini içerecek ve başlığı NAME şeklinde olacak. İkinci sutünda .metadata.namespace bilgileri olacak ve başlığı NSPACE şeklinde olacaktır.

Aşağıda örnek çıktısı gözükmektedir. Sütun başlığı verdiğimiz tanım olmuştur.



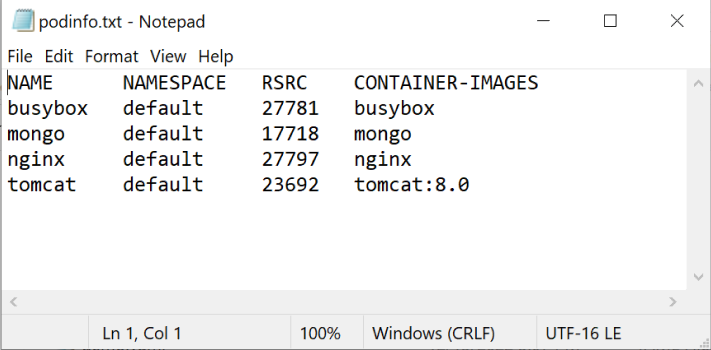
Yine dilenirsek farklı değerler eklenerek tabloyu uzatabiliriz.

kubectl get pods -o=custom-columns=NAME:.metadata.name,NAMESPACE:.metadata.namespace,RSRC:.metadata.resourceVersion,CONTAINER-IMAGES:.spec.containers[\*].image



Komut sonuna >> ifadesi eklenerek dosyaya çıktıyı kayıt edebiliriz.

kubectl get pods -o=custom-columns=NAME:.metadata.name,NAMESPACE:.metadata.namespace,RSRC:.metadata.resourceVersion,CONTAINER-IMAGES:.spec.containers[\*].image >> podinfo.txt



--no-headers parametresini kullanarak tüm başlıklar  temizleyebiilr ve sadece çıktı listelemesi yapabiliriz.

Uygulama :

**Mariadb imageını kullanan ismi mymariadb olan bir pod oluşturun. Root password bilgisini ekleyin**

kubectl run mymariadb --image=mariadb --restart=Never --env=MYSQL\_ROOT\_PASSWORD='Passw0rd!'

**mariadb'nin ayakta olup olmadığını kontrol edin**

kubectl get po mymariadb

**mariadb pod event'larını kontrol edin ve pod oluşturma adımlarını inceleyin**

kubectl describe pod mymariadb

**mariadb podun loglarını görüntüleyin**

kubectl logs mymariadb

**mymariadb pod içerisinde "udemy" adında yeni bir database oluşturun ve tüm databaseleri listeleyin**

#create database udemy;

#show databases;

**mymariadb poduna giriş yapmadan database path'ini listeleyin(/var/lib/mysql/)**

kubectl exec mymariadb -- ls /var/lib/mysql/

**mymariadb'ye uygulama üzerinden erişmek için port yönlendirme komutunu çalıştırın**

kubectl port-forward mymariadb 8080:80

**Busybox image'ını kullanan ve ekrana "How are you" yazan yeni bir pod oluşturun(pod içinde echo parametresini kullanın).**

kubectl run busybox-1 --image=busybox --restart=Never -it --rm -- echo "How are you"

**Busybox image'ını kullanan ve "sleep 3600" komutunu çalıştıran yeni bir pod oluşturun(araştır - pod içerisinde komut çalıştırırken -c opsiyonunu kullanın)**

kubectl run busybox-2 --image=busybox --restart=Never -- /bin/sh -c "sleep 3600"

**Nginx image'ını kullanan yeni bir pod oluşturun. Env bilgisi olarak udemy**=**kubernetes bilgisini ekleyin**

kubectl run mynginx --image=nginx --env=udemy=kubernetes

**Pod içerisine girmeden env bilgisini listeleyin - eklediğimiz env bilgisinin geldiğini  görüntüleyin(araştır - POD env bilgileri env parametresiyle görüntülenir)**

kubectl exec mynginx -- env

**Aşağıdaki özelliklerde ki Pod'u mybox.yaml dosyasına kaydedin. (pod içerisinde çalıştırılacak komut için -c opsiyonunu kullanın)**

1. isim: busybox
2. Image: busybox
3. Env: var1=value1
4. Baslatma: Asla
5. Çıktı: YAML
6. Pod içinde Çalıştırılacak Komut: "touch password"
7. YAML isim: mybox.yaml

kubectl run busybox --image=busybox --env=var1=value1 --restart=Never --dry-run=client -o yaml -- /bin/sh -c "touch password" >> mybox.yaml

Pod Label ve Selector

Kubernetes üzerinde oluşturduğumuz container yığınlarını label ve selector ile ayırt etmekteyiz.

Örneğin podlar;backend,frontend vs etiketler vereceğiz. Yada selector içerisinde bu etiketleri kullanarak kullanıcıların bu podlara dengeli bir şekilde dağımalarını sağlayacağız.

* Kubectl run myapp --image=nginx --labels=”env=frontend”
* Kubectl run myapp --image=nginx --labels=”env=frontend,tier=prod”
* Kubectl label pods myapp “env=demo,tier=backend”-> sonradan label atadık.
* Kubectl node “disktype=ssd”
* Kubectl get pods --show-labels

**Selector**

**Equality-based**

= == !=

Env=production

Tier !=frontend

**Set-Based**

İn notin exists

Env in(production)(env=production)

Tier notin(frontend,backend)

* Kubectl get pod --selector=”env=production,tier=frontend”
* Kubectl get pods --selector=”tier!=production”
* Kubectl get pods –l “env in(production)”
* Kubectl get pods –l “tier notin(frontend)”
* Kubectl get po –k8s-label-4 env=demo --overwrite -> aslında env=prod diye bir label var ancak --overwrite ile biz yinede yazmasını söyledik
* Kubectl get label po –all status=healty-> tüm podlar status=healty etiketini ekledik
* Kubectl get pods --selector=”env=prod” -> env=prod olan lebalları getirdi
* --selector = -l
* Kubectl delete po –l “env in (prod)” yada env=prod-> env=prod olan labelları sildi.

#### ****alias (takma ad) Kullanımın Önemi:****

Özel kullanımda yada CKA sertifika sınavında kısaltma kullanarak iş yapmak oldukça önemlidir. Bildiğiniz gibi CKA Sertifika sınavında sizlere 2 saatlik bir zaman dilimi verilir ve verilen LAB örneklemesini yapmanız beklenir. Her komutu uzun uzun yazıp zaman kaybetmek yerine başlangıçta en çok kullanılacak komutları takma isim vererek kısaltıp kullanabilirsiniz. Böylece komutu her seferinde uzun uzun yazıp zaman kaybetmek yerine hızlıca çıktıya erişebilirsiniz.

**Örnek Kullanım:**

alias k='kubectl'

alias kg='kubectl get'

alias kgpo='kubectl get pod'

alias kdp='kubectl describe pod'

alias kd='kubectl delete'

alias kdf='kubectl delete -f'

alias ke='kubectl explain'

alias kgl='kubectl get pods --show-labels'

alias ks='kubectl get namespaces'

alias kga='kubectl get pod --all-namespaces'

alias kgla='kubectl get all --show-labels'

alias c='clear'

alias kdr='kubectl run --dry-run=client -o yaml'

alias kd="kubectl delete --grace-period=0 --force"

Yukarıda **Linux işletim sistemi için**örnek oluşturulmuş kısaltmalar yer almaktadır. Sizde kendinize göre bir standart belirleyerek bu kısaltmaları kullanabilirsiniz.

**Not:**CKA sertifika sınavı için Linux shell ve test editör (Vim) tecrübenizi geliştirin.

**Uygulama**

**Podlar üzerindeki label'ları listeleyin**

kubectl get pods --show-labels

**Tüm namespacelerde bulunan podları label'ları ile birlikte listeleyin**

kubectl get po --show-labels --all-namespaces

**nginx2 podunun env label'ını env=demo olarak etiketleyin**

kubectl label po nginx2 env=demo –overwrite

**env etiketine sahip pod'ları listeleyin**

kubectl get po --selector env

**env=demo olan podu listeleyin. Bunun için Equality ve Set based yazım şeklini kullanın**

#kubectl get po --selector env=demo #kubectl get po --selector 'env in (demo)'

**status=healthy ve env!=demo olan podları listeleyin. Bunun için Equality ve Set based yazım şeklini kullanın**

#kubectl get po -l "status=healthy,env!=demo" #kubectl get po -l "status in (healthy),env notin (demo)"

**Podlardaki status etiketini kaldırın**

kubectl label po nginx1 nginx2 nginx3 status- # veya kubectl label po nginx{1..3} status-

**env=production olan labela sahip podları silin**

kubectl delete pods -l env=production

**nginx2 podunu team=dev etiketini ekleyin ve label listelemesi yapın**

#kubectl label pods nginx2 team=dev #kubectl get pods --show-labels

**nginx2 pod detaylandırını görüntüleyin ve labels etiket bölümünü inceleyin**

kubectl describe po nginx2

**env=demo ve team=dev label'a sahip olan podu silin**

kubectl delete pods -l env=demo,team=dev

Formun Üstü

**"Node" üzerinde tanımlı label'ları görüntüleyin**

kubectl get nodes --show-labels

**node üzerine disktype=ssd etiketi ekleyin ve görüntüleyin**

#kubectl label nodes disktype=ssd #kubectl get nodes --show-labels

**node üzerine eklediğiniz disktype=ssd etiketi kaldırın**

kubectl label node disktype-

**Kubectl config Yapılandırması Formun Altı**

Kubectl config –h -> config dosyası yönetme komutları

Kubectl config view -> kube dosyası altındaki config dosyası içeriğini veriri

Kubectl get config get-contexts-> config dosyasına ekli bütün clusterları verir.

Kubectl config use-contex (cluster\_name)->clusterlar arası geçiş yaparız.

Kubectl config current-context-> varsayılan cluster bilgisi

Minikube start –p (cluster\_name)-> yeni bir cluster oluşturma

Minikube profile list -> cluster bilgisi (config current-context) ile aynı

Minikube delete –p (cluster\_name)-> cluster kaldırma

**Uygulama:**

**Development user için tanımlanan client-certificate file ismi nedir**

C:\users\emre\.minikube\profiles\development\client.crt

**Production user için tanımlanan client-key bilgisi nedir**

**Produciton\client.key**

**kubeconfig dosya path'ine ulaşin**

$HOME/.kube/config # Linux C:\Users\XXXXXXXX\.kube\config #Windows

**Kubeconfig dosya içeriğini komut satırında görüntüleyin**

kubectl config view

**Minikube üzerine research adında ikinci yeni bir cluster ekleyin(virtualbox kullanıyorsanız sanal makinanın oluşturulduğunu gözlemleyi**

minikube start -p research

**kubeconfig dosyasında tanımlı tüm clusterları görüntüleyin**

kubectl config get-clusters

**kubeconfig dosyasında tanımlı tüm contextleri görüntüleyin. Yıldızlı contexti bulun**

kubectl config get-contexts

**Demo ortamında kubeconfig'deki tüm userlerı görüntüleyin**

kubectl config get-users

**kubeconfig dosyasında tanımlı aktif contexti görüntüleyin**

kubectl config current-context

**kubeconfig dosyasında tanımlı**sadece **aktif context'i görüntüleyin(araştır)**

kubectl config view –minify

**Diğer cluster'ı aktif hale getirin(minikube) ve aktif context'i görüntüleyin**

 kubectl config use-context minikube $ kubectl config current-context

**research için cluster,context silin ve kubeconfig dosyasını görüntüleyin**

# kubectl config delete-cluster research # kubectl config delete-context research # kubectl config view

**Minikube üzerindeki tüm cluster profilleri silin**

# minikube delete –all

**Yeni bir cluster oluşturun**

minikube start

**2.BOLUM: Kubernetes (Declarative)~Yaml:**

Her yaml dosayalarında şu nesneler kesin olmalıdır:

Apiversion->Kubernetes Apı Sürümü tanımlar (Pod->v1 ,recplicationcontroller->v1 service->v1,replicaset->apps/v1,deployment->apps/v1,daemonset->apps/v1,job->apps/v1) kubectl explain komutu veya dokumanstasyon sayfasından görebiliriz  
kind-> Nesne türünü tanımlar(pod, repicaset,deplyment,job,deamonset,replicacontroller,service)  
metadata-> obje ile ilgili benzersi olan tanımala(isim,namespace,labels)

Spec -> oluşturulacak Nesneyle ilgli Durum detaylarını tanımlar

Kubernetes nesnesi üzerinde doğrudan bi arama yapacaksak –filed-selector komutunu kullanabilirz.(ancak oldukça kısıtlıdır.)

Ör:

Kubectl get pods --field-selector metadata.name=myApp

Kubectl get pods --field-selector metadata.namespace=Prdoduction

Kubectl get pods --field-selector metadata.namespace !.=Project  
 Kubectl get services –all-namespaces –field-selector metadata.namespace!=default

Yaml dosyasını çalıştırma:

Kubectl create –filename my-manifest-1.yaml(güncelle yapıcaksak hata verir )

Kubectl –filename my-manidest-1.yaml

Kubectl apply –f my-manifest-2.yaml

Kubectl apply –f ./my1.yaml -f ./my2.yaml -> birden fazla yaml dosyasını aynı anda çalıştırmak istersek

Kubectl apply –f <https://git.io/vPieo> -> uzaktaki web üzerinde bulunan yaml dosyasında çalıştırabiliriz.

Kubectl edit po nginx-pod-first -> çalışan pod üzerinde değişiklik yapmak istersek ve elimize yaml dosyası yoksa çalıştırarak txt üzerinde ekleme yada çıkarma yapabiliriz.(işlem sonrasında dosyayı kaydetip kapatmamız yeterli)(bu bilgi yaml dosyasına eklenmez)(eğer kalıcı bir değişiklik yapıcaksak yaml üzerinde değişiklik yapmalı ve çalıştırmalıyız)

Kubectl describe po/nginx-first-pod -> ile pod hakkında detaylı açıklamaya sahip olabiliriz.

#### --dry-run Komut Detayları (Hatırlatma)

**--dry-run**opsiyonu kubectl run komutunda yazdığınız ifadeyi canlı sistem üzerinde çalıştırmadan önce test etmemize imkan sağlamaktadır. **-o yaml**parametresiyle kullanıldığında YAML dosyasını elle yazmamıza gerek kalmadan sistem tarafından  yazdırılmasını sağlar.  CKA sertifika sınavlarında hayat kurtaran komuttur.

Üç opsiyonla birlikte gelmektedir. Bunlar **client**, **server**, **none**

✸ Bilgi Notu - 4 ' bölümünde bu konu açıklanmıştır.

Örneğin deployment nesnesi YAML dosyasını sistem tarafından oluşturmak istersek aşağıdaki komutu kullanacağız.

**#kubectl create deployment nginx --image=nginx --dry-run=client -o yaml**

Komut çıktısı aşağıdaki şekilde olacaktır

1. apiVersion: apps/v1
2. kind: Deployment
3. metadata:
4. creationTimestamp: null
5. labels:
6. app: nginx
7. name: nginx
8. spec:
9. replicas: 1
10. selector:
11. matchLabels:
12. app: nginx
13. strategy: {}
14. template:
15. metadata:
16. creationTimestamp: null
17. labels:
18. app: nginx
19. spec:
20. containers:
21. - image: nginx
22. name: nginx
23. resources: {}
24. status: {}

YAML dosyasının sistem tarafından otomatik olarak oluşturulmasını sağlamak istersek  "--dry-run=client"  parametresini kullanabiliriz. Akabinde > opsiyonuyla belirlediğimiz dizine kaydedebiliriz.

**# kubectl create deployment --image=nginx nginx --dry-run=client -o yaml > nginx-deployment.yaml**

#### kubectl explain (Hatırlatma)

Nesne detayları hakkında bilgi sahibi olmak ve incelediğimiz nesnenin alt parametlerini listelemek istendiğinde kullanılmaktadır. Aynı zamanda bu komut ile incelediğimiz nesnenin kubernetes dökümantasyon sayfasındaki linkine de ulaşabiliyoruz.

Nasıl Kullanılır

kubectl explain komutundan sonra detaylarına bakmak istediğimiz nesne ismini yazıyoruz

**kubectl explain pod**

#### kubectl edit (Hatırlatma)

Kubernetes üzerinde çalışan nesnelerin ayarlarını hızlıca değiştirmek için kullanılmaktadır.

**Uygulama:**

**firstyaml podunun durumunu ve ip bilgisini öğrenin**

kubectl get po -o wide

**firstyaml podu içerisinde çalışan container ismini maincontainer olarak değiştirin**

kubectl edit po firstyaml

**Çalışan firstyaml pod'u, yaml formatında secondyaml.yaml olarak kaydedin ve dosyayı inceleyin**

kubectl get po firstyaml -o yaml >> secondyaml.yaml

**firstyaml.yaml dosyasını imperative yöntemle oluşturun, dosyayı thirdyaml.yaml olarak kaydedin ve inceleyin**

kubectl run firstyaml --image=httpd --dry-run=client -o yaml > thirdyaml.yaml

**firstyaml poduna 8080 portu üzerinden port yönlendirmesi yapın ve browser üzerinde sayfanın çalıştığını kontrol edin**

kubectl port-forward firstyaml 8080:80

**firstyaml nesnesini yaml dosyası üzerinden silin**

kubectl delete -f firstyaml.y.aml

Yaml File ile Merhaba Dünya:

Command : container içimde çalıştırılacak komutu (docker ismi -> entrypoint)

Args : Komut içerisinde eklenecek bağımsız değişkenleri tanımlamaktayız.(docker ismi -> cmd)

* Konteyner içinde command ve args komutları kullanılmazsa , Docker image içerisinde belirlenen default değereler kullanılacaktır.
* Konteyner içerisinde command değeri tanımlanır ve args değeri boş geçilirse, docker image içerisinde ki entrypoint ve varsayılan cmd değeri yok sayılır.
* Konteyner içerisinde yalnızca args değeri tanımlanırsa,docker image içerisindeki ki enrtypoint değeri tanımladığımız args değeri ile birleştirilerek çalıştırılır.
* Konteyner içerisinde command veargs tanımlanırsa docker image içerisinde ki değerler yok sayılır. Tanımladığımız command args değeriyle birlikte çalıştırılır.

**Pod Yaşam Döngüsü:**

RestartPolicy->Pod’un olası bi hatada ne yapacağını belirlemekteyiz(

Always-> conatiner içerisinde çalışan uygulama her hangi bir sebepden doşayı kapanırsa o container yeniden başlatır varsayılan değerdir.

On-failure->Sadece hata alındığında container yeniden başlatılmaya zorlanır.

Never->contaier ne durumda kapanırsa kapansın hiçbir zaman yeniden başaltılmayacaktır.

)

**1. Pending**

Pod, Kubernetes Scheduler tarafından bir Node’a yerleştirilmeyi bekliyor. Bu durumda Pod henüz çalışmaya başlamamıştır çünkü gerekli kaynaklar sağlanmamış veya eksik.

**2. ContainerCreating**

Pod’un container’ları oluşturuluyor. Bu aşamada Docker image’ı çekiliyor, volume’ler bağlanıyor ve network ayarları yapılıyor.

**3. ImagePullBackOff**

Pod, belirtilen Docker image’ı çekmeye çalışıyor ancak bu işlem başarısız oluyor. Kubernetes, image’ı yeniden çekmek için bekliyor.

**4. Running**

Pod ve içindeki container’lar başarıyla çalışmaya başlamış durumda. Bu durumda Pod, talepleri karşılamaya hazırdır.

**5. Succeeded**

Pod yaşam döngüsünü başarıyla tamamlamış. Genellikle tek seferlik işlemler (Jobs veya tek seferlik container’lar) bu duruma geçer.

**6. Failed**

Pod, yaşam döngüsünü tamamlayamamış ve başarısız olmuş. Container, bir hata koduyla sonlanmış olabilir.

**7. CrashLoopBackOff**

Pod, çalışmaya başlıyor ancak sürekli çöküyor. Kubernetes, container’ı tekrar tekrar başlatmayı deniyor ancak başarılı olamıyor.

**Çoklu Konteyner Kullanımı :**

* Multi-container kullanımı aynı pod içerisinde birden fazla konteyner oluşturmamıza imkan vermektedir.
* Pod içerisinde ki her iki konteyner tek bir konteynermiş gibi hareket eder,oluşturulunca brlikte oluşuturulur, silinince birlikte silinir.
* Her iki konteyner birbiriyle localhost seviyesinde iletişim geçmektedir.
* Tek bir volume oluşturup her iki konteyner üzerine bağlanabilir ve kullanılabilir.

**Not: pod içerisindeki birden fazla pod olduğunda istediğimiz konteyner’a bağlanma ;  
kubectl exec –it multi-container-pod --container=(container\_name) --/bin/bash**

**(-c)**

**Uygulama:**

**multipod pod içerisindeki container2 içerisine girmeden dizin listelemesi yapın. readmeBLUE dosyasının olmadığı kontrol edin. Akabinde loglarını görüntüleyin**

##kubectl exec -it multipod -c container2 -- ls ##kubectl logs multipod -c container2

**multipod pod içerisindeki container3 içerisine girmeden dizin listelemesi yapın. container3 loglarını görüntüleyin**

##kubectl exec -it multipod -c container3 -- ls ##kubectl logs multipod -c container3

**multipod için 8080:80 doğru port yönlendirmesi yapın. Browser üzerinde 8080 portuna erişin ve nginx ana sayfasının geldiğini kontrol edin**

kubectl port-forward multipod 8080:80

**multipod pod içerisindeki container1 içerisine girmeden dizin listelemesi yapın. Çıktıdaki readmeBLUE file'ını cat komutuyla görüntüleyin**

##kubectl exec -it multipod -c container1 -- ls ##kubectl exec -it multipod -c container1 -- cat readmeBLUE

**Init Container :**

Bir uygulamamız var bu uygulamamız farklı servislere bağımlı (pod içerisinde çalışan uygulamamız uzaktaki bir servise ihtiyaç duyuyor )

Uzaktaki uygulama ayağa kalkmadan pod ayağa kalkarsa hatalar veriyor olucak bu yüzden ilk önce bağımlı servislerin ayağa kalkması gerekiyor işte bu kontrolü init container aracaılığı ile sorunsuz yapabilmekteyiz. Pod içerisinde oluşturuluna init containerlar sayesinde öncelikli olarak talep edilen servisin ayağa kalkıp kalkmadığı kontrol edilir. Servis ayağa kalkana kadar ana uyguluma bekler.böylece ana uygulama hata vermeden çalışacaktır.

Bazır durumlarda ise anauygulama bazı config dosyların en güncel halini alarak ayağa kalkması gerekir bu durumda pod içerisinde ekleyeceğimiz init containerlar sayesinde güncel config dosyasını sisteme çektikten sonra ana uygulamamız devreye girecek en güncel haliyle çalışmaya başlayacaktır. Bu mimaride ilk init containerlar ayağa kalkar işlemini yapar sonra ana uygulama devreye girer.

**Resource Requirement & Limit:**

Cpu :”1” = cpu:”1000” = cpu:”1000m” -> 1cpu core kullan

Cpu :”0.5” = cpu:”500” = cpu:”500m”-> 1cpu core’un %50’ni kullan

Memory : Bir pod üzerine talep edilen kadar memory ataması yapılır ancak limitden daha fazla bellek kullanılmasına izin verilmez.Pod üzerine verilen limit aşılırsa Pod restart edilir. (mb,gb,)(kibi,mebi,gibi,tebi,pedi,expi)

Request-> node üzerinde olması gereken değeri (olaması gereken min değer)

Limit -> pod’un kullancağı üst sınır.

Kubectl top (pod –node )-> hangi nesnenin donandım kaynak kullanımın görmek (edone yüklememiz gerekiyor )(metrics-server)

Minikube adonns list -> minikube üstünde edoneleri listemele

Minikube addons enable metrics-server -> metrics-server aktifleştirdik

Kubectl top Pod -A -> tüm namespaclerde çalışan podları gösterir.

**Namespace Kullanımı :**

* Kubernetes içerisindeki podları veya objeleri ayırmamıza /gruplandırmamıza imkan sağlayan teknolojidir.
* Bir namespace içerisinde oluşturuluan nesnelere diğer namespace gruplarında gözükmeyecektir.
* Namespaci kubernetesteki odacıklar olarak görebiliriz.
* Kubernetes içerisinde varsayılan olarak gelen namespace default namespacedir.
* Kubernetes ilk kurulduğunda 4 namespace dahili olarak gelmektedir
* Default
* Kube-system
* Kube-public
* Kube-node-lease

Kubectl create namespace (isim)-> yeni name space oluşturma

Kubectl get pod --namespace=(isim)->belirli namespace altıdaki podları görüntelme.

Kubectl create –f proje.yaml --namespace=(isim) ->declaratice yöntem ile oluşturulan bir yaml dosyasını özel bir namespace’de oluşturabiliriz.

Kubectl exec –it proje –namespace=dev --/sh/bash -> namespace bağlanma

Kubectl delete ns (isim) ->namespace silme

Kubectl get po –n (isim) -> belirtilen namespace altındaki podları listeler.

Kubectl exec –it pod-namespace --namespace Project -- bash -> belirtilen namespace içindeki pod ‘a bağlanma

Kubectl port-forward pod-namespace -n Project 8080:80 -> Project namespace altındaki pod-namespace podu 80 portuna yönlendirdik.

**Label – Selector :**

Label, Kubernestes üzerine her türlü nesneye verebildiğimiz ve bir nesneyi diğer nesnelerden ayırmamıza sağlayan anahtar –değer eşleniğidir.

Etiketler sayesinde nesnelere bizlerin anlayabileceği ve gruplama yapmamıza imkan sağlayan bilgileri eklemekteyiz.

Metadata altında labels olarak oluşturacağız.

Test.com/tier:frontend

“opsiyonel” anahatar değer

Etiketler nesne oluştururluken yada çalışırken eklenebilir

Tes.com kısmı zorunlu değildir.

Kubernestes.io veya k8s.io önekleri kubernestes için ayrılımıştır.

Anahtar ve değer alanları 63 karakter veya daha az olmalıdır

Alfanumerik karakterla başlanmalı ve bitmelidir.

Kubectl delete pods –l “env=production,tier=frontend” -> label sildik

Kubectl get pod –show-labels -> labelları görüntüleme

Uygulama:

**tomcat-labelpod-2 üzerinde label değişikliği yapın(app.kubernetes.io/version="9.0") :** kubectl label pod tomcat-labelpod-2 app.kubernetes.io/version="9.0" --overwrite

**Sistem üzerindeki ("app.kubernetes.io/version: "8.0") etiketine sahip podları IP'leriyle birlikte görüntüleyin(Equality-based)**

kubectl get po -l app.kubernetes.io/version="8.0" -o wide

**Sistem üzerinde etiketi (app.kubernetes.io/name=tomcat) olan ve versionu 8.0'e eşit olMAyan podları** **IP'leriyle birlikte** **listeleyin(Set-based)**

ubectl get pods -l 'app.kubernetes.io/name in (tomcat),app.kubernetes.io/version notin (8.0)' -o wide

**Cluster üzerindeki nodeları labellarıyla listeleyin(Eklemiş olduğunuz etiketi görün)**

kubectl get node --show-labels

**Node üzerine eklediğiniz (kubernetes.io/say=hello) etiketini kaldırın ve kontrol edin**

ubectl label node minikube kubernetes.io/say- ##kubectl get node --show-labels

**Pod Annotation :**

Kubernetes nesneleri üzerinde ek açıklamlar yapmak için kullanılır. Burada girilen değerler kubernetes ‘in umrunda değildir.

K8s üzerinde projenizi okuyacak kişilere ek bilgiler vermek isterseniz örneğin proje sahibi ,mail adresi veya kaynak bilgileri gibi tanımlamalar annotation karamı ile yapılmaktadır.

Metadata altına ekleyeceğiz.

İmparitve yöntemlede yapılabilir;

kubectl annotate pod pod-annotation description=”version-12”

güncelleme yapılabilir:

kubectl annotate pod pod-annotation description=”version-13” --overwrite

**Environment Variable:**

K8s üzerinde pod içerisinde ortam değişkenleri tanımlamak için kullanılmaktadır.

Örneğin : db erişim bilgileri imaja gömmek çok kullanışlı ve güvenli değildir bunları imaja gömmeden k8s üzerinde dışarıdan okunabilcek şekilde tanımlayarak kullanırız.

\*\*Containers altında tanımlarız ‘’env” olarak adlandırırız.

-env :

-name:DBHOST DBHOST=”dbhost.local”

Value=”dbhost.local”

production ortamlarında bu şekilde kullanılmaz Secrets ve configmaps şekilde kullanılır. yani değer oluşturulur ama value kısmı bir dosyadan okutulur.

Not : Kubernetes'te **printenv**, bir container’ın **ortam değişkenlerini (environment variables)** görüntülemek için kullanılır.

Uygulama :

**annotations bölümünde ki kbarticle bilgisini kaldırın ve detayları görüntüleyin.**

##kubectl annotate pod mysql kbarticle- ##kubectl describe pod mysql

**mysql-container içerisine girmeden printenv komutunu çalıştırın(verdiğiniz env bilgilerinin eklendiğini kontrol edin)**

kubectl exec mysql -c mysql-container – printenv

**phpmyadmin içerisine girmeden printenv komutunu çalıştırın(verdiğiniz env bilgilerinin eklendiğini kontrol edin)**

kubectl exec mysql -c phpmyadmin – printenv

**db=mysql etiketine sahip pod'u listeleyin**

kubectl get pods -l db=mysql

**Mysql pod üzerine app=phpmyadmin label'ını ekleyin**

kubectl label pods mysql app=phpmyadmin

**Default Namespace 'nde ki pod'ları labelları ile birlikte listeleyin**

kubectl get pods --show-labels

**YAML dosyası üzerinden pod'u silin**

kubectl delete -f enviroment-annotation.yaml

**Kubernetes Scheduling :**

Belirtilen pod’un hangi node üzerinde oluşacağını belirtir .  
bunu yaparken;  
 filters(filtreleme)  
 score(puanlama)

Kullanır.

Scheduler node seçiminde şıu 3 methodu kullanılır;

1. Node Selector
2. Affinity
3. Taint ve Toleration

Node Selector( en bait yoldur) bunun için ilk label ataması yapılır. Yaml dosyasına noda üzerinde aranacak label bilgisi eklenecek. Bunu için spec altında nodeSelector altında hangi node üzerinde oluşturlmasını istiyorsak ona ait olan label bilgisini ekleyeceğiz

**Affinity Kullanımı(Node Affinity):**

Belirtilen pod’un hangi node üzerinde oluşacağını belirtir. Mantıksal ifadeler yazmak istediğimizde yani daha karmaşık işlemlerde kullanılır. Nodeselector’den daha fazla opsiyon sunar.

2tür affinity modeli bulunur;

1. Node Affinity
2. POD Affinity

İlk önce node affinity bakacağız   
önce worker node ‘lara label ataması yapılır.

Aynı nodeselector deki gibi pod oluşturulurken label etiketi sorgulanarak seçim işlemi yapılır.

Node affinity üzerinde iki tip seçim türü vardır ;

1. hard type (talep edlen label node üzerinde kesin olmalıdır.) yoksa pending’e uygun node bulana kadar bekletilecektir.

* Required (label kesin olmalı)
* duringScheduling(pod un ilkez oluşturulduğu durumu ifade eder. label kesin olması gerektiğini şart koşar)
* ignoredDuringExecution (label kaldırılırsa elleme kalsın)
* requiredDuringExecution (label kaldırılırsa çalışan podları tahliye et)

1. soft type (talep edilen label’a sahio node varsa öncelikli olarak bu nodelar üzerinde pod oluşturulmaya çalışılır. Yoksa uygun olan diğer nodelar üzerinde pod oluşturulur.)

* predderred(talep edilen lable node üzerinde barsa oluşturulur yoksa uygun olan herhangi birinde oluşturulur.)
* duringScheduling
* ignoredDuringExecution (label kaldırılırsa elleme kalsın)

Spec altında conatiners altında affinity nodeAffinity adında bir başlık açacağı

NodeSelectorTerms:

matchExpressions:

key :

operator: (in,notin,exists,doesnotexist)

value: (labelları buraya yazıyoruz)

**Affinity Kullanımı(Pod Affinity):**

Birbrine bağımlı olan podları aynı node üzerinde yada zone’da nasıl çalıştırabliriz bunu göreceğiz.  
  
ör: Farklı lokasyonlarda çalışan bir k8s clusteri olduğunu düşünelim bu yapı üzerinde db,app ve cache mekanisması olan bir uygulama çalıştırma istiyoruz yaml dosyasını çalıştırdık ve Scheduler servisi uygun olan nodeları belirledi   
db ankara app ve cache istanbulda ama farklı nodler üzeinde oluşturdu app pod db birleşem yapmak istediğinde ankaraya gidiecek ve işlemini yapıcak iki farklı lokasyonda bulunan bu podlar network üzerinden haberlşecek ama buda maliyet gideri olarak karşımıza çıkacaktır.

Node affinity özelliğini kullanarak bu iki podu aynı node üzerinde oluşturabiliriz .Ancak karmaşık sistemlerin olduğu sistemde bunun takibini yapmak zor olucaktır . bunu yerini app pod podunu oluşturmadan önce db hangi lokayonda ve podda olduğunu kontrol et ve ardından ona gore aynı node üzerinde yada lokasayonda oluştur dersek problem ortadan kalkmış olur.

Herhangi bir server üzerinde k8s servisini aktif hale getirdiğimizde node üzerine otomatik olarak atanan bazı labellarbulunmakta;  
beta.kubernetes.io/arch=amd64

Beta.kubernetes.io/hostname=minikube

Beta.kubernetes.io/os=Linux

Bunlara ek olarak cloud üzerinde k8s ‘in hangi availibility zone de ve region da olduğunu belirten label bilgisi sistem tarafından otomatik olarak eklenmektedir.

Topology.kubernetes.io/zone

Topology.kubernetes.io/region

Bu tanımlara bakılarak node’un hangi bölgede veya hostname olduğu kontrol edilecek ve işlem yapılacktır.

Nodeaffinity ile aynı ;  
spec altında affinity altında podAffinity şeklinde ve yine soft yada hard type türlerinden birini yazacağız.  
  
Not : label bilgisini nerde sorgulancağını topologykey kısmında belirteceğiz ;

Eğer kubernetes.io/zone dersek aynı zone üzerinde bu label’a sahip bir pod varsa bu nesneyi o node üzerinde oluştur

Eğer kubernetes.io/hostname dersek üzerinde bu label’a sahip bir pod varmı sorgulancak varsa bu nesneyi o node üzerinde oluştur

Eğer uygun bir ifade bulunamzsa required ise oluşturulmayacak predderred ise uygun olan pod üzerinde oluşturulacak

Dediklerimizin tam tersini söylemek istersek yani belitirlen key ve value eşit olan pod üzerinde “oluşturma” demek istersek PODANTIAFFINITY kullanacağız. Yani kısaca belirtilen label bilgisine sahip olan bir pod’un olmadığı bir zone tercih edilmesi gerektiği talep edilir.

**Uygulama :**

**Minikube üzerine iki tane daha yeni node ekleyin.Toplam 3 adet node olmasını sağlayın. Sorunsuz çalıştığınız teyit edin.(İki workshop için bu nodeları silmeyiniz)**

##minikube node add ##minikube node list

**minikube-m03 node üzerinde ki labelları listeleyin**

kubectl get node minikube-m03 --show-label

**minikube-m03 node üzerine disktype=ssd etiketini ekleyin ve görüntüleyin**

##kubectl label node minikube-m03 disktype=ssd ##kubectl get node minikube-m03 --show-labels

**minikube-m03 node üzerine eklediğimiz "disktype" etiketini kaldırın. Node etiketini kontrol edin**

##kubectl label nodes minikube-m03 disktype- ##kkubectl get nodes minikube-m03 --show-labels

**Pod durumunu ip bilgisiyle sürekli görüntüleyin**

kubectl get po -o wide –watch

**minikube-m02 node'un labellarını listeleyi**

kubectl get node minikube-m02 --show-labels

**minikube-m02 node üzerine disksize=large etiketini ekleyin**

kubectl label node minikube-m02 disksize=large

**Taint ve Toleration :**

Affinity kullanamadığımız durumlarda kullanırız örenğin 3 node yapılı bir cluster’ımız var bu nodelardan biri grafiksel özelliklere sahip ve burada özel podların çalışmasını isityoruz bu gibi durumlarda kullanılır.

Taint = özel olan node verilen ifade (ör: hardware=gpu)  
Toletarion= poda beliren ifadedir.(ör: hardware=gpu )  
Toletarion bilgisi olan podlar sadece Taint bilgisi olan podlar üzerinde oluşturulur.

Kubectl taint nodes [node name] [key=value]:TAINT\_EFFECT

Spec atına tolerations adında yeni bir bölüm açacağız containers ile eşit dizinde ancak sonra kullanılıyor genelde.

Tolerations:

* key :”dedicated”  
  operator : “equal”

value : “devp”

effect:”noSchedule

operator bölümünde equal ve exists yazabiliriz.  
equal -> eşit

Amaç: Taint’in anahtar-değer eşleşmesini kontrol eder.

Pod’un toleration’ında belirtilen değer, Node üzerindeki taint’in anahtar ve değerine tam olarak eşleşmek zorundadır.

exists -> mevcut

Amaç: Taint’in yalnızca anahtarının var olup olmadığını kontrol eder.

Pod, taint’in değerini umursamaz, sadece belirtilen anahtarın Node üzerinde var olup olmadığını kontrol eder.

Taint Effect kısımı 3 değer almaktadır.

NoSchedule : taint tanımıyla aynı olmayan yeni podlar node üzerinde oluşturulmasın. Node üzerinde çalışan podlara dokunulmaz

PreferNoSchedule: Taint tanımıyla uyuşmayan yani podlar başka seçenek kalmazsa yani node üzerinde pod oluşuturulamazsa bu node üzerinde oluşturulabilir. Node üzerinde çalışan podlara dokunulmaz

NoExecute:Dikkatli kullanılmalı. Taint bilgisi eklendikten sonra node üzerindeki podlar otomatik olarak Node üzerinden kaldırılacaktır.

Ve farklı bir node üzerinde oluşturulamaya abşalancaktır o yüzden dikkatli olunmalıdır.

Taint bilgisini çıkartmak için ise “–“ kullanırılır. -> kubectl taint nodes minikube dedicated-

**Kubernetes Cordon**

**Nedir ?**

Kubernetes cluster içerisinde bulunan node'ları pasif moda çekmek için kullanılmaktadır. cordon aktif edildiğinde node üzerindeki unschedulable seçeneği true olarak işaretlenir ve node artık yeni bir nesne kabul etmez.

**Kullanım:**

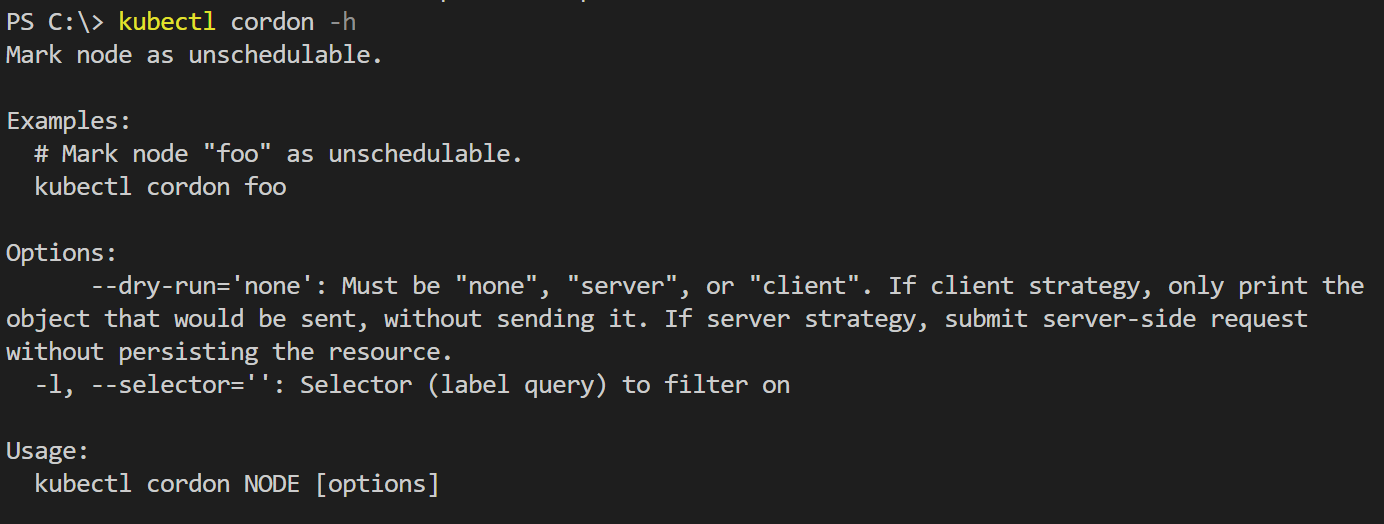
**kubectl cordon NODE**

komutuyla ilgili node unschedulable=true olarak işaretlenir ve node yeni gelen talepleri üzerine almaz.

**kubectl uncordon NODE**

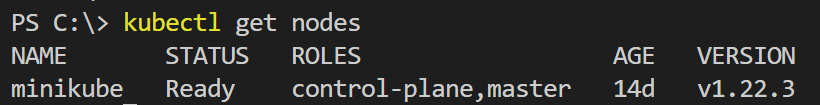
komutuyla ilgili node unschedulable=false olarak işaretlenir ve node yeni gelen talepleri kabul eder

--help diyerek detaylara ulaşabiliriz

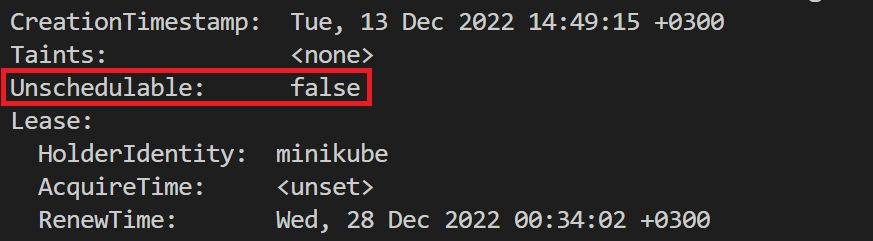


**Örnek:**

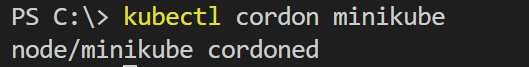
**1- Cordon özelliğini aktif olmadan önce node listemesi**



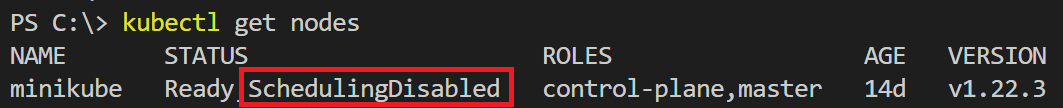
**2- Cordon özelliğini aktif olmadan önce node detaylarında unschedulable özelliği false olarak gözükür**



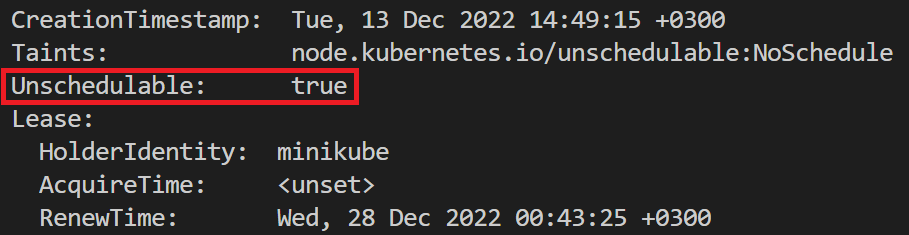
**3- Minikube node'u üzerinde cordon özelliğinin aktif hale getirilmesi.**



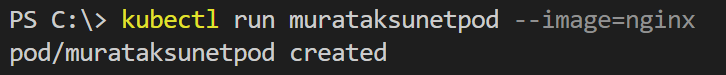
**4- Node detaylarına baktığımızda Minikube Node'unun SchedulingDisabled olarak işaretlendiğini görürüz**



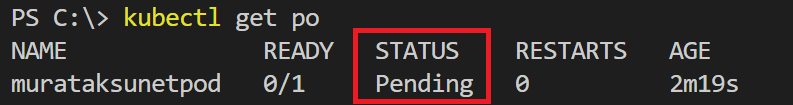
**5- Yine Node detaylarında unschedulable özelliği true olarak değiştirilir. Bunu gördüğümüzde Node üzerine POD kabul etmeyeceğini anlayabiliriz**



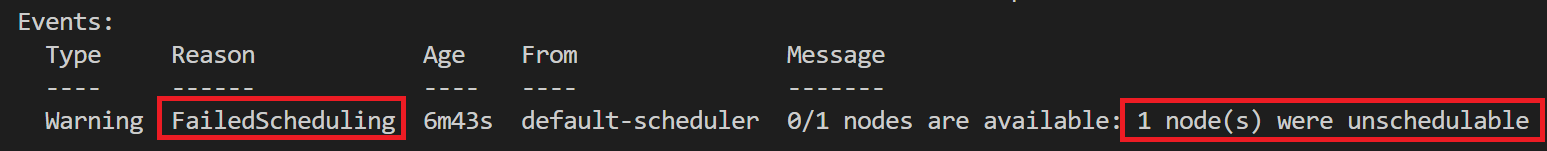
**5- Cordon özelliği aktifken yeni bir pod oluşturma**



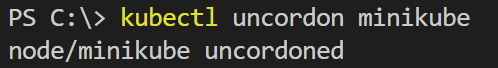
**6- Cluster içerisinde uygun bir node olmadığı için POD pending modda beklemede kalır**



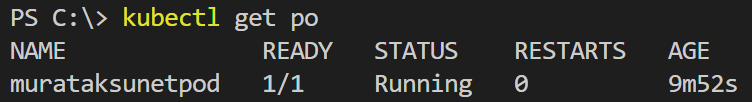
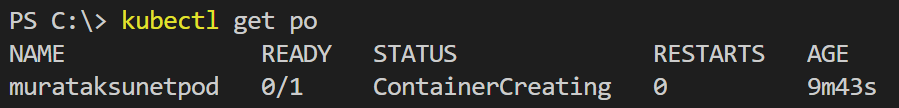
**7- POD detaylarına bakıldığında FailedScheduling olarak uyarı verdiğini görürüz**



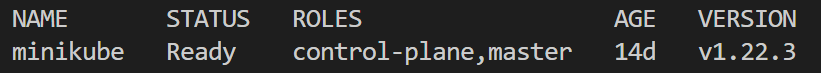
**8- Node'u eski haline almak için uncordon komutunu çalıştırırız**



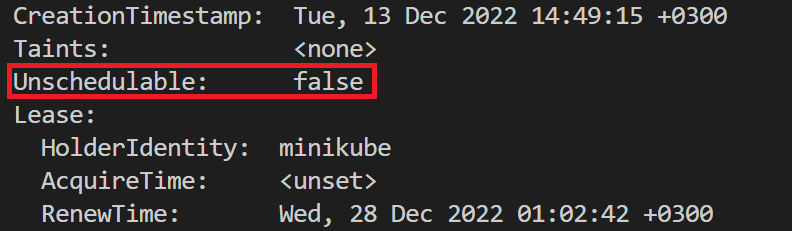
**9- POD listelemesi yaptığımzda POD'un oluşturulmaya başladığını görürüz**



**10- Node durumuna baktığımda SchedulingDisabled durumunun kalktığını göreceğiz**



**11- Yine Node detaylarında unschedulable özellğinin false olarak işaretlenecektir**



-->> Kısaca kubernetes cordon komutu cluster içerisinde ki Node'ları pasif moda çekmek için kullanılmaktadır

**Kubernetes Drain**

**Nedir ?**

Kubernetes cluster içerisinde bulunan node'ları maintinance yani bakım moduna almak için kullanılmaktadır. Örneğin kernel upgrade, hardware maintenance yada sunucu reboot gibi işlemlerden önce node üzerinde drain işlemi yapılarak cluster servisinden çıkarılır

Node üzeride Replicaset veya Deployment vb nesneler varsa bunlara ait POD'lar farklı node üzerine taşınacaktır. Tekil olarak çalışan POD'lara dokunulmaz.

drain aktif edildiğinde node üzerindeki unschedulable seçeneği true olarak işaretlenir ve node artık yeni bir nesne kabul etmez. Node'u eski haline almak için uncordon işlemi uygulanır, böylece unschedulable özelliği tekrar pasif yani false olarak işaretlenir.

**Kullanım:**

**kubectl drain NODE**

komutuyla ilgili node drain moduna alınmış olur fakat üzerinde çalışan POD lara dokunulmaz. Yeni gelen POD lar kabul edilmez. unschedulable özelliği true olarak işaretlenir.

**kubectl drain NODE --force**

komutuyla ilgili node drain moduna alınması için zorlanır. ReplicationController, Job ve DaemonSet nesneleri üzerinde de işlem yapılır.

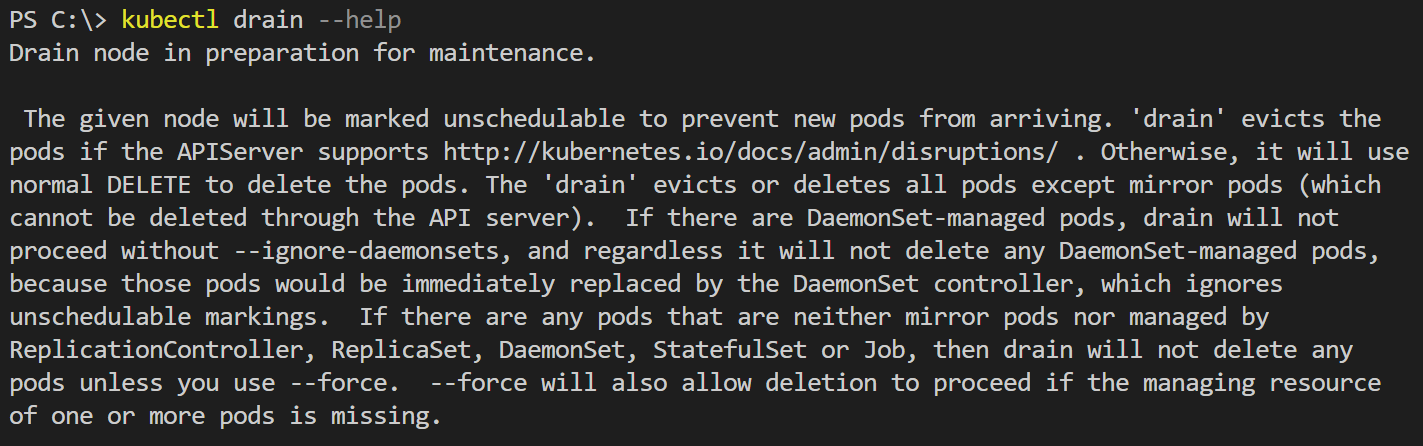
**kubectl drain NODE --force --ignore-daemonsets**

komutuyla ilgili node drain moduna alınması için zorlanır ve uyarılar yok sayılır. DaemonSet tarafından yönetilen POD'lar yok sayılır.

**kubectl drain foo --grace-period=900**

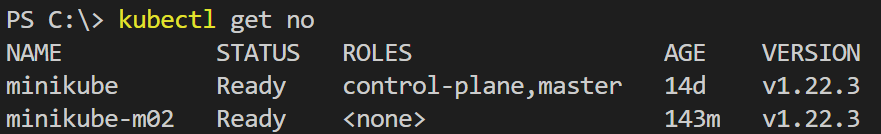
komutuyla drain işlemi belirli bir süre içinde tamamlanmaya çalışılır

--help diyerek detaylara ulaşabiliriz

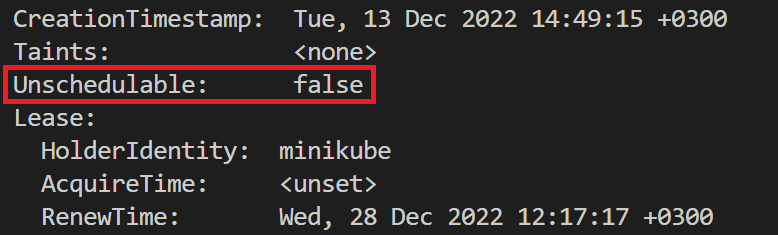


**Örnek:**

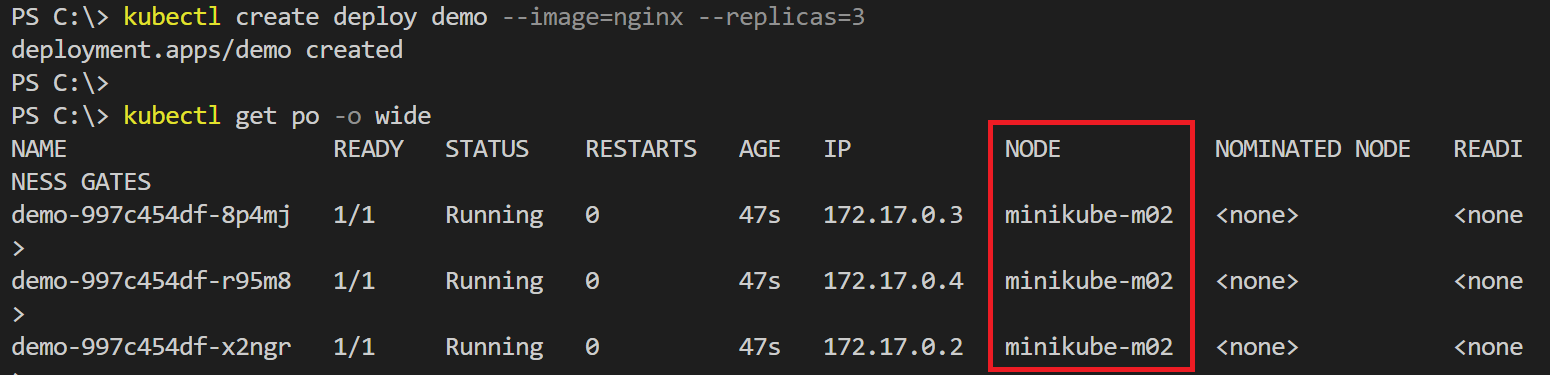
**1- Drain özelliğini çalıştırılamdan önce node durumu**



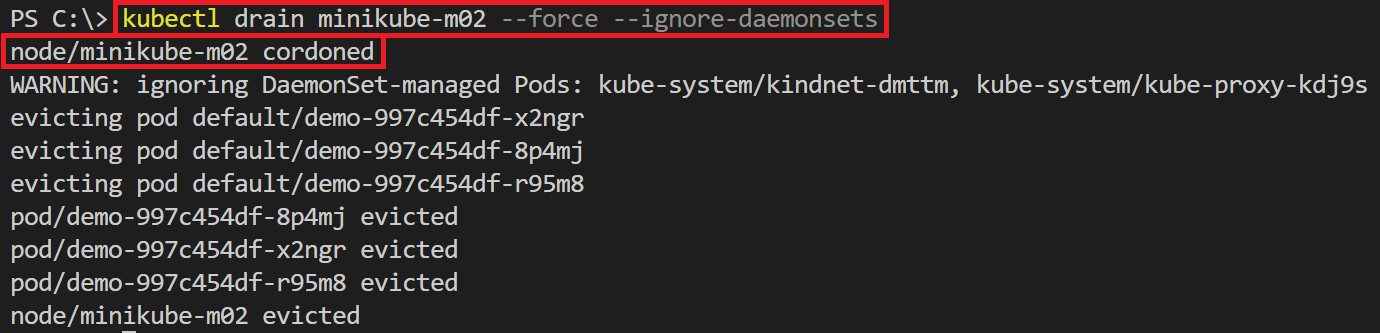
**2- Drain özelliğini aktif olmadan önce node detaylarında unschedulable özelliği false olarak gözükür**



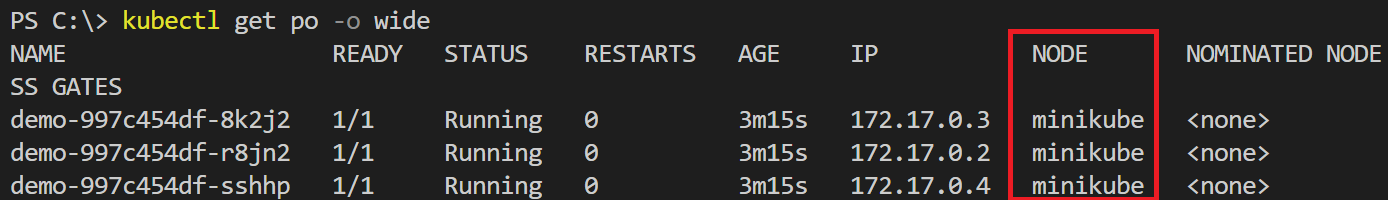
**3- Demo adında bir deployment nesnesi oluşturduk. Deployment nesnesi 3 adet POD 'u minikube-m02 node'u üzerinde oluşturdu**



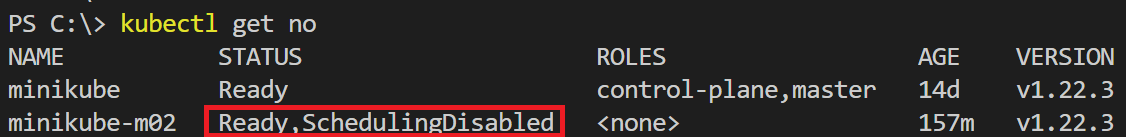
**4- minikube-m02 Node'u üzerinde drain komutu çalıştırıldıktan sonra POD'lar farklı bir node üzerinde kaydırılır ve NODE cordoned olarak işaretlenir yani yeni gelen POD'ları kabul etmez**



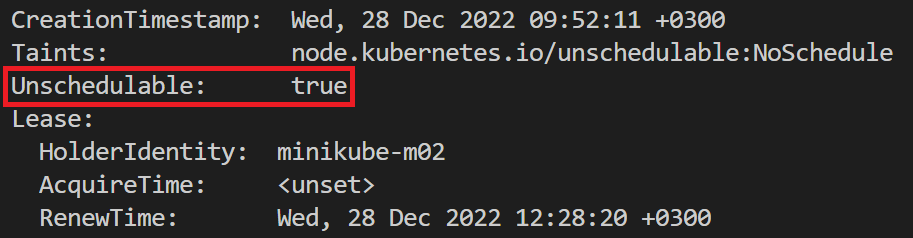
**5- minikube-m02 üzerinde çalışan POD'lar minikube üzerine taşınmış durumda**



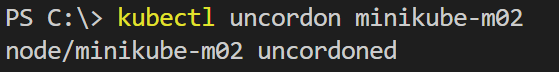
**6- Nodeları listelediğimizde minikube-m02  Node'u Status olarak SchedulingDisabled şeklinde işaretlendiği görülür. Buda yeni POD kabul etmeyeceği anlamına gelir.**



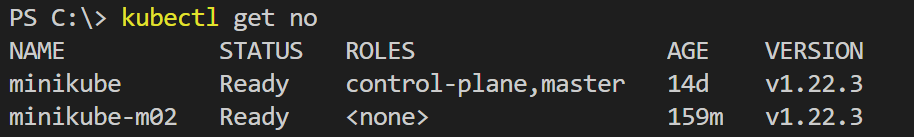
**7- minikube-m02  Node'un detaylarına baktığımızda unschedulable özelliğini true şeklinde değiştirildiğini görürüz**



**8- Node üzerinde işlemimiz tamamladıktan sonra tekrar cluster servisine dahil etmek istersek uncordon komutunu çalıştıracağız**



**9- Node üzerinde uncordon işlemini yaptıktan sonra Node yeni POD kabul edebilir duruma gelmiştir.**



**Probes Kullanımı :**

Probe’lar Pod’ların dağlık durumunu kontrol etmek için kullanılmaktadır. Probe periyodik olarak cluster ‘da yapılan diagnostik(tarama) operayonudur.

3 farklı Health Check Kontrol türü vardır ;

Readiness Probe: Uygulamanını hazır olup olmadığını k8s’e bildirmek için kullanılmaktadır. Kubernetes, ilgili Pod’a trafik göndermeden önce POD hazır durumda olup olmadığını kontrol eder. Eğer Pod’un hazır olmadığını görürse,cevap alana kadar trafik göndermeyi durdurur.

Livevess Probe:Uygulamanını sağlıklı çalışıp çalışmadığını kontrol etmek amacıyla kullanılmaktadır. Uygulama sorunsuz bir şekilde çalışıyorsa, kubernetes onu kendi haline bırakır.Uygulama cevap vermiyorsa ,Kubernetes Pod’u kaldrırı ve yerine yenisini oluşturur.

Startup Probe : ilk çalıştırılan probe’dur. Yavaş ayağa kalkan uygulamalar için kullanılmaktadır. Probe ayağaa kalkmadan pod’un k8s tarafından ortadan kaldırılması önleniş olur. Livevess probe ile birlikte kullanılabilir.

3 farklı Probe yönetmi vardır;  
ExecAction : pod içerisinde komut çalışıtmak için kullanılmaktadır. Böylece gerekli test işlemi yapılır. Konut çalıştırıldıktn sonra dönen değer 0 ise uygulama sağlıklı bir şekilde çalışıyor demektir, farklı bir değer dönerse uygulmanım crash olduğu düşünülerek gerekli restart işlemi yapılır.

TCPSocketAction: Yaml dosyasında belirtilen TCP port bilgisinin pod içerisinde ulaşılabilir olup olmadığını kontrol etmek için kullanılmaktadır.

HTTPGetAction: Yaml dosyasında belirtilen belirtilen hhtp adresine ve portuna sorunsuz bir şekilde erişip erişilmediğini kontrol etmek amacıyla kullanılmaktadır. 200 ile 400 arasında dönen değer başarılı olarak kabul edilir.

Belirtilen değerlerin dışında bir değer dönerse portun Sorunlu oldğunu düşünüp ve gerekli restart işlemi yapılır.

Bu yötemleri kullanarak uygulamayı kontrol etmekteyiz.

Probe Sonuçları

Success : konteyner test işlemi başarılı bir şekilde geçmiştir.

Failure : konteyner test işleminde hatayla karşılaşılmıştır.

Unknown : test başarısız olmuştur ama herhangi bir işlem yapılmaz .

Probe Yapılandırma :

Probe Yapılandırma (prob işlem süresini takibinin yapıldığı parametreler.)

initialDelaySeconds: Konteyner başlayıp Probe başlamadan önceki geçen süre (default: 0)

periodSeconds: Yoklama sıklığı için geçen süre (default: 10)

timeoutSeconds: Zaman aşımının sona ereceği süre (default: 1)

successThreshold: Konteynerin doğru çalışmasını belirleyeceği minimum başarılı deneme sayısı (default: 1)

failureThreshold: Yeniden başlatılacağı başarısız deneme sayısı (default: 3)

apiVersion: v1

kind: Pod

metadata:

name: liveness-command-exec

spec:

containers:

- name: liveness

image: nginx

ports:

- containerPort: 80

livenessProbe:

exec:

command:

- cat

- /usr/share/nginx/html/index.html

initialDelaySeconds: 2 # Default 0

periodSeconds: 2 # Default 10

timeoutSeconds: 1 # Default 1

successThreshold: 1 # Default 1

failureThreshold: 3 # De

**Kubernetes Volume**

K8s üzerinde kalıcı depolma yapmayı sağlar

İki farklı volume tipi bulunmaktadır;

* Geçici Volumler

1. Ephemeral(EmptyDir)-> geçici volümlerdir. Pod silindiğinde pod içerisinde üretilen cache datalrıda silinecektir. Yada cresh olursada aynı şekilde

Not : Kısıtlı memory sahip ve cache kullanımı yapan uygulmalarınız varsa yada pod üzerinde çoklu konteynerlar arasında dosya paylaşımı yapmak isterseniz emptrydir volume bunun için oldukça kullanışlıdır.

Spec altında container ile aynı tab’da “volumes” olarak tanımlanır.

Volumes altına ne kullanıcığımı belirtiriz ayrıca containers tabı altında bunu kullanacağımızıda belirtmeliyiz   
  
ör:   
 spec:

Containers:

* Name: …  
  image:  
  volumeMount:
* name: redis-storage

mountPath : /data/redis -> hangi path’e bağlayacağımızı belirttik.  
  
 ----  
  
 volumes:

* Name :redis-storage  
  emptyDir:

Ör:

spec:

Containers:

* Name: …  
  image:

volumeMount:

* name: cache-volume

mountPath : /cache

* Name: …  
  image:….  
  volumeMount:
* name: cache-volume

mountPath : /cache  
 volumes:

Name :redis-storage  
emptyDir:{}

1. HostPath -> host üzerinde oluşturulan bir dosya pod içerisine güvenli bir şekilde bağlanabilmektedir. Node üzerindeki dosyaya hızlı bir şekilde erişmek için bu volume türünü kullanabiliriz test oratmı için elverişlidir. Production ortamında mümkün olduğunca kaçınılır. Çünkü güvenlik riski taşımaktadır. Dosya dizin bağlaması yapılcaksa bu read only olarak yapılmalıdır.Tek bir node üzerinde çalışmaktadır. Pod silinir yada cresh olursa datalar saklanmaya devam eder. Ancak node donanım arzasından dolayı giderse bu datalarda gider. Bu yüzden birden fazla node olduğu clusterda hostpath volume kullanımı data kaybınına neden olacağı için risklidir. Datalar kalıcı depolma alaını üzerinde saklanmalıdır. Bu depolama alınan tüm nodeler üzerinde erilmeli ve multi-cluster desteği olmalıdır.

Node üzerindwe oluşturduğumuz bir dosya ya da dizini pod içeirisine hızlıca bağlamak istedeğimizde kullanılmaktadır.

Not: single node için deçerli çoklu ortam için kullanışlı değildir.

spec:

Containers:

* Name: …  
  image:

volumeMount:

* name: data-volume

mountPath : /usr/share/nginx/html

volumes:

Name :data-volume  
hostPath:

Path: /demodisk

Type: - > bağlanacak klasörün tipini belirtmekteyiz(

direktory-> bu direktory var oraya bağlan

directoryOrCreate-> bu dosya bu path üzerinde varsa kullan yoksa oluştur.

fileOrCreate-> bu dosya bu path üzerinde varsa kullan yoksa oluştur.(tekil dosya)  
   
 )

Not : WorkingDir : -> pod ilk ayağa kalktığında dosyaları default olarak nerde arayacağın belirlemek için kullanılır. Default çalışma dizini olarak da düşünülebilir.

* Kalıcı Volumeler :

1. Persistent Volume(PV): Hernagi node bağlı olmayan kalıcı veri saklamak için kullanılır.

Depolam alanını ifade eder.

Persistent Volume Claim(PVC): alanın pod içerisine bağlanma alanını ifade etmektedir

Serverlar üzerindeki datalar sadece bu işler için oluşturlumuş olan storage alanında saklanmalıdır. Sunucu üzerindeki dataların storage üzerindeki disklere aktarabilmesi için farklı standartlar vardır;

* SAN (stroage area network) -> blok bazlı yüksek bağlantı hızı
* İscsi
* fc
* NAS (network attack storage) -> network üzerinde dataya bağlanmak istendiğinde kullanılır
* Nfs

Eğer elimizde kubernetes içide dahili olarak gelmeyen protokollerden birini kullanan bir storage ünitniz varsa bu storage üreticisini yazdığı csı driverlerını kullanarak k8s tanıtabilir ve kullanmaya başlayabiliriz.

Csı -> Container storage interface

K8s kalıcı depolama alanı yönetimide 3 farklı kavram bulunmaktadır;

1. Persistent Volume
2. Persistent Volume Claim
3. Storage Class

Kalıcı alan yaşam döngüsü ;

* provisioning ile başlar(yani nfs ,fc,iscsi gibi tanımlamalar yapılması lazım )Persistent volume denir
* Binding (alan talep etme işlemi ) Persistent Volume Claim denir
* Using (bağlanır ve kullanılır.)
* Pod işlemi yapıp silindiğinde (reclaiming işlemi gereçekleşir )

Persistent Volume işlemi için kalıcı depolama alanı üzerinde yapılandırma işlemi yapılmalı yani storage üzerindeki dataya hangi protocol üzerinde gerçekleştireceğiz ve kapasi ne kadar olucak işte tam da bu işleme Persistent Volume denir.

Ardından Persistent Volume claim (pvc) tanımlaması yaparak oluştuduğumuz Persistent Volume üzerinden alan talep edeceğiz ardından bu pvc alanını pod içerisine bağlayıp kullanacağız

Not : protocol tanımla işlemleri yapıldıktan sonra yöentici tarafından yapılan pv ve pvc tanımına static bainding adı verilmektedir. Bu yönetem haricinde storage class üzzerinden provisioner kullanılarak dynamick bindig tanımlama yöentemi bulunmaktadır.

apiVersion: v1

kind: PersistentVolume

metadata:

name: pervolume001

labels:

type: pv001

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

 **ReadWriteOnce**: Volume sadece tek bir node üzerine bağlanabilir.

 **ReadOnlyMany**: Volume birden fazla node’a sadece okunabilir modda bağlanır.

 **ReadWriteMany**: Volume birden fazla node’a hem okuma hem de yazma modunda bağlanabilir.

 **ReadWriteOncePod**: Volume tek bir Pod’a okuma-yazma şeklinde bağlanabilmektedi

apiVersion: v1

kind: PersistentVolume

metadata:

name: pervolume001

labels:

type: pv001

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain // volume ile işimiz bittiğinde ne olacak

### PersistentVolumeReclaimPolicy Açıklamaları:

* **Retain**: Volume üzerindeki veri korunur. PersistentVolume, silindikten sonra bile veri sistemde kalır. Manuel temizleme gerektirir.
* **Recycle**: Volume üzerindeki veri temizlenir (örneğin, rm -rf komutu ile). Volume, tekrar kullanılabilir hale gelir.
* **Delete**: PersistentVolume ve bağlı olan fiziksel storage otomatik olarak silinir. Bu işlem kalıcı olarak veriyi ortadan kaldırır.

apiVersion: v1 # API sürümü; bu PersistentVolume için v1 kullanılıyor.

kind: PersistentVolume # Kaynağın türü; burada kalıcı bir depolama (PersistentVolume) tanımlanıyor.

metadata: # PersistentVolume ile ilgili meta veriler.

name: pervolume001 # PersistentVolume'un adı.

labels: # PersistentVolume için etiketler (labels).

type: pv001 # Etiket tanımı; burada "type" etiketi "pv001" olarak tanımlanmış.

spec: # PersistentVolume'un özellikleri.

capacity: # PersistentVolume'un depolama kapasitesini belirtir.

storage: 10Gi # Depolama alanı miktarı; burada 10 GiB (Gibibyte) olarak belirtilmiş.

accessModes: # PersistentVolume'un erişim modlarını tanımlar.

- ReadWriteOnce # Volume yalnızca tek bir node üzerinde okuma-yazma olarak bağlanabilir.

persistentVolumeReclaimPolicy: Retain # ReclaimPolicy, PersistentVolume'un kullanım ömrü sona erdiğinde ne yapılacağını belirtir.

### **persistentVolumeReclaimPolicy Açıklaması:**

**persistentVolumeReclaimPolicy** bir **PersistentVolume** kullanım dışı kaldığında (örneğin, bir PersistentVolumeClaim silindiğinde) yapılacak işlemi tanımlar. Tanımlanabilecek değerler:

1. **Retain**: Volume üzerinde yer alan veriyi korur. Manuel müdahale olmadan veri veya volume silinmez.
2. **Recycle**: Volume üzerindeki veri silinir ve yeniden kullanılabilir hale getirilir.
3. **Delete**: PersistentVolume ve fiziksel depolama sistemi (örneğin, bir cloud disk) tamamen silinir.

accessModes:

- ReadWriteOnce # Volume yalnızca bir node üzerinde okuma-yazma modunda bağlanabilir.

persistentVolumeReclaimPolicy: Retain # Volume kullanım dışı kaldığında (ör. bir PVC silindiğinde) veriyi korur.

volumeMode: Filesystem # Volume'un işletim sistemi tarafından bir dosya sistemi olarak mı yoksa blok cihaz olarak mı kullanıldığını belirtir.

nfs: # Volume'un NFS (Network File System) protokolü kullanılarak tanımlandığını belirtir.

server: 192.168.99.100 # NFS sunucusunun IP adresi.

path: /nfs/mysql # NFS sunucusunda kullanılacak dizin yolu.

 **persistentVolumeReclaimPolicy**:

* Volume kullanım dışı kaldığında yapılacak işlemi belirler. (Retain: Veriyi korur, manuel müdahale gerekebilir.)

 **volumeMode**:

* **Filesystem**: Volume bir dosya sistemi olarak kullanılır ve dosyalar, dizinler oluşturulabilir.
* **Block**: Volume bir blok cihaz olarak kullanılır, dosya sistemi eklenmeden doğrudan kullanılabilir.

 **nfs**:

* **server**: Volume'un NFS sunucusunun IP adresi. PersistentVolume, bu sunucuyla iletişim kurar.
* **path**: NFS sunucusundaki dosya yolunu belirtir. Burada, /nfs/mysql dizini kullanılacaktır.

fc: # Fiber Channel protokolü ile tanımlanan bir volume.

targetWWNs: ["50060e801049cfd1"] # Fiber Channel Dünya Çapında Ad (WWN) hedef adresleri listesi.

lun: 0 # Logical Unit Number (LUN); depolama biriminin tanımlayıcı numarası.

readOnly: false # Volume'un yalnızca okunabilir olup olmadığını belirtir. (false: okuma-yazma modunda kullanılabilir.)

 **fc**:

* Fiber Channel (FC) protokolü, yüksek hızlı depolama ağlarına (SAN) bağlanmak için kullanılır. Bu yapılandırma, Kubernetes'in FC protokolü üzerinden bir depolama birimine erişmesine olanak tanır.

 **targetWWNs**:

* WWN (World Wide Name), bir Fiber Channel depolama cihazını benzersiz şekilde tanımlamak için kullanılır. Buradaki değer, FC bağlantısı yapılacak hedef cihazın WWN'sini belirtir.

 **lun**:

* Logical Unit Number (LUN), bir depolama birimindeki belirli bir mantıksal birimi temsil eder. Örneğin, lun: 0 ile belirtilen birim kullanılacaktır.

 **readOnly**:

* **true**: Volume yalnızca okuma modunda bağlanabilir.
* **false**: Volume hem okuma hem yazma modunda kullanılabilir. Burada false olduğu için okuma-yazma işlemleri desteklenir.

volumeMode: Filesystem # Volume'un bir dosya sistemi olarak kullanılacağını belirtir.

local: # Volume'un yerel bir depolama kaynağı olduğunu belirtir.

path: /mnt/disks/ssd1 # Volume'un fiziksel depolama cihazında bulunduğu yol.

 **volumeMode: Filesystem**:

* Volume, işletim sistemi tarafından bir dosya sistemi olarak kullanılacak. Örneğin, bu volume üzerinde dosya ve klasör oluşturulabilir.
* Alternatif olarak **Block** seçilirse, volume bir blok cihaz olarak kullanılabilir.

 **local**:

* PersistentVolume'un yerel bir depolama kaynağına (node üzerinde doğrudan bağlı) dayandığını belirtir.

 **path: /mnt/disks/ssd1**:

* Volume'un bulunduğu yerel depolama cihazının dosya sistemi üzerindeki yolu. Bu durumda /mnt/disks/ssd1 yoluna bağlanılacaktır.

volumeMode: Filesystem # Volume'un bir dosya sistemi olarak kullanılacağını belirtir.

hostPath: # Volume'un, ana makinedeki (host) bir yol üzerinden tanımlandığını belirtir.

path: "/mnt/data" # Ana makine üzerindeki kullanılacak yol.

 **olumeMode: Filesystem**:

* Volume'un bir dosya sistemi olarak kullanılacağını belirtir. Örneğin, bu volume içinde dosya ve klasörler oluşturulabilir.

 **hostPath**:

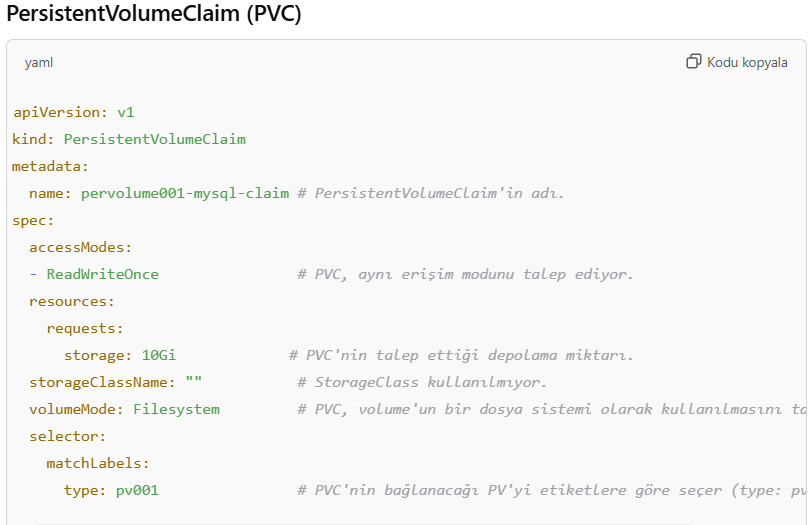
* Kubernetes cluster'ında bir node üzerindeki yerel dosya sistemini volume olarak kullanmayı sağlar.
* Bu yapı, genellikle test ortamlarında veya yalnızca bir node üzerinde çalışacak workload'lar için kullanılır.

 **path: "/mnt/data"**:

* Volume'un ana makinedeki (host) /mnt/data yoluna bağlanacağını belirtir.
* Bu, node üzerindeki fiziksel bir dizini Kubernetes pod'una bağlamanızı sağlar.

Hangi pv tanımında hangi driver modelin kullanıyorsak ona ait özellikleri yaml üzerinde belirtmeliyiz.(k8s sayfasında hangi volume driverini kullanacksak onunla ilgili yapılaması gerekn seçenekleri inceleyebiliriz)





 **PersistentVolume (PV)**:

* Kubernetes'in fiziksel veya sanal depolama kaynaklarını temsil eder.
* **labels**: PV'nin özelliklerini belirtmek için kullanılır. PVC'nin bu PV'ye bağlanması için bir eşleşme sağlar.

 **PersistentVolumeClaim (PVC)**:

* Kullanıcı veya uygulamanın depolama ihtiyaçlarını tanımlar.
* **selector.matchLabels**: PVC'nin belirli bir PV'ye bağlanmasını sağlar. Burada, type: pv001 etiketi üzerinden eşleşme sağlanmıştır.

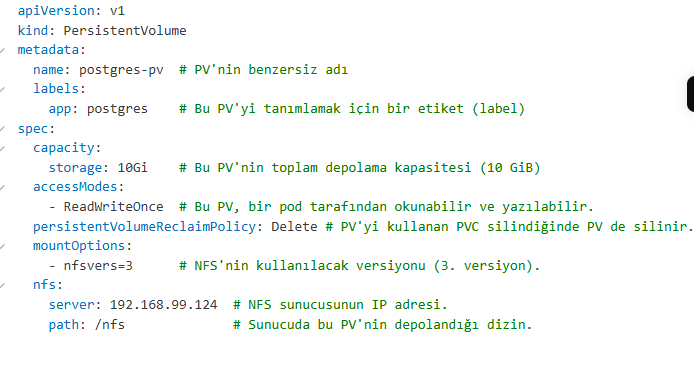
 **Eşleşme**:

* PVC'deki **selector.matchLabels** kısmı, PV'nin **metadata.labels** kısmındaki değerle eşleşir. Bu sayede PVC, doğru PV'ye bağlanır.

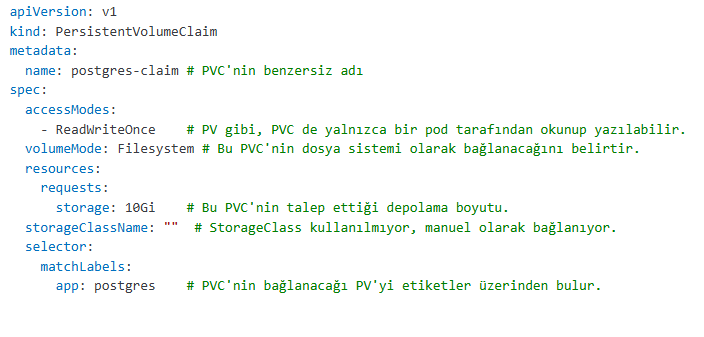


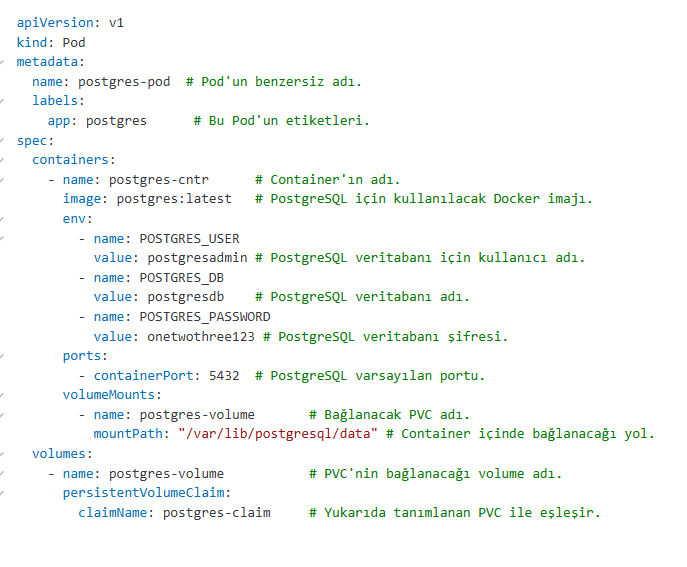
PV ve PVC Örnek :

PV:



PVC :



Pod:  


Storage Class :

* **Storage Class**, Kubernetes'te **Persistent Volume (PV)** oluşturmak için kullanılan bir şablondur.
* Farklı depolama türlerini (**hızlı**, **yavaş**, **dağıtılmış**) otomatik olarak yönetir.
* **Dinamik PV sağlama** (dynamic provisioning) sağlar. Yani, bir **Persistent Volume Claim (PVC)** oluşturulduğunda, otomatik olarak uygun bir PV tahsis edilir.
* Depolama biriminin hangi özelliklere sahip olacağını tanımlar (örneğin, hız, erişim türü, replikasyon).
* Farklı uygulamalar için farklı performans gereksinimlerine göre optimize edilmiş depolama sınıfları oluşturabilirsiniz.

**Fast (Hızlı Depolama):**

* **Tanım:** Yüksek performans gerektiren iş yükleri için kullanılır.
* **Örnek Kullanım:**
  + Veritabanları (PostgreSQL, MySQL).
  + Gerçek zamanlı analiz araçları.
* **Özellikleri:**
  + SSD (Solid State Drive) kullanır.
  + Yüksek okuma/yazma hızına sahiptir.

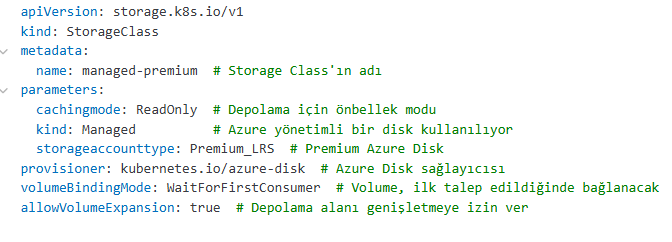
**Slow (Yavaş Depolama):**

* **Tanım:** Yüksek hız gerektirmeyen, maliyetin önemli olduğu durumlar için kullanılır.
* **Örnek Kullanım:**
  + Yedekleme alanları.
  + Arşivleme sistemleri.
  + Düşük erişim sıklığı olan veri.
* **Özellikleri:**
  + HDD (Hard Disk Drive) kullanır.
  + Daha düşük maliyetlidir ancak okuma/yazma hızları daha düşüktür.

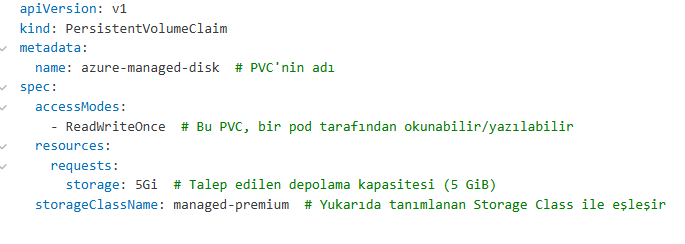
### **Storage Class Ne İşe Yarar?**

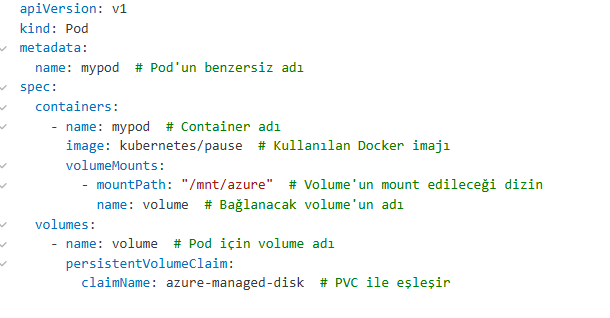
* Uygulamaların performans veya maliyet gereksinimlerine göre doğru depolamayı otomatik olarak seçmeyi sağlar.
* **Fast**, **Slow**, ve **Distributed** depolama türlerini bir arada kullanarak her uygulama için optimize edilmiş bir altyapı kurabilirsiniz.

Storageclass:



Pvc :



Pod:  


### volumeBindingMode **Nedir?**

* **Amaç:** PVC ile PV’nin bağlanma zamanlamasını ve işlemini kontrol etmek.

Parametreleri ;

### **1.** Immediate

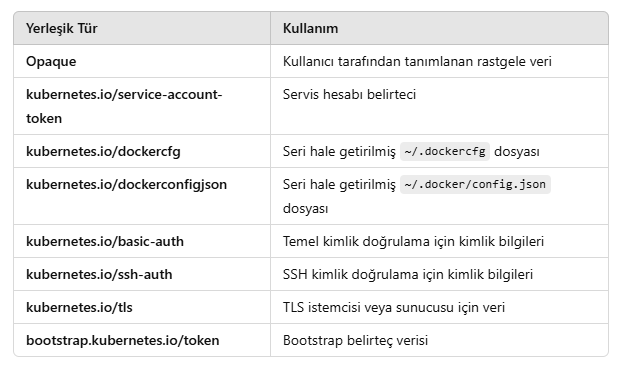
* **PVC ne zaman bağlanır?**  
  PVC, oluşturulduğu anda bir PV ile eşleştirilir.
* **Ne zaman kullanılır?**
  + **Tek bölgeli (single-zone)** ortamlar.
  + Hızlıca bağlanması gereken basit iş yükleri.
* **Avantaj:**  
  PVC hemen kullanıma hazır olur.
* **Dezavantaj:**  
  Çok bölgeli (multi-zone) sistemlerde sorun yaratabilir.

### **2.** WaitForFirstConsumer

* **PVC ne zaman bağlanır?**  
  PVC, bir **Pod** tarafından talep edildiği anda bağlanır.
* **Ne zaman kullanılır?**
  + **Çok bölgeli (multi-zone)** veya bölgeye duyarlı (zone-aware) altyapılar.
  + Kaynakların optimize edilmesi gerektiğinde.
* **Avantaj:**  
  PVC, Pod’un çalışacağı doğru bölgede (zone) bağlanır.
* **Dezavantaj:**  
  PVC'nin bağlanması, Pod'un oluşturulmasını bekler, bu biraz gecikmeye neden olabilir.

**Secrets –ConfigMap İşlemleri :**

**Secret Nedir ?**



Secret, Kullanıcı adı, şifre,token gibi hasas verileri saklamamıza ve yönetmemize imkan sağlayan k8s objesidir..Secret nesnesi sayesinde yaml dosyası içerisinde kullanamdığımız hassas verileri güven altında alabilmekteyiz.  
  
nasıl okuyacağız ?

* Secret nesnesi tanımlıyoruz
* Etcd db kayıt ediliyor
* Oluşturcağımz pod üzerinden bu bilgiyi sorguluyoru
* Bunu etcd db üzerinde arıyoruz ve bulunan secret bilgisi bulunan pod üzerine yönlendirilmekte ve kullanılmakta.
* Yani podlar bağımsız oluşturulmaktadır oluştululan secretler pod oluşutulurken sorgulanmakta ve kullanılmaktadır.

Hem impartive hemde declarative yöntem ile yapılabilmektedir.

* Kubectl create secet generic mysecret -> secret oluşturma
* Kubectl get secret- > secretları listeler.
* Kubectl create secet generic mysecret

--from-literal=db\_username=devuser

--from-literal=db\_password =’t0p-Secret123! ’

Dosyadan okuma:

Kubectl create secret generic mysecret3

--from-literal=db\_server=./server.txt

--from-literal=db\_ username =./username.txt

--from-literal=db\_password=./password.txt

Json formattai file secret olark ekleyebiliriz

Kubectl create secret generic mysecret4 –from-file=config.json

Decrative yöntem ile

Secret file :

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: Opaque

stringData:

db\_server: db.mydomain.com

db\_username: admin

db\_password: t0p-Secret123!

apiVersion: v1

kind: Pod

metadata:

name: secret-env-pod

spec:

containers:

- name: secretcontainer

image: redis

env:

- name: DB\_HOST

valueFrom:

secretKeyRef:

name: mysecret

key: db\_server

- name: DB\_USER

valueFrom:

secretKeyRef:

name: mysecret

key: db\_username

- name: DB\_USER\_PASSWORD

valueFrom:

secretKeyRef:

name: mysecret

key: db\_password

restartPolicy: Never

bir path üzerine mound edebiliriz->

apiVersion: v1

kind: Pod

metadata:

name: secretpodvolume

spec:

containers:

- name: secretcontainer

image: redis

volumeMounts:

- name: secret-vol

mountPath: /secret

volumes:

- name: secret-vol

secret:

secretName: mysecret

tüm secretleri pod içerisinde env olarak ekleyebiliriz->  
  
apiVersion: v1

kind: Pod

metadata:

name: secretpodallenv

spec:

containers:

- name: secretcontainer

image: redis

envFrom:

- secretRef:

name: mysecret

kubectl delete secret mysecre1 -> diyerek silebiliriz

**ConfigMap?**

Elimizde hasas olmayan data varsa bunu container içerisine aktarıp env olarak kullanamak istiyorsak kullanmalıyız.

Yada hasas olmaayan dataları container üzerine mound edip uygulamaın bu dosyayı okumasını istiyorsak kullanırız.

Uygulamalarımızda kulladığımız conf datalarını merkezi bir noktadan yönetmemizi sağlayan bi k8s objesidir.

* Kubectl create configmap myconfig1
* Kubectl get configmap
* Kubectl create configmap myconfig2

--from-literal=db\_server=db.domain.com

--from-literal= database=mydatabase

--from-literal=ui.textcolour=black,

* Kubectl create configmap myconfig3

--from-file=db\_server=/path/to/bar/file1.txt

--from-file= database=/path/to/bar/file2.txt

--from-file=ui.textcolour=/path/to/bar/file3.txt

yaml ile tanımlamak istersek ;

apiVersion: v1

kind: ConfigMap

metadata:

name: myconfigmap

data:

db\_host: "db.domain.com"

database: "mydatabase"

ui.properties: |

color.good=purple

color.bad=yellow

allow.textmode=true

confimapleri pod içinde kullanma :

apiVersion: v1

kind: Pod

metadata:

name: configmapped

spec:

containers:

- name: configmapcontainer

image: redis

env:

- name: DB\_SERVER

valueFrom:

configMapKeyRef:

name: myconfigmap

key: db\_host

- name: DATABASE

valueFrom:

configMapKeyRef:

name: myconfigmap // data okumak istediğimiz confmap ismi

key: db\_host // hangi değeri okuyakca onun anahtar değerini giriyoruz.

confipmapdeki tüm içeriği container içerinde bir path mount etmek istersek :

apiVersion: v1

kind: Pod

metadata:

name: configmappod

spec:

containers:

- name: configmapcontainer

image: redis

env:

- name: DB\_SERVER

valueFrom:

configMapKeyRef:

name: myconfigmap

key: db\_host

- name: DATABASE

valueFrom:

configMapKeyRef:

name: myconfigmap

key: database

volumeMounts: // container içerinde kullanmak içinde mount tanımladık.

- name: config-vol

mountPath: "/config"

readOnly: true

volumes:

- name: config-vol

configMap:

name: myconfigmap // eklemek istediğimiz confimap ismi yazıyoruz

**ReplicationController ve Replicaset:**

Oluşturduğumuz belli sayıdaki pod’un sorunsuz bir şekilde çalışığ çalışmadığını kontrol altında tutarlar. Eğer podlardan biri çakılırsa bu denetleyiciler devreye girerek dizayn state ile mevcut durumu current state değerini eşitlemeye çalışmaktadır.

Replicaset ,replicationcontroller’in güncel halidir.

ReplicationController nesnesiyle hem pod scale işlemleri hemde container image sürüm güncelleme işlemleri yapılmaktaydı. Hantal olduğu ve hatlar yaptığı gözlemlendi. Ardından replicationcontroller , replicaset ve deployemnt set olarak ikiye ayrıldı.

Replicaset nesnesi denetleyici olarak,deployment nesnesi ise rollout işlemleri için kullanılmaya başlandı.

Replicationcontorller equality-based yazım şeklini benimserken Replicaset hem set-based hemde equlity-based yazım şeklini benimser.

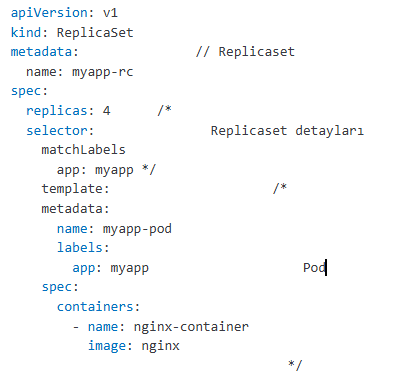
RC VE RS tanımları:  
  
Replicas(kopya sayısı ) tanımıyla verilen değere göre pod seti oluşturulacak ve sürekli takip edilecektir.

Pod Templete : Pod seti oluşturulurken hem pod için girilecek standart ayarlar pod seti üzerinde yapılacaktır.  
 ör: podlar üzerine hangi label bilgisi atancak hangi imaj kullanılcak container isim standartı ne olacak gibi bilgiler pod template üzerinde ayarlanmaktadır.

Selector: replicaset yada replication controller kendi yönettikleri podlara hangi label üzerinden erişm sağlayacaklar ve işlem yapacaklar bunu belirlemekteyiz.

Pod’lara verdiğimiz labelları selector yardımyıla seçip yardım olacağız.

Örnek yaml dosyası :



Replicaset deployment nesnesi ile birlikte kullanılmaktadır.

Scale (ölçeklendirme)-> pod sayısını manuel yada otomatik olarak çoğalta bilir yada azaltabiliriz bunu için ;  
 kubectl scale replicas=rs/myapp

Kubectl scale replicas=10 rc/myapp

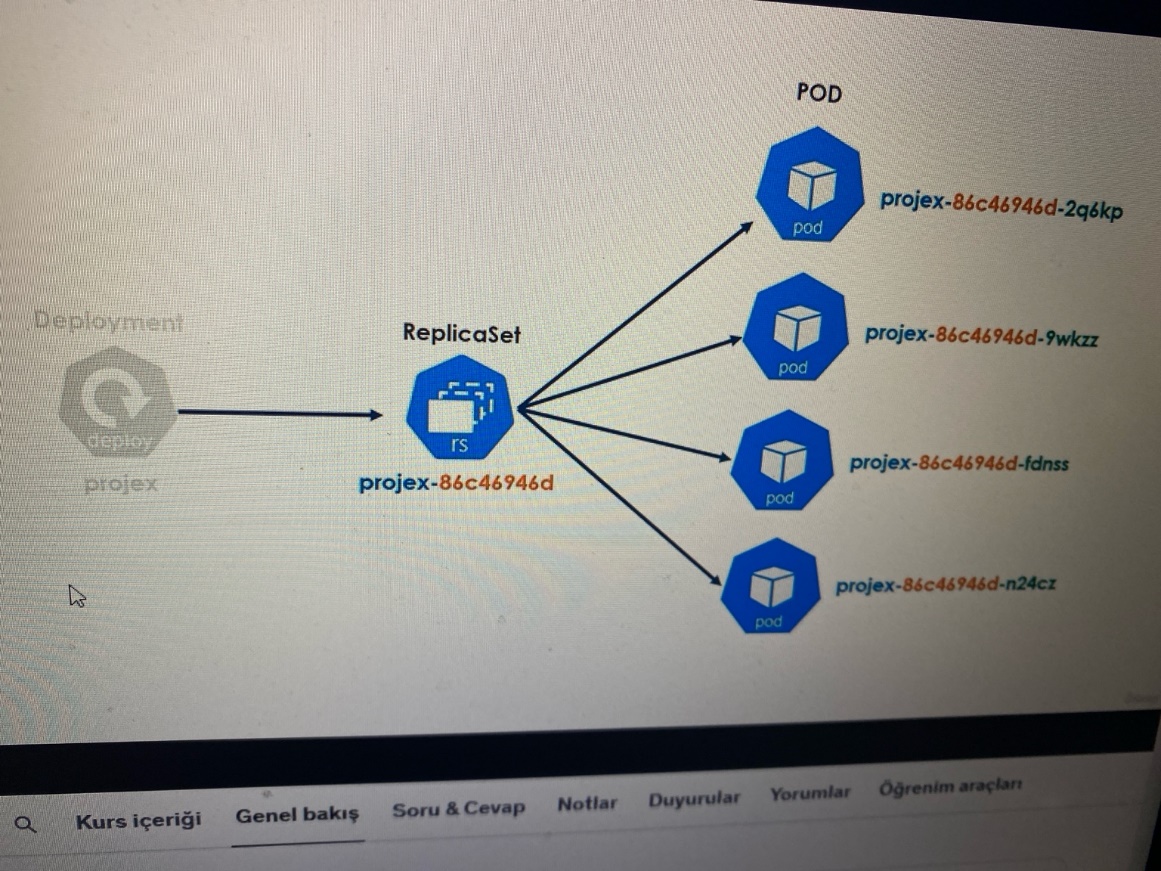
Kubectl scale replicas=10 –f rs.yaml -> yaml dosyası ile scale işlemi yaptık.

Kubectl autoscale rs –min=2 –max=5 --cpu-percent=80 -> min 2 max 5 podu cpu itüliazyonu %80 olana kadar otomatik olarak oluşutacak veya sistemden kaldırılacak

Kubectl delete replicaset myapp -> replicaset sildik.

Kubectl delete replicaset myapp –cascade=false -> replicaseti sil podlara dokunma

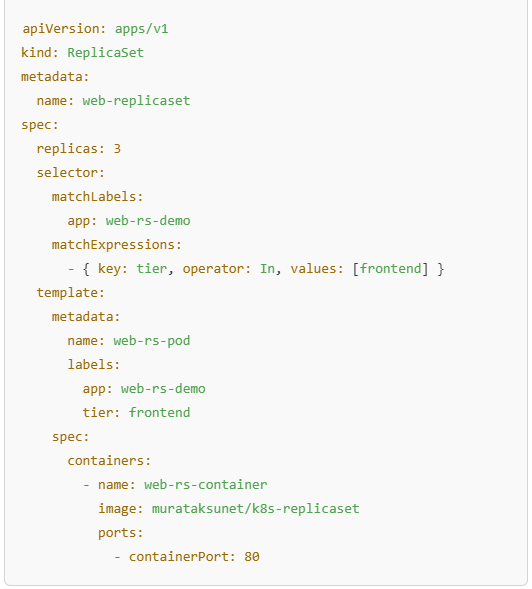
Deployment nesnei oluşturduğumda sistem otomati arkada bir replicaset nesni oluşuturur ve podları bu rs üzerinden takip eder rs deployment neseninin ismin ön ek olarak alır rs içeriisnde oluştuulan pod nesnelerinin isimleride rs nesnesini isimlerini ön ek olark alır



Replication Controller Uygulaması :



Replicaset Uygulaması:



**Deployment:**

Deployment nesnesi,kubernetes üzerinde bulunan replicationcontroller yerine geliştirilmiş şuan en çok kullanılan kubernetes objelerinden biridir.

Deployment nesnesinin üç farklı görevi vardır;

1. Replicaset Yönetimi
2. Rolling Update
3. Rollback
4. Replicaset Yönetimi:

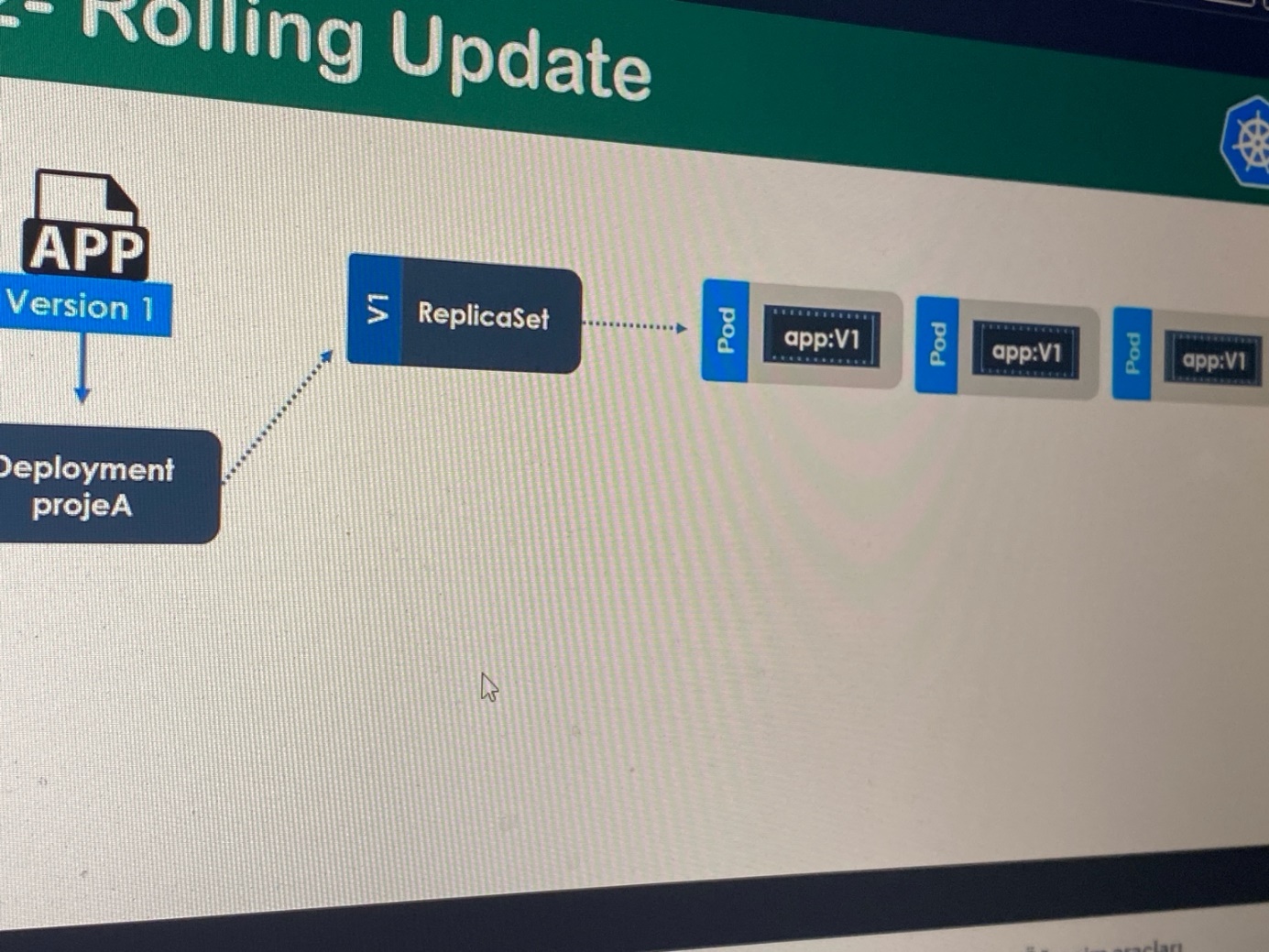
Belirmiş olduğumuz Replicas değerine göre arka tarafta replicaset nesnei oluşturmuş olur böylece uygulamalrımızın devamlı olur .

Eğer Uygulamamızın farklı imaj sürümleri varsa tekbir deployment nesnesi üzerinde birden fazla rc oluşturup yönetim işlemi yapabiliriz.

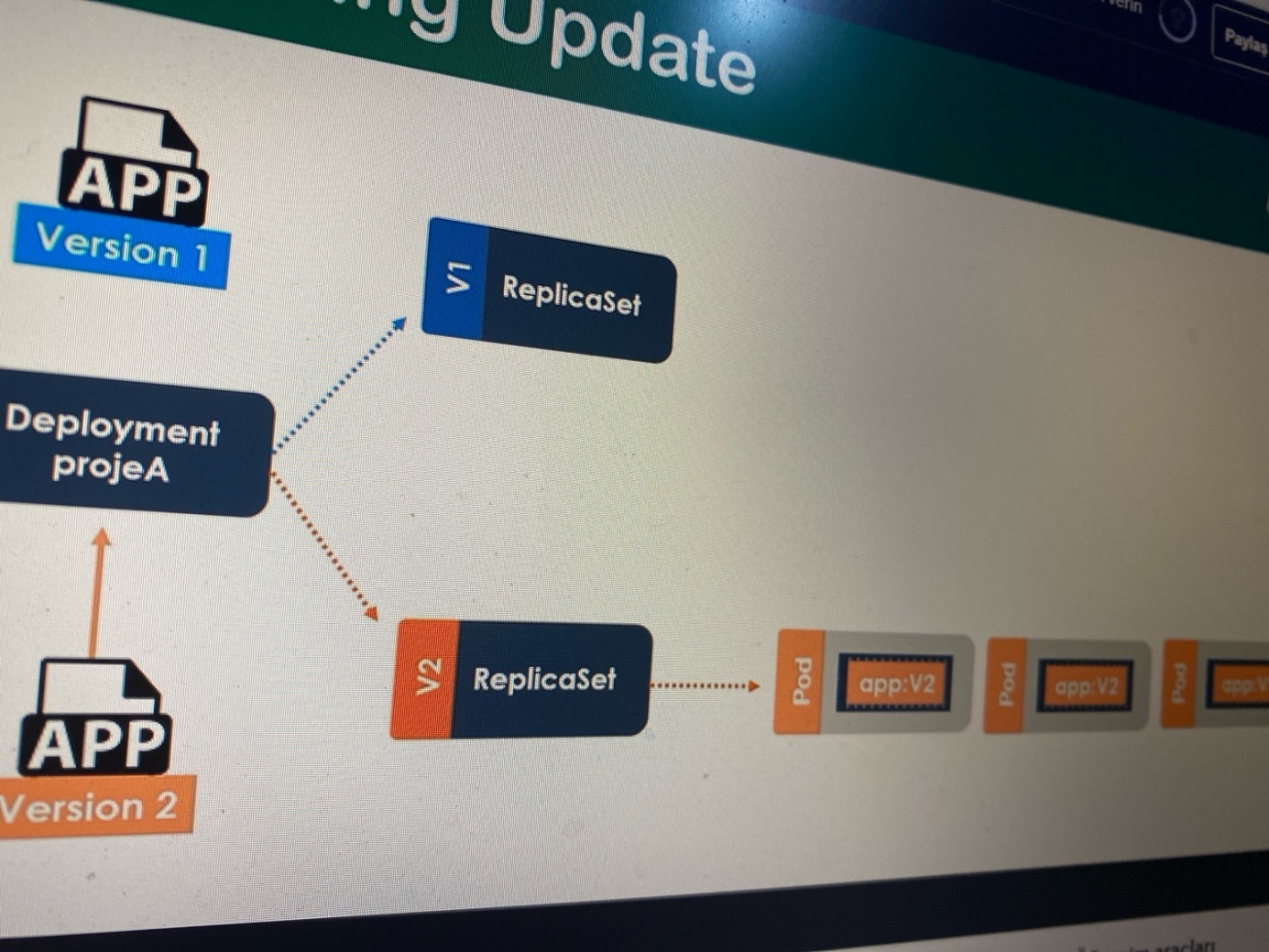
Her bir imaj güncellemesi yeni bir rc oluşturulmasına neden olur.

1. Rolling Update :

Uygulamamızın v1 imajı kullanılarak deployment nesnesi oluşturuldu bu deployemnt nesnesi arka tarafta replicaset nesnesi oluşturacak ve replicaset talep edilen pod sayısı oluşturacak ve çalıştıracak.  
  
Bizim uygulamamızın V2 imajı çıktı bu durumda deployemnt üzerine gidip çalışan imajları yeni sürümle güncelle diyerek güncelleme işlemi yapılacak eski sürümle çalışan podlar kaldırılacak ve yeni sürümle podlar oluşturulacak ve bu podlar üzerinde sistemin çalışması sağlancak.

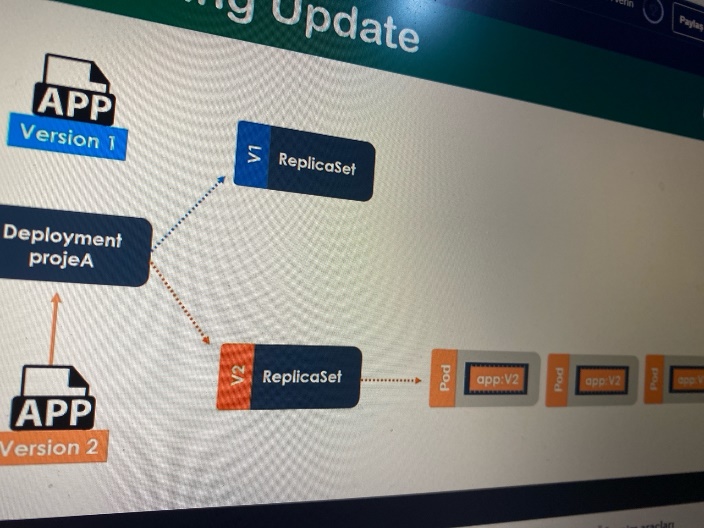


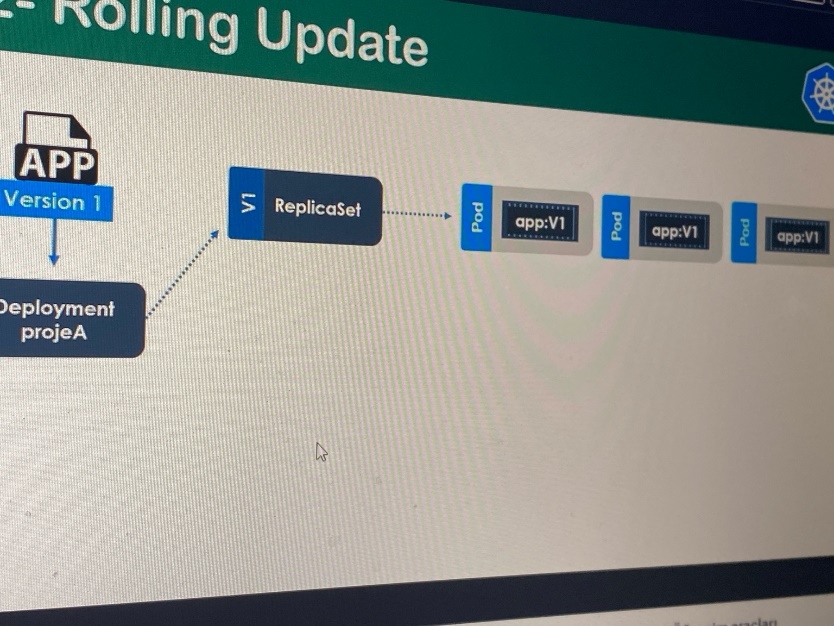
Container içeride çalışan imaj sürümlerinin Deployment nesnesi üzerinden güncellem işlemine Rolling update denir.



1. Rollback:

Yeni sürüme geçtik yeni imaj sürümünde sistemin çalışmasını etkileyen sorunlar çıktı bir önceki versiona döndürmeye yarar.





Birden fazla imaj güncellem işlemi yaptığımızda bunlardan istediğimize geri dönebiliriz.

Komutlar:

kubectl create deployment/mydep2 --image=nginx --replicas=5

kubectl get deployment

kubectl get replicaset

 **NAME**: ReplicaSet’in adı (örneğin, mydep2-6dff996d85).

 **DESIRED**: İstenen (oluşturulması planlanan) Pod sayısı (örneğin, 5).

 **CURRENT**: Şu anda mevcut olan Pod sayısı.

 **READY**: Hazır Pod sayısı.

 **AGE**: ReplicaSet’in çalışmaya başlamasından bu yana geçen süre.

kubectl scale deployment/mydep2 --replicas=2

kubectl edit deployment/web-deploy

 **replicas: 7** olarak değiştirilirse, Pod sayısı 7’ye yükseltilir.

 **selector: matchLabels** bölümünde değişiklik yapılarak hangi Pod'ların eşleşeceği belirlenebilir.

Hatıtrlatnam :

kubectl create deployment mydep3 --image=nginx --dry-run=client -o yaml > demodeploy.yaml ->

 **kubectl create deployment**: Yeni bir Deployment oluşturur.

 **mydep3**: Deployment adı.

 **--image=nginx**: Kullanılacak konteyner imajı.

 **--dry-run=client**: Komut sadece simüle edilir, Kubernetes kümesinde herhangi bir değişiklik yapılmaz.

 **-o yaml**: Çıktıyı YAML formatında üretir.

 **> demodeploy.yaml**: Üretilen YAML çıktısı demodeploy.yaml adlı dosyaya kaydedilir.

Deployment :



**Kubernestes Deployment Strateji Türleri :**

Containerlar içerisinde imajları güncelle işlemlerini iki farklı rollout stratejisi bulunur

1. Recreate : eski podların tamamı aynı anda kaldırılır yeni sürümlü podlar aynı anda hizmete başlar. Bu yapı eski sürümle yeni sürüm uyuşmalığı yada hataları gibi durumlarda gerçekleşir.
2. Rolling Update : Defaulttur. Varolan tüm podların aynı anda kaldırmaz belirli bir sıra ile kaldırır.

Aynı anda kaç pod kaldıralacağını seçebiliriz;

MaxUnvailable ->aynı anda silinecek pod sayısı belirtir. % ile de değer verebiliriz

maxSurge -> geçiş sırasında sistemde toplamda en fazla kaç pod olacağını belirlemekteyiz. (güncelencek olan)

minReadySecods-> yeni oluşturulan bir podun kullanılabilir olması için geçmesi gereken min süreyi belirtmekteyiz (default : 0 )



komutlar :

kubectl rollout history deployment/nginx-deployment

 **kubectl rollout history** → Deployment veya başka bir kaynağın güncelleme geçmişini gösterir.

 **deployment/nginx-deployment** → Güncelleme geçmişi incelenecek Deployment.

 **REVISION**: Deployment’ın kaçıncı sürümde olduğunu gösterir.

 **CHANGE-CAUSE**: Eğer güncellemeye bir açıklama eklenmişse burada gösterilir (bu örnekte <none> olarak görünmektedir).

kubectl rollout undo deployment/nginx-deployment**kubectl rollout undo** → Bir önceki başarılı Deployment sürümüne geri döndürür.

* **deployment/nginx-deployment** → Geri alınacak Deployment.

**Bu komut çalıştırıldığında:**

* Kubernetes, Deployment’ın son güncellemesini iptal eder.
* Eski çalışan Pod sürümüne geri döner.

kubectl rollout undo deployment/nginx-deployment --to-revision=1

Bu komut, **nginx-deployment** adlı Deployment’ı **belirtilen sürüme (revision 1) geri döndürür**.

* **--to-revision=1** → Deployment’ı doğrudan **1. sürüme** geri döndürür

 **Güncelleme geçmişini listele** → kubectl rollout history

 **Son güncellemeyi geri al** → kubectl rollout undo

 **Belirli bir sürüme geri dön** → kubectl rollout undo --to-revision=X

kubectl set image deployment/nginx-deployment nginx-container=nginx:1.16.1 --record=true

 **kubectl set image** → Mevcut bir Deployment içindeki bir konteynerin imajını güncellemek için kullanılır.

 **deployment/nginx-deployment** → Güncellenmek istenen Deployment’ın adı.

 **nginx-container=nginx:1.16.1** → Güncellenecek konteynerin adı ve yeni Docker imajı.

 **--record=true** → Yapılan değişikliği **Deployment’ın sürüm geçmişine kaydeder**.

Yani **CHANGE-CAUSE kısmna komut sonuna ,** --record=true eklenerek sürüm bilgisini yazdırmış olacağız