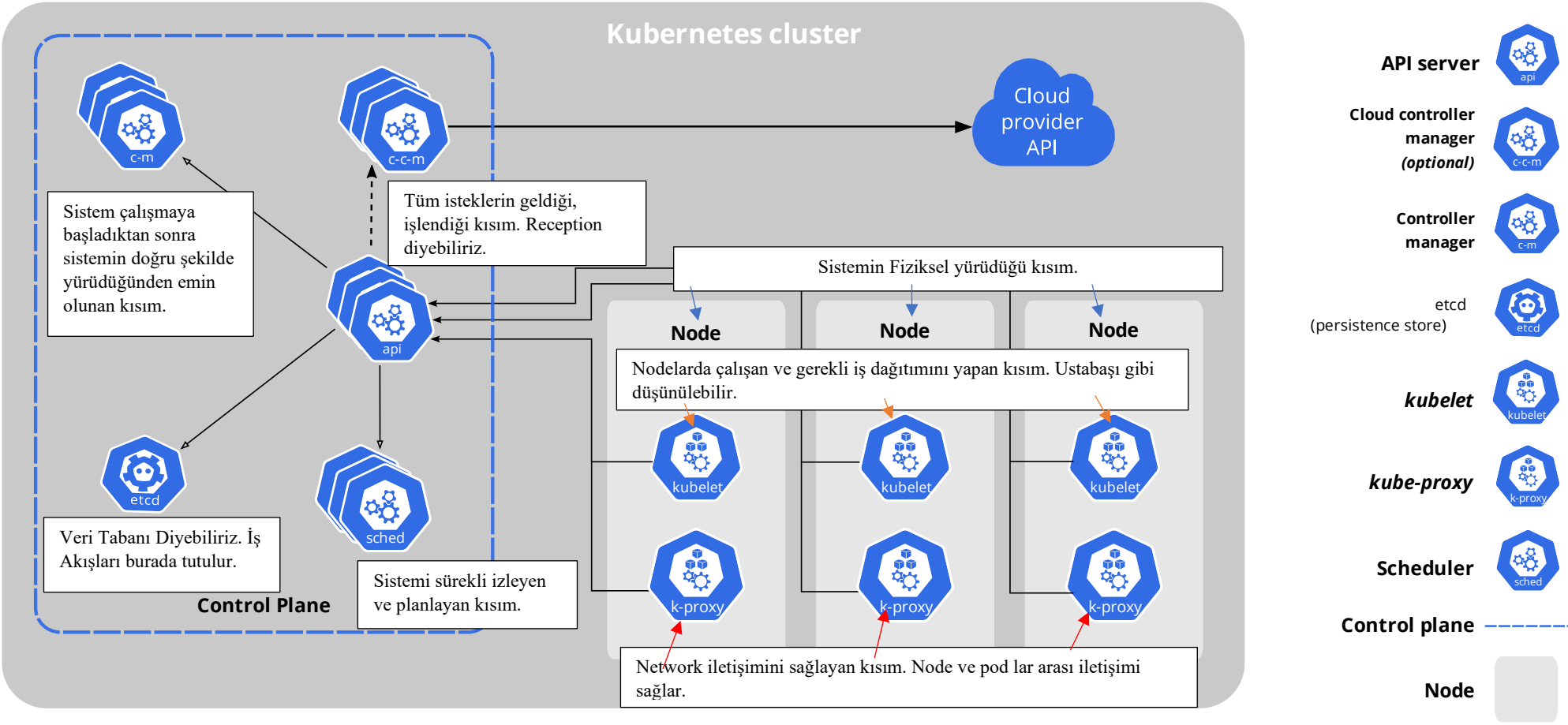
KUBERNATE

Kubernate Architecture

Declarative Yöntem ile çalışan, Container Orchestration bir araçtır. Ne yapmak istediğinizi söylüyorsunuz o sizin adınıza tüm işleri hallediyor.



KUBERNATES BİLEŞENLERİ

- ### Kube-apiserver:

  Kubernates API nı ortaya çıkartan Control Plane yapısının en önemli bileşenidir. Tüm sisteme giriş ve çıkışlar buradan yapılır. Ayrıca diğer tüm komponent ve node bileşenlerinin direkt erişimde bulunduğu tek yerdir.

  Tüm istekler buradan gerçekleşirken, Authentication ve Autohorize işlemlerinin yapıldığı kısımdır. İstekleri Rest Api isteği olarak alabileceği gibi "kubectl" gibi araçlarla da çalışabilir.

- ### Etcd:

  Tüm sistem verisini tutar. MetaData verileri ve Kübernates içinde oluşturulan tüm obje bilgilerinin tutulduğu Key-Value veri deposudur. Cluster in mevcut yapılandırmasının tamamı buradadır. Kube-ApiServer dışında diğer bileşenler etcd ile haberleşemezler.

- ### Kube-scheduler:

  Yeni oluşturulan ya da bir node a ataması yapılmamış Pod ları izler ve üzerinde çalışacakları bir nodes seçen kısımdır. Aslında sistemde var olan zorunluluklar, tainler v.s. bilgilerine göre bir pod un çalışabileceği en uygun node u seçerek oraya atamasını yapar.

- ### Kube-controller-manager:

  Her kontroller ayrı bir süreç izler, ancak karmaşıklığı azaltmak adına hepsi bir binary olarak derlenmişlerdir ve tek bir Process olarak çalışırlar. Etcd de saklanan verileri inceleyerek sistemi gözlemler, eğer istenilen durum ile hali hazırda bulunan durum arasında fark var ise sistemi istenilen ayarlarına getirir. Güncelleme, silme işlemlerini yapar. EWS, Azure gibi sistemlerin entegrasyonlarında "Cloude controler Manager" adı verilen ve yönetimi sağlayan ayrıca bir ara katman vardır. Bunları örneklemek gerekirse:

  - Node controller
  - Service Account & Token controller
  - Endpoints controller
  - Job controller

NOT: Kurbernates yapısının yönetim kısmını oluşturan (resimdeki sol taraf) Master Node olarak isimlendirilir. Sistem yükünün ve posların çalıştığı (resimdeki sağ kısım) Worker Node olarak adlandırılır.

- ### Kubelet:

  Cluster da her node için çalışan agent türü bir yapıdır. Pod üzerinde çalıştırılmak istenilen container(image) ların çalıştırılmasını sağlar. Kubelet, çeşitli sistemler aracılığı ile gerekli emirleri alır ve tanımı yapılan Pos sistemlerinin üzerinde containlerların sağlıklı bir şekilde çalışmasını garanti eder.

- ### Kube-proxy:

  Node lar üzerinde ki ağ kurallarını yönetirler. Bu ağ yapısı cluster içinde ya da dışında olabilir. Pod larımıza yapılacak olan ağ oturumlarının iletrişiminden sorumludurlar.

- Kubectl Kurulumu:

Önemli Not: Kubernates kurulumların yapmadan önce local de kubernates çalıştırılacak ise ortamda bir Docker Desktop kurulu olması faydalı olacaktır.

https://kubernetes.io/docs/tasks/tools/ adresinden işletim sistemine göre gerekli kurumların yapılması gereklidir.

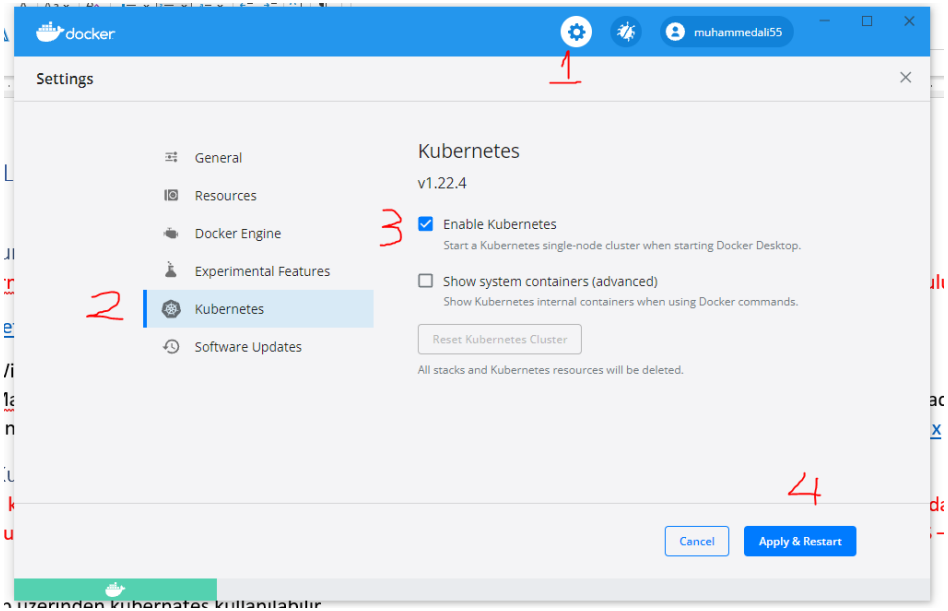
- Windows: https://dl.k8s.io/release/v1.23.0/bin/windows/amd64/kubectl.exe
- MacOS: https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/#install-with-homebrew-on-macos (adresinden brew, curl ile)
- Linux: https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-binary-with-curl-on-linux (adresinden curl, apt ile)

- Kurbernates Kurulumları:

Önemli Not: Kişisel kullanım, Test ve Geliştirme için Minikube tool ile kurulum yaparak kullanmak ilk defa kullananlar için faydalı olacaktır. Ayrıca Bulut sistemlerin sunduğu servislerde vardır. Bunları ilerleyen zamanlarda göreceğiz. Azure Kubernates Service(AKS) – Amazon EKS – Google Kubernates Engine

1- Docker Desktop: üzerinden kubernates kullanılabilir.
Yandaki şekilde görüldüğü üzere docker desktop açılarak Ayarlar->Kubernates Tıklanır ardından Enable Kubernates Seçilerek Apply & Restart denilerek aktif edilir.
Sitemin çalıştığı;
Power Shell de yazılacak olan

➢ Kubectl get nodes
Komutu ile test edilebilir. Burada docker-desktop un Çalıştığı grülebilecektir.



2- Minikube: https://minikube.sigs.k8s.io/docs/start/ adresinden işletim sistemine göre kurulumu yapılır. Ardından windows ta PowerShell (Admin modunda) açılarak

➢ Minikube start
Komutu çalıştırılır ve sistem kurulumun yapılması sağlanır. Ardından test etmek için yine komut satırında “kubectl get nodes” yazılarak minikube un çalıştığı gözlemlenir.

Önemli Not: Eğer sisteminizde birden çok sanallaştırma var ise, istediğiniz kaynağı seçmek için aşağıdaki komut satırlarını kullanınız.

- Minikube start --driver=hyper-v
- Minikube start --driver=docker
- Minikube start --driver=virtualbox

KUBECTL CONFIGURASYONLARI

Kubectl in config dosyası (C:\Users\MuhammetAli\.kube) şeklindeki bir klasör içinde adı (config) olan bir dosyadır. Bu dosyada cluster yapıları, varsa tüm kubernate bağlantı ayarlarını barındırır. Hangi kullanıcı hangi alanlara bağlanacak bilgilerin barındırır.

Kubectl yönetim aracı bağlanacağı Kubernates cluster lara config dosyası aracılığı ile ulaşır. Config dosyasına bağlantı bilgilerini ve bağlanırken kullanmak istediğimiz kullanıcı bilgilerini belirtiriz. Bu bilgileri kullanarak birden fazla namespace oluşturabilir ve contextlerini yaratabiliriz.

Bu yapılandırma dosyasının aracalığı ile ilk öğrenmemiz gereken komutlar şöyledir.

- Kubectl config get-contexts, var olan tüm bağlantıları göterir.
- Kubectl config current-contexts, aktif olan context adını verir.
- Kubectl config use-context docker-desktop, aktif context i istenilen contex ile değiştirir.

KUBECTL KOMUTLARINA GÖZATALIM

- **Kubectl get --help,** burada önemli olan help parametresinin kullanılması çünkü tüm komutları ezberlemek imkansızdır. Bu nedenle komutları kullanmak istediğinizde yardım alın.
  Önemli Not: ilk olarak kubectl yazılır, ardından bir fiil işlem yapacak kelime gelir (get gibi) sonra çalışacak komutun hangi türden bir nesne üzerinde çalışacağını belirtiyoruz(pods, nodes gibi).opsiyonel olarakte üzerinde çalışacağımız objenin özel adını giriyoruz.
- **Kubectl cluster-info,** cluster ile ilgili bilgileri verir.
- **Kubectl get pods,** aktif namespace üzerinde çalışır ve tüm pod ları listeler
- **Kubectl get pods -n** docker-desktop, şeklinde yazarsam bu docker üzerinde ki pod ların listesini dönecektir.
- **Kubectl get pods -A (--all-namespace=false) – (--all-namespace),** şeklinde ki kullanım ise tüm namespace lerde var olan pod ları listeyecektir.
  Not: Default değer true
- **Kubectl get pods -A -o wide (yaml , json),** şeklindeki kullanım tüm podları geniş şekilde detayları ile gösterir. Yaml ve json olarak ta alabilirsiniz.
- **Kubectl get pods -o json jq -r ".items[].spec.container[].name**", jq aracını kullanarak sadece isimleri listeyebilirsiniz.(NOT: Yeni kullnımlarına bakılacak.)
- **Kubectl explain pod**, pod objesinin ne olduğu ve hangi alanlarının olduğu bilgilerini döner.

- ## Pod:

  En küçük objemiz pod dur. Podlar bir ya da zorunlu olduğu durumlarda birden fazla container barındırabilirler. Ama genel tek kullanılır. Her pod un benzersiz bir uid si vardır. Her pod benzersiz bir ip adresi barındırır. Aynı pod üzerinde var olan containerlar aynı nodes üzerinde çalıştırılır ve bir birleri ile localhost üzerinden haberleşirler.

- ## Pos Oluşturmak:

  - Kubectl run bilgepod –image=nginx –restart=Never, adı bilgepod olan içerisinde ngnix imajını barındıran ve hata durumunda tekrar çalıştırılmamasını istediğim bir pod olulturuyorum. Pod ile ilgili alternatif kodlar;

    Test Et(kubectl get pods -o wide).

    Geniş bilgi için(kubectl describe pods [podname]).

    Pod a ait log bilgisi için(kubectl logs [podname]).

    Eğer canlı bir şekilde izlemek istersen(kubectl logs -f [podname])

    Pod üzerinde komut çalıştırmak istersen(kubectl exec firstpod -- hostname)

    Pod üzerine bağlanmak istersen(kubectl exec -it [podname] -- bash) birkaç komut dene -> ls , printenv, hostname v.s.

    Pod u silmek istersen (kubectl delete pods [podname])

- ## YML Dosyası ile çalışmak:

  Pod ve diğer objeleri oluşturmak için en mantıklı yöntem YML dosyaları ile çalışmaktır. Bu nedenle, pod oluşturukende yml kullanmak her zaman daha faydalı olacaktır. Bir pod dosyası şuna benzer; →

  Bir yml dosyasını kubernate declare etmek için şu komutları kullanırız;

  - Kubectl apply -f podcreate.yml

YML dosyaları ile çalışmanın güzel tarafı, podlarda bir değişiklik yapacağımız zaman sadece değiştirmek istediğimiz ya da eklemek istediğimiz parametreleri giriyor ve yeniden appyl komutunu kullanıyoruz.

podscreate.yml 1 ●

podscreate.yml > {} spec > [ ] containers > {} 0 > [ ]

io.k8s.api.core.v1.Pod (v1@pod.json)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ilkpod
  labels:
    name: front-end
spec:
  containers:
  - name: ilkpodname
    image: nginx:latest
    ports:
    - containerPort: 80
```

- Bir Pod un Yaşam Döngüsünü İzlemek

Gerçekte olmayan bir imaj dosyasını eklemek istersek, sistem bu pod u ayağa kaldıramayacak ve reset atacktır, bir yerden sonra sistem imajın ulaşılamaz olduğunu görünce sistemi hataya çekerek, belli aralıklarla tekrar tekrar deneme yapacaktır. Bu deneme süresi aralıklı artarak devam edecektir. Hadi deneyelim;

- Olmayan bir image ile pod oluştur. kubectl apply -f pod.yml
- Sistemi izlemek için -> kubectl get pods -w (izle komutudur.)
- İzleme ekranında önce hataya sonra, imagepullbackoff a geçtiği görülecektir.
- Eğer spec: altına restartPolicy: Never eklersen, image olmadığı için pod tekrar oluşturulmaycaktır.

- Label ve Selector

Sistemimizde bir çok alanada görev yapan microservis yapısında, developer, test, front-end v.s. bir çok pod olabilir. Bunların yönetimi ve gruplanması önemlidir. Bu nedenle label yapısı çok iyi kurgulanarak kullanılmalıdır. Anahtar ve Değer alanı olarak çalışırlar, dikkat etmemiz gereken konu 63 karakterden uzun isim oluşturmamaktır.

Yanda görüldüğü üzere birden çok pod var ve belli podlar belli takımlar için isimledirilmiş. Eğer tüm podları label ile görüntülemek isterseniz.

➔ Kubectl gete pods -l “group” --show-labes
➔ Kubectl gete pods -l “group=team1” --show-labes

- Annotations

Pod objelerimize özel olarak eklemek istediğimiz açıklamaları bildirmek için kullanırız. Label olarak eklenmesi sakıncalı bilgiler burada tanımlanır.

- NameSpace

Bu yapı temel olarak, file sistem kısmıdır. Sitemin işleme mantığını anlamak için şöyle bir kurgumuz olsun; bir firmada IT olarak çalışıyorum. Tüm ekiplerin erişebileceği bir dosya sistemine ihiyacım var bende bir file server kurdum ve herkese bunu açtım. Sistem bir süre iyi çalışır. Sonra herkesin tek bir alana dosyalama yapması nedeniyle yüzlerce binlerce dosya oluşur ve bunları yönetmek güvenliğini sağlamak zorlaşır. Aynı isimde dosya oluşturulamaz, oluştursa benim dosyamı ezebilir. V.s.

Bunu çözmenin kolay yolu. Her ekip için ayrı bir klasör oluşturmak ve gruplara yetki vermek yerelidir. İşte tam burada kubernate içinde de biz namespace ile klasör oluşturabilir, kota, izin gibi ayarları yaparak sistemi işler hale getirebiliriz.

➢ Kubectl get namespaces, komutu sistemde ki namespace leri listeler. İlk kurulumda 4 adet namespace vardır. Özellikle bir name space belirtilmedikçe tüm nesneler default-namespace altında oluşturulur.

Mesela farklı bir namespce altında var olan podları görüntülemek istesek, kubectl get pods --namespace kube-system komutunu kullanırız.

YML dosyası ile buna bir örnek verelim;

Yml dosyasından pod oluşturulduktan sonra

- Kubectl get pods derseniz oluşturduğunuz pod u göremezsiniz. Sebebi ise siz default namespace içnidesiniz. Bu nedenle sorgunuz şöyle olmalı
- Kubectl get pods -n develop-name
- Ayrıca kod çalıştırma v.s işlemlerinden önce de bunu belirtmelisiniz.
- Kubectl exec -it [podname] -n develop-name – bash
- Eğer varsayılan namesapace i değiştirmek istiyorsanız o zaman
- Kubectl config set-context –current –namespace=develop-name

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name:  develop-name
---
apiVersion: v1
kind: Pod
metadata:
  namespace: develop-name
  name: ilkpod
  labels:
    name: front-end
    group: team1
  annotations:
    owner: "Muhammet Ali KAYA"
    email: "muhammedali55@gmail.com"
    craeteAt: "07.02.2022"
spec:
  restartPolicy: Never
  containers:
  - name: ilkpodname
    image: nginx:latest
    ports:
    - containerPort: 80
```

- Deployment:

Burada deployment objesinin kavramak için senaryolarımızı değerlendirelim.

- Yml dosyası ile bir pod oluşturduk, podumuz A numaralı node üzerinde çalışıyor diyelim. Bir sorun oldu ve node kapandı. Böyle bir durumda kube-sched pod u başka bir node üzerinde çalıştırmaz arkadaşlar bu nedenle projemiz down olur.
- Bu sorunu aşmak için her node üzerinde çalışmak üzere örneğin, 10 pod.yml dosyası yaptık ve tüm node lar üzerinde çalıştırdık diyelim. Peki uygulamamızı güncellememiz gerekirse ne olacak? Yml dosyasını değiştireceğim, sonra tüm node larda podları tekrar güncellemem gerekecek, işler tekrar karışmaya başladı.

İşte tam burada Deployment, bizim adımıza istenilen durum ile mevcut durumu kontrol etmek üzere çalışır. Eğer sistemde bir sorun var ise pod umuzu tekrar ayağa kaldırır. Gerekirse bir node tan diğer bir node ta ayağa kaldırır. Ama sistemin çalışmasını garanti eder.

Gerek yok ama elle oluturmak isterseniz;

- Kubectl create deployment ilkdeploy --image=nginx:latest --replicas=2
- Kubectl scale deployment ilkdeploy --eplicas=5

Burada kontrol yapmak için bir bash içinde -w ile izleme ye geçin, diğer bash üzerinde “delete pods [podname]” ile her hangi bir pod u silip sistemi takip edin.

YML ile çalışmak -> Burada önemli olan selector kavramı hangi deployment ın hangi podları yöneteceği burada belirlenir.

podscreate.yml > {} spec > {} selector > {} matchLabels >
io.k8s.api.apps.v1.Deployment (v1@deployment.json)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: authdeployment
  labels:
    team: developers
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: nginx
          image:  nginx:latest
          ports:
            - containerPort: 80
```

Önemli Not: RS(ReplicaSet), Deployment çoklu pod oluştururken önce rs objesi yaratır. Eğer değişim olursa bir rs daha yaratarak yönetimi kararlı hale getirir.

Test etmek için;

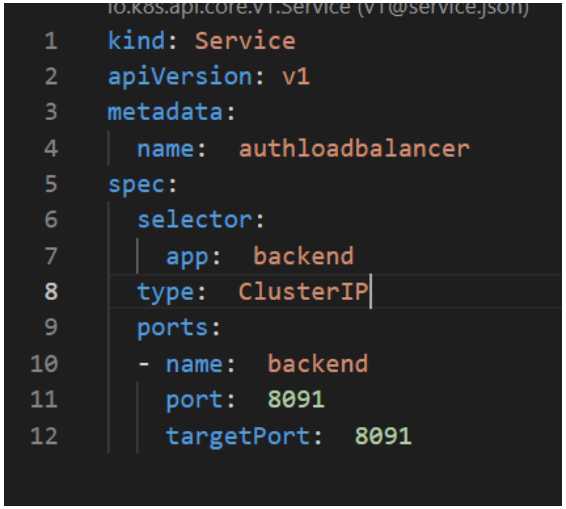
- Kubectl get deployment -w
- Kubectl get replicaset -w
- Kubectl get pods -o wide
- Watch kubectl get pods – Linux ta
- Watch kubectl get replicaset – Linux ta
- Değişliklikleri GeriAl -> kubectl rollout undo deployment authdeployment

- Service

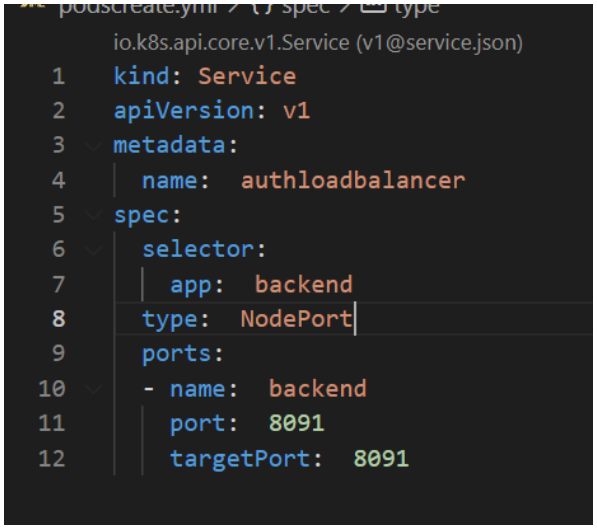
LoadBalancer mantığı için kullanılan bir yapıdır. Sorun farklı nodlarda yaratılan ve farklı bloklarda bulunan podları haberleşmesi için kullanılır. Ayrıca yatay genişlemenin mümün olmasını sağlar.

Types:

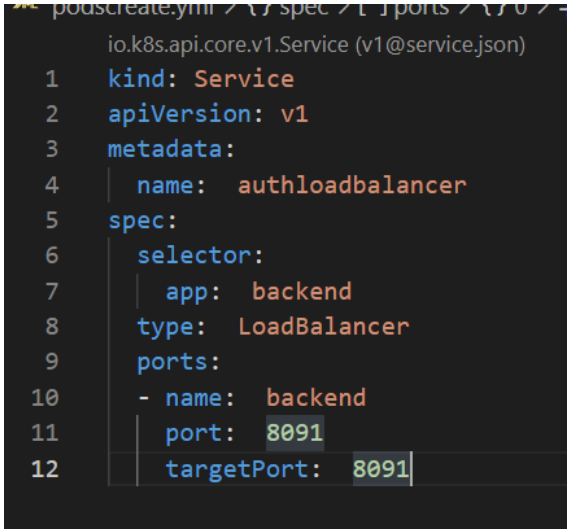
ClusterIP, localde çalışır. (Cluster-ip tabanlıdır)



NodePort, localde çalışır.(Cluster-ip tabanlıdır)



LoadBalancer, Cloude ta çalışır.



Burada Service test etmek için;

Herhangi bir pod a bağlanıp, servis e aip ip adresi üzerinde istek atabilirsiniz.

nslookup backend ile erişim sağlayabiliyor olmalısınız.

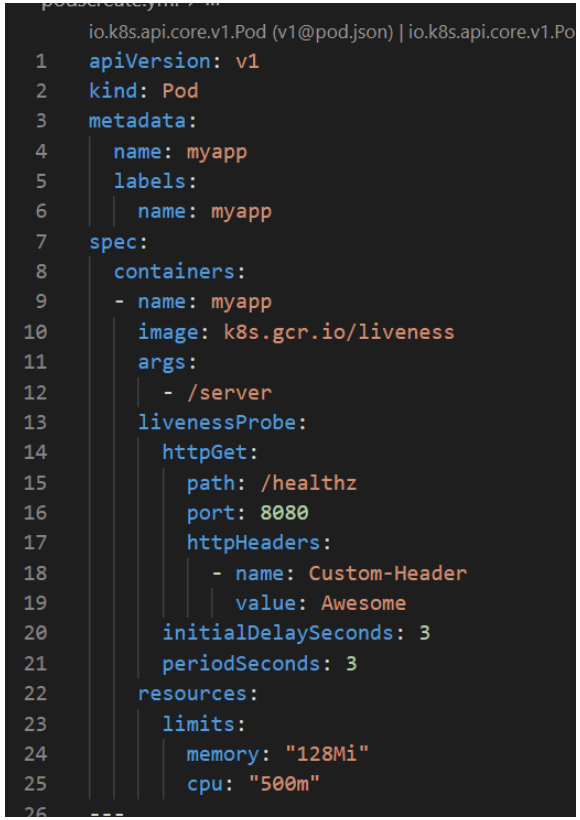
curl backend:8091

- Liveness probes

Sorun: pod larda sorun olmaya bilir, imajımız indirilmiş olabilir. Ancak uygulamamız çalışmıyor ise ne olacak? İşte burada, bizim servislerinizin ayakta olup olmadığını kontrol eden bir mekanizmadır.

Bizim uygulamalarımız resp-api olduğu için end pointlerimizin sürekli ayakta olması gereklidir. Bu nedenle bir end poine istek atarız ve bu seviş ayakta ise sorun yoktur. Ancak istemde bir sıkıntı var ise pod un tekrar restart edilmesi gereklidir. İşte bunu sağlamak için liveness probe kullanılır. httpGet ile

Diyelim ki yakta olması gereken bir DB niz var. Bunu kontrol etmek için belli bir portu dinlemeniz gerekli bunun için tcpSocket kullanırsınız.



- Resources

  Kaynakları sınırlı kullanmak için ihtiyacımızı giderir.

```
16          initialDelaySeconds: 15
17          periodSeconds: 20
18        resources:
19          limits:
20            memory: "128Mi"
21            cpu: "500m"
22    ---
```

- Environment Variable

İşletim sistemi, varible tanımlamak için kullanılır. Eğer env görmek istersen

  - Kubectl exec [podname] -- printenv

```
7   spec:
8     containers:
9     - name: myapp
10      image: <Image>
11      resources:
12        limits:
13          memory: "128Mi"
14          cpu: "500m"
15      ports:
16        - containerPort: 8080
17      env:
18        - name: MONGOURL
19          value: "192.168.1.25"
20        - name: MONGOPORT
21          value: "29851"
22
```

- Secret

Kubernates, güvenliğini sağlamamız gereken bilgileri saklamamız için secret objesini kullanımımıza sunar. Prolalar, OAuth token, ssh gibi bilgilerimizi depolar. Gizli bilgileri secret içinde saklamak pod tanımı içinde kullanmaktan daha güvenlidir.

Önemli NOT: oluşturacağımız secret ile atayacağımız pod lar aynı namespace içinde olmalıdır.

Secretleri görmek için;

- Kubectl get secrets
- Kubectl describe secrets mytoolssecret

podscreate.yml > { } spec

io.k8s.api.core.v1.Secret (v1@secret.json) | v1@secret+v1

```yaml
apiVersion: v1
kind: Secret
metadata:
  name:  mytoolssecret
type: Opaque
stringData:
  db_mongourl: "192.1.1.2"
  db_username: "admin"
  db_password: "$İffffreeeee"
---
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    name: myapp
spec:
  containers:
  - name: myapp
    image: <Image>
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
    ports:
      - containerPort: 80
    env:
      - name:  MONGOURL
        valueFrom:
          secretKeyRef:
            name:  mytoolssecret
            key:  db_mongourl
```

io.k8s.api.core.v1.Secret (v1@secret.json) | io.k8s.ap

```yaml
apiVersion: v1
kind: Secret
metadata:
  name:  mytoolssecret
type: Opaque
stringData:
  db_mongourl: "192.1.1.2"
  db_username: "admin"
  db_password: "$İffffreeeee"
---
```

- ConfigMap:

  Secret ile bire bir aynı işi yapan objedir. Bu obje key-value şeklinde değerleri tutar. Şimdi aklımda deli sorular, peki secret ile aynı işi yapıyorsa buna ne gerekvar? Aslında Secret verilerimizi BaseEncode şeklinde tutar ve ayar yaparsak etcd üzerinde auth edilerek kullanılır. Bizim gizli olmayan temel verilerimizi tutacak; Encode, Encript edilmeyecek; rahatlıkla erişebileceğimiz bir yapıyı configmap sunar.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: configmappod
spec:
  containers:
  - name: configmappod
    image: muhammedali55/XXXX
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
    env:
      - name:  MONGO_URL_POD
        valueFrom:
          configMapKeyRef:
            name:  bilgevonfigmap
            key: MONGO_URL
      - name:  MONGO_ADMIN_POD
        valueFrom:
          configMapKeyRef:
            name:  bilgevonfigmap
            key: MONGO_ADMIN
    ports:
      - containerPort: 80
```

v1@configmap+v1@pod.json | v1@configm

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: bilgevonfigmap
data:
  MONGO_URL: "localhost"
  MONGO_ADMIN: "root"
  MONGO_PASWORD: "root"
---
```

- Volume

  Senaryo: Diyelim ki bir servisler bütünümüz var ve bunlar çalıştıkları sürece log kayıtlarını tutuyorlar ve üzerinde işlem yaptıkları resim dosyalarını işleyip depoluyorlar. Böyle bir durumda, eğer bizim aktif çalışan pod umuz bir nedenle sorun yaşarsa container yeniden başlatılır yani mevcur pod silinir yerine yenisi create edilir. Böyle bir durumda depolanan tüm veriler silinir. İşte buna çözüm bulmak için volüme aktif olarak kullanılır.

  - *emptyDir: volüme ilk olarak bir pod oluşturulup bir node atandığında oluşturulur. Ve bu pod o node da çalıştığı sürece var olur. Bu birim başlangıçta boş olarak gelir. Aynı pod üzerinde var olan tüm container lar bu dosyaya okuma yazma yapabilirler. Eğer pod bir nedenle silinirse tüm veriler gider.(Çok tercih etmem)*
  - *hostPath: worker node ta var olan podlara dosya yada dizin bağlama imkanı tanır.(Bunu Hiç Tercih Etmem)*

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volumepath
  labels:
    name: volumepath
spec:
  containers:
  - name: volumacontainer
    image: muhammedali55/tinyimagevolume
    volumeMounts:
      - name:  directory-volume
        mountPath:  /logs
      - name: file-volume
        mountPath: /logs/alllogs.log
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
  volumes:
  - name:  directory-volume  # Zaten container üzerinde var olan bir dizini belirtir.
    hostPath:
      path: /logs
      type: Directory
  - name:  dircreative-volume # bir dizin oluştur ancak eğer bu dizin zaten container üzerinde var ise oluşturma
    hostPath:
      path: /temp
      type: DirectoryOrCreate
  - name: file-volume  # Eğer alllogs.log dosyası yok ise oluştur diyoruz.
    hostPath:
      path: /logs/alllogs.log
      type: FileOrCreate

```

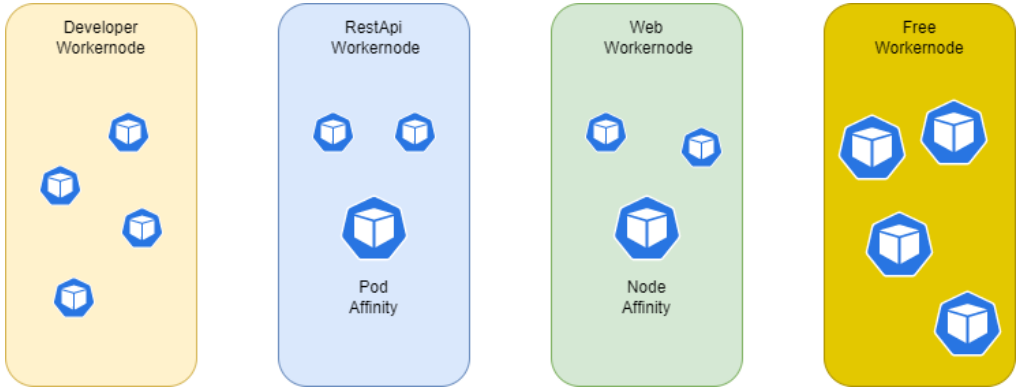
ÖNEMLİ: Volume bizim için önemli ancak şuan anlattıklarım bizim ihtiyaçlarımıza karşılık gelmemekte. Bize daha çok bağımsız çalışabilen volüme ler gerekli olnarı ayrıca işleyeceğiz.

- Node Affinity:
  Node lara etiket atayarak oluşturacağınız podların etiketli node lar üzerinde çalışmasını sağlayabilirsiniz.

- Pod Affinity:
  Podların çalışacağın nod gruplarını belirleyebiliriz.

  NOT: Yukarıda kısaca geçtiğim yöntemleri çok karmaşık ve uğraştırıcı olduğu için pek tercih etmiyorum. Zaten her zaman bu kullanımlar isteklerimize yanıt vermiyor. Bu nedenle örneklerini ilgili dosyamıza ekledim.

- Taint ve Toleration:
  Sorunu Tanımlayalım; elimizde 3 ve daha fazla workernode olsun(developernode – restapinode – webnode - freenode), gerekli tanımları yaparak rest api olanlar ilgili node ta, web olanlar ilgili node ta ayağa kalksın. Herhangi bir etikete sahip olmayan(nodeaffinity ya da podaffinity olmayan) podları rast gelen nodlar üzerinde ayağa kaldırılacak ve oluşturmak istediğimiz yapı kurgulanamayacaktır.



İşte tam burada, bize gerekli olan node lar üzerinde koşturulacak podların node tarafından belirlenmesi ve koşullarına uymayan pod yapılarını reddetmesidir. Bu sağlamak için taint ve toleration kullanılır.

Öncelikle sistemimizde çalışan workernode lara bir göz atalım

- Kubectl describe nodes minikube ya da docker-desktop

Peki bir node üzerine nasıl ekleme yapılabilir?

- Kubectl taint node minikube ya da docker-desktop developerteam=bilgeadam:NoSchedule
- Kubectl taint node minikube ya da docker-desktop developerteam-

DİKKAT: burada yazdığım minikube ya da docker-desktop ifadeleri bizim kullandığımız context için. Eğer butta bir node tanımınız var ise bu işlemi o node isimleri kullanrak yapmalısınız.

Oluşturma işleminden sonra, bunu denemek ve test etmek için sisteminizde ki podları izlemeye başlayın

- Kubectl get pods -w

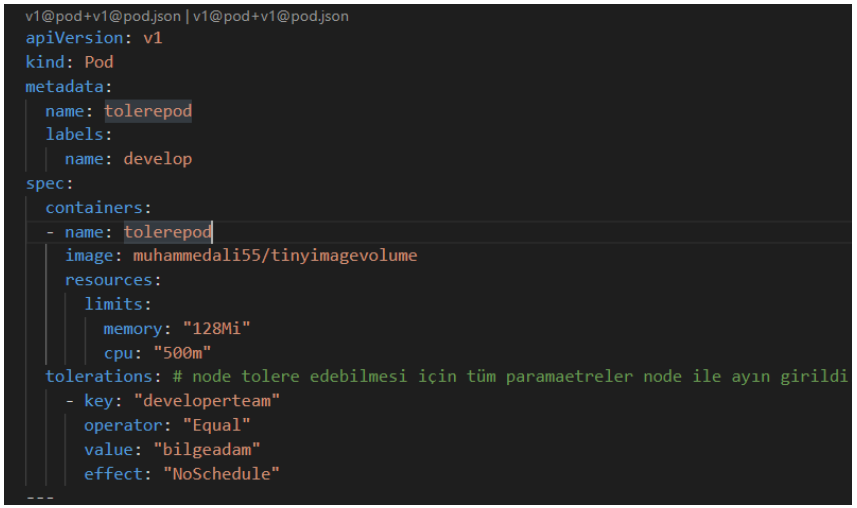
Ardından yeni bir pod oluşturmayı deneyin.

- Kubectl run deneme –image=ngnix –restart=Never

Bu işlemden sonra podları gözlemleyin. Bu işlemde pod üzerinde bir taint eklemediğim için node bu pod u tolere edemedi ve pod “pending” te takılı kaldı.

- Kubectl describe pod deneme

Yukarıdaki komutu çalıştırıp detaylara bakarsanız sorunu burada görebilirsiniz.



ÇOOK ÖEMLİ: Eğer bir pod üzerinde çalışamayacağı bir node üzerinde oluşturulursa orada kalmaya devam eder. Ne çalışır ne de silinir. İşte burada tolere edilmeyen pod ların silinmesi gereklidir bu nedenle taint yazarken şunu yazmak daha mantıklıdır

- Kubectl taint node minikube developerteam=bilgeadam:NoExecute

- ## DaemonSet:

  Sorun: diyelimki bir 100 worker node tan oluşan bir kubernate cluster yapınız var. Ve bunların tamamı üzerinde çalışması gereken bir uygulama yapımız olsun. Amacımız uygulamamızda toplayacağımız hata, işlem v.s. loğlarımızı merkezi bir yerde toplamak istiyoruz. Şimdi tüm node larda çalışacak ve topladığı log kayıtlarını aktaracak bir uygulamamız olsun.

  Burada sorunu şöyle çözemeyi deneyebiliriz. Uygulamamızı tüm node lara tektek bağlanarak kurulum yapabiliriz.

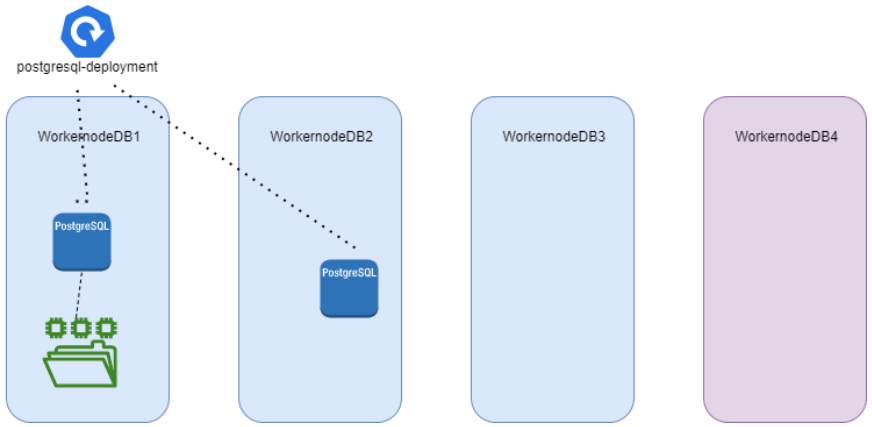
  Ya da uygulamamı container haline getirip 100 adet yml dosyayı oluşturarak yükleyebilirim. 😊

  İşte tüm bu sorunların ortasında, daemonset objesi, bize yönetilebilir bir node yönetimi sunar. Böyleye sisteme node eklendikçe üzerinde çalıştırılacak podlarda eklenir.

  Not: Şu aşamada sizler için gerekli bir obje değil bu nedenle örnek bir yapıyı ekledim.

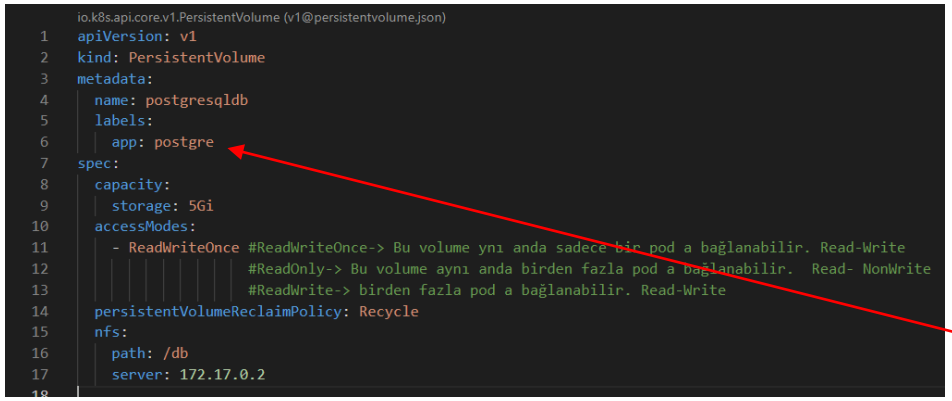
- ## Persistent Volume

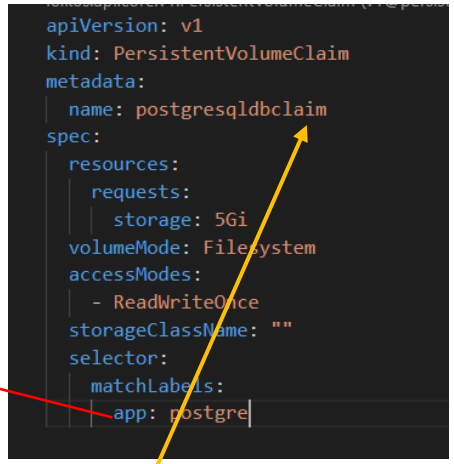
  Sorun: Diyelim ki ölçeklenebilir bir sistem ile postgresql gibi bir Veritabanını farklı node lar üzerinde çalıştırmak isteyelim. Eğer bir node üzerinde çalışan uygulamada eğer workernode kapanırsa , deployment objesi uygun bir node bularak bu uygulamayı farklı bir node üzeride çalıştırır. Ancak burada sorun şu tüm datalar diğer node üzerinde kaldı ☹

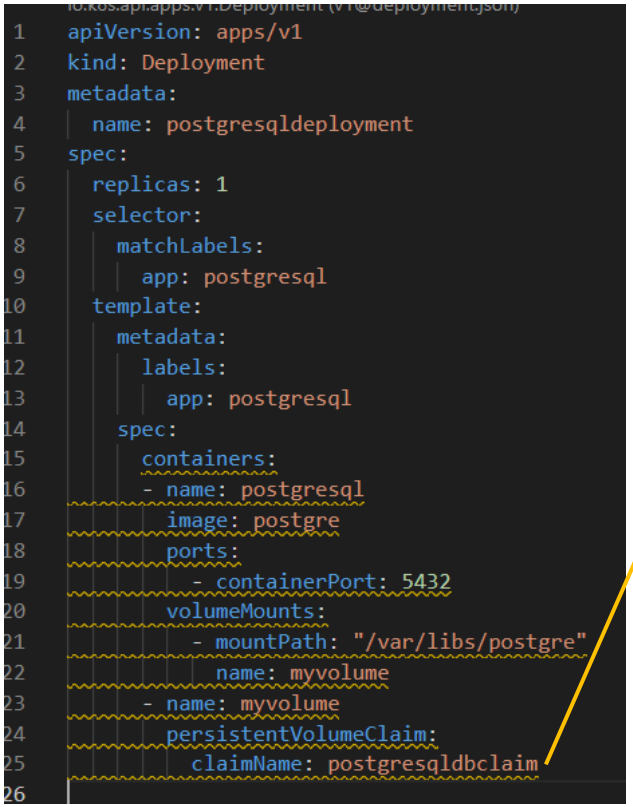


NOT: Kubernate hali hazırda üzerinde bir çok bulut sistemde var olan dosya sistemleri ile uyumlu şekilde çalışabilecek driver ları içerir. Ancak eğer 3. Parti bir yazılım ile işlemler yapacaksanız bu durumda Container Storage Interface(CSI) kullanarak işlemlerinizi yapabilirsiniz.

İlk olarak bir Persistance Volume yaratmamız gereklidir. Data sonra bunu podlara bağlayabilmek için PersistentColumeClaim obje si oluşturmalıyız.







- `docker volume create nfsvol`
- `docker network create --driver=bridge --subnet=10.255.255.0/24 --ip-range=10.255.255.0/24 --gateway=10.255.255.10 nfsnet`
- `docker run -dit --privileged --restart unless-stopped -e SHARED_DIRECTORY=/data -v nfsvol:/data --network nfsnet -p 2049:2049 --name nfssrv muhammedali/basicnfs`

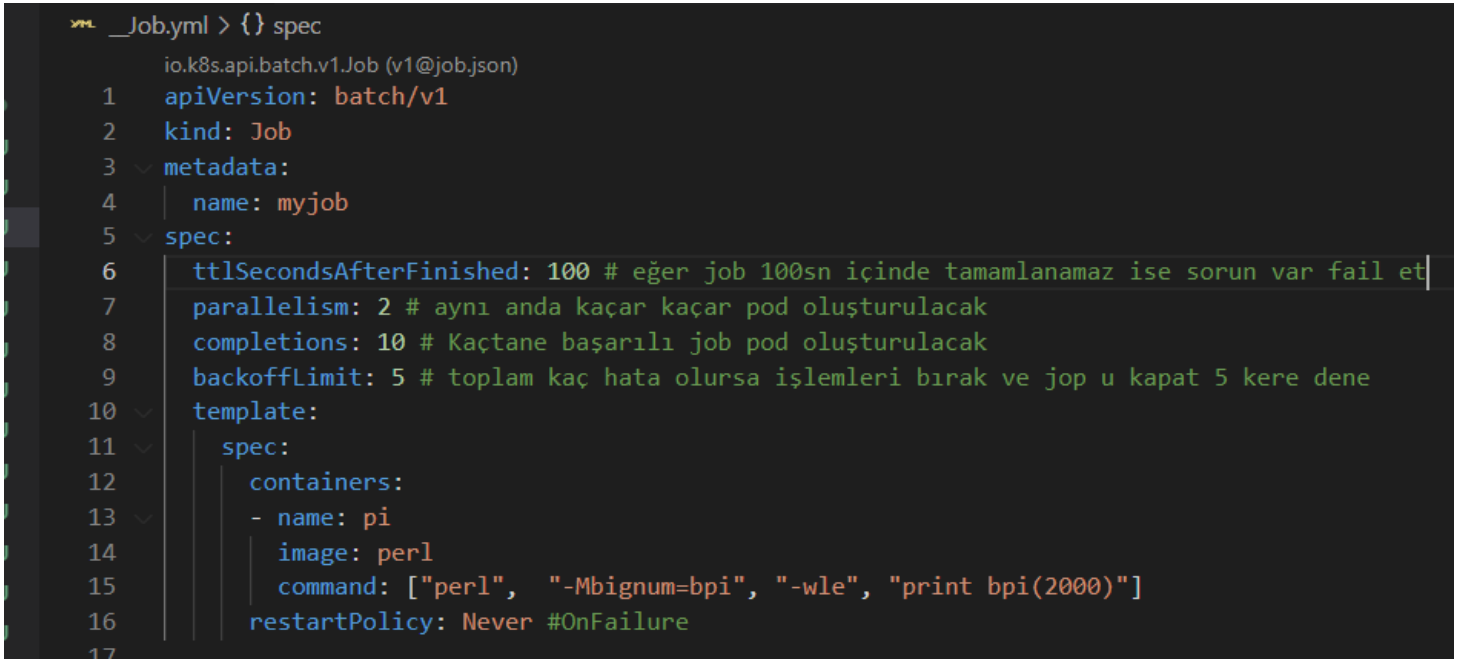
Yukarıdaki ayarları kullanarak, minikube üzerinde bir NFS ayağa kaldırabilir ve bunun üzerinden volüme işlemlerini yapabilirsiniz. İşlemler karmaşık ve sıkıntılı olduğu için bu kısımları geçiyoruz. Genellikle developer olarak bu kısımlar bizi çokta ilgilendirmiyor.

- StatefulSet:

Sorun: düşününki deployment objesi ile podlar oluşturuyorsunuz. Sisteminiz genişletmek istediğiniz de replikaları arttırarak sistemin genişlemesini bağlayabilirsiniz. Burada deployment replika setlerini rast gele genişletir ve siler. İşte tam burada NoSql çözümlerinin genişlemelerinde sorun yaşanabilir. Nasıl yani? Bir nosql çözümünde master -> slave + slave + slave şeklinde çalışır. Biz bir pod u master yaptık diyelim, diğerlerini secont olsun. Bu durumda bizim sistemimiz önce ihtiyaçtan büyüdü ve 5 pod daha ekledik, sonra sistem ihtiyaç duymayacak hale geldi ve pod sayısını eski haline getirmek istedik. Ancak deployment pod ları silerken master pod u da sildi sistem patladı ☹

- Job:

Sistemde podların oluşturulması ile ilgili işlemleri deployment lar ile yapabiliriz. Burada sıkıntı şu eğer bizim uygulamamız bir sorunla karşılaştı ve çakıldı. Bu durumda deployment objesi sorunu algılayıp tekrar pod u ayağa kaldırır. Ancak bizim bazen işlemlerini yapıp kapanması gereken podlarımız olabilir, bu durumda sıkıntı çıkacaktır. İste burada job lar bu işi bizim için yaparlar.



Sistemi izlemek için

- Kubectl get pods -w
- Kubectl get jobs -w

EVEEEEET peki job işlemlerinin belli bir zaman diliminde çalışmasını istese idik ne yapackatık?

- CronJob

İşlerin belli bir zaman diliminde yapılmasını sağlamak için kullanılır.

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
#  * * * * *
#  | | | | |
#  | | | | |
#  | | | | |
#  | | | | -------- Haftanın günleri (0 - 6)
#  | | | ---------- Ay (1 - 12)
#  | | ------------ Ay ın günü (1 - 31)
#  | -------------- Saat (0 - 23)
#  ---------------- Dakika (0 - 59)
# https://crontab.guru/ adresinsen oluşturmak istediğiniz zamanı
# deneye bilirsiniz.
#
# Örn: 1 * * * *  -> 1 dakika sonra
# Örn: 1 * 5 * * -> Ayın 5. günü 1. dakika da 2022-03-05 00:01:00
# Örn: */1 * * * * -> Her 1 dakika da bir çalış
#
```