kubernetes

Kubernetes Komponentleri:

Kube-apiserver: kube-apiserver ,kubernetes API’nı orataya çıkaran kubernetes control plane’in en önemli bileşeni ve giriş noktasıdır.Tüm diğer komponentler ve node bileşenleri direkt iletişim kurabildiği tek komponenttir . tüm iletişim apiserver üzerinden gerçekleşir. Kube-apiserver kubernetes’de kaynak oluşturma isteklerinin api doğrulamasından sorumludur.

Etcd:etcd tüm cluster verisi,metadata bilgileri ve kubernetes’de oluşturulan tüm objelerin bilgilerinin tutulduğu anahtar-değer”key-value” veri deposudur.kısac kubernetes’in çalışması için tüm veriler etcd üzerinde tutulur.Api server hariç diğer kubernetes komponentleri etc ile direkt haberleşemezler.

Kube-scheduler:yeni oluşturulan ya da bir node ataması yapılmamış Pod’ları izler ve üzerinde çalışacakları bir node seçer.

Kube-controller-manager : Mantıksal olarak ,her controller ayrı bir süreçtir,ancak karmaşıklığı azaltmak için hepsi tek bir binary olarak derlenmiştir ve tek bir process olarak çalışır.  
bu controllerların bazıları şunlardır;  
  
 Node controller   
 Job controller   
 Service Account & token controller   
 Endpoints controlller

Container runtime: containerları çalıştırmaktan sorumlu olan yazılımdır.

Kubernetes birkaç container runtime destekler:docker,containerd,crı-o

Kubelet: cluster’daki her node çalışan bir agent’tır Pod içerisinde tanımlanan containerların çalıştırılmasını sağlar.  
  
Kubelet,çeşitli mekanizmalar aracılığıyla sağlanan bir dizi pod tanımını alır ve bu pod tanımında belirtilen containerların çalışır durumda ve sağlıklı olmasını sağlar .

Kube-proxy: nodeların üstünde ağ kurallarını yönetir.Bu ağ kuralları ,cluster’ın içerisindeki veya dışındaki ağ durumlarında pod’larınızla ağ ileşimine izin verir.

Kubernetes yayın döngüsü :

Semantic versioning:

1.21.0 Patch

Majör Minör

Kubernetes senede 3 minor version yayınlar.her minör versiyon yayınlandığı tarihten itibaren 12 ay boyunca desteklenir.

Kısaca 12 ay boyunca her ay patch versiyonları yayınlanarak hatalar ve güvenlik açıkları giderilir.

Play with kubernetes:

1.master node ayağa kaldır.



2.worker nodeleri ayağa kaldır.

Kubeadm join 192.168.0.13:6443 –token xxxxxx

Kodu al ve diğer workerlara yapıştır.

3. kubectl kullanabilmek bazı ayarlamarlı yapıyor master node’da

Master node üzerinde ;  
mkdir –p $home/.kube   
sudo(sudo yazmaya gerek yok ) cp –i ….   
sudo chown ….   
  
kısımlarını alıyoruz ve yine master node üzerinde çalıştırıyoruz  
  
4. Son olarak 2. Adımdanki initilize cluster network komutunu alıyoruz ve çalıştırıyoruz.

Kuberbetes-1

Kubectl Config : kubectl kübernetes clusterlar ile iletişim kurmamızı sağlar.

* Kubectl aracı bağlanacağı kubernetes cluster bilgilerini config dosyaları aracılığıyla erişir.
* Config dosyasının içerinde kubernetes cluster bağlantı bilgilerini ver oraya bağlanırken kullanmak istediğimiz kullanıcıları belirtiriz.
* Daha sonra bu bağlantı bilgileri ve kullanıcları ve ek olarak namespace bilgilerini de oluşturarak context ‘ler yaratırız.
* Kubectl varsayıla olarak $HOME/.kube altındaki config isimli dosyaya bakar.
* Kubectl varsayılan olarak $HOME /.kube altındakşi config dosyasına bakar ama bunu KUBECONFIG enviroment variable değerini değiştirerek güncelleyebiliriz.

kubectl config

kubectl config get-contexts : kubetctl baktığı config dosaysındaki mevcut contextleri listeler.  
  
kubectl config current-context: şuanda kubectl ‘de tanımlı current context’in ne olduğunu geri dondurur.

kubectl config use-context: başka bir sunucu üzerinde işlem yapmak istiyorsak sunucuyu değiştirmemiz gerekir bunu için kullanılır.  
   
  
  
**Kubectl Kullanımı :**

kubectl + enter -> genel bilgi   
  
kubectl cluster-info -> üzerinde işlem yaptığımız kubernetes cluster üzerinde temel bilgi edinmek isteriyorsak giriyoruz.

Kubectl –help  
  
kubectl + eylem + hangi tip obje üzerinde uygulama isityorsun (kubectl get deployments)

Kubectl delete pods testpods (tek bir obje üzerine işlem yapmak istiyorsak isim belirtiyoruz)

Kubectl get |delete|edit |apply

Kubectl get pods burada namespace olark default üzerindeki podsları getirdirir ancak kubectl get pods –n kube-sytem dersek o isim üzerinde çalışan podları veriri yani kısaca –n ile isime göre arttık.  
  
tüm namespace üzerindeki podsları görmek istersek;  
kubectl get pods –all yada –A

Kubectl get pods -A –o wide -> geniş çıktı olark verir

Kubectl get pods -A –o yaml -> yaml çıktı

Kubectl get pods -A –o json -> json çıktı

-o json/ jspnpath /go-path /go-template /yaml /ysjson /wide /custom-colums

**Explain**: komut değil objerlin ne olduğunu almamıza yardım eder. Ayrıca version bilgilerinde bize verir.

Kubectl explain pod

**Kubernetes objeleri**

**Kubernetes üzerinde container imajı haline getirdiğimiz iş yüklerini koşturmak istiyoruz.Yani kubernetes üzerine birşeler diploy etmek istiyoruz. Kubernetes üzerinde çalıştırdığımız ,koşturduğumuz şeylere obje denir. Kubernetes üzerinde herşey api objesidir. Bizler koşturmak istediğimiz iş yüklerini bu objeler şeklinde oluşturap ayarlarız.**

**Pod:** podlar kubernetes’te oluşturabilceğimiz ve yönetebileceğimiz en küçük birimlerdir.

Pod’lar bi ya da daha fazla container barındırabilir. Ama çoğu durumda poflar tek container barındırır.Her pod’un bir id’si “uid” bulunur. Her pod eşsiz bir ip adresine sahiptir.

Aynı pod içirişindeki containerlar aynı node üstünde çalıştırılır ve bu containerlar birbiriyle localhost üsütnden haberleşir.

Kubectl run firstpod –image=nginx –restart=Never

Kubectl describe pods firstpod (pod hakkıda daha çok bilgi verir.)

kubectl logs firstpod -> pod loglarını verir.

Kubectl logs –f firstpod -> canlı log

Kubectl exec firstpod -- hostname -> pod içerisinde komut çalıştırmaya yarar.

Kubectl exec –it firstpod -- /bin/sh -> pod’a bağlandık.

kubectl delete pods firstpod -> pod’u sildik

**Pod Yaml :**

apiVersion:

kind: // ne oluşturmak istiyorsak onu yazıyor (pod,servis)

metadata: // obje ile ilgili uniq bilgiler (label,name,namespace)

spec:// obje özellikleri her obje tipine göre değişir(pod oluşturken container bilgileri servis oluştururken servis obesji özellelikleri)

apiVersion: v1

kind: Pod

metadata:

name:firstpod

labels:

app:front-end

spec:

containers:

-name:nginx

İmage:nginx:latest

Ports:

-containerPort:80

Not: yaml dosyası ile aynı yerde olmamız gerekli

Kubectl apply –f pod1.yaml

Kubectl describe pods firstpod(detaya baktık)

Kubecttl run secondpod –image=nginx –port=80 --labels=”app=frontend” --resart =never -> bu şekilde de yapılabilir

Öyle çok uzun(imparatif) eksstra yeni bir şey eklediğmizde podu sil tekrar oluştur vs yapmaız lazım deklaratif(dosya üzerinede oluşturma) yöntem en doğru yöntem.

Kubectl edit pods firstpod -> bu yöntem ile kubernetes classtırdan bu objenin mevcut halini yaml formatında çekere ve shall’imizde de var sayılan olarak text editöründe açar bu şekilde de ekleme yapılabilir. Dosya kayıt edilir ve kubernetes bunu günceller.(çok sık kullanılmaz.)

**Pod Yaşam Döngüsü :**

Crashloopbackoff-> container sürekli resart ediyor. Kubernetes önlem alıyor belirli sn bekliyor tekrar resart etme çalışıyor 5 dk kadar.

İmagepullbackoff-> senin istediğin imajı bulamdım bu yüzden container oluşturamadım.

Kubectl get pods –w -> izlemek anlamında herhangi bir değişiklik olduğunda gösterir.

**Çoklu Container Pod:**

Ana uygulamaya bağımlı ve onunla network seviyesinde izolasyon olmadan ve gerektiği durumda aynı depolama alt yapsını kullanacak uygulamları pod içerisinde ikinci bir container olarak çalıştırma imkanı sağlıyor . birlikte skeyle etmesi gereken birbileri ile network ve stroage seviyesinde erişmesi gerekn uygulamaları aynı pod içerinde ayrı ayrı containeler olarak çalıştırabiliyoruz. Mesela loglama yazılımlar yada ana yazılımların ihtiyaç duyacağı dosyları güncelleyen ve onlar üzerinde işlem yapan yazılımları bu şekilde deploy ediyoruz

Bir pod içeridne birden fazla container çalıştırdığımızda kubernetes bize şunun garantisini verir;  
1) aynı pod içerinde tanımlanmış containeler aynı worker nod üzerinde oluşturuluyorlar

2)containerlar ayrı ayrı birer ünitedir fakat pod’un lifescale içerinde yönetilir. Yani pod oluşturulduğunda iki container birden oluşturulur ve silinirsede birlikte silinir.

3) bu containerlar arasında network izolasyonu bulunmaz yani aynı pod içerisind çalışan containerlar localhost üzerinden ulaşabilirler bunlar network bakımında sanki aynı makinede çalışan processlerdir.

4) Tek bir volume oluşturularak her iki containerla mount edilebilir böylelikle aynı dosylar üzeinde çalışabilirsiniz

Kubectl exec -it multicontainer -c webcontainer -- /bin/sh

**Inıt Container :**Pod içerimde birden fazla container oluşturmaya olanak verir.App containerlardan farklı olarak init containerlar pod’un yaşam döngüsü boyunca çalışmaz pod oluştuğunda ilk olarak çalışır . içindeki uygulama ne yapıcaksa onu yapar ardından app container oluşur init containerlar esas uygulama çalışmadan önce tamamlanması gerek gereken şeyler olduğundan ve bunlar tanmamlanamdan uygulamyı çalıştırmak mantıksız olduğunda kullanılır.

Watch –n 2 kubectl describe pod initcontinerpod

Kubectl logs –f initcontainerpod –c initcontainer

**Label ve selector :** etiket ve etiket seçme**.**

Tier:front-end

Value”değer”

Key:”anahatar”

Etiketler obje oluşturma adımda yada oluşturulduktan sonra atanabilir.  
  
kubernetesde bağlantı etiketler sayesinde kurulur. Aynı zamanda objerli gruplama imkanda verir.

Example.com/tier:front-end

Prefix”opsiyonel” Anahtar Deger

* Prefix “önek” kısmı opsiyoneldir.Zorunlu değildir.
* Kubernetes.io/vek8s.io/önekleri ,kubernetes çekirdek bileşenleri için ayrılmışıtır .Bu bileşenler tarafından kullanılır.
* Anahtar ve değer 63 karakter veya daha az olmalıdır(boş değer olabilir)
* Alfenumerik bir karakterler([a-z 0-9 A-Z]) başlamalı ve bitmelidir.
* Tire(-),alt çizig(\_),noktalar(.) ve arasında alfanumerik değerler içerbilir.

Equalty selector kullanımı :

App isimli key’in atamış olan podları listeler;  
kubectl get pods –l “app” –show-labels   
  
-l -> laberlara göre ayrıştır.

App anahatarına firstapp atamş lablerlları listler->

Kubectl get pods –l “app=firstapp” –Show-labels

Birden fazla anahtar üzerinde listeleme ->

Kubectl get pods –l “app=firstapp,tier=front-end” –show-labels

App=firstapp olsun ama tier belirtilen değere eşit olmasın ; Kubectl get pods –l “app=firstapp,tier!=front-end” –show-labels

App anahtarı olsun tier=front-end olsun ;

Kubectl get pods –l “app,tier=front-end” –show-labels

set base selector kullanımı ;

kubectl get pods –l ‘app in (firstapp)’ –show-labels

kubectl get pods –l ‘app in (firstapp,secondapp)’ –show-labels -> normal de bu komut equalty ‘ de çalışmaz çünkü app değeri first,secondapp değerinin ikisinde eşit olması beklenir ama böyle bişey olmyacağı için olmaz ancak set base de o yada diğeri atanmış anlamında olacağı için çalışır.

Not: equalty eşitlik bazında çalışır set base ise atanmış atanmamış sorugusu yapar.

kubectl get pods –l ‘app notin (firstapp)’ –show-labels -> eşit değil anlamında

kubectl get pods –l ‘app ,app notin (firstapp)’ –show-labels-> app anahtarına sahip olacak ve app anahtarı firstapp atamamış olacak.

Kubectl get pods –l ‘!app’ –show-labels -> podlar içersinde app anahtarına eşit olmayanları listelesin  
  
 kubectl get pods –l “app in (firstapp),tier notin (frontend)” –show-labels

kubectl get pods --show-labels -> tüm podların labellarını listler

labellar sonradan ekleme ve çıkarma ;

ekleme;

kubectl label pods pod9 app=thirdapp   
  
silme ;

kubectl label pods pod9 app-   
(sondaki – değeri anahtarı ve değeri sil anlamında.)

güncelleme;  
 kubectl label –overwrite pods pod9 team=team3  
(team -> team3 yaptık )

bir namespacede tüm podlara toplu halde label ekleme;

kubectl label pods --all foo=bar   
foo bar labelını tüm podlara ekledi.

objeler arasındaa labellar üzerinde ilişki kurma :

nodeselector tanımı bize şu imkanı verir normla şartlar

altında kubectl kendi algortmasına göre seçer ama bizler nodeselector ile bu podun çalışmasını istediğmiz podları belirtebiliriz buna labellar ile yaparız.

Nodeselector:

Hdddtype:ssd -> bu şu anlama geliyor kubescl bu podu sceli edilirken hddtype=ssd olan bir bode bul ve bu podu orada çalıştır. Bu sayede pod objesi ile node obejisini birleştir.

Bir tane yaml içinde 10 tane pod oluşturduğumuz için tek seferde tamamını silebilirz-> kubectl delete –f podlabel.yaml

**Annotation:**

Example.com/notification-email:admin@k8fundamentals.com

Prefix “opsiyonel” Anahtar değer

* Objeye onu tanımlayan fakat label olarak eklememize sakıncalı olacağı bilgileri annotation “açıklama”olarak ekleriz
* Prefix “önek” kısmı opsiyoneldir.Zorunlu değildir.
* Anahttar alanı 63 karakter veya daha az olmalıdır
* Tire(-),alt çizig(\_),noktalar(.) ve arasında alfanumerik değerler içerbilir.
* Değer alanında bu kurallar geçerli değildir. Alfanümerik olmayan karakter de alabilir.

İperatif olarak ekleyebilirz:

Kubectl annotate pods annotationpod foo=bar -> ekledik

Kubectl annotate pods annotationpod foo- ->sildik

**Namespace:**

Namespaceler adalar için bir kapsam sağlar. Kaynak adlarının bir namespace içinde benzersiz olaması gerekir.Namespace biribirinin içine yerleşemez ve her kubernetes kaynağı yalnızca bir namespace içinde olabilir.

Namespaceler cluster kaynaklarını birden çok kullanıcı arasında bölmenin bir yoludur.  
varsayılan olarak 4 namespace oluşturulur;

Kubesystem-> kubernetes tarafında oluşan objelerin tutulduğu namespacedir.

Kubepublic->kimliği doğrulanmamışlarda dahil tüm kullanıcıların erişime ihitiyaç duyabileceği objelerin oluşturulabilceği yerdir.  
  
kubenotliz-> North hard with işlemleri için gereken özel bir namespacedir

Default -> bizler aksini belirtmediğimiz sürece obejler burada oluşur.

Not: kubernetes cluster -> file server namespace-> oluşturduğumuz klasör gibi düşün

kubectl get pods --namespace kube-system

kubectl get pods --all-namespaces -> listelencek tüm komutların tüm namespaceleri kapsayacak şekilde çalıştırılmasına imkan verir.

Kubectl create namespace app1 -> yeni namespace oluşturduk.

Obejelerin hangi namespacede komuşlanacağını namespace tanımı ile belirleriz. Eğer namespace tanımına bir değer girmezsek objeler default namespace üzerinde oluşur.

Kubectl get pods –n development -> development üzerinde oluşan podları listeler.

Namespace içerisindeki objelerle iletişim kurarken objenin namespacesinde belirtmemiz gererkiyor. Yani obje ile iletşim kuracaksak namespacesinde belirtmemiz gerekiyor.

Kubectl exec –it namespace –n development --/bin/sh

Büyük bir cluter altında bize ait bir namespace var ve biz sadece bu namespace ‘de çalışıyoruz her seferinde namespace belirtmen config ayarı ile yapabiliriz.

Kubectl config set-context --current --namespace=development -> bu şekilde var sayılan namespace development oldu   
  
tekrar defaulta çevirme ;  
Kubectl config set-context --current --namespace=default

Namespace silme ; Not : o namespace silidiğine tüm obejelerde silinir dikkatli ol .(eminmisin vs diye sormaz dirket siler.)  
  
  
kubectl delete namespace development

**Deployment:** deployment bir veya birden fazla podu bizim belirlediğimiz dizay state gore oluşturan ve sonrasında bu dizay state’i (istenilen durumu) mevcut duruma sürekli kaşılaştıran gerekli düzeltmeleri yapan obje tipidir. Biz bir deployment objesi oluşturmak için tanım yapar ve bu tanımın içinde oluşturmak istediğimiz podun hangi özelliklere sahip olacağını ve kaç adet oluşturacağımızı belirtiriz bu obje oluşturulduğunda belirtilen adette pod oluşturulur.

Ör:

Kubectl create deployment firstdeployment –image=nginx:latest --replicas=2(kaç tane pod oluşturacağını belirttik.)

Ör:diyelim ki biz uygulamamızın yeni versiyonunu çıkarttık ve bu imajla yeni podlar oluşturmak ve mevcut deplomentları güncellemek istiyoruz.

Kubectl set image deployment/firstdeployment nginx=httpd

Mevcut nginx imajını httpd imajı ile güncelledik.

Deployment ile pod sayısınıda çok rahat bir şekilde ayarlayabilriz;

kubectl scale deployment firstdeployment –replicas=5

arttırabilir Yada azaltabiliriz

kubectl scale deployment firstdeployment –replicas=2

deployment silme;  
kubectl delete deployment firstdeployment

bu şekilde imparetif yöntemle oluştura bilirz ama biz deplaratif yöntemle yaml ile oluşturacağız.

apiVersion: apps/v1

kind: Deployment

metadata:

  name: firstdeployment

  labels:

    team: development

spec:

  replicas: 3 // kaç anahtar oluşturmas gerektiğini söylüyoruz.

  selector:

    matchLabels:

      app: frontend

  template: // pod özellikleri

Doğru bir yaml dosyası :

apiVersion: apps/v1

kind: Deployment

metadata:

  name: firstdeployment // deployment adın

  labels:

    team: development

spec:

  replicas: 3 //tane pod

  selector:

    matchLabels: //senin yönteceğin podlar app :frontend labelına sahipolucak

      app: frontend

  template: // podlar özelliklerde olsun

   metadata:

    labels:

      app: frontend

  spec:

    containers:

    - name: nginx

      image: nginx:latest

      ports:

      - containerPort: 80

Label ve selector alanı çok önemli ;  
1- mecburi alan (deploymenta en az 1 tane selector tanımı olmalı ve aynı labellar templlate kısmındada bulunmalı. Yoksa deployment yaml dosyanızı deploy edemezsiniz.)

2- Eğer birden fazla deployment objesi oluşturacaksanız ki oluşturacağız her deployment objesinde farklı label ve selector kullanamız gerekecek .

3- Aynı labellerı kullandığımız başka objeler oluşturursak sıkıntı çıkartır.

**ReplicaSet:**

Bir Replicaset’in amacı,herhangi bir zamada çalışan kararlı bir replika pod setini sürdürmektir.Bu nedenle genellikle belirli sayıda özdeş pod’un kullanılabilirliği garanti etmek için kullanılır.

Özetle bir bir deployemnt objesi oluşturduğumuzda bu obje kendi yönettiği bir replicaset objesi oluşturur ve bu replicaset objeside podları oluşturur ve yönetir. Biz deployment tanımında bir değişiklik yaparsak örneğin imajı güncellersek deployment bu yeni tanımla yeni bir replicaset objesi daha oluşturur.   
ilk oluştuduğu replicaset objesi yavaş yava ş kendi oluşturduğu podları silmeye başlar. Ve yeni replica sette yeni podları oluşturur. Burada silme ve oluşturma işlemlerini sırasını biz belirleyebiliriz.

Kubectl get replicaset

Kubectl set image deployment/firstdeployment nginx=http ->nginx isimli containerın imajını güncelledik yeni bir replica set oluştu

Son yaptığımız işlemi geri almak istiyorsak;  
kubectl rollout undo deployment firstdeployment

Biz deployment oluştururuz replica set oluşturmayız(istersek oluştururz)bunu yaparsak geçişleride manuel yapmamız gerekir bu yüzden biz sadece deployment üzerinde çalışırız o gerekeni yapar.  
ayrıca replica set üzerinde bir değişlik yaptığımıda replica set o değişikliği mevcut podlara uygulamaz. Ancak yeni bir pod oluşturacaksa bu yeni değişiliği dikkate alarak oluşturur.

Kubectle rs firstrs -> replicaseti sildik

**Rollout ve RollBack:**

 strategy:

    type: Recreate

bizler bu deployment güncellediğmizde rollout işlemlerin nasıl yapılacağını belirleriz iki tip rollout stratejisi vardı;

1-Recreate -> ben deployemntta değişiklik yaparsam öncelikle tüm podları sil ve bu işlemem tamamlandıktan sonra yeniden oluştur.(ör eski ve yeni versionların birlikte çalışması sakıncalıysa ) her zaman kullanmayız geçiş sırasında kısa süreli kesinti oluşabilir.

2-RollingUpdate

strategy:

    type: RollingUpdate

    rollingUpdate:

      maxUnavailable: 2

      maxSurge: 2

varsayılan stratejidir. Recreate tam tersidir. Ben bir değişiklik yaptığımda direkt siliip yeniden oluşturma bu işlemi aşamalı olarak yap. Bu aşamaları belirleyeceğimiz iki opsiyon mevcuttur ;

maxUnavailable -> ben bir deploymantta değişiklik yaptığımda burada belirtiğin kadarı podu sil . % değeride girilebilir.

maxSurge-> Bu geçiş sırasında toplam pod sayısının en fazla kaç olabileceği belirler.

Ayrıca biz bu ayarlamaları yapmadığımız zamanda varsayılan olarak %25 set edilir.

Kubectl apply –f deployrolling.yaml --record

Kubectl rollout history deplotment rolldeployment -> rollout hsitory komutu bizlere deplotment üzerinde olan değişiklikleri gösterir.

Kubectl rollout history deployment rolldeployment –revision=2 -> bu listedeki ikinci revisyonu göster (yukardaki komutta bir geçmiş veriyor bize bizde bu komutta o geçmişin2. Maddesindeki duruma(yapılan işleme) detaylı şekilde baktık )

Kubectl rollout undo deployment rolldeployment -> bir önceki haline gelir

Kubectl rollout undo deployment rolldeployment –to-revision=1,2,3,4(yukardaki listedeki geçmiş durumlardan birine dönebiliriz orada 4 madde vardı farklı yerlerde daha fazla olacaktır.)

Kubectl rollout status deployment rolldeployment –w -> deploymentımızda canlı olarak ne oluyor bunu status ile görürüz.

Kubectl rollout pause deployment rolldeployment -> bir değişiklik yaptığımzda bir pronlemle vs karşılaştığımıda geri almaktansa işlemi duraklartır. Bu şekilde sıkıntının nerden kaynaklandığını bulmada yardımcı olur. Devam ettirmek için ise (Kubectl rollout resume deployment rolldeployment) komutu çalıştırılır.

**Kubernetes Ağ Altyapısı:**

Kubernetes Ağ Kuralları:

Kubernetes Kurulumda pod’lara ip dağıtılması için bir adres aralığı yada kubernetes terminolojisinde bilinen --pod-netwok-cidr belirlenir.

Kubernetes’de her pod bu cidr bloğunda bir eşsiz ip adresine sahio olur.

Aynı cluster içerisinde tüm podların varsayılan olarak birbiriyle herhangi bir kısıtlama olmadan ve NAT yani netwok address translation olmadan haberleşebilirler.

Cloud Native Computing foundation projesi olan cnı,

Linux containerkarda ağ arabirimlerini yapılandırmak için eklentiler yazabilmesini sağlayan spesifikasyonları belirler.

CNI,yalnızca containerların ağ bağlantısıyle ve containerların silindiğinde ayrılan kaynakların kaldırılmasıyla ilgilenir bu odak nedeniyle cnı eniş bir desteğe sahiptir ve spesifikasyonun uygulanması kolaydır.

**Service :**

Bir dizi pod üzerinde çalışan bir uygulamayı ağ hizmeti olarak göstermenin soyut bir yolu.

Kubernetes ile,uygulamamızı tanıdık olmayan bir hizmet bulma mekanizması kullanacak şekilde değiştirmeniz gerekmez. Kubernetes pod’lara kendi IP adreslerini ve bir dizi pod için tek bir DNS adı verir ve bunlar arasında yük dengeleyebilir.

Dört tür servis vardır;

ClusterIP: Servisi bir küme içi IP'de açığa çıkarır. Bu değerin seçilmesi, Servisin yalnızca küme içinden erişilebilir olmasını sağlar. Bu, varsayılan ServiceType'tır.

NodePort: Servisi her bir düğümün IP'sinde sabit bir portta (NodePort) açığa çıkarır. ClusterIP Servisi, NodePort Servisinin yönlendirdiği yere otomatik olarak oluşturulur. Küme dışından, <NodeIP>:<NodePort> isteğinde bulunarak NodePort Servisine ulaşabilirsiniz.

LoadBalancer: Servisi, bulut sağlayıcının yük dengeleyicisini kullanarak harici olarak açığa çıkarır. Harici yük dengeleyici yönlendirmelerinin bulunduğu NodePort ve ClusterIP Servisleri otomatik olarak oluşturulur.

ExternalName: Servisi, externalName alanının içeriğine (örneğin, foo.bar.example.com) eşler ve değeriyle birlikte bir CNAME kaydı döndürür. Herhangi bir türde vekil ayarlaması yapılmaz.

**Liveness Probes:**

Kubelet,bir containerın ne zaman yeniden başlatılacağını bilmek için liveness probe kullanılır.Örneğin ,Liveness probe ,bir uygulamanın çalıştığı ancak ilerleme sağlayamadığı bir kitlenmeyi yakalayabilir. Böyle bir durumda bir container’ı yeniden başlatmak,hatalara rağmen uygulamayı daha kullanılabilir hale getirmeye yardımcı olabilir.

**Readiness Probes:**

Kubelet ,bir containerın ne zaman trafiği kabul etmeye hazır olduğunu bilmek için readiness probeları kullanılır. Bir pod tüm containerlar hazır olduğunda hazır kabul edilir.Readiness probe sayesinde bir pod hazır olana kadar services arkasına eklenmez.

**Resource Limits:**

Podların Cpu ve memory kısıtlamaları. Containelar aksi belirtilmediği sürece üstünde çalıştıkları hosttun tüm kaynklarına erişebilirler.

Request: //min boş olması gereken kaynak

Memory:”64m”  
 cpu:”250m”

limit: // max kullanabilceği kaynak

Memory:”250m”  
 cpu:”0.5”  
eğer container’ın daha çok kaynağa ihtiyacı olursa “oomkilled “ durumuna geçer ver restart eder.

**Enviroment Variables :**

Bizler çalıştırdığımız ortama göre değiştirilebilcek (db bilgileri gibi test ,port ortamlarında yarı olabilir)bilgileri container imajlarına hard-code olarak gömmeyiz onun yerine uygulamalarımda bunların çalıştırıldıkları sistemden okuyacakları şekilde bir değişken olarak tanımlarız Enviroment variables’dan okuması için tanım yaparız

env:

-name:USER

value:”EMRE”

-name:database  
 value:”testdb.example.com”

**Kubernetes 102**

**Volume:**

**Secret:**

Hassas verileri uygulama deploy etmeden dayrı yönetmemize olanak sağlar.

Kubernetes Secret,parolalar,OAUTH token,ssh anahtarları gibi hassas bilgileri depolamamıza ve yönetmemize olanak sağlar gizli bilileri bir secret içerinde saklamak onu bir pod tanımına veya bir container imajına koymaktan daha güvenli ve esnektir.

Örnek: Yaml File:  
  
apiVersion: v1

kind: Secret

metadata:

  name: mysecret

type: Opaque

stringData:

  db\_server: db.example.com

  db\_username: admin

  db\_password: P@ssw0rd!

Not: oluşturduğumuz secretlar ile ataycağımız podlar aynı namespacede olamalı.

Type:

8 değişik tipe secret oluştuabiliriz  
  
opaque(generic) : defaulttur.

İparatif yönetmle oluşturma:

kubectl create secret generic mysecret2 –from-literal=db\_server=db.example.com –from-literal=db\_username=admin –from-literal=ddb\_password=P@SSW0RD

biz bu bilgileri shall den girdik eğer birisi bizim history görürse problem yaratabilir. Bunun içinde iparatif şekilde yapıp bilgileri ayrı ayrı dosyalardan alabiliriz.

Bu şekilde bilgileri dosyadan okuması gerektiğini söyleyebiliriz.

kubectl create secret generic mysecret3 –from-file=db\_server.=db.example.txt –from-file=db\_username=admin.txt –from-file=db\_password=password.txt

başka bir örnek

kubectl create secret generic mysecret4 –from-file=config.json (dosyanın ismi anahatar değerim dosya içeriğide value değerim olacak).

Bu değerleri volume yada env üzerinden aktarabiliriz.

apiVersion: v1

kind: Pod

metadata:

  name: secretpodvolume

spec:

  containers:

  - name: secretcontainer

    image: ozgurozturknet/k8s:blue

    volumeMounts:

    - name: secret-vol

      mountPath: /secret

  volumes:

  - name: secret-vol

    secret:

      secretName: mysecret3

---

apiVersion: v1

kind: Pod

metadata:

  name: secretpodenv

spec:

  containers:

  - name: secretcontainer

    image: ozgurozturknet/k8s:blue

    env:

      - name: username

        valueFrom:

          secretKeyRef:

            name: mysecret3

            key: db\_username

      - name: password

        valueFrom:

          secretKeyRef:

            name: mysecret3

            key: db\_password

      - name: server

        valueFrom:

          secretKeyRef:

            name: mysecret3

            key: db\_server

---

apiVersion: v1

kind: Pod

metadata:

  name: secretpodenvall

spec:

  containers:

  - name: secretcontainer

    image: ozgurozturknet/k8s:blue

    envFrom:

    - secretRef:

        name: mysecret3

Not: secretler uygulama ile hasas verileri ayrımaya yarıyan kubernetes objeleri

Biz bir secret oluşturduğumuzda etcd ‘de base64 encoded olarak tutulyor yani encripted değil eğer istemediğmiz biri etcd de erişirse bu hasas verilerde sahip olabilir bunun için etcd encribted olarak saklaycak ayaralamalar yapabiliriz. Cloud servisler sağlayıcılarda bu zaten devrede oluyor. Kendi kurup yönettiğimiz cluster da bunu manuel olarak devreye almamız gerekiyor.(encribting-scret-data-rest)

Biz bir secret oluştuduğumuz zaman varsayılan olarak diğer userlar secretlara erişim haklarına sahiptir. Bu ve benzeri durumlar engellenbilir bunu ilerde göreceğiz.

**ConfigMap:**