KUBERNETES

Sanallaştırma

Eski donanım kaynakları, performans değerleri oldukça düşüktü. Eski donanım kaynakları üzerinde tek bir işletim sistemi vardı.

Geleneksel mimaride, en alt katta donanım, bir üst katta işletim sistemi, en üst katta ise uygulamalar vardı.

Zamanla donanım kaynakları gelişti. Yüksek performanslı donanım kaynağı üzerinde tek bir işletim sistemi mantıksızdı. Bu mantıkla paylaşımlı donanım ortaya çıktı. Yıllar içerisinde de sanallaştırma teknolojisi ortaya çıktı.

Sanallaştırma ile donanım kaynakları havuz haline getirildi. Aynı donanım üzerinde birden çok işletim sistemi kuruldu. Bu bize var olan donanım kaynaklarını verimli bir şekilde kullanma imkanı sundu.

Sanal katman üzerindeki her bir işletim sisteminin kendi CPU'su, RAM'i, diski ve network kartı bulunur.

Sanallaştırma Katmanı = Hypervisor

Container Teknolojisi

Günümüzün popüler uygulama çalıştırma teknolojisidir.

Uygulamaları birbirinden izole hale getirmek için iki teknoloji geliştirildi.

1- Namespaces
   Uygulamanın görebileceği alanları kısıtlar.
2- Control Groups
   Uygulamanın kullanacağı donanım kaynağını kısıtlar.

İşletim sistemine eklenen bu iki teknoloji sayesinde uygulamalar birbirinden bağımsız hale getirilebildi.

Altı farklı namaspace bulunur.

- pid
- network
- mount
- user
- Ipcs
- uts

Her bir namaspace'nin farklı kısıtlama görevi vardır.

Uygulamamızın ihtiyaç duyduğu tüm gereklerin bir kutuya koyduğumuzu ve bu kutunun namespace ve control groups'lar ile kısıtladığımızı düşünelim. Bu kutuya **container** deriz. Her bir container birbirinden izoledir.

Container içerisindeki her bir uygulamanın hostname, IP address ve disk alanı bulunur. Yani uygulama çalıştığı işletim sisteminin sadece kendisine ait olduğunu düşünür.

Geleneksel Mimari → Sanal Mimari → Konteyner Mimari

DevOps'ta temel amaç tüm yazılım sürecinin otomasyon haline getirildiği bir yapı kurmaktır. Bu da container mimarisi ile hızlı ve güvenli bir şekilde yapılabilmektedir.

Docker Nedir?

Konteyner teknolojisini kullanarak uygulama geliştirmeyi, dağıtmayı ve çalıştırmayı kolaylaştıran açık kaynak kodlu bir platformdur.

Docker uygulamaları izole hale getirmek için işletim sistemindeki namaspace ve control groupsları kullanır.

Docker'dan öncede container teknolojisi vardır.

Microservice Mimarisi

Bu mimari yapı, birbirinden bağımsız olarak çalışan ve birbirleriyle haberleşerek bir bütün olarak hareket eden servisleri ifade eder.

Microserviceler, sadece bir işi yapan çok küçük kod parçalarıdır. Her servis birbirinden bağımsız olarak geliştirilir.

Amaç uygulamayı parça parça geliştirmektir. Serviceler birbiri ile API Geteway üzerinden haberleşirler.

Konteyner Orkestrasyon Nedir?

Container Orchestration araçları, sistem üzerinde oluşturulmuş tüm containerları sorunsuz bir şekilde yönetebilir.

Bu araçlar, konteyner mimarisinde oluşturduğumuz konteyner yığınlarının dağıtılması, ölçeklendirilmesi ve yönetimini otomatikleştirmek için kullandığımız yönetim araçlarıdır.

Kubernetes

Sistemimizde fazla sayıda container bulununca ortaya bunları yönetim sorunu çıktı. Bu yönetim için geliştirilen teknolojilerden biride Kubernetes'tir. Kubernetes bu yönetimi merkezi bir yerden gerçekleştirir.

Kubernetes, konteyner yığınlarının dağıtılması, ölçeklendirilmesi ve yönetimini otomatikleştirmek için oluşturulmuş açık kaynak kodlu konteyner orkestrasyon aracıdır.

Lokalde ya da bulut ortamında kullanılabilir.

Kubernetes,

- Yönlendirme
- Ölçeklendirme
- Replicasyon
- Otomatik Sorun Giderme
- Sıfır Kesinti ile Güncelleme

gibi konularda yönetim sağlar.

Kubernetes tek bir uygulama olarak karşımıza çıkmamaktadır.

Kubernetes de her bir makinaya node denilmekte.

Kubernetes mimarisinde, en alt katta donanım bir üst katmanında kubernetes, onun üstünde docker engine aracı, bu araç içerisinde de proje jodu bulunur.

Kuberbetes, Google tarafından GO dilinde yazılmıştır.

Docker ile Kubernetes arasında ki fark nedir?

Docker, container teknolojisini kullanarak uygulamaların paketlenme işlemiyle, kubernetes ise containerların sorunsuzca çalışıp çalışmadığını kontrol ederek, yönetim işeri ile ilgisenir.

DevOps

Devops kavramındaki temel amaç geleneksel, hantal yazılım mimarisi yerine, her aşamanın otomasyon haline geldiği ve sürecin çok daha hızlı bir şekilde yönetilerek yazılım aşamalarının sorunsuzca yapıldığı yeni bir mimarinin ortaya konulmasıdır.

Devops, bir yazılım ya da uygulamaya verilen isim değildir.

Temel amaç, yazılım geliştirme yaşam döngüsünü minimuma indirerek yazılım sürecinde ki tüm aşamaları otomasyon haline getirildiği bir mimari ortaya koymaktır.

Devops kültürü, yazılımın hızlı ve yüksen kalitede yayınlama sürecini oluşturan her şeyin birleşimidir diyebiliriz.

DevOps Mühendisi temel olarak DEVelopment ve Operations ekibiyle birlikte çalışan, sorunları sistematik bir şekilde çözüme kavuşturan, operasyon, test ve geliştirme birimleri arasında kordinasyonu sağlayan, ürün entegrasyonu, dağıtım süreçlerini aşamalı bir şekilde otomatikleştirmeye çalışan bizimler olarak görev almaya başladı.

Borg → Omega → Kubernetes

(Proprietary (Proprietary) (open-source)

Neden Kubernetes?

Taşınabilirlik	Ölçeklendirme	Yünsek Erişilebilirlik	Açık Kaynaklı
Çoklu Bulut Desteği	Monitor Performans	BT Maliyet Optimizasyonu	Farklı Konteyner Runtime Desteği
Pazar Lideri	Production Lokalde Test Etme	Docker Compose'dan Kubernetes2e Geçiş	Performans Test Senaryo

Kubernetes mimarisi arka tarafta Kubernetes cluster üzerinde çalışan bir mimaridir. **Cluster** yapısı, birden fazla bilgisayarın tek bir sistemmiş gibi davranmasına imkan sağlayan bir teknolojidir.

Cluster içerisine alınan her bir sunucuya **node** denir. Nodelar üzerinde projemiz için oluşturulan **pod**'lar çalışmaktadır. Pod içerisinde proje kodumuz koşmaktadır.

Kubernetes içerisinde iki çeşit node bulunur.

- Master(Control Panel) Node
- Worker Node

Master node üzerindeki bileşenler, tüm clusterı yönetmek için çalışırlar.

Worker node ise işçi node'dır. Tüm esas iş worker node'ler üzerine yapılmaktadır. Worker node'lar yaptıkları tüm işleri master noe rapor etmeli.

Master Node(Control Panel)

- API Server
  Kubernetes'in beynidir. Kubernetes üzerine gelen tüm taleplerin yönetiminden sorumludur. API'ye REST call'lar üzerinden erişilir.
- Etcd(Store)
  Tüm dataların tutulduğu depolama alanıdır. Veriler key value database olarak adlandırılan bir yapıda saklanır.
- Controller Manager
  Kubernetes'in kontrol merkezidir. Tüm nesneleri denetler. Geçerli durum ile istenen durumu izler. 4 farklı kontrolcü bulunur.
  1- Node controller
  Node ayakta mı değil mi?
  2- Replication controller
  Olması gereken kadar kopya var mı?
  3- Endpoints controller
  Pod ve service'lerin endpoint'lerini oluşturur.
  4- Service Account & Token controllers
  Yeni namaspaceler için standart hesapları ve API Access tokenları oluşturur.
- Scheduler
  Yeni oluşturulan containerların hangi node üzerine atanacağına karar verir. Clusterın organizasyon birimidir.

■ Kube-api server, hem dikey (yani daha güçlü makinelerde çalıştırarak) hem de yatay (birden fazla kopya oluşturarak) olarak ölçeklenebilir. Bu özellik, Kubernetes'te yüksek verim ve esneklik sağlamak için kritiktir.

Worker Node

- kubelet

  Master node'nin ajanı şeklinde çalışır. Node içerisinde gerçekleşen her şeyi master node iletir.

- Container runtime
  Konteyner yönetiminden sorumludur. İçerisinde pod'lar bulunur.
- kube-proxy
  Kubernetes networkünden sorumludur. Pod'lara ip adresi verilmesi ve load balancing işlemlerini yapmaktadır.

Pod oluşturma aşamaları:

1- Image oluştur.

2- Image Docker Hub ortamına aktarılır.
3- Yüklenen imageden yeni bir pod oluştur komutunu çalıştır.
4- Bu talep API Server'a iletilir.
5- Pod oluşturmak için uygun worker node karar verilir.
6- Worker node üzerindeki kubelet ile iletişime geçilir
7- Kubelet Docker'a image bilgisini verir.
8- Docker Hub üzerinden image indirilir ve pod oluşturulup çalıştırırlır.

Pod

Kubernetesin en küçük yapı taşıdır. Podlar containerların çalışma alanıdır. Podlar içerisinde containerlar çalışmaktadır. Container içerisinde uygulama kodumuz koşar.

Containerlar direkt çalışmaz. Çalışması için podların içerisinde olmaları gerekir.

Her bir podun bir ip adresi bulunmaktadır.

kubectl run pod_name –image=image_name:versiyon: belirli bir image kullanarak single bir pod oluşturma

kubectl get pod pod_name: pod bilgisini getirme

Kubernetes Nesneleri

Kubernetes nesneleri, uygulamamızı kubernetes üzerinde çalıştırmak, ölçeklendirmek ve yönetmek için oluşturulmuş kalıcı varlıklardır.

- Pod
- Namespaces
- ReplicationController
- Replicaset
- Deployment
- Service
- StatefulSet
- DaemonSet
- ConfigSet
- Volume

Nesne yönetimi için iki farklı yöntem vardır.

1- Imperative
Tek tek komutlar girilerek süreçler yönetilir.
2- Declarative
Yapılacak işlemler bir veya birden fazla script dosyası içerisine yazılıp çalıştırılarak oluşturma ve yönetim işlemi yapılır. Script dosyası yaml ya da json formatında olabilir.

Kubernetes komut satırı yönetim arayüzü aracı kubectl'dir.

- Geçerli bir json dosyası aynı zamanda geçerli bir yaml dosyasıdır.

Label ve Selector

Kubernetes üzerinde nesneleri oluştururken, bu nesne hangi iş için hizmet verecekse onu ifade eden etiket ataması yaparız. Oluştururken verdiğimiz label değeri o nesneyi diğer nesnelerden ayırır. Bu etiket ile nesneyi bulup üzerinde işlem yaparız. Label tanımlaması anahtar ve değer şeklinde yapılır.

Her bir label üzerinde çalıştığı pod'u açıklamaktadır.

Selector kavramı, etiketlediğimiz nesneleri bulup işlem yaptırmak istediğimizde kullandığımız bir tanımdır. Kubernetes üzerindeki nesneleri aramak için kullanılır.

Selector yapısının iki türü bulunur.

- Equality-based
  ReplicationControllers
  Services
- Set-based
  ReplicaSets
  Deploymonts
  Jobs
  DeamonSet

Etiket arama kriterine kesinlikle uyulmalıdır. Birden fazla kriter varsa aralarına virgül konulur.

- label

kubectl run myapp –image=nginx –labels="env=frontend": pod'a label etikedi verme

kubectl run myapp –image=nginx –labels="env=prod,tier=frontend": pod'a birden fazla label verme

kubectl label pods myapp "env=demo, tier=backend": çalışan bir pod'a sonradan label ataması yapma

kubectl label node "disktype=ssd": node içerisine label ataması yapma

kubectl label pod k8s-label-4 env=demo –overwrite: çalışan podun var olan label değerini değiştirme

kubectl label pod –all status=healty: Çalışan tüm podlara yeni label ekleme

kubectl get pods –show-labels: podlara verilen tüm labelları listeleme

- selector

İki yazım yolu vardır.

1- Equality-based
   = == !=
   Örn: env = production / tier != frontend
2- Set-based
   İn notin exists
   Örn: env in (production) / tier notin (frontend,backend)

kubectl get pods –selector="env=production, tier=frontend": verilen selector adında olanları getir

kubectl get pods –selector=”tier!=production”: verilen selector adı dışında olanları getir

kubectl get pods -l “env in (production)”: verilen selector adında olanları getir

kubectl get pods -l “tier notin (frontend)”: adı verilen selector dışındakilari getir

kubectl get pods –field-selector metadata.name=myApp: Nesne ismi girilen değer olanı getirir.

Namespacea

Namespace, Kubernetes içerisinde ki Pod’lar vaya objeleri gruplamamıza imkan sağlayan teknolojidir. Kubernetes cluster içerisinde farklı virtual cluster alanları oluşturmak için kullanılır.

Bir namespace grubu içerinde oluşturulan nesne diğer namespace içerisinden görünmeyecektir.

Kubernetes ilk kurulduğunda 4 namespace dahili olarak gelmektedir.

- Default
  Varsayılan namespace
- kube-system
  Kubernetes sitemi tarafından oluşturulan nesneler için
- kube-pubkic
  Tum nesneler tarafından erişilmesi istenen nesneler için
- Kube-node-lease

Namespace isimleri birbirinden bağımsız olmak zorundadır.

Bir namespace içerisinde oluşturulan nesneler diğer namespace gruplarında gözükmemektedir.

kubectl create namespace namespace_name: yeni bir namespace oluşturma

kubectl get po –namespace=namespace_name: ismi girilen namespace’i listeleme

kubectl create -f proje.yaml –namespace=namespace_name: yamlda grilen nesneler verilen namespace altında oluşur

kubectl exec -it proje –namespace=namespace_name -- /bin/bash: özel bir namespace alanındaki poda giriş yapmak

kubectl get pod –all-namespace: tüm namespaceler içerisindeki tüm podları listeler

kubectl get pod -A: tüm namespaceler içerisindeki tüm podları listeler

kubectl run prod-app --image=nginx -n poject: yeni podu verilen namespace altında oluştururuz

Desired state ve Actual state

Desired State olması istenilen, hedeflenen durumu ifade eder.

Actual State, şuan ki var olan durumu göstermektedir.

Hedeflenen duruma desired state etiketi, mevcut duruma ise current satate etiketi vurulur.

Replication

Elimizdeki datanın birden fazla kopyasının oluşturulmasıdır.

İstenilen sayıda kopyanın olup olmadığını sürekli kontrol eden iki yapı vardır. Bunlar:

1- ReplicationController
2- ReplicaSet

Bu iki nesnenin amacı belirli bir zamanda çalışan podların kararlı ve replica bir şekilde çalışmasını sapğlamaktır. Desired state ve actual state bu nesneler tarafından kullanılmaktadır.

Bu iki nesne denetleyici konumundadır. Her iki nesnede aynı işi yapmaktadır. ReplicaSet nesnesi ReplicationController nesnesinin yeni halidir. Eski sürümde yaşanan sorunlar nedeni ile bu yapı ReplicaSet ve Deployment olarak ikiye ayrıldı.

Bu denetleyicinin bize sağladığı faydalar:

- High Availabilty
  Yüksek erişilebilirliktir. Yani sistemin kesinti olmadan sürekli hizmet vermesidir.
- Load Balance
  Birden fazla kopya yük dengeleme işini sorunsuz yapmamıza olanak sağlar. Böylece bir pod üzerinde aşırı trafik oluşması engellenmekte ve trafik farklı podlara gönderilerek yük dengelenir.
- Scale
  Sistemdeki kullanıcı sayısı arttığında var olan pod sayısının otomatik olarak ayarlanmasıdır.

Deployment

Deployment nesneleri uygulamamızın kopya sayısını ve image versiyon güncellemesi olduğunda güncelleme stratejilerini belirleyen, yanlış bir durum olduğunda da güncel versiyondan önceki versiyonlara geçiş yapmamızı sağlayan nesnelerdir.

Deployment nesneleri replicaset nesneleri ile birlikte çalışmaktadır. Deployment ile Rolling update ve rollback özellikleri kullanılarak güncelleme ve geri alım yapılabilir.

Tüm uygulamalarımızı deployment nesnesi üzerinde yayınlarız.

Kubernetes Service

Service nesnesi uygulamalar arasında iletişim kurmamızı veya cluster dışından gelen isteklerin pod'lara dengeli bir şekilde dağıtılmasını sağlayan bir objedir.

Mantıksal bir grup oluşturarak kullanıcıların ya da uygulamaların podlara güvenli bir şekilde erişmesini sağlayan bir nesnedir.

Service nesnesinin sağladığı yararalar:

- Static ip adress
- Load balancing

Kubernetes üzerinde 3 farklı service türü bulunur. Bunlar:

1- ClusterIP
2- NodePort
3- Load Balancer

Kubernetes Network

- Aynı pod içeriisinde container'ların iletişimi?
  Localhost seviyesinde iletişim gerçekleşir.
- Aynı node içerisindeki pod'ların iletişimi?
  Bridge üzerinden iletişim sağlanır.
- Farklı node'ler arasındaki pod'ların iletişimi?
  Overlay network üzerinden haberleşirler.
- Service ve pod'lar arasında ki iletişim?
  Label ve selector aracılığı ile iletişim kurar.

Minikube Nedir?

Minikune, kubernetes üzerinde yapmak istediğimiz testlerimizi ve geliştirmelerimizi local bilgisayarımızda çalıştırmak için oluşturulmuş mini kubernetes cluster uygulamasıdır.

Minikube, kubernetes projesi altında hizmet veren alt bir projedir.

minikube node list: var olan node'leri listeleme

minikube node add: yeni node ekleme

Kubernetes CLI(Komut Satırı)(Imperative)

Kubectl, kubernetes konut satırı aracıdır. Kubectl aracılığı ile API Server ile konuşuruz.

kubectl [command] [type] [name] [flag]

kubectl komutları

kubectl –help: var olan komutlar ile ilgili bilgi

kubectl version: kubernetes versiyon bilgisi

kubectl version –short: kısa kubernetes versiyon bilgisi

kubectl describe pod pod_name: objenin detaylı bilgisi

kubectl delete pod pod_name: oluşturulan podu silme

kubectl logs pod_name: podun loguna bakma

kubectl exec -it pod_name --bash: container içerisine giriş yapma

kubectl apply –file test.yml: dışardaki bir yaml dosyasını çalıştırmak

kubectl explain service: kubernetes üzerindeki nesnelerin detayları

kubectl create nesne_type: Verilen türde nesne oluşturma

kubectl get nesne_type: kubernetes üzerinde tipi verilen nesneleri getirme

kubectl describe nesne_type nesne_name: tipi verilen nesne hakkında ayrıntılı bilgi

kubectl get nodes: var olan nodeları listeler

kubectl get po/pods/pod: var olan podları listeleme

kubectl get pod/pod_name: ismi verilen podu listeler

kubectl get all: sistemdeki tüm nesneler listelenir

kubectl get po -o name: podların sadece isim bilgisi listelenir

kubectl get nodes-o name: nodelerin sadece isim bilgisi listelenir

kubectl get nesne_type -o wide: türü girilen nesneler hakkında daha fazla özellik listelenir

kubectl get nodes minikube -o yaml: ilgili node yaml formatında listelenir

kubectl get nodes minikube -o yaml > demo.yaml: Çıktı yaml dosyasına gönderilir.

kubectl describe po pod_name: çalışan pod hakkında ayrıntılı bilgi

kubectl explain pod: pod nesnesinin detayları

kubectl delete nesne_type nesne_name: girilen nesne türündeki adı verilen nesne silinir

kubectl delete -f xx.yaml: Oluşturulmuş yaml dosyasını siler

kubectl delete rs replicaset_name: oluşturulan replicaseti siler

kubectl delete po –all: oluşturulan tüm podları siler

kubectl delete all: sistemde oluşturulan tüm nesneleri siler

kubectl get pods --output name: default namespace'deki tüm pod'ların sadece isimleri ile listeleme

kubectl logs --tail=3 tomcat: logların sadece 3 satırını listeler

kubectl exec pod_name – komut_nmae: poda giriş yapmadan girilen kodu çalıştırır

kubectl cp lokaldeki_dosya pod_name:nereye_kopyalanacak: lokaldeki bir dosyayı pod içerisine kopyalama

kubectl cp namespace_name/pod_name:kopyalanacak_dosya lokalde_nereye_kopyalanacak: Pod içerisinden lokale dosya kopyalama

kubectl port-forward pod_name node_port:pod_port: pod yönlendirme işlemi

kubectl port-forward –address 0.0.0.0 pods/pod_name 8080:80: pc'deki tüm ip adreslerinden pod içerisine erişim verilir

kubectl get pods -o=custom-columns=NAME:.metadata.name: podların sadece isimlerini listeler

kubectl exec myweb1 – date: poda bağlanmadan date bilgisini alma

kubectl exec -it pod_name – printenv: pod üzerindeki tanımlamaları kontrol etme

Short name	Full name
csr	certificatesigningrequests
cs	componentstatuses
cm	configmaps
ds	daemonsets
deploy	deployments
ep	endpoints
ev	events
hpa	horizontalpodautoscalers
ing	ingresses
limits	limitranges
ns	namespaces
no	nodes
pvc	persistentvolumeclaims
pv	persistentvolumes
po	pods
pdb	poddisruptionbudgets
psp	podsecuritypolicies
rs	replicasets
rc	replicationcontrollers
quota	resourcequotas
sa	serviceaccounts
svc	services

$HOME/.kube/config

Bu dosya içeriği üç bölümden oluşur.

- Cluster
  Bağlanacağımız cluster bilgileri bulunur.(Development,AWS,Production...)
- Context
  Hangi kullanıcının hangi cluster hesabına bağlanacağı bilgisini tanımlarız.(dev@development, cloud@aws, admin@production...)
- Users
  Cluster'lara hangi kullanıcı ile bağlanacağımız bilgisi yer alır.(Dev User, Cloud User, Admin...)

kubectl config: komut satırı üzerinden kube-config dosyasını yönetmen için kullanılacak komutlar listelenir

kubectl config view: kube dosyası altındaki config dosyasının içeriğini listeler

kubectl config get-contexts: config dosyasındaki tüm clusterlar listelenir

kubectl config use-context cluster_name: ismi girilen cluster'a geçiş yapar

kubectl config current-context: varsayılan bağlantı bilgisi gelir

kubectl config view –ninify: sadece aktif context'i listeler

---

kubectl edit nesne_type nesne_name: Çalışan nesne üzerinde değişiklik yapmamızı sağlar(komut çalışınca nesne detayları txt üzerinden bize gelir)

kubectl logs pod_name –follow: log dosyasını sürekli izler

Kubernetes YAML File(Declarative)

Gerçek hayatta biz tüm projelerimizi yaml dosyası üzerine yazmakta ve nesneleri buradan yönetmekteyiz.

Yamal dosyasında üç anahtar bulunur.

- apiVersion:
  Oluşturulacak objenin kubernetes API sürümünü tanımlar
  kubectl explain nesne_name: versiyon bilgisi gelir
- kind:
  Nesne türünü tanımlar
- metadata:
  Nesne ile ilgili **benzersiz** olan tanımlamalar yapılır

Bu anahtarlar her kubernetes obje tanımında olması gereken tanımlardır.

- spec:
  Oluşturulacak nesne ile ilgili durum detaylarını tanımlar

Bu bölüm bazı nesnelerde vardır.

kubectl create –filename test.yaml: kubernetes üzerinde yaml dosyasını çalıştırma

kubectl apply –filename test.yaml: kubernetes üzerinde yaml dosyası çalıştırma(nesne üzerinde güncelleme durumu varsa kullanılmalı)

kubectl apply -f test1.yaml -f test2.yam: birden fazla yaml dosyasını çalıştırma

kubectl apply -f https://git.io/tests: uzaktaki yaml dosyasını çalıştırma

Kalıcı bir değişiklik yapmak istiyorsak önce yaml dosyası üzerinde yapmalı sonra sistem üzerinde bu yaml dosyasını çalıştırmalşıyız.

command & args

Pod oluştururken container içerisinde komut çalıştırmak istersek iki yöntem vardır. Bunlar:

- command
  Konteyner içerisinde çalıştırılacak komutu tanımlar
- arg
  Komut içerisine eklenecek bağımsız değişkenleri tanımlar

Açıklama	Kubernetes ismi	Docker ismi
Container tarafından çalıştırılan komut	command	entrypoint
Command'a iletilen argümanlar	args	cmd

Podların üretildiği imagelar içerisinde container çalışmaya başlarken hangi parametre ile başlayacağını belirten başlangıç parametresi bulunur. Kubernetes tarafında bu değişkenleri command ya da arg değişkenleri ile yönetmekteyiz.

- Konteyner için command ve args komutları kullanılmazsa, Docker image içerisinde belirlenen default değerler kullanacaktır.

- Konteyner için command değeri tanımlanır ve args değeri boş geçilirse, Docker image içerisindeki EntryPoint ve varsayılan Cmd değerleri yok sayılır.
- Konteyner için yalnızca args değeri tanımlanırsa, Docker image içerisindeki EntryPoint değeri tanımladığımız args değeri ile birleştirilerek çalıştırılır.
- Konteyner için command ve args tanımlanırsa, Docker image içerisindeki değerler yok sayılır. Tanımladığımız command args değerleriyle birlikte çalışır.
- Yaml içerisinde kullandığımız command ve args parametreleri, image içerisindeki commad ve args parametrelerini ezer.

Pod Yaşam Döngüsü

Pending → Creating → ImagePullBackOff → Running → Succeeded → Failed → CrashLoopBackOff

Declarative ya da imparative yöntemle pod oluşturulmak istenir.

Pod oluşturma talebi API Server'a iletilir. API Server varsayılan ayarlar ile istenilen ayarları eşitleyip yeni bir pod nesnesi tanımlar. Bu podu Etcd(Store) kaydeder.

Bu aşamadan sonra pod pending aşamasına geçer. Scheduler, Etcd(Store)'da bekleyen bir pending aşamasında bekleyen kayıt görürse hemen işleme başlar. Verilen kriterlere uygun node bulur ve bu Etcd(Store)'da bulunan bu kayda node bilgisini ekler.

Bu aşamadan sonra pod creating aşamasına geçer.

Worker node üzerindeki kubelet ajanı API Server'dan Etcd(Store) üzerine düşen talepleri sürekli izler. Kendi üzerine tanımlanan podu görür ve hemen işleme başlar.

Worker node ilk olarak container imagesinin lokalde olup olmadığını kontrol eder. Lokalde yokse repositoryden imageyi çeker. Eğer image indirilemezse ilk ErrorImagePull ardından da ImagePullBackOff uyarısını ekrana basar. Eğer image node üzerinden indirilirse kubelet bileşeni node üzerindeki container engine ile haberleşir ve ilgili containerın oluşmasını ister.

Bu aşamada pod running moda geçmiştir.

Pod içerisindeki conteinerlar doğal olarak işlemlerini tamamlayıp kapanırsa pod succeeded olarak işaretlenir.

Eğer restartPolicy ayarı always olarak ayarlanmışsa container ne durumda kapanırsa kapansın pod yeniden başlatılmaya zorlanır. Container hiçbir zaman compleated ya da failed olarak işaretlenmez. Pod işlemini tamamlar compleated olarak işaretlenir fakat restartPolicy ayarı always olduğu için pod yeniden başlatılmaya zorlanır.

Bu döngü bu şekilde devam ederken kubernetes bazı şeylerin ters gittiğini düşünür ve podu CrashLoopBackOff durumuna sokar.

kubectl get pod -w: pod durumlarını sürekli izler

Çoklu Konteyner Kullanımı

Podlar içerisinde birden fazla container çalıştırabiliriz. Peki neden buna ihtiyaç duyarız? Bunun nedenlerinden biri birbirine bağımlı uygulamaların aynı pod üzerinde çalışma ihtiyacıdır. Kubernetes, network ve storage seviyesinde konuşması gereken uygulamaları aynı pod içerisine koymamıza izin verir.

kubectl exec -it multi-container-pod --container=container-1 -- /bin/bash: çoklu container yapısında bir containere bağlanma

kubectl logs multi-container-pod -c container-1: çoklu container yapısında bir containerın logunu listeleme

Init Container

Eğer ana uygulamamız ayağa kalkmadan bağımlı olduğu diğer uygulamaların ayağa kalkması gerekiyorsa bunun kontrolu gerekir. Bu kontrol işlemini init container aracılığı ile yapılabilir.

Pod içerisinde oluşturulan Init container sayesinde talep edilen servisin ayağa kalkıp kalkmadığı kontrol edilir. Servis ayağa kalkana kadar ana uygulama beklemede kalır. Servisin ayağa kalktığından emin olununca ana uygulama ayağa kalkar. Böylece ana uygulama hata vermeden çalışmış olur.

Inıt containerlar belirli bir sıraya göre devreye girerler. En sonda ise ana container devreye girer.

Multi containerlardan farkı, init containerlar görevi bitince kapanır. Multi containerde ise her container bağımsız olarak çalışır.

kubectl logs –container=init_container_name pod_name: init containerın podlarını listeler

Resource Requirement & Limit

- **CPU**
  CPU üzerinde kısıtlama işlemi core sayısına göre yapılır. % kaçlık kısmı kullanılacaksa belirlemek gerekir.
  1 CPU Core Kullan => cpu:”1” veya cpu:”1000” veya cpu:”1000m”
  1 CPU Core’un %50’sini kullan => cpu:”0.5” veya cpu:”500” veya cpu:”500m”
- **Memory**
  Bir pod üzerine talep edilen memory ataması yapılır. Bu kiobyte, megabyte, kibibayt,mebibayt... cinsinden olabilir.

Kaynak kısıtlaması yapmak istediğimizde bu yapıları kullanarak kısıtlama yapabiliriz.

kubectl top node/pod: kubernetes üzerindeki node ya da podların memory ya da cpu değerlerini gösterir

Pod Annotation

Nesne üzerinde ek açıklamalar yapmak istediğimizde annotations kullanırız.

- Bu proje ne zaman oluşturuldu?
- Bu projeyi kim oluşturdu?
- Proje erişim bilgisi nedir?
- Proje imageregistry bilgisi nedir?
- Proje sürüm bilgisi nedir?
- Proje dökümantasyon adresi?
- Projeye ait özel bilgiler?

Burada girilen değerler Kubernetes’in umurunda değildir.

kubectl annotate pod pod-annotation Description='Version-12': poda yeni bir annotation değeri verme

kubectl annotate pod pod-annotation Description='Version-13' –overwrite: podda var olan annotation değerini değiştirme

Environment Variable

Kubernetes üzerinde Pod içerisinde ortam değişkenleri tanımlamak için kullanılmaktadır.

Yaml içerisinde bu değerler üç şekilde yazılır.

1- Key Value
   Açık açık yazılması
2- Secrets
   Değişkenlerin dışarıdaki bir txt dosyasından okutulması
3- ConfigMaps
   Değişkenlerin dışarıdaki bir txt dosyasından okutulması

Kubernetes Scheduling İşlemleri

Scheduler, podlar için uygun nodeleri seçer. Scheduler, bu seçme işlemini iki adımda gerçekleştirir.

1- Filtering
   Talep edilen özelliklerde var olan nodeleri filtreler.
2- Scoring
   Listeye düşen worker nodeler üzerinde puanlama yapılır.

Scheduler, node seçimi için üç farklı yöntem kullanır.

- Node Selector
  Bir podu belirlediğimiz bir node üzerinde çalıştırmanın en kolay yoludur.

  kubectl label node minikube-m03 disktype=ssd: ssd label ataması

- Affinity
  Node Affinity
  Worker nodelara label ataması yapılır.
  Podlar oluşturulurken label etiketi sorgulanarak seçim işlemi yapılır.
  1- Hard Type modelinde, talep edilen label yani etiket bilgisi node üzerinde kesinkez olmak zorundadır.
     requiredDuringSchedulingIgnoredDuringExecution → nodedan label kalkınca podları dode üzerinden tahliye et
     requiredDuringSchedulingrequiredDuringExecution → nodedan label kalkınca podlara dokunma
  2- Soft Type modelinde ise talep edilen etikete sahip node varsa öncelikli olarak bu nodelar üzerine pod oluşturmaya çalışır. Yoksa uygun olan diğer nodelar üzerinde pod oluşturulur.
     preferredDuringSchedulingIgnoredDuringExecution
  Pod Affinity
  Birbirine bağımlı olan podların aynı node üzerinde ya da aynı zone üzerindeki nodeler üzerinde nasıl çalıştırılır, bununla ilgilenir.
- Taint ve Toleration
  Belli nodeler üzerinde belli podların oluşmasını engellemiş oluruz. Yani nodelerin sadece belli pod grubuna hizmet etmesi sağlanır.

kubectl taint nodes [node name][key=value]:TAINT_EFFECT: noda taint bilgisi ekleme
Taint Effect değerleri:

1- NoSchedule
taint tanımı ile aynı olmayan yeni podlar bu node üzerinde oluşturulmasın
2- PreferNoSchedule
taint tanımı ile aynı olmayan podlar başka bir seçenek yoksa bu node üzerinde oluşturulsun
3- NoExecute
taint bilgisi eklenince node üzerindeki podlar kaldırılacak

Probes Kullanımı

Pod üzerindeki uygulamaların sorunsuz olarak çalışıp çalışmadığının kontrol edilmesi gerekir. Probe'lar podların sağlık durumunu kontrol etmek için kullanılmaktadır. Probe periyodik olarak clusterda yapılan diagnostik(tarama) operasyonudur.

Health Check Türleri

- Readiness Probe
  Uygulamamızın hazır olup olmadığını Kubernetes'e bildirmek için kullanılır. Kubernetes, ilgili poda trafik göndermeden önce podun hazır durumda olup olmadığını kontrol eder. Eğer podun hazır olmadığını görürse, cevap alana kadar trafik göndermeyi durdurur.
- Livevess Probe
  Uygulamanın sağlıklı çalışıp çalışmadığını kontrol etmek amacıyla kullanılır. Uygulama sorunsuz bir şekilde çalışıyorsa, Kubernetes onu kendi haline bırakır. Uygulama cevap vermiyorsa, Kubernetes podu kaldırır ve yerine yenisini oluşturur.
- Startup Probe
  İlk çalıştırılan Probe'dur. Yavaş ayağa kalkan uygulamalar için kullanılmaktadır. Probe ayağa kalkmadan podun kubernetes tarafından ortadan kaldırılması önlenmiş olur. Liveness Probe ile birlikte kullanılabilir.

Probe Yöntemleri

- ExecAction
  Pod içerisinde komut çalıştırmak için kullanılmaktadır. Böylece gerekli test işlemi yapılır. Komut çalıştırıldıktan sonra dönen değer 0 ise uygulama sağlıklı bir şekilde çalışıyor demektir. Farklı bir değer dönerse uygulamanın crash olduğu düşünülerek grekli restart işlemi yapılır.
- TCPSocetAction
  YAML dosyasında belirtilen TCP post bilgisinin pod içerisinde ulaşılabilir olup olmadığını kontrol etmek için kullanılır.
- HTTPGetAction
  YAML dosyasında belirtilen http adresine ve portuna sorunsuz bir şekilde erişilip erişilmediğini kontrol etmek için kullanılır. 200 ile 400 arasında dönen değer başarılı olarak kabul edilir. Belirtilen değerlerin dışında bir değer dönerse podun sorunlu olduğu düşünülür ve gerekli restart işlemi yapılır.

Bu yöntemleri kullanarak uygulamaları kontrol etmekteyiz.

Probe Sonuç

- 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

- 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)

Kubernetes Volume

Konteyner içerisindeki dataların konteyner silinse bile kalıcı olması gerekir.

Pod silinse bile veriler pod dışındaki güvenli volüme alanlarına yazıldığı için silinmez, güvende kalır. Data kaybı olmaz.

İki farklı volume vardır.

1- **Ephemeral Volume**
   EmptyDir
   Geçici volümelerdir. Podlar silinince silinir. Pod çalıştığı sürece EmptyDir volume varlığını devam ettirir. Pod içerisindeki tüm containerlar silinince bu volume silinir.
   Kısıtlı memory sahip ve cache kullanımı yapan uygulamalarımız varsa ya da pod üzerinde çoklu konteynerlar arasında dosya paylaşımı yapmak istersek bu volume bunun için oldukça kullanışlıdır.
   Pod oluştururken otomatik oluşturulur.
2- **HostPath Volume**
   Node üzerindeki bir dosyaya hızlı bir şekilde erişmek istersek bu volume türünü kullanabiliriz.
   Tek bir node üzerinde çalışır. Single node için kullanımı elverişlidir.
   Podlar silinince silinmez. Fakat node donanım arızasından giderse volumede gider.

Kalıcı Depolama

Provisioning → Binding → Using → Reclaiming

Persistent Volume(PV): Herhangi bir noda bağlı olmayan, kalıcı olarak veri saklamak için kullanılan nesneyi ifade eder. Statik ya da dinamik olarak yapılabilir.

Persistent Volume Claim(PVC): PV'nin pod içerisine bağlanma işlemini ifade eder.

Sistem yöneticisi → Persistent volume oluştur.

Yazılımcı → Havuzdan volume talep et(Claim) Volume POD'a bağla.

Sunucu üzerindeki dataların, storage üzerindeki disklere aktarılabilmesi için farklı standartlar geliştirilmiştir. Bunlar;

- SAN
  Blok bazlı yüksek bağlantı hızı talep edildiğinde kullanılır. fc, iscsi ...
- NAS

Network taarafından dataya erişilmek istendiğinde kullanılır. nfs ...

Bunlardaki temel amaç uygulamaların ya da kullanıcıların veriye güvenli ve hızlı bir şekilde erişmesidir. Bu yapıda yapılacaklar sırası ile şunlardır:

1- Depolama alanının sağlanması,
2- Depolama alanının ihtiyaçlar doğrultusunda yapılandırılması,
3- Depolama alanına bağlanmak isteyen uygulamaların sorunsuzca bağlanmalarını sağlamak,

İki volume bağlama şekli:

- Persistent volume claim + Persistent volume = Static binding
- Starage class + Storage provisioner = Dynamic binding

StorageClass Nedir?

Volume tarafındaki manuel tanımlama işlemini otomatize etmek için yani dinamik bir hale getirmek için geliştirilmiştir.

Storage üniteleri:

- Fast
- Slow
- Distributed (paylaşımlı)

Kubernetes Secret – ConfigMap İşlemleri

Secret

Sistem üzerindeki hassas verileri saklamamıza ve yönetmemize imkan sağlayan objedir. Secret nesnesi sayesinde yaml dosyası içerisinde kullandığımız hassas verileri güven altına alabilmekteyiz.

1- Talep oluştur
2- Secret oluştur ve etcd kaydet
3- Pod için secret sorgulaması yap
4- Secret kullan

Secretlar onları kullanan podlardan bağımsız oluşturulmaktadır.

kubectl create secret generic secren_name: standart boş yeni secret oluşturma

kubectl get secret: secretları listeleme

ConfigMap

Uygulamalarımızda kullandığımız konfigürasyon datalarını merkezi bir noktadan yönetmemizi sağlayan bir kubernetes objesidir. Hassas olmayan verilerle ilgilenir.

ConfigMap içerisindeki dataları iki şekilde kullanabiliriz.

- Bunlardan ilki içeriklerin environmant vairiable yani ortam değişkeni olarak tanımlanmasıdır.
- Bir diğeri ise belirlediğimiz path üzerine configmap içeriklerinin mount edilmesi ve kullanılmasıdır.

Secret kullanımına benzemektedir.

kubectl create configmap configmap_name: boş bir configmap oluşturma

kubectl get configmap: configmapleri listeleme

kubectl create configmap configmap_name –from-file=file_yol: dosyadan verileri maunt etme

Kubernetes Deployments

Replication Controller ve ReplicaSet

Kubernetes üzerinde iki farklı denetleyici vardır.

- Replication Controller
  Bu nesne ile hem pod scale işlemleri, hemde container image sürüm güncelleme işlemleri yapılmaktaydı.
- ReplicaSet
  ReplicaSet nesnesi denetleyici olarak, deploymöent nesnesi ise rollout işlemleri için kullanılmaya başladı.

Bu iki kontrolerda belirli sayıdaki podun sorunsuz çalışıp çalışmadığı kontrol altında tutarlar.

Peplication Controller(RC) → ReplicaSet(RS) + Deployment

Eski yeni

Replicas: Replicas tanımı ile verilen değere göre pod seti oluşturulacak ve sürekli takip edilecektir.

Pod Template: Pod seti oluşturulurken her pod için girilecek standart ayarlar pod seti üzerinde yapılacaktır.

Selector: Label etiketine göre podları arayıp bulur.

kubectl scale replicas = 10 rs/myapp:: scale işlemi

kubectl scale replicas = 10 rc/myapp: scale işlemi

kubectl scale replicas = 10 -f rs.yaml: yaml dosyası üzerinden scale işlemi

kubectl autoscale rs –min=0 –max=5 –cpu-percent=80: belirli şartlarla scale işlemi

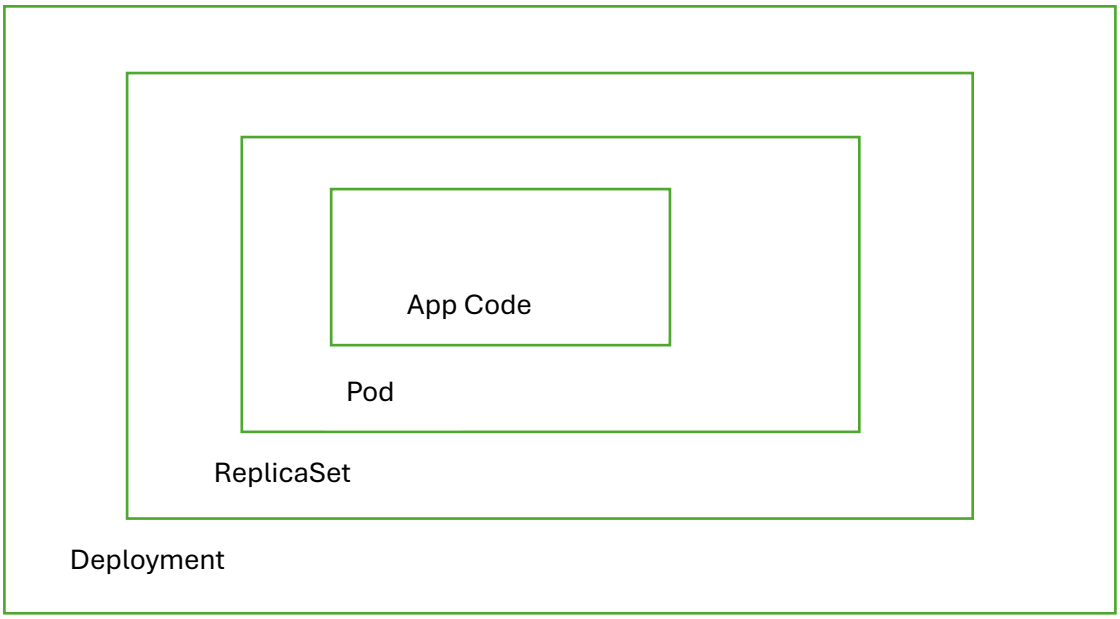
kubectl delete replicaset myapp: replicaset nesnesini ve ona bağlı podlar silinir.

kubectl delete replicaset myapp –cascade=false: sadece replicaset nesnesini silme

Deployment

Bundan sonra projelerimizi deployment üzerinden çalıştıracağız.

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



Deployment nesnesinin başlıca üç görevi vardır.

1- Replicaset Yönetimi
2- Rolling Update
3- Rollback

kubectl create deployment_name –image=image_name: deployment oluşturma

kubectl get deployment: deploymentları listeleme

kubectl create deployment_name –image=nginx –dry-run=client -o yaml > test.yaml: yaml dosyasından deployment oluşturma

kubectl create deployment_name –image=image_name –replicas=5: birden fazla kopyası olan deployment oluşturma

kubectl scale deployment_name –replicas=rempica_sayısı: deploymenta ait replikaları artıma yada azaltma

Deployment Stratejileri

Recreate

Eski sürümler tek seferde ortadan kalkar. Sonrasında yeni image sürümlü podlar oluşturulur ve hizmet vermeye başlar.

Rolling Update

Yeni sürüme geçiş sıra ile olur. Bir anda olmaz. Varsayılan StrategyType'dir.

MaxUnavailable seçeneği ile, aynı anda silinecek pod sayısını belirtmekteyiz.

MaxSurge seçeneğinde ise, geçiş sırasında sistemde toplamda en fazla kaç pod olacağını belirtmekteyiz.

minReadySeconds özelliği opsiyoneldir. Bu özellikle yeni oluşturulan bir podun kullanılabilir olması için geçmesi gereken minimum süreyi belirtmektedir.

Kuberbetes Services İşlemleri

Persistent Volume | Pod | ReplicaSet | Deployment | Service

|---------- Hazırlık ---------- ----------| |-- Sunum --|

Service nesnesi cluster dışından gelen istekleri karşılayıp podlara dengeli bir şekilde dağıtan, cluster içinde ki uygulamaların birbiriyle iletişim kurmasını sağlayan bir nesnedir.

Service nesnesini podları mantıksal olarak gruplandırma şeklinde düşünebiliriz.

Kubernetes üzerinde yapabileceğimiz servis türleri:

- ClusterIp
  Dahili yani içerdeki servis işlemleri için kullanılır.
- NodePort
  Kullanıcılar dışarıdan uygulama üzerine bağlanıp hizmet almak istediklerinde kullanılır.
- Load Balancer
  Taleplerin Load Balancer üzerinden yapıldığı durumlarda oluşturulan servicedir.

targetPort → Konteynerın dinleyeceği port numarasını belirlemek için kullanılır.

nodePort → Node üzerinde belirlenen port adresi ile servise bağlanmak istediğinde kullanılır.

NodePort Service

Bu service türü, kullanıcıların podlara cluster dışındaki platformlar üzerinden eriştirilmek istendiğinde kullandığımız servis türüdür.

Nodelara port ataması yapılır ve bunlar üzerinden erişim yapılır. Bu port aralığı 30000 – 32767 ‘ dır. Port başına sadece bir service çalıştırılabilir.

kubectl get endpoints: Oluşturulan service nesnesinin gelen talepleri nereye yönlendirir, bunu gösterir

LoadBalancer Service

Bulut sağlayıcılar üzerinde çalışan uygulamalarımızı Externel IP üzerinden erişime açmak istediğimizde kullandığımız service türüdür.

Her servis türü için bir IP adresi tanımlanır. Doğrudan IP adresi üzerinden uygulamamıza bağlanırız.

minikube tunel: Load balancer service türünü simüle etmek için kullandığımız komutlardan biridir.

ClusterIP Service

Cluster içerisinde oluşturduğumuz bir uygulamayı cluster da bulunan diğer servislerle konuşturmak istediğimizde kullandığımız servis türüdür. Cluster dışı erişime izin verilmez.

Standart servis türüdür.