İleri Seviye Kubernetes Eğitimi
DevOps Uygulamaları
ve Bulut Bilişim Altyapıları

Mehmet Çağdaş SAYGILI
Kıdemli DevOps Mühendisi

Hoşgeldiniz..

Bu eğitimde, modern yazılım geliştirme süreçlerinde sıkça kullanılan araçları ve alt yapıları ve sanallaştırma teknolojilerini öğrenerek, DevOps ve bulut bilişim uygulamaları konularında bilgi sahibi olacağız.

Eğitimin içeriği, pratik uygulamalar ve örneklerle desteklenmiş, katılımcıların bu araçları etkin bir şekilde kullanabilmelerini sağlamayı hedeflemiştir.

Eğitimde İnceleyeceğimiz Konular

Vagrant: Sanal geliştirme ortamları oluşturma ve yönetme.

Docker: Konteyner tabanlı uygulama geliştirme ve dağıtım.

Kubernetes: Konteyner orkestrasyonu ve yönetimi.

Helm Chart: Kubernetes uygulama paketleme ve dağıtımı.

PostgreSQL: İleri düzey ilişkisel veri tabanı yönetimi.

Vagrant Nedir?

Vagrant, yazılım geliştiricileri için kolay ve hızlı sanal makineler (VM) oluşturmayı sağlayan bir araçtır. Sanal ortamları otomatikleştirir ve yönetir.

Tekrar Edilebilir Ortamlar: Her geliştirici aynı ortamda çalışır.

Platform Bağımsızlığı: Vagrant, Windows, macOS ve Linux'ta çalışır.

Kolay Entegrasyon: VirtualBox, VMware, Docker gibi sağlayıcılarla çalışır.

Yazılım Geliştirme ve Test Ortamları

Eğitim ve Demolar

DevOps Süreçleri

Vagrant Temel Bileşenleri

Vagrantfile

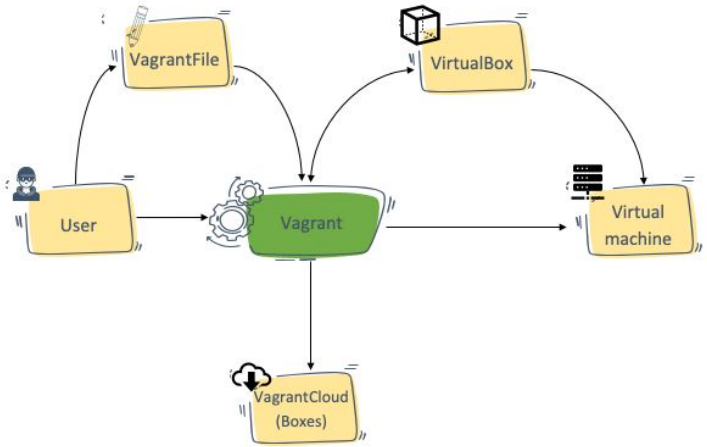
Vagrant'ın yapılandırma dosyasıdır. VM yapılandırmalarını (RAM, CPU, disk, vb.) içerir. Proje kök dizininde bulunur.

Provider

Vagrant'ın VM'leri oluşturmak için kullandığı sanallaştırma yazılımıdır. (VirtualBox, VMware, Docker.)

Plugin

Vagrant işlevselliğini genişleten eklentilerdir. (vagrant-vbguest, vagrant-aws.)



Box

Önceden yapılandırılmış sanal makine imajlarıdır.

Vagrant, box'ları kullanarak VM'leri hızlıca oluşturur. (ubuntu/bionic64)

Vagrant Kurulumu ve Temel Kullanım

Vagrant ve Oracle Virtualbox ürünlerinin kurulumları gerçekleştirilir.

Temel Komutlar

vagrant init: Proje için bir Vagrantfile oluşturur.

vagrant up: VM'i başlatır.

vagrant status: VM durumlarını gösterir.

vagrant halt: VM'i durdurur.

vagrant destroy: VM'i siler.

vagrant ssh: VM içerisine SSH bağlantısı yapar.

(Vagrant-Cheat-Sheet)

Vagrant BOX Yapılandırması (Demo)

https://app.vagrantup.com/boxes/search internet sitesinden BOX bilgisi araştırılabilir.

vagrant box add ubuntu/focal64

vagrant box list

vagrant box repackage ubuntu/focal64 virtualbox 20201204.0.0

mv package.box ubuntu-focal64-20201204-0-0.box

vagrant box add test ubuntu-focal64-20201204-0-0.box

vagrant box remove test

Vagrant Kurulumu ve Bağlantı (Demo)

https://github.com/mcsaygili/k8s-egitim-notlari adresinden Vagrantfile dosyasına erişim sağlanabilir.

vagrant init ubuntu/jammy64

vagrant ssh

vagrant halt

vagrant destory

```
Vagrant.configure("2") do |config|
  # All Vagrant configuration is done here. The most common configuration
  # options are documented and commented below. For a complete reference,
  # please see the online documentation at vagrantup.com.

  # Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "precise64"

  # The url from where the 'config.vm.box' box will be fetched if it
  # doesn't already exist on the user's system.
  # config.vm.box_url = "http://domain.com/path/to/above.box"

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # config.vm.network :forwarded_port, guest: 80, host: 8080

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network :private_network, ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  # config.vm.network :public_network

  # If true, then any SSH connections made will enable agent forwarding.
  # Default value: false
```

Docker Nedir?

Docker, uygulamaları konteynerler içinde izole edilmiş olarak çalıştıran bir platformdur.

Geliştiricilerin uygulamaları, tüm bağımlılıkları ile birlikte paketlemelerine ve her ortamda çalıştırmalarına olanak tanır.

Hafif: Konteynerler, sanal makinelerden daha az kaynak kullanır.

Taşınabilir: Herhangi bir yerde çalıştırılabilir (laptop, sunucu, bulut).

Hızlı: Konteynerler hızlı başlar ve durur.

(Docker-Architecture)

Mikroservis mimarileri

CI/CD süreçleri

Test ve geliştirme ortamları

Docker Temel Bileşenleri

Docker Engine

Docker'ın çalıştığı temel motor.

Konteynerleri oluşturur, çalıştırır ve yönetir.

Docker Image

Uygulama ve bağımlılıklarını içeren sabit bir dosya.

Bir konteyneri çalıştırmak için gerekli tüm bileşenleri barındırır.

Docker Container

Bir Docker imajının çalışan örneği.

İzole bir ortamda çalışan bir uygulama olarak düşünülebilir.

Docker Hub

Önceden oluşturulmuş Docker imajlarını bulabileceğiniz ve paylaşabileceğiniz bir kayıt deposu.

Kendi imajlarınızı da yükleyebilirsiniz.

Docker Kurulumu ve Temel Kullanım

Docker ürününün kurulumları gerçekleştirilir.

Temel Komutlar

docker pull <image>: Docker imajını indirir.

docker run <image>: İmajdan bir konteyner oluşturur ve çalıştırır.

docker ps: Çalışan konteynerleri listeler.

docker stop <container>: Bir konteyneri durdurur.

docker rm <container>: Bir konteyneri siler.

docker system prune: Docker sisteminde dahil olan tüm bileşenleri temizler.

(Docker-Cheat-Sheet)

Dockerfile ve Örnek Uygulama

Dockerfile

Docker imajlarını oluşturmak için kullanılan bir betik dosyasıdır. Bir uygulamanın nasıl paketleneceğini ve yapılandırılacağını tanımlar. https://github.com/mcsaygili/k8s-egitim-notlari adresinden Dockerfile dosyasına erişim sağlanabilir.

docker build -t my-app .

docker run my-app

docker images

docker stats

docker system df

docker system info

docker system events

Kubernetes Nedir?

Kubernetes (K8s), konteynerleştirilmiş uygulamaları otomatikleştirmek, ölçeklendirmek ve yönetmek için açık kaynaklı bir platformdur. Google tarafından geliştirilmiş ve CNCF (Cloud Native Computing Foundation) tarafından yönetilmektedir.

Otomatik Konteyner Orkestrasyonu: Konteynerlerin yönetimini, dağıtımını ve ölçeklendirilmesini otomatikleştirir.

Kendi Kendini İyileştirme: Konteynerlerin sağlık durumunu izler ve hatalı konteynerleri yeniden başlatır.

Hizmet Keşfi ve Yük Dengeleme: Konteynerler arasında ağ trafiğini yönetir ve dağıtır.

Depolama Orkestrasyonu: Yerel ve bulut depolama sistemleri ile entegre çalışır.

Yüksek Erişilebilirlik: Uygulamaların sürekli çalışmasını sağlamak için çeşitli stratejiler kullanır, böylece hizmet kesintilerini en aza indirir.

Kubernetes Ne İşe Yarar?

Uygulama Dağıtımı: Uygulamaları hızlı ve güvenilir bir şekilde dağıtma.

Ölçeklenebilirlik: Trafik yoğunluğuna göre uygulamaları yatay olarak ölçeklendirme.

Yüksek Kullanılabilirlik: Uygulamaların kesintisiz çalışmasını sağlama.

Kaynak Yönetimi: Donanım kaynaklarını verimli bir şekilde kullanma ve yönetme.

Kullanım Alanları

Mikroservis Mimarileri, CI/CD Süreçleri

Bulut Bilişim Uygulamaları, Veri Tabanı Sistemleri, Loglama Sunucuları

Büyük Veri İşleme

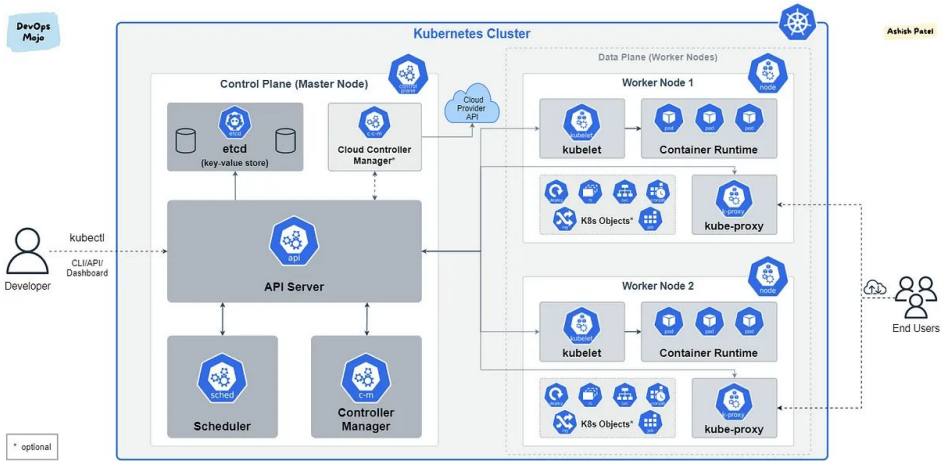
Genel Mimari ve Komponentler

Genel Mimari

Kubernetes, master node ve worker node'lar olmak üzere iki ana bileşenden oluşur.

Master Node: Kümenin kontrol düzlemi.

Worker Node: Uygulamaların çalıştığı düğümler.



Komponentler

API Server: Kubernetes API'sine gelen tüm istekleri karşılar.

etcd: Tüm küme verilerini saklayan dağıtılmış bir anahtar-değer deposu.

Controller Manager: Kümeyi yönetmek için kontrol döngülerini çalıştırır. (Node Controller, Replication Controller)

Scheduler: Yeni oluşturulan pod'ları uygun node'lara atar.

Kubelet: Her node'da çalışan, pod'ların durumunu izleyen ajan.

Kube Proxy: Ağ trafiğini yöneten ve dağıtan bir ağ bileşeni.

Master Node

Master Node Bileşenleri

API Server: Kullanıcıların, CI/CD araçlarının ve diğer kontrol bileşenlerinin Kubernetes API'si ile etkileşime girdiği yer.

etcd: Kümeye ait tüm yapılandırma verilerini ve durum bilgilerini saklar.

Controller Manager: ReplicationController, Endpoints Controller, Namespace Controller gibi çeşitli kontrol döngülerini içerir.

Scheduler: Yeni pod'ların hangi node'da çalışacağına karar verir.

Görevleri

Kümeyi yönetir ve izler. Kaynak tahsisini düzenler.

Yeni pod'ları başlatır ve mevcut pod'ların durumunu izler.

Worker Node

Worker Node Bileşenleri

Kubelet: Pod'ların çalışmasını sağlar ve durumu hakkında bilgi toplar.

Kube Proxy: Ağ trafiğini yönlendirir ve dağıtır.

Container Runtime: Pod'ları çalıştırmak için kullanılan konteyner motoru (Docker, containerd, CRI-O).

Görevleri

Pod'ları çalıştırmak ve yönetmek.

Kaynak kullanımı ve performansını izlemek.

API server ile iletişim kurarak komutları uygulamak.

Workload Tipleri: Pod

Pod Nedir?

Kubernetes'in en küçük ve en basit dağıtım birimi.

Bir veya birden fazla konteyner içerebilir.

Özellikleri

Paylaşılan Ağ: Konteynerler aynı IP adresini ve port alanını paylaşır.

Paylaşılan Depolama: Konteynerler aynı depolama birimlerini paylaşabilir.

Yaşam Döngüsü: Pod'lar geçici yapılardır ve bir node'da çalışır.

Workload Tipleri: Deployment

Deployment Nedir?

Pod'ların birden fazla kopyasını yönetmek için kullanılır.

Uygulama güncellemeleri ve ölçeklendirmeleri için bir kontrol mekanizması sağlar.

Özellikleri

Özellikler: Yüksek erişilebilirlik, otomatik geri dönüş (rollback), kademeli dağıtım (rolling update).

Yönetim: Pod'ların istenen durumu ve mevcut durumu arasında uyum sağlar.

Workload Tipleri: StatefulSet

StatefulSet Nedir?

Durum bilgisi olan uygulamalar için kullanılır.

Özellikleri

Kararlı Kimlik: Her pod benzersiz bir kimlik ile oluşturulur ve korunur.

Siparişli Dağıtım: Pod'lar belirli bir sırayla başlatılır ve durdurulur.

Kalıcı Depolama: Her pod kalıcı bir depolama alanı ile ilişkilendirilir.

Kullanım Alanları

Veritabanı Sistemleri, Depolama Sistemleri (Minio)

Workload Tipleri: DaemonSet

DaemonSet Nedir?

Her node'da bir pod çalıştırmak için kullanılır.

Özellikleri

Kapsama: Her node'da aynı pod'un bir kopyasını çalıştırır.

Güncellemeler: Yeni node'lar eklendiğinde otomatik olarak pod dağıtır.

Kullanım Alanları: Sistem monitörleri, log toplama ajanları, ağ izleme araçları.

Workload Tipleri: Job

Job Nedir?

Belirli bir görevi tamamlamak için bir veya daha fazla pod çalıştırır.

Özellikleri

Tek Seferlik Görevler: Görev tamamlandığında pod'lar durdurulur.

Yeniden Başlatma: Başarısız olan pod'lar yeniden başlatılır.

Kullanım Alanları: Veri işleme görevleri, yedekleme işleri.

Workload Tipleri: CronJob

CronJob Nedir?

Belirli zamanlarda veya aralıklarla tekrarlanan işler için kullanılır.

Özellikleri

Zamanlama: Cron formatında zamanlama belirtilebilir.

Otomatik Yönetim: Belirli zamanlarda otomatik olarak job çalıştırır.

Kullanım Alanları: Periyodik veri işleme, rapor oluşturma, bakım görevleri.

Service ve Servis Türleri

Service Nedir?

Pod'lar arasında ağ trafiğini yöneten ve yük dengeleyen bir soyutlama katmanıdır.

Servis Türleri

ClusterIP: İç ağda erişim sağlar.

NodePort: Her node'da belirli bir port açarak dış erişim sağlar.

LoadBalancer: Bulut sağlayıcıların yük dengeleyicilerini kullanarak dış erişim sağlar.

ExternalName: DNS adını bir hizmete yönlendirir.

Ingress Nedir ve Ne Amaçla Kullanılır?

Ingress Nedir?

HTTP ve HTTPS isteklerini kümedeki hizmetlere yönlendiren bir nesnedir.

Özellikler

Yönlendirme: URL yoluna veya ana makine adına göre yönlendirme yapar.

TLS/SSL Desteği: HTTPS için sertifika yönetimi sağlar.

Yük Dengeleme: Trafiği birden fazla hizmet arasında dengeleyebilir.

Kullanım Alanları

HTTP/HTTPS yük dengeleme. Gelişmiş yönlendirme ve trafiği yönetme. Güvenli iletişim sağlama.

Network Policies Nedir ve Ne Amaçla Kullanılır?

Network Policies Nedir?

Kubernetes'te ağ trafiğini kontrol etmek için kullanılan bir güvenlik mekanizmasıdır.

Özellikler

İzin Verme ve Engelleme: Belirli pod'lar arasında ağ trafiğine izin verebilir veya engelleyebilir.

İzole Ağlar: Pod'lar arasında izole ağlar oluşturabilir.

Güvenlik: İç ve dış trafiği sınırlandırarak güvenliği artırır.

Kullanım Amaçları

Mikroservisler arasında güvenli iletişim sağlama. Trafiği kontrol etme ve sınırlama. Ağ saldırılarını önleme.

Service Discovery Nedir?

Service Discovery Nedir?

Hizmetlerin birbirini bulmasını ve iletişim kurmasını sağlar.

Kubernetes'te Service Discovery

DNS: Kubernetes DNS, hizmetler için DNS kayıtları oluşturur ve yönetir.

Environment Variables: Pod'lar başlatıldığında hizmet bilgileri çevre değişkenleri olarak eklenir.

Özellikler

Otomatik Kayıt: Yeni hizmetler otomatik olarak keşfedilir.

Dinamik Güncelleme: Hizmet bilgileri dinamik olarak güncellenir.

Uygulama Yaşam Döngüsü Yönetimi

Yaşam Döngüsü Yönetimi

Başlatma: Uygulama pod'larının oluşturulması ve başlatılması.

İzleme: Pod'ların durumu ve performansının izlenmesi.

Güncelleme: Uygulamaların kademeli veya anında güncellenmesi.

Ölçeklendirme: Trafik veya yük yoğunluğuna göre pod'ların ölçeklendirilmesi.

Durdurma ve Silme: Uygulamaların düzgün bir şekilde durdurulması ve silinmesi.

Özellikler

Otomatik Yeniden Başlatma: Başarısız olan pod'ların otomatik olarak yeniden başlatılması.

Durum Kontrolleri: Uygulamaların sağlık durumunu kontrol etmek için liveness ve readiness prob'ları.

Geri Dönüş: Hatalı güncellemelerde otomatik geri dönüş mekanizması.

Uygulama Yaşam Döngüsü Yönetimi - Liveness Probe

Tanım

Liveness Probe, Kubernetes'in bir pod'un çalışıp çalışmadığını kontrol etmesini sağlar.

Uygulama içinde bir hata oluştuğunda Kubernetes'in bu pod'u yeniden başlatmasına olanak tanır.

Özellikleri

Health Check: Uygulamanın canlı olup olmadığını kontrol eder.

Otomatik Yeniden Başlatma: Eğer bir pod liveness probe kontrolünden geçemezse, Kubernetes pod'u yeniden başlatır.

Çeşitli Kontrol Yöntemleri: HTTP, TCP ve komut tabanlı kontroller kullanılabilir.

Uygulama Yaşam Döngüsü Yönetimi - Readiness Probe

Tanım

Readiness Probe, Kubernetes'in bir pod'un trafiği alıp alamayacağını kontrol etmesini sağlar.

Pod hazır değilse, Kubernetes bu pod'a trafik yönlendirmez.

Özellikleri

Traffic Management: Pod'un uygulama trafiğini almaya hazır olup olmadığını kontrol eder.

Uygulama Hazırlığı: Uygulamanın tam olarak başlatılıp başlatılmadığını belirler.

Çeşitli Kontrol Yöntemleri: HTTP, TCP ve komut tabanlı kontroller kullanılabilir.

Deployment Stratejileri Nedir?

Tanım

Deployment stratejileri, Kubernetes'te uygulamaların nasıl dağıtılacağını ve güncellemelerin nasıl yapılacağını belirleyen yöntemlerdir.

Uygulama güncellemeleri sırasında kesintisiz hizmet sağlamak ve riski minimize etmek için kullanılır.

Özellikleri

Kesintisiz Dağıtım: Kullanıcılar için minimum kesinti ile uygulama güncellemeleri sağlar.

Geri Alma Mekanizmaları: Hatalı dağıtımları geri alma imkanı sunar.

Ölçeklendirme: Yeni sürümlerin aşamalı olarak dağıtılmasına olanak tanır.

Recreate Stratejisi

Tanım

Tüm eski pod'ları durdurur ve ardından yeni pod'ları başlatır.

Özellikler

Basitlik: En temel dağıtım stratejisidir.

Kesinti: Kullanıcılar için kısa süreli kesintiye neden olabilir.

Kullanım Alanları: Küçük ve hızlı başlatılabilen uygulamalar için uygun.

Rolling Update Stratejisi

Tanım

Yeni pod'ları aşamalı olarak başlatır ve eski pod'ları aşamalı olarak durdurur.

Özellikleri

Kesintisiz Dağıtım: Kullanıcılar için minimum kesinti sağlar.

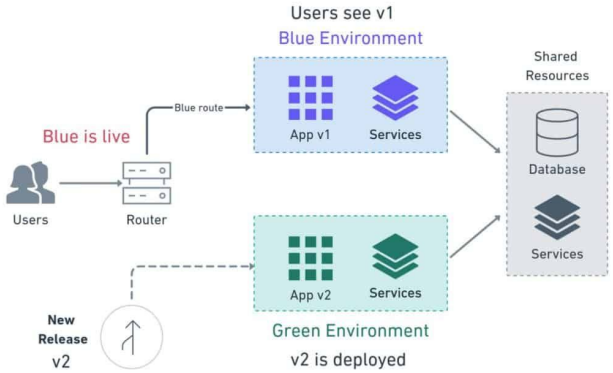
Kontrollü Geçiş: Aşamalı geçiş ile riski azaltır.

Geri Alma: Hatalı dağıtımlarda kolayca geri dönülebilir.

Blue-Green Deployment Stratejisi

Tanım

Yeni bir sürüm (blue) başlatılır ve eski sürüm (green) ile paralel olarak çalışır. Yeni sürüm hazır olduğunda, trafiği yeni sürüme yönlendirir.



Özellikleri

Kesintisiz Geçiş: Kullanıcılar için kesintisiz geçiş sağlar.

Hızlı Geri Dönüş: Hatalı dağıtımlarda hızlı geri dönüş imkanı.

Kaynak Kullanımı: Geçici olarak iki kat kaynak kullanımı gerektirir.

Örnek Adımlar

Yeni sürümü başlatın ve test edin (blue). Trafiği yeni sürüme yönlendirin. Eski sürümü kapatın (green).

Canary Deployment Stratejisi

Tanım

Yeni sürümü küçük bir kullanıcı grubuna dağıtarak başlar ve kademeli olarak tüm kullanıcılara yayar.

Özellikleri

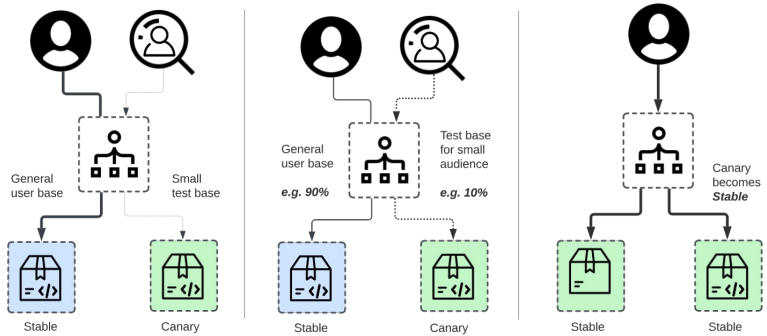
Risk Azaltma: Hatalı sürümleri erken tespit etme imkanı.

Kademeli Geçiş: Yeni sürüm kademeli olarak tüm kullanıcılara yayılır.

Geri Alma: Sorun tespit edilirse, dağıtım durdurulabilir veya geri alınabilir.

Örnek Adımlar

Yeni sürümü küçük bir kullanıcı grubuna dağıtın. Performansı izleyin ve sorunları tespit edin. Kademeli olarak dağıtımı genişletin.



A/B Testing Deployment Stratejisi

Tanım

İki veya daha fazla sürümü aynı anda çalıştırarak farklı kullanıcı gruplarına yönlendirir ve performansı karşılaştırır.

Özellikleri

Karşılaştırma: Farklı sürümlerin performansını ve kullanıcı tepkilerini karşılaştırma imkanı.

Kullanıcı Geri Bildirimi: Gerçek kullanıcı verileri ile en iyi sürümü belirleme.

Karmaşıklık: Yönlendirme ve izleme daha karmaşık olabilir.

Örnek Adımlar

İki farklı sürümü aynı anda çalıştırın. Kullanıcıları farklı sürümlere yönlendirin. Performansı ve kullanıcı tepkilerini izleyin. En iyi performans gösteren sürümü seçin.

Kubernetes'te Storage Yönetimi Nedir?

Tanım

Kubernetes, uygulamaların veri saklama ihtiyaçlarını karşılamak için çeşitli depolama çözümleri sunar.

Kalıcı verilerin yönetimi, taşınabilirlik ve ölçeklenebilirlik sağlar.

Özellikleri

Kalıcı Depolama: Uygulama pod'ları yeniden başlatılsa bile veriler kalıcı olarak saklanır.

Esneklik: Çeşitli depolama türleri ve sağlayıcıları ile çalışabilir.

Dinamik Yönetim: Depolama kaynakları otomatik olarak oluşturulabilir ve yönetilebilir.

Depolama Türleri

Ephemeral (Geçici) Depolama

Pod'ların yaşam süresi boyunca geçici olarak veri saklar.

Pod yeniden başlatıldığında veriler kaybolur.

Kullanım Alanları: Geçici dosyalar, önbellekler.

Persistent (Kalıcı) Depolama

Pod'ların yaşam süresinden bağımsız olarak verileri saklar.

Pod yeniden başlatılsa bile veriler kalıcıdır.

Kullanım Alanları: Veritabanları, dosya sistemleri.

Depolama Sağlayıcıları

Yerel Depolama: Node'ların yerel disklerini kullanır.

Ağ Depolama: NFS, GlusterFS, Ceph, AWS EBS, GCP Persistent Disks, Azure Disks vb.

Persistent Volume (PV) ve Persistent Volume Claim (PVC)

Persistent Volume (PV)

Kubernetes kümesinde kalıcı depolama sağlamak için kullanılan kaynak.

Yönetici tarafından oluşturulur ve küme genelinde kullanıma sunulur.

Persistent Volume Claim (PVC)

Pod'ların ihtiyaç duyduğu kalıcı depolama kaynaklarını talep etmek için kullanılan nesne.

Kullanıcı tarafından oluşturulur ve uygun bir PV ile eşleştirilir.

StorageClass

Tanım

Farklı depolama gereksinimlerini karşılamak için kullanılacak depolama türlerini ve politikalarını tanımlar.

Dinamik olarak PV oluşturmayı sağlar.

Özellikleri

Dinamik Provisioning: Kullanıcı taleplerine göre otomatik olarak PV oluşturur.

Çeşitli Depolama Seçenekleri: Farklı depolama sağlayıcıları ve performans düzeyleri sunar.

Politika Yönetimi: Depolama politikalarını ve ayarlarını belirler.

Dynamic Volume Provisioning

Tanım

PVC oluşturulduğunda otomatik olarak uygun bir PV oluşturur.

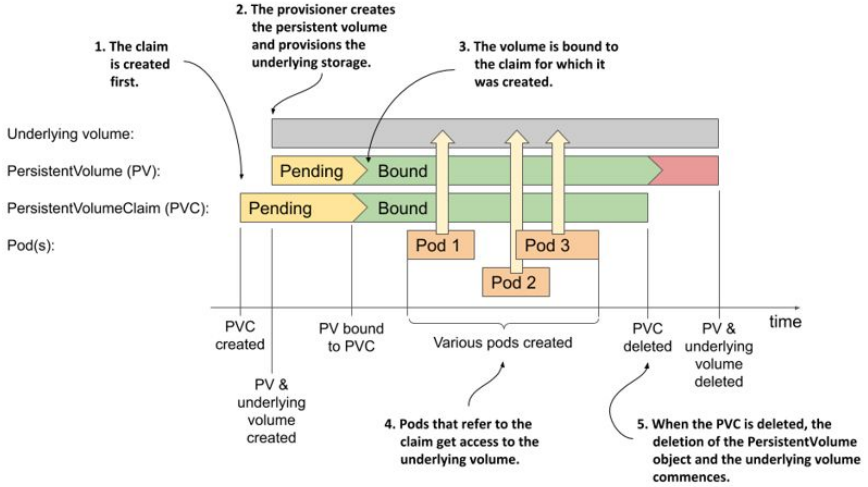
Kullanıcıların manuel olarak PV oluşturmasına gerek kalmaz.

Özellikleri

Otomasyon: PV'lerin otomatik olarak oluşturulması ve yönetilmesi.

Esneklik: Farklı depolama gereksinimleri için uygun PV'lerin dinamik olarak sağlanması.

Kolay Yönetim: Kullanıcıların PVC oluşturarak depolama kaynaklarına kolayca erişebilmesi.



StatefulSet ile Kalıcı Depolama

StatefulSet Nedir?

Durum bilgisi olan uygulamalar için kullanılan Kubernetes nesnesi.

Her pod için benzersiz ve kalıcı depolama sağlar.

Özellikleri

Kararlı Kimlik: Her pod benzersiz bir kimlik ile oluşturulur ve korunur.

Kalıcı Depolama: Her pod'a özgü kalıcı depolama sağlanır.

Siparişli Dağıtım: Pod'lar belirli bir sırayla başlatılır ve durdurulur.

Volumes ve VolumeMounts

Volumes Nedir?

Pod'lara eklenebilen depolama kaynakları.

Çeşitli depolama türleri: EmptyDir, HostPath, ConfigMap, Secret, NFS, vs.

VolumeMounts Nedir?

Pod'lar içindeki konteynerlerin volume'ları bağladığı yerler.

Her konteyner, volume'ları belirli bir dosya yoluna bağlayabilir.

Persistent Storage ile Yedekleme ve Geri Yükleme

Yedekleme

Kalıcı verilerin düzenli olarak yedeklenmesi.

Yedekleme araçları: Velero, Heptio Ark, vs.

Geri Yükleme

Yedeklerden verilerin geri yüklenmesi.

Veritabanı yedekleri, dosya yedekleri, vs.

Velero kullanarak Kubernetes kaynaklarını ve PV'leri yedekleme ve geri yükleme.

velero backup create my-backup --include-namespaces my-namespace

velero restore create --from-backup my-backup

Kubernetes'te Uygulama Konfigürasyonu

Tanım

Kubernetes, uygulama konfigürasyonlarını yönetmek için çeşitli yöntemler sunar.

Uygulama konfigürasyonlarını dışsallaştırarak yönetilebilir ve yeniden kullanılabilir hale getirir.

Özellikleri

Modülerlik: Konfigürasyonları uygulama kodundan ayırır.

Taşınabilirlik: Farklı ortamlar arasında konfigürasyonları kolayca taşır.

Güvenlik: Hassas verileri güvenli bir şekilde yönetir.

Kullanım Alanları

Uygulama ayarları, yapılandırma dosyaları, çevresel değişkenler.

ConfigMap Nedir?

Tanım

ConfigMap, uygulama konfigürasyon verilerini anahtar-değer çiftleri olarak saklayan bir Kubernetes nesnesidir.

Uygulama yapılandırmalarını çevresel değişkenler, komut satırı argümanları veya konfigürasyon dosyaları olarak pod'lara enjekte edebilir.

Özellikler

Anahtar-Değer Çiftleri: Yapılandırma verilerini anahtar-değer çiftleri olarak saklar.

Esneklik: Konfigürasyon verilerini dosya, dizin veya çevresel değişken olarak sağlayabilir.

Paylaşılabilirlik: Birden fazla pod arasında aynı konfigürasyonları paylaşabilir.

Secret Nedir?

Tanım

Secret, hassas verileri (şifreler, API anahtarları, sertifikalar) güvenli bir şekilde saklamak ve yönetmek için kullanılan bir Kubernetes nesnesidir.

Verileri base64 kodlaması ile saklar ve pod'lara güvenli bir şekilde enjekte eder.

Özellikleri

Güvenlik: Hassas verileri şifreli bir şekilde saklar.

Erişim Kontrolü: Hassas verilere erişimi kısıtlar ve yönetir.

Çevresel Değişkenler ve Dosyalar: Verileri çevresel değişkenler veya dosyalar olarak pod'lara enjekte eder.

ConfigMap ve Secret Kullanımının Avantajları

ConfigMap Avantajları

Modülerlik: Uygulama konfigürasyonlarını koddan ayırır.

Paylaşılabilirlik: Birden fazla pod arasında aynı konfigürasyonları paylaşabilir.

Güncelleme Kolaylığı: Uygulama yeniden başlatılmadan konfigürasyonları güncelleyebilir.

Secret Avantajları

Güvenlik: Hassas verileri güvenli bir şekilde saklar ve yönetir.

Erişim Kontrolü: Hassas verilere erişimi kısıtlar ve yönetir.

Çeşitli Kullanım Yöntemleri: Çevresel değişkenler veya dosyalar olarak pod'lara enjekte edebilir.

Ortak Avantajlar

Esneklik: Konfigürasyonları ve hassas verileri farklı yöntemlerle pod'lara enjekte edebilir.

Yönetim Kolaylığı: Konfigürasyon ve hassas verileri merkezi olarak yönetir.

Güvenlik ve Uygulama Yönetimi: Uygulama konfigürasyonları ve hassas veriler için güvenli ve yönetilebilir bir çözüm sunar.

K3s Kurulumu

https://docs.k3s.io/quick-start adresindeki yönergeler izlenebilir.

curl -sfL https://get.k3s.io | sh -

kubectl get pods

kubectl get logs -f <pod_name>

kubectl get crd

kubectl get nodes

(Kubernetes-Cheat-Sheet)

Kubectl Nedir?

Tanım

Kubectl, Kubernetes kümelerini yönetmek için kullanılan komut satırı aracıdır.

Kubernetes API'si ile etkileşimde bulunarak kümelerdeki kaynakları oluşturur, yapılandırır ve yönetir.

Özellikleri

Komut Satırı Yönetimi: Kubernetes kaynaklarını CLI üzerinden yönetir.

Çok Yönlülük: Tüm Kubernetes nesneleri üzerinde operasyonlar gerçekleştirebilir.

Otomasyon: Skriptler ve CI/CD süreçleri için kullanılabilir.

Kubectl Temel Komutlar

kubectl get <resource>: Belirli bir kaynak türünün listesini alır.

kubectl get pods

kubectl get services

kubectl get deployments

kubectl describe <resource> <name>: Belirli bir kaynağın ayrıntılarını gösterir.

kubectl describe pod my-pod

kubectl describe service my-service

kubectl apply -f <file>: YAML veya JSON dosyalarından kaynak oluşturur veya günceller.

kubectl apply -f deployment.yaml

kubectl delete <resource> <name>: Belirli bir kaynağı siler.

kubectl delete pod my-pod

Kubectl İleri Düzey Komutlar

kubectl logs my-pod

kubectl exec my-pod -- ls /app

kubectl port-forward <pod/service> <local-port>:<remote-port>: Bir pod veya servis portunu yerel bilgisayara yönlendirir.

kubectl port-forward svc/my-service 8080:80

kubectl proxy: Yerel bir HTTP proxy başlatır ve Kubernetes API'sine erişim sağlar.

kubectl proxy

kubectl label <resource> <name> <label-key>=<label-value>: Bir kaynağa etiket ekler.

kubectl label pod my-pod environment=production

kubectl get <resource> -l <label-key>=<label-value>: Etikete göre kaynakları seçer.

kubectl get pods -l environment=production

Kubens Nedir?

Tanım:

Kubens, Kubernetes namespace'leri arasında hızlı geçiş yapmak için kullanılan bir komut satırı aracıdır.

Kubernetes kümesindeki farklı namespace'ler arasında kolayca geçiş yapmayı sağlar.

Özellikler:

Hızlı Geçiş: Namespace'ler arasında hızlı ve kolay geçiş.

Kullanım Kolaylığı: Basit ve sezgisel komutlar.

Entegrasyon: Kubectl ile entegre çalışır.

Kullanım Alanları:

Farklı projeler veya ortamlarda çalışırken namespace geçişi. İzole edilmiş çalışma alanları arasında hızlı geçiş.

Kubens Kurulumu ve Kullanımı

https://github.com/ahmetb/kubectx /opt/kubectx repository adresinden kubens ve kubectx indirilebilir.

sudo git clone https://github.com/ahmetb/kubectx /opt/kubectx

sudo ln -s /opt/kubectx/kubens /usr/local/bin/kubens

kubens default

Kubectx Nedir?

Tanım

Kubectx, Kubernetes kümeleri (clusters) arasında hızlı geçiş yapmak için kullanılan bir komut satırı aracıdır.

Kubernetes bağlamlarını (contexts) yönetmek ve değiştirmek için kullanılır.

Özellikler

Hızlı Geçiş: Kubernetes kümeleri arasında hızlı ve kolay geçiş sağlar.

Kullanım Kolaylığı: Basit ve sezgisel komutlar sunar.

Entegrasyon: Kubectl ile entegre çalışır.

Kullanım Alanları

Farklı projeler veya ortamlarda çalışırken kümeler arasında geçiş yapmak.

Geliştirme, test ve üretim ortamları arasında hızlı geçiş.

Kubectx Kurulumu ve Kullanımı

https://github.com/ahmetb/kubectx /opt/kubectx repository adresinden kubens ve kubectx indirilebilir.

sudo git clone https://github.com/ahmetb/kubectx /opt/kubectx

sudo ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx

kubectx default

Helm Nedir?

Tanım

Helm, Kubernetes uygulamalarını paketlemek, dağıtmak ve yönetmek için kullanılan bir araçtır.

Kubernetes için bir "paket yöneticisi" olarak düşünülebilir.

Özellikler

Paketleme: Uygulamaları Chart olarak paketler.

Yeniden Kullanılabilirlik: Aynı Chart farklı ortamlarda kullanılabilir.

Versiyonlama: Uygulama sürümlerini yönetir ve izler.

Kullanım Alanları

Karmaşık uygulama dağıtımları, Uygulama güncellemeleri ve sürüm yönetimi, Ortamlar arası taşıma

Helm Chart Nedir?

Tanım

Helm Chart, bir Kubernetes uygulamasını tanımlayan dosyalar ve dizinlerin bir koleksiyonudur.

Uygulamanın tüm kaynaklarını ve yapılandırmalarını içerir.

Chart Bileşenleri

Chart.yaml: Chart meta verilerini içerir.

values.yaml: Varsayılan yapılandırma değerlerini içerir.

templates/: Kubernetes manifest şablonlarını içerir.

charts/: Bağımlı Chart'ları içerir.

README.md: Chart hakkında bilgi verir.

Helm Kurulumu ve Temel Komutlar

Helm kurulumu gerçekleştirilir.

Temel Komutlar

helm create <chart-name>: Yeni bir Chart oluşturur.

helm install <chart>: Bir Chart'ı kümede dağıtır.

helm upgrade <release> <chart>: Var olan bir dağıtımı günceller.

helm delete <release>: Bir dağıtımı siler.

helm repo add <repo-name> <repo-url>: Yeni bir depo ekler.

helm list: Tüm dağıtımları listeler.

Helm ile PostgreSQL Kurulumu

Tanım

Helm kullanarak PostgreSQL veritabanını Kubernetes kümesine kolayca kurabilirsiniz.

Helm, PostgreSQL gibi karmaşık uygulamaların Kubernetes üzerinde yönetimini basitleştirir.

Özellikler

Kolay Kurulum: Tek bir komutla PostgreSQL kurulumunu yapabilirsiniz.

Yapılandırma: ConfigMap ve Secret kullanarak PostgreSQL yapılandırmalarını kolayca yönetebilirsiniz.

Güncellemeler: Uygulamayı kolayca güncelleyebilir ve sürüm yönetimi yapabilirsiniz.

Helm ve PostgreSQL Chart Hazırlığı

Helm Repository Ekleme

Helm chart repository'leri ekleyerek PostgreSQL chart'ını kullanabilirsiniz.

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

PostgreSQL Chart Arama

Eklenen repository'den PostgreSQL chart'ını arayabilirsiniz.

helm search repo postgresql

PostgreSQL Kurulumu için Helm Komutu

Temel Kurulum

Helm ile PostgreSQL'i kurmak için aşağıdaki komutu kullanabilirsiniz.

helm install my-postgresql bitnami/postgresql

Özelleştirilmiş Kurulum

Values.yaml dosyasını kullanarak kurulum yapılandırmalarını özelleştirebilirsiniz.

helm install my-postgresql bitnami/postgresql -f custom-values.yaml

PostgreSQL Kurulumu Sonrası Yapılandırma

Kullanıcı ve Parola Bilgilerini Almak

Kurulum sonrası PostgreSQL kullanıcı ve parola bilgilerini almak için aşağıdaki komutu kullanabilirsiniz.

export POSTGRESQL_PASSWORD=$(kubectl get secret --namespace default my-postgresql -o jsonpath="{.data.postgresql-password}" | base64 --decode)

PostgreSQL'e Bağlanmak

PostgreSQL pod'una bağlanarak veritabanı yönetim komutlarını çalıştırabilirsiniz.

kubectl run my-postgresql-client --rm --tty -i --restart='Never' --namespace default --image bitnami/postgresql --env="PGPASSWORD=$POSTGRESQL_PASSWORD" --command -- psql --host my-postgresql -U myuser -d mydatabase

Güncellemeler ve Yükseltmeler

Helm ile PostgreSQL Güncellemeleri

PostgreSQL chart'ını güncellemek için aşağıdaki komutu kullanabilirsiniz.

helm upgrade my-postgresql bitnami/postgresql

Values.yaml Değişiklikleri ile Güncelleme

Değişiklik yaptığınız Values.yaml dosyasını kullanarak PostgreSQL'i güncelleyebilirsiniz.

helm upgrade my-postgresql bitnami/postgresql -f custom-values.yaml

Güncelleme Sonrası Durumu Kontrol Etme

Güncelleme sonrası PostgreSQL pod'larının durumunu kontrol etmek için aşağıdaki komutu kullanabilirsiniz.

kubectl get pods -l app=my-postgresql

PostgreSQL Yedekleme ve Geri Yükleme

Yedekleme

PostgreSQL veritabanını yedeklemek için pg_dump komutunu kullanabilirsiniz.

kubectl exec -it my-postgresql-0 -- pg_dump -U myuser -d mydatabase > mydatabase_backup.sql

Geri Yükleme

Yedeklenen veritabanını geri yüklemek için pg_restore komutunu kullanabilirsiniz.

kubectl exec -i my-postgresql-0 -- psql -U myuser -d mydatabase < mydatabase_backup.sql

kubectl exec -it my-postgresql-0 -- pg_dumpall -U myuser > all_databases_backup.sql

kubectl exec -i my-postgresql-0 -- psql -U myuser -f all_databases_backup.sql

Kubernetes Üzerinde PostgreSQL Kurulumu (Manuel)

Tanım

Kubernetes üzerinde PostgreSQL kurulumu, manuel olarak YAML dosyaları ile yapılandırılarak gerçekleştirilir.

StatefulSet, PersistentVolume, PersistentVolumeClaim, ve Service kullanılarak PostgreSQL yapılandırılır.

Özellikler

Esneklik: Özelleştirilebilir ve yönetilebilir.

Kontrol: Tüm yapılandırmalar manuel olarak yapılır, bu sayede daha fazla kontrol sağlar.

Öğrenme: Kubernetes kaynaklarını ve bileşenlerini daha iyi anlama fırsatı sunar.

PersistentVolume ve PersistentVolumeClaim

PersistentVolume (PV)

Kalıcı depolama sağlar ve düğümler arasında paylaştırılabilir.

Kurulum için YAML dosyası oluşturulur.

PersistentVolumeClaim (PVC)

Pod'ların ihtiyaç duyduğu depolama kaynaklarını talep eder.

Kurulum için YAML dosyası oluşturulur.

PostgreSQL Deployment ve Service

Deployment

PostgreSQL pod'larını oluşturur ve yönetir.

Kurulum için YAML dosyası oluşturulur.

Service

PostgreSQL'e dışarıdan erişimi sağlar.

Kurulum için YAML dosyası oluşturulur.

PostgreSQL Konfigürasyon ve Çalıştırma

Adım 1: PersistentVolume Oluşturma:

PV ve PVC YAML dosyalarını uygulayın.

kubectl apply -f pv-postgresql.yaml

kubectl apply -f pvc-postgresql.yaml

Adım 2: Deployment ve Service Oluşturma:

PostgreSQL Deployment ve Service YAML dosyalarını uygulayın.

kubectl apply -f postgresql-deployment.yaml

kubectl apply -f postgresql-service.yaml

Adım 3: Durumu Kontrol Etme

PostgreSQL pod'larının ve servisinin durumunu kontrol edin.

kubectl get pods

kubectl get services

Örnek Komutlar

kubectl get pvc

kubectl describe pod postgresql

PostgreSQL'e Erişim ve Yönetim

PostgreSQL Pod'una Erişim

PostgreSQL pod'una erişmek için aşağıdaki komutu kullanın.

kubectl exec -it <postgresql-pod-name> -- psql -U myuser -d mydatabase

Yedekleme

kubectl exec -it <postgresql-pod-name> -- pg_dump -U myuser mydatabase > mydatabase_backup.sql

Geri Yükleme

kubectl exec -i <postgresql-pod-name> -- psql -U myuser mydatabase < mydatabase_backup.sql


thank you
danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
dank je
gracias
tapadh leat
хвала
asante
manana
obrigada
tenki
kiitos
dankie
nanni
nandri
dhanyavad
bedankt
hvala
mauruuru
köszönöm
djiere dieuf
mochchakkeram
enkosi
bayarlalaa
grâcie
akun
dankon
ačiū
dziękuję
chnorakaloutioun
gratias ago
gràcies
sulpáy
дякую
mamnun
murakoze
chokrane
go raibh maith agat
sobodi
dėkuji
sukriya
kop khun krap
ţaiku
obrigado
sagolun
arigatô
takk
dakujem
trugarez
mèsi
ありがとう
tanemirt
grazie
najis tuke
terima kasih
rahmet
merci
didi madloba
kam sah hamnida
rahmat
diolch
dhanyavadagalu
shukriya
мерси
merci
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ