

Kubernetes





Başlamadan Önce Gereksinimler

- <https://kubernetes.io/>
- **minikube:** <https://github.com/kubernetes/minikube>
- **kubectl:** <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- **kind:** <https://kind.sigs.k8s.io/>
- **Docker Desktop:** <https://www.docker.com/products/docker-desktop/>
- **Cloud platform:** (digital ocean, GCP,AWS or Azure)
- **IDE:** visual studio or IntelliJ IDEA...



Kubernetes Nedir?

- Kubernetes, konteynerlerin yönetimi için tasarlanmış bir orkestrasyon platformudur.
- Google tarafından Go programlama dili kullanılarak geliştirilmiştir.
- Google, Kubernetes'i Cloud Native Computing Foundation (CNCF) organizasyonuna bağışlamıştır.
- Kubernetes'in ilk sürümü 2015 yılında piyasaya sürülmüştür.
- Platform, kullanıcıların serbestçe erişip kullanabilecegi açık kaynaklı bir yazılımdır.
- Uygulamaların dağıtımını, ölçeklendirilmesini ve operasyonlarını otomatikleştirir.
- AWS, Azure, GCP ve Digital Ocean gibi birden fazla bulut platformunu destekler.

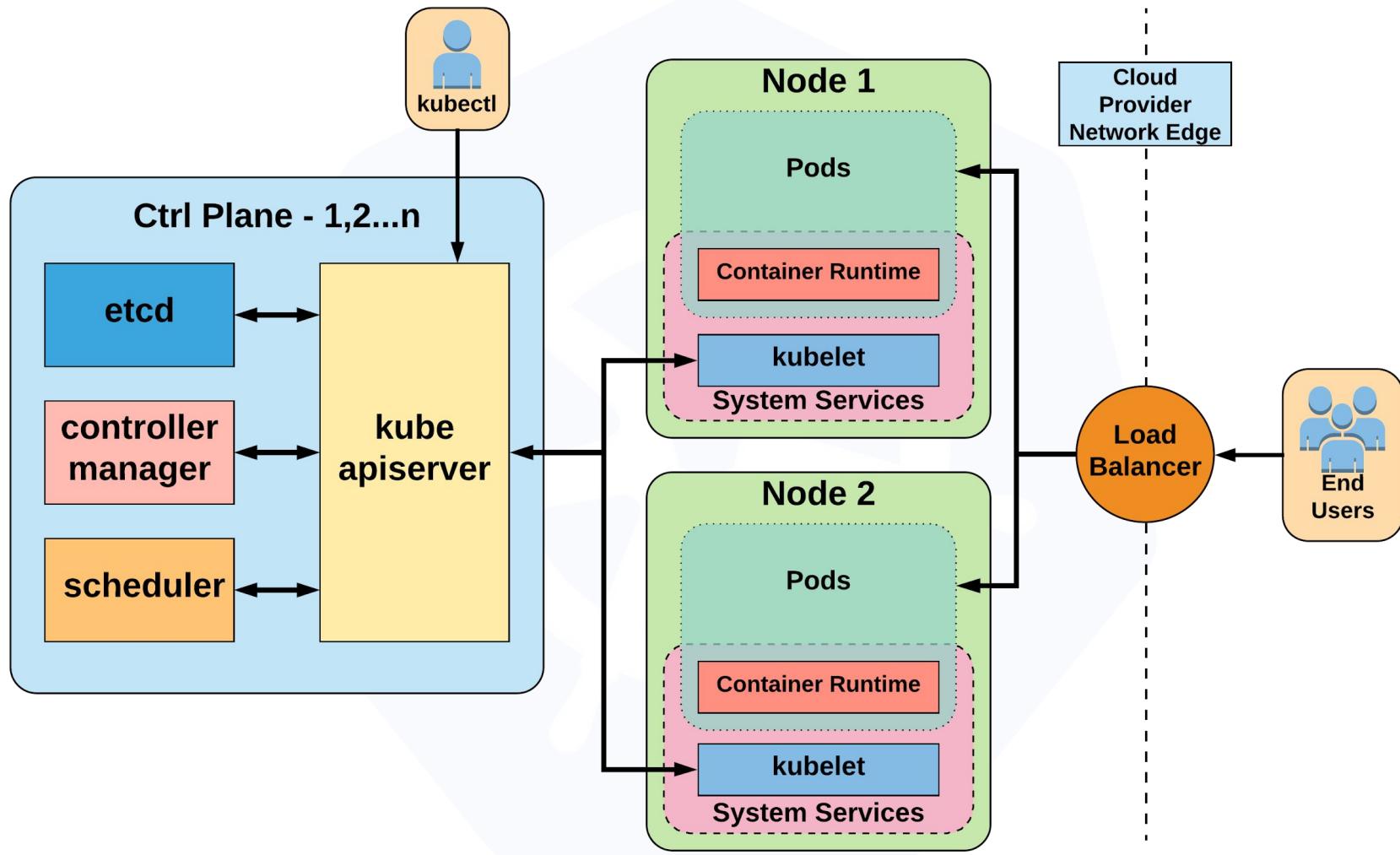


Kubernetes Hangi Problemleri Çözer?

- Container 'ları manuel olarak yönetme zorluğunu ortadan kaldırır
- Otomatik Ölçeklendirme
- Yük Dengeleme
- Yüksek erişilebilirlik ve kesintisiz çalışma
- Kaynakların Verimli Kullanımı



Kubernetes Architecture



Control Plane or Master Node

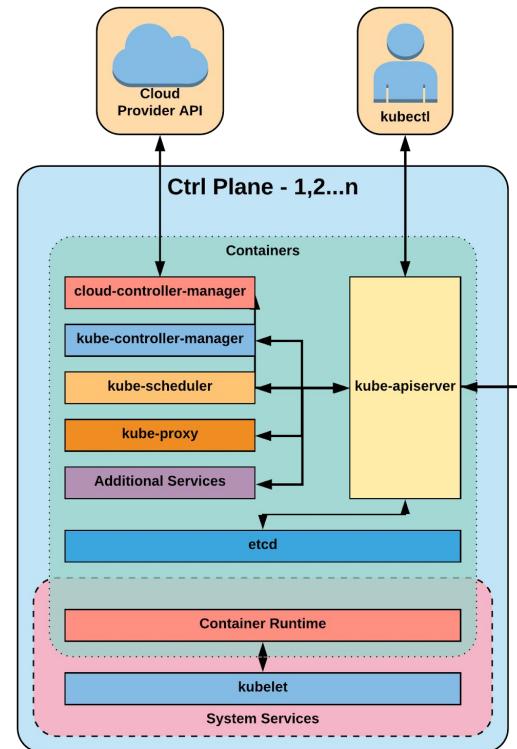


Master Node(Control Plane): Kubernetes kümесinin kontrol merkezidir. Tüm kararlar burada alınır ve cluster yönetimi sağlanır.

Ana görevleri, kümedeki **worker node'ların** yönetimi, **API isteklerinin** işlenmesi, **pod'ların** dağıtılması ve **kaynak yönetimi** gibi işlevleri içerir.

Bileşenler:

- **kube-apiserver**
- **etcd**
- **kube-scheduler**
- **kube-controller-manager**





kube-apiserver

- Kubernetes kontrol düzlemi ve veri deposuna REST tabanlı bir arayüz sağlar.
- Tüm istemciler ve uygulamalar yalnızca API Server üzerinden Kubernetes ile etkileşim kurar.
- Kimlik doğrulama ve yetkilendirme işlemlerini yönetir.
- İstek doğrulama ve mutasyonlarını gerçekleştirir.
- Kabul kontrolü mekanizmalarını çalıştırır.
- Kümeye giriş kapısı görevi görerek, veri deposuna ön yüz sağlar.

etcd



- etcd, Kubernetes'in cluster datastore'u olarak görev yapar.
- Kubernetes ile ilgili amacı, güçlü, tutarlı ve yüksek erişilebilirliğe sahip bir anahtar-değer mağazası sağlamaktır.
- Cluster durumu için kalıcı veri depolama sağlar.
- Pod, Service, ConfigMap, Secret gibi Kubernetes nesnelerini ve yapılandırma bilgilerini saklar.
- Küme durumu ve ağ politikaları gibi kritik bilgilerin güvenli bir şekilde depolanmasından sorumludur.





kube-controller-manager

- Kube-controller manager, tüm çekirdek bileşenlerin kontrol döngülerini yöneten ana daemon'dur.
- API server aracılığıyla cluster durumunu izler ve cluster'in istenen duruma yönlendirilmesini sağlar.
- Farklı kontrol döngüleri (controller loops) çalıştırır, her biri belirli bir Kubernetes nesnesini yönetir (ör. ReplicaSet, Node, Endpoints, vb.).
- İstenilen durum ile mevcut durum arasındaki farkları kontrol eder ve gerekirse müdahale eder.



kube-scheduler

- Kube Scheduler, iş yüklerinin hangi kaynaklarda çalışacağını belirler.
- İş yüklerinin, CPU ve bellek gibi genel donanım gereksinimlerini karşılamasını sağlar.
- Pod'ların belirli nodlarda toplanmasını (affinity) veya dağıtılmmasını (anti-affinity) sağlar, bu da yüksek verimlilik ve yüksek kullanılabilirlik sağlar.
- Pod'ların ve nod'ların etiketlerine göre yerleştirme yaparak, belirli özelliklere sahip kaynakları hedefler.
- Kaynakları etkili bir şekilde kullanarak, cluster'ın performansını ve verimliliğini artırır.

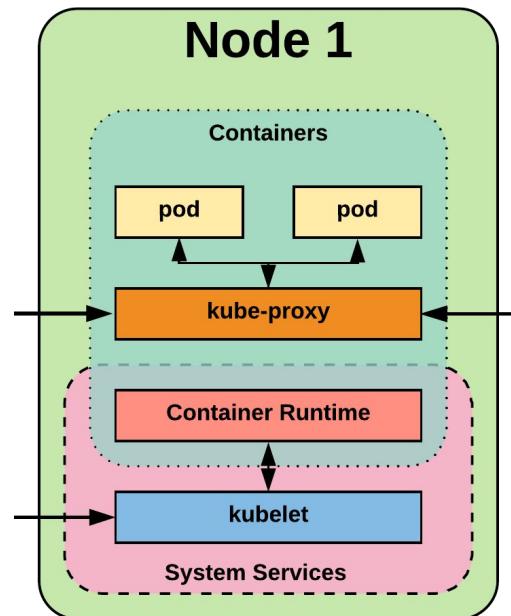


Worker Node

- Worker Nodes, Kubernetes kümesinde konteynerleştirilmiş uygulamaların çalıştığı sunuculardır.
- Her Worker Node, üzerinde bir veya daha fazla Pod çalıştırır ve konteynerlerin ağ, depolama gibi kaynaklara erişimini sağlar.

Bileşenler:

- kubelet:
- kube-proxy:
- Container Runtime:





kubelet

- Kubelet, her bir node üzerindeki pod'ların yaşam döngüsünü yönetmekle sorumlu olan ajandır.
- Kubelet, aşağıdaki kaynaklardan YAML formatında yazılmış konteyner manifestlerini okuyabilir:
 - Dosya Yolu: Konteyner manifestlerinin bulunduğu dosya yolu.
 - HTTP Endpoint: HTTP üzerinden erişilebilen endpoint.
 - etcd İzleme: etcd üzerinden yapılan değişiklikleri izleyerek güncellemeleri alır.
 - HTTP Sunucu Modu: Basit bir API aracılığıyla konteyner manifestlerini kabul eden HTTP sunucusu modu.



kube-proxy

Her bir node üzerindeki ağ kurallarını yönetir.

Kubernetes cluster servisleri için bağlantı yönlendirme ve yük dengeleme işlemlerini gerçekleştirir.

Mevcut Proxy Modları:

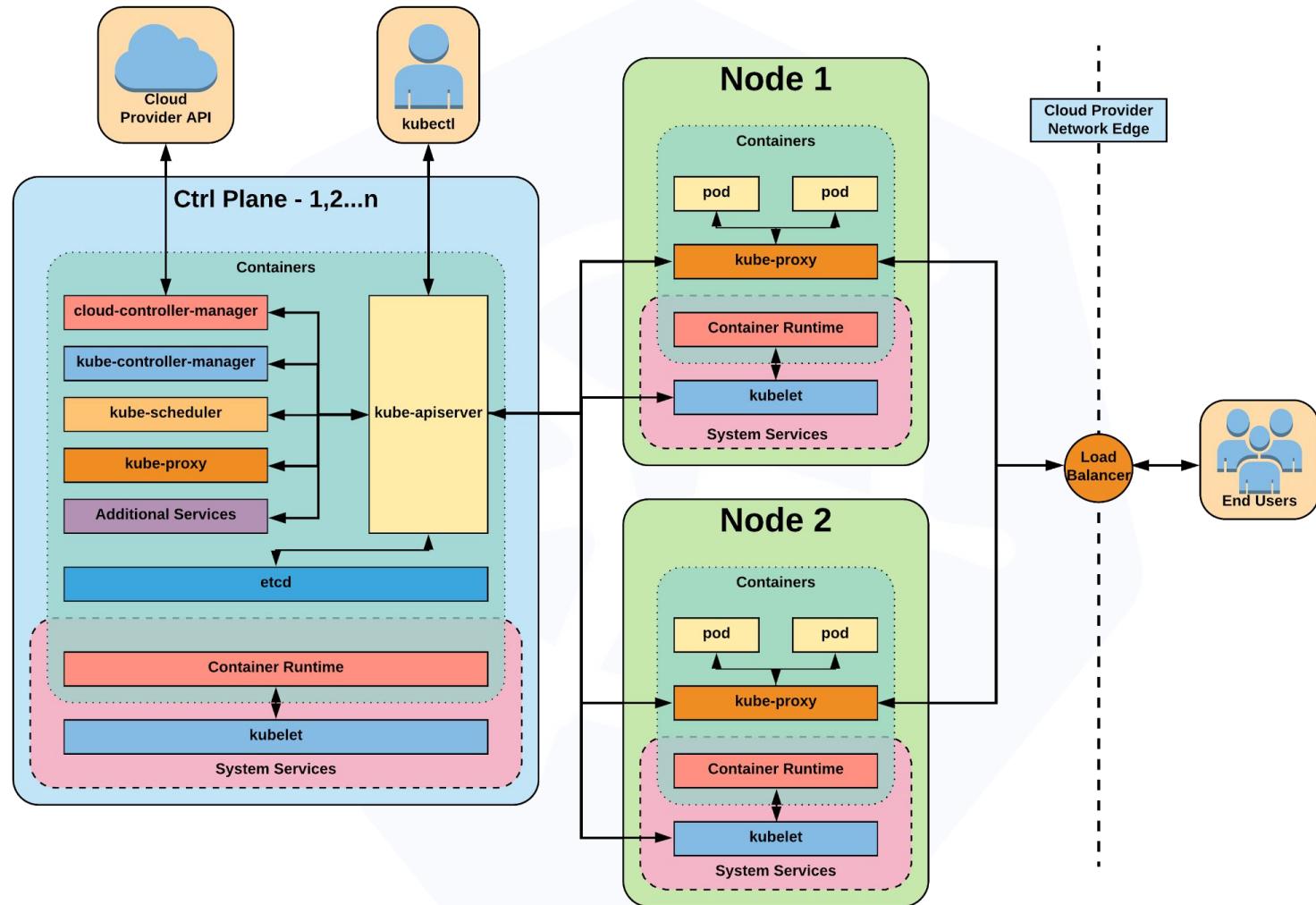
- **Userspace:** Eski ve daha az verimli bir yöntem. Bağlantı yönlendirme ve yük dengeleme işlemleri kullanıcı alanında gerçekleştirilir.
- **iptables:** Ağ kuralları, iptables kullanılarak uygulanır. Daha verimli ve yaygın olarak kullanılan bir yöntemdir.
- **ipvs (Varsayılan, Destekleniyorsa):** IP Virtual Server (ipvs) kullanılarak yapılan yük dengeleme, yüksek performans ve daha fazla özelleştirme seçeneği sunar.

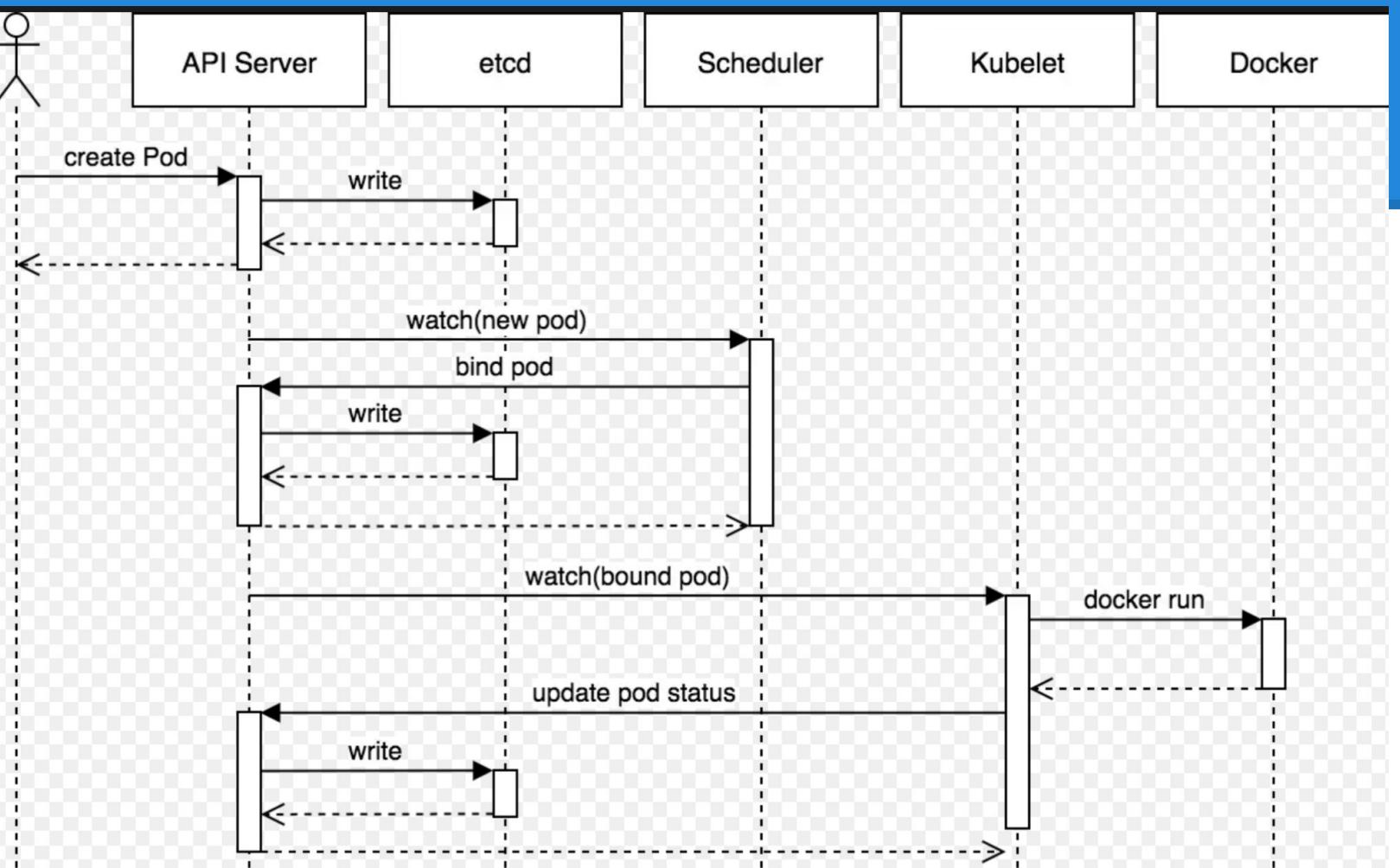


Container Runtime Engine

Container runtime, CRI (Container Runtime Interface) uyumlu bir uygulamadır ve konteynerleri çalıştırıp yönetir.

- **Containerd:** Docker'ın bir bileşeni olarak başlayan, bağımsız bir konteyner çalışma ve yönetim aracıdır.
- **Cri-o:** Kubernetes için özel olarak geliştirilmiş, hafif ve verimli bir container runtime'dır.
- **Rkt:** CoreOS tarafından geliştirilen, güvenli ve izole edilmiş konteynerler çalıştırırmak için kullanılan bir container runtime'dır.
- **Kata Containers (eski adıyla Clear Containers ve Hyper):** Daha güçlü bir güvenlik sağlamak için hafif sanal makinelerle konteynerleri çalıştırıran bir container runtime'dır.
- **Virtlet:** VM CRI uyumlu bir runtime olup, Kubernetes üzerinde sanal makineler çalıştırırmak için kullanılır.





Optional Services

Architecture Overview



cloud-controller-manager

Kubernetes'teki daemon, bulut sağlayıcısına özgü bilgileri ve entegrasyon yeteneklerini Kubernetes'in temel kontrol döngüsüne sağlayan bir bileşendir. Bu daemon, bulut ortamlarının yönetimini kolaylaştırarak, Kubernetes'in genel işlevselliğini artırır.

- **Node controller:** Düğüm durumunu izler ve yönetir.
- **Route controller:** Ağ trafigini yönlendirme işlevini üstlenir.
- **Service controller:** Servislerin oluşturulması ve yönetimi ile ilgilenir.
- **Ek controller:** Kalıcı Hacim Etiketleri gibi özel durumları işlemek için ek bir kontrolör eklenir.



Cluster DNS

- Kubernetes, serviceler için **Cluster'a DNS** hizmeti sağlar. Bu özellik, uygulamalar arasında kolay ve etkili bir iletişim kurulmasına olanak tanır.
- Bu DNS çözümleme yeteneği, **CoreDNS** üzerine inşa edilmiştir. CoreDNS, esnek ve genişletilebilir bir DNS sunucusudur ve Kubernetes'in dinamik yapılandırma ihtiyaçlarını karşılamak için ideal bir çözümüdür.
- Bu sistem, service adlarını IP adreslerine dönüştürerek, uygulama bileşenlerinin birbirleriyle sorunsuz bir şekilde etkileşimde bulunmasını sağlar. Böylece, Kubernetes ortamında uygulama geliştirme ve yönetimi daha da kolaylaşır.

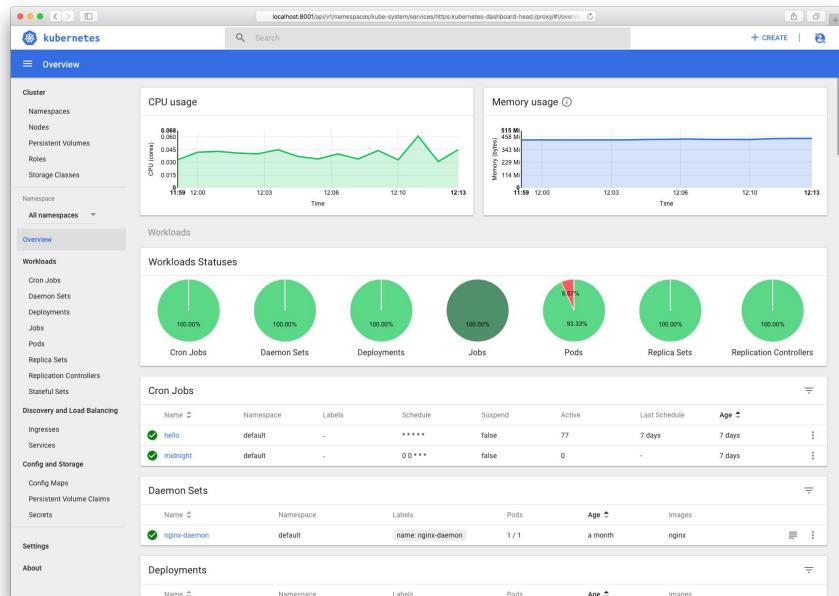
Kube Dashboard



Kubernetes Dashboard, Kubernetes kümesi için sınırlı ama genel amaçlı bir web ön yüzüdür.

Özellikleri

- Kullanıcı Dostu Arayüz:** Kullanıcıların Kubernetes kaynaklarını görselleştirmesine ve yönetmesine olanak tanır.
- Temel Yönetim İşlemleri:** Pod'lar, hizmetler ve diğer kaynaklar üzerinde temel yönetim işlemleri gerçekleştirmeyi sağlar.
- Durum İzleme:** Küme durumu ve kaynakların performansı hakkında bilgi sunar.





Metrics API Server

- **Metrics API:** Mevcut metriklerin toplanması ve sunulması için kullanılan güncel bir çözümdür.
- **Kullanım Kolaylığı:** Metrics API, Kubernetes yöneticilerine ve geliştiricilerine, kaynak kullanımı ve performans hakkında anlık bilgiler sağlar.

Networking

Architecture Overview



Kubernetes Networking

- **Pod Network**

- Pod'lar arası iletişim için kullanılan ve bir **CNI (Container Network Interface)** eklentisi tarafından yönetilen küme genel bir ağdır. Bu ağ, her pod'un diğer pod'larla sorunsuz bir şekilde iletişim kurmasını sağlar.
- Farklı CNI eklentileri, ağ yapılandırmalarını özelleştirmek ve yönetmek için kullanılabilir.

- **Service Network**

- Service keşfi için **kube-proxy** tarafından yönetilen **Virtual IP**'lerin küme genel bir aralığını ifade eder. Bu yapı, hizmetlerin adreslenmesini ve erişimini kolaylaştırır.
- **Virtual IP'ler:** Her service, kendi sanal IP'sine sahiptir, bu sayede kullanıcılar service'lere sabit bir adres üzerinden erişebilir.
- **Service discovery:** Kube-proxy, gelen trafiği uygun pod'lara yönlendirerek, service'lerin düzgün çalışmasını sağlar.



Container Network Interface (CNI)

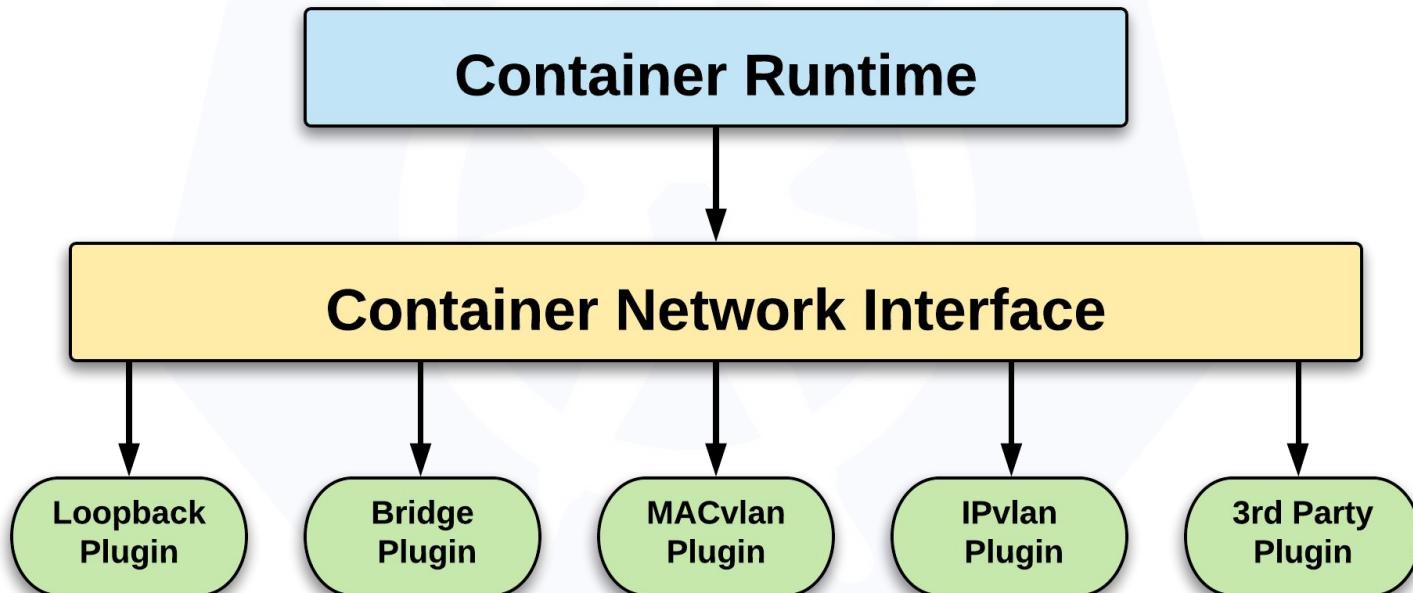
- Kubernetes'teki pod ağı, Container Network Interface (CNI) ile yapılandırılır. CNI, konteyner çalışma zamanı ile bir ağ uygulama eklentisi arasında bir arayüz işlevi görür.

CNI'nin Özellikleri

- CNI, konteynerlerin ağ kaynaklarına erişimini yönetir ve ağ yapılandırmalarını basit bir şekilde sağlar.
- CNI, Cloud Native Computing Foundation (CNCF) projesi olarak, konteyner tabanlı uygulamaların ağ yönetimini standartlaştırmak amacıyla geliştirilmiştir.
- CNI, yapılandırma işlemleri için basit bir JSON şeması kullanır, bu da ağ eklentilerinin kolayca tanımlanmasını ve entegrasyonunu sağlar.

C N I

CNI Overview



CNI Plugins

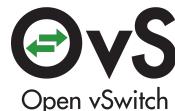


- Amazon ECS
- Calico
- Cillium
- Contiv
- Contrail
- Flannel



cilium

- GCE
- kube-router
- Multus
- OpenVSwitch
- Romana
- Weave





Fundamental Networking Rules

- Pod İçindeki Konteynerler Arası İletişim: Bir pod içerisindeki tüm konteynerler birbirleriyle kesintisiz bir şekilde iletişim kurabilirler.
- Pod'lar Arası İletişim: Tüm pod'lar, NAT (Network Address Translation) olmadan birbirleriyle iletişim kurabilirler.
- Node ile Pod'lar Arası İletişim: Tüm nod'lar, NAT olmadan tüm pod'larla (ve tersine) iletişim kurabilirler.
- Pod'un Görüntülediği IP: Bir pod'un kendi kendini gördüğü IP, diğer pod'lar ve nod'lar tarafından görülen IP ile aynıdır.

Temel Ağ kuralları ve uygulamaları



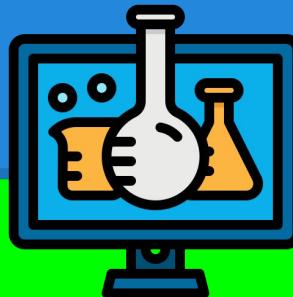
● Container-to-Container

- **Aynı Ağ Ad Alanında:** Bir pod içindeki tüm konteynerler aynı ağ ad alanı içinde bulunur ve aynı IP adresini paylaşırlar.
- **Yerel İletişim:** Bu yapı, konteynerler arasında localhost üzerinden doğrudan ve hızlı iletişim sağlar.

● Pod-to-Pod

- Her pod, yaşam döngüsü süresince benzersiz bir IP adresi alır.
- pod'lar yeniden başlatıldığında veya başka bir pod oluşturulduğunda, her pod'un IP adresi değişebilir.

Kubernetes Cluster Kurulumu





Kubernetes cluster kurulumu

- Kind install on ubuntu
- Minikube install on Mac OS
- Kubernetes cluster install on Ubuntu
- Digitalocean kubernetes cluster kurulumu
- AWS EKS kubernetes cluster kurulumu
- GKE cluster kurulumu
- AKS cluster kurulumu

Minikube



Minikube Cluster başlatma:

minikube start

Minikube Dashboard başlatma

minikube dashboard

Deployment

kubectl create deployment hello-minikube
--image=kicbase/echo-server:1.0

NodePort Service

kubectl expose deployment hello-minikube --type=NodePort
--port=8080

Service ulaşılabilme için

minikube service hello-minikube

Stop your local cluster: minikube stop

Delete your local cluster: minikube delete

Click on the buttons that describe your target platform. For other architectures, see [the release page](#) for a complete list of minikube binaries.

Operating system ←

Architecture

Release type

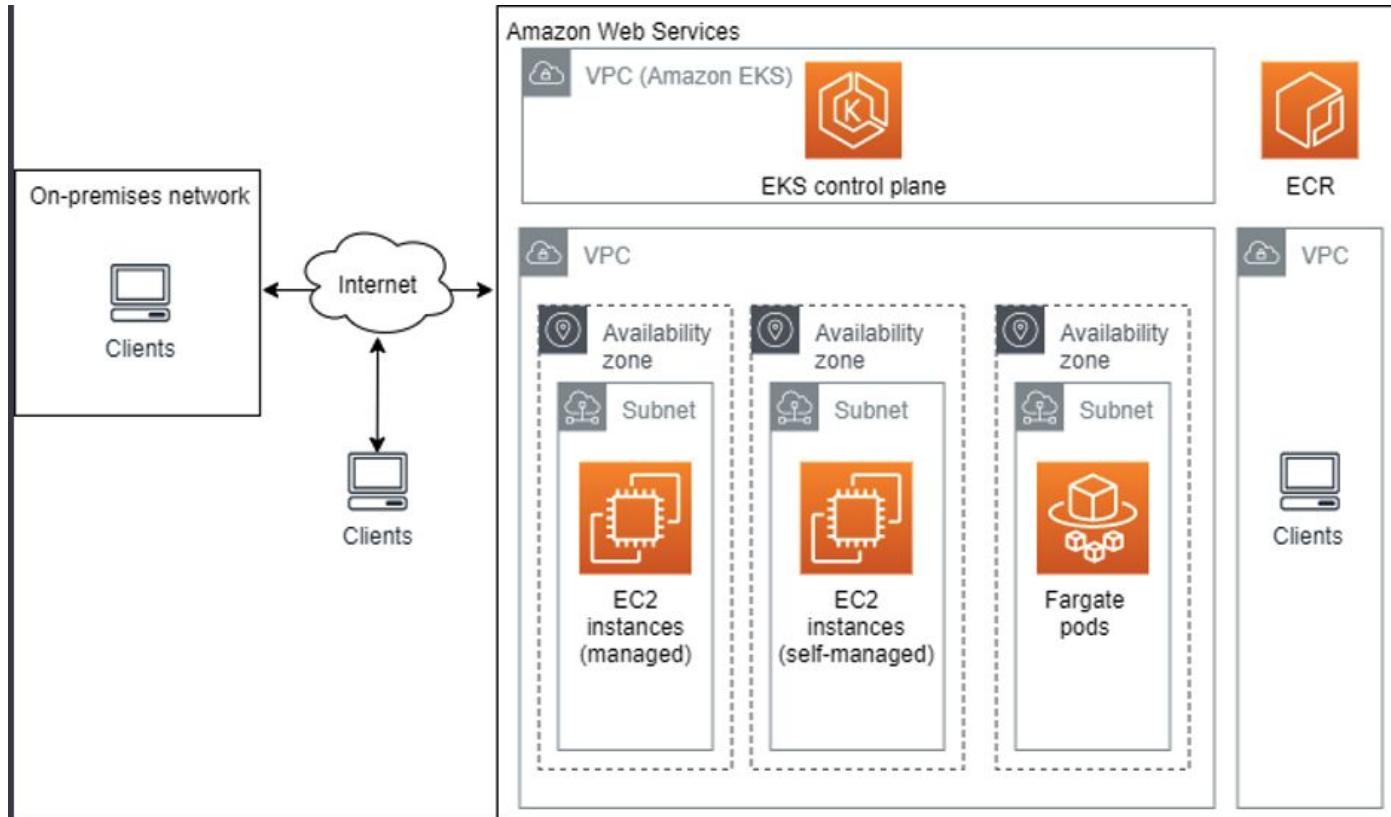
Installer type

To install the latest minikube stable release on x86-64 macOS using binary download:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
sudo install minikube-darwin-amd64 /usr/local/bin/minikube
```

<https://minikube.sigs.k8s.io/docs/>

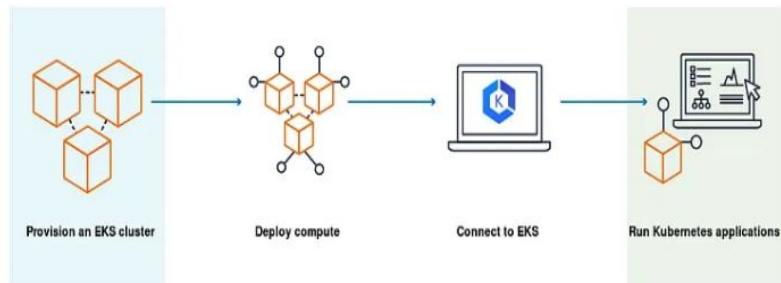
AWS EKS



AWS EKS



How does Amazon EKS work?



<https://docs.aws.amazon.com/eks/latest/userguide/getting-started-console.html>

<https://eksctl.io/installation/>

<https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html>

DigitalOcean Kubernetes



How to Create a Kubernetes Cluster Using the DigitalOcean CLI

1. Install [doctl](#), the DigitalOcean command-line tool.

2. [Create a personal access token](#) and save it for use with doctl.

3. Use the token to grant doctl access to your DigitalOcean account.

```
doctl auth init
```

4. Finally, run `doctl kubernetes cluster create`. Basic usage looks like this, but you can read the usage docs for more details:

```
doctl kubernetes cluster create <name> [flags]
```

The following example creates a cluster named `example-cluster` in the `nyc1` region with a node pool, using Kubernetes version `1.28.2-do.0`:

```
doctl kubernetes cluster create example-cluster --region nyc1 --version 1.28.2-do.0 --maintenance-window satu
```

The screenshot shows the DigitalOcean dashboard interface. At the top right, there's a navigation bar with 'Dashboard', 'Regions', 'Products', 'Marketplace', 'Logs', and 'Metrics'. Below the navigation is a search bar and a 'Estimated costs: \$0.00' indicator. The main area features a project titled 'example-project' with the subtitle 'Just trying out DigitalOcean / For documentation.' Below the project title are tabs for 'Resources', 'Activity', and 'Settings'. On the left, there's a sidebar with various service icons: Droplets (selected), App Platform, Functions, Databases, Domains/DNS, Cloud Firewalls, Reserved IPs, Load Balancers, and Resource Alerts. A red box highlights the 'Kubernetes' icon under the 'Droplets' section. To the right of the sidebar, there are several cards: 'Welcome to DigitalOcean!', 'Spin up a Droplet', 'Deploy your web app', and 'Getting Started'. The bottom right corner has a decorative background image of a whale.

<https://docs.digitalocean.com/products/kubernetes/how-to/create-clusters/>



Kind (Kubernetes IN Docker)

Cluster Oluşturma:

```
kind create cluster --name <cluster-name>
```

Cluster Silme:

```
kind delete cluster --name <cluster-name>
```

Varolan Cluster'ları Listeleme:

```
kind get clusters
```

Kubeconfig Ayarlarını Alma:

```
kind get kubeconfig --name <cluster-name>
```

Cluster'a Node Ekleme (Multi-node Cluster için):

```
kind create cluster --config <config.yaml>
```

Kind Cluster İçindeki Node'ları Listeleme:

```
kubectl get nodes
```

Docker Olarak Çalışan Cluster'ları Kontrol Etme:

```
docker ps
```

On Linux:

```
# For AMD64 / x86_64
[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-amd64
# For ARM64
[ $(uname -m) = aarch64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-linux-arm64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
```

On macOS:

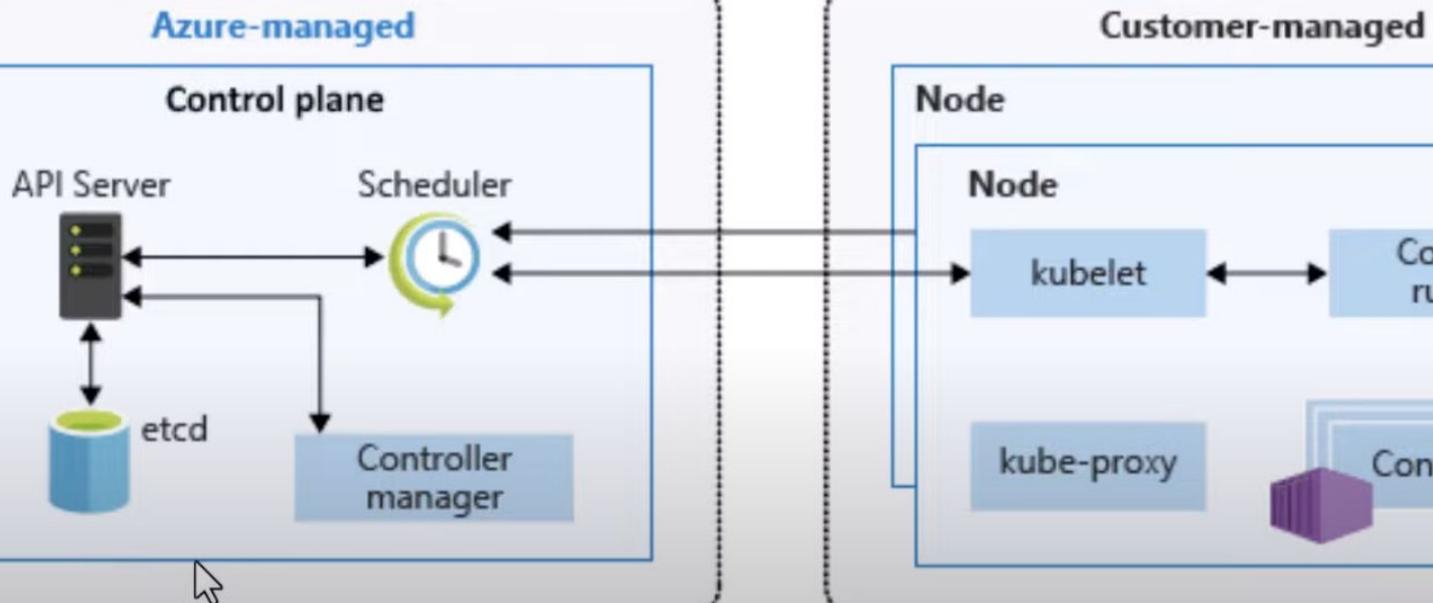
```
# For Intel Macs
[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-darwin-amd64
# For M1 / ARM Macs
[ $(uname -m) = arm64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.24.0/kind-darwin-arm64
chmod +x ./kind
mv ./kind /some-dir-in-your-PATH/kind
```

On Windows in PowerShell:

```
curl.exe -Lo kind-windows-amd64.exe https://kind.sigs.k8s.io/dl/v0.24.0/kind
Move-Item .\kind-windows-amd64.exe c:\some-dir-in-your-PATH\kind.exe
```

<https://kind.sigs.k8s.io/>

AKS cluster



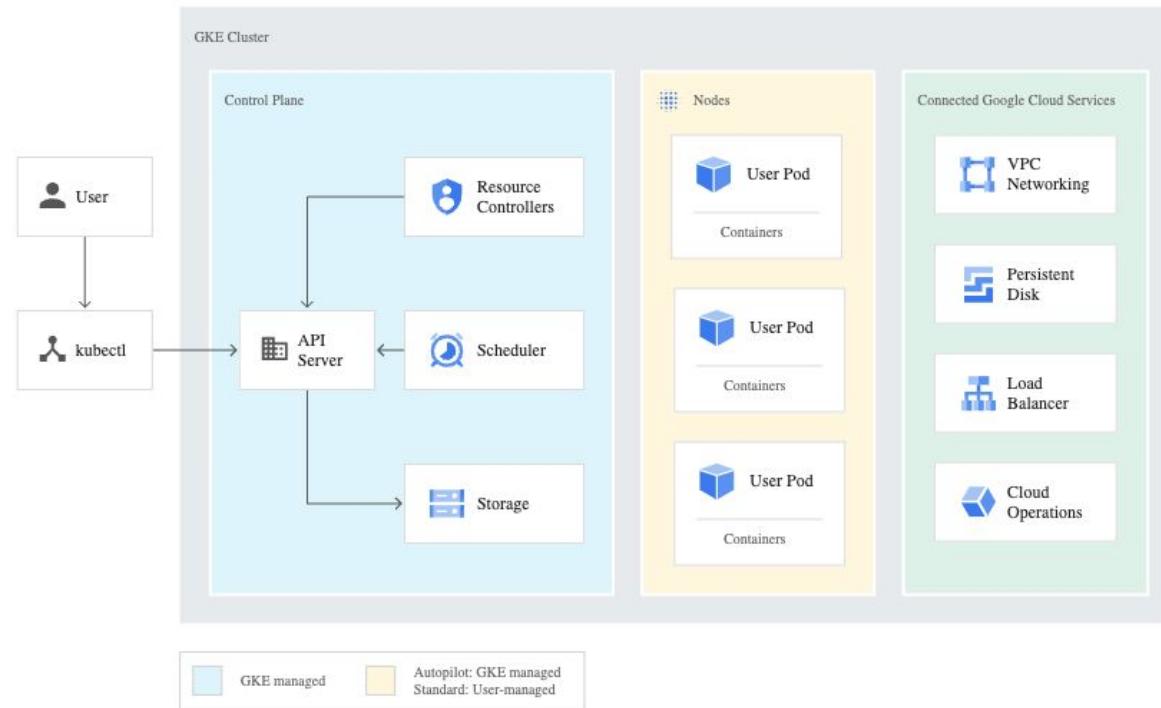


AKS cluster

Node Pool

- It is set of virtual machine with same configurations
- System Node Pool
 - CoreDNS, metrics server, tunnel front and oms agent
- User Node Pool
 - Deploy Applications

GKE Cluster Architecture





GKE has two types of cluster

Autopilot mode			Standard mode
Optimised Kubernetes cluster with a hands-off experience			Kubernetes cluster with node configuration flexibility
Scaling	Automatic based on workload	You configure scaling	
Nodes	Google manages and configures your nodes	You manage and configure your nodes	
Configuration	Streamlined configuration ready to use	You can configure all options	
Workloads supported	Most workloads except these limitations	All Kubernetes workloads	
Billing method	Pay per pod	Pay per node (VM)	
SLA	Kubernetes API and node availability	Kubernetes API availability	

GKE



Kubernetes Engine

Overview

Learn about Enterprise

All Fleets

Resource Management

- Overview
- Clusters
- Workloads
- Teams
- Applications
- Secrets & ConfigMaps
- Storage
- Object Browser
- Rollout Sequencing
- Backup for GKE

Posture Management

GET STARTED NEW FLEET DASHBOARD

Build applications and services to run in GKE

Review the documentation, training, and features to help you onboard, manage, and scale GKE.

Learn about containerized applications

GKE provides the operational power of Kubernetes while managing many of the underlying components, such as the control plane and nodes, for you. To learn how GKE can help you scale, automate, and manage your workloads, review the documentation and tutorials.

[EXPLORE TUTORIALS](#)

Create a new cluster or deploy a container

For a more managed Kubernetes experience, spin up an Autopilot cluster that's optimized for most production workloads. You can also deploy your containerized workloads to GKE clusters. This operation creates a cluster if one doesn't exist. [Learn about modes of operations](#)

[CREATE CLUSTER](#) [DEPLOY CONTAINER](#)

Scale beyond a single team or cluster

Spend less time and effort managing the platform more time creating amazing applications and experiences for your customers. GKE Enterprise gives you an integrated and consistent way to configure, secure, protect, and monitor your container workloads.

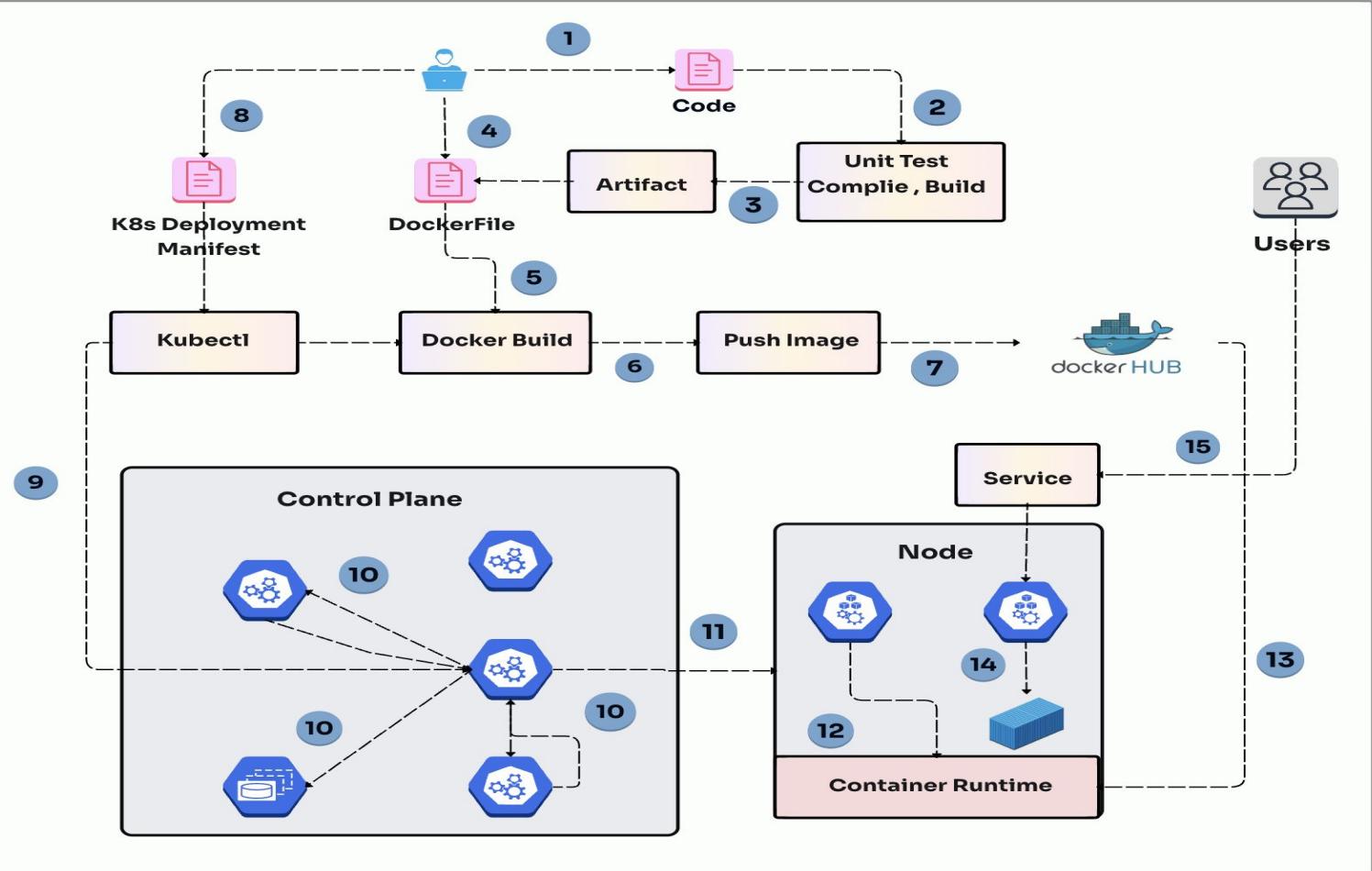
[LEARN AND ENABLE](#)

Fast-track your onboarding with sample configurations

<https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl>

<https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview>

How Application Deploy on Kubernetes





Kubeconfig file

Example Kubeconfig File

Here is an example of a Kubeconfig. It needs the following key information to connect to the Kubernetes clusters.

- 1 **certificate-authority-data:** Cluster CA
- 2 **server:** Cluster endpoint (IP/DNS of the master node)
- 3 **name:** Cluster name
- 4 **user:** name of the user/service account.
- 5 **token:** Secret token of the user/service account.

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: <ca-data-here>
  server: https://your-k8s-cluster.com
  name: <cluster-name>
contexts:
- context:
  cluster: <cluster-name>
  user: <cluster-name-user>
  name: <cluster-name>
current-context: <cluster-name>
kind: Config
preferences: {}
users:
- name: <cluster-name-user>
  user:
    token: <secret-token-here>
```

How to Connect to a Kubernetes Cluster

@hellodeolu



CLIENT

Connecting to a Kubernetes cluster remotely, you usually use `kubectl`. A command-line tool for interacting with kubernetes resources and performing operations on the cluster. Kubectl uses the credentials and configuration info stored in the `.kube/config` file to authenticate and authorize requests to the API server.

```
{  
  KUBE_API_ADDRESS = ...  
  kubectl config set-cluster k8s-1 \  
    --certificate-authority=ca.crt \  
    --embed-certs=true \  
    --server=https://$ \  
    (KUBE_API_ADDRESS):6443  
  kubectl config set-credentials admin \  
    --client-certIFICATE=admin.crt \  
    --client-key=admin.key \  
  kubectl config set-context k8s-1 \  
    --cluster=k8s-1 \  
    --user=admin \  
  kubectl config use-context k8s-1  
}
```

```
kubectl config view  cat .kube/config  
apiVersion: v1  
kind: Config  
clusters:  
  - name: k8s-1  
    cluster:  
      certificate-authority-data: <cert. data>  
      server: https://127.0.0.1:41284  
users:  
  - name: deolu  
    user:  
      client-certificate-data: <cert. data>  
      client-key-data: <key data>  
contexts:  
  - name: deolu@k8s-1  
    context:  
      cluster: k8s-1  
      user: deolu  
      namespace: dev  
current-context: deolu@k8s-1
```



KUBERNETES CLUSTER

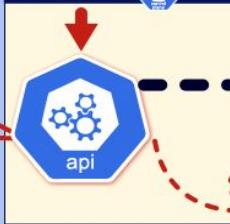
Worker Node-1



Worker Node-2



Master Node



HTTP request

response

- Processes the request
- Performs the requested operation
- Returns a response to kubectl



kubernetes cluster commands

List all cluster contexts

```
kubectl config get-contexts -o=name
```

Set the current context

```
kubectl config use-context <cluster-name>
```

Validate the Kubernetes cluster connectivity

```
kubectl get nodes
```

Using Kubeconfig File With Kubectl

```
kubectl get nodes --kubeconfig=$HOME/.kube/dev_cluster_config
```

Delete Cluster Context From Kubeconfig

```
kubectl config get-contexts -o=name
```

```
kubectl config delete-context <context-name>
```

```
kubectl config use-context <context-name>
```

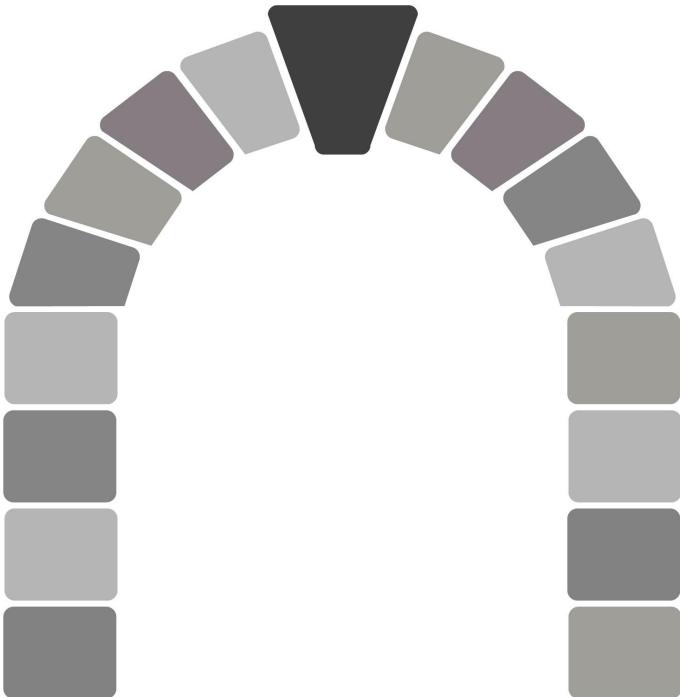
API Overview



REST API, Kubernetes'in temel taşını oluşturur.

Kubernetes ekosistemindeki her şey, bir API nesnesi olarak temsil edilir

- Tüm kaynaklar (podlar, hizmetler, düğümler vb.) **API nesneleri** olarak tanımlanır ve yönetilir.
- Kullanıcılar ve uygulamalar, Kubernetes ile etkileşimde bulunmak için REST API'yi kullanarak kaynakları oluşturabilir, güncelleyebilir ve silebilir.
- API, küme durumunu kontrol etme ve yönetme yeteneği sağlar, bu da Kubernetes'in güçlü ve esnek bir platform olmasını sağlar.



[Image Source](#)

API Versioning



- Kubernetes, API olgunluk düzeylerini üç katmana ayırır. Bu düzeyler, `apiVersion` nesnesinde de referans alınır.
- **Alpha:** Alpha sürümleri, potansiyel olarak hatalı olabilir ve sıkılıkla değişiklik gösterir. Varsayılan olarak devre dışı bırakılmıştır.
- **Beta:** Beta sürümleri, test edilmiş ve stabil kabul edilmiştir; ancak API şeması değişebilir.. Varsayılan olarak etkinleştirilmiştir.
- **Stable:** Stabil sürümler, yayınlanmış, güvenilir ve API şeması değişmeyecek şekilde tasarlanmıştır. Varsayılan olarak etkinleştirilmiştir.

Format:

`/apis/<group>/<version>/<resource>`

Examples:

`/apis/apps/v1/deployments`

`/apis/batch/v1beta1/cronjobs`



Object Model

Tüm Kubernetes nesneleri aşağıdaki alanları içermelidir:

- **apiVersion:** Nesnenin hangi API sürümüne ait olduğunu belirtir.
- **kind:** Nesnenin türünü tanımlar (örneğin, Pod, Service vb.).
- **metadata:**
 - **name:** Nesnenin benzersiz adını belirtir.
 - **namespace:** Nesnenin hangi ad alanında bulunduğu gösterir.
 - **uid:** Nesneye atanın benzersiz kimlik.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  namespace: default
  uid: f8798d82-1185-11e8-94ce-080027b3c7a6
```



Object Model Requirements

- `apiVersion`: Kubernetes API version of the Object
- `kind`: Type of Kubernetes Object
- `metadata.name`: Unique name of the Object
- `metadata.namespace`: Scoped environment name that the object belongs to (will default to current).
- `metadata.uid`: The (generated) uid for an object.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  namespace: default
  uid: f8798d82-1185-11e8-94ce-080027b3c7a6
```



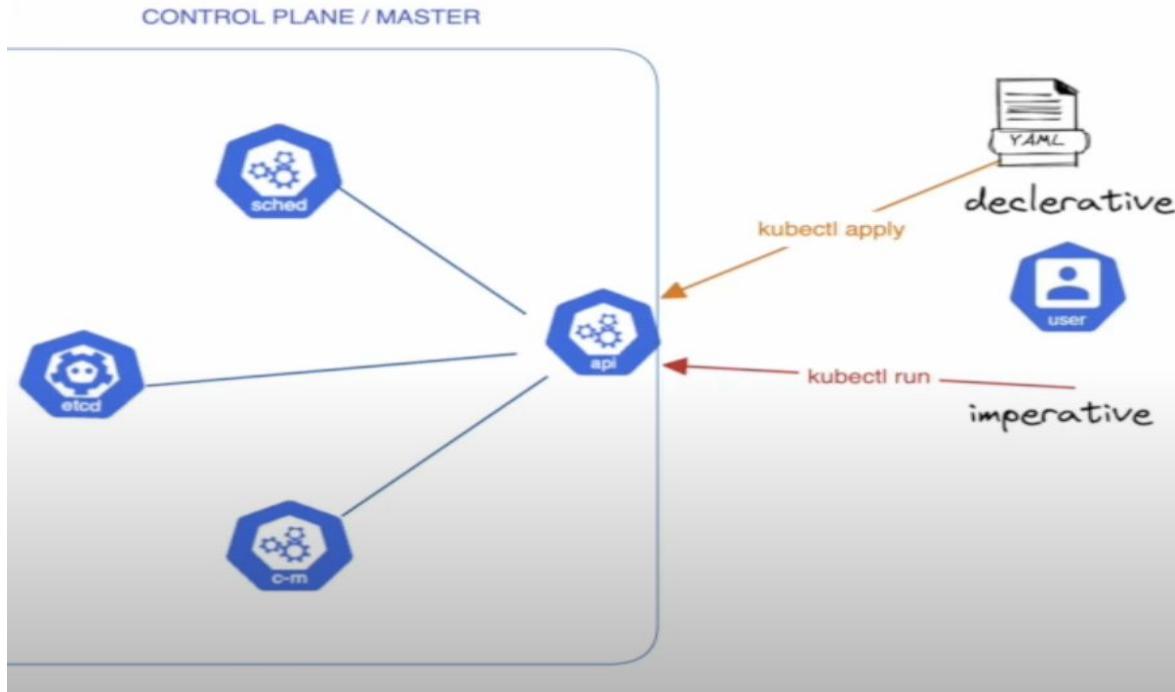
Object Expression - YAML

- Kubernetes nesneleri genellikle **YAML** formatında temsil edilir. YAML, insan dostu bir veri serileştirme standardıdır
- **Okunabilirlik:** YAML, insanlar tarafından okunması kolay bir format sunar.
- **Boşluk Kullanımı:** Sahipliği belirtmek için boşluk (özellikle boşluk karakterleri) hizalaması kullanılır.
- **Temel Veri Türleri:**
 - **mappings** - hash or dictionary,
 - **sequences** - array or list
 - **scalars** - string, number, boolean etc

Kubernetes kaynaklarını yönetme metodları



Kubernetes kaynaklarını yönetme



Declarative: Kubernetes kaynaklarının durumunu bir YAML veya JSON dosyasında tanımlayarak, bu dosyayı kullanarak Kubernetes nesnelerini yönetmeyi sağlar.

Imperative: Kubernetes nesnelerini doğrudan komut satırından yönetmeyi sağlar.



Declarative & Imperative

Özellik	Declarative	Imperative
Tanım	Kaynakların istenen son durumu tanımlanır.	Belirli komutlarla işlem yapılır.
Kullanım	YAML/JSON dosyalarıyla kaynak yönetimi	Komut satırıyla doğrudan yönetim
Takip Edilebilirlik	Sürüm kontrolüne uygundur.	İzlenmesi zordur, her değişiklik manuel yapılır.
İdempotentlik	İdempotenttir, tekrar çalıştırırmak güvenlidir.	Her komut yeni bir işlem başlatır.
Örnek Komut	<code>kubectl apply -f deployment.yaml</code>	<code>kubectl create deployment my-app --image=nginx</code>



YAML nedir?

YAML (YAML Ain't Markup Language), veri serileştirme (data serialization) formatıdır.

YAML temel bileşenleri:

- **Mappings (haritalar veya sözlükler)**
- **Sequences (diziler veya listeler)**
- **Scalars (skaler değerler)**

1-Mappings (Haritalar/Sözlükler):

```
person:  
  name: Alice  
  age: 30  
  city: Istanbul
```

2-Sequences (Diziler/Listeler):

```
fruits:  
  - apple  
  - banana  
  - cherry
```

3-Scalars (Skaler Değerler):

```
name: Alice      # String  
age: 30          # Number  
married: false   # Boolean
```



Tüm Yapıların Birlikte Kullanıldığı Örnek:

```
person:  
  name: Alice          # Scalar (String)  
  age: 30              # Scalar (Number)  
  married: false       # Scalar (Boolean)  
  children:  
    - name: Bob        # Mapping inside a sequence  
      age: 5  
    - name: Charlie  
      age: 3  
  address:             # Mapping  
    street: Main Street # Scalar (String)  
    city: Istanbul       # Scalar (String)
```



Kubernetes Manifest Yapısı?

Kubernetes YAML Temel Bileşenleri

Her Kubernetes manifest dosyasında aşağıdaki temel alanlar bulunur:

1. **apiVersion**: Kubernetes API'nin hangi versiyonunun kullanılacağını belirtir.
2. **kind**: Oluşturulan kaynağın türü (örneğin, Pod, Service, Deployment).
3. **metadata**: Kaynakla ilgili meta verileri içerir. Genellikle `name` , `namespace` , `labels` gibi bilgiler burada tanımlanır.
4. **spec**: Kaynağın istenilen durumu ve yapılandırmasını belirler. Bu kısım kaynağına göre değişir.



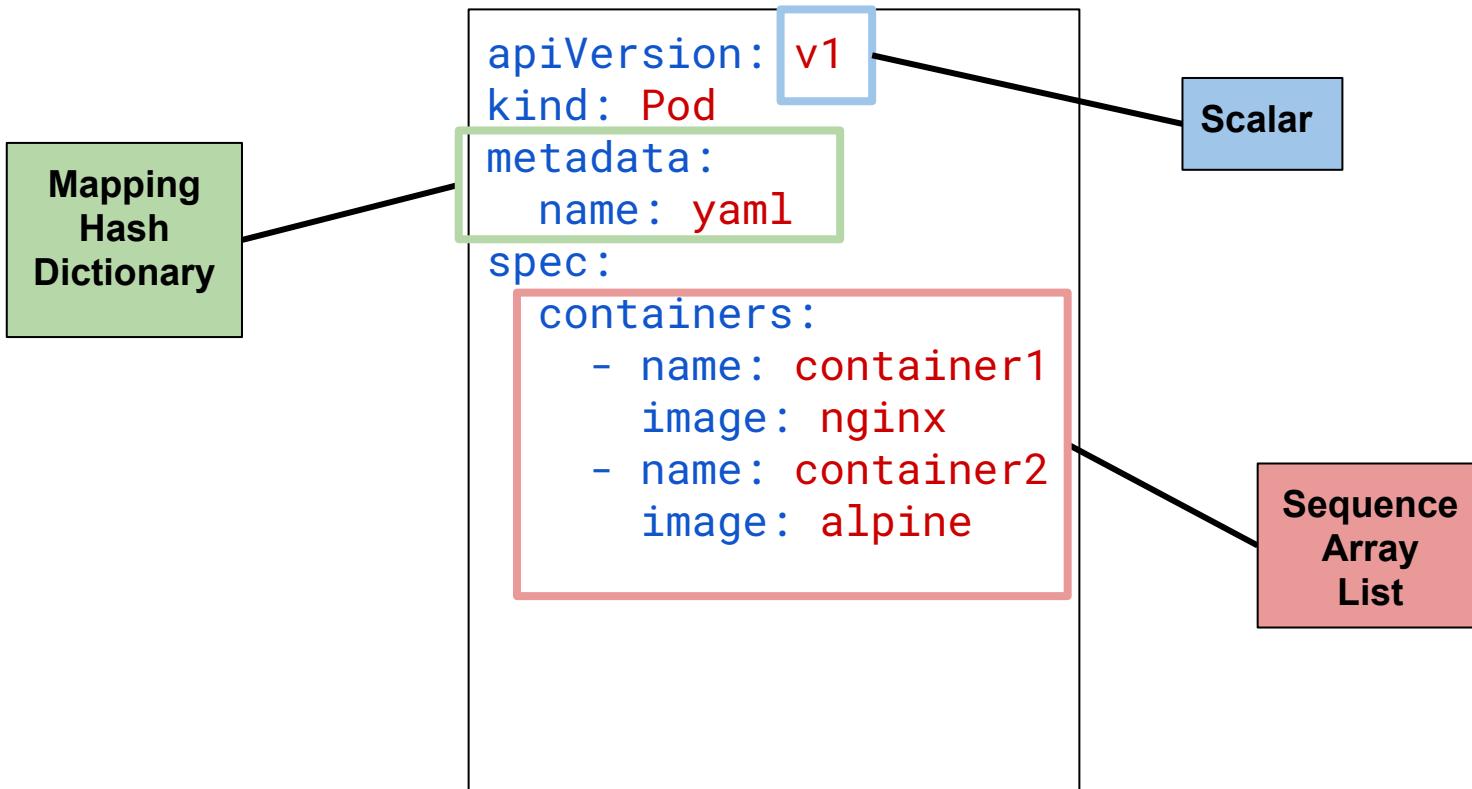
Örnek Kubernetes Pod Manifestesi:

yaml

```
apiVersion: v1          # Mapping: API versiyonu
kind: Pod               # Mapping: Kaynak türü (Pod)
metadata:               # Mapping: Pod'un metadata bilgileri
  name: my-pod         # Scalar: Pod'un adı
spec:                   # Mapping: Pod'un özellikleri
  containers:          # Sequence: Birden fazla container olabilir
    - name: nginx        # Mapping: İlk container için isim
      image: nginx:latest # Scalar: Container imajı
      ports:               # Sequence: Container'ın portları
        - containerPort: 80 # Scalar: Port numarası
```



Object Expression - YAML





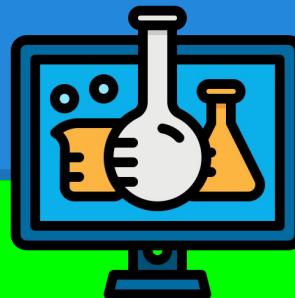
YAML vs JSON

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "pod-example"
  },
  "spec": {
    "containers": [
      {
        "name": "nginx",
        "image": "nginx:stable-alpine",
        "ports": [ { "containerPort": 80 } ]
      }
    ]
  }
}
```

YAML

Manifest & Imperative



Core Objects

- Namespaces
- Pods
- Labels
- Selectors
- Services

Namespaces



Namespace'ler, mantıksal bir küme veya ortamdır ve bir kümeyi bölmenin veya erişimi sınırlandırmanın birincil yöntemidir

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

```
$ kubectl get ns --show-labels
NAME      STATUS   AGE     LABELS
default   Active   11h    <none>
kube-public   Active   11h    <none>
kube-system   Active   11h    <none>
prod       Active   6s     app=MyBigWebApp
```



Default Namespaces

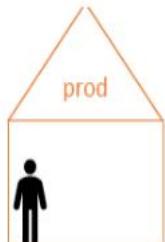
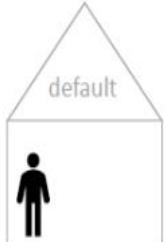
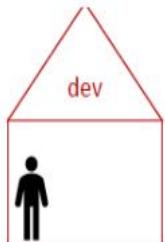
- **default:** Bir namespace belirtilmediğinde, varsayılan olarak kullanılan namespace'tır. Başka bir namespace atanmamış her nesne burada yer alır.
- **kube-system:** Kubernetes tarafından oluşturulan nesneler ve kaynaklar için kullanılan özel bir namespace'tır. Kubernetes'in kendisine ait bileşenlerin yönetildiği alandır.
- **kube-public:** Özel bir namespace'tır; tüm kullanıcılar tarafından okunabilir. Genellikle cluster'ın başlangıç yapılandırması ve ayarları için ayrılmıştır.

```
$ kubectl get ns --show-labels
NAME          STATUS   AGE    LABELS
default        Active   11h   <none>
kube-public   Active   11h   <none>
kube-system   Active   11h   <none>
```



Namespace

Kubernetes Namespaces



```
> kubectl get pods --namespace=dev
```

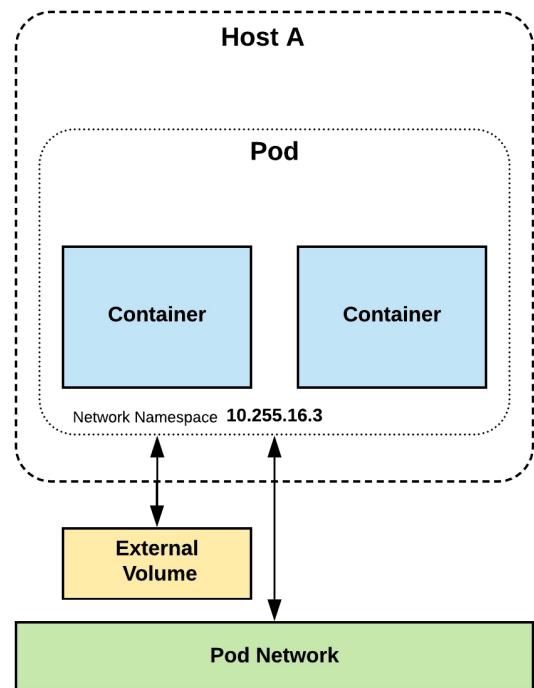
```
> kubectl get pods
```

```
> kubectl get pods --namespace=prod
```

Pod



- **Pod**, Kubernetes'in atomik birimi ya da en küçük 'iş birimi'dir.
- Kubernetes iş yüklerinin temel yapı taşıdır.
- Podlar, birden fazla konteyneri içerebilir ve bu konteynerler aynı disk alanlarını , ağ alanını paylaşır ve tek bir context içinde çalışır."



Pod Examples



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
```

```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
    - name: content
      image: alpine:latest
      command: ["/bin/sh", "-c"]
      args:
        - while true; do
            date >> /html/index.html;
            sleep 5;
        done
      volumeMounts:
        - name: html
          mountPath: /html
      volumes:
        - name: html
          emptyDir: {}
```



Key Pod Container Attributes

- **name** - Konteynerin adını belirtir
- **image** - Kullanılacak konteyner imajını tanımlar.
- **ports** - Açık portların bir dizisi. Her port, dostça bir ad alabilir ve protokol belirtilebilir.
- **env** - Ortam değişkenlerinin bir dizisi
- **command** - Giriş noktası dizisi(Docker'daki `ENTRYPOINT` eşdeğeri)
- **args** - Komuta iletilmesi gereken argümanlar (Docker'daki `CMD` eşdeğeri)

Container

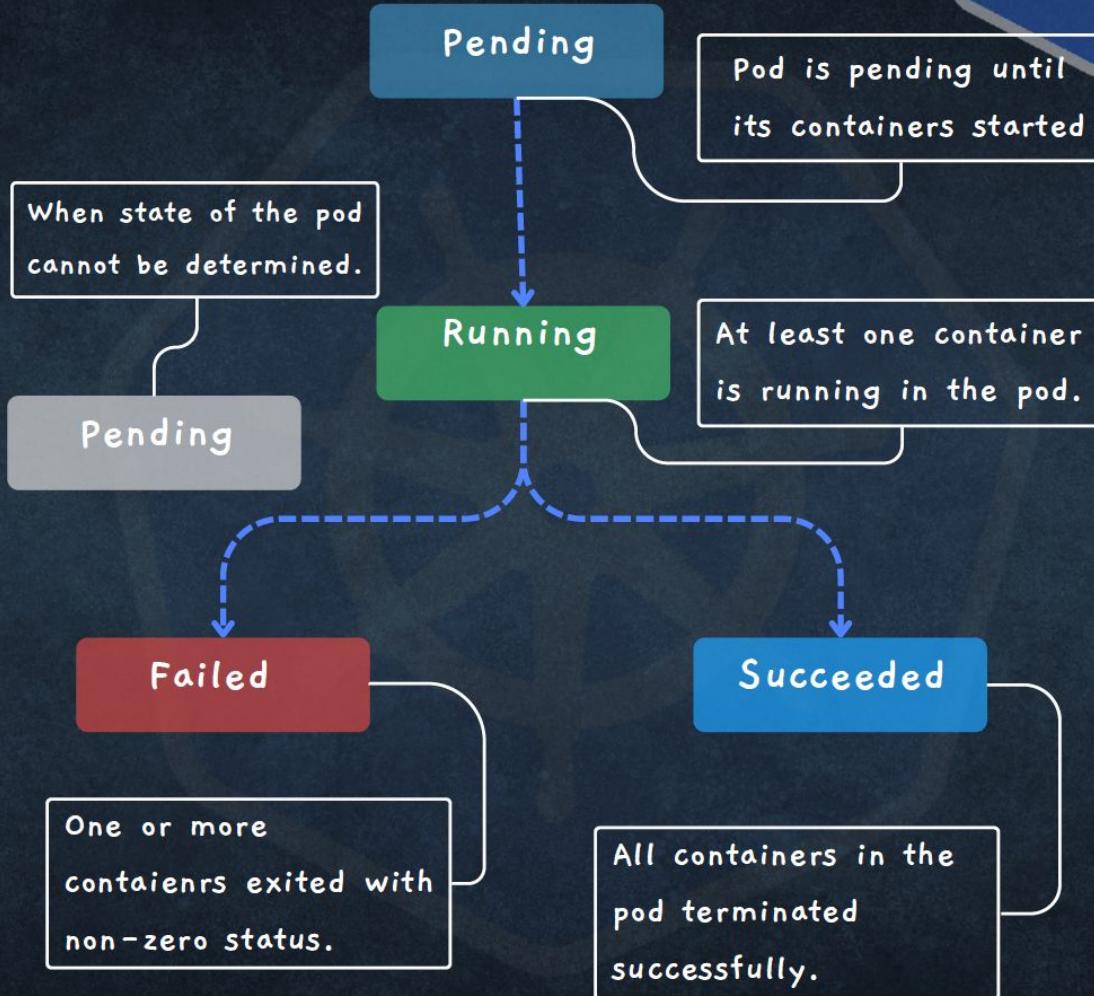
```
name: nginx
image: nginx:stable-alpine
ports:
- containerPort: 80
  name: http
  protocol: TCP
env:
- name: MYVAR
  value: isAwesome
command: ["/bin/sh", "-c"]
args: ["echo ${MYVAR}"]
```



Pod Lifecycle

Pod Yaşam Döngüsü

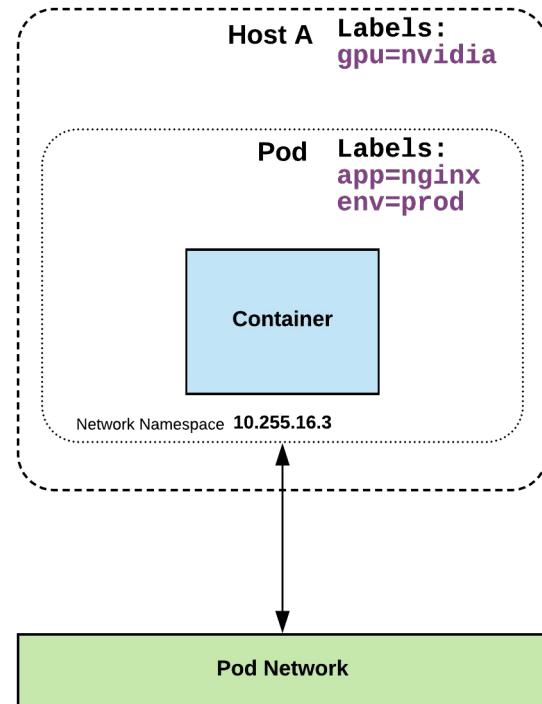
- **Pending (Beklemede)**: Pod, konteynerlerini başlatana kadar bu durumda kalır.
- **Running (Çalışıyor)**: Pod'un en az bir konteyneri çalışıyorsa bu durumda olur.
- **Succeeded (Başarılı)**: Pod'un tüm konteynerleri başarılı bir şekilde sona erdiğinde bu duruma geçer.
- **Failed (Başarısız)**: Pod'daki bir veya daha fazla konteyner, sıfır olmayan bir çıkış kodu ile sonlandıysa bu duruma geçer.
- **Unknown (Bilinmiyor)**: Pod'un durumu belirlenemediğinde bu durum ortaya çıkar.





Labels

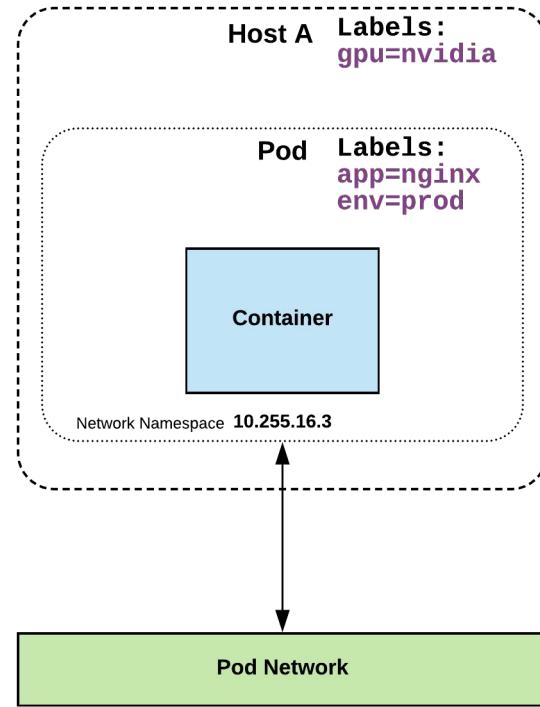
- Nesneleri veya kaynakları tanımlamak, açıklamak ve ilişkili gruplar oluşturmak için kullanılan anahtar-değer çiftleridir.
- Etiketler, benzersizlik özelliği taşımamaktadır.
- Belirli bir söz dizimi vardır ve sınırlı bir karakter kümesi kullanılarak tanımlanır..



Label Example



```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
labels:
  app: nginx
  env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```



Selectors



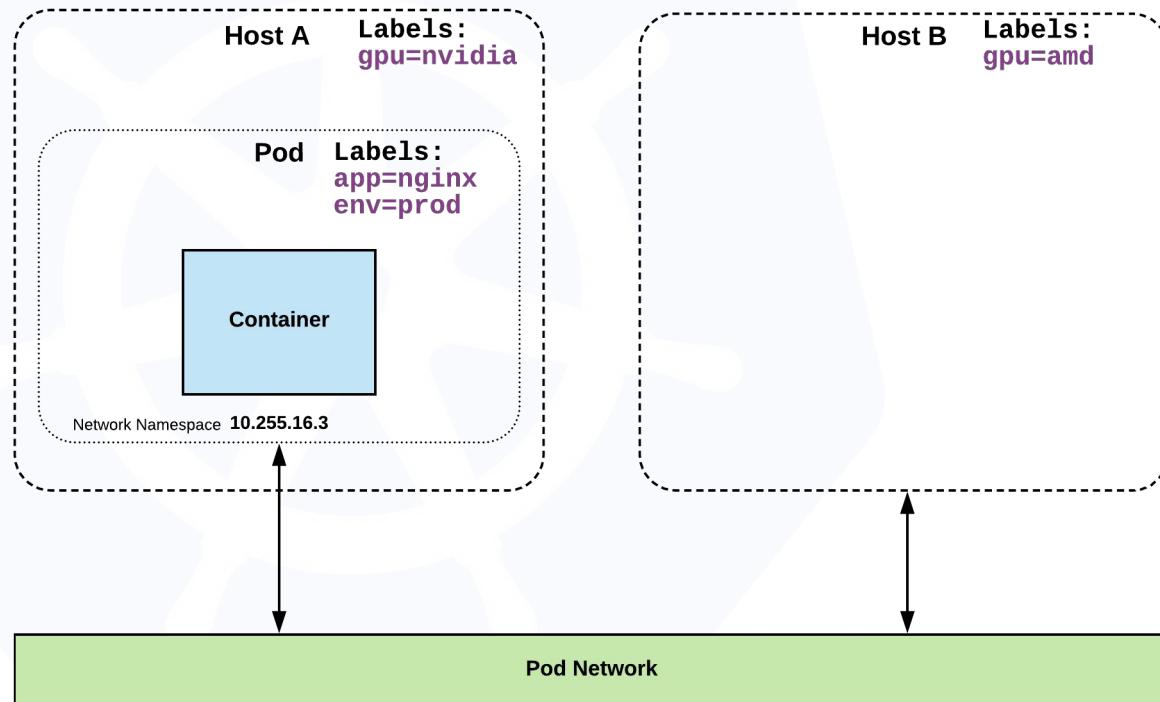
Selectors (Seçiciler): Kubernetes'te nesneleri seçmek ve yönetmek için kullanılan bir mekanizmadır.

Labels (Etiketler): Kubernetes nesnelerine eklenen anahtar-değer çiftleridir. Seçiciler, bu etiketler aracılığıyla nesneleri filtreleyip yönetebilir.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```

Selector Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
  nodeSelector:
    gpu: nvidia
```



Selector Types



Equality based nesneleri basit bir şekilde filtrelemek için kullanılır. Bu tür seçimciler, aşağıdaki operatörleri destekler: (=, ==, or !=)

```
selector:  
  matchLabels:  
    gpu: nvidia
```

Selector belirli nesneleri filtrelemek için kullanılan bir mekanizmadır. Set tabanlı seçimciler, sınırlı bir nesne alt kümesinde desteklenir, ancak bir dizi değer üzerinde filtreleme yapma imkanı sunar.

1. **In:** Belirtilen değerlerin bir kümeye içinde olup olmadığını kontrol eder.
2. **notin:** Belirtilen değerlerin kümeye dışında olup olmadığını kontrol eder.
3. **exists:** Belirtilen alanın mevcut olup olmadığını kontrol eder.

```
selector:  
  matchExpressions:  
    - key: gpu  
      operator: in  
      values: ["nvidia"]
```



Services

- Pod'lardan farklı olarak, hizmetler kalıcı bir varlıktır; pod'lar silinse bile hizmetler varlığını sürdürür.
- Pod'lardan farklı olarak, hizmetler kalıcı bir varlıktır; pod'lar silinse bile hizmetler varlığını sürdürür.
 - static cluster-unique IP
 - static namespaced DNS name

<service name>. <namespace>. svc.cluster.local

Services



- Kubernetes'teki hizmetler, eşitlik tabanlı seçimciler kullanarak hedef pod'ları belirler.
- Yük dengeleme işlemleri için **kube-proxy** kullanılır. Bu, hizmete gelen trafiği hedef pod'lar arasında dağıtır.
- **Kube-proxy**, her hizmet için ana makinenin iptables yapılandırmasında yerel girişler oluşturan bir daemon olarak çalışır. Bu, ağ trafiğinin doğru pod'lara yönlendirilmesini sağlar



Service Types

There are 4 major service types:

- **ClusterIP** (default)
- **NodePort**
- **LoadBalancer**
- **ExternalName**

ClusterIP Service



ClusterIP service yalnızca küme içindeki diğer bileşenler tarafından erişilmesine izin verir; dışarıdan erişim yoktur.

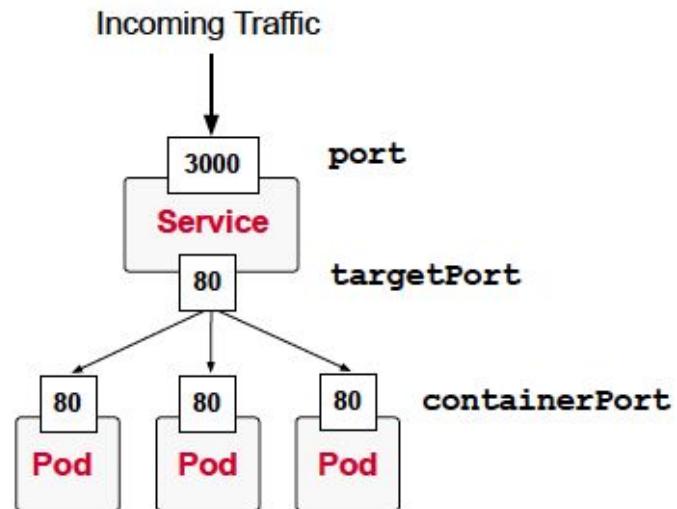
Her ClusterIP servisi, küme içinde benzersiz bir sanal IP adresine sahiptir, bu da hizmetin güvenilir bir şekilde tanımlanmasını sağlar.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```



Port Mapping

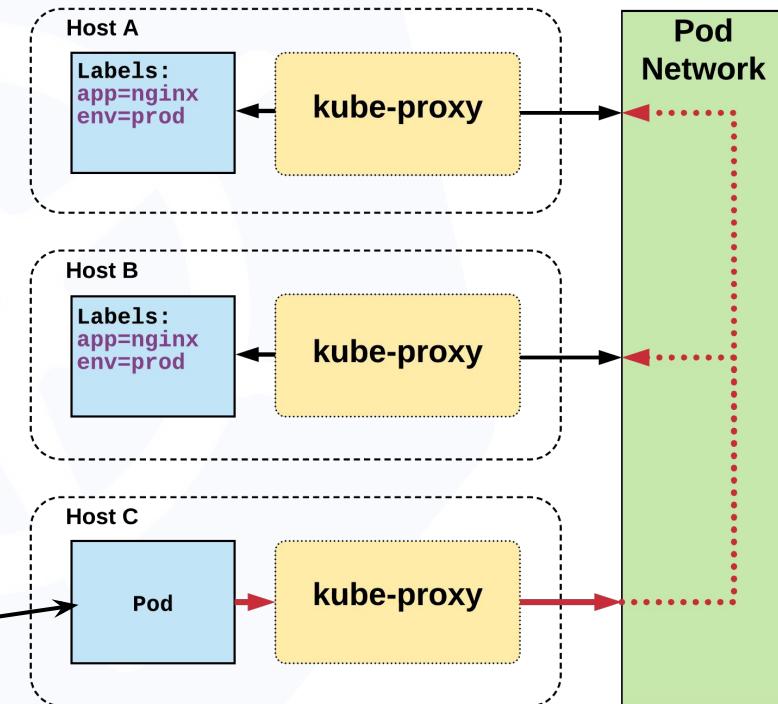
Port Mapping: Servis portu ile konteyner portu arasındaki eşlemedir. Bu, dışarıdan gelen isteklerin doğru konteynere yönlendirilmesini sağlar.



Cluster IP Service

```
Name: example-prod  
Selector: app=nginx,env=prod  
Type: ClusterIP  
IP: 10.96.28.176  
Port: <unset> 80/TCP  
TargetPort: 80/TCP  
Endpoints: 10.255.16.3:80,  
           10.255.16.4:80
```

```
/ # nslookup example-prod.default.svc.cluster.local  
Name: example-prod.default.svc.cluster.local  
Address 1: 10.96.28.176 example-prod.default.svc.cluster.local
```



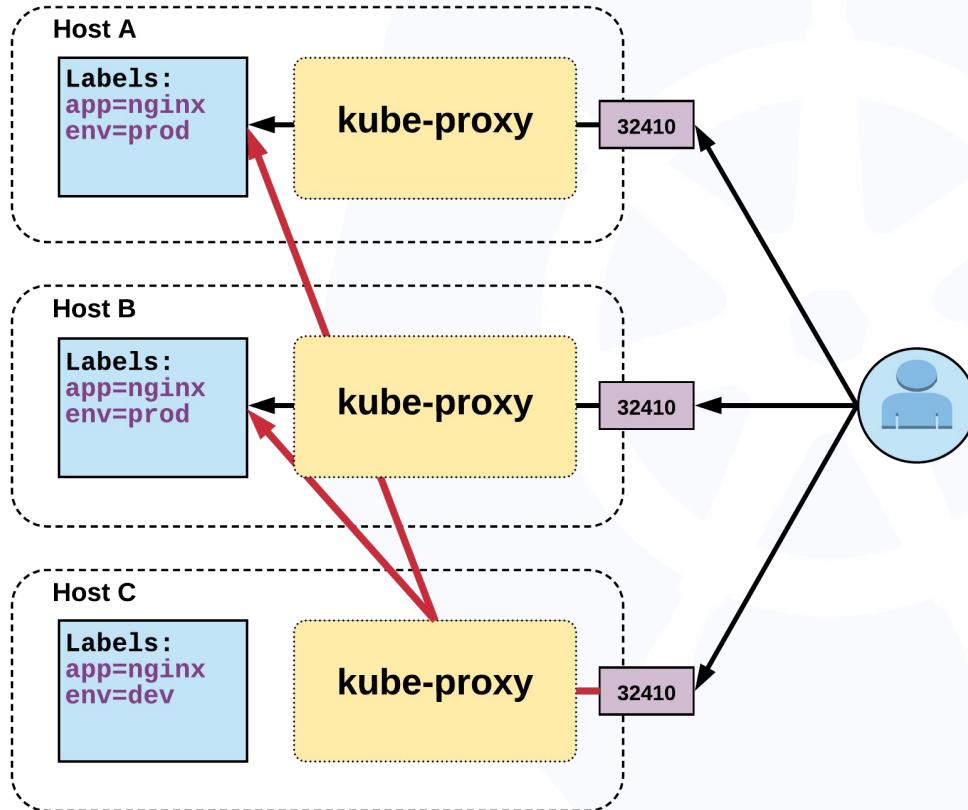
NodePort Service



- **NodePort servisi, ClusterIP servislerini** genişleterek, daha fazla erişim imkanı sunar.
- NodePort, her node'un IP adresinde belirli bir portu maruz bırakır. Bu sayede, kullanıcılar hizmete doğrudan node IP'si üzerinden erişebilir.
- Port, ya statik olarak tanımlanabilir ya da 30000 ile 32767 arasındaki bir aralıktan dinamik olarak alınabilir.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: NodePort
  selector:
    app: nginx
    env: prod
  ports:
    - nodePort: 32410
      protocol: TCP
      port: 80
      targetPort: 80
```

NodePort Service



Name:	example-prod
Selector:	app=nginx, env=prod
Type:	NodePort
IP:	10.96.28.176
Port:	<unset> 80/TCP
TargetPort:	80/TCP
NodePort:	<unset> 32410/TCP
Endpoints:	10.255.16.3:80, 10.255.16.4:80

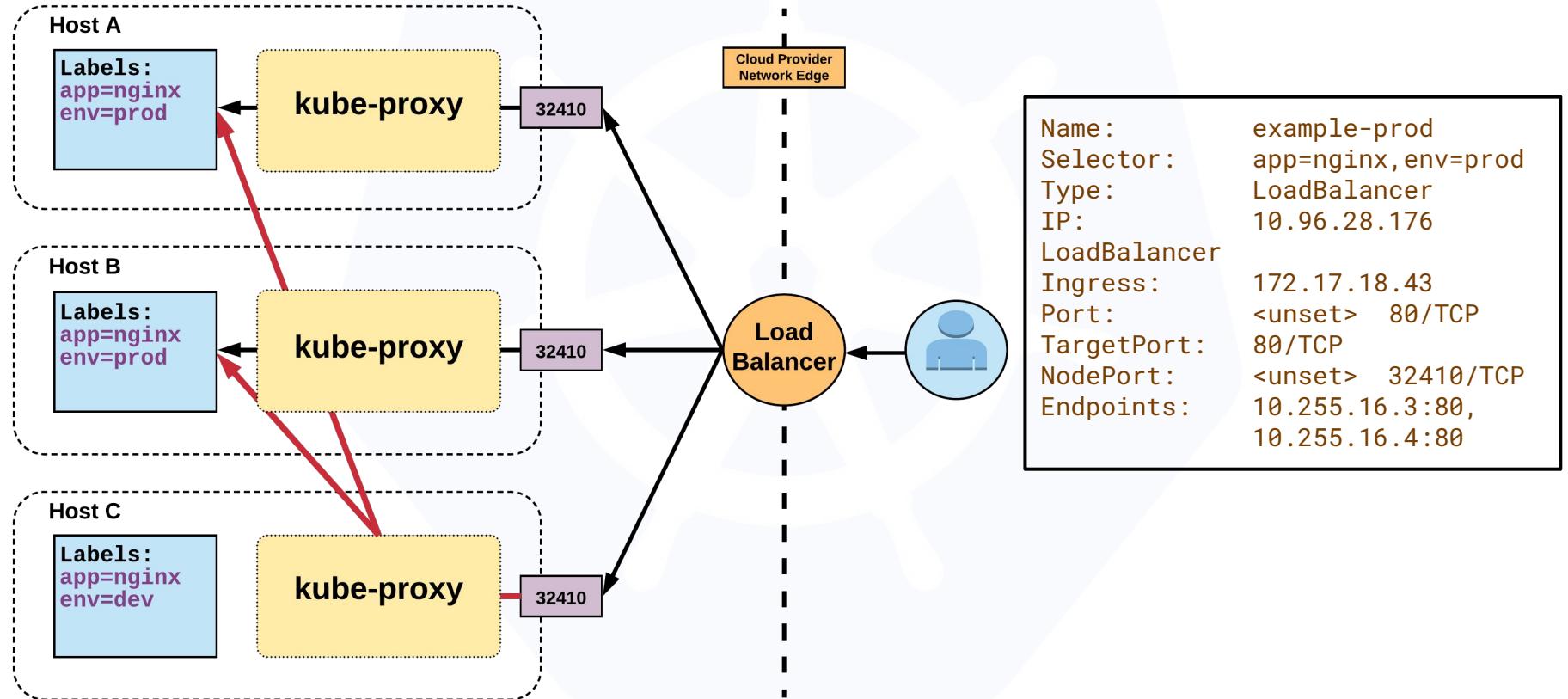
LoadBalancer Service



- **LoadBalancer** services **NodePort** hizmetlerini genişleterek daha gelişmiş bir erişim sağlar.
- gelen trafiği birden fazla node üzerinde dağıtarak yük dengeleme yapar, bu da uygulamanın daha iyi performans göstermesini sağlar.
- Loadbalancer cluster dışında uygulamalara erişim imkanı sağlar.
- LoadBalancer hizmetleri, genellikle bulut sağlayıcılarıyla entegre bir şekilde çalışır.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    env: prod
  ports:
    protocol: TCP
    port: 80
    targetPort: 80
```

LoadBalancer Service





ExternalName Service

- **ExternalName** is used to reference endpoints **OUTSIDE** the cluster.
- Creates an internal **CNAME** DNS entry that aliases another.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ExternalName
  spec:
    externalName: example.com
```

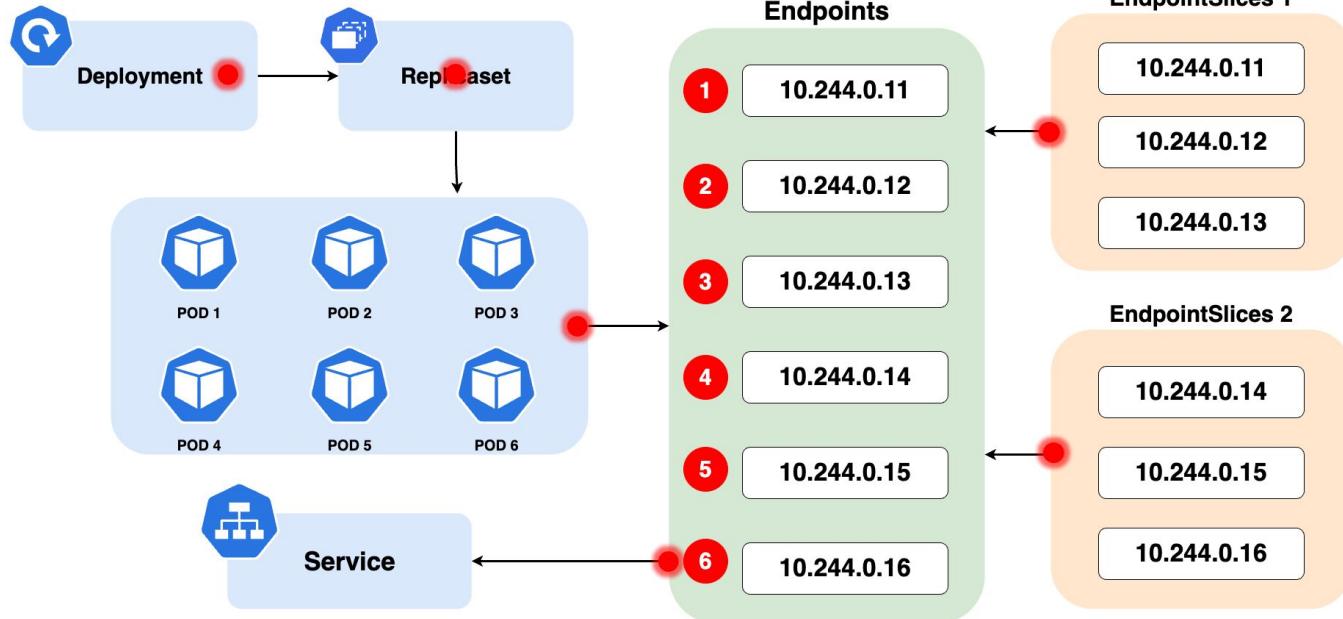


Endpoint



Endpoints & Endpoint Slices in K8s

Anvesh Muppeda





How kubernetes Endpoints work?

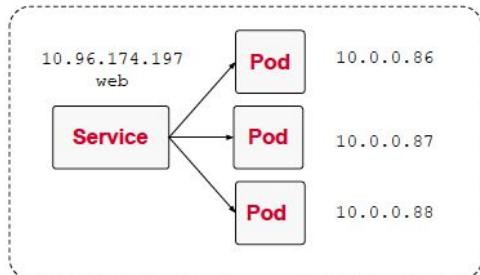
- 1. Service Creation:** When you create a service in Kubernetes, you define a selector that matches a set of pods. This selector determines which pods the service will route traffic to.
- 2. Endpoint Creation:** Kubernetes automatically creates an Endpoint object associated with the service. This object contains the IP addresses and ports of the pods that match the selector.
- 3. Updating Endpoints:** As pods are added or removed, or their statuses change, Kubernetes continuously updates the Endpoint object to reflect the current set of matching pods. This ensures that the service always routes traffic to the appropriate pods.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 10.0.0.1
      - ip: 10.0.0.2
    ports:
      - port: 80
```



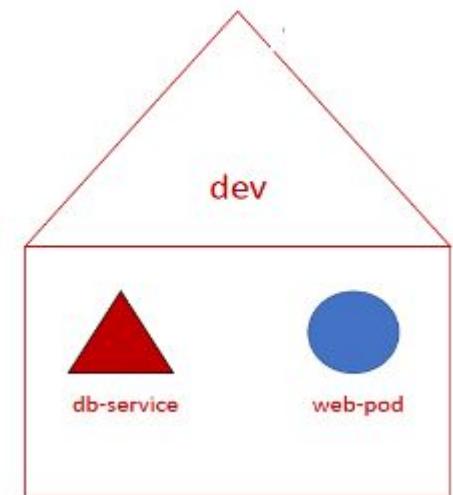
DNS For Services

db-service.dev.svc.cluster.local



Kubernetes DNS Service

Hostname	IP Address
web	10.96.174.197

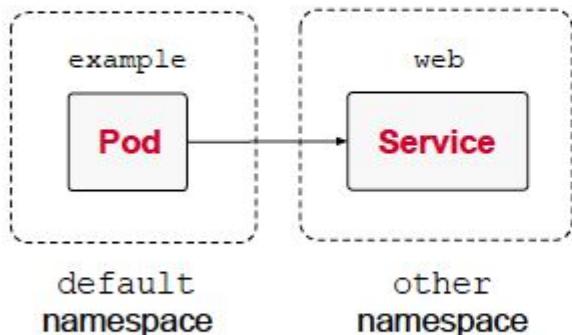


web.default.svc.cluster.local



Resolving a Service by DNS

Resolve by namespace, type, root domain from another namespace



```
$ curl http://web.other ← Namespace  
Hello World
```

```
$ curl http://web.other.svc ← Type  
Hello World
```

```
$ curl http://web.other.svc.cluster.local  
Hello World
```

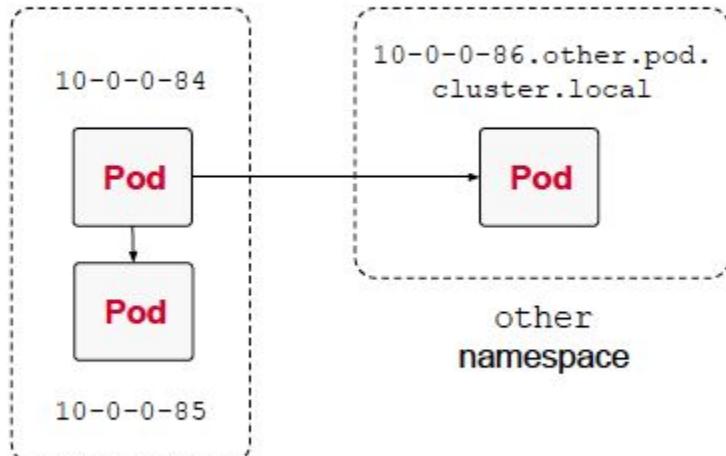
Root Domain





Resolving a Pod by DNS

DNS records are not created by default, resolve by IP address



default
namespace

```
$ curl 10-0-0-85
Hello World

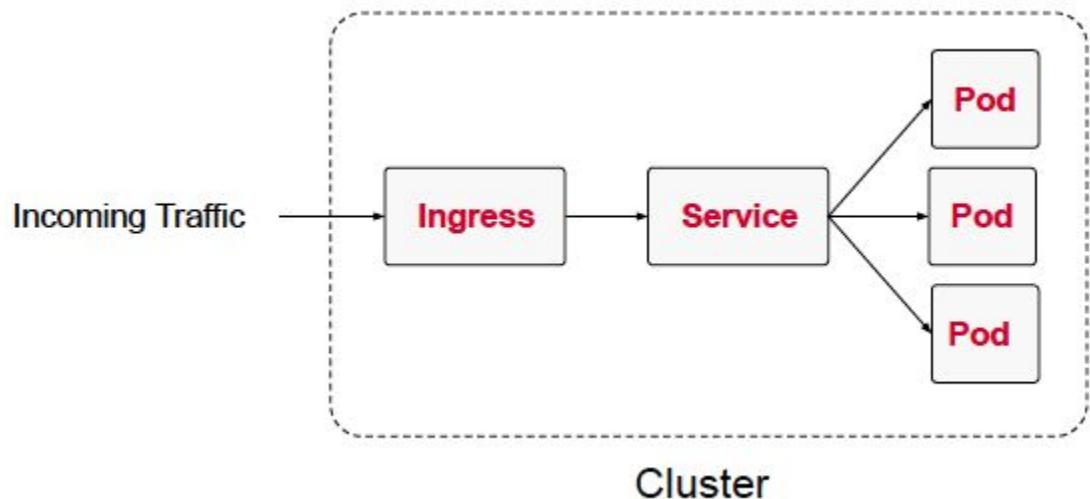
$ curl
10-0-0-86.other.pod.cluster
.local
Hello World
```



Ingress Nedir?

Ingress, Kubernetes ortamında dış dünyadan gelen HTTP ve HTTPS trafiğini yönetmek için kullanılan bir kaynak türüdür. Ingress, belirli kurallara göre gelen trafiği bir veya daha fazla servise yönlendirir. Bu, uygulamaların dışarıdan erişimini kolaylaştırır ve yük dengeleme, SSL sonlandırma gibi işlevleri destekler.

Ingress Controller: Ingress kaynaklarını yöneten bir bileşendir. Popüler Ingress Controller örnekleri arasında **NGINX**, **Traefik** ve **HAProxy** bulunur.





Deployment-Service-Ingress

Deployment configuration:

```
Deployment
  kind: Deployment
  metadata:
    name: nginx-deployment
    labels:
      app: nginx
  spec:
    replicas: 3
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
          - name: nginx
            image: nginx:1.7.9
            ports:
              - containerPort: 80
```

Service configuration:

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Ingress configuration:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
    - host: foo.bar.com
      http:
        paths:
          - path: /foo
            backend:
              serviceName: my-service
              servicePort: 80
        - path: /bar
            backend:
              serviceName: my-other-service
              servicePort: 80
```

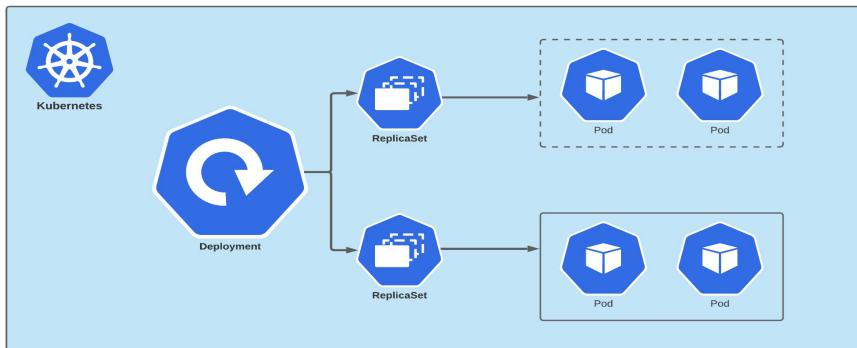
- Replicaset
- Deployment
- Deployment strategists



Replicaset

ReplicaSet, Kubernetes'te çalışan bir grup Pod'un belirtilen sayıda çalışmasını sağlamak için kullanılan bir Kubernetes kaynağıdır.

- Yalnızca Pod'ların sayısını yönetir.
- Pod'ları oluşturur ve ölçeklendirir, ancak güncellemeleri ve versiyon yönetimini yönetmez.
- Deployment'lar daha kapsamlı yönetim ve güncelleme özellikleri sunduğu için replicasetler çok kullanılmaz.



```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx
```



Deployment

- **Deployment**, Pod'ları yönetmek için geliştirilmiş bir kaynak türüdür. Yüksek erişilebilirlik ve ölçeklenebilirlik sağlamak amacıyla Pod'ların oluşturulması, güncellenmesi ve yönetilmesi için kullanılır.
- Pod'ların veri saklama veya paylaşma amacıyla vollumelerle bağlanması sağlar.
- Pod'lar, belirli etiketlerle seçilir. Bu etiketler, Deployment'in hangi Pod'ları oluşturacağını ve yöneteceğini belirler.
- Deployment, belirli sayıda Pod'ın her zaman çalışır durumda olmasını sağlar. replicas alanı ile bu sayı belirlenir.





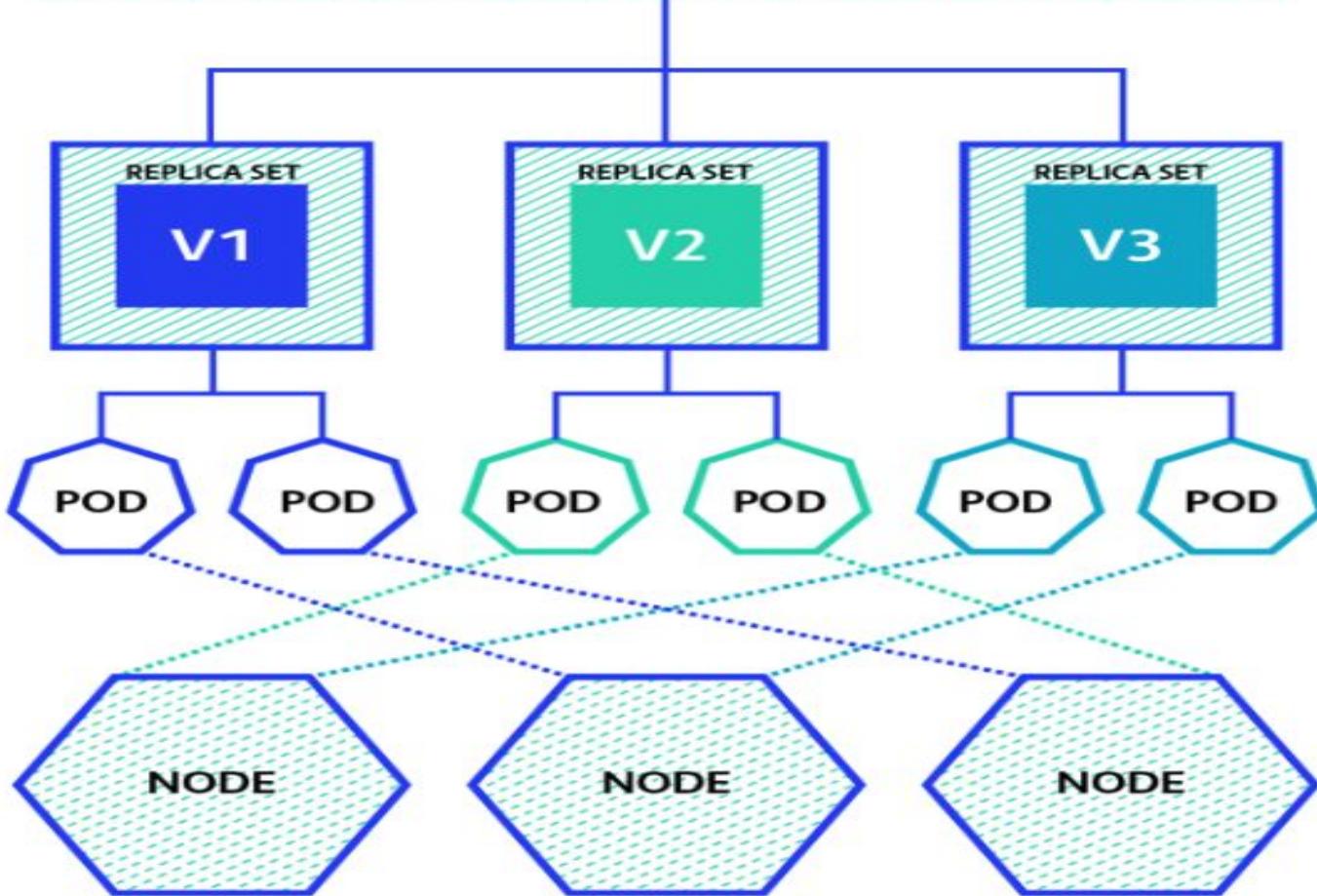
Deployment

- Güncellemelerde yaşanabilecek sorunlara karşı, önceki sürümü geri dönme yeteneği sağlar.

Güncelleme Stratejileri:

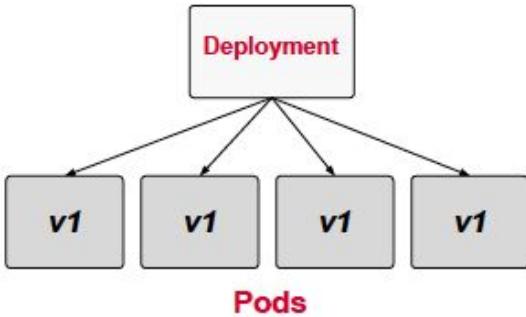
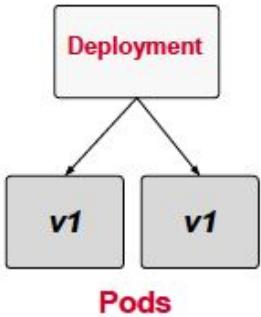
- **Recreate**: Mevcut Pod'lar sonlandırılır ve yeni Pod'lar oluşturulur.
- **RollingUpdate**: Pod'lar kademeli olarak güncellenir. `maxSurge` ve `maxUnavailable` parametreleri ile kontrol sağlanır.
- Pod'lar, belirli etiketlerle seçilir. Bu etiketler, Deployment'in hangi Pod'ları oluşturacağı ve yöneteceğini belirler.

DEPLOYMENT





Manuel Scaling a Deployment



```
# Check current deployment replicas
$ kubectl get deployments my-deploy
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
my-deploy  2        2           2           9h

# Scaling from 2 to 4 replicas
$ kubectl scale deployment my-deploy --replicas=4
deployment.extensions/my-deploy scaled

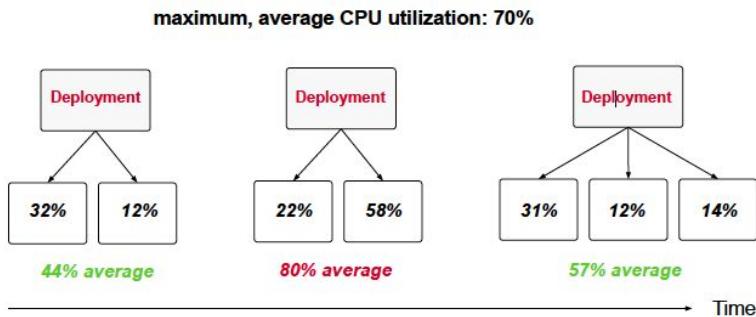
# Check the changed deployment replicas
$ kubectl get deployment my-deploy
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
my-deploy  4        4           4           9h
```



Autoscaling a Deployment

Autoscaling a Deployment

“Don’t make me think. Autoscale based on CPU utilization.”



Create Horizontal Pod Autoscaler

```
# Maintain average CPU utilization across all Pods of 70%
$ kubectl autoscale deployments my-deploy --cpu-percent=70
--min=1 --max=10
horizontalpodautoscaler.autoscaling/my-deploy autoscaled
```

```
# Check the current status of autoscaler
$ kubectl get hpa my-deploy
NAME      REFERENCE          TARGETS      MINPODS
MAXPODS   REPLICAS   AGE
my-deploy  Deployment/my-deploy  0%/70%    1
10        4           23s
```

Deployment



- **strategy**: Pod'ları güncelleme yöntemini, türüne göre açıklar. Geçerli seçenekler **Recreate** veya **RollingUpdate**'dir
 - **Recreate**: Yeni Pod'lar oluşturulmadan önce tüm mevcut Pod'lar sonlandırılır.
 - **RollingUpdate**: Pod'lar, **maxSurge** ve **maxUnavailable** parametrelerine göre sırayla güncellenir.

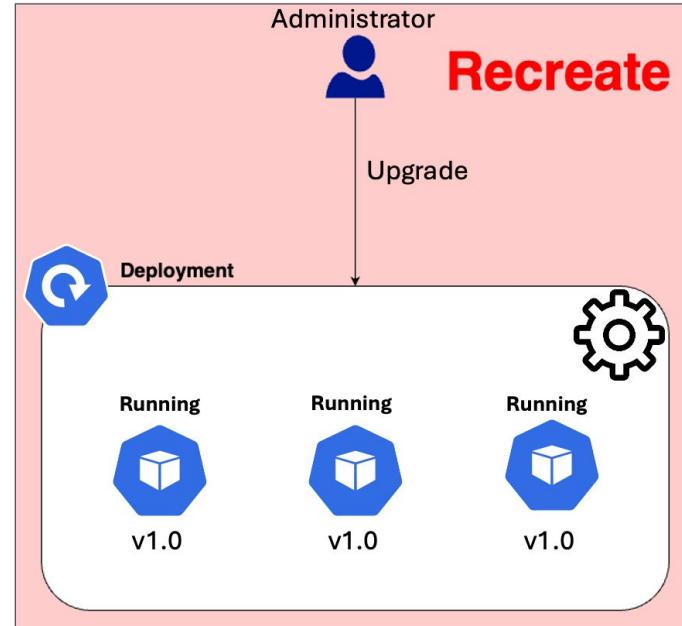
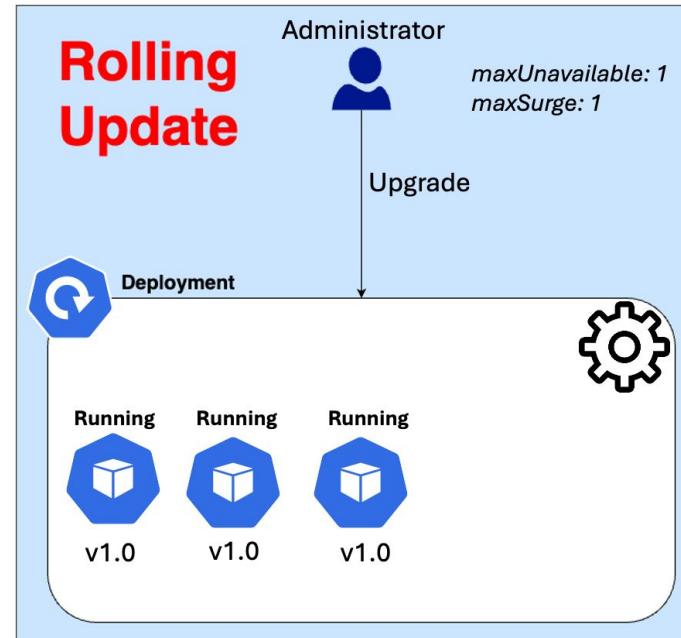
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```



Rolling update-Recreate



Anvesh Muppeda



Deployment Strategies



Rollout nedir?

- Bir uygulamanın yeni bir sürümünün dağıtımını başlatmak için kullanılır.
- Bir önceki sürümden farklı olarak yeni bir sürümde yer alan güncellemeleri sağlar.
- Yeni bir sürümün başarılı bir şekilde dağıtıımı için bir dizi adımı otomatikleştirir.
- Yeni bir sürümün yavaş yavaş tüm pod'lara dağıtılmasını ve eski sürümün aşamalı olarak kaldırılmasını sağlar.
- Uygulamanın yeni sürümüne geçiş sırasında yaşanabilecek olası kesintileri en aza indirir.



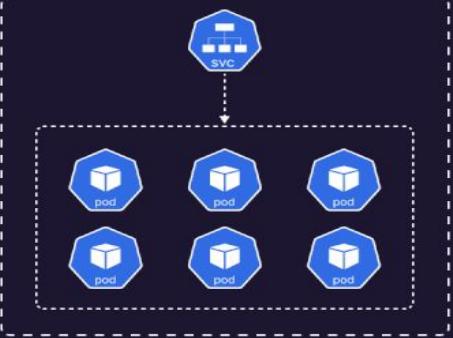
Rollback nedir?

- Uygulamanın bir önceki sürümüne geri dönmek için kullanılır.
- Bir uygulama sürümünün yanlışlıkla veya hatalı bir şekilde dağıtıldığı durumlarda kullanılır.
- Uygulama sürümünün hızlı bir şekilde geri alınmasını ve bir önceki çalışan sürümme geçiş yapılmasını sağlar.
- Geri alma işlemini otomatikleştirir ve hızlı bir şekilde tamamlamasını sağlar.

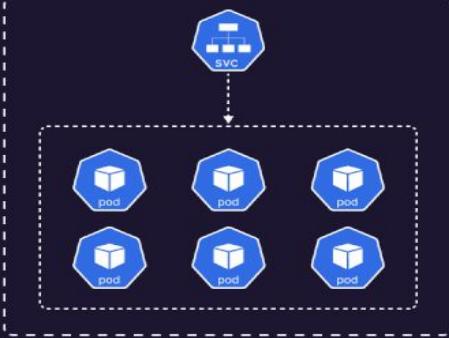


KUBERNETES DEPLOYMENT STRATEGIES

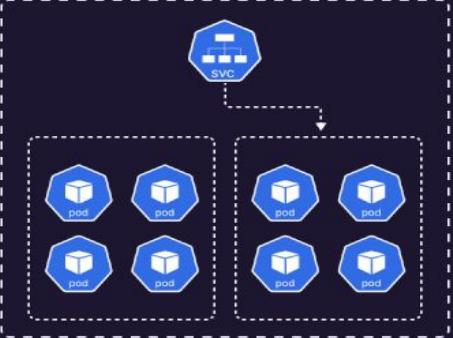
1. RECREATE



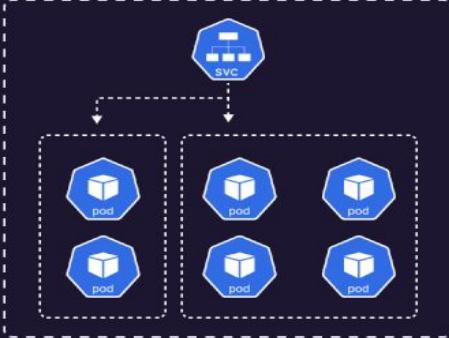
2. ROLLING



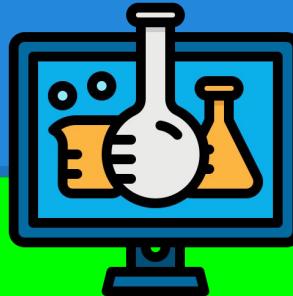
3. BLUE/GREEN



3. CANARY



Deployment



Storage

- **Volumes**
- **Persistent Volumes**
- **Persistent Volume Claims**
- **StorageClass**



Storage

Pod'lar kendi başlarına kullanışlıdır, ancak birçok iş yükü, konteynerler arasında veri alışverişi yapmayı veya bir tür veriyi kalıcı hale getirmeyi gerektirir.

Bu amaçla **Volumes**, **PersistentVolumes**,
PersistentVolumeClaims, ve **StorageClasses**'lar
kullanılır.



Volumes

- **Pod'un yaşam döngüsüne** bağlı olarak kullanılan depolama alanıdır.
- Bir Pod, bir veya daha fazla türde volume'a sahip olabilir.
- Pod içindeki tüm konteynerler, bu volume'ları kullanabilir.
- Pod yeniden başlatıldığında volume'lar varlıklarını korur, ancak verilerin kalıcılığı volume türüne bağlıdır.



Volume Types

Host Based

- EmptyDir
- HostPath
- Local

Distributed File System

- NFS
- Ceph
- Gluster
- FlexVolume
- PortworxVolume
- Amazon EFS
- Azure File System

Block Storage

- Amazon EBS
- OpenStack Cinder
- GCE Persistent Disk
- Azure Disk
- vSphere Volume

Others

- iScsi
- Flocker
- Git Repo
- Quobyte



Volumes

- **volumes**: Pod'a eklenmesi gereken volum nesnelerinin bir listesi. Listede bulunan her nesne, kendine özgü bir isme sahip olmalıdır.
- **volumeMounts**: Poda eklenen volumlerin isimleri ve mount edildiği klasörleri belirttiğimiz bölümdür

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-example
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
          readOnly: true
    - name: content
      image: alpine:latest
      command: ["/bin/sh", "-c"]
      args:
        - while true; do
            date >> /html/index.html;
            sleep 5;
        done
      volumeMounts:
        - name: html
          mountPath: /html
  volumes:
    - name: html
      emptyDir: {}
```



Persistent Volumes

- **PersistentVolume (PV)**: Kalıcı bir depolama kaynağını temsil eder.
- PV'ler, bir cluster genelinde kullanılan ve arka planda bir depolama sağlayıcısına (örneğin, NFS, GCEPersistentDisk, RBD vb.) bağlı kaynaklardır.
- Genellikle bir sistem yönetici tarafından sağlanır.
- Yaşam döngüleri bir Pod'dan bağımsız olarak yönetilir.
- Bir Pod'a doğrudan bağlanamaz. Bunun yerine, **PersistentVolumeClaim (PVC)** aracılığıyla kullanılır.

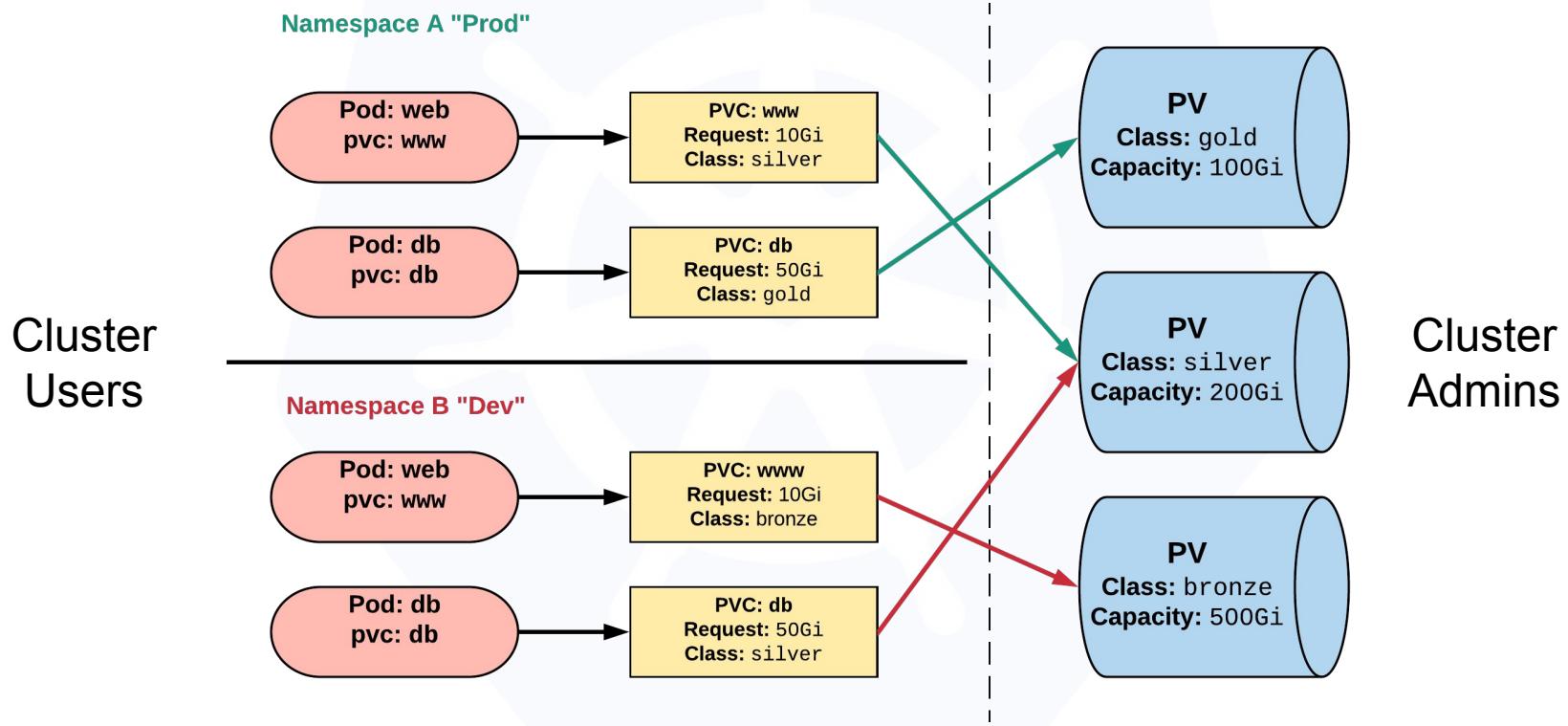


PersistentVolumeClaims

PersistentVolumeClaim (PVC): Belirli bir ad alanı (namespace) içinde yapılan depolama talebidir.

- PVC, doğrudan bir depolama kaynağına bağlanmak yerine, bir dizi gereksinimi karşılar.
- Bir uygulamanın depolama talebinin birçok farklı altyapı veya sağlayıcı üzerinde taşınabilir olmasını sağlar.
- PVC, bir uygulamanın gereksinim duyduğu depolama miktarı, erişim modu (okuma/yazma) ve performans gibi kriterleri belirtir.

Persistent Volumes and Claims

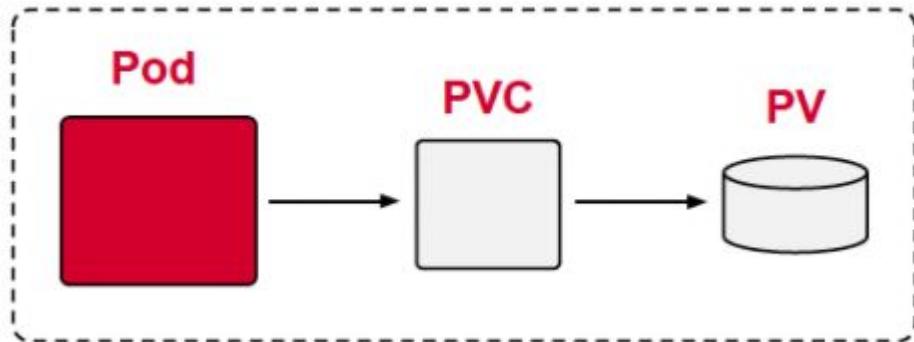




Mounting a Claim

```
apiVersion: v1
kind: Pod
metadata:
  name: clarus-pod
  labels:
    app: clarus-web
spec:
  volumes:
    - name: clarus-pv-storage
      persistentVolumeClaim:
        claimName: clarus-pv-claim
  containers:
    - name: clarus-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: clarus-pv-storage
```

1. Create Static Persistent Volumes OR Dynamic Volumes (using Storage Class)
2. Persistent Volume Claim is created and bound static and dynamic volumes.
3. Pods refer PVC to mount volumes inside the Pod.



PersistentVolume



- **capacity.storage**: Kullanılabilir toplam depolama alanını belirtir.
- **volumeMode**: Volume'un türünü belirler; **Filesystem** veya **Block**.
- **accessModes**: Volume'a erişim yöntemlerini belirten bir listedir. Desteklenen erişim modları şunlardır:
 - **ReadWriteOnce**
 - **ReadOnlyMany**
 - **ReadWriteMany**

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfsserver
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /exports
    server: 172.22.0.42
```



PersistentVolume

- **persistentVolumeReclaimPolicy:** Silinen PVC'ler için uygulanan davranışı belirler. Seçenekler şunlardır:
 - **Retain** - Manuel temizleme gerektirir.
 - **Delete** - sağlayıcı tarafından otomatik olarak silinir.
- **storageClassName:** PVC'lerin başvurabileceği isteğe bağlı bir **storage class** adıdır.
- **mountOptions:** PersistentVolume (PV) için isteğe bağlı olarak belirtilen mount seçenekleridir.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfsserver
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /exports
    server: 172.22.0.42
```

PVs and PVCs with Selectors



```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-selector-example
  labels:
    type: hostpath
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-selector-example
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      type: hostpath
```

PVs and PVCs



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: clarus-pv-vol
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/home/ubuntu/pv-data"
```

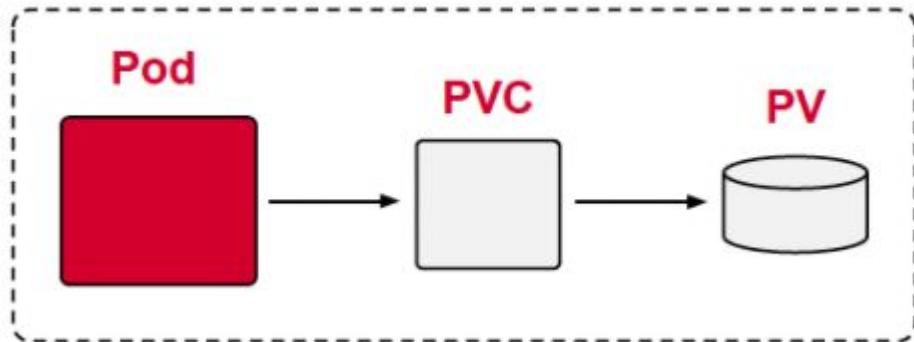
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: clarus-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```



Mounting a Claim

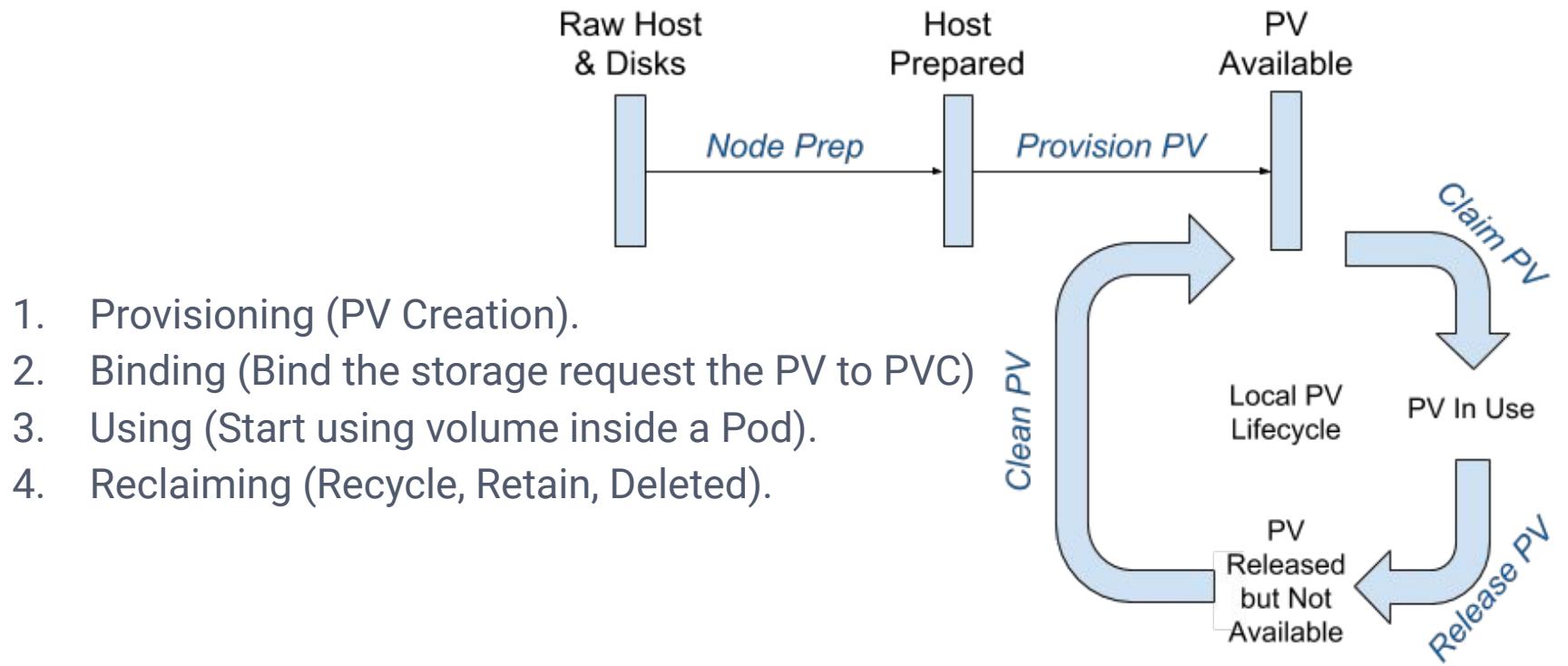
```
apiVersion: v1
kind: Pod
metadata:
  name: clarus-pod
  labels:
    app: clarus-web
spec:
  volumes:
    - name: clarus-pv-storage
      persistentVolumeClaim:
        claimName: clarus-pv-claim
  containers:
    - name: clarus-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: clarus-pv-storage
```

1. Create Static Persistent Volumes OR Dynamic Volumes (using Storage Class)
2. Persistent Volume Claim is created and bound static and dynamic volumes.
3. Pods refer PVC to mount volumes inside the Pod.





PersistentVolume life cycle is





PV Phases(Aşamaları)

Available

PV oluşturulduktan sonra, herhangi bir PVC tarafından talep edilene kadar kullanılabilir durumda bekler.

Bound

Bir PVC, PV'ye bağlandığında bu aşama gerçekleşir.

Released

PVC silindiğinde PV serbest bırakılır

Failed

PV'nin geri kazanılmasıında bir sorun yaşandığında, bu aşamaya girer



StorageClass

- Storage classes farklı türdeki depolama kaynaklarını bir araya getirerek, yöneticilere belirli politikalar ve özellikler belirleme imkanı sunar
- Depolama sınıfları, dış depolama sistemi ile iş birliği içinde çalışarak dinamik olarak depolama sağlama yeteneği sunar. Bu, PVC talep edildiğinde **otomatik** olarak uygun bir PV'nin oluşturulmasını sağlar.
- Depolama sınıfları sayesinde, PVC talep edildiğinde uygun bir PV otomatik olarak sağlanabilir.



Available StorageClasses

- AWSElasticBlockStore
- AzureFile
- AzureDisk
- CephFS
- Cinder
- FC
- Flocker
- GCEPersistentDisk
- Glusterfs
- iSCSI
- Quobyte
- NFS
- RBD
- VsphereVolume
- PortworxVolume
- ScaleIO
- StorageOS
- Local



Internal Provisioner



StorageClass

- **provisioner**: Defines the '*driver*' to be used for provisioning of the external storage.
- **parameters**: A hash of the various configuration parameters for the provisioner.
- **reclaimPolicy**: The behaviour for the backing storage when the PVC is deleted.
 - **Retain** - manual clean-up
 - **Delete** - storage asset deleted by provider

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
  provisioner: kubernetes.io/gce-pd
  parameters:
    type: pd-standard
    zones: us-central1-a, us-central1-b
  reclaimPolicy: Delete
```



StorageClass

1.PVC, StorageClass'tan bir talepte bulunur.

uid: 9df65c6e-1a69-11e8-ae10-080027a3682b

PVC: www
Request: 10Gi
Class: standard

Class:
Standard

2.StorageClass, talebi dış depolama sistemi ile API aracılığıyla işleme alır.

API

External
Storage
System

PVC-WWW
Class: standard
Capacity: 10Gi

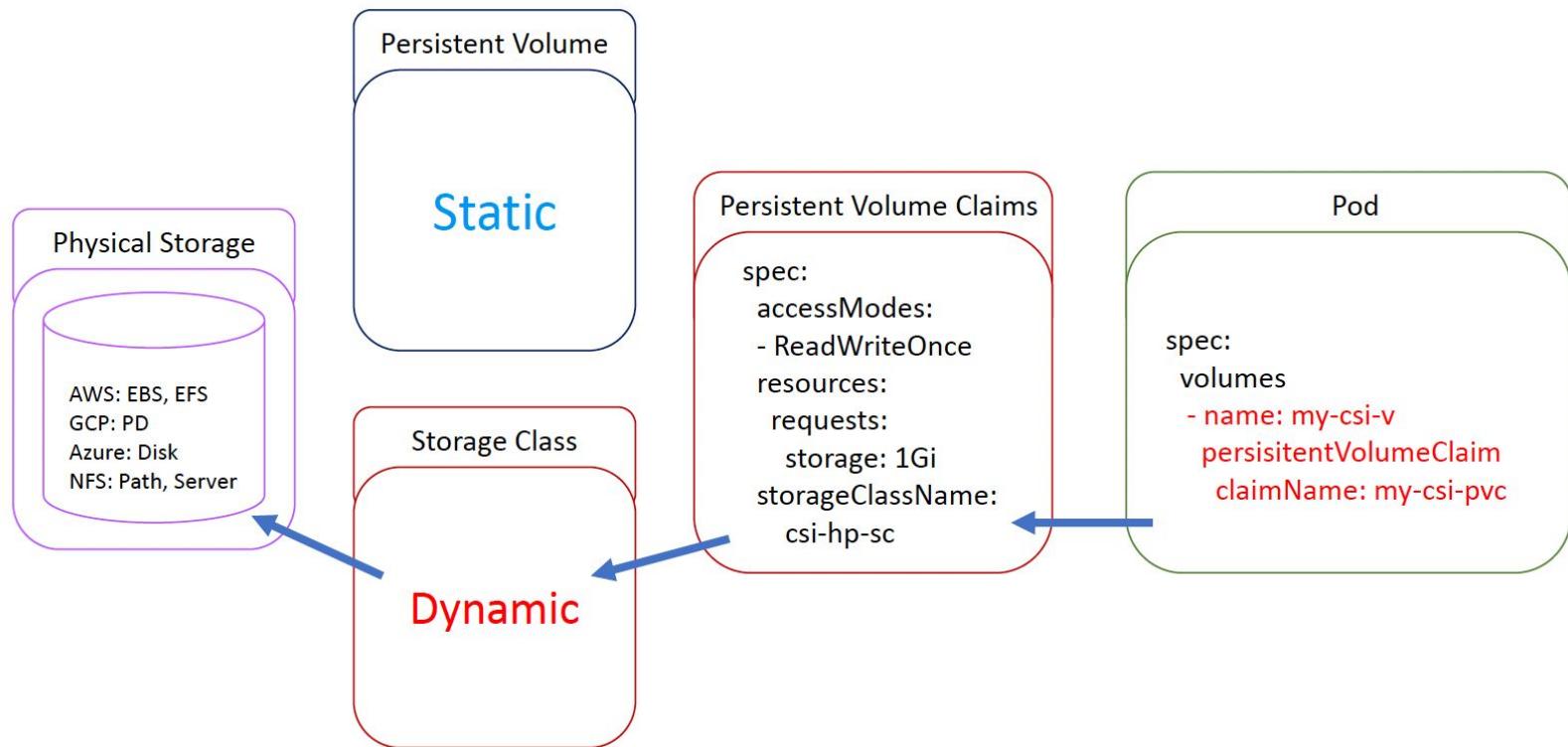
pv: pvc-9df65c6e-1a69-11e8-ae10-080027a3682b

4.Oluşturulan PV, talep eden PVC'ye bağlanır.

3. External storage system, PVC talebini kesin olarak karşılayan bir PV oluşturur.

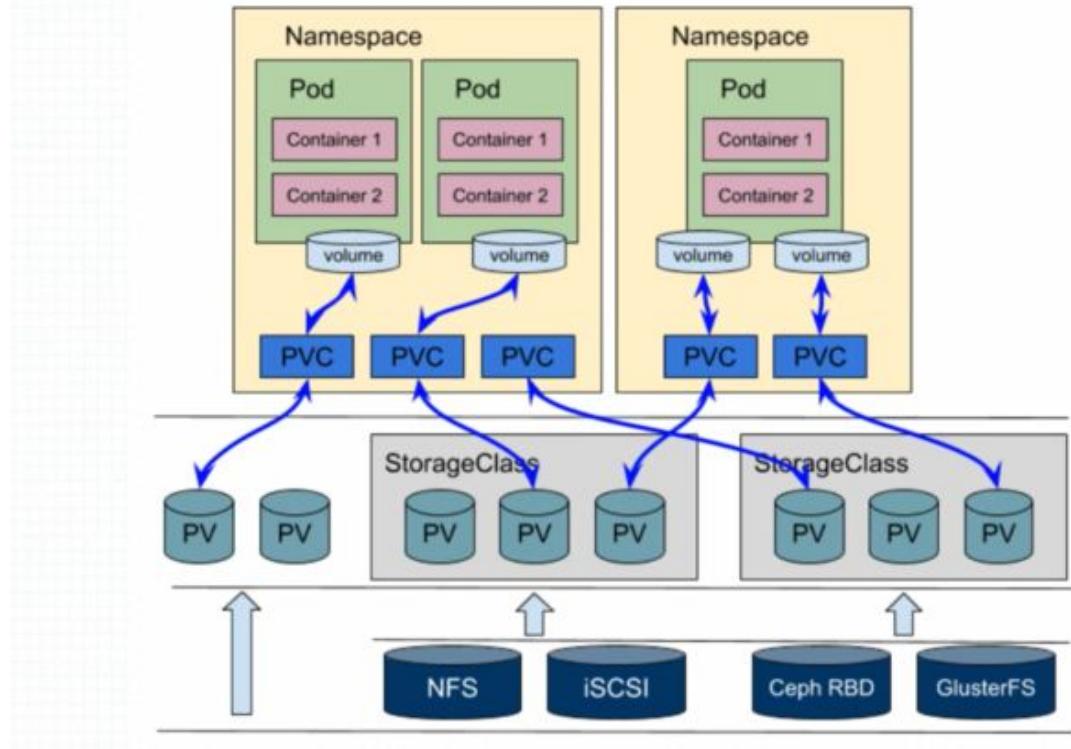


Storage Class, PV, PVC and Pods





● Binding PV to PVC





Kubernetes Volume summary

Volume

- A **Persistent Volume** is the physical storage available.
- **Storage Class** is used to configure custom Storage option (nfs, cloud storage) in the cluster. They are the *foundation of Dynamic Provisioning*.
- **Persistent Volume Claim** is used to mount the required storage into the Pod.

Persistent Volume

Access Mode

- **ReadOnlyMany**: Can be mounted as read-only by many nodes
- **ReadWriteOnce**: Can be mounted as read-write by a single node
- **ReadWriteMany**: Can be mounted as read-write by many nodes

Storage Class

Persistent Volume Claim

Volume Mode

- There are two modes
 - **File System** and or
 - raw Storage **Block**.
- Default is **File System**.

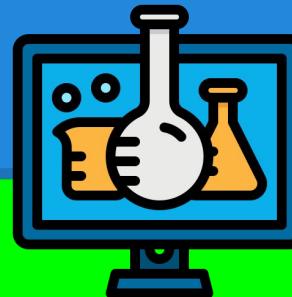
Reclaim Policy

Retain: The volume will need to be reclaimed manually

Delete: The associated storage asset, such as AWS EBS, GCE PD, Azure disk, or OpenStack Cinder volume, is deleted

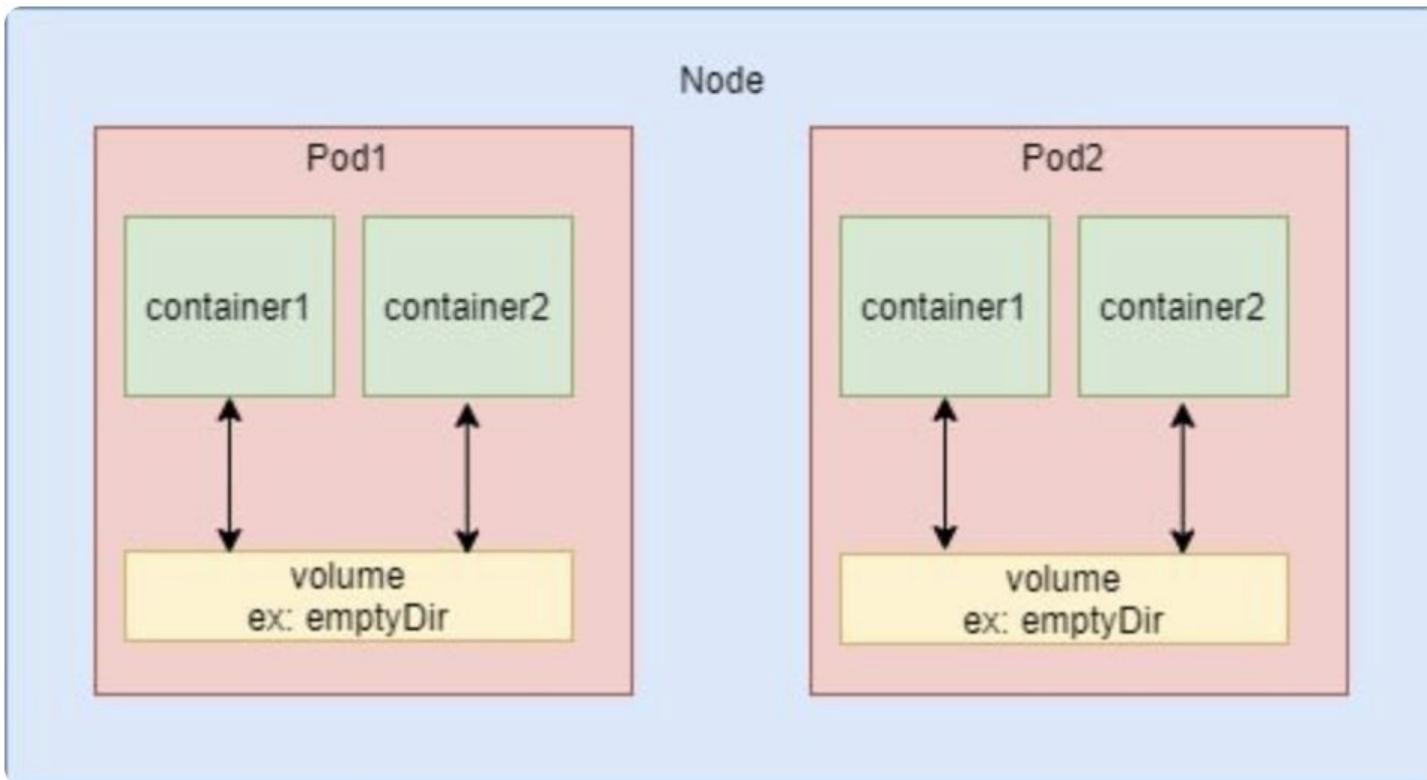
Recycle: Delete content only (rm -rf /volume/*) - **Deprecated**

Working with Volumes



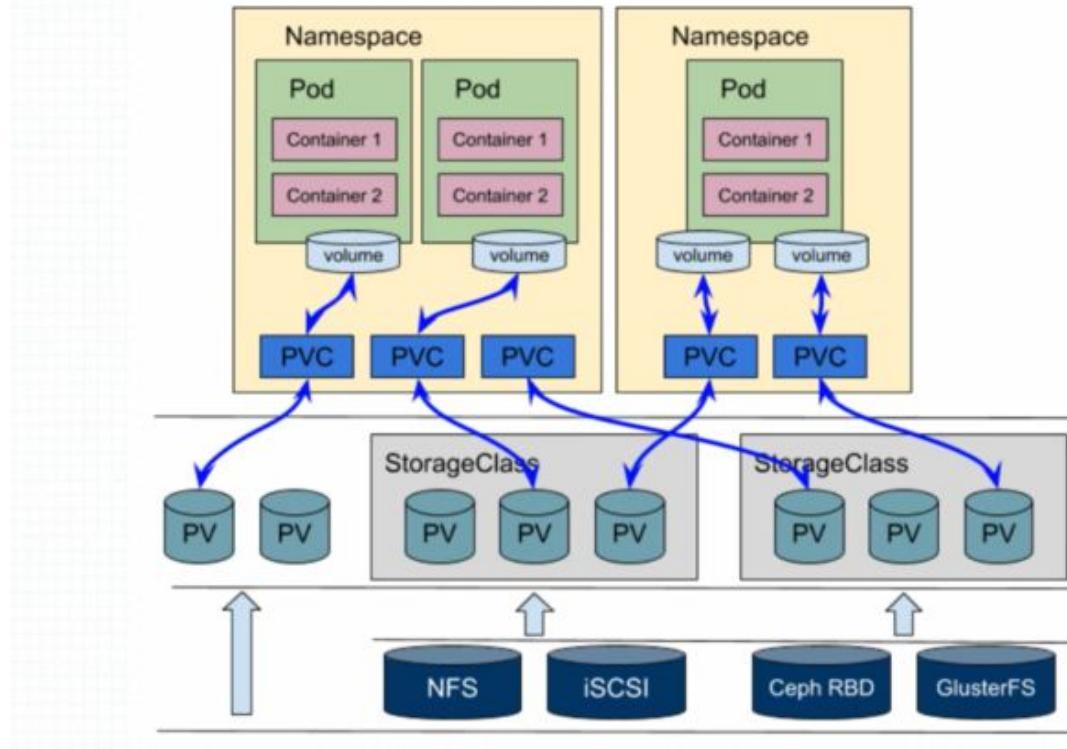


2- EmptyDir LAB





2-Binding PV to PVC LAB



Resource Quotas

Concepts and Resources



Resource Quota Nedir?

Resource Quota, belirli bir namespace içindeki kaynakların (CPU, bellek, pod sayısı vb.) sınırlarını tanımlamak için kullanılır. Bu, bir ekip veya uygulamanın belirli kaynakları aşmasını önler.

Temel Özellikler:

- **Kaynak Kontrolü:**
 - Namespace içindeki toplam kaynak kullanımını sınırlar.
- **Adil Paylaşım:**
 - Farklı uygulamalar veya ekipler arasında kaynakların adil bir şekilde dağıtılmasını sağlar.
- **Hata Önleme:**
 - Aşırı kaynak kullanımını önleyerek uygulamaların kararlılığını artırır.



Resource Quota ile Sınırlandırılabilen Kaynaklar

Kubernetes'te Resource Quota ile kontrol edilebilecek kaynaklar şunlardır:

- **Pod sayısı:** Bir namespace içinde oluşturulabilecek pod sayısını sınırlayabilirsiniz.
- **CPU ve bellek:** Bir namespace içinde kullanılabilen CPU ve bellek miktarını sınırlayabilirsiniz.
- **PersistentVolumeClaims (PVC):** Depolama için oluşturulacak kalıcı hacim taleplerini sınırlayabilirsiniz.
- **Services:** Namespace içinde oluşturulabilecek servis sayısını sınırlayabilirsiniz.
- **ConfigMap ve Secrets:** Oluşturulacak ConfigMap ve Secret nesnelerinin sayısını sınırlayabilirsiniz.



Namespace Bazlı Kısıtlamalar

CPU ve RAM Kısıtlamaları

Kubernetes, konteynerler için CPU ve bellek taleplerini belirleyerek kaynak kısıtlamalarını yönetir. Bu, uygulamaların ihtiyaç duyduğu kaynakların belirlenmesi anlamına gelir.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: my-resource-quota
  namespace: my-namespace
spec:
  hard:
    requests.cpu: "4"          # Toplamda en fazla 4 CPU
    requests.memory: "8Gi"      # Toplamda en fazla 8 GiB bellek
    limits.cpu: "10"            # Toplamda en fazla 10 CPU limiti
    limits.memory: "16Gi"       # Toplamda en fazla 16 GiB bellek limiti
```



Container Kısıtlamalarını Tanımlama

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: my-namespace
spec:
  containers:
    - name: mycontainer
      image: nginx
      resources:
        requests:
          cpu: "0.5"          # En az 0.5 CPU talep eder
          memory: "200Mi"     # En az 200Mi bellek talep eder
        limits:
          cpu: "1"            # En fazla 1 CPU kullanabilir
          memory: "500Mi"     # En fazla 500Mi bellek kullanabilir
```

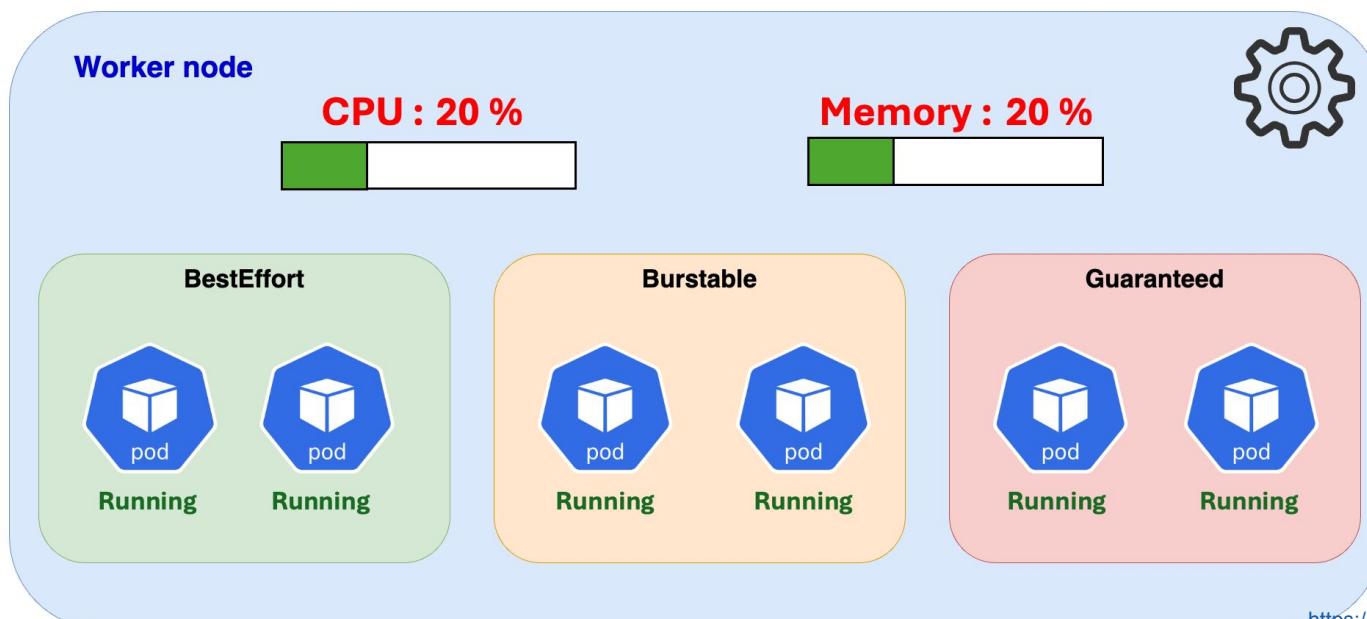


Kubernetes defines three QoS classes for Pods

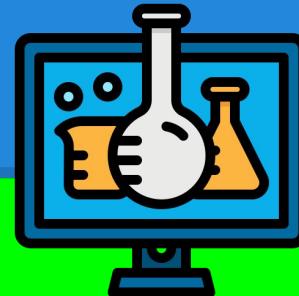


Quality of Service(QoS) in Kubernetes

Anvesh Muppeda



ResourceQuota



Workloads

- **DaemonSet**
- **StatefulSet**



DaemonSet

DaemonSet, Kubernetes'te bir podun, her bir node üzerinde çalıştırılmasını sağlamak amacıyla kullanılan bir nesnedir. Eğer bir DaemonSet oluşturursanız, bu DaemonSet'e bağlı pod, cluster'daki her bir node üzerinde otomatik olarak başlatılır. Bu, özellikle log toplama, sistem monitor edilmesi gibi her bir node'da çalışan işlemler için kullanılır.

DaemonSet'in Kullanım Alanları

DaemonSet, genellikle aşağıdaki senaryolarda kullanılır:



- Node Başına İzleme Uygulamaları:** Örneğin, her node üzerinde çalışan bir log toplama agenti (Fluentd, Filebeat) veya bir monitoring agenti (Prometheus Node Exporter) dağıtmak.
- Node Başına Network veya Güvenlik Uygulamaları:** Her node'da çalışan network yönetimi veya güvenlik yazılımları (Cilium, Calico gibi network plugin'leri).
- Node'a Özel İşlemler:** Her node'da sabit çalışan işlemler veya alt yapı hizmetleri (örn: dosya sistemi sağlama hizmetleri).



DaemonSet'in Özellikleri ve Avantajları

- Her Node'da Pod Çalıştırma:** DaemonSet ile bir pod her node üzerinde otomatik olarak çalışır.
- Node Eklendiğinde Otomatik Pod Başlatma:** Cluster'a yeni bir node eklenirse, DaemonSet otomatik olarak bu node'da bir pod başlatır.
- Cluster'dan Node Çıkarıldığında Pod Silme:** Node cluster'dan ayrılrsa, o node üzerinde çalışan pod otomatik olarak silinir.
- Node Pool ve Tolerations ile Esneklik:** DaemonSet, belirli node'larda çalışacak şekilde özelleştirilebilir. Örneğin, **tolerations** ve **nodeSelector** kullanarak DaemonSet'i yalnızca belirli bir label'a sahip nodelar üzerinde çalıştırabilirsiniz.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: logging
spec:
  selector:
    matchLabels:
      name: fluentd
  template:
    metadata:
      labels:
        name: fluentd
    spec:
      containers:
      - name: fluentd
        image: fluent/fluentd:latest
        resources:
          limits:
            memory: "200Mi"
            cpu: "500m"
        volumeMounts:
        - name: varlog
          mountPath: /var/log
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
```



StatefulSet

StatefulSet, Kubernetes'te **durum bilgisi olan** uygulamaların yönetilmesi için kullanılan bir nesne türdürü. StatefulSet, her bir pod'un kimliğini (isim, ağ kimliği ve disk gibi) korur ve düzgün bir şekilde ölçeklendirme, yeniden başlatma ve güncelleme yapmayı sağlar.

StatefulSet'in Temel Özellikleri:

- Kimlik Koruma:** Her bir pod'a benzersiz bir kimlik atanır ve bu kimlik, pod sırasıyla korunur (örn. my-app-0, my-app-1). Pod yeniden başlatılsa bile bu kimlik kaybolmaz.
- Kalıcı Depolama:** Her pod için bağımsız bir kalıcı depolama alanı (PersistentVolume) kullanılır, böylece veriler pod yeniden başlasa bile korunur.
- Sıralı Dağıtım ve Güncelleme:** Pod'lar sırasıyla oluşturulur ve güncellenir. Bir pod başarılı olmadan bir sonraki pod başlatılmaz.
- Sıralı Ölçeklendirme:** Pod sayısı artırılırken veya azaltılırken son pod'dan başlanarak sıralı bir şekilde ölçeklenir.
- Ağ Kimliği:** Her pod, benzersiz bir ağ kimliğine sahip olur ve bu kimlik pod yeniden başlatılsa bile korunur.



StatefulSet Ne zaman Kullanılır?

StatefulSet, aşağıdaki durumlarda tercih edilir:

- **Veritabanları**: MySQL, PostgreSQL gibi durum bilgisi olan uygulamaların çalıştırılmasında kullanılır.
- **Dağıtık Sistemler**: Cassandra, Kafka, Zookeeper gibi dağıtık veri tabanları ve sistemler.
- **Sıralı ve Bağlı Podlar**: Podlar arasında bir bağımlılık varsa veya her pod'un sıralı olarak çalışması gerekiyorsa (örneğin, bir primary-replica ilişkisi).



StatefulSet



```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-example
spec:
  replicas: 2
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: stateful
  serviceName: app
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  template:
    metadata:
      labels:
        app: stateful
<continued>
```

<continued>

```
spec:
  containers:
    - name: nginx
      image: nginx:stable-alpine
      ports:
        - containerPort: 80
      volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
    - metadata:
        name: www
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: standard
    resources:
      requests:
        storage: 1Gi
```



Özellik	Deployment	StatefulSet	DaemonSet
Pod İsimlendirme	Pod isimleri rastgele oluşturulur ve yenilenir.	Pod isimleri belirli bir sıra numarasıyla tanımlanır (örneğin: mysql-0 , mysql-1).	Pod isimleri genellikle standarttır ve tekil ID ile isimlendirilmez.
Sıra ve Güncellemeler	Güncellemeler paralel olarak yapılır.	Güncellemeler sıralı olarak yapılır; bir Pod güncellenmeden diğerine geçilmez.	Güncellemeler paralel olarak yapılır, ancak her node'da bir Pod çalıştırır.
Ağ Erişimi	Pod'lar arasında doğrudan IP adresi ile erişim yoktur.	Pod'lar arasında sabit ve kalıcı DNS isimleri ile erişim sağlanır.	Pod'lar, her node üzerinde çalıştığı için, node'lar arasında doğrudan erişim sağlar.
Depolama	Genellikle ephemeral (geçici) depolama kullanılır.	Kalıcı (persistent) depolama ile kullanılır ve her Pod'a özel PersistentVolumeClaim tanımlanabilir.	Pod'lar genellikle ephemeral (geçici) depolama kullanır, ancak PersistentVolume kullanabilir.
Pod'ların Yönetimi	Pod'lar aynı anda kopyalanır ve ölçeklenir.	Pod'lar tek tek ve belirli bir sırayla kopyalanır ve ölçeklenir.	Her node üzerinde bir Pod çalıştırılır; Pod sayısı node sayısına bağlıdır.



Deployment Kullanım Durumları

- **Stateless Uygulamalar:** Web sunucuları ve API'ler gibi durum (state) gerektirmeyen uygulamalar.
- **Ölçeklenebilirlik:** Pod sayısını yatay olarak artırıp azaltma gereksinimi.
- **Rolling Updates:** Uygulamayı kesintisiz olarak güncelleme ihtiyacı.
- **Basit Konfigürasyon:** Karmaşık durum yönetimi gerektirmeyen durumlar.
- **Geçici Depolama:** Verinin geçici olduğu ve kalıcılık gerekmeyen senaryolar.



Statefulset Kullanım Durumları

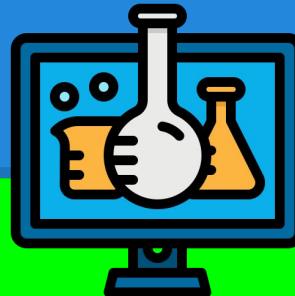
- **Stateful Uygulamalar:** Veritabanları, mesajlaşma sistemleri gibi durum (state) gerektiren uygulamalar.
- **Sabit DNS İsimleri:** Her Pod'un sabit bir DNS ismine sahip olması gereken durumlar.
- **Kalıcı Depolama:** Her Pod için kalıcı ve ayrılmış depolama alanı gereksinimi.
- **Sıralı Güncellemeler:** Pod'ların sıralı ve belirli bir sırayla güncellenmesi gereken durumlar.
- **Yüksek Durum Yönetimi:** Durum yönetimi ve sıralı başlatma gereksinimi olan senaryolar.



Daemenset Kullanım Durumları

- **Node Bazında Uygulamalar:** Her node üzerinde çalışması gereken uygulamalar (örneğin, log toplayıcılar).
- **Node'a Özgү İhtiyaçlar:** Her node'da tekil bir Pod'un bulunması gereken durumlar.
- **Sistem ve İzleme Araçları:** Node seviyesinde sistem izleme ve yönetim araçları.
- **Sabit Çalışan Pod'lar:** Her node'da sürekli olarak çalışan uygulamalar.
- **Otomatik Dağıtım:** Yeni node'lara otomatik olarak Pod dağıtıımı yapılması gereken durumlar.

Statefulset Daemonset



Configuration

- **ConfigMap**
- **Secret**

Concepts and Resources

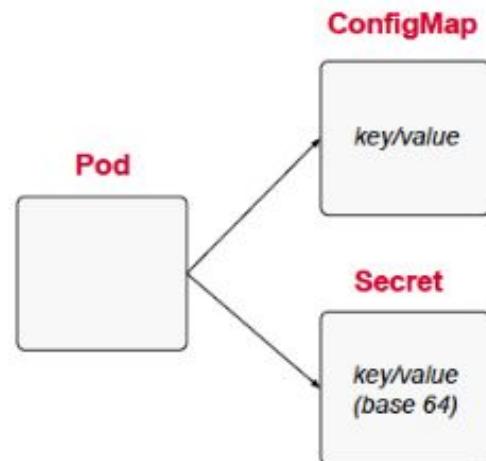


Centralized Configuration Data

Kubernetes, uygulama yapılandırmalarını yönetmek için çeşitli nesneler kullanır. Bu nesneler, uygulamaların çalışma zamanında ihtiyaç duyduğu yapılandırma verilerini merkezi bir şekilde saklamaya yardımcı olur.

- **Nesne Referansları Kullanarak Enjekte Etme:**

- ConfigMap ve Secret nesneleri, pod'lara çalışma zamanında yapılandırma verisi enjekte etmek için nesne referansları aracılığıyla kullanılır.
- Örneğin, bir pod oluşturulurken, bir ConfigMap veya Secret'tan değerler, ortam değişkenleri veya dosya olarak pod'a dahil edilebilir.





Configuration

Kubernetes, yapılandırmayı uygulama veya konteynerden ayırmak için entegre bir desen sunar.

Bu desen, iki Kubernetes bileşenini kullanır: **ConfigMaps** ve **Secrets**.

- **ConfigMaps:** Yapılandırma verilerini yönetir ve uygulamaların bu verilere erişimini sağlar. ConfigMaps, genellikle yapılandırma dosyalarını, ortam değişkenlerini ve diğer konfigürasyon bilgilerini içerir.
- **Secrets:** Hassas verileri, örneğin şifreleri, API anahtarlarını ve diğer gizli bilgileri güvenli bir şekilde yönetir. Secrets, bu bilgilerin şifrelenmiş ve güvenli bir şekilde saklanmasılığını sağlar.



ConfigMap

Kubernetes içinde dışa aktarılmış veriler saklanır.

Bu veriler çeşitli yollarla referans alınabilir:

- **Ortam değişkeni:** Uygulama konteynerlerine ortam değişkenleri aracılığıyla erişim sağlar.
- **Komut satırı argümanı (ortam değişkeni aracılığıyla):** Ortam değişkenlerinden elde edilen veriler, komut satırı argümanları olarak kullanılabilir.
- **Volume mount olarak dosya olarak enjekte edilme:** Veriler, bir volume mount aracılığıyla dosya olarak konteyner içine enjekte edilebilir.

Veriler şu şekilde oluşturulabilir:

- **Manifests:** Yapılandırma dosyaları kullanılarak.
- **Literals:** Doğrudan belirtilmiş değerler.
- **Volume mount:** Dizinlerden veri alınarak.
- **Dosyalar:** Doğrudan dosyalardan veri oluşturularak.

ConfigMap



`data`: ConfigMap içeriğinin anahtar-değer çiftlerini içerir.

Her anahtar, bir yapılandırma öğesini temsil eder ve bu anahtarın karşılığı olan değer, yapılandırma verisinin içeriğini belirtir.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  state: Michigan
  city: Ann Arbor
  content: |
    Look at this,
    its multiline!
```

ConfigMap Example

All produce a **ConfigMap** with the same content!

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  city: Ann Arbor
  state: Michigan
```

```
$ kubectl create configmap literal-example \
> --from-literal="city=Ann Arbor" --from-literal=state=Michigan
configmap "literal-example" created
```

```
$ cat info/city
Ann Arbor
$ cat info/state
Michigan
$ kubectl create configmap file-example --from-file=cm/city --from-file=cm/state
configmap "file-example" created
```

```
$ cat info/city
Ann Arbor
$ cat info/state
Michigan
$ kubectl create configmap dir-example --from-file=cm/
configmap "dir-example" created
```



Secret

- Secrets, yapılandırma bilgilerini saklamak için ConfigMap ile benzer şekilde çalışır ancak hassas verilerin yönetimi için özel olarak tasarlanmıştır.
- Secrets içindeki veriler, **base64** formatında kodlanarak saklanır.
- Secrets, **etcd** veri tabanında şifrelenmiş olarak saklanır. (**if configured!**).
- Kullanıcı Adı/Şifreler, Sertifikalar veya Konteynerde Saklanmaması Gereken Diğer Hassas Bilgiler İçin İdealdır
- Bir Manifest, Literal, volume mount veya Dosyalar Üzerinden Oluşturulabilir:

Secret



- **type**: Kubernetes içinde üç farklı Secrets türü bulunmaktadır:
 - **docker-registry** - Genellikle Docker registry veya diğer konteyner kayıt defterleri için kullanıcı adı ve şifre bilgilerini içerir.
 - **generic/Opaque** - Farklı kaynaklardan alınan literal değerleri saklar.
 - **tls** - sertifika temelli secret
- **data**: **Secrets** içindeki **data** alanı, base64 ile kodlanmış anahtar-değer çiftlerini içerir.

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-secret
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password: bXlwYXNzd29yZA==
```

Secret Example

All produce a **Secret** with the same content!

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-example
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password: bXlwYXNzd29yZA==
```

```
$ kubectl create secret generic literal-secret \
> --from-literal=username=example \
> --from-literal=password=mypassword
secret literal-secret created
```

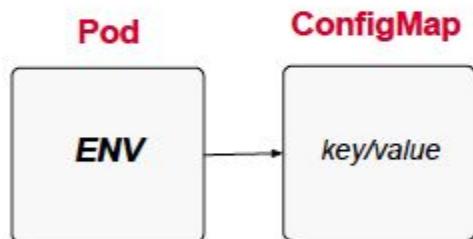
```
$ cat info/username
example
$ cat info/password
mypassword
$ kubectl create secret generic dir-secret --from-file=secret/
Secret "file-secret" created
```

```
$ cat secret/username
example
$ cat secret/password
mypassword
$ kubectl create secret generic file-secret --from-file=secret/username --from-file=secret/password
Secret "file-secret" created
```

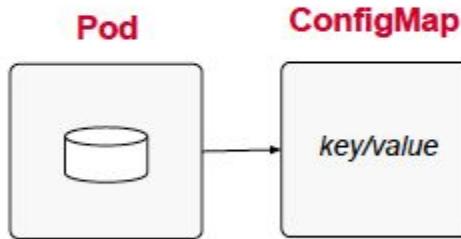


Mounting a ConfigMap

Two options for consuming data



Injected as environment variables



Mounted as volume

Injecting in a Command



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cm-cmd-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cm-cmd-example
  template:
    metadata:
      labels:
        app: cm-cmd-example
    spec:
      containers:
        - name: mypod
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args: ["echo Hello ${CITY}!"]
          env:
            - name: CITY
              valueFrom:
                configMapKeyRef:
                  name: manifest-example
                  key: city
            restartPolicy: Always
```

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-example
type: Opaque
data:
  username: dXNlcg== # base64 encoded 'user'
  password: cGFzc3dvcmQ= # base64 encoded 'password'
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  city: "Istanbul"
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: secret-cmd-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: secret-cmd-example
  template:
    metadata:
      labels:
        app: secret-cmd-example
    spec:
      containers:
        - name: mypod
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args: ["echo Hello ${USERNAME}!"]
          env:
            - name: USERNAME
              valueFrom:
                secretKeyRef:
                  name: manifest-example
                  key: username
            restartPolicy: Always
```



Injecting as a Volume

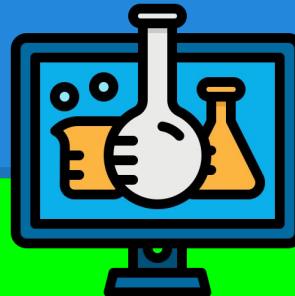
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: secret-vol-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: secret-vol-example
  template:
    metadata:
      labels:
        app: secret-vol-example
    spec:
      containers:
        - name: mypod
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args: ["cat /mysecret/username"]
      volumeMounts:
        - name: secret-volume
          mountPath: /mysecret
  volumes:
    - name: secret-volume
      secret:
        secretName: manifest-example
```

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-example
type: Opaque
data:
  username: dXNlcg== # base64 encoded 'user'
  password: cGFzc3dvcmQ= # base64 encoded 'password'
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  city: "Istanbul"
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cm-vol-example
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cm-vol-example
  template:
    metadata:
      labels:
        app: cm-vol-example
    spec:
      containers:
        - name: mypod
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args: ["cat /myconfig/city"]
      volumeMounts:
        - name: config-volume
          mountPath: /myconfig
  volumes:
    - name: config-volume
      configMap:
        name: manifest-example
```

Working with Configmaps and Secrets



Workloads

- Job
- Cronjob



Job

- Job controller, bir veya daha fazla pod'un başarılı bir şekilde çalışmasını ve sonlanmasını sağlar..
- Belirtilen tamamlanma (**completion**) ve/veya paralellik (**parallelism**) koşulunu sağlamak için işin yürütülmesine devam eder.
- İş tamamlanana kadar pod'lar temizlenmez. İşin kendisi silinene kadar pod'lar tutulur.



Job



- **backoffLimit**: Bir işin başarısızlık sayısını belirtir. Bu sayede işin tamamlanması için belirlenen sayıda başarısızlık yaşanırsa iş başarısız(**failed**) olarak kabul edilir.
- **completions**: İşin tamamlanması için gerekli olan başarılı pod sayısını belirtir.
- **parallelism**: Aynı anda çalışacak pod sayısını belirtir.
- **spec.template.spec.restartPolicy**: Jobs sadece **Never** veya **OnFailure** değerini destekler.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-example
spec:
  backoffLimit: 4
  completions: 4
  parallelism: 2
  template:
    spec:
      restartPolicy: Never
    <pod-template>
```

Job



```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-example
spec:
  backoffLimit: 4
  completions: 4
  parallelism: 2
  template:
    spec:
      containers:
        - name: hello
          image: alpine:latest
          command: ["/bin/sh", "-c"]
          args: ["echo hello from $HOSTNAME!"]
  restartPolicy: Never
```

```
$ kubectl get pods --show-all
NAME           READY   STATUS    RESTARTS   AGE
job-example-dvxd2  0/1    Completed  0          51m
job-example-hknns  0/1    Completed  0          52m
job-example-tphkm  0/1    Completed  0          51m
job-example-v5fvq  0/1    Completed  0          52m
```

```
$ kubectl describe job job-example
Name:           job-example
Namespace:      default
Selector:       controller-uid=19d122f4-1576-11e8-a4e2-080027a3682b
Labels:         controller-uid=19d122f4-1576-11e8-a4e2-080027a3682b
                job-name=job-example
Annotations:    <none>
Parallelism:   2
Completions:   4
Start Time:    Mon, 19 Feb 2018 08:09:21 -0500
Pods Statuses: 0 Running / 4 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=19d122f4-1576-11e8-a4e2-080027a3682b
            job-name=job-example
  Containers:
    hello:
      Image:  alpine:latest
      Port:   <none>
      Command:
        /bin/sh
        -c
      Args:
        echo hello from $HOSTNAME!
      Environment:  <none>
      Mounts:      <none>
      Volumes:     <none>
  Events:
    Type  Reason        Age   From           Message
    ----  ----        --   --            --
    Normal SuccessfulCreate 52m  job-controller  Created pod: job-example-v5fvq
    Normal SuccessfulCreate 52m  job-controller  Created pod: job-example-hknns
    Normal SuccessfulCreate 51m  job-controller  Created pod: job-example-tphkm
    Normal SuccessfulCreate 51m  job-controller  Created pod: job-example-dvxd2
```



CronJob

CronJobs, düzenli olarak tekrar eden işlerin belirli bir zaman diliminde çalıştırılmasını sağlar ve yönetir.

CronJobs, işlerin belirli bir zaman aralığında çalışmasını sağlar. Zamanlama, Unix **cron** formatına göre tanımlanır.

UTC Kullanımı: CronJobs zamanlaması yalnızca UTC (Koordinatlı Evrensel Zaman) kullanılarak ayarlanır.



CronJob



- **schedule:** İşin ne zaman çalışacağını belirten cron zamanlama ifadesidir.
- **successfulJobHistoryLimit:** Bu özellik, başarılı olarak tamamlanmış işlerin ne kadarının saklanacağını belirler
- **failedJobHistoryLimit:** Bu özellik, başarısız olarak tamamlanmış işlerin ne kadarının saklanacağını belirler.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        <pod template>
```

CronJob

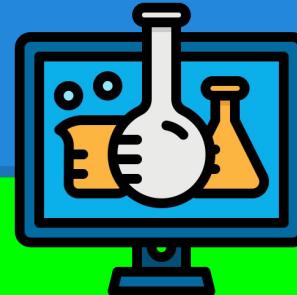


```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        spec:
          containers:
            - name: hello
              image: alpine:latest
              command: ["/bin/sh", "-c"]
              args: ["echo hello from $HOSTNAME!"]
      restartPolicy: Never
```

```
$ kubectl get jobs
NAME           DESIRED   SUCCESSFUL   AGE
cronjob-example-1519053240  4          4           2m
cronjob-example-1519053300  4          4           1m
cronjob-example-1519053300  4          4           26s
```

```
$ kubectl describe cronjob cronjob-example
Name:           cronjob-example
Namespace:      default
Labels:         <none>
Annotations:   <none>
Schedule:      */1 * * * *
Concurrency Policy: Allow
Suspend:        False
Starting Deadline Seconds: <unset>
Selector:       <unset>
Parallelism:   2
Completions:   4
Pod Template:
  Labels:  <none>
  Containers:
    hello:
      Image:  alpine:latest
      Port:   <none>
      Command:
        /bin/sh
        -c
      Args:
        echo hello from $HOSTNAME!
      Environment:  <none>
      Mounts:      <none>
      Volumes:     <none>
  Last Schedule Time: Mon, 19 Feb 2018 09:54:00 -0500
  Active Jobs:    cronjob-example-1519052040
Events:
  Type  Reason          Age   From           Message
  ----  ----
  Normal SuccessfulCreate 3m   cronjob-controller  Created job cronjob-example-1519051860
  Normal SawCompletedJob  2m   cronjob-controller  Saw completed job: cronjob-example-1519051860
  Normal SuccessfulCreate 2m   cronjob-controller  Created job cronjob-example-1519051920
  Normal SawCompletedJob  1m   cronjob-controller  Saw completed job: cronjob-example-1519051920
  Normal SuccessfulCreate 1m   cronjob-controller  Created job cronjob-example-1519051980
```

Job CronJob



HELM



Helm Nedir?

Helm, Kubernetes uygulamalarını yönetmek ve dağıtmak için kullanılan bir paket yöneticisidir. Helm, uygulamaların Kubernetes kümesine kolayca kurulmasını, güncellenmesini ve yönetilmesini sağlar.



Temel Kavramlar

Chart: Helm uygulamalarının paketleridir. Bir chart, uygulama bileşenlerini, yapılandırmalarını ve bağımlılıklarını tanımlar. Her chart, bir dizin yapısına sahiptir ve Kubernetes kaynaklarını tanımlayan YAML dosyalarını içerir.

Release: Bir chart'ın belirli bir sürümünün Kubernetes kümesine dağıtılmasıdır. Her release, bir isimle tanımlanır ve istenildiğinde güncellenebilir veya silinebilir.

Repository: Helm chart'larının saklandığı bir yerdir.

HELM CHART TUTORIAL



devopscube.com



Helm Chart Release Information
(Dynamic)



values.yaml

```
● ● ●
replicaCount: 2

image:
  repository: nginx
  tag: "1.16.0"
  pullPolicy: IfNotPresent

service:
  name: nginx-service
  type: ClusterIP
  port: 80
  targetPort: 9000

env:
  name: dev
```

chart.yaml

```
● ● ●
apiVersion: v2
name: nginx-chart
description: Nginx Chart
type: application
version: 0.1.0
appVersion: "1.0.0"
```



Neden Helm Kullanılır?

Kolay Kurulum:

- Uygulamaların kurulumu ve yapılandırılması için karmaşık YAML dosyaları yerine, tek bir chart ile işlem yapılabilir.

Versiyon Kontrolü:

- Uygulamaların farklı versiyonlarını yönetmek ve güncellemek kolaydır.

Bağımlılık Yönetimi:

- Uygulama bağımlılıklarını otomatik olarak yönetir. Bir chart, diğer chart'lara bağımlı olabilir.

Yeniden Kullanılabilirlik:

- Chart'lar, birden fazla ortamda (geliştirme, test, üretim) tekrar kullanılabilir.



Helm kurulumu ve yapısı

```
# Homebrew ile macOS üzerinde kurulum  
brew install helm  
  
# Linux için curl kullanarak kurulum  
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

Helm ile wordpress kurulumu

1-Helm Repository Ekleme

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

2- WordPress için Chart Kurulumu

```
helm install my-wordpress bitnami/wordpress
```

3. Kurulum Sonrası Bilgiler

```
kubectl get pods
```

Örnek Helm Chart Yapısı

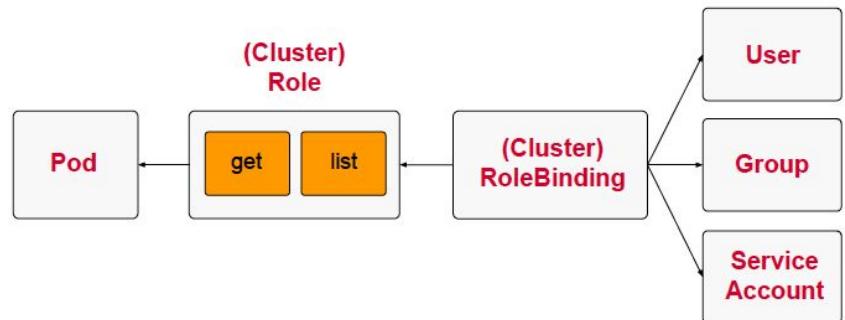
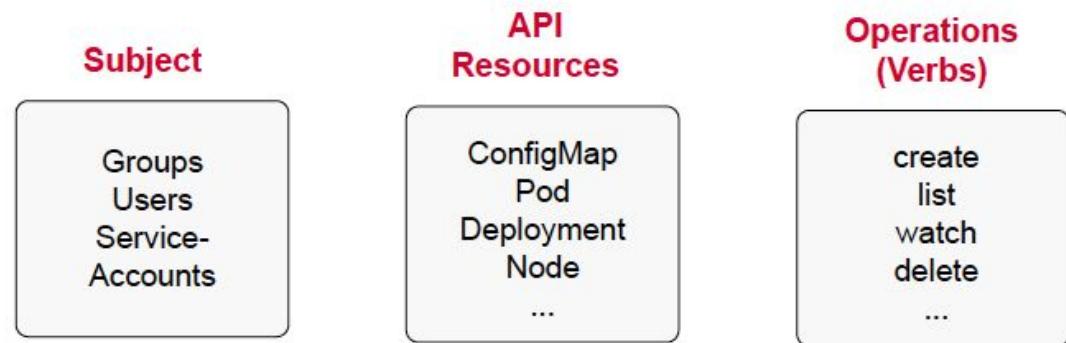
```
mychart/  
  Chart.yaml      # Chart bilgileri  
  values.yaml    # Varsayılan yapılandırmalar  
  templates/  
    deployment.yaml # Deployment tanımı  
    service.yaml   # Service tanımı
```

RBAC



RBAC (Role Based Access Control)

RBAC, sistemdeki kullanıcıların ve uygulamaların kaynaklara erişimini yönetmek için kullanılan bir erişim kontrol modelidir. Kubernetes, RBAC'ı kullanarak güvenli bir erişim yönetimi sağlar.





RBAC Role

Role : Belirli bir ad alanında kaynaklara erişim izni tanımlayan bir nesnedir.

Örnek izinler: get, list, create, update, delete.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
rules:
- apiGroups: []
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

API Resources

Operations



RBAC RoleBinding

RoleBinding: Bir Role'ü belirli bir kullanıcı veya grup ile ilişkilendiren nesnedir. Kullanıcılara veya gruplara, belirli bir ad alanındaki kaynaklara erişim sağlar.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

One or many subjects

Reference to role



RBAC ClusterRole + ClusterRoleBinding

ClusterRole ve **ClusterRoleBinding**, Kubernetes'te rol tabanlı erişim kontrolü için küme düzeyinde kullanılan iki önemli bileşendir.

ClusterRole

- Tüm ad alanlarındaki kaynaklara erişim sağlayabilir.
- Düğümlere ve kaynak olmayan uç noktalar gibi öğelere erişim izni verebilir.

ClusterRoleBinding

- Bir ClusterRole'ü, kümedeki tüm ad alanlarına bağlar. Bu sayede, belirli bir rolü tüm ad alanlarında kullanmak mümkün hale gelir.