daire, renklilik, simetri, bakışım, kalıp, desen, düzen içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**T.C. CUMHURBAŞKANLIĞI**

**GENEL SEKRETERLİĞİ**

**YAZILIM DAİRE BAŞKANLIĞI**

**CONTAİNER / SANALLAŞTIRMA ÖDEVİ**

**RECEP ÖZTÜRK**

**08.08.2025**

# İÇİNDEKİLER

[İÇİNDEKİLER 2](#_Toc205565786)

[1. SANALLAŞTIRMA NEDİR? 3](#_Toc205565787)

[1.1. Sanallaştırmanın Temel Prensibi 3](#_Toc205565788)

[1.2. Hypervisor Nedir? 3](#_Toc205565789)

[1.3. Sanallaştırmanın Faydaları 4](#_Toc205565790)

[1.4. Geleneksel Sanallaştırma ile Container Yaklaşımı Arasındaki Temel Fark 4](#_Toc205565791)

[2. CONTAİNER TEKNOLOJİSİ NEDİR? 4](#_Toc205565792)

[2.1. Container ve Geleneksel Sanallaştırma Karşılaştırması 4](#_Toc205565793)

[2.2. Container Yapısının Bileşenleri 5](#_Toc205565794)

[2.3. Neden Container Kullanılır? 5](#_Toc205565795)

[2.4. Container Teknolojilerinin Evrimi 6](#_Toc205565796)

[3. DOCKER, CONTAİNERD, CRI-O KARŞILAŞTIRMASI 6](#_Toc205565797)

[3.1. Docker Nedir? 6](#_Toc205565798)

[3.2. containerd Nedir? 6](#_Toc205565799)

[3.3. CRI-O Nedir? 7](#_Toc205565800)

[3.4. Karşılaştırmalı Tablo 7](#_Toc205565801)

[3.5. Güncel Eğilim 8](#_Toc205565802)

[4. CONTAİNER KULLANIM SENARYOLARI 8](#_Toc205565803)

[4.1. Mikroservis Mimarisi 8](#_Toc205565804)

[4.2. CI / CD (Sürekli Entegrasyon / Sürekli Dağıtım) 8](#_Toc205565805)

[4.3. Taşınabilirlik ve Ortam Bağımsızlığı 9](#_Toc205565806)

[4.4. Bulut Bilişim ve Orkestrasyon 9](#_Toc205565807)

[4.5. Eğitim, Test ve Otomasyon Ortamları 9](#_Toc205565808)

[4.6. Edge Computing ve IoT 10](#_Toc205565809)

[4.7. Uygulama Modernizasyonu ve Legacy Sistemler 10](#_Toc205565810)

[5. ASP.NET CORE WEB API GELİŞTİRME VE DOCKER CONTAİNER OLUŞTURMA 10](#_Toc205565811)

[6. KUBERNETES NEDİR? DOCKER ve CONTAINERD ile FARKI 12](#_Toc205565812)

[6.1. Kubernetes’in Temel Özellikleri 13](#_Toc205565813)

[6.2. Docker ve Kubernetes İlişkisi 13](#_Toc205565814)

[6.3. containerd ile Kubernetes İlişkisi 13](#_Toc205565815)

[6.4. Karşılaştırmalı Tablo 13](#_Toc205565816)

[6.5. Sonuç 14](#_Toc205565817)

[7. KUBERNETES CLUSTER ÜZERİNDE API CONTAINER’ININ ÇALIŞTIRILMASI 14](#_Toc205565818)

[7.1. Cluster Oluşturulması 14](#_Toc205565819)

[7.2. Docker Image’ın Oluşturulması 15](#_Toc205565820)

[7.3. Kubernetes Deployment ve Servis Tanımı 16](#_Toc205565821)

[7.4. API’ye Erişim ve Pod Yanıtları 16](#_Toc205565822)

[8. SONUÇ 17](#_Toc205565823)

[9. KAYNAKÇA 18](#_Toc205565824)

# SANALLAŞTIRMA NEDİR?

Sanallaştırma, fiziksel bir bilgisayar sisteminin donanım kaynaklarını yazılım aracılığıyla izole edilerek birden çok sanal ortamda kullanılabilir hale getirme işlemidir. Bu teknoloji sayesinde bir fiziksel sunucu üzerinde birden çok bağımsız işletim sistemi veya uygulama aynı anda çalıştırılabilir. Sanallaştırma, bilgi işlem altyapısında kaynak kullanım verimliliğini artırırken, ölçeklenebilirlik ve bakım kolaylığı gibi avantajlar da sağlar.

## Sanallaştırmanın Temel Prensibi

Sanallaştırma teknolojisinin temelinde, donanım kaynaklarının (CPU, RAM, disk, ağ arayüzü gibi) bir sanal makine monitörü (hypervisor) tarafından mantıksal olarak ayrılması yatar. Her bir sanal ortam, izole bir sistem gibi çalışır ve kendi işletim sistemine, uygulamalarına ve dosya sistemine sahip olabilir.

## Hypervisor Nedir?

Hypervisor, fiziksel donanım ile sanal makineler arasında bir ara katman olarak görev yapan yazılımdır. İki ana türü vardır:

* Type 1 (Bare-metal) Hypervisor: Doğrudan donanım üzerinde çalışır. VMware ESXi, Microsoft Hyper-V ve Xen bu tür hypervisor’lara örnektir.
* Type 2 (Hosted) Hypervisor: Mevcut bir işletim sistemi üzerinde çalışır. VirtualBox ve VMware Workstation buna örnektir.

## Sanallaştırmanın Faydaları

Sanallaştırma teknolojileri, özellikle kurumsal bilgi sistemlerinde aşağıdaki avantajları sağlamaktadır:

* Kaynakların Daha Etkin Kullanımı: Tek bir fiziksel makinede birden fazla hizmetin çalışmasına olanak sağlar.
* İzolasyon: Her bir sanal makine, diğerlerinden bağımsız olarak çalışır ve güvenlik açısından avantaj sağlar.
* Kolay Yedekleme ve Geri Yükleme: Sanal makineler kolaylıkla kopyalanabilir ve yedeklenebilir.
* Taşınabilirlik: Sanal makineler farklı fiziksel sunucular arasında taşınabilir.
* Test ve Geliştirme Ortamları: Geliştiriciler farklı işletim sistemlerini tek makine üzerinde test edebilirler.

## Geleneksel Sanallaştırma ile Container Yaklaşımı Arasındaki Temel Fark

Geleneksel sanallaştırmada her sanal makine kendi işletim sistemini çalıştırırken, container mimarisinde tüm uygulamalar ortak bir çekirdeği (kernel) paylaşır. Bu sayede container'lar daha hızlı başlatılır ve daha az kaynak tüketir.

# CONTAİNER TEKNOLOJİSİ NEDİR?

Container teknolojisi, bir uygulamanın ve onun tüm bağımlılıklarının, kütüphanelerinin ve yapılandırmalarının izole bir çalışma ortamında bir araya getirilmesini sağlayan hafif bir sanallaştırma yöntemidir. Bu teknoloji sayesinde uygulamalar, geliştirildiği ortamdan bağımsız olarak farklı sistemlerde tutarlı biçimde çalıştırılabilir.

## Container ve Geleneksel Sanallaştırma Karşılaştırması

|  |  |  |
| --- | --- | --- |
| **Özellik** | **Geleneksel Sanallaştırma** | **Container Teknolojisi** |
| İşletim Sistemi | Her sanal makine kendi işletim sistemini taşır | Tüm container’lar aynı işletim sisteminin çekirdeğini paylaşır |
| Başlatma Süresi | Dakikalar | Saniyeler |
| Kaynak Kullanımı | Göreceli olarak yüksek | Düşük |
| İzolasyon Düzeyi | Tam işletim sistemi düzeyinde | Proses düzeyinde |
| Taşınabilirlik | Sınırlı | Yüksek |

Container’lar, sanal makinelerin aksine yalnızca gerekli kütüphaneleri ve bağımlılıkları içerir; bu sayede daha küçük boyutludur ve hızlı bir şekilde çalıştırılabilir.

## Container Yapısının Bileşenleri

Bir container sistemi genel olarak aşağıdaki bileşenleri içerir:

* Image (İmaj): Container’ın çalışması için gerekli olan her şeyin (uygulama kodu, bağımlılıklar, yapılandırma dosyaları vb.) tanımlandığı kalıp dosyasıdır.
* Container: Image’tan üretilerek çalışan, izole edilmiş süreçtir. Bir image’tan birden çok container oluşturulabilir.
* Container Runtime: Container’ların çalışmasını sağlayan yazılım motorudur. Docker Engine, containerd ve CRI-O örnek olarak verilebilir.
* Dockerfile: Image oluşturmak için kullanılan komutların tanımlandığı dosyadır.

## Neden Container Kullanılır?

Taşınabilirlik: Uygulama bir kez paketlenip container içine alındığında her ortamda aynı şekilde çalışır (örneğin geliştirme, test ve üretim ortamları).

Hafiflik: Geleneksel sanallaştırmanın aksine ek işletim sistemi yükü taşınmaz.

Hız: Container’lar saniyeler içinde başlatılabilir veya durdurulabilir.

Modülerlik ve Mikroservis Uyumu: Büyük sistemler küçük container’lara bölünerek yönetilebilirlik ve ölçeklenebilirlik artırılır.

Kolaylaştırılmış CI/CD: Yazılım geliştirme ve dağıtım süreçlerinde otomasyon ve tutarlılık sağlar.

## Container Teknolojilerinin Evrimi

chroot (1979): Dosya sistemini izole etmek için kullanılan ilk yöntem.

LXC (Linux Containers): 2008’de ortaya çıkan daha gelişmiş Linux tabanlı container teknolojisidir.

Docker (2013): Container teknolojisini kullanıcı dostu ve yaygın kullanılabilir hale getirerek devrim yaratmıştır.

Podman, containerd, CRI-O: Modern container runtime alternatifleridir.

# DOCKER, CONTAİNERD, CRI-O KARŞILAŞTIRMASI

Container teknolojileri arasında en çok bilinen araç Docker’dır. Ancak özellikle Kubernetes’in gelişimiyle birlikte containerd ve CRI-O gibi daha yalın ve modüler çözümler de yaygınlaşmıştır. Bu bölümde bu üç teknolojinin ne olduğu, nasıl çalıştığı ve aralarındaki benzerlik-farklılıklar ele alınacaktır.

## Docker Nedir?

Docker, container teknolojisini yaygınlaştıran açık kaynaklı bir platformdur. Uygulamalar ve bağımlılıklarını container olarak paketlemeye, dağıtmaya ve çalıştırmaya olanak sağlar. Hem container engine (Docker Engine), hem de kullanıcı dostu bir CLI arayüzü sunar.

Docker, arka planda containerd isimli bir alt katmanı kullanarak container’ları yönetir.

Bileşenleri:

* Docker CLI (komut satırı)
* Docker Daemon (dockerd)
* Docker Engine (containerd + runc)

## containerd Nedir?

containerd, Docker’ın alt seviyeli container yönetim bileşeninden ayrılarak bağımsız hale getirilmiş, hafif ve modüler bir container runtime’dır. CNCF (Cloud Native Computing Foundation) projesidir ve Kubernetes tarafından da doğrudan desteklenir.

Özellikleri:

* Image çekme ve saklama
* Container lifecycle yönetimi (başlatma, durdurma, silme)
* Ağ ve depolama eklentileri ile entegrasyon
* Docker CLI içermez, doğrudan API üzerinden kontrol edilir

## CRI-O Nedir?

CRI-O, Kubernetes için özel olarak geliştirilmiş bir container runtime’dır ve CRI (Container Runtime Interface) standardına tam uyumludur. “Dockerless Kubernetes” anlayışının temelini oluşturur.

Amaç: Kubernetes’in ihtiyacı olan minimum işlevselliği sunmak.

Özellikleri:

* Sadece Kubernetes ile çalışmak üzere optimize edilmiştir
* Daha az bağımlılık ve daha az bellek tüketimi
* OCI (Open Container Initiative) standardına uygundur

## Karşılaştırmalı Tablo

|  |  |  |  |
| --- | --- | --- | --- |
| **Özellik / Sistem** | **Docker** | **containerd** | **CRI-O** |
| Kullanım Amacı | Geliştirme, test, dağıtım | Ara katman, Kubernetes ve Docker için temel motor | Yalnızca Kubernetes için |
| CLI (Komut Satırı) | Evet | Hayır (API tabanlı) | Hayır (Kubelet ile entegre) |
| Kubernetes Uyumluluğu | Dolaylı (Artık önerilmiyor) | Evet | Evet |
| CRI Desteği | Kısıtlı | Yüksek | Doğrudan Destekli |
| Performans | Orta | Yüksek | Yüksek |
| Bellek / CPU Kullanımı | Görece yüksek | Düşük | Çok Düşük |

## Güncel Eğilim

Kubernetes, v1.24 sürümünden itibaren Docker runtime desteğini kaldırmıştır. Bunun yerine containerd ve CRI-O gibi CRI uyumlu alternatifler kullanılmaktadır. Bu değişim, sistemlerin daha yalın ve modüler hale getirilmesini amaçlamaktadır.

# CONTAİNER KULLANIM SENARYOLARI

Container teknolojisi, esnekliği, hafifliği ve taşınabilirliği sayesinde günümüzde yazılım geliştirme, dağıtım ve işletim süreçlerinde vazgeçilmez hâle gelmiştir. Uygulama geliştiricilerden sistem yöneticilerine, DevOps ekiplerinden bulut sağlayıcılarına kadar çok geniş bir kullanıcı kitlesi tarafından farklı kullanım senaryolarında tercih edilmektedir.

## Mikroservis Mimarisi

Modern yazılımlarda yaygın olarak kullanılan mikroservis mimarisi, büyük uygulamaların küçük ve bağımsız servisler olarak tasarlanmasına dayanır. Container'lar, her bir mikroservisi izole bir ortamda çalıştırmak için idealdir.

Faydaları:

* Her servis farklı programlama diliyle geliştirilebilir.
* Servisler bağımsız olarak dağıtılabilir ve ölçeklenebilir.
* Hatalı bir servis tüm sistemi etkilemeden yeniden başlatılabilir.

## CI / CD (Sürekli Entegrasyon / Sürekli Dağıtım)

Container’lar, sürekli entegrasyon (CI) ve sürekli dağıtım (CD) süreçlerinin otomasyonu için çok uygun ortamlar sunar.

Faydaları:

* Build ortamları container içinde izole şekilde çalıştırılabilir.
* Versiyonlama ve test işlemleri güvenli şekilde yönetilir.
* Kod her ortama tutarlı biçimde dağıtılır.

## Taşınabilirlik ve Ortam Bağımsızlığı

Container'lar sayesinde bir uygulama; geliştirici bilgisayarında, test ortamında ya da bulut sunucusunda aynı şekilde çalışır. “Bende çalışıyordu” problemi ortadan kalkar.

Senaryo Örnekleri:

* Geliştirici: macOS
* Test: Ubuntu
* Üretim: Google Cloud (GKE)

Uygulama hepsinde aynı container ile çalışır.

## Bulut Bilişim ve Orkestrasyon

Container teknolojisi, özellikle Kubernetes gibi orkestrasyon sistemleriyle birlikte bulut ortamlarında yaygın olarak kullanılır. Public cloud (AWS, Azure, GCP) ve private cloud altyapılarında uygulama dağıtımı, servis keşfi, otomatik ölçekleme gibi işlemler container mimarisiyle yapılır.

Faydaları:

* Kaynaklar verimli kullanılır
* Otomatik yük dengeleme yapılabilir
* Hatalı pod’lar yeniden başlatılabilir

## Eğitim, Test ve Otomasyon Ortamları

Container’lar sayesinde farklı konfigürasyonlardaki sistemler tek bir fiziksel bilgisayarda hızlıca kurulabilir. Özellikle üniversite laboratuvarlarında, yazılım test süreçlerinde ve eğitim platformlarında tercih edilir.

Örneğin:

* PostgreSQL container’ı
* Redis test ortamı
* Python+Flask geliştirme alanı

## Edge Computing ve IoT

Kaynakların sınırlı olduğu uç cihazlarda da container’lar tercih edilir. Özellikle IoT cihazlarında, sensör verilerinin işlenmesi gibi görevler için container'lar hafif yapıları sayesinde avantaj sağlar.

## Uygulama Modernizasyonu ve Legacy Sistemler

Eski sistemlerin yeniden yazılmadan container içine alınarak modern platformlarda çalıştırılması mümkündür. Bu sayede hem maliyet düşer hem de modern altyapılardan faydalanılır.

# ASP.NET CORE WEB API GELİŞTİRME VE DOCKER CONTAİNER OLUŞTURMA

Bu bölümde, sistem bilgilerini dönen bir ASP.NET Core Web API projesi geliştirilmiştir. Proje, terminal üzerinden oluşturulmuş ve daha sonra çalıştırılarak test edilmiştir. İlk olarak aşağıdaki komutlar kullanılarak proje klasörü oluşturulmuş ve içerisine yeni bir Web API projesi eklenmiştir:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Komutun çalışmasıyla birlikte SystemInfoApi adlı bir proje klasörü oluşmuş ve gerekli bağımlılıklar yüklenmiştir. Daha sonra proje dizinine geçilerek aşağıdaki komutla uygulama çalıştırılmıştır:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Uygulama çalıştırıldığında terminalde “Now listening on: http://localhost:5154” çıktısı alınmıştır. Bu, uygulamanın yerel olarak 5154 portu üzerinden yayın yaptığını göstermektedir.

Proje içerisine, sistemsel bilgileri dönen bir API geliştirmek amacıyla Controllers klasörü altında SystemInfoController.cs adında yeni bir controller eklenmiştir. Bu controller, çalıştığı süreçteki process ID, çalıştığı makinenin adı, işletim sistemi versiyonu ve zaman gibi bilgileri JSON formatında döndürmektedir. Controller içeriği aşağıdaki gibidir:

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Controller eklendikten sonra uygulama yeniden çalıştırılmış ve web tarayıcısı üzerinden http://localhost:5154/systeminfo adresine yapılan GET isteği sonucunda aşağıdaki gibi bir JSON çıktısı elde edilmiştir:

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

Bu çıktı, API’nin başarılı şekilde çalıştığını göstermektedir. Ayrıca, uygulamanın Program.cs dosyasında builder.Services.AddControllers(); ve app.MapControllers(); satırlarının yer almasıyla, eklenen controller’ın devreye alınması sağlanmıştır.

Uygulama çalıştırılırken geliştirici modunda HTTPS yönlendirmesine dair bir uyarı alınmış fakat bu uyarı çalışmayı etkilememiştir. Sistem HTTP üzerinden istekleri başarılı şekilde yanıtlamıştır.

Bu adımlarla birlikte geliştirilen API, container ortamına alınmaya hazır hâle gelmiştir. Bir sonraki aşamada Docker kullanılarak bu projenin imajı oluşturulacak ve container üzerinde çalıştırılması sağlanacaktır.

# KUBERNETES NEDİR? DOCKER ve CONTAINERD ile FARKI

Kubernetes (k8s), container tabanlı uygulamaların dağıtımını, ölçeklenmesini ve yönetimini otomatikleştiren açık kaynaklı bir konteyner orkestrasyon platformudur. 2014 yılında Google tarafından geliştirilmiş ve daha sonra Cloud Native Computing Foundation (CNCF) bünyesinde açık kaynak olarak yayımlanmıştır.

Kubernetes, birçok container’ı bir arada ve organize şekilde çalıştırmak için geliştirilmiştir. Tek bir makinede çalışan container yapıları, üretim ortamında yetersiz kaldığında Kubernetes devreye girerek bu yapıları kümeler (cluster) halinde yönetir.

## Kubernetes’in Temel Özellikleri

Pod yönetimi: Kubernetes’te container’lar doğrudan değil, “pod” adı verilen yapılarda çalıştırılır. Bir pod içinde bir veya daha fazla container yer alabilir.

Otomatik yeniden başlatma: Hatalı çalışan container’lar otomatik olarak yeniden başlatılır.

Yük dengeleme: Gelen istekler pod’lar arasında dağıtılır (load balancing).

Otomatik ölçeklendirme: Trafik arttıkça yeni pod’lar oluşturulabilir.

Servis keşfi (Service Discovery): Pod’lar arası iletişim DNS üzerinden kolayca sağlanabilir.

Dağıtık mimari desteği: Uygulamalar birçok fiziksel ya da sanal sunucuya dağıtılarak çalıştırılabilir.

## Docker ve Kubernetes İlişkisi

Docker, container'ları oluşturmaya ve çalıştırmaya yarayan bir container motorudur (runtime). Kubernetes ise bu container’ları yönetmeye ve orkestrasyonuna odaklanır.

Başlangıçta Kubernetes, container’ları çalıştırmak için Docker’ı kullanıyordu. Ancak Kubernetes 1.24 sürümünden itibaren Docker desteği resmi olarak kaldırılmış ve onun yerine CRI (Container Runtime Interface) uyumlu motorlar kullanılmaya başlanmıştır.

## containerd ile Kubernetes İlişkisi

containerd, Docker’ın iç bileşenlerinden ayrıştırılarak geliştirilen bağımsız, hafif bir container runtime’dır. Kubernetes ile CRI üzerinden uyumlu çalışır. Günümüzde Kubernetes cluster’ları varsayılan olarak containerd veya CRI-O kullanır.

## Karşılaştırmalı Tablo

|  |  |  |  |
| --- | --- | --- | --- |
| **Özellik** | **Docker** | **containerd** | **Kubernetes** |
| Tanım | Container oluşturucu ve çalıştırıcı | Hafif container runtime | Container orkestrasyon sistemi |
| Görev | Container imajı oluşturmak ve çalıştırmak | Container’ları arka planda yönetmek | Container’ları dağıtmak, ölçeklemek |
| CLI (Komut Arayüzü) | Var | Yok (API tabanlı) | kubectl ile yönetilir |
| CRI Uyumluluğu | Kısmen | Evet | CRI uyumlu motorlarla çalışır |
| Kullanım Amacı | Tekil container geliştirme | Docker yerine sistem seviyesi kullanım | Çoklu container yönetimi ve orkestrasyonu |

## Sonuç

Özetle Docker ve containerd, bir veya birkaç container’ı başlatmak için kullanılırken, Kubernetes çok sayıda container’ı aynı anda kontrol etmek ve sistem ölçeğinde yönetmek için kullanılır. Kubernetes, container motorlarından bağımsız olarak çalışır; yalnızca CRI uyumlu bir runtime’a ihtiyaç duyar. Modern uygulama dağıtımlarında, container'lar çoğunlukla Kubernetes ile yönetilirken, arka planda containerd veya CRI-O gibi daha hafif ve Kubernetes'e özel motorlar tercih edilmektedir.

# KUBERNETES CLUSTER ÜZERİNDE API CONTAINER’ININ ÇALIŞTIRILMASI

Bu bölümde, daha önce geliştirilen SystemInfoApi projesi bir Docker imajı haline getirilmiş ve ardından Kubernetes ortamında en az üç instance (pod) olarak dağıtılmıştır. Her pod’a yapılan isteklerde, o pod’a özel sistem bilgileri alınması sağlanmıştır.

## Cluster Oluşturulması

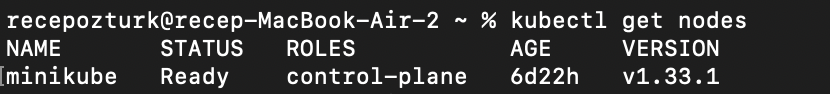
Kubernetes ortamı Minikube ile başlatılmıştır. Gerekli servislerin çalıştığı minikube status komutu ile, küme içerisindeki node'un durumunu gösteren kubectl get nodes komutu ile doğrulama yapılmıştır. Küme içerisinde bir adet “minikube” node'u “Ready” olarak görünmektedir.

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

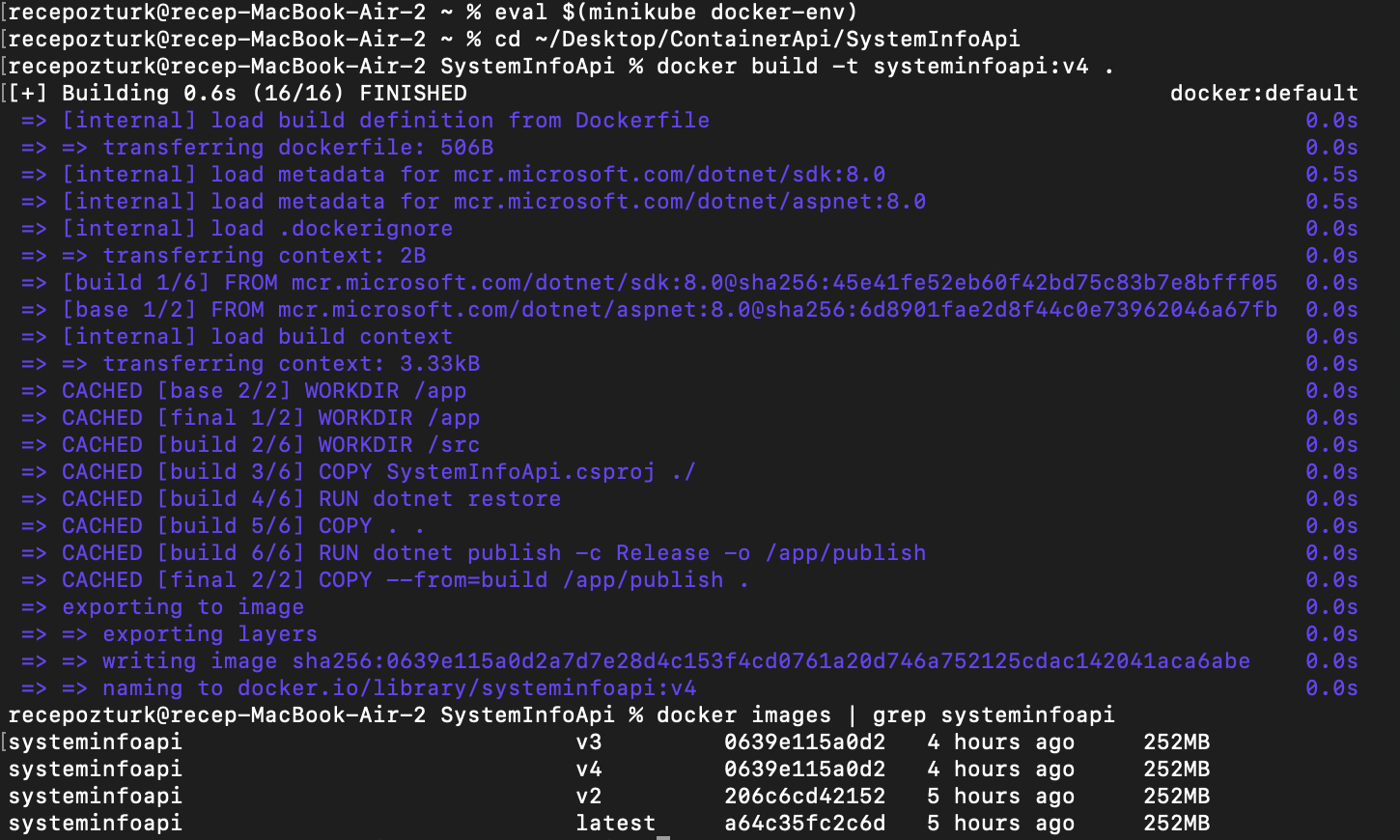
metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.



## Docker Image’ın Oluşturulması

Minikube ortamında çalışacak Docker imajı oluşturulmuştur. Bunun için terminal ortamı eval $(minikube docker-env) komutu ile Minikube’a yönlendirilmiş, ardından Dockerfile kullanılarak docker build komutu ile systeminfoapi adında bir image build edilmiştir.



## Kubernetes Deployment ve Servis Tanımı

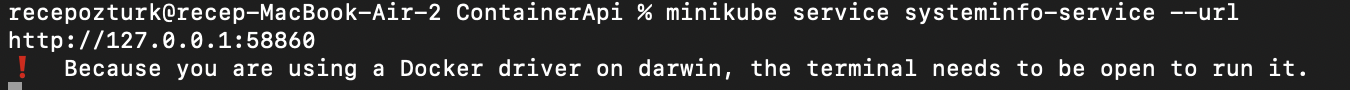
Proje için Kubernetes ortamında bir deployment ve bir servis tanımı yapılmıştır. Deployment dosyasında 3 pod çalışacak şekilde replicas: 3 olarak ayarlanmıştır. Service tanımı ise dış erişime izin verecek şekilde NodePort olarak yapılandırılmıştır. kubectl get pods komutu ile tüm pod’ların çalıştığı görülmektedir.

metin, ekran görüntüsü, yazı tipi, siyah içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

## API’ye Erişim ve Pod Yanıtları

Minikube üzerinde çalışan systeminfo-service NodePort servisine açılan tünel üzerinden (http://127.0.0.1:58860/systeminfo) erişim sağlanmıştır. Yapılan HTTP GET isteklerinde dönen JSON yanıtlar incelendiğinde machineName alanının farklı pod adlarıyla geldiği görülmüştür. Bu durum, Kubernetes servis katmanı aracılığıyla isteklerin çoklu pod’lara dağıtıldığını ve yük dağıtımının başarıyla gerçekleştirildiğini göstermektedir.

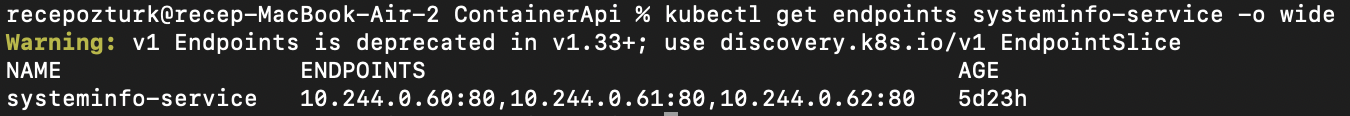


metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

kubectl get endpoints systeminfo-service -o wide çıktısında servisin üç farklı pod IP:port’una yönlendirme yaptığı görülmüştür. Bu, yük dağıtımının yalnızca pod yanıtlarında değil, Kubernetes düzeyinde de doğrulandığını göstermektedir.



# SONUÇ

Bu çalışma kapsamında, sanallaştırma ve container teknolojilerinin temel prensipleri, bileşenleri ve avantajları ayrıntılı olarak incelenmiş; Docker, containerd ve CRI-O gibi farklı container runtime çözümleri teknik açıdan karşılaştırılmıştır. Ayrıca container teknolojisinin modern yazılım geliştirme süreçlerindeki önemi, kullanım senaryoları ve orkestrasyon sistemleriyle olan ilişkisi ortaya konmuştur. Uygulama aşamasında ise ASP.NET Core tabanlı bir Web API projesi geliştirilmiş, Dockerfile aracılığıyla imajı oluşturulmuş ve Minikube üzerinde çalışan Kubernetes cluster’ına üç pod olarak dağıtılmıştır. Kubernetes’in yük dengeleme ve servis yönlendirme özellikleri test edilmiş, farklı pod’lardan alınan yanıtlar ve kubectl get endpoints çıktıları ile sistemin ölçeklenebilirlik ve yüksek erişilebilirlik sağladığı kanıtlanmıştır. Elde edilen sonuçlar, container tabanlı uygulamaların Kubernetes ile kullanıldığında esnek, taşınabilir, yönetilebilir ve güvenilir bir altyapı sunduğunu göstermektedir. Bu sayede modern yazılım geliştirme, dağıtım ve işletim süreçlerinde hem verimlilik hem de sürdürülebilirlik açısından önemli kazanımlar elde edilmektedir.

# KAYNAKÇA

* Cloud Native Computing Foundation. (2024). containerd. <https://containerd.io/>
* Docker Inc. (2025). What is Docker?. <https://www.docker.com/resources/what-container>
* Google Cloud. (2024). Kubernetes overview. <https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-overview>
* Kubernetes Documentation. (2025). Service. <https://kubernetes.io/docs/concepts/services-networking/service/>
* Kubernetes Documentation. (2025). Deployments. <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
* Microsoft. (2025). ASP.NET Core Web API documentation. <https://learn.microsoft.com/aspnet/core/web-api>
* Microsoft. (2025). Introduction to containers. <https://learn.microsoft.com/virtualization/windowscontainers/about/>
* Red Hat. (2025). What is CRI-O?. <https://www.redhat.com/en/topics/containers/what-is-cri-o>
* VMware. (2024). What is virtualization?. <https://www.vmware.com/topics/glossary/content/virtualization>