Teknoloji Fakültesi

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Kubernetes Cluster Yönetim Aracı

BİTİRME PROJESİ

2. ARA RAPORU

Bilgisayar Mühendisliği Bölümü

DANIŞMAN

Prof. Dr. Ali Buldu

İSTANBUL, 2025

MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Marmara Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Öğrencileri Tarık Uçar, Ömer Faruk Yiğit ve Ege Alican Kaleli tarafından "Kubernetes Cluster Yönetim Aracı" başlıklı proje çalışması, xxx tarihinde savunulmuş ve jüri üyeleri tarafından başarılı bulunmuştur.

Jüri Üyeleri

Dr. Öğr. Üyesi xxx xxx Marmara Üniversitesi	**(Danışman)**	(İMZA)……..
Prof. Dr. Xxx xxx Marmara Üniversitesi	(Üye)	(İMZA)……..
Prof. Dr. Xxx xxx Marmara Üniversitesi	(Üye)	(İMZA)……..

ÖNSÖZ

Proje çalışmamız süresince karşılaştığım bütün problemlerde, sabırla yardım ve bilgilerini esirgemeyen, tüm desteğini sonuna kadar yanımda hissettiğim değerli hocalarım, sayın Dr. Öğr. Üyesi Xxx xxx ve sayın Prof. Dr. Xxx xxx' a en içten teşekkürlerimi sunarım.

Bu proje çalışması fikrinin oluşması ve ortaya çıkmasındaki önerisi ve desteğinden dolayı değerli hocam Dr. Öğr. Üyesi Xxx xxx' a teşekkür ederim.

Proje çalışmam sırasında maddi ve manevi desteklerini esirgemeyen okul içerisinde ve okul dışında her zaman yanımda olan değerli çalışma arkadaşlarım ve hocalarım Doç. Dr. Xxx xxx ve Dr. Öğr. Üyesi ' xxx xxx a sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER

ÖZET

Kubernetes Cluster Yönetim Aracı

Bu çalışmada, Kubernetes tabanlı sistemlerin yönetimini kolaylaştırmak amacıyla geliştirilen bir Kubernetes Cluster Yönetim Aracı sunulmaktadır. Kubernetes, dağıtık sistemlerin otomatik ölçeklendirilmesi, yönetilmesi ve dağıtılması için kullanılan popüler bir konteyner orkestrasyon platformudur. Ancak, Kubernetes'in karmaşık yapısı ve yönetim süreçlerinin detaylı bilgi gerektirmesi, özellikle yeni kullanıcılar için zorluklar yaratmaktadır.

Geliştirilen yönetim aracı, Kubernetes cluster'larının yönetimini daha erişilebilir hale getirmek için kullanıcı dostu bir arayüz ve otomasyon mekanizmaları sunmaktadır. Bu kapsamda, temel Kubernetes işlemleri (pod yönetimi, servis yönetimi, ölçeklendirme, izleme ve log analizi) tek bir platform üzerinden yönetilebilmektedir. Aracın sağladığı görsel arayüz sayesinde, Kubernetes'in CLI tabanlı kullanımına alternatif bir yaklaşım sunulmaktadır. Ek olarak, sistem kaynaklarının verimli kullanımı ve otomatik hata tespiti gibi özellikler ile yöneticilere operasyonel kolaylık sağlanmaktadır.

Sonuç olarak, bu çalışma Kubernetes ortamlarının daha etkin yönetilmesine olanak tanıyan, kullanım kolaylığı sağlayan ve operasyonel süreçleri optimize eden bir yönetim aracı geliştirmeyi hedeflemektedir. Bu araç, hem küçük ölçekli hem de büyük ölçekli Kubernetes altyapıları için uygun olacak şekilde tasarlanmıştır.

Mart, 2025

Öğrenciler

Tarık UÇAR

Ege Alican KALELİ

Ömer Faruk YİĞİT

ABSTRACT

Kubernetes Cluster Management Tool

This study presents a Kubernetes Cluster Management Tool designed to simplify the administration of Kubernetes-based systems. Kubernetes is a widely used container orchestration platform that enables the automated scaling, management, and deployment of distributed systems. However, the complexity of Kubernetes and the detailed knowledge required for its management pose challenges, especially for new users.

The developed management tool provides a user-friendly interface and automation mechanisms to enhance the accessibility of Kubernetes cluster management. Within this scope, essential Kubernetes operations (such as pod management, service management, scaling, monitoring, and log analysis) can be handled through a single platform. The graphical interface offered by the tool serves as an alternative to the CLI-based Kubernetes usage. Additionally, features such as efficient resource utilization and automatic error detection provide operational convenience to administrators.

In conclusion, this study aims to develop a management tool that facilitates the effective administration of Kubernetes environments, enhances usability, and optimizes operational processes. This tool is designed to be suitable for both small-scale and large-scale Kubernetes infrastructures.

March, 2025

Students

Tarık UÇAR

Ege Alican KALELİ

Ömer Faruk YİĞİT

1. GİRİŞ

Kubernetes, modern bulut bilişim dünyasında konteyner tabanlı uygulamaların yönetimini kolaylaştıran en popüler orkestrasyon sistemlerinden biri haline gelmiştir [1]. Google tarafından geliştirilip açık kaynak olarak sunulan Kubernetes, mikro hizmet mimarilerini destekleyen, ölçeklenebilir ve dayanıklı bir yapı sağlamaktadır [2]. Ancak Kubernetes'in esnekliği ve geniş özellik seti, yönetim sürecini karmaşık hale getirmektedir. Özellikle büyük ölçekli sistemlerde Kubernetes kümelerinin yönetimi, sürekli izleme, kaynak optimizasyonu, otomatik ölçekleme, güvenlik ve hata toleransı gibi konuları içeren kapsamlı bir süreç gerektirmektedir [3].

Günümüzde Kubernetes yöneticileri, kümeleri etkin bir şekilde yönetmek için çeşitli araçlar kullanmaktadır. Prometheus, Grafana, Helm ve ArgoCD gibi araçlar, Kubernetes ortamlarının izlenmesini, yapılandırılmasını ve sürekli entegrasyon/sürekli dağıtım (CI/CD) süreçlerinin yönetilmesini sağlamaktadır [4]. Bununla birlikte, farklı araçların entegre edilmesi ve yönetilmesi, deneyimli kullanıcılar için bile zaman alıcı ve karmaşık bir süreç olabilmektedir. Bu nedenle, Kubernetes kümelerinin daha verimli ve merkezi bir şekilde yönetilmesine olanak tanıyacak yeni nesil yönetim araçlarına olan ihtiyaç giderek artmaktadır [5].

Bu çalışmada, Kubernetes küme yönetimini kolaylaştırmak amacıyla geliştirilmiş mevcut araçlar incelenerek, bu araçların eksik yönleri ve kullanım zorlukları analiz edilecektir. Ayrıca, Kubernetes kümelerinin yönetimini daha etkin hale getirmek için geliştirilebilecek yeni bir yönetim aracının tasarım ilkeleri ele alınacaktır. Çalışmanın temel amacı, Kubernetes yönetimini daha erişilebilir, verimli ve otomatik hale getirecek bir çözüm sunmaktır.

2. LİTERATÜR TARAMASI

2.1 Kubernetes Yönetim Araçlarının Genel İncelemesi

Kubernetes yönetimini kolaylaştırmak için geliştirilen birçok açık kaynak ve ticari çözüm bulunmaktadır. Rancher, Lens ve OpenShift gibi platformlar, Kubernetes kümelerini görselleştirme, yapılandırma ve izleme konularında önemli kolaylıklar sağlamaktadır [6]. Rancher, özellikle çoklu küme yönetimi konusunda öne çıkarken, Lens ise Kubernetes kaynaklarını grafiksel bir arayüz üzerinden yönetmeyi mümkün kılmaktadır [7]. Red Hat tarafından geliştirilen OpenShift ise Kubernetes'in kurumsal kullanımını destekleyen ek güvenlik ve otomasyon özellikleri sunmaktadır [8].

2.2 Kubernetes İzleme ve Gözlemlenebilirlik Çözümleri

Kubernetes ortamlarında sistem sağlığını ve performansını izlemek için çeşitli gözlemlenebilirlik araçları kullanılmaktadır. Prometheus, Kubernetes kümelerinden metrik toplamak için en yaygın kullanılan zaman serisi veritabanıdır [9]. Prometheus ile birlikte Grafana kullanılarak metrikler görselleştirilmekte ve sistem performansı analiz edilebilmektedir [10]. Ayrıca, OpenTelemetry ve Jaeger gibi dağıtık izleme araçları, Kubernetes kümelerinde hata ayıklama ve performans analizi yapmayı kolaylaştırmaktadır [11].

2.3 Kubernetes Otomatikleştirme ve CI/CD Araçları

Kubernetes kümelerinde uygulama dağıtımlarını ve yönetim süreçlerini otomatikleştirmek için çeşitli araçlar bulunmaktadır. ArgoCD ve FluxCD, Kubernetes ortamları için GitOps yaklaşımını destekleyen popüler sürekli dağıtım (CD) araçlarıdır [12]. Jenkins ve Tekton gibi CI/CD araçları, Kubernetes üzerine sürekli entegrasyon süreçlerini entegre etmek için yaygın olarak kullanılmaktadır [13]. Helm ise Kubernetes uygulamalarının paketlenmesini ve yönetilmesini kolaylaştıran bir araç olarak öne çıkmaktadır [14].

2.4 Kubernetes Güvenlik Yönetimi Çözümleri

Kubernetes'in esnekliği ve ölçeklenebilirliği, güvenlik açısından bazı riskleri de beraberinde getirmektedir. Pod güvenliği, RBAC (Role-Based Access Control) ile erişim denetimi ve ağ politikalarının yapılandırılması gibi konular, Kubernetes yönetiminde kritik rol oynamaktadır [15]. Istio ve Linkerd gibi servis mesh çözümleri, güvenli servis iletişimi ve trafiğin yönetimi konusunda Kubernetes yöneticilerine ek yetenekler sağlamaktadır [16]. Ayrıca, Falco ve Kyverno gibi güvenlik araçları, Kubernetes ortamlarında tehdit algılama ve politika yönetimi için kullanılmaktadır [17].

2.5 Kubernetes Yönetim Araçlarına Duyulan İhtiyaç

Yapılan araştırmalar, mevcut Kubernetes yönetim araçlarının belirli konulara odaklandığını ancak kapsamlı bir yönetim deneyimi sunmada yetersiz kaldığını göstermektedir. Örneğin, Rancher ve Lens görselleştirme açısından güçlü araçlar olmasına rağmen, CI/CD veya güvenlik yönetimi gibi konularda sınırlı yeteneklere sahiptir [18]. Benzer şekilde, ArgoCD ve FluxCD sürekli dağıtım süreçlerini otomatize ederken, genel küme yönetimi için yeterli desteği sunmamaktadır [19]. Bu nedenle, Kubernetes yönetimini tek bir çatı altında toplayarak daha verimli ve kullanıcı dostu hale getirecek entegre bir yönetim aracına duyulan ihtiyaç giderek artmaktadır.

Bu çalışmada, mevcut Kubernetes yönetim araçlarının güçlü ve zayıf yönleri değerlendirilecek ve eksiklikleri giderecek bir yönetim aracı için temel tasarım ilkeleri ele alınacaktır. Bu sayede, Kubernetes yöneticilerinin karşılaştığı zorlukları azaltan, yönetimi kolaylaştıran ve operasyonel verimliliği artıran bir çözüm geliştirilmesi hedeflenmektedir.

3. BULGULAR VE TARTIŞMA

3.1 Kubernetes Ortam Kurulumu

Bu çalışmada, Kubernetes tabanlı bir altyapı oluşturulmuş ve sistemin temel bileşenleri yapılandırılmıştır. Kubernetes kümesinin kurulumu sırasında, yönetim kolaylığı ve ölçeklenebilirlik ön planda tutulmuştur. Kurulum aşamasında, küme bileşenlerinin stabil çalışmasını sağlamak için gerekli yapılandırmalar yapılmış, temel testler gerçekleştirilmiştir. Kubernetes ortamının başarılı bir şekilde çalıştığı doğrulanmış ve projeye uygun şekilde optimize edilmiştir.

3.2 GitHub Organizasyonu ve Repo Yönetimi

Proje yönetimi sürecinde, kod ve yapılandırma dosyalarının merkezi bir yerde toplanması için GitHub üzerinde bir organizasyon oluşturulmuştur. Organizasyon altında, farklı bileşenler için ayrı ayrı depolar açılmış ve sürüm kontrol mekanizması etkin bir şekilde yapılandırılmıştır. Geliştirme sürecinde, ekip üyelerinin katkılarını takip etmek ve versiyon yönetimini sağlamak amacıyla Git akış süreçleri belirlenmiştir.

3.3 KIND (Kubernetes in Docker) Araştırması

Proje kapsamında Kubernetes ortamlarının yerel makinelerde daha hızlı ve verimli bir şekilde oluşturulabilmesi için Kind (Kubernetes in Docker) incelenmiştir. Kind, konteyner tabanlı bir Kubernetes kümesi oluşturma imkanı sunduğu için geliştirme ve test süreçlerinde avantaj sağlamaktadır. Araştırma sonucunda, Kind'in test ortamlarında kullanılabilirliği değerlendirilmiş ve avantajları ile sınırlamaları belirlenmiştir.

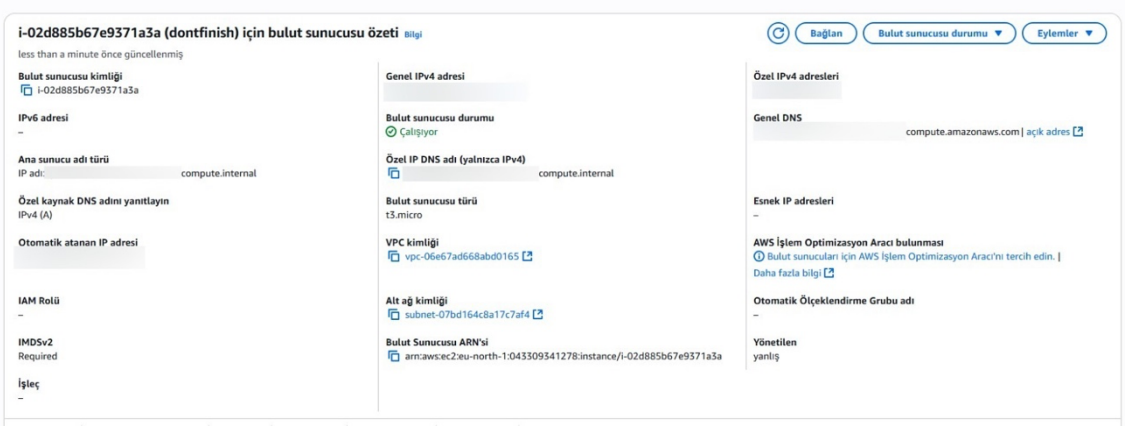
3.4 Teknoloji Seçimi

Proje kapsamında kullanılacak teknolojiler belirlenmiş olup, geliştirme sürecinde backend tarafında JavaScript kullanılması kararlaştırılmıştır. Aynı zamanda, frontend için de JavaScript tabanlı bir framework seçilmiştir. Bu seçimler, geliştirici deneyimi, topluluk desteği ve sistem uyumluluğu göz önünde bulundurularak yapılmıştır.

Kullanılan yazılım dillerinin entegrasyonu ve modüler yapıya uygunluğu değerlendirilerek proje mimarisine en uygun yapı oluşturulmuştur.

3.5 Ortam Kurulumu

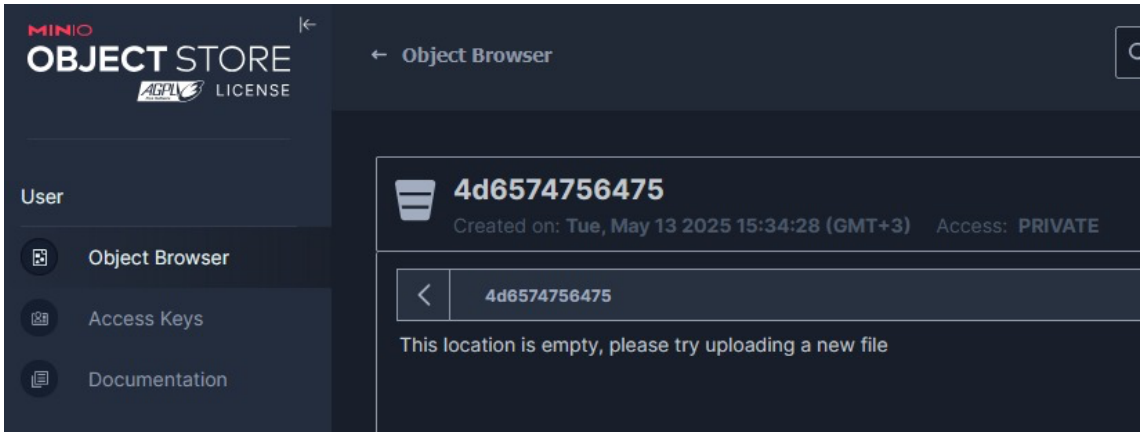
Gerçekleştirilen sanal makine kurulumu sürecinde Amazon Web Services (AWS) platformu kullanılarak "ana makine" olarak adlandırılan bir EC2 (Elastic Compute Cloud) örneği başarıyla yapılandırılmıştır. Kurulum esnasında Ubuntu 22.04 LTS işletim sistemi tercih edilmiş ve bu işletim sistemi üzerinde temel sistem güncellemeleri tamamlandıktan sonra, uygulama geliştirme ve sunucu yönetimi için gerekli olan yazılımlar yüklenmiştir. Bu kapsamda, öncelikle OpenSSH hizmeti etkinleştirilerek uzaktan bağlantı sağlanmış, ardından Docker, PostgreSQL, Minio gibi klasik bileşenler kurulmuştur. Ek olarak, versiyon kontrolü için Git, paket yönetimi için entegre edilmiştir. Kurulum sürecinde karşılaşılan temel zorluklardan biri, güvenlik duvarı ve AWS güvenlik grubu yapılandırmalarının senkronize edilmesi gerekliliği olmuştur. Ancak bu yapılandırmalar başarıyla sağlandıktan sonra makinenin hem iç erişimi hem de dışa açık servisleri sorunsuz şekilde çalıştırılabilmiştir. Sonuç olarak, bu ana makine yazılım geliştirme, test ve dağıtım süreçleri için uygun, güvenli ve ölçeklenebilir bir altyapı sunmaktadır.



3.6 Minio Kurulumu

Amazon Web Services (AWS) ortamında yapılandırılan "ana makine" adlı EC2 örneği, Ubuntu 22.04 LTS işletim sistemi üzerinde başarıyla kurulmuş ve temel sistem güncellemeleri tamamlanmıştır. Geliştirme ve servis sunumu için gerekli olan temel bileşenler (Apache, MySQL, PHP, Git, Node.js, npm vb.) kurularak makinenin işlevselliği genişletilmiştir. Bu sürecin önemli bir parçası olarak, bulut tabanlı nesne depolama hizmeti sağlayan **MinIO** yazılımı da sisteme entegre edilmiştir. MinIO

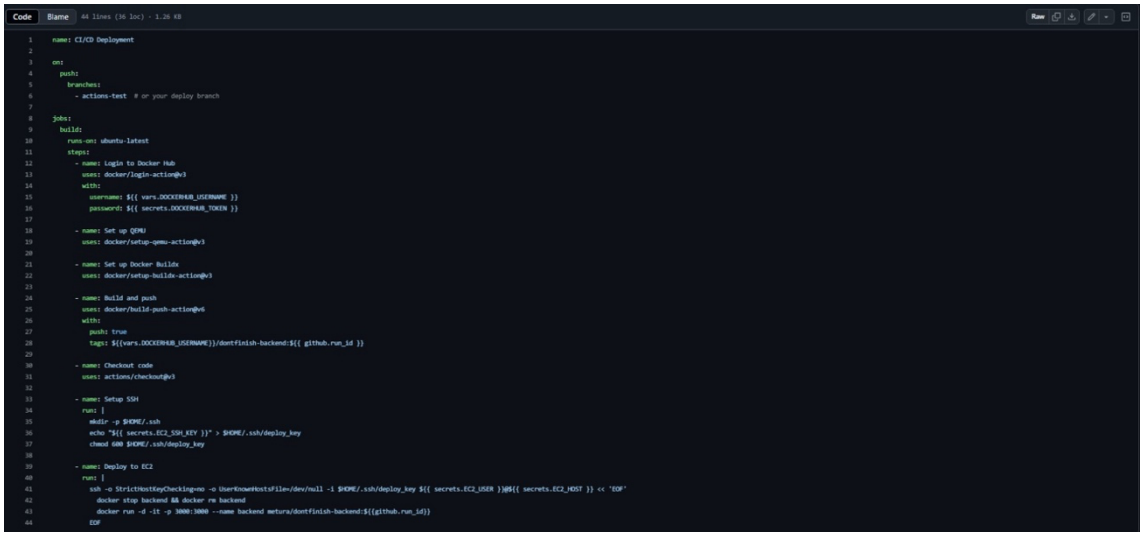
kurulumu sırasında öncelikle ilgili ikili dosya indirilmiş, çalıştırılabilir hale getirilmiş ve servis olarak sistem başlangıcına eklenmiştir. Servisin çalışabilmesi için gerekli dizin yapıları oluşturulmuş, erişim anahtarları tanımlanmış ve HTTP üzerinden çalışan bir port aracılığıyla dış erişime açılmıştır. AWS güvenlik grubu ayarlarında ilgili portlara izin verilerek MinIO arayüzüne web tarayıcısı üzerinden erişim sağlanabilmiştir. MinIO'nun kurulumu sayesinde, uygulama geliştirme sürecinde kullanılacak olan dosya ve medya içeriklerinin yüksek erişilebilirlik ve performansla yönetilmesi mümkün hale gelmiştir. Bu durum, ana makinenin yalnızca bir uygulama sunucusu değil, aynı zamanda esnek bir veri depolama çözümü olarak da kullanılmasını sağlamaktadır.



3.7 CI/CD Süreçlerinin Tamamlanması

Uygulamanın kesintisiz ve otomatik bir şekilde dağıtılabilmesi amacıyla GitHub Actions kullanılarak bir CI/CD (Sürekli Entegrasyon/Sürekli Dağıtım) süreci yapılandırılmıştır. Görselde yer alan yaml yapılandırma dosyası, kaynak kodun belirli bir dalına (actions-test) yapılan her "push" işlemiyle tetiklenen bir iş akışını tanımlamaktadır. Bu akışta ilk olarak Docker Hub'a giriş yapılmakta, ardından farklı mimarileri destekleyebilmek amacıyla QEMU ve Docker Buildx kurulumu gerçekleştirilmektedir. Takip eden adımda, uygulamanın Docker imajı oluşturulmakta ve Docker Hub üzerine dontfinish-backend:<github.run_id> etiketiyle push edilmektedir.

Kodun güncel hali EC2 üzerindeki ana makineye deploy edilebilmesi için SSH anahtarları güvenli bir şekilde yapılandırılmakta ve ssh bağlantısı aracılığıyla uzak sunucuya erişilmektedir. EC2 tarafında önce mevcut backend konteyneri durdurulmakta, ardından yeni versiyon güncellenmiş imajla başlatılmaktadır. Docker konteyneri, 3000 numaralı port üzerinden dış erişime açılacak şekilde yapılandırılmıştır. Bu yapı sayesinde, kod deposuna yapılan her güncellemenin otomatik olarak test edilip dağıtılması sağlanmakta ve manuel müdahaleye gerek kalmadan güvenilir, hızlı ve tekrarlanabilir bir dağıtım süreci elde edilmektedir.



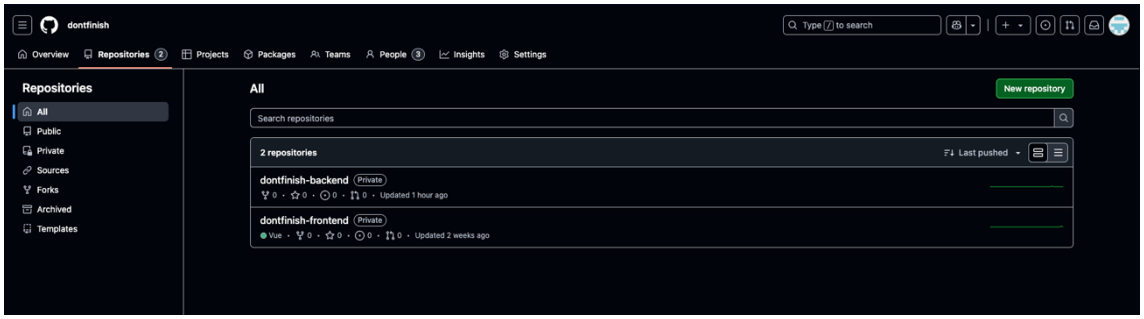
```yaml
name: CI/CD Deployment

on:
  push:
    branches:
      - actions-test  # or your deploy branch

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ vars.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build and push
        uses: docker/build-push-action@v6
        with:
          push: true
          tags: ${{vars.DOCKERHUB_USERNAME}}/dontfinish-backend:${{ github.run_id }}

      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup SSH
        run: |
          mkdir -p $HOME/.ssh
          echo "${{ secrets.EC2_SSH_KEY }}" > $HOME/.ssh/deploy_key
          chmod 600 $HOME/.ssh/deploy_key

      - name: Deploy to EC2
        run: |
          ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null -i $HOME/.ssh/deploy_key ${{ secrets.EC2_USER }}@${{ secrets.EC2_HOST }} << 'EOF'
            docker stop backend && docker rm backend
            docker run -d -it -p 3000:3000 --name backend metura/dontfinish-backend:${{github.run_id}}
          EOF
```

3.8 Yazılım Geliştirme

Geliştirme süreci, öncelikle istemci ve sunucu tarafının ayrı ayrı ele alındığı iki farklı proje yapısıyla ilerletilmiştir. GitHub üzerinde *dontfinish* adlı organizasyon altında barındırılan iki özel (private) depo — dontfinish-backend ve dontfinish-frontend — bu ayrımı yansıtmaktadır. Backend tarafı; API servislerini, veritabanı işlemlerini ve MinIO entegrasyonunu yöneten bileşenleri içermekte olup, frontend tarafı ise Vue.js ile geliştirilmiş kullanıcı arayüzü bileşenlerini kapsamaktadır.

Backend deposu, CI/CD süreçlerinin otomatikleştirilmesi amacıyla yapılandırılmış GitHub Actions betiklerini barındırmaktadır. Kod her güncellendiğinde bu betikler devreye girmekte; Docker imajı otomatik olarak oluşturulmakta, Docker Hub'a yüklenmekte ve ardından AWS EC2 üzerindeki ana makineye otomatik olarak dağıtılmaktadır. Bu mimari sayesinde kod değişiklikleri en kısa sürede canlı ortama aktarılmakta, böylece hataların erken fark edilmesi ve çözülmesi sağlanmaktadır.

Frontend deposu ise benzer şekilde bağımsız bir geliştirme ve dağıtım sürecine sahiptir. Her iki yapının da sürüm kontrolü, erişim yönetimi ve takım içi iş birliği GitHub üzerinden sağlanmaktadır. Bu yapı, projenin sürdürülebilirliği, izlenebilirliği ve dağıtım verimliliği açısından büyük avantaj sunmaktadır.



3.9 Gelecek Çalışmalar

İlerleyen süreçte, Kubernetes ortamına yönelik ölçeklendirme ve güvenlik önlemlerinin detaylandırılması planlanmaktadır. Ayrıca, GitHub organizasyon yönetiminin daha verimli hale getirilmesi hedeflenmektedir. Frontend ve backend geliştirme süreçlerinde, belirlenen teknolojilerin etkin kullanımı sağlanarak projenin geliştirilmesi sürdürülecektir.

KAYNAKLAR

1. **Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J.** (2016). *Borg, Omega, and Kubernetes*. ACM Queue, 14(1), 70–93.
2. **Hightower, K., Burns, B., & Beda, J.** (2017). *Kubernetes: Up and Running: Dive into the Future of Infrastructure*. O'Reilly Media.
3. **Vaquero, L. M., & Cuevas, R.** (2019). *Kubernetes and the challenge of distributed system observability*. IEEE Internet Computing, 23(3), 15–22.
4. **Fischer, S., & Kapoor, A.** (2020). *Cloud-native monitoring with Prometheus and Grafana*. Packt Publishing.
5. **Turnbull, J.** (2018). *The Kubernetes Book*. Independently published.
6. **Rancher Labs** (2022). *Rancher: Multi-Cluster Kubernetes Management*.
7. **Mirantis** (2023). *Lens - The Kubernetes IDE*.
8. **Red Hat** (2023). *OpenShift Documentation*.
9. **Prometheus Authors** (2023). *Prometheus Documentation*.
10. **Grafana Labs** (2023). *Grafana Documentation*.
11. **OpenTelemetry Authors** (2023). *OpenTelemetry Documentation*.
12. **ArgoCD Authors** (2023). *ArgoCD - Declarative GitOps Continuous Delivery*.
13. **Jenkins Project** (2023). *Jenkins User Documentation*.
14. **Helm Authors** (2023). *Helm - The Kubernetes Package Manager*.
15. **The Kubernetes Authors** (2023). *RBAC Authorization in Kubernetes*.
16. **Buoyant** (2023). *Linkerd: The Service Mesh for Kubernetes*.
17. **Sysdig** (2023). *Falco: Cloud-native Runtime Security*.
18. **Chamberlain, R.** (2021). *Kubernetes Operators and Multi-cluster Management*. Manning Publications.
19. **Weaveworks** (2023). *FluxCD - GitOps for Kubernetes*.