MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BULUT TABANLI DEVOPS UYGULAMALARI İLE YAZILIM GELİŞTİRME ve DAĞITIM SÜREÇLERİNİN İYİLEŞTİRİLMESİ

ONUR YURTSEVER

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği
Anabilim Dalı
Bilgisayar Mühendisliği Programı

DANIŞMAN
Dr. Öğr. Üyesi Eyüp Emre Ülkü

EŞ DANIŞMAN
Doç. Dr. Kazım Yıldız

İSTANBUL, 2025

MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BULUT TABANLI DEVOPS UYGULAMALARI İLE YAZILIM GELİŞTİRME ve DAĞITIM SÜREÇLERİNİN İYİLEŞTİRİLMESİ

ONUR YURTSEVER

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği
Anabilim Dalı
Bilgisayar Mühendisliği Programı

DANIŞMAN

Dr. Öğr. Üyesi Eyüp Emre Ülkü

EŞ DANIŞMAN

Doç. Dr. Kazım Yıldız

İSTANBUL, 2025

TEŞEKKÜR

Bu tez çalışmasındaki her aşamada bana yardımcı olan değerli tez danışmanım Dr. Öğr. Üyesi Eyüp Emre ÜLKÜ'ye, yüksek lisans ders aşamasında ve tez sürecinde her zaman desteklerini gördüğüm eş danışmanım Doç.Dr. Kazım Yıldız'a, tez için gerekli bulut bilişim altyapısını sağlayan Commencis Teknoloji A.Ş'ye teşekkür ederim. Tezimi aileme ithaf ederim.

Aralık,2025 **Onur Yurtsever**

İÇİNDEKİLER

TEŞEKKÜR ..... **i**
İÇİNDEKİLER ..... **ii**
ÖZET ..... **iii**
ABSTRACT ..... **iv**
SEMBOLLER ..... **v**
KISALTMALAR ..... **vi**
ŞEKİL LİSTESİ ..... **viii**
TABLO LİSTESİ ..... **x**
1. GİRİŞ **1**
1.1. Literatür Taraması 4
2. YAZILIM GELİŞTİRME SÜREÇLERİNİN EVRİMİ **11**
2.1. Yazılım Geliştirme Döngüsü 11
2.2. Waterfall Şelale Modeli 12
2.3. Agile (Çevik) Yöntem 13
2.4. DevOps 15
2.5. Sürekli Entegrasyon 18
2.6. Sürekli Test 18
2.7. Sürekli Teslimat ve Sürekli Dağıtım 19
2.8. DevOps Yaşam Döngüsü 20
2.9. DevOps Pratiklerinin Adaptasyon Zorlukları 21
2.10. DevSecOps 21
3. BULUT BİLİŞİM **23**
3.1. Bulut Bilişimin Karakteristik Özellikleri 23
3.2. Bulut Bilişimin Avantajları 24
3.3. Bulut Bilişimin Dezavantajları 25
3.4. Bulut Bilişim Çeşitleri 26
3.5. Bulut Bilişim Modelleri 27
3.6. Sunucusuz Modeller 29
4. UYGULAMALARIN CANLI ORTAMA DAĞITILMASI **31**
4.1. Monolitik ve Mikroservis Uygulamalar 31
4.2. Bare Metal (Fiziksel) Sunucular 33
4.3. Sanallaştırma 34

4.4. Container (Konteynır) Teknolojisi 36
4.5. Kubernetes 38
4.6. GitOps 41
4.7. Kod Olarak Altyapı (Infrastructure as a Code) 43
5. GÜNÜMÜZDE VERİ VE YAPAY ZEKA UYGULAMALARI 46
5.1. DataOps 47
5.2. ETL 47
5.3. Gerçek Zamanlı Veri İşleme 48
5.4. Öneri Sistemleri 48
5.5. Generative Artificial Intelligence (Yenilikçi Yapay Zeka) 48
6. MATERYAL VE YÖNTEM 50
6.1. Uygulama 51
6.2. Geliştirilen Mikroservisler 55
6.3. AWS 58
6.4. AWS'nin Global Altyapısı 60
6.5. AWS'nin Ağ Altyapısı 61
6.6. AWS'nin Güvenlik Altyapısı 62
6.7. Kullanılan AWS Servisleri 63
6.8. Uygulamanın Bulut Ortamına Kurulması 66
6.9. Uygulamanın Kubernetes'e Kurulumu 78
6.10. DevOps Otomasyonu 81
6.11. GitOps Otomasyonu 90
6.12. Veri ve Yapay Zeka Servislerinin Kurulumları 93
6.13. Mikroservisler Arası Gecikmelerin Ölçülmesi 101
6.14. İzleme 103
7. SONUÇ 105
7.1. Tartışma 105
7.2. Değerlendirme 106
7.3. Öneriler 108

ÖZET

BULUT TABANLI DEVOPS UYGULAMALARI İLE YAZILIM GELİŞTİRME ve DAĞITIM SÜREÇLERİNİN İYİLEŞTİRİLMESİ

Bu tez çalışmasının amacı, bulut tabanlı DevOps uygulamaları kullanılarak yazılım geliştirme ve dağıtım süreçlerinin uçtan uca otomatikleştirilmesi ve performans açısından iyileştirilmesidir. Bu kapsamda DevOps, GitOps, IaC (Kod Olarak Altyapı) ve DataOps yaklaşımları, modern bir bulut-yerel mimari üzerinde bütüncül bir biçimde ele alınmış ve gerçekçi bir mikroservis tabanlı uygulama senaryosu üzerinde uygulanmıştır.

Çalışmada, tasarlanan DevOps ve GitOps tabanlı CI/CD hattı ile tekil kod değişikliklerinin derleme, test, güvenlik taraması ve dağıtım adımlarından geçerek üretim ortamına aktarım süreleri ölçülmüştür. Elde edilen sonuçlar, literatürde büyük ölçekli projelerde günler veya aylar sürebilen geleneksel dağıtım süreçlerinin, önerilen mimari ile dakikalar mertebesine indirilebildiğini nicel olarak ortaya koymuştur. Altyapı katmanında ise tüm bulut kaynaklarının IaC yaklaşımıyla tanımlanması sayesinde Kubernetes kümesi, sanal makineler ve yardımcı servislerin saniyeler ile dakikalar içerisinde otomatik olarak sağlanabildiği gözlemlenmiştir.

Ayrıca, sunucusuz bulut servisleri üzerine inşa edilen veri mimarisi sayesinde altyapı yönetim yükü azaltılmış; kaynak sağlama, ölçekleme ve işletim süreçlerinin büyük ölçüde bulut sağlayıcı tarafından yönetilmesi mümkün hale getirilmiştir. Bu mimari üzerinde DataOps yaklaşımıyla hem toplu veri işleme boru hatları hem de gerçek zamanlı akış iş yükleri kurulmuş; öneri sistemi ve bilgi tabanlı sohbet robotu gibi yapay zekâ bileşenleri aynı veri altyapısı üzerinden entegre edilmiştir. Sonuç olarak, yazılım, veri ve yapay zekâ yaşam döngüsünün uçtan uca otomatik, izlenebilir ve düşük gecikmeli bir şekilde yönetilebildiği gösterilmiştir.

ABSTRACT

IMPROVING SOFTWARE DEVELOPMENT AND DEPLOYMENT PROCESSES WITH CLOUD-BASED DEVOPS APPLICATIONS

The aim of this thesis is to automate and improve end-to-end software development and deployment processes by utilizing cloud-based DevOps practices. In this context, DevOps, GitOps, Infrastructure as Code (IaC), and DataOps approaches are integrated within a modern cloud-native architecture and applied to a realistic microservice-based application scenario.

In the study, a DevOps- and GitOps-driven CI/CD pipeline is designed, and the execution times of individual code changes from build, test, security scanning, and deployment stages to the production environment are quantitatively measured. The experimental results demonstrate that deployment processes, which may take days or even months in traditional large-scale software projects reported in the literature, can be reduced to the order of minutes using the proposed architecture. On the infrastructure layer, defining all cloud resources through IaC enables Kubernetes clusters, virtual machines, and auxiliary services to be provisioned automatically within seconds to minutes.

Furthermore, by leveraging serverless cloud services, the operational overhead of infrastructure management is significantly reduced, allowing resource provisioning, scaling, and operational tasks to be largely handled by the cloud provider. On this serverless foundation, both batch-oriented data pipelines and real-time streaming workloads are implemented using a DataOps perspective. In addition, artificial intelligence components such as a recommendation system and a knowledge-based chatbot are integrated over the same data infrastructure. As a result, the study demonstrates that the software, data, and AI lifecycle can be managed in an end-to-end automated, observable, and low-latency manner.

KISALTMALAR

API : Application Programming Interface (Uygulama Programlama Arayüzü)

ARM : Advanced RISC Machines / Arm architecture (Arm mimarisi)

AWS : Amazon Web Services (Amazon Web Servisleri)

BT : Bilgi Teknolojileri

CI/CD : Continuous Integration / Continuous Delivery (or Deployment) (Sürekli Entegrasyon / Sürekli Teslimat (veya Dağıtım))

CPU : Central Processing Unit (Merkezi İşlem Birimi (işlemci))

EBS : Elastic Block Store (Esnek Blok Depolama)

EC2 : Elastic Compute Cloud (Esnek Hesaplama Bulutu)

ECR : Elastic Container Registry (Esnek Konteyner Kayıt Deposu)

EKS : Elastic Kubernetes Service (Yönetilen Kubernetes Hizmeti)

ELB : Elastic Load Balancing (Esnek Yük Dengeleme)

ELT : Extract, Load, Transform (Çıkar, Yükle, Dönüştür)

ETL : Extract, Transform, Load (Çıkar, Dönüştür, Yükle)

FX : Foreign Exchange (Döviz (piyasası/işlemleri))

HTTP : HyperText Transfer Protocol (Köprü Metni Aktarım Protokolü)

IAAS : Infrastructure as a Service (Hizmet Olarak Altyapı)

IAC : Infrastructure as Code (Kod Olarak Altyapı)

IAM : Identity and Access Management (Kimlik ve Erişim Yönetimi)

IOT : Internet of Things (Nesnelerin İnterneti)

IP : Internet Protocol (İnternet Protokolü)

JSON : JavaScript Object Notation (JavaScript Nesne Gösterimi)

KMS	:	Key Management Service (AWS KMS) (Anahtar Yönetim Hizmeti)
NOSQL	:	Not Only SQL (SQL-dışı veritabanları)
PAAS	:	Platform as a Service (Hizmet Olarak Platform)
RBAC	:	Role-Based Access Control (Rol Tabanlı Erişim Denetimi)
RDS	:	Relational Database Service (İlişkisel Veritabanı Hizmeti)
REST	:	Representational State Transfer (Temsili Durum Aktarımı)
S3	:	Simple Storage Service (Basit Depolama Hizmeti)
SAAS	:	Software as a Service (Hizmet Olarak Yazılım)
TLS	:	Transport Layer Security (Taşıma Katmanı Güvenliği)
VPC	:	Virtual Private Cloud (Sanal Özel Bulut)
YAML	:	YAML Ain't Markup Language (YAML (biçim belirtim dili))

ŞEKİL LİSTESİ

Şekil 2.1 DevOps yaklaşımında geliştirmeden canlı ortama geçiş aşamaların özetlenmesi ..20

Şekil 2.2 DevOps Yaşam Döngüsü 21

Şekil 3.1 Bulut Bilişim Hizmet Modelleri 28

Şekil 4.1 Fiziksel Sunucuların Yapısı.....34

Şekil 4.2 Yazılım Düzeyinde Sanallaştırma 36

Şekil 6.1 Kullanıcının genel mimarisi 52

Şekil 6.2 Uygulamanın giriş kısmı 53

Şekil 6.3 Kullanıcı uygulamaya giriş yaptıktan sonra kullanıcıya özet sayfa.....53

Şekil 6.4 Kullanıcının beğendiği filmleri gösteren sayfa.....54

Şekil 6.5 Kullanıcının yapay zeka ile interaktif olarak konuşabildiği sayfa 54

Şekil 6.6 Kullanıcının film araması yapabileceği sayfa.....54

Şekil 6.7 Kullanıcının dizi araması yapabileceği sayfa 55

Şekil 6.8 Kullanıcının yeni üyelik açabileceği sayfa 55

Şekil 6.9 Kullanılan programlama dillerin yüzdesel dağılımı 58

Şekil 6.10 Data yığınına ait ekran görüntüsü.....77

Şekil 6.11 Ağ bileşenlerinin kurulum süresi.....77

Şekil 6.12 S3 kovasına ait ekran görüntüsü.....78

Şekil 6.13 Kubernetes altyapısı 79

Şekil 6.14 Kubernetes servislerinin AWS servisleriyle olan etkileşimi.....80

Şekil 6.15 DevOps Otomasyon Mimarisi.....82

Şekil 6.16 İlk otomasyona ait ekran görüntüsü.....89

Şekil 6.17 Film servisine ait otomasyon.....90

Şekil 6.18 Arama servisine ait otomasyon.....90

Şekil 6.19 GitOps mimarisi.....91

Şekil 6.20 GitOps sürecinin ilk aşaması – Güncellemenin yapılması.....92

Şekil 6.21 GitOps sürecinin ikinci aşaması – Uygulamanın senkronize edilmesi.....92

Şekil 6.22 GitOps sürecinin üçüncü adımı – Yeni podların ayağa kalkması.....92

Şekil 6.23 ArgoCD ekran görüntüsü.....93

Şekil 6.24 Veri ve Yapay Zeka mimarisi.....94

Şekil 6.25 Filmler için ETL Glue job'ı.....96

Şekil 6.26 Diziler için ETL job'ı.....97

Şekil 6.27 Kinesis ile gerçek zamana yakın veri akışının gösterilmesi.....98

Şekil 6.28 Diziler için QuickSight panosu.....99

Şekil 6.29 Filmler için QuickSight panosu.....100

Şekil 6.30 Bedrock job süresi.....101

Şekil 6.31 Linkerd ekran görüntüsü.....102

Şekil 6.32 Küme sağlığı hakkında genel bilgi içeren panel.....104

Şekil 6.33 Küme sağlığı hakkında panel.....104

TABLO LİSTESİ

Tablo 6.1 Uygulamanın klasör yapısı 51

Tablo 6.2 AWS servislerinin kurulum süresi 66

Tablo 6.3 Kubernetes'e ait nodegroup (sanal makine) yapısı 81

Tablo 6.4 DevOps otomasyonunda kullanılan testler ve açıklamaları 84

Tablo 6.5 Bütün Servislerin aynı anda çalıştırmasına ait çalışma süreleri 85

Tablo 6.6 Bütün servislerin aynı anda tekrar çalışmasına ait otomasyon süreleri 87

Tablo 6.7 Servislerin ayrı ayrı güncellenmesi 88

Tablo 6.8 ETL joblarının tamamlama süreleri 96

Tablo 6.9 Personalize servisine ait makine öğrenmesi seti eğitim süreleri 97

Tablo 6.10 Mikroservisler arasındaki gecikme süreleri 102

1. GİRİŞ

Bilgi teknolojisinin hızlı evrimi, yazılımların geliştirilme, dağıtılma ve tüketilme biçimini kökten değiştirmiştir. Son yirmi yılda kuruluşlar, fiziksel sunucularda çalışan monolitik uygulamalardan bulut destekli esnek, dağıtık sistemlere geçiş yapmıştır. Bu geçiş, yazılım geliştirme yaşam döngüsünü yeniden tanımlayan bulut bilişim, mikro hizmetler, DevOps, GitOps, DataOps ve konteyner orkestrasyonu gibi birkaç önemli paradigma tarafından desteklenmiştir. Aynı zamanda, yalnızca ölçeklenebilir altyapı değil, aynı zamanda gelişmiş veri odaklı yaklaşımlar da gerektiren öneri sistemleri ve akıllı sohbet robotları gibi yeni uygulama alanları ortaya çıkmıştır.

Bu tez, bu paradigmaları pratikte inceleyerek, modern yazılım mühendisliğindeki etkileşimlerine ve performans üzerindeki ölçülebilir etkilerine odaklanmaktadır. Çalışma özellikle DevOps'ta işlem hattı yürütme süreleri, bulut kaynak sağlama hızı, ETL (Extract, Transform, Load veya Çıkar, Dönüştür, Yükle) işlem süresi, akış gecikmeleri, Kubernetes yanıt verme hızı ve mikro hizmetten mikro hizmete iletişim gecikmeleri gibi metrikleri değerlendirmektedir. Tez, teorik kavramları ampirik sonuçlarla birleştirerek, çağdaş mimarilerin ve uygulamaların yazılım güvenilirliğini, ölçeklenebilirliğini ve verimliliğini nasıl şekillendirdiğine dair bütünsel bir bakış açısı sunmaktadır.

Bulut bilişim, esnek ölçeklenebilirlikle bilgi işlem, depolama ve ağ kaynaklarına isteğe bağlı erişim sunan modern uygulama dağıtımının omurgasını oluşturur. Geleneksel şirket içi altyapılar önemli miktarda ön yatırım ve uzun tedarik döngüleri gerektirirken, bulut platformları kaynakların saniyeler içinde örneklendirilmesine olanak tanır.

Daha geniş bulut paradigması içinde, sunucusuz bilişim güçlü bir soyutlama olarak ortaya çıkmıştır. Geliştiriciler, sunucuları doğrudan yönetmek yerine, yalnızca gerçek tüketim için faturalandırılan, isteğe bağlı olarak yürütülen işlevleri veya kısa ömürlü hizmetleri tanımlar. Bu model, operasyonel ek yükü azaltır, inovasyonu hızlandırır ve maliyet verimliliğiyle uyumludur. Kaynak oluşturma sürelerinin ölçülmesi (örneğin, bir sanal makinenin, konteynerin veya sunucusuz işlevin ne kadar hızlı örneklendirilebileceği), bulut benimsemenin pratik avantajlarını anlamada merkezi bir ölçüttür.

Monolitik uygulamalar, başlangıçta tasarlanması daha kolay olsa da, ölçeklendirme ve değişen gereksinimlere uyum sağlama konusunda zorluk çeker. Mikroservis mimarisi,

uygulamaları bağımsız ve birbirinden bağımsız servislere ayırarak bu zorluğun üstesinden gelir. Her mikro servis belirli bir iş kapasitesinden sorumludur ve diğerleriyle API'ler (Application Programming Interface / Uygulama Programlama Arayüzü) veya mesajlaşma protokolleri aracılığıyla iletişim kurar.

DevOps, geliştirme ve operasyonları otomasyon, sürekli entegrasyon, sürekli teslimat veya sürekli dağıtımı vurgulayan birleşik bir döngüye entegre eder. Temel amaç, kod yazma ile kullanıcılara sunma arasındaki süreyi azaltmaktır. Derlemeleri, testleri ve dağıtımları otomatikleştirerek kuruluşlar daha hızlı sürüm döngüleri ve daha yüksek kaliteli yazılımlar elde edebilirler. Ancak güvenlik giderek daha fazla birinci sınıf bir endişe haline gelmiştir. DevSecOps, güvenlik kontrollerini (güvenlik açığı taraması, politika uygulama ve uyumluluk denetimleri gibi) doğrudan otomasyon süreçlerine entegre ederek DevOps ilkelerini genişletir. Güvenliği ayrı bir aşama olarak ele almak yerine, DevSecOps bunu yaşam döngüsü boyunca paylaşılan bir sorumluluk haline getirir.

GitOps, versiyon kontrol sistemlerini altyapı ve uygulama yapılandırmaları için tek doğru kaynak olarak ele alarak DevOps'u temel alır. Geliştiriciler, değişiklikleri bulut ortamlarına manuel olarak uygulamak yerine, istenen durum tanımlarını (örneğin, Kubernetes için) versiyon kontrol sistemine işler. Daha sonra sistem bu istenen durumları sağlamaya çalışır. Bu yaklaşım, altyapı değişiklikleri için denetlenebilirlik, tekrarlanabilirlik ve sürüm kontrolü sağlar. Kubernetes kaynaklarının yapılandırma değişikliklerine uyum hızının ölçülmesi, bir sistemin bir işleme tabi tutulduktan sonra amaçlanan durumuna ne kadar hızlı ulaşabildiğini yansıttığı için GitOps için özellikle önemlidir.

IaC (Infrastructure as a Code – Kod Olarak Altyapı), modern yazılım mühendisliğinin en kritik yapı taşlarından biridir. IaC, altyapı kaynaklarının manuel olarak değil, kod aracılığıyla tanımlanmasını ve yönetilmesini sağlar, AWS (Amazon Web Services – Amazon Web Servisleri) CloudFormation gibi araçlar, altyapının sürüm kontrolüne alınmasını, otomatik olarak test edilmesini ve tekrar tekrar dağıtılmasını sağlar. Bu yaklaşım, altyapı değişikliklerinin tıpkı uygulama yazılımları gibi versiyon kontrol sisteminde de izlenebilmesi nedeniyle hem DevOps hem de GitOps ilkeleriyle uyumludur.

Literatürde ayrı ayrı ele alınan bulut bilişim üzerinde sunucusuz sistemler kullanarak DevOps/GitOps/DataOps yaklaşımlarını tek bir kurumsal senaryoda birleştirip ölçümlerle doğrulaması bakımından özgün katkı sunmaktadır. Bu özgünlük, Kod olarak altyapı yapılan

kurulumların ve kaynakların takibini otomatik olarak hale getirilmesi, DevOps otomasyonunda her bir adımın (test, derleme, güvenlik taraması, dağıtım) detaylı süre analizlerinin yapılması, GitOps süreçlerinde kod değişikliklerinden canlı ortam senkronizasyonuna kadar geçen sürelerin izlenmesi ve bu üç yaklaşımın entegre edildiği hibrit mimaride uçtan uca performans metriklerinin bir arada sunulmasından kaynaklanmaktadır.

Bu çalışmanın amacı; bulut-yerel ilkeleri temel alan bir başvuru mimarisi üzerinde **DevOps, GitOps, IaC (Infrastructure as Code) ve DataOps** yaklaşımlarını bütüncül biçimde birleştirerek yazılımın geliştirme–test–dağıtım süreçlerini **otomatik**, **izlenebilir** ve **tekrarlanabilir** hale getirmek; bu mimariyi **mikroservis** tabanlı bir uygulama ile gerçek ortamda kurup çalıştırarak sağlanan **operasyonel çeviklik** ve **verimlilik** kazanımlarını **nicel olarak ölçüp** ortaya koymaktır. Bu kapsamda tezde, bulut bilişim entegrasyonu, konteyner imajlarının kurumsal ölçekte **sürüm yönetimi ve otomasyonu**, otomatik derleme/test/dağıtım **otomasyonu**, bildirimsel **GitOps** işleyişi, veri hattında **gerçek zamanlı akış** süreçleri ve yapay zeka süreçleri uçtan uca entegre edilmektedir.

Bu hedef doğrultusunda:

- Altyapı sağlama (provisioning) süreleri, sürekli entegrasyon ve dağıtım adımları ve sürüm yayılım/geri alma gecikmeleri ölçülecek,
- Mikroservisler arası uçtan uca gecikmeler ve ölçekleme tepkileri ile sistemin gözlemlenebilirlik metrikleri raporlanacak,
- Veri hattında toplu işler ve akış iş yüklerinin tamamlama süreleri karşılaştırılacak,
- Elde edilen bulgularla önerilen mimarinin güvenilirlik ve performans etkileri değerlendirilecek ve benzer kurumsal senaryolar için tekrarlanabilir bir şablon sunulacaktır.

Bu tezde geliştirilen modelin hedefi, bulut-yerel ilkeler doğrultusunda tasarlanan bir başvuru mimarisi üzerinde DevOps, GitOps, IaC ve DataOps yaklaşımlarını bütüncül biçimde entegre ederek yazılım, veri ve yapay zekâ yaşam döngüsünü uçtan uca otomatik, izlenebilir ve tekrarlanabilir hâle getirmek; bu mimariyi gerçekçi bir mikroservis tabanlı kurumsal senaryo üzerinde uygulayarak altyapı sağlama, CI/CD işlem hattı, veri/akış süreçleri ve

yapay zekâ bileşenlerine ilişkin zaman ve performans metriklerini nicel olarak ortaya koymak ve böylece benzer kurumsal ortamlarda yeniden kullanılabilecek ölçülebilir bir model sunmaktır.

1.1. Literatür Taraması

Bu çalışmada, sunucusuz (serverless) mimari ile DevOps uygulamalarının son on yıldaki evrimi sistematik biçimde incelenmekte ve serverless DevOps framework kavramı tarihsel bir perspektifle tanımlanmaktadır. 2015–2016 döneminde AWS Lambda gibi platformlarla altyapı yönetim yükünün azaldığı, ancak CI/CD entegrasyonunun, izleme ve hata ayıklama yeteneklerinin oldukça ilkel kaldığı; 2017–2019 döneminde Serverless Framework, AWS CodePipeline ve IaC yaklaşımları ile uçtan uca otomatik dağıtım hatlarının olgunlaşmaya başladığı gösterilmektedir. 2019 sonrasında mikroservislerle bütünleşen serverless mimariler sayesinde tutarlılık, ölçeklenebilirlik ve tekrarlanabilirlik artarken, 2021–2024 döneminde çoklu bulut (multicloud), maliyet optimizasyonu, gözlemlenebilirlik ve güvenlik taramalarının CI/CD hatlarına dâhil edilmesiyle serverless DevOps çerçevelerinin olgun bir bulut-yerel teslimat modeli sunduğu vurgulanmaktadır. Çalışmanın sonucu olarak, sunucusuz DevOps çerçevelerinin, uygun tasarlandığında, geleneksel altyapıya kıyasla daha çevik, ölçeklenebilir ve maliyet-etkin yazılım teslimi sağladığı; buna karşın güvenlik, gözlemlenebilirlik ve çoklu bulut yönetiminin hâlâ önemli araştırma ve iyileştirme alanları olduğu ifade edilmektedir [1].

Bu araştırma makalesi, bulut-yerel (cloud-native) CI/CD hatlarının kurumsal (enterprise) yazılım projelerinde dağıtım verimliliği üzerindeki etkisini inceleyerek, alanyazındaki vaka çalışmaları, endüstri raporları ve ampirik bulguları sentezlemektedir. Çalışma, GitOps yaklaşımı, Kubernetes tabanlı orkestrasyon ve IaC'nin bir araya gelmesiyle oluşan bulut-yerel CI/CD hatlarının performansını dört temel DevOps metriği üzerinden tartışmaktadır: dağıtım sıklığı, değişiklikler için geçiş süresi (lead time), ortalama kurtarma süresi (MTTR) ve değişiklik başarısızlık oranı. Literatürden derlenen nitel ve nicel kanıtlar, Kubernetes-yerel boru hatları, otomatik testler, geri alma (rollback) mekanizmaları ve gözlemlenebilirlik araçları (Prometheus, Grafana vb.) ile desteklenen cloud-native CI/CD yaklaşımlarının, geleneksel Jenkins/Bamboo temelli monolitik hatlara kıyasla dağıtım sıklığını ve hızını anlamlı düzeyde artırdığını, MTTR ve değişiklik başarısızlık oranlarını ciddi biçimde düşürdüğünü göstermektedir. Bununla birlikte, çalışma büyük ölçekli kurumsal ortamlarda

bu yaklaşımların benimsenmesinin dik bir öğrenme eğrisi, karmaşık teknoloji yığınları ve kültürel dönüşüm gerektirdiğini; GitOps süreçlerine, konteyner orkestrasyonuna ve beyanî (declarative) altyapıya hâkim uzman ekipler olmaksızın beklenen kazanımların tam olarak elde edilemeyeceğini vurgulamakta ve bu bağlamda bulut-yerel CI/CD dönüşümleri için iyi uygulama (best practice) önerileri sunmaktadır[2].

Bu çalışmada, GitHub, Jenkins ve sanal makinelerin bütünleşik kullanımıyla uçtan uca bir CI/CD boru hattı kuran DevOps uygulama çerçevesi hem kavramsal hem de uygulamalı olarak ele alınmaktadır. GitHub, merkezi sürüm yönetimi ve işbirliği platformu olarak konumlandırılırken, Jenkins derleme, test ve dağıtım görevlerini otomatikleştiren orkestrasyon sunucusu olarak yapılandırılmakta; sanal makineler ise üretim ortamını yansıtan izole test ve dağıtım ortamları sağlayarak çeviklik ve ölçeklenebilirlik kazandırmaktadır. Makale, geleneksel, manuel ağırlıklı geliştirme-dağıtım süreçleriyle bu entegre CI/CD hattını karşılaştırmakta; otomasyon, sürekli geri bildirim döngüleri ve izleme sayesinde dağıtım sürelerinin kısaldığını, insan hatalarına bağlı başarısızlıkların azaldığını ve ekipler arası işbirliğinin güçlendiğini göstermektedir. Ayrıca, entegrasyon karmaşıklığı, altyapı yönetim yükü ve güvenlik gibi konularda karşılaşılan zorluklar ile bunlara yönelik çözüm stratejileri tartışılmakta; sonuç olarak GitHub–Jenkins–VM entegrasyonunun, kurumsal bağlamda DevOps kültürünü destekleyen, tekrarlanabilir ve izlenebilir bir CI/CD altyapısı sunduğu ileri sürülmektedir [3].

Bu makale, bulut altyapılarında web uygulamaları için AWS tabanlı bir CI/CD boru hattının tasarım ve performans değerlendirmesine odaklanmaktadır. Çalışmada, farklı programlama dilleriyle geliştirilmiş gerçek zamanlı web uygulamalarının kaynak kodu GitHub ve Amazon S3 üzerinde tutulmakta; AWS CodePipeline, CodeBuild, Elastic Beanstalk, EC2 ve RDS bileşenleri kullanılarak derleme, konteyner imajı üretimi ve otomatik dağıtım uçtan uca otomatikleştirilmektedir. Deneysel sonuçlar, farklı dil ve çatıların ortalama dağıtım süresinin yaklaşık 60 saniye olduğunu, kaynak kod deposu olarak Amazon S3'ün GitHub'a göre marjinal olarak daha hızlı dağıtım süreleri sağladığını ve CloudFormation tabanlı dağıtımların süre bakımından Elastic Beanstalk'a göre daha verimli, buna karşın Elastic Beanstalk'ın kullanım kolaylığı açısından daha üstün olduğunu ortaya koymaktadır. Çalışma, böylece AWS ekosisteminde CI/CD kavramlarının pratik bir uygulamasını sunmakta; farklı araç ve servis kombinasyonlarının dağıtım süresi, esneklik ve operasyonel

karmaşıklık üzerindeki etkilerini karşılaştırmalı olarak göstererek bulut tabanlı web uygulaması geliştirme süreçlerinde otomasyonun somut faydalarını nicel verilerle desteklemektedir [4].

Bu çalışmada, ilişkisel veritabanı uygulamalarında CI/CD uygulamalarının faydaları iki endüstriyel vaka çalışması üzerinden nicel ve nitel yöntemlerle ölçülmektedir. Oracle tabanlı iki büyük ölçekli veritabanı projesinde, mevcut durumda el ile yürütülen entegrasyon ve dağıtım süreçleri analiz edilerek sürümleme stratejileri, veritabanı şema migrasyon araçları, otomatik testler, statik kod analizi ve izleme bileşenlerini içeren bir CI/CD boru hattı tasarlanmış ve projelere entegre edilmiştir. Öncesi-sonrası karşılaştırmalar, otomatikleştirilmiş boru hattı devreye alındıktan sonra başarısız dağıtım sayısında kayda değer azalma (bazı test ortamlarında %90'a varan düşüş), sistem istikrarında artış ve dağıtım sayısında artış olduğunu, dolayısıyla daha küçük ve sık değişikliklerle riskin azaldığını göstermektedir. Geliştiricilerle yapılan görüşmeler, manuel adımların ortadan kaldırılması sayesinde bilişsel yükün azaldığını, geri bildirim döngülerinin hızlandığını ve sürüm pencerelerine sıkı bağımlılığın sorgulanır hâle geldiğini ortaya koymaktadır. Yazarlar, önerilen veritabanı CI/CD hattının, versiyon kontrol stratejisi, migration aracı, otomatik test ve izleme gibi önkoşulları tanımlayan bir "referans mimari" olarak kullanılabileceğini; bu önkoşullar sağlanmadan veritabanı projelerinde CI/CD benimsemenin güç olduğunu vurgulamakta ve sonuç olarak sürekli entegrasyon ve teslim uygulamalarının veritabanı geliştirme süreçlerinde hem kaliteyi hem de ekip üretkenliğini belirgin biçimde iyileştirdiği sonucuna varmaktadır [5].

Ullah ve çalışma arkadaşları, uluslararası ölçekte bulut bilişim benimsemesini etkileyen faktörleri incelemek amacıyla **otomatik sistematik literatür taraması (ASLR)** yürütmüş ve örgütlerin buluta geçişinde rol oynayan fırsat, zorluk ve değerlendirme kriterlerini sınıflandırmıştır. Bulgular, bulutun kurumlara sunduğu başlıca imkânların ölçeklenebilirlik, esneklik, BT altyapısını sadeleştirme, yenilik hızı ve maliyet avantajı olduğunu; buna karşılık güvenlik ve mahremiyet riskleri, mevzuata uyum belirsizlikleri, hizmet kesintileri, veri egemenliği sorunları ve tedarikçiye kilitlenme (vendor lock-in) riskinin benimsemeyi sınırlayan temel dezavantaj alanları olarak öne çıktığını göstermektedir. Çalışmanın sonuç kısmında, bulut bilişimin teknik ve ekonomik potansiyelinin yüksek olmakla birlikte, kurumların bu teknolojiyi değerlendirirken sadece teknik performans kriterlerine değil; risk

yönetimi, hukuki çerçeve, sözleşme tasarımı ve yönetişim mekanizmalarını da içeren bütüncül bir değerlendirme rehberine ihtiyaç duyduğu vurgulanmaktadır [6].

Bu kapsamlı güvenlik anketi, bulut bilişimdeki temel tehditleri (veri ihlalleri, hizmet engelleme saldırıları, çok kiracılı mimaride izolasyon sorunları, sanallaştırma açıkları, zayıf API'ler vb.) sistematik biçimde sınıflandırmakta ve bunları kamu, özel ve hibrit bulut dağıtım modellerinin avantaj ve dezavantajlarıyla ilişkilendirmektedir. Çalışma, kamu bulutlarının ölçeklenebilirlik ve kullanım kolaylığı gibi güçlü yanlarına rağmen güvenlik ve kaynak konumu üzerindeki kontrolün sınırlı olmasının önemli bir dezavantaj olduğunu; özel bulutlarda kontrol ve özelleştirilebilirliğin yüksek ancak maliyet ve uzmanlık ihtiyacının fazla olduğunu; hibrit bulutun ise esneklik ve maliyet kontrolü açısından dengeli bir seçenek sunarken görünürlük ve entegrasyon karmaşıklığı gibi yeni zorluklar yarattığını ortaya koymaktadır. Sonuç kısmında, bulutun avantajlarının güvenlik riskleriyle birlikte "paylaşılan sorumluluk modeli" kapsamında yeniden tanımlanması gerektiği; kurumların ağ düzeyi, sanallaştırma katmanı ve uygulama düzeyinde çok katmanlı güvenlik kontrolleri uygulamadan bu avantajlardan güvenli şekilde yararlanamayacağı vurgulanmaktadır [7].

Uzoma, Idoko ve Enyejo (2024), sunucusuz hesaplama ve mikroservis mimarisinin bulut-yerel uygulamaların ölçeklenebilirliği ile blokzincir birlikte çalışabilirlik çerçeveleri üzerindeki etkisini hem literatür incelemesi hem de nicel performans analiziyle değerlendirmektedir. Çalışmada önerilen metrikler (ölçeklenebilirlik performans indeksi, throughput, gecikme, kaynak kullanımı ve birlikte çalışabilirlik başarım oranı) üzerinden monolitik, mikroservis, sunucusuz ve bunların blokzincir ile bütünleştirilmiş varyantları karşılaştırılmış; sonuçlar sunucusuz ve mikroservis mimarilerinin monolitik yapıya kıyasla belirgin biçimde daha yüksek ölçeklenebilirlik ve kaynak verimliliği sağladığını göstermiştir. Buna karşın blokzincir entegrasyonunun, fikir birliği ve doğrulama süreçleri nedeniyle gecikmeyi artırdığı ve çapraz-zincir işlem başarı oranını düşürdüğü ortaya konmuştur. Yazarlar, bulut-yerel sistemler için sunucusuz ve mikroservis kombinasyonunu güçlü bir mimari seçenek olarak önermekte; ancak blokzincir birlikte çalışabilirlik katmanlarının gecikme ve başarı oranı açısından optimize edilmesi gerektiğini vurgulamaktadır [8].

Ramadoni, Utami ve Al Fatta (2021), geleneksel DevOps uygulamalarında geliştiricilerin küme üzerinde doğrudan yetkiye sahip olmasından doğan güvenlik sorunları ile dağıtımı geri alma (rollback) işlemlerinin zahmetli ve hataya açık olmasını tartışarak, Argo CD ile

gerçekleştirilen GitOps yaklaşımının bu sorunları nasıl azalttığını deneysel olarak incelemektedir. Çalışmada Nginx uygulamasına ait Kubernetes tanımları bir Git deposunda tutulmuş, Argo CD'nin bu depo ile kümenin gerçek durumu arasındaki farklılıkları tespit ettiği, istenen durumdan sapma olduğunda bunu gösterdiği ve tek bir komutla kümeyi tekrar Git'teki tanıma döndürerek hem güncelleme hem de geri alma işlemlerini oldukça basitleştirdiği gösterilmiştir. Bulgular, bildirimci (declarative) ve Git odaklı bu yaklaşımın güvenliği güçlendirdiğini, küme yapılandırmasının Git'teki kayıtlı hâl ile uyumlu kalmasını sağladığını ve geri alma sürecini klasik CI/CD boru hatlarına göre daha hızlı ve izlenebilir kıldığını; sonuçta GitOps'un altyapıyı kodla yöneten ve Git kültürüne sahip kurumlar için DevOps olgunluğunu artıran, ancak yazılım yaşam döngüsünün bütünü için test ve izleme araçlarıyla birlikte düşünülmesi gereken bir yöntem olduğunu ortaya koymaktadır [9].

Bu makale, veri boru hatlarının tasarımında konteynerleştirilmiş mimariler (Docker/Kubernetes) ile sunucusuz mimarileri kavramsal düzeyde karşılaştıran ve her iki yaklaşımı maliyet, ölçeklenebilirlik ve sürdürülebilirlik eksenlerinde sistematik olarak tartışan bir derleme niteliğindedir. Yazar, konteynerleştirmenin tutarlı çalışma zamanı ortamı, taşınabilirlik ve Kubernetes tabanlı otomatik ölçeklenebilirlik gibi güçlü yönlerini vurgularken; yönetim karmaşıklığı, güvenlik yüzeyi ve operasyonel uzmanlık ihtiyacını başlıca sınırlılık olarak öne çıkarır. Buna karşılık sunucusuz mimarilerde "kullandığın kadar öde" maliyet modeli, otomatik ölçeklenme ve düşük operasyonel yük; düzensiz veya patlamalı iş yüklerine sahip veri boru hatları için önemli avantajlar olarak sunulmakta, ancak soğuk başlangıç gecikmeleri, yürütme süresi sınırları, vendor lock-in ve izleme/hata ayıklama güçlükleri kritik dezavantajlar olarak çerçevelenmektedir. Çalışmanın önemli katkısı, toplu işleme, gerçek zamanlı akış, karmaşık bağımlılıklar ve maliyet optimizasyonu gibi tipik senaryolar üzerinden hangi durumda konteyner, hangi durumda sunucusuz mimarinin daha rasyonel bir tercih olacağını örneklerle açıklayan pratik bir karar çerçevesi ve hibrit mimari önerisidir; özellikle uzun süreli ve bağımlılığı yüksek iş yükleri için konteynerleri, olay-tabanlı patlamalı yükler için sunucusuz fonksiyonları birleştiren karma yaklaşımları ön plana çıkarır. Bununla birlikte makale, deneysel bir benchmark veya nicel performans ölçümü sunmadığı için daha çok mimari kararları yönlendiren kavramsal bir arka plan sağlamaktadır; dolayısıyla tezinde bu çalışmayı, senin gerçekleştirdiğin ölçümlere dayalı ampirik karşılaştırmaları teorik olarak gerekçelendiren, konteyner–serverless–hibrit üçlüsünü çerçeveleyen bir üst kaynak olarak konumlandırmak uygun olacaktır [10].

Bu makale, AWS SageMaker üzerinde makine öğrenimi modellerinin uçtan uca nasıl geliştirilebileceğini ve ölçeklenebilir biçimde üretime alınabileceğini açıklayan teknik bir incelemedir. Çalışmada; verinin S3 üzerinde saklanması, SageMaker bileşenleri ile veri ön işleme ve özellik çıkarımı yapılması, ardından yönetilen eğitim altyapısı kullanılarak model eğitimi ve hiperparametre ayarı gerçekleştirilmesi ele alınmaktadır. Devamında, eğitilen modellerin gerçek zamanlı uç noktalar üzerinden otomatik ölçeklenebilir şekilde sunulması ve SageMaker Model Monitor ile model performansı ile veri kaymasının izlenmesi tartışılmaktadır. Makale, ayrıntılı sayısal karşılaştırmalardan çok, SageMaker'ın veri hazırlama, eğitim, dağıtım ve izleme adımlarını tek bir bulut platformunda nasıl bir araya getirdiğini gösteren mimari ve süreç odaklı bir genel bakış sunmaktadır [11].

Bu makale, büyük veri işleme ile makine öğreniminin aynı mimari içinde bütünleştirilmesini amaçlayarak AWS EMR (Apache Spark) ve AWS SageMaker entegrasyonu üzerinden ölçeklenebilir, otomatik ve maliyet-etkin bir ML boru hattı tasarlamakta ve bunu deneysel bir vaka çalışmasıyla değerlendirmektedir. Önerilen mimaride, ham veriler EMR üzerinde Spark ile dağıtık biçimde temizlenip dönüştürülmekte, daha sonra SageMaker üzerinde dağıtık eğitimle modeller eğitilmekte ve gerçek zamanlı/batch uç noktalar üzerinden üretime alınmakta; Step Functions, Lambda ve CloudWatch kullanılarak veri alma–ön işleme–eğitim–dağıtım adımları uçtan uca otomatikleştirilmektedir. Deneysel sonuçlar, dağıtık Spark kullanımıyla veri ön işlem süresinin belirgin biçimde kısaldığını, SageMaker'ın yönetilen eğitim altyapısının tek node'a göre yaklaşık %34 daha düşük eğitim süresi sağladığını ve Spot instance kullanımıyla toplam maliyetin yaklaşık %50 oranında azaltılabildiğini göstermekte; buna karşın EMR–S3–SageMaker arasındaki veri transfer gecikmeleri, otomatik ölçeklemenin her senaryoda verimli çalışmaması ve gerçek zamanlı çıkarım gecikmesi gibi pratik sorunlara dikkat çekilmektedir. Çalışma, bu sorunlara yönelik olarak Data Wrangler ile entegrasyon, yönetilen ölçekleme, checkpointing ve serverless inference gibi optimizasyon stratejileri önermekte; genel olarak EMR ve SageMaker'ın birlikte kullanımıyla büyük ölçekli ML iş yükleri için yapılandırılabilir, ölçeklenebilir ve görece maliyet-etkin bir boru hattı kurulabileceği sonucuna varmakta, ancak çoklu bulut birlikte çalışabilirliği ve gerçek zamanlı işleme gibi konuları geleceğe dönük araştırma alanı olarak bırakmaktadır [12].

Literatürde bulut üzerinde DevOps, GitOps ve DataOps yaklaşımlarını ya ayrı ayrı ele alan ya da yalnızca belirli CI/CD metriklerine (deployment sıklığı, lead time, MTTR, CFR vb.) odaklanan, çoğunlukla tek bir uygulama veya sınırlı bir boru hattı senaryosu inceleyen çalışmalar bulunurken, bu tez çalışmasının katkısı; bu yaklaşımları tek bir kurumsal senaryoda bütüncül bir referans mimari üzerinde **birleştirip** uçtan uca **nicel olarak** değerlendirmesidir. Çalışmada altyapı sağlama süreleri, CI/CD boru hattındaki test, derleme, güvenlik taraması ve dağıtım adımlarının çalışma süreleri, GitOps ile kod değişikliklerinden canlı ortama senkronizasyona kadar geçen gecikmeler, mikroservisler arası uçtan uca yanıt süreleri, toplu ETL ve gerçek zamanlı akış iş yüklerinin tamamlama süreleri ile öneri sistemi ve bilgi tabanlı sohbet robotu gibi yapay zekâ bileşenlerinin eğitim ve çalıştırma süreleri sistematik biçimde ölçülmekte ve karşılaştırmalı olarak raporlanmaktadır. Böylece tez, literatürde daha çok kavramsal düzeyde veya sınırlı metrik setleriyle ele alınan cloud-native CI/CD tartışmasını; DevOps, GitOps, IaC ve DataOps'un tek bir gerçekçi film/dizi platformu senaryosu üzerinde birlikte çalıştığı, tekrar edilebilir bir mimari şablon ve ayrıntılı süre/performans metrikleri setiyle sayısal olarak derinleştirmekte, benzer kurumsal ortamlarda hem yazılım teslimatı hem veri/AI boru hatları için kıyaslama yapılabilecek somut bir deneysel zemin sunmaktadır.

2. YAZILIM GELİŞTİRME SÜREÇLERİNİN EVRİMİ

Yazılım geliştirme, günümüzün en dinamik ve hızla değişen disiplinlerinden biridir. Teknolojik gelişmeler, farklı kullanıcı beklentileri ve artan rekabet, yazılım üretim süreçlerini sürekli olarak dönüştürmektedir. Bu bağlamda, Yazılım Geliştirme Yaşam Döngüsü, yazılım projelerinin kavramsallaştırmadan üretim ve bakıma kadar tüm sürecini tanımlayan temel bir çerçeve sunar. Ancak bu çerçeve, tek ve sabit bir metodoloji dayatmaz; zaman içinde değişen ihtiyaçlara göre farklı uygulama stillerine yol açmıştır.

İlk aşamalara hakim olan Waterfall (Şelale) modeli, yazılım geliştirme yöntemlerinin doğrusal ve plan odaklı bir yorumuydu. Net planlama ve kapsamlı dokümantasyon gibi avantajlar sunsa da, değişime dirençli yapısı, günümüzün hızla değişen ihtiyaçlarını karşılayamamıştır. Bu noktada, daha kısa döngülere, müşteri odaklılığa ve esnekliğe odaklanan yeni Agile (Çevik) yaklaşımlar, yazılım geliştirmede önemli bir paradigma değişimi yaratmıştır.

Çevik yaklaşımın getirdiği bu hız, operasyonel süreçlere de yansıtılması ihtiyacını doğurmuş ve böylece DevOps kültürü ortaya çıkmıştır. Sürekli entegrasyon, sürekli teslimat veya sürekli dağıtım kavramlarıyla geliştirme ve operasyonlar arasındaki engeller ortadan kaldırıldı. DevOps'a kavramına ek olarak, güvenliği sürecin en başından itibaren entegre etmeyi hedefleyen DevSecOps yaklaşımı benimsendi. Tüm bu dönüşümler, yazılım geliştirme yaşam döngüsünün durağan olmadığını, sürekli geliştiğini ve uyum sağladığını göstermektedir.

2.1. Yazılım Geliştirme Döngüsü

Yazılım geliştirme döngüsü, yazılım geliştirme sürecini sistematik ve mantıksal olarak yapılandıran belirli aşamalardan oluşan bir metodolojidir. Yeni yazılım oluşturma, mevcut yazılımı değiştirme veya iyileştirme süreçlerini kapsar. Temel amacı, belirtilen zaman dilimleri ve maliyetler dahilinde müşteri beklentilerini karşılayan yüksek kaliteli yazılımlar geliştirmektir. Kalite güvencesi ve planlı süreç yönetimi ön plandadır.

Yazılım Geliştirme Yaşam Döngüsü, yazılım projelerini iyi tanımlanmış aşamalar boyunca yönlendiren yapılandırılmış bir yol haritası sunar. Her aşama, güvenilir, yüksek kaliteli ve sürdürülebilir bir ürün oluşturmaya katkıda bulunur.

- Gereksinim Analizi: Bu aşamada, müşteri ihtiyaçları kesin ve anlaşılır bir şekilde toplanır, netleştirilir ve belgelenir. Net bir gereksinim tanımı, tüm sürecin temelini oluşturur.
- Tasarım: Gereksinimlere dayanarak, yazılımın yapısı, mimarisi ve iş akışı tasarlanır. Bu adım, teknik çerçeveyi tanımlar ve kodlama için bir taslak görevi görür.
- Uygulama ve Birim Testi: Sistem, tasarım özelliklerine göre kodlanır. Tek tek modüller geliştirilir ve kendi başlarına doğru çalıştıklarından emin olmak için test edilir.
- Test: Tüm sistem bir bütün olarak test edilir. Hatalar belirlenir, düzeltilir ve dağıtımdan önce ürünün genel kalitesi sağlanır.
- Bakım: Dağıtımdan sonra, yazılım güncellemeler, hata düzeltmeleri ve geliştirmelerle korunur. Bu aşama, ürünün uzun vadeli sürdürülebilirliğini ve sürekli iyileştirilmesini sağlar.

Yazılım geliştirme yaşam döngüsü, sabit bir yapı göstermeyip zamanla değişmiştir. Başlangıçta daha katı ve sıralı modeller öne çıkarken, günümüzde daha esnek ve uyumlu yöntemler tercih edilmektedir. Bu durum, yazılım geliştirme süreçlerinin ihtiyaçlara ve koşullara göre sürekli evrildiğini ortaya koymaktadır.

2.2. Waterfall Şelale Modeli

Şelale modeli, yazılım geliştirmede en eski ve en bilinen yöntemlerden biridir. Doğrusal ve sıralı bir yaklaşıma dayanır; her aşama tamamlanmadan bir sonraki aşamaya geçilemez. Bu aşamalar genellikle gereksinim analizi, tasarım, geliştirme, test, dağıtım ve bakımı içerir.

Şelale modelinde, geliştirme süreci sıralı adımlarla ilerler. İlk olarak, tüm gereksinimler ayrıntılı olarak tanımlanır ve belgelenir. Bunu takiben, genel sistem mimarisi ve tasarımı hazırlanır. Tasarım tamamlandıktan sonra, yazılım kodlama aşaması başlar ve uygulama geliştirilir. Kodlama tamamlandıktan sonra, yazılım hatalarının tespit edilip düzeltildiği test aşaması başlar. Son olarak, ürün kullanıcıya sunulur, dağıtılır ve bir bakım süreciyle yazılımın sürdürülebilirliği sağlanır.

Şelale modelinin avantajları arasında, proje adımlarının kolay planlanmasını, izlenmesini ve yönetilmesini sağlayan açık ve düzenli yapısı yer alır. Aşamaların sıralı yapısı, zaman ve maliyet tahminlerinde yüksek öngörülebilirlik sağlar. Ayrıca, her aşamada üretilen kapsamlı

dokümantasyon, proje boyunca izlenebilirliği ve kontrolü kolaylaştırır. Bu model, gereksinimlerin en baştan tanımlandığı projelerde etkili bir şekilde çalışır ve özellikle katı düzenlemelere sahip sektörlerde tercih edilir.

Ancak, Şelale modelinin dezavantajları da vardır. En önemli sorunu esneklik eksikliğidir; bir aşama tamamlandıktan sonra geriye dönüp değişiklik yapmak oldukça zordur. Müşteri katılımı genellikle yalnızca başlangıçta ve sonda gerçekleştiğinden, beklentiler uyumsuz olabilir. Çoğu hata son aşama olan test aşamasında keşfedildiğinden, düzeltme maliyetleri artar. Ayrıca, uzun teslim süreleri ve değişen gereksinimlere uyum sağlayamama, bu modeli dinamik projeler için yetersiz kılar[14].

2.3. Agile (Çevik) Yöntem

Geleneksel yazılım geliştirme yöntemleri zahmetli, katı ve kapsamlı dokümantasyon gerektiriyordu. Gereksinimler en baştan tanımlanıp donduruluyor ve sonraki değişikliklere izin verilmiyordu. Bu durum, değişen müşteri ihtiyaçlarını karşılamada önemli zorluklar yaratıyordu.

Çevik yazılım geliştirme, insanlara ve müşteri memnuniyetine odaklanan yinelemeli ve artımlı bir yaklaşımdır. 2001 yılında yayınlanan Çevik manifestoda özetlenen ilkelere dayanarak, çalışan yazılım", müşteriyle sürekli iş birliği, değişime uyum sağlama ve kişilerarası etkileşim süreç boyunca ön plandadır. Çevik metodolojinin temel amacı, müşteri ihtiyaçlarını sürekli geri bildirimle karşılayarak kısa döngülerde çalışan yazılımlar sunmaktır. Bu, müşteri memnuniyetini artırır ve değişen koşullara hızlı adaptasyon sağlar. Ayrıca, ürünü erken bir aşamada kullanıma sunmak yatırım getirisini hızlandırır ve riskleri azaltır[15].

Şelale modelinin en büyük sorunu, test aşamasının sürecin en sonunda gerçekleştirilmesi ve hataların ancak proje neredeyse tamamlandığında keşfedilmesiydi; bu da maliyetli ve zaman alıcı çözümlere yol açıyordu. Çalışmada da belirtildiği gibi, Çevik yaklaşımların ortaya çıkmasının temel nedeni budur. Ancak Çevik metodoloji, bu sorunu kısa ve tekrarlayan yinelemelerle çözer: Her yinelemenin sonunda müşteriye çalışan bir sürüm sunulur, hatalar erken yakalanır ve müşteri geri bildirimleri sürece dahil edilir. Bu, değişen gereksinimlere kolayca uyum sağlanmasını ve projenin daha esnek ve müşteri odaklı bir şekilde ilerlemesini sağlar[16].

Çevik metodoloji, kısa ve yinelemeli döngüler (iterasyonlar) aracılığıyla çalışan yazılımlar üretmeyi ve müşteri geri bildirimleriyle sürekli iyileştirmeyi hedefler. Süreç, kapsamın başlangıçta tanımlandığı ancak proje boyunca gelişime açık kaldığı Sürüm Kapsamı aşamasında gereksinimlerin belirlenmesiyle başlar. Ardından, her yineleme için önceki yinelemelerin başarılarına göre uyarlanmış ayrıntılı bir Döngü Planı geliştirilir. Yapım Döngüsü aşamasında, yazılım parça parça ve yinelemeli bir şekilde geliştirilir ve her yinelemenin sonunda kullanıcıya somut ve kullanılabilir bir sürüm sunulur. Ardından, Müşteri Kontrol Noktası aşamasında, kalite müşteriyle birlikte gözden geçirilir, son çalışma doğrulanır ve geri bildirim toplanır. Son adım olan Sürüm Sonrası İnceleme, tüm süreci değerlendirir ve öğrenilen derslere ve müşteri geri bildirimlerine dayanarak bir sonraki sürüm için iyileştirme planları geliştirir. Bu döngüsel yapı sayesinde ürün sürekli olarak olgunlaşır, değişen gereksinimlere hızla uyum sağlar ve müşteri ihtiyaçlarını daha etkili bir şekilde karşılar[17].

Çevik yaklaşımın temel avantajları aşağıda sıralanmıştır [18]:

- **Değişime uyum sağlama**: Çevik yaklaşım, değişen gereksinimleri bir engel olarak değil, yazılımı son Kullanıcınının ihtiyaçlarına göre uyarlama fırsatı olarak görür.
- **Yüksek kalite ve hız**: Kısa yinelemeler, yüksek kaliteli ürünlerin hızlı ve verimli bir şekilde teslim edilmesini sağlar.
- **Erken geri bildirim**: Her yinelemenin sonunda müşteri geri bildirimi alınır ve bu da risklerin ve kusurların erken tespit edilmesini sağlar.
- **Sürekli iyileştirme**: Şeffaflık, güven ve hesap verebilirlik, sürekli öğrenmeyi ve sürekli süreç iyileştirmeyi destekler.
- **Karmaşık projelerin basitleştirilmesi**: Büyük projeler daha küçük bileşenlere bölünür; bu da iletişimi, bilgi paylaşımını ve genel üretkenliği artırır.
- **Müşteri memnuniyeti**: Değeri zamanında sunmak, müşteriler ve paydaşlar arasında memnuniyeti artırır.
- **Maliyet optimizasyonu**: Gereksiz işleri azaltarak Çevik yaklaşım, yatırım getirisini en üst düzeye çıkarır ve tasarımda basitlik sağlar.

Yukarıda sayılan avantajların yanı sıra, Çevik yaklaşımın aşağıdaki olumsuzlukları vardır [18][19]:

- **Büyük projelerde zayıf etkinlik**: Çevik, küçük ve aynı yerde bulunan ekiplerle en iyi şekilde çalışır, ancak ekip büyüdükçe verimliliği düşer. Büyük ölçekli projelerde maliyetler genellikle orantısız bir şekilde yükselir ve koordinasyon zorlaşır; bu da faydalarından daha ağır basabilir.
- **Dokümantasyon eksikliği**: Çevik, kapsamlı dokümantasyon yerine çalışan yazılıma önem verir ve bu da sonraki bakım aşamalarında sorunlara yol açabilir. Yeterli spesifikasyonlar olmadan ekipler proje hedeflerini gözden kaçırabilir ve gelecekteki geliştiriciler sistemi anlamakta veya güncellemekte zorlanabilir.
- **Bitmeyen proje riski**: Çevik sürekli değişiklikleri memnuniyetle karşıladığı için, projeler net bir bitiş noktası olmadan kolayca süresiz olarak uzayabilir. Bu durum, planlama, bütçeleme ve nihai teslimatta öngörülebilirliği azaltarak, sıkı teslim tarihlerine veya sözleşme gerekliliklerine uyum sağlamayı zorlaştırır.
- **Tasarım sınırlamaları**: Çevik, genellikle yapılandırılmış ürün tasarımını ihmal ederek, geliştirme ve kodlama faaliyetlerine büyük ölçüde odaklanır. Ön mimariye bu kadar önem verilmemesi, özellikle güçlü bir tasarım temeli gerektiren karmaşık sistemlerde uzun vadeli zorluklara neden olabilir.
- **Varsayımlara bağımlılık:** Çevik yaklaşım, yüksek düzeyde müşteri katılımı, güçlü iletişim ve yetenekli geliştirme ekipleri varsayar. Bu koşullar karşılanmazsa, metodolojinin başarısız olma olasılığı yüksektir ve bu gibi durumlarda alternatif yaklaşımlar daha uygun olabilir.
- **Legacy (eski) sistemler**: Çevik yaklaşımları legacy sistemlere uygulamak zordur; bu sistemler yinelemeli döngüler ile geliştirmeye uyacak şekilde kolay değiştirilemez.

2.4. DevOps

DevOps, yazılım yaşam döngüsü boyunca iş birliğini, otomasyonu ve sürekli iyileştirmeyi vurgulayan bir yazılım geliştirme metodolojisidir[20].

Çevik yöntemler yazılım geliştirmede esneklik ve hız sunarken, geliştirme (Dev) ve operasyon (Ops) ekiplerinin ayrı çalışmaları nedeniyle verimsizlikler ortaya çıkmıştır. Farklı hedef ve önceliklerden kaynaklanan bu ayrım, yazılım ürünleri ve hizmetleri arasındaki etkileşimde daha düşük verimliliğe yol açmıştır. Bu sorunu ele almak için, Patrick Debois tarafından 2009 yılında ortaya atılan DevOps yaklaşımı, geliştirme ve operasyon ekipleri

arasındaki iletişimi ve iş birliğini güçlendirmeyi amaçlamaktadır. DevOps, geliştirme ve operasyonları entegre ederek Çevik yaklaşımın eksikliğini giderir; bu da geliştirme ve test süreçlerini hızlandırır, ürün sürüm sürelerini kısaltır, bakım süreçlerini basitleştirir ve hata ve başarısız değişiklik olasılığını azaltır. DevOps, Çevik yaklaşımın bir uzantısı olarak düşünülebilir[21].

DevOps Manifestosu, 2009 yılında Patrick Debois liderliğinde ortaya çıkmış ve operasyonel süreçlere Çevik yaklaşımların uygulanmasıyla şekillenmiştir. Manifesto'nun temeli, geliştirme ve operasyon ekipleri arasındaki engelleri kaldırarak daha yakın bir iş birliği ve ortak sorumluluk yaratmaktır. Bu yaklaşım, yazılım geliştirme ve teslimat süreçlerinde hıza, güvenilirliğe ve sürekli teslimata öncelik verir. DevOps, yalnızca bir araç seti veya teknik bir yöntem olarak değil, aynı zamanda ekipler arasında güven, şeffaflık ve iletişim olmadan başarılı olamayacağı için kültürel bir dönüşüm olarak tanımlanır. Manifesto, yazılım yaşam döngüsünü uçtan uca, sürekli entegrasyon, otomasyon, ölçüm ve geri bildirim mekanizmalarıyla kapsar. Böylece, müşteri ihtiyaçlarına hızla uyum sağlayan, hataları erken tespit eden ve sürekli değer üreten bir yazılım geliştirme ekosistemi oluşturur[22][23].

Endüstride DevOps, otomasyon, sürekli entegrasyon ve sürekli dağıtım yoluyla hızlı ve güvenilir yazılım sunmak için yazılım geliştirme ve operasyon ekiplerinin iş birliği içinde yürüttüğü bir çalışmadır. Netflix, Amazon ve sağlık kuruluşları gibi çeşitli sektörlerdeki şirketler, ölçeklenebilirlik, hız ve müşteri memnuniyetinde avantajlar elde etmek için DevOps'u kullanmaktadır[24].

DevOps metodolojisinin temel avantajları aşağıdaki gibi sıralanmıştır [25]:

- **Gelişmiş İş Birliği**: Geliştirme, operasyon ve kalite güvence ekipleri arasındaki geleneksel engelleri ortadan kaldırır.
- **Daha Hızlı Pazara Sunma Süresi**: Yinelemeli geliştirme ve sürekli geri bildirimle değişen gereksinimlere hızlı yanıt verir ve küçük ve artımlı sürümler sunar.
- **Artan Güvenilirlik**: Sürekli testler (birim, entegrasyon, uçtan uca) aracılığıyla hataları erken tespit ederek güvenilirliği artırır. Sorunları gerçek zamanlı olarak izleyen ve hızlı çözüm sağlayan sürekli izleme araçlarıyla istikrarlı sistem performansı sağlar.
- **Maliyet Verimliliği**: Kaynak kullanımını optimize eder ve gereksiz masrafları en aza indirir.

Yazılım endüstrisi, kaliteyi koruyarak sık güncellemeler sunma baskısı altındadır. Güçlü bir kalite güvencesi olmadan yapılan hızlı sürümler, hatalara yol açarak kullanıcıların memnuniyetsizliğine neden olabilir. DevOps yaklaşımı, kaynak kontrolü, otomatik testler, ekipler arası iş birliği ve sürekli teslimat uygulamaları sayesinde bu sorunlara çözüm sunar. Böylece kuruluşlar, hızla değişen pazarlarda rekabet gücünü koruyacak şekilde daha kısa sürüm döngüleri, erken geri bildirim imkânı ve güvenilir ürünler geliştirebilir [26]. Bu çerçevede, Sürekli Test, Sürekli Entegrasyon, Sürekli Teslimat ve Sürekli Dağıtım gibi uygulamalar DevOps'un temel ilkelerini somutlaştırır.

Bu kavramları açıklamadan önce vurgulanması gereken iki temel unsur bulunmaktadır: sürüm kontrol sistemleri ve yazılım testleri.

Sürüm kontrol sistemleri, yazılım üzerinde yapılan değişiklikleri kaydeden, bu değişikliklerin yazarlarını izleyen ve proje geçmişini koruyan temel araçlardır. Bu sistemlerin amacı; iş birliğini kolaylaştırmak, yedekleme sağlamak, hata durumlarında önceki sürümlere geri dönmeye imkân tanımak ve yazılım yaşam döngüsü boyunca kod tabanının bütünlüğünü güvence altına almaktır. Günümüzde bu sistemler arasında **Git**, dağıtık mimarisi sayesinde en yaygın kullanılan çözüm haline gelmiştir. Git, her geliştiriciye merkezi bir sunucuya bağlı kalmadan çalışabilme, deponun tam bir yerel kopyasını saklama, dallanma (branching) ve birleştirme işlemleri yapma olanağı sunar. Bu esneklik, iş birliğini hızlandırır, paralel geliştirmeyi destekler ve üretkenliği artırır. Ayrıca Git, Agile ve DevOps ortamlarında daha hızlı teslimat döngüleri sağlayarak **CI/CD (Continuous Integration / Continuous Delivery – Sürekli Entegrasyon / Sürekli Teslimat)** süreçlerinin uygulanmasında temel bir unsur haline gelmiştir [27].

Yazılım testi, yazılım son kullanıcılara ulaşmadan önce hataları tespit etmeyi amaçlar. Temel olarak dört aşamaya ayrılır: birim testi, entegrasyon testi, sistem testi ve kabul testi. Birim testi, tek tek bileşenleri doğrularken, entegrasyon testi modüller arasında düzgün veri akışı sağlar. Sistem testi, işlevsel ve işlevsel olmayan gereksinimlerin karşılanıp karşılanmadığını değerlendirir ve kabul testi, sözleşmesel yükümlülüklere uyumu doğrular. Yazılım testi, önceden tanımlanmış senaryolar aracılığıyla kullanıcı etkileşimlerini simüle ederek, uygulamaların hem güvenilir hem de müşteri beklentileriyle uyumlu olmasını sağlar[28].

Bu iki temel, yani sürüm kontrolü ve yazılım testi, DevOps süreçlerinin temelini oluşturur ve bu süreçlerin prensipleri Sürekli Test, Sürekli Entegrasyon, Sürekli Teslimat ve Sürekli Dağıtım aracılığıyla genişletilir ve otomatikleştirilir.

2.5. Sürekli Entegrasyon

Sürekli Entegrasyon, yazılım değişikliklerinin düzenli olarak geliştirme yapılan bir versiyon kontrol sistemine işlendiği ve tüm geliştiricilerin kod tabanının en güncel sürümüyle çalışmasını sağlayan bir geliştirme uygulamasıdır. Bir değişiklik eklendiğinde, sistem otomatik olarak bir derlemeyi tetikler ve testleri yürütür. Bu otomasyon, insan hatası olasılığını azaltır, yazılımın tutarlılığını artırır ve geri bildirim döngüsünü hızlandırır.

Sürekli entegrasyonun önemli bir özelliği, her bir işlemenin kendi kendini test edecek şekilde tasarlanmış otomatik bir derleme tarafından doğrulanmasıdır. Bu, birim testlerinin ve entegrasyon testlerinin anında çalıştırıldığı ve ekiplerin hataları geliştirme sürecinin erken aşamalarında tespit etmelerine olanak tanıdığı anlamına gelir. Manuel testler yerine otomatik testlere güvenerek, hataların ortaya çıkma olasılığını en aza ve elde edilen sonuçların belirsizliğini azaltarak güvenilirliğini artırır.

Özünde, sürekli entegrasyon, yalnızca otomatik araçların kullanımıyla sınırlı değildir; aynı zamanda sık geliştirme, otomatik testler ve hızlı geri bildirime yönelik kültürel ve prosedürel bir bağlılığı temsil eder. Kuruluşların kod kalitesini korumasına, entegrasyon sorunlarını azaltmasına ve çevik ve iş birliğine dayalı yazılım geliştirme uygulamalarını desteklemesine olanak tanır[29].

2.6. Sürekli Test

Sürekli Test, DevOps otomasyonunda, entegrasyon sürecinin her aşamasında otomatik testlerin sürekli olarak yürütüldüğü temel bir uygulama olarak tanımlanır. Testlerin geliştirmenin sonuna kadar ertelendiği geleneksel yaklaşımların aksine, Sürekli Test, her kod değişikliğinin onaylanmasının hemen ardından otomatik testleri içerir. Her değişiklik, birim, entegrasyon ve işlevsel düzeyler de dahil olmak üzere önceden tanımlanmış bir otomatik test paketine tabi tutulan derleme sürecini tetikler. Bu testlerden herhangi biri başarısız olursa, derleme reddedilir ve geliştiriciye anında geri bildirim sağlanarak hatalı kodun süreç hattında daha fazla yayılması önlenir. Bu yaklaşım sayesinde, yazılım hataları mümkün olan en erken anda tespit edilerek hataları düzeltmenin maliyeti ve karmaşıklığı azaltılır. Dolayısıyla

Sürekli Test, testi nihai bir kalite eşiğinden entegre ve sürekli bir etkinliğe dönüştürerek, kalite güvencesini tüm yazılım yaşam döngüsüne yerleştirir ve hızlı, güvenilir ve müşteri odaklı sürümlerin oluşturulmasını sağlar[30].

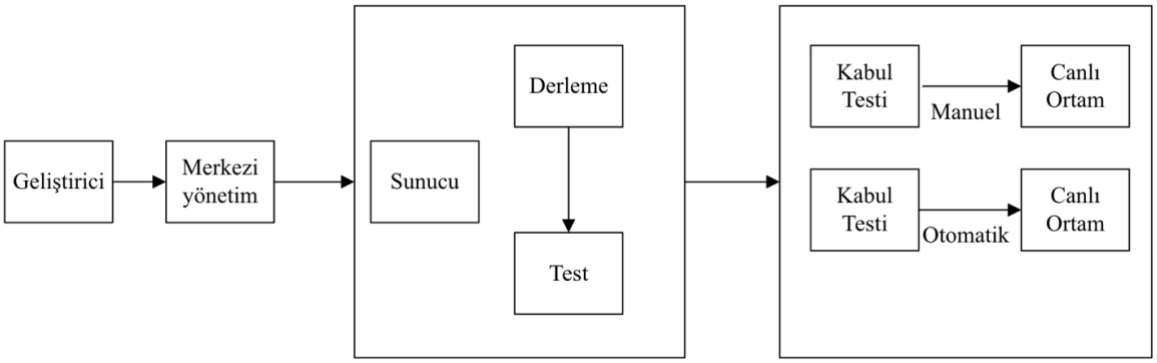
2.7. Sürekli Teslimat ve Sürekli Dağıtım

Sürekli teslimat (Continuous Delivery), sürekli entegrasyon'u takip eden ve yazılımın her zaman yayınlanabilir durumda olmasını sağlayan aşamadır. Kod değişiklikleri otomatik derlemeler ve testler aracılığıyla entegre edilip doğrulandıktan sonra, ortaya çıkan ürün, herhangi bir zamanda son kullanıcıya dağıtılabilecek şekilde hazırlanır. Temel özelliği, üretime aktarımın yine de manuel bir karar gerektirmesi ve bu sayede kuruluşlara zamanlama ve risk üzerinde kontrol sağlamasıdır. Sürekli dağıtım ise, son adımı otomatikleştirerek bu uygulamayı genişletir. İşlem hattından başarıyla geçen her değişiklik, insan müdahalesi olmadan anında üretim ortamına dağıtılır [31].

Şekil 2.1, modern yazılım mühendisliğinde birbiriyle bağlantılı uygulamalar olarak sürekli entegrasyon, sürekli teslimat ve sürekli dağıtım arasındaki ilişkiyi göstermektedir.

- Sürekli Entegrasyon otomasyonun temeli oluşturur. Geliştiriciler, kod değişikliklerini sıklıkla paylaşılan bir havuzda birleştirir ve burada otomatik derlemeler ve testler ile doğruluğu doğrular. Bu uygulama, geri bildirim döngülerini kısaltır, entegrasyon sorunlarını azaltır ve yazılımın her zaman oluşturulabilir durumda olmasını sağlar.
- Sürekli teslimat, sürekli entegrasyon üzerine kurulur. Otomatik derleme ve testlerden geçtikten sonra yazılımın her zaman üretime hazır durumda kalmasını sağlar. Teslimat, genellikle sistemin herhangi bir zamanda son kullanıcıya dağıtılabileceği anlamına gelir, ancak nihai sürüm kararı manuel kalır.

Sürekli Dağıtım, teslimatın ötesine geçer. önceki aşamalarından geçen her değişiklik, insan müdahalesi olmadan otomatik olarak üretim ortamına aktarılır. Bu, manuel onay adımlarını ortadan kaldırarak son kullanıcılara hızlı ve sürekli yeni özellik akışı sağlar.

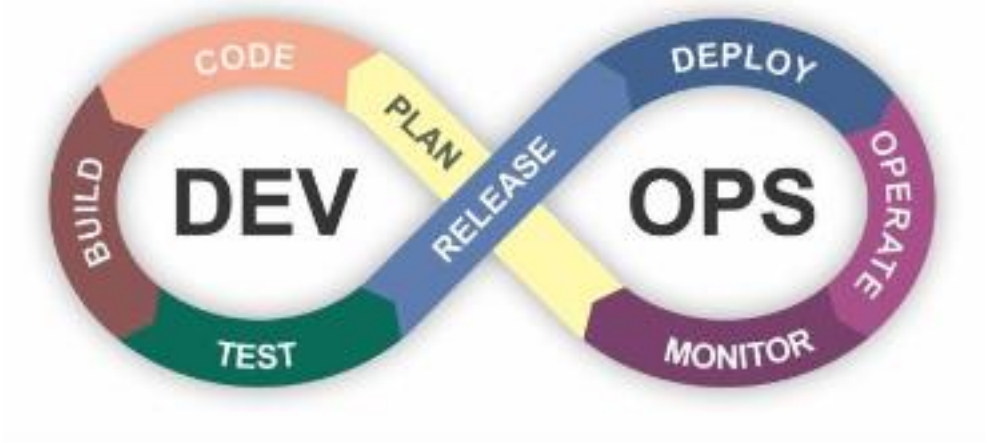


Şekil 2.1 DevOps yaklaşımında geliştirmeden canlı ortama geçiş aşamaların özetlenmesi

2.8. DevOps Yaşam Döngüsü

DevOps yaşam döngüsü, yazılım geliştirme ve operasyonlarını entegre ederek yazılım sistemlerinin daha hızlı teslimatını, daha yüksek kaliteyi ve sürekli iyileştirilmesini sağlayan yinelemeli ve sürekli bir süreç olarak tanımlanabilir. Otomasyon, iş birliği ve geri bildirimi vurgulayarak, her aşamanın bir sonrakini bilgilendirdiği ve geliştirdiği kapalı bir döngü oluşturur. İzole aşamalar olarak işlev görmek yerine, yaşam döngüsü planlamadan izlemeye kadar kesintisiz bir akışı temsil eder ve sistemlerin güvenilir, uyarlanabilir ve kullanıcı ihtiyaçlarına duyarlı kalmasını sağlar [32][33].

- Planlama: Gereksinimler belirlenir, hedefler tanımlanır ve geliştirme ve operasyon ekipleri, uygulama için bir yol haritası oluşturmak üzere iş birliği yapar.
- Kodlama ve Oluşturma: Yazılım, kalite ve tutarlılığı sağlayan otomatik entegrasyon süreçleri aracılığıyla geliştirilir, incelenir ve yürütülebilir eserlere dönüştürülür.
- Test: Birim, entegrasyon ve işlevsel testler de dahil olmak üzere sürekli test uygulamaları, yazılımın güvenilirliğini doğrular.
- Yayınlama ve Dağıtım: Doğrulandıktan sonra, yeni bir sürüm yayınlanmaya hazır hale gelir ve operasyonel ortama aktarılır. Bu aşamada, Sürekli Teslimat sürüm için manuel onayı içerirken, Sürekli Dağıtım, yapıtların üretime taşınması sürecini tamamen otomatikleştirir.
- İşletme ve İzleme: Sistem, performansının ve güvenliğinin sürekli olarak gözlemlendiği ortamda çalışır. İzleme sonuçları planlama aşamasına geri bildirim sağlayarak, DevOps yaşam döngüsünü karakterize eden sürekli bir iyileştirme ve geliştirme döngüsü oluşturur.



Şekil 2.2 DevOps Yaşam Döngüsü

2.9. DevOps Pratiklerinin Adaptasyon Zorlukları

Sistematik inceleme, DevOps'un önemli faydalar sunmasına rağmen, başarılı bir şekilde benimsenmesinin birçok zorlukla engellendiğini vurgulamaktadır.

En büyük engellerden biri, yetenekli profesyonellerin eksikliği ve yeni DevOps araçları ve uygulamalarıyla ilişkili dik öğrenme eğrisidir. Kuruluşlar genellikle eğitim ve yetenek geliştirmeye yeterli yatırım yapmakta zorlanır ve bu da beceri açığını daha da kötüleştirir. Bir diğer kalıcı zorluk ise, çalışanların iş güvencesizliğinden korkması, geleneksel süreçleri tercih etmesi veya DevOps dönüşümleri kapsamında sorumluluklar değiştiğinde değişim yorgunluğu yaşaması nedeniyle değişime dirençtir.

Teknik açıdan bakıldığında, eski sistemler ve güncelliğini yitirmiş altyapılar, modern otomasyon, test ve izleme araçlarının entegrasyonunu zorlaştırır. Bu sınırlamalar, önemli miktarda ön yatırım gerektirir ve DevOps uygulamalarına geçişi yavaşlatır. İletişim bozuklukları ve bölünmüş kurumsal kültürler, geliştirme ve operasyon ekipleri arasındaki iş birliğini daha da engelleyerek, DevOps'un temel sorumluluk paylaşım ilkesini zayıflatır[34].

2.10. DevSecOps

DevSecOps, "Geliştirme, Güvenlik ve Operasyonlar" kavramlarından türetilmiştir ve güvenliği en başından itibaren modern yazılım geliştirme yaşam döngüsüne entegre etmeyi amaçlayan bir paradigma olarak tanımlanır. Geleneksel yazılım geliştirme süreçlerinde güvenlik genellikle son aşamaya bırakılır ve bu da güvenlik açıklarının gecikmeli tespitine ve artan iyileştirme maliyetlerine yol açar. DevSecOps, bu sorunu güvenlik kontrolleri

sürecin başına alma yaklaşımıyla ele almayı amaçlar; güvenlik kontrolleri, yazılım geliştirme sürecinin planlama ve tasarım aşamalarından itibaren uygulanır. DevSecOps'un temel felsefesi, güvenliğin ayrı bir ekip tarafından yönetilen bağımsız bir süreç değil, yazılım geliştirme, operasyon ve güvenlik ekipleri arasında paylaşılan ortak bir sorumluluk olduğudur. Bu sayede güvenlik, yazılım geliştirme sürecinin her aşamasına entegre edilerek yazılımın son kullanıcıya daha güvenilir, daha hızlı ve daha sürdürülebilir bir şekilde teslim edilmesi sağlanır

DevSecOps'un uygulanmasındaki temel ilkeler şunlardır:

- Güvenlik testleri ve kontrolleri, kodlama ve entegrasyon aşamalarına entegre edilerek olası açıklar erken aşamada tespit edilir.
- Güvenlik testleri DevOps otomasyonun her aşamasında çalıştırılır.
- Otomatik güvenlik taramaları, güvenlik açığı yönetimi ve iyileştirmelerle hem hız hem doğruluk arttırılır.
- Güvenlik artık yalnızca ayrı bir güvenlik ekibinin sorumluluğunda değildir; geliştiriciler, operasyonlar ve güvenlik uzmanları tarafından ortaklaşa yönetilir

DevSecOps, güvenliğin DevOps sürecinin her aşamasına entegre edilmesiyle uygulanır:

- Plan: Güvenlik gereksinimleri yazılım gereksinimleriyle birlikte tanımlanır.
- Kod: Güvenli kodlama standartları uygulanır ve geliştiriciler temel güvenlik açıkları konusunda eğitilir.
- Derleme: Kaynak kod güvenlik taramaları otomatik araçlar kullanılarak gerçekleştirilir.
- Test: Saldırı simülasyonları, canlı ortam benzeri koşullar altında gerçekleştirilir.
- Yayınlama ve Dağıtım: Güvenlik taramalarında başarısız olan hiçbir sürüm üretime sunulmaz.
- İzleme: Canlı ortam güvenlik açısından sürekli izlenir ve olaylar tespit edildiğinde otomatik uyarı mekanizmaları tetiklenir

Bu uygulamalar, güvenlik ihlallerini sürecin geç aşamalarında tespit etmek yerine, en erken aşamada önlemeyi mümkün kılar[35][36][37][38].

3. BULUT BİLİŞİM

Günümüzde veri miktarının katlanarak artması, yapay zeka ve makine öğrenmesi uygulamalarının yaygınlaşması, işletmelerin geleneksel veri depolama ve işleme yöntemlerini yetersiz kılmaktadır. Özellikle büyük veri analitiği ve yapay zeka modellerinin eğitimi için gereken hesaplama gücü, şirketleri daha esnek ve ölçeklenebilir çözümlere yönlendirmektedir. Sosyal medya platformları, IoT (Internet of Things / Nesnelerin Interneti) cihazları ve akıllı sistemlerden akan devasa veri hacmi, geleneksel sunucu altyapılarının karşılayamayacağı boyutlara ulaşmıştır. Bu noktada cloud computing (bulut bilişim), kuruluşlara ihtiyaç duydukları depolama alanını, işlem gücünü ve esnekliği sağlayan modern bir teknoloji paradigması olarak öne çıkmaktadır. Bulut bilişim, işletmelere sadece ihtiyaç duydukları kadar kaynak kullanma ve bu kaynakları dinamik olarak ölçeklendirme imkanı sunarak, hem maliyet etkinliği hem de operasyonel verimlilik sağlamaktadır.

Bulut bilişiminin genel ve yegâne tanımını yapmak oldukça zordur, çünkü bulut bilişim teknolojisi sürekli gelişen yapısı ve çok yönlü kullanım alanlarıyla dinamik bir kavramdır. Farklı kurum ve kuruluşlar kendi perspektiflerine göre bulut bilişimi farklı şekillerde tanımlamaktadır. Bulut bilişim, internet üzerinden dinamik olarak kaynak tahsisi, izleme ve verimli kullanım sağlayan sanal bir kaynak havuz olarak tanımlanabilir.. Geçmiş ve gerçek zamanlı verileri entegre ederek daha doğru analizler ve akıllı hizmetler sunar. Kaynakları yazılım aracılığıyla otomatik olarak yönetir, böylece kullanıcıların sunucu veya çözüm satın alma ihtiyacını ortadan kaldırır. Kullanıcılar, ihtiyaçlarına göre talep üzerine bilişim kaynaklarına erişebilir[39]. Başka bir tanıma göre, bulut bilişim, veri merkezlerinde bilgi işlem, depolama ve ağ yönetim işlevlerini merkezi hale getirerek, müşterilerin verilere web tarayıcısı aracılığıyla erişebilmesini sağlayan bir bilişim paradigmasıdır[40].

3.1. Bulut Bilişimin Karakteristik Özellikleri

Bulut bilişiminin karakteristik özellikleri aşağıda sunulmuştur[41][42]:

- **Talep Üzerine Self-Hizmet**: Kullanıcılar, sağlayıcı ile insan etkileşimine ihtiyaç duymadan otomatik olarak sunucu zamanı ve ağ depolama gibi bilişim kaynaklarını kendi başlarına sağlayabilir.

- **Geniş Ağ Erişimi**: Kaynaklara ağ üzerinden erişilir ve bu kaynaklar, mobil telefonlar, tabletler, dizüstü bilgisayarlar gibi farklı istemci platformları tarafından standart mekanizmalarla kullanılabilir.
- **Kaynak Havuzlama**: Sağlayıcı, kaynaklarını birden çok kullanıcıya hizmet etmek için havuzlar ve taleplere göre fiziksel ve sanal kaynakları dinamik olarak atar. Kullanıcılar genellikle kaynakların tam konumunu bilmez ancak daha üst düzeyde bir konumu (ör. ülke, eyalet, veri merkezi) belirtebilir.
- **Hızlı Esneklik**: Kaynaklar talebe göre hızla artırılabilir veya azaltılabilir. Kullanıcılar, kaynakların sınırsız göründüğü ve ihtiyaç duyulduğu anda kullanılabileceği bir deneyim yaşar.
- **Ölçümlenebilir Hizmet**: Bulut sistemleri, kaynak kullanımını kontrol eder ve optimize eder; depolama, işlem gücü, bant genişliği gibi hizmet türlerine uygun olarak ölçümleme yapılır. Sağlayıcı ve kullanıcı için kaynak kullanımında şeffaflık sağlanır.
- **Maliyet Etkinliği**: Bulut, ekonomik enerji santrallerine yakın ve uygun maliyetli bölgelerde konuşlandırılabilir.
- **Çoklu Kullanıcı:** Bulut sağlayıcıları, altyapıyı birden fazla müşteriye uygun teknik bölümlendirmelerle kiralayarak talebe göre fiziksel hizmetleri sanallaştırılmış yazılım modeli olarak sunabilir.

3.2. Bulut Bilişimin Avantajları

Bulut bilişim, bilgi işlem kaynaklarına internet üzerinden erişim sağlayan bir teknolojidir. Bu yaklaşım, kullanıcıların kaynaklara ihtiyaç duyduklarında kolayca ulaşmalarını ve bunları esnek bir şekilde kullanmalarını mümkün kılar. Hem bireyler hem de işletmeler için birçok avantaj sunmaktadır. Bulut bilişimin en temel avantajları aşağıda sıralanmıştır [43][44]:

- Bulut bilişim, çoklu veri kopyaları, hata tolerans mekanizmaları ve düğümlerin değiştirilebilirliği gibi teknolojilerle yüksek güvenilirlik sağlar. Dağıtık bilişim, düğümlerin otomatik tahsisi ve arızalı düğümlerin görevlerinin diğer düğümlere devredilmesiyle kendini gösterir. Dağıtık depolama ise verilerin otomatik bölümlenmesi, yedekli depolama ve birim arızalarında verilerin yedeklenmesiyle sağlanır.

- Bulut bilişim, kullanıcı büyümesine bağlı olarak ölçeğini dinamik olarak ayarlayabilir. Hizmet sağlayıcılar, kullanıcı artışına göre sunucu düğümlerini artırarak hizmet kapasitelerini geliştirebilir. Bu, hem kullanıcılar hem de sağlayıcılar için maliyet avantajı sağlar. Esnek kaynak planlaması, kaynakların talebe göre tahsisini mümkün kılar.
- Bulut bilişim, kullanıcıların ihtiyaçlarına göre farklı modülleri seçip yalnızca kullandıkları kaynaklar için ödeme yapmalarını sağlar. Gereksiz kaynak kullanımını önleyerek kaynak havuzuna kolayca geri dönüş yapılabilir. Veriler, bulut üzerinde tek bir kopya olarak saklanır ve internet bağlantısıyla tüm cihazlardan aynı anda erişilebilir, böylece veri ve uygulamaların cihazlar arasında kolayca paylaşılmasını sağlar.
- Bulut bilişim, "kullandıkça öde" modelini kullanır. Hizmet sağlayıcılar, altyapıya yatırım yapmadan ihtiyaçlarına göre kaynak kiralayabilir ve yalnızca kullandıkları kadar ödeme yapar.
- Bulut ortamında kaynaklar talebe göre hızlıca tahsis edilebilir ve serbest bırakılabilir. Bu sayede, hizmet sağlayıcılar talep düşük olduğunda kaynakları serbest bırakarak işletim maliyetlerinden büyük ölçüde tasarruf edebilir.
- merkezlerinde havuzlanan büyük kaynaklar kolayca erişilebilir hale getirilir. Hizmet sağlayıcılar, ani hizmet talep artışlarını (ör. flash-crowd etkisi) karşılamak için hizmetlerini hızlıca genişletebilir. Bu model "surge computing" olarak adlandırılır.
- Bulutta barındırılan hizmetler genellikle web tabanlıdır ve internet bağlantısı olan masaüstü, dizüstü, cep telefonu gibi çeşitli cihazlardan kolayca erişilebilir.
- Hizmet altyapısını bulutlara dış kaynak kullanarak devreden hizmet sağlayıcılar, donanım arızaları gibi iş risklerini altyapı sağlayıcılara aktarır. Bu, uzmanlaşmış altyapı sağlayıcılarının riski daha iyi yönetmesini sağlar ve aynı zamanda donanım bakımı ve personel eğitimi masraflarını azaltır.

3.3. Bulut Bilişimin Dezavantajları

Bulut bilişim, birçok avantajıyla birlikte popülerlik kazansa da, kullanıcıların dikkat etmesi gereken bazı endişeler ve dezavantajlar da barındırır. Güvenlik, maliyet, veri saklama ve uyumluluk gibi kritik alanlarda ortaya çıkan bu sorunlar, bulut hizmetlerini seçerken kullanıcıların bilinçli kararlar almasını gerektirir. Bu endişeler, birkaç başlık altında

incelenmiştir[45][46][47]:

Güvenlik endişeleri:

- Altyapının paylaşımlı kullanımından kaynaklanan mimari ve kaynak izolasyonu sorunları,
- Verinin üçüncü taraf sunucularda depolanmasından doğan güvenlik riskleri

Erişebilirlik endişeleri:

- Platformlar internet üzerinden sağlandığından sürekli internet bağlantısı gerekliliği,
- Bağlantı hızı ve kesinti problemleri,
- Hizmet kesintilerinde veri ve uygulamalara erişememe riski,
- Doğal afetler ve sunucu arızaları durumundan yaşanabilecek sorunları

Üçüncü taraf sorunları:

- Yasal ve düzenleyici gerekliklere uyum zorlukları,
- Tedarikçeye bağımlılık sorunları

Bunlara ek olarak, küçük ve orta işletmeler için daha fazla endişeler mevcuttur:

- Veri taşıma ve entegrasyon süreçliliğin karmaşıklığı,
- Uzun vadeli maliyet artışları riski

Bütün bulut bilişim sağlayıcıları, yukarıda belirtilen endişelere çözüm getirmeye çalışmıştır. Örneğin; AWS KMS (Key Management Service – Anahtar Yönetim Sistemi) ile bir çok kaynağı bulut içerisinde şifreleme imkanı sunmuştur[48]. Kesintiler için Amazon, Simple Storage Service (Standard için yıllık kesinti en fazla 8.76 saat olacaktır. Eğer kesinti belli değerler üzerinde olursa Amazon sağlayıcılara kredi vermektedir [49]. Kesintileri önlemek için, Amazon veri merkezleri yedekli ve fazla olarak kurmuştur [50]. Izolasyon açısından ise, AWS, fiziksel altyapı üzerinde çalışan sanal makineleri birbirinden izole etmek için özel bir teknoloji kullanır. Her müşterinin sanal makinesiayrı bir güvenlik bölgesi içinde çalışır ve diğer müşterilerin sanal makinelerinden tamamen izole edilmiştir [51].

3.4. Bulut Bilişim Çeşitleri

Günümüzde işletmeler, dijital dönüşüm süreçlerinde bulut bilişime yönelirken karşılarına çıkan en kritik kararlardan biri de uygun bulut modelinin seçimidir. Bulut bilişim modelleri,

organizasyonların verilerini ve uygulamalarını nerede ve nasıl barındıracaklarını belirleyen temel mimari yaklaşımlardır. Her işletmenin kendine özgü ihtiyaçları, güvenlik gereksinimleri, bütçe kısıtlamaları ve uyumluluk zorunlulukları göz önüne alındığında, doğru deployment modelinin seçimi kritik önem taşımaktadır.

Bulut bilişimde dört temel model bulunmaktadır. Her bir model, farklı iş ihtiyaçlarına ve güvenlik gereksinimlerine cevap verecek şekilde tasarlanmıştır.

Public Cloud (Genel Bulut), internet üzerinden herkesin erişimine açık olan ve çoklu kiracı (multi-tenancy) modeli ile çalışan bir bulut hizmet modelidir. Kullanıcılar bu modelde, servis sağlayıcının sunduğu altyapı, uygulama ve depolama gibi hizmetlere kimlik bilgilerini girerek her yerden erişebilir. Her Kullanıcınının verisi diğerlerinden izole edilmiş sanal ortamlarda tutulur ve bu sayede güvenlik sağlanırken, aynı zamanda kaynakların verimli kullanımı ve ölçeklenebilirlik de mümkün olur [52].

Private Cloud (Özel bulut), organizasyonların kendi veri merkezlerinde veya güvenilir bir üçüncü parti sağlayıcıda barındırılan, sadece tek bir kuruluşa hizmet veren bulut altyapısıdır. Özellikle finansal kurumlar, sağlık kuruluşları ve devlet kurumları gibi yüksek güvenlik ve veri gizliliği gerektiren sektörlerde tercih edilir. Geleneksel bulut hizmetlerine kıyasla daha fazla kontrol ve özelleştirme imkanı sunar [53].

Community Cloud (Topluluk Bulutu), ortak gereksinimlere sahip organizasyonlar tarafından paylaşılan bir bulut altyapısıdır. Bu yapı, topluluğun özel ihtiyaçlarına göre özelleştirilir ve genellikle topluluk içinden seçilen bir yönetici organizasyon tarafından işletilir [54].

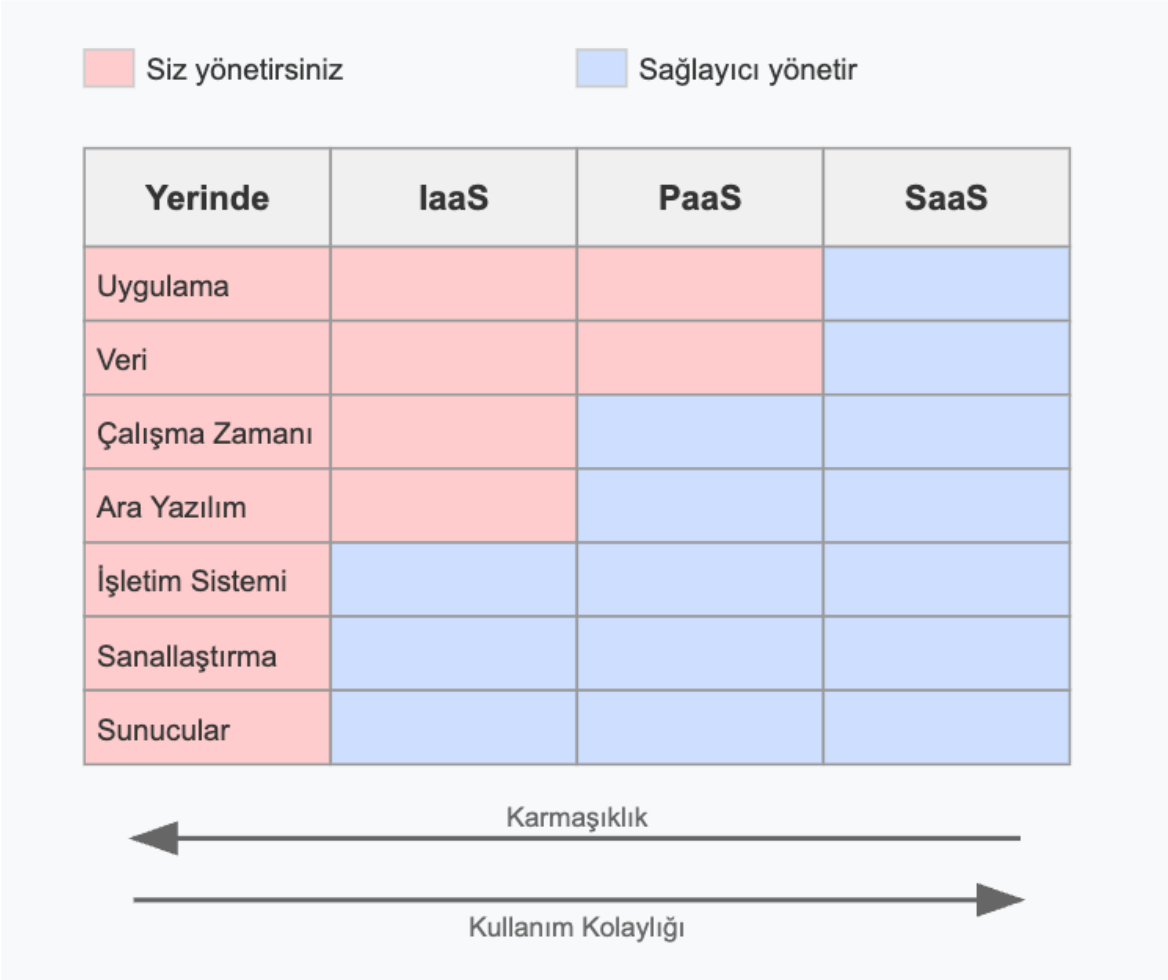
Hibrit Bulut (Hybrid Cloud), genel ve özel bulut modellerinin avantajlarını birleştiren ve her iki yaklaşımın kısıtlamalarını aşmaya çalışan karma bir bulut dağıtım modelidir. Bu yapıda, hizmet altyapısının bir kısmı özel bulutta çalışırken, diğer kısmı genel bulut üzerinde çalışır. Hibrit bulut, hem genel hem de özel buluta kıyasla daha fazla esneklik sunar [55].

3.5. Bulut Bilişim Modelleri

Bulut bilişim, günümüzde dijital dönüşümün kalbinde yer alarak, şirketlerin teknolojik ihtiyaçlarına hızlı, esnek ve maliyet etkin çözümler sunmaktadır. Fiziksel sunucu ve altyapı yönetiminin zorluklarını ortadan kaldıran bu teknoloji, kaynakların ihtiyaca göre dinamik olarak sağlanmasını mümkün kılar. Bulut hizmet modelleri, farklı ihtiyaçlara yönelik olarak

altyapıdan uygulama geliştirme platformlarınave tamamen sunucusuz çalıştırma çözümlerine (Serverless) kadar çeşitlilik gösterir.

Bu resim, bulut bilişim hizmet modellerinde kullanıcı ve sağlayıcı arasındaki yönetim sorumluluklarının nasıl dağıldığını göstermektedir [56].



Şekil 3.1 Bulut Bilişim Hizmet Modelleri

On-Premise (Yerinde) model, tüm bilişim altyapısının şirketin kendi veri merkezinde bulunduğu ve tamamen kullanıcı tarafından yönetildiği bir yapıdır. Sunucular, ağ cihazları, sanallaştırma, işletim sistemleri ve uygulamalar dahil tüm bileşenler kullanıcı kontrolündedir. Bu modelde tam kontrol sağlanırken, donanım bakımı, güvenlik, güncelleme ve ölçeklendirme gibi süreçler de kullanıcıya ek yük getirir.

IAAS (Infrastructure as a Service – Altyapı Olarak Servis) modeli, kullanıcıya temel bulut altyapısı üzerinde tam kontrol yetkisi sağlar. Bu altyapı; sanal sunucular, depolama, ağ

ve diğer temel bilişim kaynaklarını içerir. Kullanıcı, işletim sistemlerini, yazılımları ve uygulamaları kendisi kurar, yapılandırır ve yönetir [57]. Bu esneklik, özellikle özelleştirilebilir çözümler veya özel konfigürasyonlar gerektiren projeler için idealdir. Ancak, altyapının üzerinde çalışacak yazılımların ve sistem güncellemelerinin sorumluluğu kullanıcıya aittir. Bu durum, IAAS'yi geleneksel **On-Premise** yapısına kıyasla daha düşük maliyetli ve ölçeklenebilir bir alternatif haline getirirken, yönetim karmaşıklığını tamamen ortadan kaldırmaz.

PAAS (Platform as a Service – Hizmet Olarak Platform) modeli, kullanıcılara yazılım uygulamaları geliştirmek, test etmek ve dağıtmak için gerekli olan platformu sunar. Bu modelde altyapı, işletim sistemi, runtime ve yazılım geliştirme araçlarının yönetimi servis sağlayıcı tarafından yapılırken, kullanıcı yalnızca uygulama kodlaması ve veriler üzerinde çalışır [58]. PAAS, yazılım geliştirme süreçlerini hızlandırırken, altyapı yönetim yükünü ortadan kaldırır. Böylece geliştiriciler, kaynak ayarlama, bakım ve güncelleme gibi işlemler yerine doğrudan uygulama geliştirmeye odaklanabilir. Ancak bu model, Kullanıcınının platform üzerinde belirli bir yapı ve sınırlamalar dahilinde çalışmasını gerektirir.

SAAS (Software-as-a-Service – Hizmet Olarak Yazılım) modeli, kullanıcıya internet üzerinden erişilebilen, barındırılan yazılım uygulamaları sunar. Bu uygulamalar, bulut sağlayıcısı tarafından yönetilir ve güncel tutulur. SAAS, çoklu kullanıcı desteği (multitenancy) sağlar, talep üzerine kullanılabilir ve kolayca ölçeklenebilir. Kullanıcılar yalnızca uygulamayı kullanmakla sorumluyken, altyapı ve platform yönetimi tamamen sağlayıcı tarafından yapılır [59]. SAAS örnekleri arasında e-posta uygulamaları, ofis araçları, müşteri ilişkileri yönetimi yazılımları, faturalandırma sistemleri ve finansal yönetim uygulamaları bulunur. Bu tür yazılımlar genellikle işletmelerin operasyonel süreçlerini kolaylaştırmak için kullanılır.

3.6. Sunucusuz Modeller

Bulut bilişim teknolojilerinin gelişimiyle birlikte altyapı yönetimini kullanıcıdan tamamen kaldıran yeni modeller ortaya çıkmıştır. Bu modellerden biri olan **Serverless Computing (Sunucusuz Bilişim)**, modern uygulama geliştirme süreçlerinde büyük bir dönüşüm sağlamaktadır. Serverless, uygulamaların çalıştırılması için gerekli olan sunucu yönetimi, bakım ve yapılandırma gibi işlemleri tamamen bulut sağlayıcısına devreden bir hizmet modelidir[60]. Geleneksel sistemlerde geliştiriciler veya bilişim teknoloji ekipleri bu

süreçleri manuel olarak yürütmek zorundaydı. Bu durum, zaman kaybına, yüksek maliyetlere ve kaynakların doğru ölçeklenememesi gibi problemlere yol açıyordu. Sunucuz modeller, bu sorunları ortadan kaldırarak, geliştiricilerin sadece uygulama koduna odaklanmasını ve sistemin otomatik olarak ölçeklenmesini sağlar.

Sunucusuz bilişim, bulut uygulamalarının geliştirilmesi ve dağıtımını, altyapıdaki sunucular ve kaynaklar ile ilgilenmeye gerek kalmadan mümkün kılan bir bulut bilişim paradigmasıdır. Bu yaklaşımda operasyonel sorumluluklar bulut sağlayıcısına devredilir ve kullanıcılar yalnızca işlem süresi, bellek ve depolama alanı gibi kullanılan kaynaklar üzerinden ücretlendirilir. Temel özellikleri arasında sunucu yönetiminin ortadan kalkması, değer odaklı hizmet sunumu, otomatik ve sürekli ölçeklenebilirlik ile yerleşik hata toleransı yer almaktadır. Sunucusuz mimaride bir bulut işlevi, uygulamanın iş mantığını yerine getiren, tek bir işlevsel sorumluluğa sahip küçük, durumsuz ve isteğe bağlı bir hizmet olarak tanımlanmaktadır[61].

AWS, Microsoft Azure ve Google Cloud gibi büyük bulut sağlayıcıları kendi sunucusuz platformlarını sunar. AWS, yürütme süresi ve kullanılan kaynaklara göre otomatik olarak ölçeklenen ve ücretlendirilen olay odaklı bir işlem hizmeti olan AWS Lambda'yı sunar. Microsoft Azure, sunucuları yönetmeden olay odaklı kod yürütmeyi sağlayan, otomatik ölçekleme ve tüketime dayalı fiyatlandırma sunan Azure Functions'ı sunar. Google Cloud, talep üzerine ölçeklenen ve gerçek kullanım başına faturalandırılan, hizmetleri oluşturmak ve bağlamak için sunucusuz bir yürütme ortamı olan Cloud Functions'ı sunar[62].

Sunucusuz sistemlerin temel avantajları arasında hem işlem hem de veri genelinde kullanım başına ödeme ekonomisi (boş kapasite yerine işlev çağrısı başına ve veri hacmine göre ödeme), boştayken sıfıra ölçeklenebilen otomatik ölçekleme ve çoklu veri merkezi yedekliliği sayesinde kullanıma hazır yüksek kullanılabilirlik yer alır. Maliyet/ölçeğin ötesinde, sunucusuz sistemler kaynak bazında ayrıntılı kontrol (işlevler, kuyruklar, tablolar, kovalar) sağlayarak ekiplerin tek tip ayarlamalar yapmak yerine her bir parçayı maliyet, performans ve hatta sürdürülebilirlik açısından optimize edebilmesini sağlar. Güvenlik, belirli kayıtlara/özniteliklere daraltılabilen en az ayrıcalık ilkesiyle iyileştirilir. Etkinlik odaklı, gevşek bağlı hizmetler yinelemeli, artımlı geliştirmeyi kolaylaştırdığı için teslimat hızları artar[63].

4. UYGULAMALARIN CANLI ORTAMA DAĞITILMASI

Uygulamaların dağıtımı, yazılımın yalnızca geliştirme ortamında değil, gerçek kullanıcıların erişebildiği üretim ortamlarında da çalışabilir olmasını sağlamak için zorunludur. Ancak güvenilirlik, ölçeklenebilirlik ve sürekli erişilebilirlik gereksinimleri, uygulamaların farklı ortamlara yönelik yöntemlerle dağıtılmasını zorunlu kılmaktadır. Uygulama dağıtım yöntemleri, bilgi teknolojilerinin gelişimiyle paralel olarak önemli bir dönüşüm geçirmiştir. İlk dönemlerde kullanılan bare metal sunucular, donanıma doğrudan erişim sağlasa da sınırlı esneklik sunmaktaydı; sanal sunucu (virtualization) teknolojileri, kaynakların daha verimli kullanılmasına ve aynı fiziksel altyapı üzerinde birden fazla işletim sistemi çalıştırılmasına imkân tanımıştır. Daha sonra ortaya çıkan konteyner teknolojileri, hafif yapıları ve taşınabilirlik özellikleri sayesinde modern yazılım geliştirme süreçlerini hızlandırmıştır. Günümüzde ise Kubernetes gibi konteyner orkestrasyon araçları, ölçekleme, yüksek erişilebilirlik ve otomasyon kabiliyetleriyle uygulama dağıtımında endüstri standardı haline gelmiştir.

Bunun yanı sıra, tüm bileşenlerin sıkı şekilde bağlı olduğu ve tek bir birim olarak dağıtıldığı monolitik mimarilerden, uygulamaların daha küçük ve bağımsız dağıtılabilir servisler halinde parçalandığı mikroservis mimarilerine geçiş, esnekliği ve geliştirme hızını daha da artırmıştır. Monolitik yapılar başlangıçta basitlik sağlasa da zamanla ölçeklenmesi ve bakımı zorlaşırken, konteynerleştirme ile birlikte kullanılan mikroservisler, dağıtık ortamlarda çeviklik, dayanıklılık ve sorunsuz ölçekleme imkânı sunmaktadır.

4.1. Monolitik ve Mikroservis Uygulamalar

Monolitik uygulamalar, tüm yazılım bileşenlerinin tek bir bütün halinde çalıştığı geleneksel bir mimari yaklaşımı ifade etmektedir. Bu mimaride; bir yazılımı oluşturan bütün bileşenler - kimlik doğrulama, iş mantığı, veritabanı katmanı, sunum katmanı,işletim sistemi soyutlama katmanı- tek bir kod tabanı içerisinde yer almaktadır. İç iletişim genellikle mesajlaşma, kütüphaneler, fonksiyon çağrıları, paylaşılan bellek veya veri kanalları aracılığıyla sağlanır. Monolitik uygulamaların avantajları arasında daha kolay dağıtım, hızlı prototipleme ve uçtan uca test imkânı bulunmaktadır. Bununla birlikte, bu mimarinin bazı dezavantajları da vardır. Tüm bileşenlerin aynı anda ölçeklenmesi gerektiğinden kaynak kullanımı verimsizdir ve donanım eklemek saatler hatta günler alabilmektedir. Tek kod tabanının zamanla daha

karmaşık hale gelmesi, modüller arasında sıkı bağımlılıklar yaratmakta; bu durum yeni özelliklerin geliştirilmesini ve teknolojik yeniliklere uyum sağlanmasını zorlaştırmaktadır. Ayrıca, tüm uygulamanın başlatılması gerektiğinden başlangıç süreleri uzamakta ve küçük bir değişiklik dahi tüm sistemin yeniden derlenip dağıtılmasını zorunlu kılmaktadır [64].

Mikroservis mimarisi ise, büyük ve karmaşık sistemlerin küçük, bağımsız servisler (mikroservisler) koleksiyonuna ayrıldığı bir yazılım tasarım paradigmasını temsil etmektedir. Bu servislerin her biri, diğerlerinin işleyişini etkilemeden bağımsız olarak geliştirilebilir, dağıtılabilir, güncellenebilir ve ölçeklendirilebilir. Sonuç olarak, mikroservis tabanlı sistemler monolitik mimarilere kıyasla daha düşük bir bağlılık derecesine sahiptir ve değişen taleplere hızlı uyum sağlamanın kritik olduğu endüstriyel bağlamlar için son derece uygun bir yapı sunmaktadır [65].

Monolitik uygulamalar ile mikroservislerin karşılaştırılması sonucunda, mikroservislerin çeşitli açılardan üstün olduğu görülmektedir [66]:

- **Bakım Kolaylığı:** Bağımsız, kendi başına dağıtılabilen servisler, ekiplerin kodu ayrı ayrı test etmesine ve değiştirmesine imkân tanıyarak kodun anlaşılabilirliğini ve uzun vadeli sürdürülebilirliğini artırır.
- **Ölçeklenebilirlik:** Her mikroservis farklı makinelerde çalıştırılabilir ve bağımsız şekilde ölçeklendirilebilir, böylece darboğazlar tüm sistemin yeniden dağıtılmasına gerek kalmadan giderilebilir.
- **Yerine Koyulabilirlik:** Küçük boyutları sayesinde servislerin yeniden yazılması veya değiştirilmesi daha kolay ve maliyet-etkindir.
- **Hata Toleransı:** Tek bir mikroservisin başarısızlığı tüm sistemi etkilemez, hatalı servisler hızlıca yeniden başlatılabilir.
- **Güvenilirlik:** Temiz arayüzlere sahip küçük ve basit bileşenlerden oluşan büyük sistemler daha güvenilir ve yönetilebilir hale gelir.

Bu dönüşüme örnek olarak, akademik çalışmalarda incelenen **Danske Bank'ın FX (Foreign Exchange (Döviz piyasası; kur işlemleri)) Core (çekirdek) sistemi** verilebilir. Sistemin monolitik yapıdan mikroservislere geçişi, önemli ölçüde iyileştirmeler sağlamıştır. Öncelikle, sistemin ölçeklenebilirliği büyük oranda artmış ve daha önce tüm yapının bir

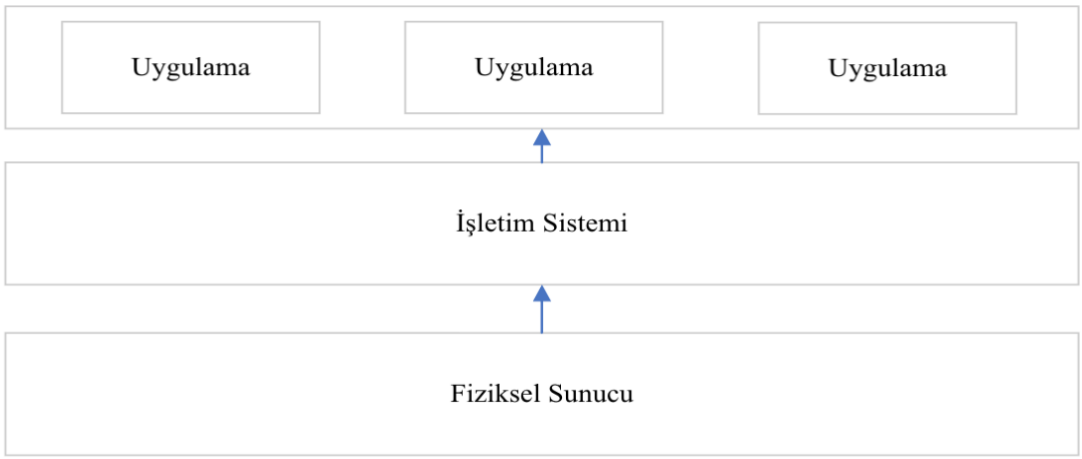
bütün olarak genişletilmesi gerekirken artık yalnızca gerekli parçaların ölçeklenmesi mümkün hale gelmiştir. Bu durum hem kaynak kullanımını optimize etmiş hem de yoğun dönemlerde sistemin performansını güvence altına almıştır. Ayrıca, bakım ve geliştirme süreçleri kolaylaşmış; mikroservislerin bağımsız çalışması sayesinde ekipler yeni özellikleri daha hızlı geliştirebilmiş ve değişiklikleri tüm sistemi etkilemeden uygulayabilmiştir. Bu dönüşüm, karmaşık ve yönetimi zor monolitik yapının yerine daha modüler ve esnek bir sistem getirerek sürdürülebilirliği güçlendirmiştir. Sonuç olarak FX Core sistemi daha sağlam, güvenilir ve uzun vadede genişlemeye uygun bir hale gelmiştir [67].

Bu mimari dönüşümün daha iyi anlaşılabilmesi için, öncelikle bu sistemlerin üzerinde çalıştığı altyapılara bakmak gerekmektedir; ilk olarak **bare metal sunucular (fiziksel sunucular)** incelenmelidir.

4.2. Bare Metal (Fiziksel) Sunucular

Fiziksel sunucu (bare metal server), kaynakları sanallaştırılmamış ve birden fazla kullanıcı arasında paylaşılmamış, tamamen adanmış bir makineyi ifade etmektedir. Bu nedenle, bu tür bir sunucuda yürütülen iş yükleri, altta yatan donanım bileşenlerinin yerel performansından tam anlamıyla yararlanabilmektedir [68].

Bu modelde bir fiziksel sunucu, yalnızca tek bir işletim sistemi çalıştırmakla sınırlıdır ve dolayısıyla yalnızca o işletim sistemi için tasarlanmış uygulamalar çalıştırılabilmektedir. Bu yaklaşımın en önemli avantajı, işletim sisteminin tüm hesaplama kaynakları üzerinde tam kontrol sağlamasıdır. Bu durum, ek yükü azaltarak özellikle yüksek performans gerektiren görevler için etkili hale getirmektedir. Ancak, düşük hesaplama talebi söz konusu olduğunda kaynakların önemli bir kısmı atıl kalmakta, bu da verimsizliğe yol açmaktadır. Yeni hizmetler eklendikçe gerekli fiziksel sunucu sayısı hızla artmakta, bu da yönetimi zorlaştırmakta ve maliyetleri yükseltmektedir. Ayrıca, yeni bir sunucunun kurulumu zaman alıcıdır; işletim sisteminin yüklenmesi, ağ ve sürücülerin yapılandırılması, uygulamaların dağıtılması gibi süreçler çoğu zaman günler hatta haftalar sürebilmektedir. Bu sınırlılıklar, kaynak kullanımını optimize edecek ve yönetimi kolaylaştıracak daha esnek teknolojilere duyulan ihtiyacı ortaya koymaktadır. Şekil 4.1, fiziksel sunucuların görselleştirilmiş halini göstermektedir [69].



Şekil 4.1 Fiziksel Sunucuların Yapısı

4.3. Sanallaştırma

Sanallaştırma, ilk kez 1960'larda IBM (International Business Machines Corporation) ana bilgisayarları tarafından ortaya konulan, aynı fiziksel donanım üzerinde birden fazla sanal makinenin eşzamanlı olarak çalıştırılmasını mümkün kılan ve farklı işletim sistemi ortamlarının tek bir platform üzerinde bir arada bulunmasını sağlayan bir bilgi işlem teknolojisidir. Bu teknoloji, kaynak kullanımını optimize etmek ve erişilebilirliği sağlamak amacıyla geleneksel olarak kurumsal bilgi işlem ve bulut bilişim bağlamlarında yaygın olarak kullanılmıştır [70][71].

Sanallaştırma, **donanım düzeyi** ve **yazılım düzeyi** olmak üzere iki yaklaşıma ayrılmaktadır. Donanım düzeyinde sanallaştırma, bare-metal sanallaştırma gibi yöntemlerle fiziksel kaynakların doğrudan tahsis edilmesini sağlarken; yazılım düzeyi sanallaştırma, uygulamaları işletim sistemi tarafından yönetilen izole süreçler olarak soyutlamaktadır. Bu iki yaklaşımın entegrasyonu, özellikle sistem güvenilirliğini ve performansını artırmada uygun, daha esnek ve verimli bir çalışma ortamı sunmaktadır [72].

Sanallaştırmanın üç temel özelliği bulunmaktadır [73]:

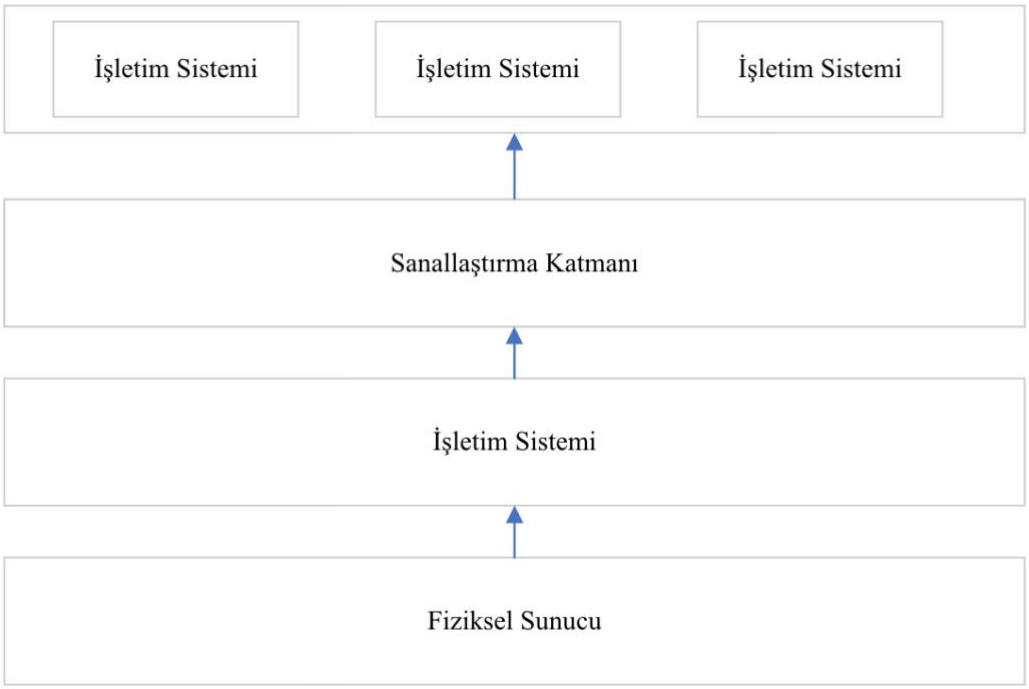
- **Bölümlendirme (partitioning):** Birden fazla işletim sistemi ve uygulamanın aynı fiziksel sistem üzerinde çalıştırılmasına imkân tanır.

- **Yalıtım (isolation):** Bir sanal makinedeki hata veya verinin diğerlerini etkilemesini engeller.
- **Kapsülleme (encapsulation):** Bir sanal makinenin tek bir taşınabilir dosya olarak temsil edilmesini sağlar.

Bu özellikler, belleğe, ağlara, depolamaya, donanıma, işletim sistemlerine ve uygulamalara uygulanabilmekte olup, yazılım ile donanım arasındaki bağı koparak bulut bilişimin temelini oluşturmaktadır. Sanallaştırmanın başlıca faydaları ise şu şekilde özetlenebilir [74][75][76]:

- **Sunucu konsolidasyonu ve maliyet tasarrufu:** Gerekli fiziksel sunucu sayısını azaltarak donanım, lisans, enerji ve soğutma giderlerini düşürür.
- **Kaynakların verimli kullanımı:** Birden fazla işletim sistemi ve uygulamanın aynı fiziksel makineyi paylaşmasına imkân tanıyarak genel kaynak kullanımını artırır.
- **Yüksek erişilebilirlik ve felaket kurtarma:** Canlı göç (live migration), yedekleme ve sanal makinelerin kesinti olmaksızın geri yüklenmesini destekleyerek iş sürekliliğini sağlar.
- **Ölçeklenebilirlik ve esneklik:** Değişen iş yükü taleplerine yanıt verebilmek için CPU (Central Processing Unit / Merkezi İşlem Birimi) , bellek ve depolama gibi kaynakların kolayca ayarlanmasına olanak tanır.
- **Basitleştirilmiş yönetim:** Merkezi yönetim, şablonlardan otomatik dağıtım ve BT (Bilişim Teknolojisi) altyapısının daha kolay bakımını sağlar.
- **Geliştirilmiş güvenlik ve izolasyon:** Her sanal makine izole çalıştığından, birindeki hata veya saldırı diğerlerini etkilemez.

Çalışmalarda en çok kullanılan yaklaşım ise **yazılım düzeyi sanallaştırma**dır [77].



Şekil 4.2 Yazılım Düzeyinde Sanallaştırma

Şekil 4.2'de yazılım düzeyi sanallaştırma mimarisi gösterilmektedir. Bu mimaride fiziksel sunucu üzerinde birincil işletim sistemi barındırılmakta, üzerine kurulan hipervizör yazılımı donanım kaynaklarını soyutlamakta ve her biri kendi işletim sistemini çalıştıran birden fazla sanal makinenin eşzamanlı olarak işletilmesine imkân vermektedir. Bu katmanlı yapı, tek bir fiziksel altyapı içerisinde izolasyon, esneklik ve verimli kaynak kullanımını sağlamaktadır.

4.4. Container (Konteynır) Teknolojisi

Konteyner, işletim sistemi düzeyinde çalışan hafif bir sanallaştırma teknolojisidir. Sanal makinelerden farklı olarak, çekirdek de dâhil olmak üzere tüm işletim sistemini sanallaştırmaz; bunun yerine, ana işletim sistemi çekirdeği hem ana sistem hem de tüm konteynerler arasında paylaşılır. Konteynerler esas olarak süreçleri, dosyaları ve kaynak erişimlerini yalıtarak tam bir misafir işletim sistemine ihtiyaç duymadan bağımsız bir çalışma ortamı sağlar [78].

Docker, uygulamaların oluşturulmasını, dağıtılmasını ve yönetilmesini kolaylaştıran, önde gelen açık kaynaklı konteynerleştirme platformudur. Yazılım geliştirme süreçlerini

hızlandırmakta ve geliştiricilere araç, yığın (stack) ve ortam seçiminde esneklik sağlamaktadır [79]. Konteyner teknolojisi, sürekli dağıtım ve uygulamaların kesintisiz çalıştırılmasında kritik bir rol oynamakta; mikroservis mimarilerini, bulut hizmetlerini ve benzeri senaryoları desteklemektedir [80].

Konteynerler, işletim sistemi düzeyinde sanallaştırma yoluyla çalışır. Her konteyner kendi dosya sistemine, kullanıcı alanına ve kütüphanelerine sahip olup bağımsız bir işletim sistemi örneği soyutlaması sunar. Bu yapı esas olarak Linux çekirdeğinin iki temel özelliğine dayanmaktadır [81]:

- **Ad alanları (Namespaces):** Konteyner içindeki süreçlerin ana sistemde neleri görebileceğini sınırlayarak izolasyon sağlar.
- **Kontrol Grupları (Control Groups-cgroups):** Konteyner içindeki süreçlerin CPU (Central Processing Unit – Merkezi İşlem Birimi), bellek gibi kaynak kullanımını sınırlar ve düzenler.

Konteyner teknolojisinin birçok avantajı bulunmaktadır [82][83]:

- **Hafif ve Hızlı:** Tam bir işletim sistemi yükünü ortadan kaldırdığı için daha küçük boyutlu ve daha hızlı başlatılabilir.
- **Kaynak Verimliliği:** Sanal makinelere kıyasla aynı donanım üzerinde daha fazla uygulama çalıştırılabilir, maliyet etkinliği artar.
- **Hızlı Dağıtım:** Konteynerler saniyeler içinde dağıtılıp yeniden başlatılabilir, bu da çevik geliştirme ve CI/CD süreçlerini destekler.
- **Tutarlılık:** Geliştirme, test ve üretim ortamlarında aynı çalışma koşullarını sunarak tekrarlanabilirliği garanti eder.
- **Taşınabilirlik:** Docker, uygulamaları bağımlılıklarıyla birlikte paketleyerek farklı platformlar ve bulutlar arasında taşınabilir hale getirir.
- **Kaynak Kontrolü:** cgroups aracılığıyla CPU, bellek ve ağ kaynaklarının adil bir şekilde tahsis edilmesini sağlar.
- **Mikroservis Desteği:** Her mikroservisin birbirinden bağımsız çalışmasına imkân tanır.

Araştırmalarda, konteynerlerin sanal makinelerle karşılaştırmaları da yapılmıştır. Örneğin, yapılan bir çalışmada konteynerlerin hafif, taşınabilir ve enerji verimli bir sanallaştırma sunduğu ve bunun IoT ile uç bilişim senaryolarında avantaj sağladığı gösterilmiştir [84]. Başka bir çalışmada ise hipervizör tabanlı sanallaştırma ile konteyner çözümleri karşılaştırılmış ve konteynerlerin CPU, bellek, disk ve ağ performansında neredeyse donanım seviyesinde verim sağladığı, hipervizörlerin ise özellikle diske daha yüksek ek yük getirdiği, ancak daha güçlü izolasyon ve geniş işletim sistemi uyumluluğu sunduğu ortaya konmuştur [85].

Docker, istemci–sunucu mimarisi üzerine kuruludur. Docker'ın temel bileşenleri şu şekilde tanımlanmaktadır [86]:

- **Image (Imaj):** Uygulama ve bağımlılıklarını içeren salt-okunur şablon; konteynerler bu imajlardan üretilir.
- **Container (Konteynır):** Bir imajdan başlatılan, izole ve taşınabilir çalışma ortamı.
- **Dockerfile:** Docker imajlarını oluşturmak için kullanılan yapılandırma dosyası.
- **Registry/Repository:** Docker Hub gibi imajların depolandığı ve dağıtıldığı sistem.
- **Volumes:** Konteyner yaşam döngüsünün ötesinde veri kalıcılığı sağlar.
- **Networks (Ağ):** Konteynerlerin ve dış sistemlerin iletişimini sağlar.
- **Compose/Stacks:** Çoklu konteyner uygulamalarını tek bir servis olarak tanımlama ve yönetme araçları.
- **Docker Daemon:** Arka planda çalışan, imajları, konteynerleri, ağları ve volume'leri yöneten servis. Docker API aracılığıyla istemciden gelen talepleri dinler ve uygular.
- **Docker Client:** Docker Daemon ile etkileşimi sağlayan komut satırı arayüzü. Geliştiriciler ve yöneticiler için birincil araçtır.
- **Docker Registry:** Docker imajlarının merkezi depolama ve dağıtım servisi.
- **Docker Engine:** Docker Daemon, Docker Client ve API'lerden oluşan, konteynerlerin çalıştırılması ve yönetilmesi için gerekli tam çalışma ortamı.

4.5. Kubernetes

Geleneksel dağıtım yöntemlerinde, uygulamaların ayrı ayrı fiziksel makinelerde barındırılması gerekmekteydi. Bu yaklaşım, kaynak kullanımında verimsizliğe ve

ölçeklenebilirliğin sınırlı kalmasına yol açmaktaydı. Sanal makineler, tek bir ana bilgisayar üzerinde birden fazla işletim sisteminin çalıştırılabilmesini mümkün kılarak bu sorunu kısmen çözmüş olsa da, her bir misafir işletim sisteminin tam yükü, ek maliyet yaratarak verimsizliklere neden olmaya devam etmiştir. Konteynerler ise hafif, işletim sistemi düzeyinde sanallaştırma yaklaşımı ile uygulamaların aynı çekirdeği paylaşırken birbirinden izole kalmasını ve kaynakların daha verimli kullanılmasını sağlamıştır. Ancak, çok sayıda konteynerin manuel olarak yönetimi karmaşık ve hata yapmaya açık bir süreçtir. **Kubernetes** bu sorunu çözerek otomatik orkestrasyon sağlamış, birleşik yönetim, esnek ölçeklendirme, kesintisiz güncellemeler ve güvenilir yük dengeleme kabiliyetleri ile modern bulut-yerel altyapıların temel platformu haline gelmiştir [87].

Konteyner orkestrasyonu, konteyner tabanlı uygulamaların dağıtımı, ölçeklendirilmesi, farklı ana makineler arasında taşınması, kaynak tahsisi ve yük dengelemesini kapsayan otomatik yönetim sürecini ifade etmektedir. Yüzlerce konteynerin manuel olarak yönetilmesi verimsiz ve hata yapmaya yatkın olacağından, orkestrasyon bu operasyonel karmaşıklığı ortadan kaldırmakta; basitleştirilmiş operasyonlar, dayanıklılık ve artan güvenlik sağlamaktadır. Böylelikle, yazılım geliştirme ve dağıtım süreçlerinin genel karmaşıklığı azalmaktadır [88].

Kubernetes, açık kaynaklı bir orkestrasyon platformu olup konteyner tabanlı uygulamaların dağıtımını, ölçeklenmesini ve yönetimini otomatikleştirmektedir. Geliştiricilerin altyapı ayrıntılarıyla uğraşmadan uygulama geliştirmesine olanak tanımakta ve deklaratif yaklaşımı sayesinde hedeflenen durumun korunmasını sağlamaktadır. Ayrıca, kendini iyileştirme (self-healing), hata tespiti ve kurtarma mekanizmaları ile güvenilirlik sunmakta; bu yönüyle özellikle mikroservis tabanlı web uygulamaları için uygun bir çözüm oluşturmaktadır. Konteynerlerin hafif yapısı ve hızlı oluşturulup silinebilmeleri, sanal makinelere kıyasla kaynakların daha etkin kullanılmasına olanak tanımaktadır. Bu özellikleriyle Kubernetes, modern bulut altyapılarında karmaşık uygulamaların yönetimini kolaylaştıran güçlü ve esnek bir çözüm olarak öne çıkmaktadır [89].

Günümüzde büyük bulut sağlayıcıları Kubernetes'i yönetilen hizmet olarak sunmaktadır. Google Cloud Platform, **Google Kubernetes Engine (GKE)** aracılığıyla konteyner orkestrasyonu sağlarken; Amazon Web Services, **Elastic Kubernetes Service,**Oracle Cloud

Infrastructure, **Oracle Kubernetes Engine** ile dağıtım ve yönetimi basitleştirmekte; IBM Cloud ise **IBM Kubernetes Service** ve Knative ile sunucusuz iş yüklerini dönüştürmektedir. Microsoft Azure, **Azure Kubernetes Service** ile Kubernetes'i bulut ekosistemine entegre etmektedir. Bu hizmetlerin ortak noktası, geliştiriciler için altyapı yönetimini azaltmak ve Kubernetes'in otomatik olarak güncellenebilir ve dağıtılabilir özelliklerine doğrudan erişim sağlamaktır. Sağlayıcılar farklı sanallaştırma teknolojileri, sistem modelleri ve yatay dağıtım yöntemleri kullansa da, Kubernetes genellikle konteyner tabanlı bulut mimarilerinin temel taşı olarak konumlandırılmaktadır [90].

Kubernetes mimarisi, iki temel bileşenden oluşmaktadır: **Kontrol Düzlemi (Control Plane)** ve **İşçi Düğümü (Worker Node)** [91].

- **Kontrol Düzlemi,** kümenin "beyni" olarak işlev görmektedir. Dört ana bileşen içerir:
  - kube-apiserver*:* Küme ile kullanıcıların ve bileşenlerin etkileşim kurmasını sağlayan merkezi API arayüzü.
  - etcd*:* Küme yapılandırmalarını ve durumunu saklayan dağıtık anahtar-değer veritabanı.
  - kube-controller-manager*:* Kümenin mevcut durumunu izleyen ve bunu hedeflenen durum ile uyumlu hale getirmek için gerekli işlemleri gerçekleştiren bileşen.
  - kube-scheduler*:* Pod'ların hangi işçi düğümlerinde çalışacağını belirleyen bileşen.
- **İşçi Düğümü**, uygulamaların fiilen çalıştığı makinelerden oluşur. Temel bileşenleri şunlardır:
  - kubelet*:* Her düğümde çalışan ajan olup pod içindeki konteynerlerin çalışmasını garanti eder.
  - kube-proxy*:* Düğümün ağ trafiğini yönetir ve servisler arası iletişimi sağlar.
  - Pod'lar*:* Kubernetes'in en küçük yürütme birimi olup bir veya daha fazla konteyner içerebilir.

Kubernetes'in temel çalışma modeli deklaratif yaklaşıma dayanmaktadır. Bu modelin merkezinde **istenen durum (desired state)** kavramı yer almaktadır. İstenen durum, YAML (YAML Ain't Markup Language-YAML İşaretleme Dili Değildir) manifest dosyaları ile

tanımlanmakta; kullanılacak konteyner imajı, kaç adet kopya (replica) olması gerektiği veya hangi portların açılacağı gibi ayrıntılar bu dosyalarda belirtilmektedir [92].

Kubernetes'te tanımlanan temel nesneler şunlardır [93]:

- **Deployment:** Pod'ları yönetir, hedeflenen kopya sayısını korur, güncellemeler ve geri almaları kolaylaştırır.
- **Service:** Pod'lar için sabit bir erişim noktası sağlar, yük dengeleme işlevi görür ve dış dünyaya erişimi mümkün kılar. Türleri arasında ClusterIP, NodePort ve LoadBalancer yer almaktadır.
- **Namespace:** Kaynakların mantıksal olarak ayrılmasını ve çok kullanıcılı/ekipli ortamlarda izole yönetimi sağlar.
- **ConfigMap & Secret:** Yapılandırma verilerini (ConfigMap) ve parola gibi hassas bilgileri (Secret) güvenli şekilde saklar ve pod'lara ortam değişkeni ya da hacim (volume) olarak aktarır.
- **PersistentVolume & PersistentVolumeClaim:** Pod'ların kalıcı veri depolamasını sağlar, depolama altyapısı ile uygulamaları birbirinden ayırır.

4.6. GitOps

GitOps, Kubernetes ile birlikte ortaya çıkan bir paradigma olarak, DevOps uygulamalarını genişleterek hem uygulama hem de altyapı yapılandırmalarında Git'i tek doğruluk kaynağı (single source of truth) olarak kullanmaktadır. GitOps, sistemin hedeflenen durumunun (desired state) sürüm kontrollü depolarda tanımlanmasını ve bu durumun çalışma ortamındaki mevcut durum ile istenen durumun eşitlenmesi esas alan **deklaratif bir yaklaşımı** vurgulamaktadır [94][95][96]. Bu bağlamda, **ArgoCD** ve **Flux** gibi temel araçlar, otomatik çekme (pull-based) mekanizmaları aracılığıyla dağıtımların Git'te tanımlanmış spesifikasyonlarla uyumlu hale getirilmesini sağlamakta ve üretim ortamına doğrudan erişim ihtiyacını ortadan kaldırmaktadır [97][98]. Bu yaklaşım, altyapı ve uygulama değişikliklerinin izlenebilirliğini, tekrarlanabilirliğini ve denetlenebilirliğini kolaylaştırarak uyumluluk ve yönetişim süreçlerine katkıda bulunmaktadır [96].

GitOps, geleneksel DevOps uygulamalarına kıyasla çeşitli operasyonel faydalar sağlamaktadır. Bunlar arasında daha hızlı ve güvenilir dağıtımlar, Git geçmişi üzerinden kolaylaştırılmış geri alma (rollback) işlemleri ve azaltılmış manuel müdahale öne çıkmaktadır [95][98][96]. Tüm değişikliklerin sürüm kontrollü olması sayesinde, ekipler altyapı ya da uygulama durumlarını önceki bir Git commit'ine dönerek hızlıca denetleyebilmekte ve geri alabilmektedir [85][86]. Dağıtım mantığının çalışma ortamından ayrıştırılması sistemin dayanıklılığını artırmakta ve dağıtık ekipler arasında işbirliğini kolaylaştırmaktadır [99][96]. Ayrıca, GitOps'un Kubernetes ile uyumluluğu, mikroservis tabanlı mimarilerin yönetiminde gözlemlenebilirliği, güvenliği ve ölçeklenebilirliği güçlendirmektedir [94][100].

Faydalarına rağmen GitOps, bazı sınırlılıklara sahiptir. Kubernetes ve Git merkezli araçlara bağımlılığı, GitOps'un kaynak kısıtlı veya heterojen ortamlarda, örneğin IoT Edge dağıtımlarında uygulanabilirliğini sınırlayabilmektedir [99][101]. Ayrıca GitOps, deklaratif yapılandırmalar üzerinde katı kısıtlamalar getirmekte, bu durum ise emirsel (imperative) iş akışlarına alışkın ekiplerin zihniyet ve araç setinde değişiklik gerektirmektedir [84][89]. Bununla birlikte, gizli bilgilerin (secrets) yönetimi, büyük ölçekli Git depoları ve karmaşık bağımlılık zincirlerinin idaresi, operasyonel ek yük ve performans sorunları doğurabilmektedir [98][96].

GitOps genellikle DevOps'un bir evrimi olarak görülmekle birlikte, aslında onun bir tamamlayıcısı olarak değerlendirilmelidir [84]. DevOps; geliştirme ve operasyon ekiplerinin kültürel ve süreçsel entegrasyonuna, otomasyona ve sürekli geri bildirime odaklanırken [95][96], GitOps bu ilkeleri **deklaratif altyapı** ve **çekme tabanlı otomasyon** ile operasyonelleştirmektedir. Böylelikle insan hatası riski azaltılmakta ve dağıtımların tutarlılığı artırılmaktadır [85][86]. GitOps, kod olarak altyapı uygulamaları ile operasyonel durum yönetimi arasındaki boşluğu daraltarak DevOps'un üzerine inşa edilmektedir [96].

GitOps, özellikle Kubernetes ile konteyner tabanlı mikroservislerin yönetildiği bulut-yerel ortamlarda önemli avantajlar sunmaktadır [94][97][98]. GitOps, endüstri ve akademide yapılandırma uyumsuzluğu (configuration drift) yönetimi, CI/CD otomasyonu ve güvenli çoklu ortam dağıtımları gibi senaryolarda başarıyla uygulanmıştır [94][97][100]. Telekomünikasyon ve imalat gibi sektörler de GitOps'u, sırasıyla ağ işlevi yapılandırmaları

ve siber-fiziksel sistem orkestrasyonunda kullanarak, yaklaşımın geleneksel web uygulamalarının ötesindeki uyarlanabilirliğini göstermiştir [100][101].

GitOps, sürekli dağıtım ve altyapı otomasyonunun evriminde önemli bir adımı temsil etmektedir. Sürüm kontrolü, deklaratif yapılandırma ve otomatik uzlaştırma ilkelerini birleştirerek modern yazılım sistemlerinin yönetimi için güçlü bir çerçeve sunmaktadır [95][96]. Gelecekteki araştırmalar, GitOps uygulamalarının standartlaştırılması, çoklu küme (multi-cluster) ve çoklu bulut (multi-cloud) desteğinin geliştirilmesi ve IoT Edge veya gömülü sistemler gibi kısıtlı ortamlara uyarlamalar üzerine yoğunlaşabilir [99][101].

4.7. Kod Olarak Altyapı (Infrastructure as a Code)

IaC normalde bir kullanıcı tarafından manuel olarak gerçekleştirilmesi gereken pek çok dağıtım, yapılandırma ve yönetim görevini otomatikleştirmeyi mümkün kılan bir yaklaşımdır [102]. IaC, altyapının kodlanabilir betikler aracılığıyla tanımlanmasını ve yönetilmesini sağlayarak, deneysel ve üretim ortamlarında tutarlı ve tekrarlanabilir dağıtımlar sunmaktadır [103]. Bu uygulama, ağ ve sistem yapılandırmalarının otomatik olarak tanımlanıp yönetilmesini içerir ve altyapıyı bir yazılım varlığı olarak ele almaktadır [104].

Terraform ve Ansible gibi **IaC araçları**, altyapı yapılandırmalarının kod şeklinde tanımlanmasına ve yönetilmesine imkân tanımakta, bu sayede dağıtımların tutarlı, tekrarlanabilir ve sürüm kontrollü olmasını garanti etmektedir [105]. İstemcilere özgü YAML veya JSON (JavaScript Object Notation - JavaScript Nesne Gösterimi) betikleri ile istenen durum tanımlanabilmekte, bu sayede her dağıtım, ortamın mevcut durumundan bağımsız olarak aynı sonucu üretmektedir. Bu süreç **idempotentlik** ilkesi üzerine inşa edilmiştir. Ayrıca, IaC betikleri test, versiyonlama ve sürekli entegrasyon süreçleriyle birleştirilerek güvenilir ve izlenebilir dağıtımların elde edilmesine katkı sağlamaktadır [103].

Modern IaC çözümleri, mevcut altyapıyı hedeflenen durumla karşılaştırmakta ve bu duruma ulaşmak için gerekli adımları otomatik olarak belirlemektedir. Örneğin, Terraform gibi araçlarla istenen durum genel amaçlı programlama dilleri (TypeScript, Python vb.) kullanılarak tanımlanabilmektedir [106].

IaC'nin pek çok avantajına rağmen, kaliteye ilişkin çeşitli zorluklar barındırdığı da bilinmektedir. Diğer yazılım varlıklarında olduğu gibi, Terraform dosyaları, yanlış yapılandırmalar ve hatalardan kaynaklanan sorunlara açık olup, bu durum güvenilirlik ve sürdürülebilirliği olumsuz etkileyebilir. Geleneksel yazılım kodlarında olduğu gibi IaC betikleri de ölçek büyüdükçe hatalara yatkındır ve bu hatalar ciddi sorunlara yol açabilmektedir. Ayrıca, sürdürülebilirlik açısından bulut kaynaklarının gereksiz aşırı tahsisi, uygun olmayan sunucuların seçilmesi, verimsiz iş yükü dağılımı, gereksiz ağ trafiği veya enerji tasarrufu sağlayan özelliklerin göz ardı edilmesi gibi durumlar, altyapıda kaynak israfına yol açabilmektedir.Terraform özelinde bakıldığında ise yanlış yapılandırmalar, teknik borç ve güvenlik açıkları, pratik benimsemeyi zorlaştırmakta ve altyapı kesintilerine sebep olabilmektedir [104][107].

IaC'nin uygulanmasında güvenlik, kritik bir unsur olarak öne çıkmaktadır. IaC süreçlerinde ortaya çıkabilecek tehditlerin sistematik bir şekilde listelenmesi henüz mümkün olmasa da, bu durum **DevSecOps** ekiplerinin erken aşamalardan itibaren birlikte çalışmasını zorunlu kılmaktadır. Bu işbirliği, risk analizi, risk tahmini ve risk azaltma süreçlerinin etkin şekilde yürütülmesi için gereklidir [103].

IaC'nin güvenlik açıkları, saldırganların hem fiziksel hem de sanal kaynaklara erişim sağlamasına imkân verebilir. Bu nedenle, IaC ile ilgili güvenlik kusurlarının incelenmesi ve analiz edilmesi acil bir gereksinimdir [103].Örnek olarak varsayılan kullanıcı hesabı kullanımı, boş şifreler, şüpheli yorum satırları, sabitlenmiş gizli anahtarlar, TLS (Transport Layer Security-Taşıma Katmanı Güvenliği) olmadan HTTP (Hypertext Transfer Protocol-Hiper Metin Aktarım Protokolü) kullanımı, geçersiz IP (Internet Protocol Address - Internet Protokolü Adresi) adresi atamaları ve zayıf kriptografi algoritmalarının tercih edilmesi [108].

IaC'nin pratik uygulamaları, çeşitli endüstri senaryolarında ortaya konmuştur. Bulut-yerel teknolojiler, ölçeklenebilir, güvenli ve dayanıklı uygulamaların geliştirilmesine olanak tanırken, IaC bu altyapı bileşenlerinin dağıtılmasında temel bir bileşen olarak kullanılmaktadır [105]. Eğitim alanında ise IaC, yazılım laboratuvar sınavlarının otomatik olarak teslim edilmesinde kullanılmış; yüksek erişilebilirlik ve ölçeklenebilirlik sağlayan bir bulut tabanlı mimari, modern bulut teknolojileri ve DevOps ilkeleriyle desteklenmiştir [109].

Buna ek olarak, tıbbi görüntüleme analitiği uygulamalarında da IaC çerçeveleri başarıyla kullanılmış, belirlenen süreler içinde gerekli altyapılar oluşturularak yöntemin alan spesifik uygulamalarda da etkin olduğu gösterilmiştir [110].

5. GÜNÜMÜZDE VERİ VE YAPAY ZEKA UYGULAMALARI

Günümüz dijital çağında veri, yenilikçiliğin, karar alma süreçlerinin ve rekabet avantajının temel dayanaklarından biri haline gelmiştir. Kuruluşlar, yalnızca geçmiş davranışları anlamak için değil; aynı zamanda gelecekteki eğilimleri tahmin etmek, süreçleri otomatikleştirmek ve kişiselleştirilmiş deneyimler sunmak için giderek daha fazla veri odaklı yaklaşımlara yönelmektedir. Veri hacminin ve hızının hızlı artışı, toplu (batch) analizler ile gerçek zamanlı zekâyı bütünleştirebilen güçlü otomasyonların gerekliliğini ortaya koymaktadır.

Bu gereksinimleri karşılamak amacıyla AWS Glue kullanılarak bir **ETL hattı** tasarlanmıştır. AWS Glue, verilerin otomatik olarak çıkarılmasını, dönüştürülmesini ve yüklenmesini (Extract, Transform, Load) sağlamaktadır. Böylece ham ve heterojen veri kaynakları, ileri düzey analizler için uygun, yapılandırılmış, tutarlı ve sorgulanabilir formatlara dönüştürülmektedir. Buna paralel olarak, **Amazon Kinesis** gerçek zamanlı veri işleme için kullanılmış, sistemin sürekli akan veri akışlarını düşük gecikme ile alıp analiz etmesine imkân tanımıştır. Toplu ve akış verilerinin bu bütünleşmesi, hem tarihsel hem de gerçek zamanlı içgörüler için birleşik bir temel sunmaktadır.

Bu veri altyapısının üzerine gelişmiş yapay zekâ uygulamaları inşa edilmiştir. **Amazon Personalize** kullanılarak bir öneri sistemi geliştirilmiş, kullanıcı etkileşim geçmişlerinden yararlanılarak uyarlanabilir ve kişiselleştirilmiş içerik önerileri üretilmiş ve kullanıcı etkileşimi artırılmıştır. Ayrıca, **Amazon Bedrock** ile bilgi tabanlı bir sohbet robotu geliştirilmiş, yapılandırılmış bilgi tabanının entegrasyonu sayesinde bağlama duyarlı ve akıllı etkileşim yetenekleri sağlanmıştır.

Bu bileşenler bir araya geldiğinde, veri ile yapay zekânın yakınsamasının stratejik önemini ortaya koymaktadır. Ölçeklenebilir ETL süreçleri, gerçek zamanlı analitik, kişiselleştirilmiş öneriler ve yapay zekâ destekli sohbet sistemlerinin entegrasyonu sayesinde, kuruluşların ham verileri yenilik, verimlilik ve kullanıcı odaklı çözümleri doğrudan destekleyen **eyleme dönüştürülebilir zekâya** çevirebildikleri gösterilmiştir.

5.1. DataOps

DataOps, veri yönetim süreçlerini kolaylaştırmak, güvenilir hale getirmek ve veri değerini en üst düzeye çıkarmak için geliştirilmiş bir metodolojidir. Kavram, yazılım geliştirme dünyasında yaygın olan **DevOps** yaklaşımının, veri analitiği alanına uyarlanmış biçimi olarak tanımlanmaktadır [111]. DataOps, veri mühendisliği, veri bilimi ve operasyon ekipleri arasındaki iş birliğini artırmakta, veri hatlarının otomasyonunu sağlamakta ve veri kalitesini sürekli olarak izlemektedir [112]. Bu yaklaşım, toplu (batch) ve gerçek zamanlı (streaming) veri işleme senaryolarını kapsamaktadır [113].

DataOps'un başlıca avantajları arasında hız, esneklik, kalite güvencesi ve maliyet optimizasyonu yer almaktadır. Özellikle büyük veri ortamlarında sürekli entegrasyon ve sürekli teslim (CI/CD) yaklaşımları, veri hatlarının daha az hata ile çalışmasına imkân vermektedir [114].

5.2. ETL

ETL farklı kaynaklardan verilerin çıkarılması, dönüştürülmesi ve hedef sisteme yüklenmesi sürecidir [115]. **Extract** aşamasında veriler kaynak sistemlerden alınır; **Transform** aşamasında veriler temizlenir, birleştirilir ve analiz için uygun biçime dönüştürülür; **Load** aşamasında ise işlenmiş veriler hedef veritabanına veya veri ambarına aktarılır [116].

ETL süreci, veri bütünlüğü ve kalite kontrolü açısından kritik öneme sahiptir [117]. Başlıca avantajları arasında veri standardizasyonu, merkezi yönetim ve analiz süreçlerinin hızlanması bulunmaktadır. Ancak, büyük veri hacimlerinde dönüşüm aşaması darboğaz oluşturabilmektedir [118].

Alternatif bir yaklaşım olan **ELT (Extract, Load, Transform)**, dönüşüm işlemini hedef sistem üzerinde gerçekleştirmektedir ve özellikle bulut tabanlı veri ambarlarında tercih edilmektedir [119]. ETL dönüşümü yüklemeden önce gerçekleştirirken, ELT dönüşümü yüklemeden sonra yapmaktadır [120][121].

5.3. Gerçek Zamanlı Veri İşleme

Gerçek Zamanlı Veri Akışı, verilerin üretildiği anda işlenmesine olanak tanıyan bir veri işleme yaklaşımıdır [122]. Bu yöntem, düşük gecikme ile anlık veri analizini mümkün kılmaktadır [123]. Finans, IoT, e-ticaret ve güvenlik izleme gibi alanlarda yaygın olarak kullanılmaktadır [124]. Başlıca avantajları arasında hızlı karar alma, sürekli güncellenen verilerle çalışma ve anlık anomali tespiti bulunmaktadır [125].

Gerçek zamanlı veri akışı sistemleri genellikle mesaj kuyrukları, akış işleme motorları ve veri depolama bileşenlerinden oluşmaktadır [126].

5.4. Öneri Sistemleri

Öneri sistemleri (Recommendation Systems), kullanıcıların potansiyel ilgilerine uygun öğeleri (ör. film, kitap, ürün, müzik) tahmin ederek sunmayı amaçlayan, uzmanlaşmış bilgi filtreleme mekanizmalarıdır. Temel amaçları, kullanıcı deneyiminin kalitesini artırmak, etkileşimi teşvik etmek ve karar alma süreçlerini kolaylaştırmaktır. Kullanıcı davranışları ve tercihlerine ilişkin geçmiş verilerden yararlanarak kişiselleştirilmiş çıktılar üreten bu sistemler, modern dijital ekosistemlerde arama maliyetini azaltmada ve genel memnuniyeti artırmada merkezi bir rol oynamaktadır. Özellikle derin öğrenme ve grafik tabanlı sinir ağlarındaki ilerlemeler, öneri sistemlerinin etkinliğini daha da artırmış, daha büyük ölçeklerde bağlamsal farkındalık ve hassasiyetle çalışmasını sağlamıştır [127].

5.5. Generative Artificial Intelligence (Yenilikçi Yapay Zeka)

Son yıllarda yapay zekâdaki hızlı ilerlemeler, **üretici yapay zekâ (generative AI),** sohbet robotları ve **bilgiyle zenginleştirilmiş üretim (retrieval-augmented generation – RAG)** gibi yaklaşımları öne çıkarmıştır. Bu teknolojiler, kullanıcı etkileşimini derinleştiren, bilgiye erişimi hızlandıran ve kişiselleştirilmiş deneyimler sunan sistemlerin temelini oluşturmaktadır. Hem teknik katkıları hem de toplumsal etkileri açısından akademik literatürde geniş ölçüde incelenmektedir. Özellikle giderek dijitalleşen dünyada, bilgi kirliliği, doğruluk ve etkileşim sorunlarına çözüm potansiyeli ile önemi daha da artmaktadır.

Üretici yapay zekâ, eğitim verilerinden öğrenilen kalıplara dayanarak yeni ve anlamlı içerikler üretebilen yöntemleri ifade etmektedir. Bu içerikler metin, görsel, ses, video ya da programlama kodu olabilir. **GitHub Copilot** gibi örnekler, bu teknolojinin iş dünyasını ve günlük yaşamı nasıl dönüştürdüğünü göstermektedir. Geleneksel olarak yalnızca insana özgü kabul edilen yaratıcı işlevler artık yapay zekâ sistemleri tarafından gerçekleştirilebilmektedir [128].

Retrieval-Augmented Generation (RAG) ise büyük dil modellerinin sınırlılıklarını gidermek için geliştirilmiş bir yaklaşımdır. Temel olarak, yalnızca model parametrelerine gömülü bilgiye dayanmadan, harici bilgi kaynaklarından (ör. web, veritabanları, bilgi grafikleri) içerik çıkararak daha güncel, doğru ve izlenebilir çıktılar üretmeyi mümkün kılmaktadır. **Agentic RAG**, bu yaklaşımı bir adım öteye taşıyarak yapay zekâ ajanlarının süreci otonom olarak planlamasına ve gerektiğinde farklı araçlar (arama motorları, sorgulama dilleri, grafik tabanlı sorgular vb.) kullanmasına imkân tanır. Son olarak, **human-in-the-retrieval (getirme adımında insan denetimi)** yaklaşımı, yalnızca pasif bilgi kaynaklarını değil, insan uzmanların örtük bilgisini de sürece dahil ederek sistemin doğruluğunu, bağlamsal uygunluğunu ve güvenilirliğini artırmaktadır [129].

6. MATERYAL VE YÖNTEM

Bu çalışma, modern yazılım geliştirme metodolojilerinin yazılım geliştirme süreçleri üzerindeki etkisini incelemeyi ve geleneksel yöntemlerle karşılaştırıldığında ölçülebilir kazanımları belirlemeyi amaçlamaktadır. Özellikle, mikroservis mimarisi, bulut-yerel teknolojiler, yapay zekâ/makine öğrenmesi entegrasyonları ve DevOps uygulamalarının yazılım geliştirme döngüsüne katkıları sayısal metrikler aracılığıyla değerlendirilmiştir.

Vaka çalışması kapsamında, geleneksel monolitik yapı yerine mikroservis mimarisi benimsenmiş ve sekiz bağımsız servis tasarlanmıştır. Her bir servis, kendi yaşam döngüsü içerisinde geliştirilmiş, test edilmiş ve dağıtılmıştır. Bu modüler yaklaşım, servisler arası bağımlılıkları azaltmış ve paralel geliştirme süreçlerini mümkün kılmıştır.

Buna ek olarak, bulut-yerel teknolojiler entegre edilmiştir. AWS üzerinde Kubernetes kümeleri kurulmuş ve esnek, yüksek erişilebilir sistemler tasarlamak için konteynerleştirme kullanılmıştır. **Otomatik ölçeklendirme, yük dengeleme** ve **Kod Olarak Altyapı (Infrastructure as Code – CloudFormation)** kullanımı, dağıtım süreçlerini otomatikleştirerek manuel operasyonel yükü azaltmıştır.

Ayrıca, AI/ML tabanlı bir veri hattı geliştirilmiştir. **AWS Personalize** kullanılarak bir öneri motoru entegre edilmiş, **AWS Bedrock** aracılığıyla bir yapay zekâ sohbet robotu geliştirilmiş ve **Kinesis** ile gerçek zamana yakın veri akışı sağlanmıştır. Bu mimari, kullanıcı davranışlarının anlık analizine, makine öğrenmesi modellerinin otomatik olarak eğitilmesine ve güncellenmesine zemin hazırlamıştır.

DevOps metodolojileri, GitOps tabanlı süreçler, CI/CD hatları ile otomatik test ve izleme mekanizmalarıyla desteklenmiştir. Bu yaklaşım, sürekli entegrasyon (CI) ve hızlı teslimatı (CD) mümkün kılmış; test otomasyonu sayesinde kalite güvencesini güçlendirmiştir. Ayrıca, **DynamoDB, S3 ve Kinesis** gibi bulut-yerel veri servisleri kullanılarak ölçeklenebilir ve yüksek performanslı bir veri yönetim sistemi oluşturulmuştur.

Elde edilen sonuçlar, modern yazılım geliştirme metodolojilerinin geleneksel yaklaşımlara kıyasla ölçülebilir avantajlar sunduğunu ortaya koymuştur. Özellikle, mikroservis mimarileri geliştirme hızını ve ölçeklenebilirliği artırmış; bulut-yerel teknolojiler operasyonel süreçleri

otomatikleştirmiş; yapay zeka entegrasyonu veri odaklı işlevselliği güçlendirmiş; DevOps uygulamaları ise kalite güvencesini pekiştirmiştir. Bu bulgular, modern yazılım ekosistemi içerisinde teknoloji ve mimari seçimlerinin yalnızca teorik değil, aynı zamanda sayısal metriklerle desteklenebilir olduğunu göstermektedir.

6.1. Uygulama

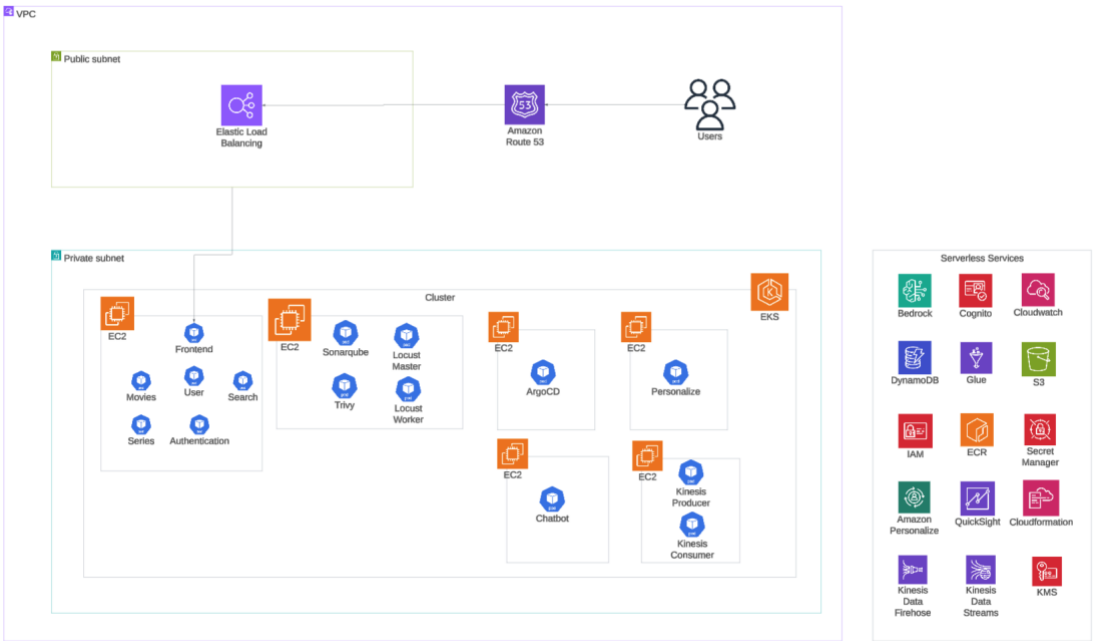
Github deposu (repository) [130] adresinde yer almakta olup, proje kapsamında kullanılan tüm kod tabanı ve yapılandırma dosyalarını barındırmaktadır. Tez kapsamında kullanılan GitHub deposu, altyapı otomasyonu, veri yönetimi ve mikroservis geliştirme süreçlerinin düzenli bir şekilde yürütülmesine imkân tanıyacak şekilde yapılandırılmıştır. Depoda yer alan cloudformation_templates klasörü, Infrastructure as Code (IaC) yaklaşımıyla hazırlanmış CloudFormation betiklerini barındırmakta ve altyapı kaynaklarının otomatik, tutarlı ve tekrarlanabilir biçimde oluşturulmasını sağlamaktadır. dataset ve initial_data klasörleri sırasıyla çalışma kapsamında kullanılan veri kümelerini ve başlangıç verilerini içermekte olup, veri yönetiminde bütünlük ve erişilebilirlik sağlamaktadır. kubernetes_manifest klasörü, dağıtım, servis ve yapılandırma dosyaları gibi Kubernetes'e özgü tanımları barındırarak mikroservislerin bulut-yerel mimaride esnek ve ölçeklenebilir şekilde yönetilmesine katkı sunmaktadır. scripts klasörü otomasyon betiklerini içererek tekrarlayan operasyonel işlemlerin kolaylaştırılmasını sağlarken, services klasörü ise tüm mikroservislerin bağımsız kod tabanlarını barındırarak modüler geliştirme ve ölçeklenebilirlik imkânı tanımaktadır. Son olarak, kök dizin (root directory) manifest ve log dosyalarını barındırarak depo yapısına bütünsel bir görünüm kazandırmaktadır. Bu klasörleme yapısı, proje kapsamında geliştirilen sistemin izlenebilir, modüler ve yeniden üretilebilir olmasına önemli katkılar sunmaktadır.

Tablo 6.1 Uygulamanın klasör yapısı

Klasör Adı	İçerik
cloudformation_templates	IaC CloudFormation betiklerini saklamaktadır.
dataset	Tüm veri kümelerini saklamaktadır.
initial_data	Başlangıç verilerini saklamaktadır.
kubernetes_manifest	Kubernetes dosyalarını saklamaktadır.
scripts	Otomasyon betiklerini saklamaktadır.

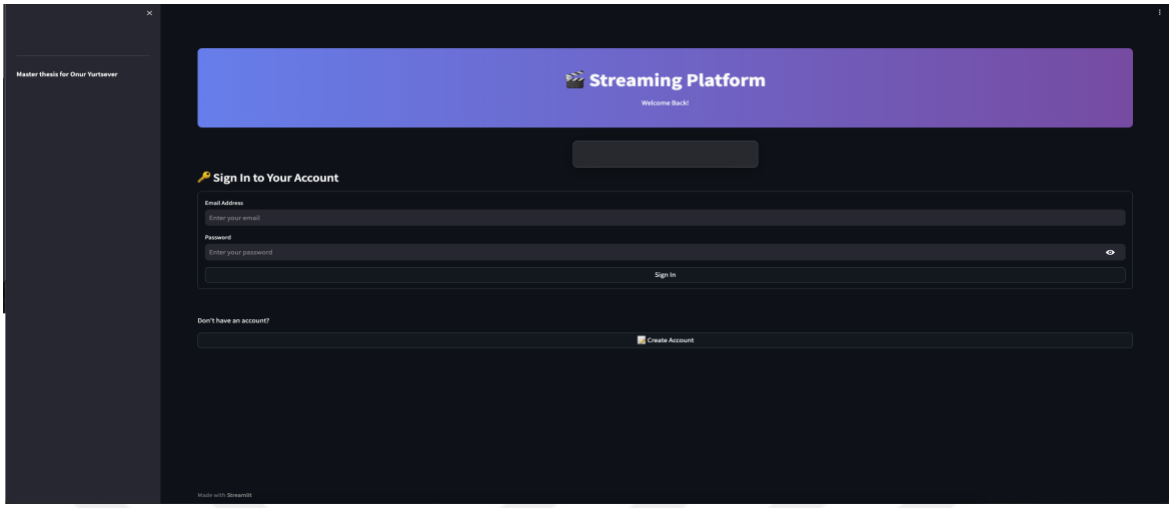
services	Tüm mikroservislerin kod tabanını saklamaktadır.
Root directory (Kök dizin)	Manifest ve log dosyalarını barındırmaktadır.

Şekil 6.1'de tez kapsamında geliştirilen film/dizi platformunun AWS üzerindeki yüksek seviyeli mimarisi gösterilmektedir. Kullanıcılar, Route 53 ile adlandırılan ve genel alt ağda konumlanan bir Elastic Load Balancer üzerinden sisteme erişmektedir. Özel alt ağda frontend ve mikroservisler (Movies, Series, User, Search, Authentication) bir EC2 örneği üzerinde çalışmakta; buna ek olarak Sonarqube, Trivy, Locust ve ArgoCD gibi DevOps araçlarını barındıran EC2 örnekleri (instances) ile Kinesis producer/consumer, Personalize senaryosu ve sohbet robotu bileşenleri yer almaktadır. EKS kümesi, mikroservis tabanlı uygulamanın konteyner orkestrasyonunu sağlamaktadır. Sağ tarafta yer alan blokta ise Bedrock, Cognito, CloudWatch, DynamoDB, Glue, S3, Kinesis, QuickSight, CloudFormation ve KMS gibi sunucusuz ve yönetilen hizmetler gösterilmekte; bu hizmetler uygulamanın, veri hattı ve yapay zekâ bileşenleri tarafından kullanılmaktadır.

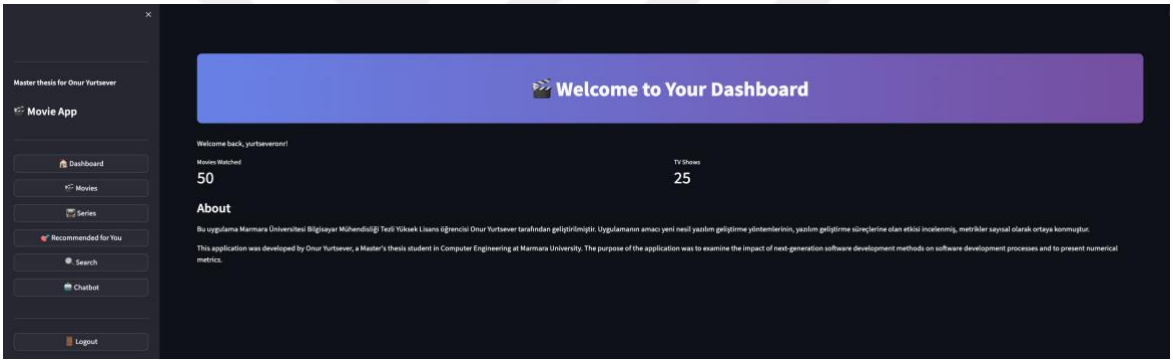


Şekil 6.1 Kullanıcının genel mimarisi

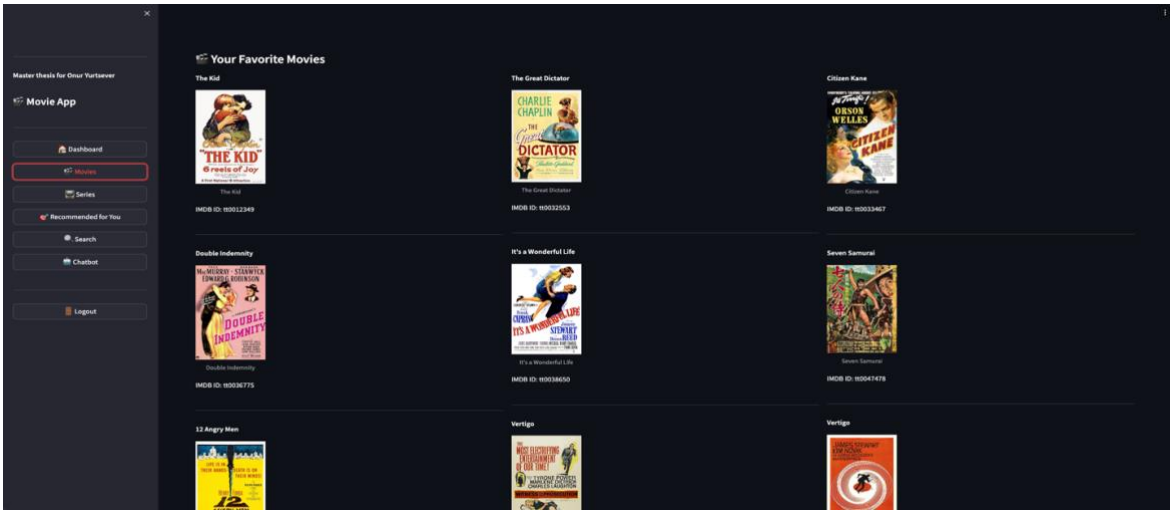
Aşağıdaki ekran görüntüleri geliştirilen ekran görüntülerini içermektedir.



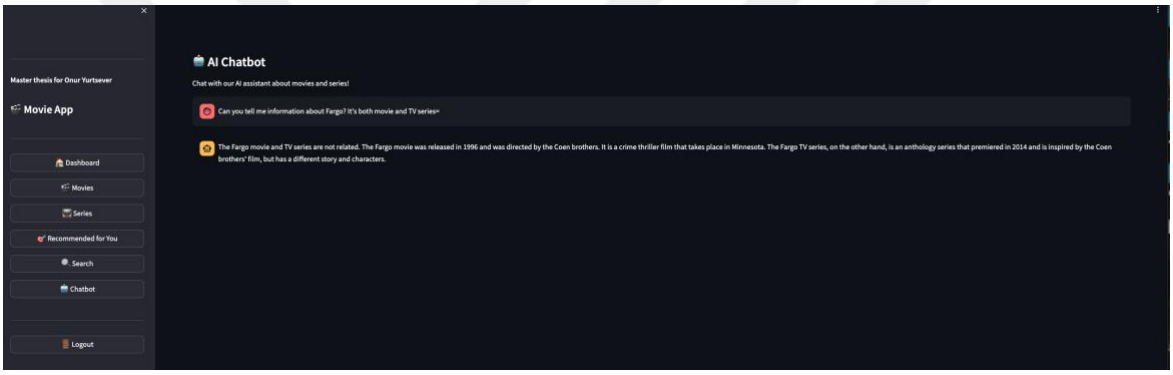
Şekil 6.2 Uygulamanın giriş kısmı



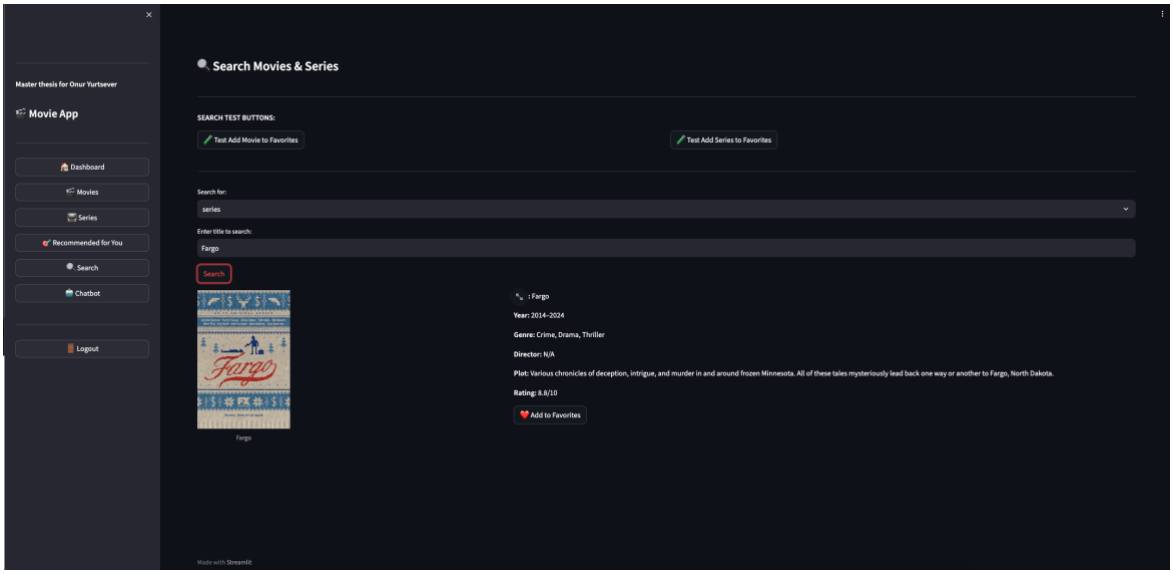
Şekil 6.3 Kullanıcı uygulamaya giriş yaptıktan sonra kullanıcıya özet sayfa



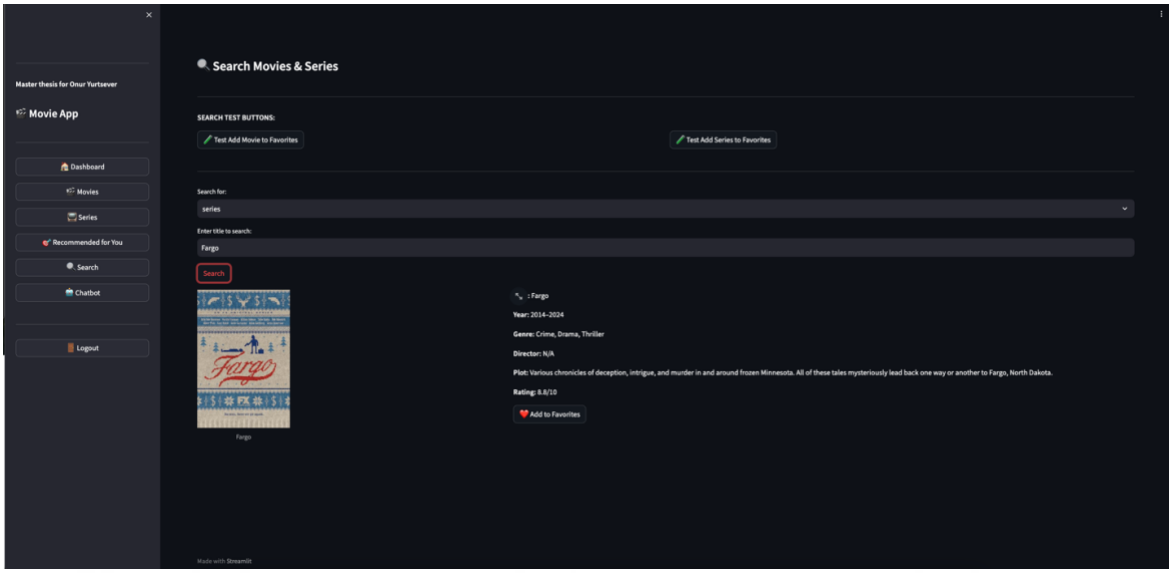
Şekil 6.4 Kullanıcının beğendiği filmleri gösteren sayfa



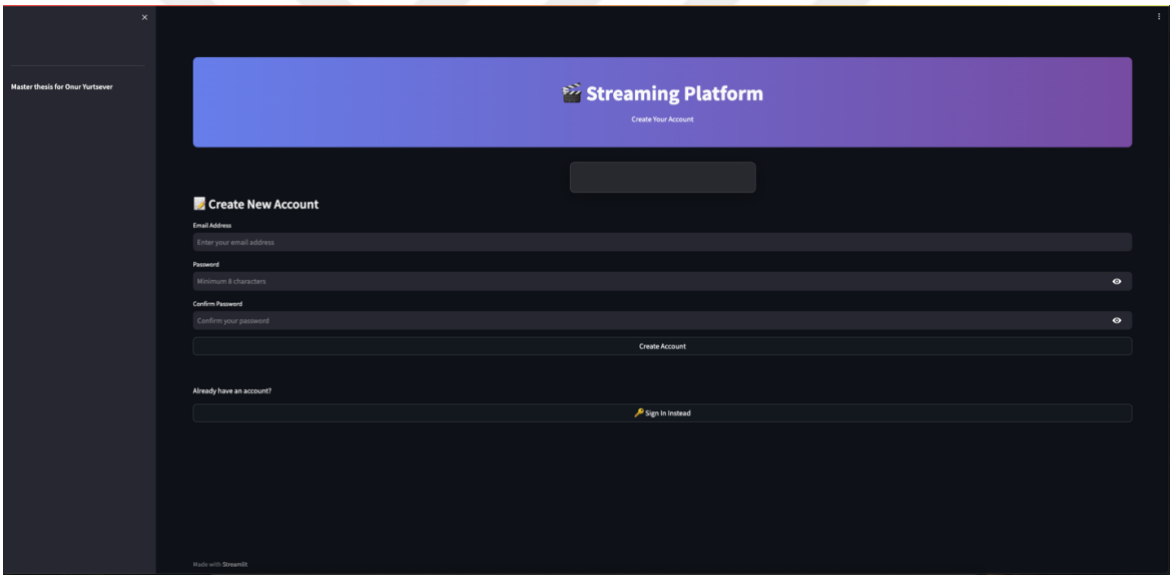
Şekil 6.5 Kullanıcının yapay zeka ile interaktif olarak konuşabildiği sayfa



Şekil 6.6 Kullanıcının film araması yapabileceği sayfa



Şekil 6.7 Kullanıcının dizi araması yapabileceği sayfa



Şekil 6.8 Kullanıcının yeni üyelik açabileceği sayfa

6.2. Geliştirilen Mikroservisler

Authentication (Doğrulama) servisi, kullanıcı kaydı, giriş, e-posta doğrulama ve oturum yönetimini gerçekleştirmektedir. Python programlama dili ve Flask framework'ü kullanılarak geliştirilmiş olup bazı kütüphaneler ile desteklenmektedir. AWS Cognito servisi üzerinden güvenli kimlik doğrulama ve kullanıcı yönetimi sağlamaktadır. 5000 portunda çalışmakta olup hata yönetimi ve istek doğrulama özelliklerine sahiptir.

Frontend (Önyüz) servisi, kullanıcı arayüzünü, kontrol panelini, film/dizi listeleme ve öneri ekranlarını sunmaktadır. Python programlama dili ve Streamlit framework'ü ile geliştirilmiştir. Bazı kütüphaneler ile API entegrasyonu sağlamakta, duyarlı tasarım, kimlik doğrulama akışı, gerçek zamanlı güncellemeler sunmakta ve Streamlit'in varsayılan 8501 portunda çalışmaktadır.

Movie (Film) servisi, film verilerini yönetmekte, ve film detaylarını sağlamaktadır. Go programlama diliyle geliştirilmiş olup Go'nun standart kütüphanesi ve Gorilla/mux yönlendirme kütüphanesini kullanmaktadır. AWS DynamoDB ile entegredir ve 8081 portunda çalışmaktadır.

Dizi servisi ise dizi verilerini ve detaylarını yönetmektedir. Movie Service ile benzer teknoloji yığınına sahiptir, AWS DynamoDB ile entegredir ve 8082 portunda çalışmaktadır.

Arama servisi, film ve dizi arama işlemlerini gerçekleştirmektedir. Node.js ve TypeScript kullanılarak geliştirilmiştir. Express framework'ü üzerinden API sağlamaktadır. harici API ile entegre çalışmakta, AWS DynamoDB üzerinden veri erişimi sağlamakta ve 8084 portunda hizmet vermektedir.

Kullanıcı servisi, kullanıcı profillerini ve favori listesi yönetimini gerçekleştirmektedir. Python dili ve FastAPI framework'ü ile geliştirilmiştir. Kullanıcı verilerini AWS DynamoDB entegrasyonu ile yönetmekte ve 8080 portunda çalışmaktadır.

Makine öğrenme servisi, AWS Personalize kullanarak film ve dizi önerileri üretmektedir. Python programlama dili ve FastAPI framework'ü ile geliştirilmiş olup Boto3 kütüphanesi üzerinden AWS Personalize entegrasyonu sağlamaktadır. Gerçek zamanlı çıkarım (inference) sunmakta ve 8000 portunda çalışmaktadır.

Bedrock Backend (Arka yüz) servisi yapay zekâ destekli sohbet işlevi sunmaktadır. Python dili ve Flask framework'ü kullanılarak geliştirilmiştir. AWS Bedrock Agent entegrasyonu ile doğal dil işleme gerçekleştirmekte, boto3 kütüphanesi aracılığıyla AWS servislerine bağlanmaktadır. 8091 portunda çalışmaktadır.

Kinesis Producer Service, kullanıcı etkileşimlerini gerçek zamanlı toplamaktadır. Node.js ile geliştirilmiş olup AWS Kinesis Data Streams ile entegredir. Express framework üzerinden REST (Representational State Transfer / Temsili Durum Aktarımı) API sağlamaktadır ve 3000 portunda çalışmaktadır.

Kinesis Consumer Service, Kinesis'ten gelen veri akışını işlemektedir. Node.js ile geliştirilmiş olup AWS SDK üzerinden Kinesis Data Streams'ten veri okumaktadır. Gerçek zamanlı veri işleme ve analitik fonksiyonları yerine getirmektedir.

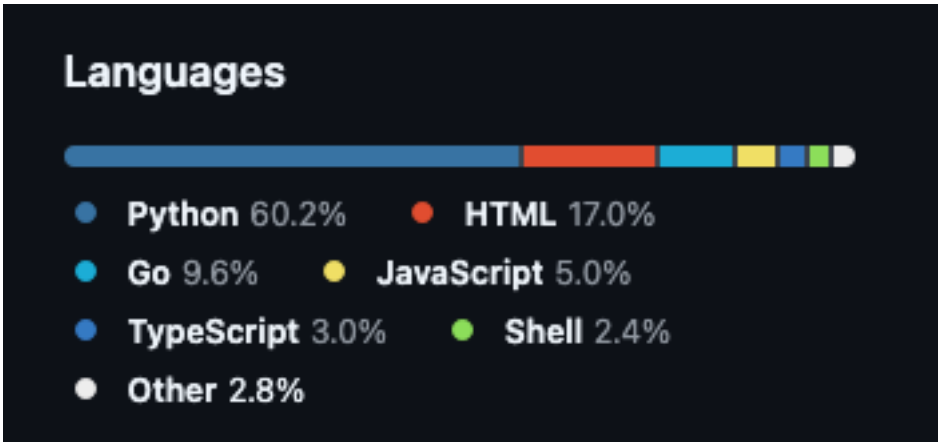
Bu mimari, sekiz bağımsız mikroservisten oluşan ölçeklenebilir ve modüler bir yapı sunmaktadır. Her bir servis, kendi programlama dili, framework'ü ve AWS servisi ile özelleşmiş olup bağımsız geliştirme ve dağıtım süreçlerini kolaylaştırmaktadır.

Farklı programlama dilleri ve teknolojilerin birleşimi, sistemin esnekliğini artırmaktadır. Python tabanlı servisler hızlı prototipleme, zengin kütüphane desteği ve AWS servisleriyle güçlü entegrasyon sağlamaktadır. Go tabanlı Movie ve Series servisleri yüksek performans, düşük bellek tüketimi ve verimlilik sunmaktadır. Node.js/TypeScript tabanlı servisler ise asenkron işlem gücü ve geniş ekosistem desteği sayesinde gerçek zamanlı veri işleme ve harici API entegrasyonlarında öne çıkmaktadır.

Bu çeşitlilik, her servisin ihtiyacına en uygun teknolojinin seçilmesine imkân tanımaktadır. Örneğin, gerçek zamanlı veri işleme için Node.js tercih edilirken, veri yoğun servislerde yüksek performans için Go kullanılmaktadır. Python'un esnekliği ve güçlü kütüphane desteği ise kullanıcı yönetimi ve makine öğrenmesi tabanlı servislerde etkili biçimde kullanılmaktadır.

Sonuç olarak, bu çok dilli ve çok teknolojili yaklaşım, sistemin yalnızca ölçeklenebilirliğini ve performansını artırmakla kalmamakta, aynı zamanda en uygun teknolojinin her servis için ayrı ayrı kullanılmasına olanak tanıyarak **çok dilli (polyglot) bir mimari** benimsemektedir. Bu mimari, farklı ekiplerin paralel iş birliğini, bağımsız servis evrimini ve uzun vadeli sürdürülebilirliği mümkün kılmaktadır.

Şekil 6.9, bu programlama dillerinin yüzdesel dağılımlarını göstermektedir.



Şekil 6.9 Kullanılan programlama dillerin yüzdesel dağılımı

6.3. AWS

AWS 2006 yılında bulut tabanlı hizmetler sunmaya başlamıştır. AWS, bilgi işlem altyapısını düşük maliyetle sunarak işletmelerin donanım ve sermaye maliyetlerini azaltmayı hedefler. Özellikle depolama için Amazon S3 ve hesaplama için EC2 (Elastic Compute Cloud) gibi hizmetler sağlar. Ayrıca, kuruluşlar için e-posta hizmeti gibi çözümler sunarak farklı ihtiyaçlara yönelik ölçeklenebilir ve esnek hizmetler sağlamaktadır.

AWS, düşük maliyet ve esnek altyapısı ile kullanıcıların donanım yönetme zorluğu yaşamadan bulut hizmetlerinden faydalanmalarını sağlar. Bu özellikleriyle işletmelerin iş yüklerini buluta taşımalarına olanak tanır.

Pazar payı açısından AWS, bulut altyapı hizmetleri pazarında lider konumdadır. Microsoft Azure ve Google Cloud ile karşılaştırıldığında AWS en yüksek pazar payına sahiptir. 2022 yılı verilerine göre AWS, EC2 ve S3 gibi hizmetleri sayesinde müşteri taleplerini dinamik ve ölçeklenebilir bir şekilde karşılamaktadır. Bu nedenle Netflix ve Dropbox gibi büyük şirketler AWS altyapısını güvenle kullanmaktadır [131].

Amazon Web Services'in (AWS) çok yönlülüğü ve sağlamlığı, farklı sektörlerde faaliyet gösteren önde gelen şirketler tarafından benimsenmesiyle en iyi şekilde ortaya konulmaktadır. Bu kuruluşlar, AWS'i kullanarak daha yüksek verimlilik, ölçeklenebilirlik, maliyet azaltımı ve yenilik kapasitesi elde etmişlerdir.

- **Netflix:** Küresel ölçekte bir yayıncılık lideri olan Netflix, milyonlarca kullanıcıya kesintisiz dijital içerik sunumunu sağlamak için AWS altyapısına büyük ölçüde

güvenmektedir. Özellikle, depolama için Amazon S3, küresel içerik dağıtımı için Amazon CloudFront ve kullanıcı verilerinin yönetimi için Amazon DynamoDB kullanılmaktadır. AWS'in yüksek bant genişliğine sahip, gecikmeye duyarlı iş yüklerini destekleme yeteneği, Netflix'in ölçeklenebilirliği ve kullanıcı deneyimi açısından kritik bir rol oynamaktadır.

- **Airbnb:** Küresel konaklama platformu olan Airbnb, geniş ölçekli operasyonlarını yönetmek ve yüksek hacimli kullanıcı trafiğini karşılamak için AWS'i kullanmaktadır. Temel hizmetler arasında hesaplama için EC2, depolama için S3, ilişkisel veritabanı yönetimi için RDS (Relational Database Service – İlişkisel Veri Tabanı) yer almakta; AWS Lambda ise sunucusuz, olay odaklı çalışmayı desteklemektedir. Bu kombinasyon, Airbnb'nin veri güvenliğini, esnekliğini ve yoğun rezervasyon dönemlerinde dahi üstün müşteri deneyimi sağlamasını mümkün kılmaktadır.
- **Lyft:** Bir araç paylaşım şirketi olan Lyft, dalgalanan kullanıcı taleplerine yanıt olarak arka uç sistemlerini dinamik biçimde ölçeklendirmek için AWS'i kullanmaktadır. EC2, S3, RDS ve CloudFront üzerinden Lyft; güvenilir yolculuk eşleştirmesi, veri depolama ve iletişimi sağlayarak ulaşım verimliliğini ve müşteri memnuniyetini artırmaktadır.
- **Capital One:** Büyük bir finans kurumu olan Capital One, BT altyapısının önemli bir bölümünü AWS'e taşıyarak verimlilik, yanıt verme hızı ve düzenleyici uyumluluğunu geliştirmiştir. EC2 ve S3 gibi hizmetleri kullanarak şirket, büyük veri kümelerini güvenli bir şekilde yönetebilmekte ve analiz edebilmekte; aynı zamanda sıkı sektör standartlarına uyum sağlamaktadır.

Bu örnekler, AWS'in eğlence, konaklama, ulaşım ve finans dahil olmak üzere birçok sektörde dijital dönüşümün vazgeçilmez bir kolaylaştırıcısı haline geldiğini ortaya koymaktadır. AWS'in küresel ölçeklenebilirlik, veri güvenliği, operasyonel esneklik ve maliyet etkinliği sunabilme kapasitesi, onu dijital çağda yenilik yapmayı ve rekabet etmeyi hedefleyen kuruluşlar için stratejik bir omurga haline getirmektedir [132].

Amazon Web Services (AWS), öncü piyasa konumu ve teknolojik olgunluğu ile diğer bulut sağlayıcılarından ayrışmaktadır. 2003 yılında Amazon'un iç altyapı yetkinliklerinden doğan ve zamanla kapsamlı bir internet işletim sistemi konseptine evrilen gelişim süreci, AWS'i

modern bulut bilişim paradigmalarının temel mimarı konumuna getirmiştir. Bu **ilk hareket avantajı**, AWS'in depolama, ağ, yapay zekâ ve kurumsal çözümleri kapsayan 200'ün üzerinde bulut hizmetinden oluşan en geniş hizmet portföyünü inşa etmesini ve 24 coğrafi bölgede 77 Erişilebilirlik Alanına (veri merkezleri) yayılan küresel bir altyapı oluşturmasını mümkün kılmıştır.

Karşılaştırıldığında, **Microsoft Azure**, Windows ekosistemleriyle derin entegrasyon ve hibrit bulut çözümleri üzerine yoğunlaşırken; **Google Cloud Platform**, makine öğrenmesi ve veri analitiği konusundaki uzmanlığıyla öne çıkmaktadır. Bununla birlikte, AWS'in pazar liderliği; geniş ölçekli kurumsal benimsemelerle kanıtlanan güvenilirliği, kapsamlı iş ortaklığı ekosistemi ve sürekli yenilik kapasitesi sayesinde korunmakta ve güçlenmektedir. Bu özellikler, AWS'in farklı endüstrilerde dijital dönüşüm stratejilerinin omurgası olarak konumunu pekiştirmektedir[133][134].

6.4. AWS'nin Global Altyapısı

AWS, dünyanın en kapsamlı ve en dayanıklı bulut altyapılarından birini işletmektedir. Tasarım prensibi, hesaplama, depolama ve ağ kaynaklarını müşterilere daha yakın konumlandırmak, böylece yüksek erişilebilirlik ve hata toleransını farklı coğrafi bölgeler genelinde güvence altına almaktır.

AWS'in temel küresel altyapısı [135] kapsamında şu şekilde tanımlanmaktadır:

- **Regions (Bölgeler):** Bir Region, belirli bir coğrafi alanı temsil etmektedir (ör. N. Virginia, Frankfurt, Tokyo). Her Region, yedekliliği ve yasal uyumluluğu sağlamak amacıyla birbirinden izole edilmiş çok sayıda veri merkezi kümesini barındırır. Müşteriler, yakınlık, yasal gereklilikler veya gecikme ihtiyaçlarına göre Region seçimi yapmaktadır.
- **Availability Zones (Erişilebilirlik Alanları):** Her bölge, en az iki (çoğu zaman üç veya daha fazla) Erişilebilirlik Alanına ayrılmıştır. Bir erişilebilirlik alanı, bağımsız güç, soğutma ve ağ altyapısına sahip bir veya birden fazla fiziksel veri merkezinden oluşur. Düşük gecikmeli bağlantılarla birbirine bağlanan bu alanlar, uygulamaların

yüksek erişilebilirlik ve hata toleransı sağlaması için erişilebilirlik alanları arasında dağıtım yapılmasına olanak tanır.

- **Edge Locations (Uç Noktalar):** AWS, Amazon CloudFront içerik dağıtım ağı kapsamında geniş bir uç nokta ağı işletmektedir. Bu noktalar, verileri son kullanıcılara daha yakın konumda önbelleğe alarak içerik dağıtımında gecikmeyi azaltmaktadır. AWS ayrıca, daha az sıklıkla erişilen içerikleri kaynak sunuculara kıyasla daha verimli şekilde sunan Bölgesel Uç Nokta Önbellekleri (Regional Edge Caches) de bulundurmaktadır.
- **Genişletilmiş Altyapı:**
    - **AWS Outposts:** AWS hizmetlerini hibrit iş yükleri için doğrudan kurum içi veri merkezlerine taşımaktadır.
    - **AWS Local Zones:** Büyük metropol alanlarına yakın konumlandırılarak oyun, canlı yayın gibi ultra düşük gecikme gerektiren uygulamalar için hesaplama, depolama ve veritabanı hizmetleri sunmaktadır.
    - **AWS Wavelength:** AWS'in hesaplama ve depolama hizmetlerini 5G ağlarına entegre ederek artırılmış/sanal gerçeklik ve uç tabanlı yapay zekâ gibi minimal gecikme gerektiren uygulamalara imkân sağlamaktadır.

6.5. AWS'nin Ağ Altyapısı

AWS, kullanıcılara sanal bir ağ katmanı oluşturmamıza imkân tanımaktadır. Böylelikle, kullanıcılar iş yükleri ağ katmanların ayırabilir. Bu yapının temel bileşenleri [136]'te özetlenmektedir:

- **VPC (Virtual Private Cloud):** AWS içerisinde kaynakların (ör. EC2 sanal makineleri) çalıştırılabildiği mantıksal olarak izole edilmiş sanal ağdır. IP adresleme, yönlendirme ve güvenlik üzerinde tam kontrol sağlamaktadır.
- **Subnets (Alt Ağlar):** VPC'yi daha küçük ağ segmentlerine ayırmakta ve her biri belirli bir Erişilebilirlik Alanına (Availability Zone) bağlı olmaktadır. Alt ağlar, internet erişimi olan *genel (*public*)* alt ağlar veya yalnızca iç iletişime izin veren, internet erişimini NAT Gateway üzerinden sağlayan özel (private*)* alt ağlar olarak yapılandırılabilmektedir.

- **Route Tables (Yönlendirme Tabloları):** Ağ trafiğinin akışını kontrol etmekte ve paketlerin hangi rotalar üzerinden yönlendirileceğini belirleyen kuralları içermektedir.
- **Internet Gateway (İnternet Geçidi):** VPC kaynaklarının (genel alt ağlardaki) genel internet ile iletişim kurmasını sağlamaktadır. VPC'ye eklenmekte ve genel yönlendirme tablolarında referans gösterilmektedir.
- **NAT Gateways:** Özel alt ağlarda bulunan kaynakların dışa doğru internet trafiği başlatmasına (örn. güncellemeler, API çağrıları) izin verirken, dışarıdan gelen istenmeyen erişimleri engellemektedir. Yüksek erişilebilirlik için, her özel alt ağın ilgili genel alt ağda konumlandırılmış bir NAT Gateway ile eşleştirildiği bir yapı kullanılmaktadır.

6.6. AWS'nin Güvenlik Altyapısı

Identity and Access Management (IAM): IAM, AWS üzerinde kimlik doğrulama ve yetkilendirme süreçlerinin merkezi kontrolünü sağlamaktadır. Kullanıcılar, gruplar, roller ve politikalar tanımlanarak asgari ayrıcalık ilkesi *(*principle of least privilege) uygulanmakta, böylece varlıkların yalnızca görevlerini yerine getirmek için gerekli en düşük izinlerle çalışması garanti altına alınmaktadır.

Security Groups (SGs): Güvenlik Grupları, EC2 sanal makineleri gibi kaynaklara bağlı durum bilgili (stateful) sanal güvenlik duvarları olarak işlev görmektedir. Gelen ve giden ağ trafiğini kaynak düzeyinde düzenleyerek izin verilen protokoller, portlar ve IP aralıkları üzerinde ayrıntılı kontrol imkânı sunmaktadır.

Key Management Service (KMS): AWS KMS, hem hareketsiz veriler (data-at-rest) hem de aktarım halindeki veriler (data-in-transit) için kullanılan kriptografik anahtarların oluşturulmasını, yönetimini ve rotasyonunu sağlamaktadır. S3, RDS ve EBS gibi temel AWS hizmetleriyle entegre çalışarak varsayılan olarak şifrelemeyi zorunlu kılmaktadır.

Secrets Manager: AWS Secrets Manager, API anahtarları, veritabanı parolaları ve erişim belirteçleri (tokens) gibi hassas kimlik bilgilerini güvenli şekilde saklamakta, döndürmekte

(rotation) ve yönetmektedir. Bu yaklaşım, uygulamalara gizli bilgilerin gömülmesi (hardcoding) gerekliliğini ortadan kaldırarak güvenlik açıklarının önüne geçmektedir.

Aktarım Halindeki Veri (Data in Transit): Ağlar üzerinden aktif olarak hareket eden, örneğin kullanıcılarla AWS hizmetleri arasındaki veya AWS bileşenleri arasındaki bilgi akışını ifade etmektedir. Olası veri yakalama (interception) veya değiştirme (tampering) risklerini azaltmak için AWS, endüstri standardı olan **Transport Layer Security (TLS)** protokolünü kullanarak iletim sırasında trafiği şifrelemektedir[137].

6.7. Kullanılan AWS Servisleri

Amazon EC2, EBS, Load Balancing ve Auto Scaling birlikte, ölçeklenebilir ve güvenilir bulut mimarilerinin omurgasını oluşturmaktadır[138][139][140][141].

- **EC2** isteğe bağlı olarak uygulamaları çalıştırabilen yeniden boyutlandırılabilir hesaplama sanal makineleri (instances) sağlamaktadır.
- **EBS (Elastic Block Store)**, EC2 sanal makineler kalıcı depolama birimleri bağlayarak veri dayanıklılığı ve yüksek performans sağlamaktadır.
- **ELB (Elastic Load Balancing)** gelen trafiği birden fazla EC2 örneğine dağıtarak darboğazları önlemekte ve yüksek erişilebilirlik sağlamaktadır.
- **Auto Scaling Groups (Otomatik Ölçeklendirme Grupları)**, talebe bağlı olarak EC2 örneklerinin sayısını otomatik biçimde ayarlamaktadır. Yüksek talepli dönemlerde ölçeklendirme (scale-out), düşük aktivitelerde küçülme (scale-in) gerçekleştirerek hem hata toleransı hem de maliyet optimizasyonu sağlamaktadır. ELB ile bütünleşik çalışarak esnek, dayanıklı ve elastik bir bulut altyapısı oluşturur.

Amazon CloudWatch, AWS kaynaklarından ve uygulamalardan metrikler, loglar ve olaylar toplayan bir izleme ve gözlemlenebilirlik hizmetidir. Gerçek zamanlı performans takibi, otomatik alarmlar ve görselleştirme panelleri sunarak güvenilirliği artırmakta, sorunları gidermeye yardımcı olmakta ve kaynak kullanımını optimize etmektedir[142].

Amazon Cognito, web ve mobil uygulamalar için kimlik doğrulama ve yetkilendirme hizmeti sunmaktadır. Kullanıcı kaydı, giriş ve erişim kontrolü sağlamaktadır. Kullanıcı havuzları ve federasyon kimlik sağlayıcıları ile kimliklerin güvenli biçimde yönetilmesine olanak tanır.

Yerleşik güvenlik ve ölçeklenebilirlik özellikleri ile sıfırdan kimlik doğrulama sistemi geliştirme gerekliliğini ortadan kaldırmaktadır[143].

Amazon DynamoDB, tam yönetilen bir veritabanı hizmetidir. Tek haneli milisaniye düzeyinde gecikme ile ölçeklenebilir performans sunmaktadır. Bölümlendirme, çoğaltma ve yüksek erişilebilirlik işlemlerini otomatik olarak yönetmektedir. Esnek veri modelleri desteklemekte ve diğer AWS servisleriyle sorunsuz entegrasyon sağlamaktadır[144].

Amazon Personalize, gerçek zamanlı kişiselleştirilmiş öneriler oluşturmak için yönetilen bir makine öğrenimi hizmetidir. ML uzmanlığı gerektirmeden kullanıcı etkileşim verileri üzerinde modeller eğitilmesine olanak tanır. Böylece ürün, içerik veya arama önerilerini kullanıcıya özel sunarak katılımı ve müşteri memnuniyetini artırmaktadır[145].

Amazon Bedrock, temel modeller (Foundation Models) için API tabanlı erişim sunarak üretken yapay zekâ (generative AI) uygulamalarının geliştirilmesini sağlamaktadır. Altyapıyı yönetme veya sıfırdan model eğitme gereksinimini ortadan kaldırır. Ölçeklenebilir, güvenli ve maliyet etkin çözümlerle yapay zekâ inovasyonunu hızlandırmaktadır[146].

AWS Glue, veri analitiği ve makine öğrenimi için verileri hazırlayan ve entegre eden tam yönetilen bir ETL hizmetidir. Şemaları otomatik olarak keşfetmekte, dönüşüm kodları üretmekte ve iş akışlarını orkestre etmektedir. Büyük ölçekli veri gölleri ve depoları üzerinde veri süreçlerini basitleştirmektedir[147].

Amazon Kinesis, gerçek zamanlı veri akışı ve analitiği için bir platformdur. Düşük gecikmeyle günlükler, IoT olayları veya tıklama akışları gibi büyük veri akışlarını toplamak, işlemek ve analiz etmek için kullanılmaktadır. Analitik ve depolama hizmetleriyle entegre olarak anlık içgörü gerektiren uygulamaları desteklemektedir[148].

Amazon S3, pratik olarak sınırsız ölçeklenebilirlik, yüksek erişilebilirlik ve dayanıklılık sunan bir nesne depolama hizmetidir. Veriler, benzersiz anahtarlarla tanımlanan nesneler olarak bucket içerisinde depolanmaktadır. Ayrıntılı erişim kontrolü, sürümleme ve yaşam döngüsü politikaları desteği ile güvenli ve maliyet etkin bir depolama sağlamaktadır. Kullanım alanları arasında veri gölleri, yedeklemeler, makine öğrenimi ve analitik uygulamalar bulunmaktadır[149].

Amazon QuickSight, hızlı ve etkileşimli veri görselleştirme için bir iş zekâsı (BI) hizmetidir. Redshift, S3 veya RDS gibi AWS veri kaynaklarına doğrudan bağlanarak paneller, raporlar ve makine öğrenimi destekli içgörüler sunmaktadır. Otomatik ölçeklenebilirliği sayesinde karmaşık altyapılara gerek kalmadan gelişmiş analitiği erişilebilir kılmaktadır[150].

Amazon CloudFormation, AWS tarafından sağlanan ve bulut altyapısını kod ile tanımlamaya imkân veren bir orkestrasyon aracıdır. Temel yaklaşımı, tüm altyapı kaynaklarını **JSON veya YAML formatında** tanımlanmış bir metin dosyası ile modellemek ve bu tanım üzerinden güvenli ve öngörülebilir şekilde kaynakların sağlanmasını (provisioning) ve güncellenmesini sağlamaktır[151]. Bu yaklaşım, yazılım geliştirme dünyasında giderek yaygınlaşan **IaC** paradigmasının AWS ekosistemindeki en belirgin örneklerinden birini oluşturmaktadır.

Amazon Neptune, hızlı, güvenilir ve tam yönetilen bir veritabanı hizmetidir. Özellikle **yüksek derecede ilişkili (highly connected) verilerin** keşfi ve işlenmesi gereken uygulamalar için tasarlanmıştır. Neptune, milyarlarca ilişkiyi depolamak ve milisaniye düzeyinde gecikme ile grafik sorguları gerçekleştirmek üzere optimize edilmiş**, amaç odaklı (purpose-built), ölçeklenebilir bir grafik veritabanı motoru**dur[152].

Amazon EKS (Elastic Kubernetes Service), AWS tarafından sunulan yönetilen bir Kubernetes hizmetidir. Kubernetes, konteyner tabanlı uygulamaların orkestrasyonu için endüstri standardı haline gelmiş olsa da, onu kendi başına kurmak, yönetmek ve sürekli güncel tutmak karmaşık ve operasyonel olarak maliyetli bir süreçtir. EKS bu süreci kolaylaştırarak, kullanıcıların Kubernetes kontrol düzlemini yönetmesine gerek bırakmaz; AWS bu kısmı otomatik olarak üstlenir. Böylece geliştiriciler, Kubernetes kümesini sıfırdan kurmakla uğraşmak yerine doğrudan uygulama geliştirme ve dağıtım süreçlerine odaklanabilirler[153].

Amazon ECR, (Elastic Container Registry / Esnek Konteyner Kayıt Deposu), konteyner imajlarını güvenli ve sürümlü biçimde depoladığın tam yönetilen bir imaj kayıt servisidir. CI/CD akışında imajlar ECR'ye push edilir, çalışma ortamları ECR'den çeker; erişim IAM politikalarıyla kontrol edilir[153].

6.8. Uygulamanın Bulut Ortamına Kurulması

Bu uygulamanın bulut ortamında kurulumu, kapsamlı bir **IaC** yaklaşımı çerçevesinde gerçekleştirilmiştir. Kurulum süreci, **AWS CloudFormation** şablonları kullanılarak uçtan uca otomatikleştirilmiş ve **birden fazla CloudFormation yığını** aracılığıyla modüler bir mimari doğrultusunda tasarlanmıştır.

Bu kapsamlı IaC yaklaşımı sayesinde, **birçok AWS servisi** ve yüzlerce kaynak yalnızca kod tabanlı yönetimle kontrol altına alınmış; böylelikle **yeniden üretilebilir (reproducible), ölçeklenebilir ve sürdürülebilir** bir altyapı elde edilmiştir.

Aşağıdaki tablo kurulum süresini ayrıntılı olarak göstermektedir:

Tablo 6.2 AWS servislerinin kurulum süresi

Yığın İsmi	**Kaynak İsmi**	**Kaynak**	**Süre (saniye)**
s3-bucket-stack	DatasetBucket	AWS::S3::Bucket	14.59
s3-bucket-stack	DatasetKMSKey	AWS::KMS::Key	16.88
s3-bucket-stack	DatasetBucketPolicy	AWS::S3::BucketPolicy	1.72
s3-bucket-stack	DatasetKMSKeyAlias	AWS::KMS::Alias	62.63
data	SeriesTable	AWS::DynamoDB::Table	33.73
data	MoviesTable	AWS::DynamoDB::Table	33.5

data	GlueServiceRole	AWS::IAM::Role	19
data	UserMoviesTable	AWS::DynamoDB::Table	13.03
data	UserSeriesTable	AWS::DynamoDB::Table	12.91
data	TVSeriesGlueJob	AWS::Glue::Job	2.78
data	MoviesGlueJob	AWS::Glue::Job	2.77
network-stack	NatGateway2	AWS::EC2::NatGateway	110.91
network-stack	NatGateway1	AWS::EC2::NatGateway	100.9
network-stack	NatGatewayEIP1	AWS::EC2::EIP	17.07
network-stack	InternetGateway	AWS::EC2::InternetGateway	17.04
network-stack	NatGatewayEIP2	AWS::EC2::EIP	16.95
network-stack	VPC	AWS::EC2::VPC	12.77
network-stack	PrivateRouteTable2	AWS::EC2::RouteTable	11.83
network-stack	PublicRouteTable	AWS::EC2::RouteTable	11.83

network-stack	PrivateRouteTable1	AWS::EC2::RouteTable	11.81
network-stack	PublicSubnet1	AWS::EC2::Subnet	7.22
network-stack	PublicSubnet2	AWS::EC2::Subnet	4.49
network-stack	PrivateSubnet1	AWS::EC2::Subnet	4.44
network-stack	PrivateSubnet2	AWS::EC2::Subnet	4.28
network-stack	InternetGatewayAttachment	AWS::EC2::VPCGatewayAttachment	2.67
network-stack	DefaultPrivateRoute2	AWS::EC2::Route	2.04
network-stack	DefaultPublicRoute	AWS::EC2::Route	1.96
network-stack	PublicSubnet1RTA	AWS::EC2::SubnetRouteTableAssociation	1.91
network-stack	PrivateSubnet2RTA	AWS::EC2::SubnetRouteTableAssociation	1.88
network-stack	DefaultPrivateRoute1	AWS::EC2::Route	1.87

network-stack	PrivateSubnet1RTA	AWS::EC2::SubnetRouteTableAssociation	1.72
network-stack	PublicSubnet2RTA	AWS::EC2::SubnetRouteTableAssociation	1.67
eks	EKSCluster	AWS::EKS::Cluster	668.43
eks	ServicesNodeGroup	AWS::EKS::Nodegroup	312.62
eks	MonitoringNodeGroup	AWS::EKS::Nodegroup	312.61
eks	RecommendationNodeGroup	AWS::EKS::Nodegroup	309.65
eks	GitOpsNodeGroup	AWS::EKS::Nodegroup	307.54
eks	StreamingNodeGroup	AWS::EKS::Nodegroup	288.29
eks	DevOpsNodeGroup	AWS::EKS::Nodegroup	287.44
eks	NodeRole	AWS::IAM::Role	19.18
eks	EKSClusterRole	AWS::IAM::Role	18.63
eks	EKSClusterSecurityGroup	AWS::EC2::SecurityGroup	8.94
eks-add-	CoreDNSAddon	AWS::EKS::Addon	39.42

ons			
eks-add-ons	VPCCNIAddon	AWS::EKS::Addon	39.23
eks-add-ons	KubeProxyAddon	AWS::EKS::Addon	33.94
eks-add-ons	PodIdentityAddon	AWS::EKS::Addon	33.87
eks-add-ons	CloudWatchObservabilityAddon	AWS::EKS::Addon	29
eks-add-ons	MonitoringRole	AWS::IAM::Role	18.73
eks-add-ons	ADOTRole	AWS::IAM::Role	18.65
eks-pod-identity	LBCRole	AWS::IAM::Role	18.88
eks-pod-identity	MonitoringRole	AWS::IAM::Role	18.78
eks-pod-identity	ServicesRole	AWS::IAM::Role	18.72
eks-pod-identity	AIRole	AWS::IAM::Role	18.66
eks-pod-identity	RecommendationRole	AWS::IAM::Role	18.63

eks-pod-identity	StreamingRole	AWS::IAM::Role	18.62
eks-pod-identity	AIExtraPolicy	AWS::IAM::ManagedPolicy	16.99
eks-pod-identity	CommonPolicy	AWS::IAM::ManagedPolicy	16.95
eks-pod-identity	ServicesExtraPolicy	AWS::IAM::ManagedPolicy	16.91
eks-pod-identity	RecommendationExtraPolicy	AWS::IAM::ManagedPolicy	16.91
eks-pod-identity	StreamingExtraPolicy	AWS::IAM::ManagedPolicy	16.87
eks-pod-identity	RecommendationPodIdentity	AWS::EKS::PodIdentityAssociation	2.41
eks-pod-identity	LBCPodIdentity	AWS::EKS::PodIdentityAssociation	2.29
eks-pod-identity	StreamingPodIdentity	AWS::EKS::PodIdentityAssociation	2.27
eks-pod-identity	AIPodIdentity	AWS::EKS::PodIdentityAssociation	2.17

eks-pod-identity	ServicesPodIdentity	AWS::EKS::PodIdentityAssociation	2.1
eks-pod-identity	MonitoringPodIdentity	AWS::EKS::PodIdentityAssociation	2.1
authentication	CognitoUserPool	AWS::Cognito::UserPool	3.05
authentication	CognitoUserPoolClient	AWS::Cognito::UserPoolClient	2.03
authentication	SecretsManagerSecret	AWS::SecretsManager::Secret	1.59
ecr	ConsumerRepository	AWS::ECR::Repository	1.95
ecr	AuthenticationRepository	AWS::ECR::Repository	1.83
ecr	SearchRepository	AWS::ECR::Repository	1.8
ecr	SeriesRepository	AWS::ECR::Repository	1.74
ecr	ProducerRepository	AWS::ECR::Repository	1.73
ecr	MoviesRepository	AWS::ECR::Repository	1.71
ecr	UserRepository	AWS::ECR::Repository	1.68
ecr	PersonalizeRepository	AWS::ECR::Repository	1.65
ecr	FrontendRepository	AWS::ECR::Repository	1.62

ecr	ChatbotRepository	AWS::ECR::Repository	1.52
genai-rag	NeptuneAnalyticsGraph	AWS::NeptuneGraph::Graph	188.48
genai-rag	BedrockKnowledgeBase	AWS::Bedrock::KnowledgeBase	67.58
genai-rag	BedrockAgent	AWS::Bedrock::Agent	19.27
genai-rag	BedrockAgentRole	AWS::IAM::Role	18.86
genai-rag	BedrockAgentAlias	AWS::Bedrock::AgentAlias	8.44
genai-rag	BedrockDataSource	AWS::Bedrock::DataSource	2.24
genai-rag	BedrockAgentCredentials Secret	AWS::SecretsManager::Secret	1.94
quicksight	MoviesAnalysis	AWS::QuickSight::Analysis	63.69
quicksight	SeriesAnalysis	AWS::QuickSight::Analysis	61.12
quicksight	SeriesDashboard	AWS::QuickSight::Dashboard	60.48
quicksight	MoviesDashboard	AWS::QuickSight::Dashboard	59.87
quicksight	MoviesTemplate	AWS::QuickSight::Template	59.74

quicksight	SeriesTemplate	AWS::QuickSight::Template	59.53
quicksight	MoviesDataSet	AWS::QuickSight::DataSet	35.16
quicksight	SeriesDataSet	AWS::QuickSight::DataSet	35.08
quicksight	SeriesDataSource	AWS::QuickSight::DataSource	19.43
quicksight	MoviesDataSource	AWS::QuickSight::DataSource	19.32
kinesis	DeliveryStream	AWS::KinesisFirehose::DeliveryStream	32.79
kinesis	deliveryRole	AWS::IAM::Role	17.24
kinesis	FirehoseLogStream	AWS::Logs::LogStream	12.53
kinesis	EventsKinesisStream	AWS::Kinesis::Stream	12.23
kinesis	FirehoseLogGroup	AWS::Logs::LogGroup	9.89
personalize-series	SeriesItemsDataset	AWS::Personalize::Dataset	403.62
personalize-series	SeriesInteractionsDataset	AWS::Personalize::Dataset	282.9
personalize-series	SeriesDatasetGroup	AWS::Personalize::DatasetGroup	22.47

personalize-series	SeriesSolution	AWS::Personalize::Solution	2.02
personalize-series	SeriesInteractionSchema	AWS::Personalize::Schema	1.6
personalize-series	SeriesItemSchema	AWS::Personalize::Schema	1.39
personalize-movies	MoviesItemsDataset	AWS::Personalize::Dataset	448.2
personalize-movies	MoviesInteractionsDataset	AWS::Personalize::Dataset	267.54
personalize-movies	MoviesDatasetGroup	AWS::Personalize::DatasetGroup	6.5
personalize-movies	MoviesSolution	AWS::Personalize::Solution	1.88
personalize-movies	MoviesItemSchema	AWS::Personalize::Schema	1.32
personalize-movies	MoviesInteractionSchema	AWS::Personalize::Schema	1.21

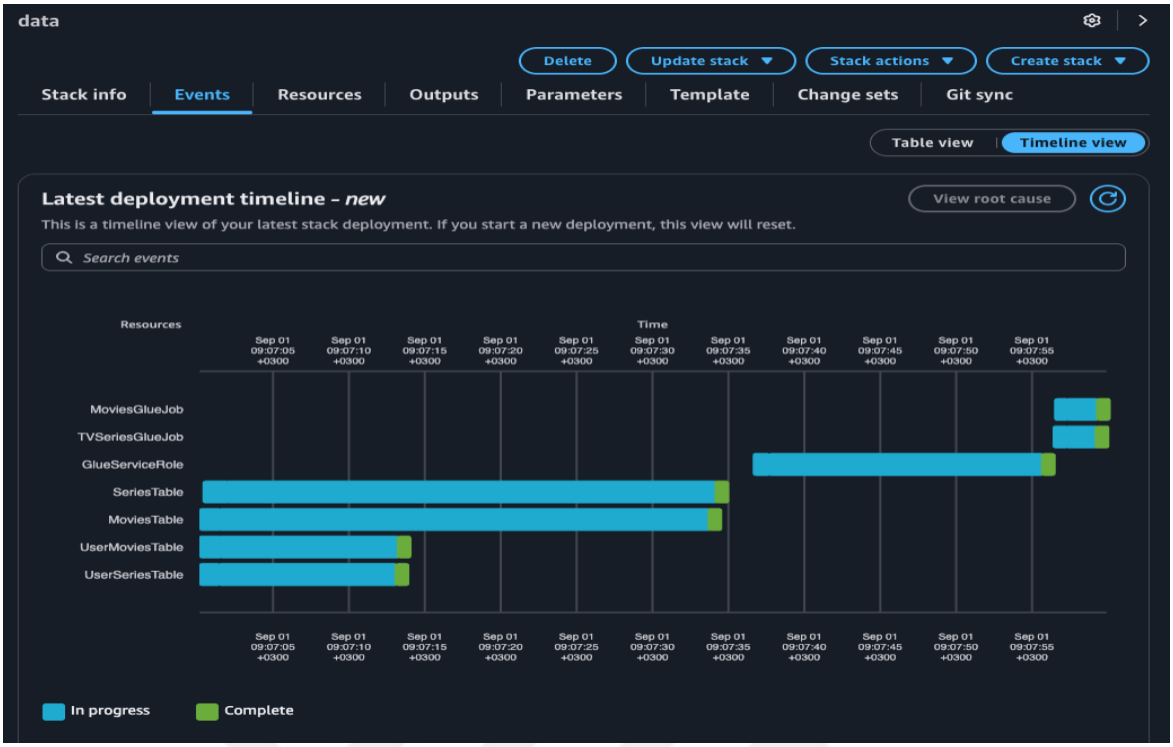
Bu çalışmada elde edilen sonuçlar, bulut tabanlı otomasyonun geleneksel yöntemlere kıyasla sağladığı hız farkını somut biçimde ortaya koymaktadır. CloudFormation şablonları aracılığıyla gerçekleştirilen dağıtımlarda, kaynakların büyük çoğunluğu **saniyeler ile dakikalar** arasında kurulmuş; örneğin Amazon EKS kümesi yaklaşık **668 saniyede (≈11 dakika)**, her bir NodeGroup ise **300 saniye civarında (≈5 dakika)** tamamlanmıştır. Bu değerler, manuel kurulum gerektiren geleneksel ortamlarda raporlanan sürelerle karşılaştırıldığında çarpıcıdır. Nitekim Wang ve Tseng (2024), VMware vSphere üzerinde Kubernetes kümesinin manuel adımlar ve klasik IaC araçları ile hazırlanmasının **75 dakika**

51 saniye sürdüğünü, geliştirdikleri "Golden IaC Framework" sayesinde bu sürenin **18 dakika 7 saniyeye** indirilebildiğini bildirmiştir[154]. Buna ek olarak, başka bir çalışmada, mevcut bare-metal bulut servislerinde (örneğin IBM, Rackspace) işletim sisteminin kurulumu, disk imajının kopyalanması ve ağ üzerinden yapılan yapılandırmalar yüzünden provisioning süreçlerinin "onlarca dakika ila saatler" arasında sürdüğü açıkça tarif edilmiştir[155].

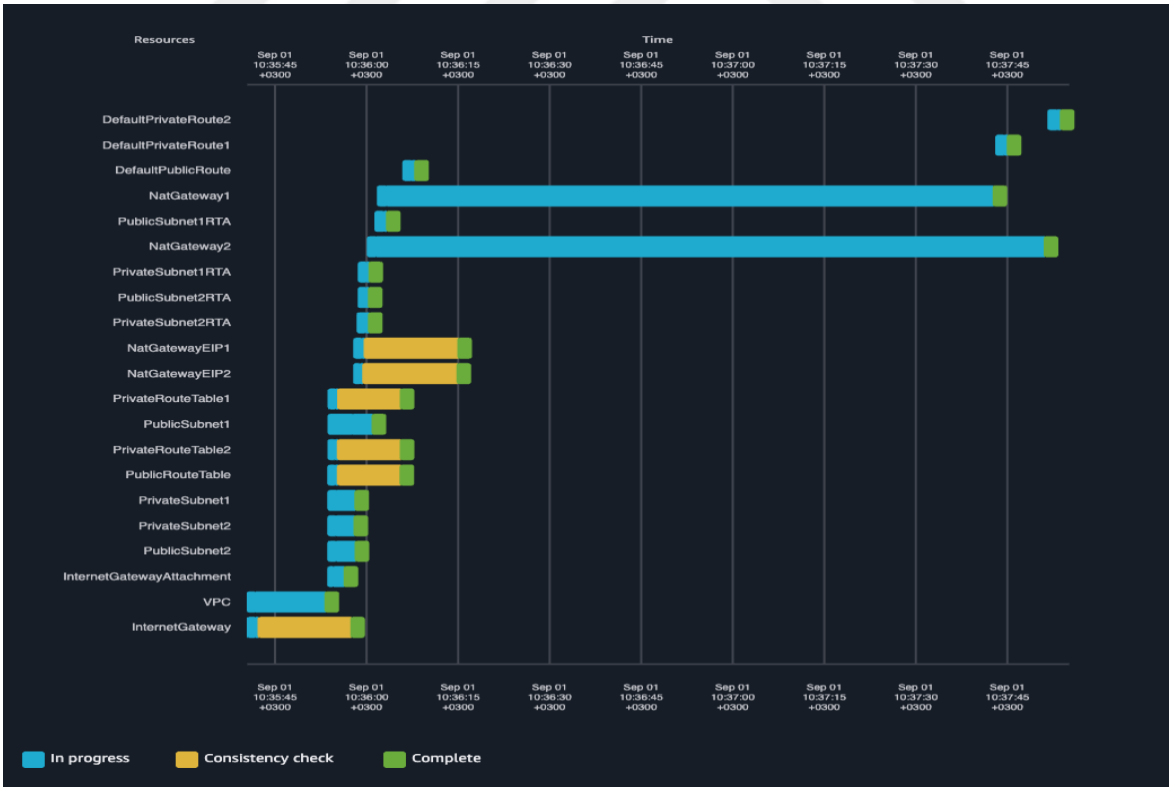
Bu kıyaslamalar, yönetilen servislerin ve serverless mimarilerin yalnızca operasyonel karmaşıklığı azaltmadığını; aynı zamanda altyapı kurulum sürelerini **saatler düzeyinden dakikalar hatta saniyeler düzeyine** indirerek modern yazılım geliştirme süreçlerinde radikal bir verimlilik artışı sağladığını göstermektedir.

Tez kapsamında gerçekleştirilen ölçümler, depolama servislerinin kurulumu açısından da serverless ve tam otomasyonun sağladığı hız avantajını ortaya koymaktadır. Örneğin Amazon S3 kovalarının (bucket) saniyeler içinde oluşturulabilmesi veya DynamoDB tablolarının yarım dakikanın altında hazır hale gelmesi, geleneksel veri depolama süreçlerinde gözlenen dakikalar–saatler seviyesindeki gecikmelere kıyasla radikal bir farktır. Bu durum, bulut tabanlı yönetilen servislerin yalnızca operasyonel kolaylık sağlamadığını, aynı zamanda depolama altyapısının **hızlı, ölçeklenebilir ve minimum insan müdahalesiyle** devreye alınabilmesini mümkün kıldığını göstermektedir.

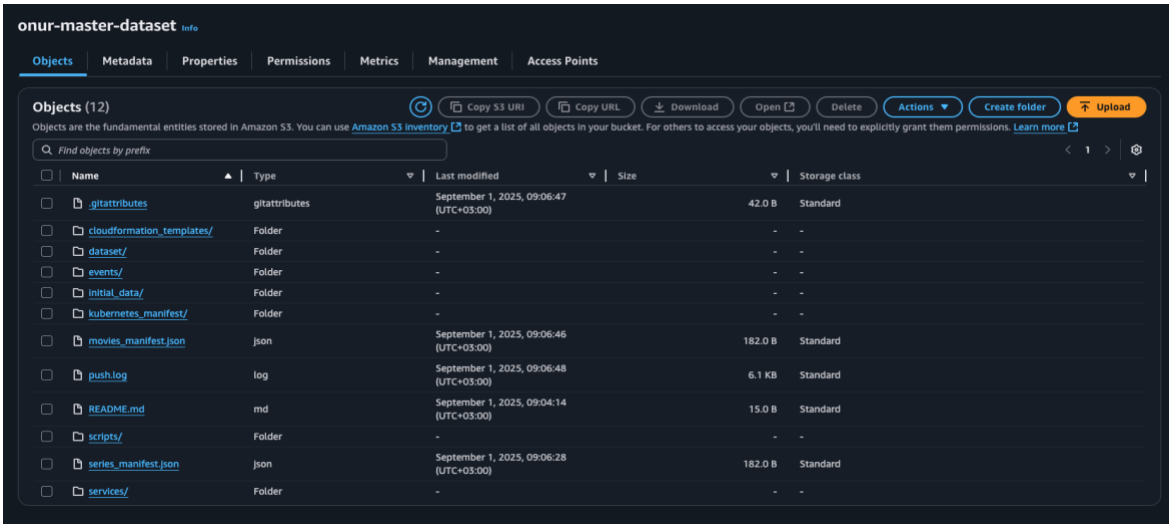
Aşağıdakiler şekiller oluşturulan kaynakların ekran görüntülerini göstermektedir:



Şekil 6.10 Data yığınına ait ekran görüntüsü



Şekil 6.11 Ağ bileşenlerinin kurulum süresi



Şekil 6.12 S3 kovasına ait ekran görüntüsü

6.9. Uygulamanın Kubernetes'e Kurulumu

Uygulama, AWS EKS üzerinde dağıtılmıştır. EKS kümesi, Infrastructure as Code yaklaşımı kapsamında CloudFormation şablonları kullanılarak inşa edilmiştir. Küme, Kubernetes 1.31 sürümü üzerinde çalışmakta olup, otomatik ölçeklenebilir düğüm grupları sayesinde dinamik kaynak yönetimi sağlamaktadır.

Servisler, mantıksal olarak farklı ad alanlarında (namespace) organize edilmiştir. Services ad alanı, temel uygulama servislerini (kimlik doğrulama, ön yüz, film, dizi, arama ve kullanıcı) içermektedir. AI ad alanında sohbet botu servisi, recommendation ad alanında Personalize servisi, streaming ad alanında ise Kinesis servisi bulunmaktadır. Her bir ad alanı, kendine özgü güvenlik politikaları ve kaynak kotaları ile yönetilmektedir.

Her servis için ayrı dağıtım manifest dosyaları oluşturulmuştur. Örneğin kimlik doğrulama servisi authentication.yml, film servisi *movies.yml* ve dizi servisi series.yml dosyaları ile tanımlanmıştır. Her bir dağıtım kapsayıcı imajı; kaynak limitleri, ortam değişkenleri ve sağlık kontrollerini içerecek şekilde yapılandırılmıştır.

Servisler arasındaki iletişim Kubernetes Service nesneleri üzerinden sağlanmaktadır. İç servisler ClusterIP türünde, dış dünyaya erişim sağlayan servisler ise LoadBalancer türünde yapılandırılmıştır. Birleşik yönlendirme ve SSL sonlandırma Ingress Controller aracılığıyla gerçekleştirilmiştir.

Kapsayıcı imajları, AWS ECR (Elastic Container Registry) üzerinde yönetilmektedir. Her servis için ayrı bir depo oluşturulmuş, sürümler etiketleme yöntemiyle takip edilmektedir. CI/CD boru hatları, yeni imajları otomatik olarak inşa edip ECR'a yüklemektedir.

Yapılandırma yönetimi Kubernetes ConfigMap ve Secret nesneleri ile sağlanmaktadır. Veritabanı bağlantı dizeleri, API anahtarları ve ortama özgü ayarlar ConfigMap'ler ile; hassas bilgiler ise şifrelenmiş biçimde Secret nesneleri ile saklanmaktadır.

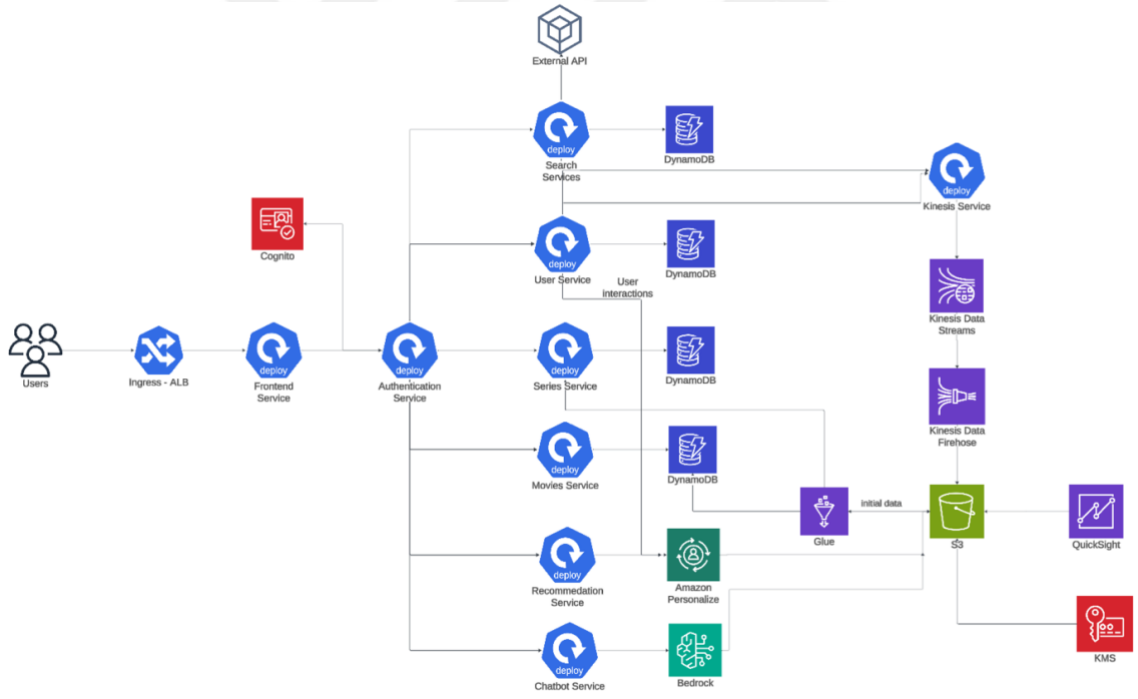
Dış depolama için AWS'nin yönetilen servisleri olan DynamoDB ve S3 entegre edilmiştir. Kubernetes yerel izleme mekanizmaları pod sağlık kontrollerini ve kaynak kullanımını izlerken, merkezi günlükleme AWS CloudWatch üzerinden sağlanmaktadır. Kubernetes olayları ve metrikleri CloudWatch'a aktarılmaktadır. Liveness ve readiness prob'ları sayesinde otomatik iyileştirme (self-healing) gerçekleştirilmiştir.

Son olarak, Kubernetes RBAC (Role-Based Access Control / Rol Tabanlı Erişim Denetimi) ile ayrıntılı yetkilendirme sağlanmış, Service Account kullanımı ile pod düzeyinde güvenlik uygulanmıştır.



Şekil 6.13 Kubernetes altyapısı

Şekil 6.13'te Kubernetes kümesi üzerinde çalışan uygulama servisleri ile bunlara bağlı bulut servisleri arasındaki etkileşim gösterilmektedir. Kullanıcılar Ingress–ALB üzerinden frontend servisine erişmekte; kimlik doğrulama süreçleri Authentication Service ve Cognito ile yürütülmektedir. Frontend, istekleri arka uçtaki mikroservislere (Search, User, Series, Movies, Recommendation, Chatbot) yönlendirmekte; her bir servis kendi verisini ilgili DynamoDB tablolarında saklamakta veya harici API'ler ile bütünleşmektedir. Kullanıcı etkileşimleri User Service üzerinden hem DynamoDB'ye yazılmakta hem de Kinesis Service aracılığıyla Kinesis Data Streams ve Kinesis Data Firehose hattına aktarılmakta, Glue ile işlenerek S3 üzerinde depolanmakta ve QuickSight ile görselleştirilmektedir. Movies Service ve Recommendation Service, Amazon Personalize ile kişiselleştirilmiş öneriler üretirken, Chatbot Service Bedrock üzerinden büyük dil modeli tabanlı sohbet işlevini sağlamaktadır. KMS, S3 ve diğer ilgili bileşenlerin şifreleme anahtarlarını yöneterek veri güvenliğini desteklemektedir.



Şekil 6.14 Kubernetes servislerinin AWS servisleriyle olan etkileşimi

Aşağıdaki tablo Kubernetesler node'larının özetini sağlamaktadır:

Tablo 6.3 Kubernetes'e ait nodegroup (sanal makine) yapısı

İsim	Sanal makine tipi	CPU	Bellek	Etiket	Açıklama
services-nodegroup	t3.xlarge	4	16 GB	node-type: services	Ana mikroservis node grubu (kimlik doğrulama, kullanıcı, film, dizi, arama servisleri)
devops-nodegroup	t3.xlarge	4	16 GB	node-type: devops	DevOps araçları node grubu (CI/CD pipeline'ları, izleme, GitHub runner'ları)
gitops-nodegroup	t3.medium	2	4 GB	node-type: gitops	GitOps iş akışı ve ArgoCD node grubu
recommendation-nodegroup	t3.medium	2	4 GB	node-type: recommendation	Makine öğrenimi servisleri ve Personalize entegrasyonu node grubu
streaming-nodegroup	t3.medium	2	4 GB	node-type: streaming	Kinesis üretici/tüketici servisleri node grubu
ai-nodegroup	t3.large	2	8 GB	node-type: ai	Yapay zekâ servisleri ve chatbot için ayrılmış node grubu (Amazon Bedrock entegrasyonu)

Kubernetes'te kullanılan sanal makineler yukarıdaki tabloda verilmiş olup; sanal makine tipi ücret açısından makul değerlerde seçilmiştir.

6.10. DevOps Otomasyonu

DevOps otomasyonları, manuel müdahale ihtiyacını ortadan kaldırarak yazılım teslimat süreçlerini hızlandırır ve daha güvenilir hale getirir. Bu süreç, kod değişiklikleri algılandığında otomatik olarak başlar ve kesintisiz test, güvenlik kontrolleri ve dağıtımın gerçekleştirilmesini sağlar. Otomasyon yalnızca zaman tasarrufu sağlamakla kalmaz, aynı zamanda süreçlerin standartlaştırılmasını ve tekrarlanabilir olmasını da garanti eder. Her aşama için gereken sürenin azaltılması, ekiplerin daha hızlı geri bildirim almasına ve yeni özellikleri kullanıcılara daha kısa sürede sunmasına imkân tanır.

Şekil 6.13, DevOps otomasyonun görsel halini içermektedir.



Şekil 6.15 DevOps Otomasyon Mimarisi

Süreç, geliştiricinin GitHub'a kodunu yüklemesi (commit) ile başlar ve hazırlık aşaması olarak tanımlanabilecek bir dizi adımdan geçer. Bu aşamada öncelikle kaynak kod çalışma ortamına çekilir (checkout), ardından AWS kimlik bilgileri yapılandırılır ve gerekli araçlar (Python, Docker, Kubernetes ve AWS çalışma ortamı gibi) yüklenerek çalışma ortamı hazırlanır. Her ne kadar bu hazırlık aşaması tek bir adım gibi görünse de, Otomasyonun temelini oluşturan çoklu süreçlerden meydana gelir ve sonraki adımların kesintisiz çalışması için zorunludur.

Hazırlığın ardından Pipeline kalite kontrol aşamasına geçer. Bu aşama, yazılımın güvenilirliğini ve sürdürülebilirliğini sağlamak açısından kritik bir rol oynar. Öncelikle birim testleri (Unit Tests) çalıştırılır ve kodun temel işlevselliğinin korunup korunmadığı doğrulanır. Ardından test kapsamı ölçülür ve böylece kodun büyük kısmının testlerle kapsanıp kapsanmadığına dair nicel bir kriter elde edilir. Bu süreci statik kod analizi takip eder; burada Sonarqube gibi araçlar devreye alınır ve kod kalitesi, güvenlik açıkları ve en iyi uygulama standartlarına uyumluluk denetlenir. Tüm bu adımların amacı yazılımın yalnızca çalışır durumda olmasını değil, aynı zamanda güvenli, sürdürülebilir ve uzun vadeli bakım yapılabilir olmasını sağlamaktır.

Kalite kontrollerinin ardından pipeline, inşa ve güvenlik aşamasına girer. Uygulama bir Docker imajı olarak paketlenir (Build Docker Image) ve ardından Güvenlik Taraması (Security Scan) yapılır. Bu noktada Trivy gibi araçlar imaj içerisindeki potansiyel güvenlik açıklarını tarar. Bu adım, tedarik zinciri güvenliği açısından hayati öneme sahiptir; çünkü güvenlik açısından zayıf bir imajın hayata geçirilmesi tüm sistem için ciddi riskler taşır. Daha sonra Yük Testi (Load Test) ile sistemin yük altında nasıl davrandığı ölçülür. Bu test yalnızca işlevsel doğrulamanın ötesine geçerek yazılımın ölçeklenebilirliği ve dayanıklılığını değerlendirir.

Pipeline'ın bir sonraki aşaması dağıtım ve teslimdir. Güvenliği sağlanmış ve test edilmiş Docker imajı AWS ECR'ye gönderilir (Push to ECR). Bu merkezi depo, imajın EC2 tarafından alınarak kullanılabilmesi için zorunlu bir saklama alanıdır. Ardından yeni sürüm EKS üzerinde dağıtılır (Deploy to Kubernetes). Bu aşama, sürekli teslim vizyonunun gerçekleştiği noktadır; geliştiricinin commit'inden üretim ortamına kadar olan yol tamamen otomatik hale gelmiştir.

Son aşama ise tamamlama ve temizlik (post) aşamasıdır. Burada gereksiz kaynaklar temizlenir, güvenlikle ilgili son adımlar atılır ve otomasyon kapatılır. Görünüşte basit gibi dursa da bu adım CI/CD süreçlerinde tekrarlanabilirlik ve sürdürülebilirlik açısından kritik öneme sahiptir; çünkü sonraki çalıştırmalar için hatasız bir başlangıç ortamı sağlar.

Bu bütüncül yapı sayesinde otomasyon, yalnızca bir dağıtım aracı olmanın ötesine geçerek yazılım yaşam döngüsünün her noktasına değer katar. Hazırlık aşaması, kalite kontrol, güvenlik ve performans testleri, inşa ve dağıtım ile ardından gelen temizlik süreci sayesinde sistem hem hız hem de güvenilirlik açısından optimize edilir. Dolayısıyla bu DevOps otomasyonu, yazılım geliştirme ve teslimat süreçlerinin otomasyonunun ötesinde, güvenlik, kalite ve performans boyutlarını da entegre eden çağdaş yazılım mühendisliği yaklaşımlarına akademik olarak entegre bir örnek sunmaktadır.

Çalıştırılan bütün adımlar aşağıdaki tabloda özetlenmiştir:

Tablo 6.4 DevOps otomasyonunda kullanılan testler ve açıklamaları

Adım İsmi	**Adım Numarası**	**Açıklaması**
Setup Job (**Job Kurulumu**)	1	Depoyu alır ve GitHub Actions runner'ını hazırlar.
Checkout (**Checkout**)	2	Kod deposunu runner'a indirir ve tüm dosyalara erişim sağlar.
Configure AWS Credentials (**AWS Kimlik Bilgilerini Yapılandır**)	3	AWS ile kimlik doğrulaması yapar ve ECR ile EKS erişimini sağlar.
Setup Environment (**Ortam Kurulumu**)	4	Python, Docker, kubectl ve AWS CLI gibi gerekli araçları yükler.
Unit Tests (**Birim Testleri**)	5	Servisin birim testlerini çalıştırır ve kod kalitesini doğrular.
Code Coverage (**Kod Kapsamı**)	6	Test kapsamı raporu oluşturur ve kodun ne kadarının test edildiğini ölçer.
Static Code Analysis (**Statik Kod Analizi**)	7	SonarQube ile kod kalitesi, güvenlik açıkları ve en iyi uygulamalar kontrol edilir.
Build Docker Image (**Docker İmajı Oluştur**)	8	Servisi paketler ve Docker imajı haline getirir.

Security Scan (**Güvenlik Taraması**)	9	Trivy ile Docker imajındaki güvenlik açıklarını tarar.
Load Test (**Yük Testi**)	10	Locust ile performans testi yapar, servisin yük altındaki davranışını ölçer.
Push to ECR (**ECR'ye Yükle**)	11	Docker imajını AWS ECR deposuna yükler.
Deploy to Kubernetes (**Kubernetes'e Yükle**)	12	Servisi EKS'e dağıtır ve yeni imajla günceller.
Post (**Son İşlemler**)	13	Dağıtım tamamlanır, güvenlik temizliği yapılır ve CI/CD döngüsü kapatılır.

Aşağıdaki tablo, her servisin otomasyon adımları için harcanan süreyi saniye cinsinden göstermektedir. Kod yüklemeleri (commit) eşzamanlı olarak gönderilmiş olsa da, servisler sırasıyla dağıtılmıştır ve her satır ilgili servisin adım adım işlem sürelerini göstermektedir. Görüldüğü üzere birçok adım saniyeler içinde tamamlanırken, bazı adımlar daha uzun sürmektedir. Bu yapı, adımların büyük ölçüde otomatikleştirildiğini, manuel müdahale ihtiyacını ortadan kaldırarak süreci önemli ölçüde kısalttığını ortaya koymaktadır. Sonuç olarak, servis geliştirme ve test aşamasından canlı dağıtıma kadar geçen süre geçmişe kıyasla önemli ölçüde hızlanmış, hızlı teslimat ve güncellemeler mümkün hale gelmiştir.

Tablo 6.5 Bütün Servislerin aynı anda çalıştırmasına ait çalışma süreleri

Servisler	**Adımlar**												
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
Frontend	1	2	1	14	24	2	16	344	31	31	27	17	1

Authentication	1	1	1	20	16	1	14	11	7	31	13	10	1
User	1	1	1	14	3	2	14	58	22	31	14	9	1
Movie	1	1	1	14	1	1	10	33	55	31	17	13	1
Series	1	2	1	14	1	1	11	48	142	31	15	13	0
Search	1	2	1	16	7	0	20	31	26	31	31	16	0
Chatbot	1	2	0	14	9	7	14	14	4	31	6	24	1
ML	1	2	0	52	9	2	14	47	15	31	16	19	1
Producer	2	2	0	14	4	0	26	36	6	31	11	34	1
Consumer	2	1	1	14	6	0	19	39	11	31	15	36	0

İlk çalıştırmada, özellikle bazı servislerde adımlar yüzlerce saniyeye kadar sürerken, ikinci çalıştırmada commit'ler eşzamanlı yapılmasına rağmen toplam otomasyon süresinde önemli bir azalma gözlemlenmiştir. Tablo incelendiğinde birçok adımın saniyelere düştüğü, maksimum sürelerin ise önceki çalıştırmaya kıyasla belirgin şekilde azaldığı görülmektedir. Bu fark, GitHub ortamında önbellekleme (caching) mekanizmalarının uygulanması sayesinde elde edilmiştir; bu sayede bağımlılıklar yeniden indirilmek yerine önbellekten okunmuştur. Bu yaklaşım, her servisin çok daha hızlı tamamlanmasını sağlamış, tüm otomasyonun bitiş süresini önemli ölçüde azaltmış ve sürüm (release) sürecini ciddi anlamda hızlandırmıştır. Bu durum özellikle sürekli dağıtım gerçekleştiren ekipler için kritik bir avantaj sunmakta, her commit'ten sonraki bekleme süresini en aza indirerek geliştirme hızını doğrudan artırmaktadır.

Tablo 6.6 Bütün servislerin aynı anda tekrar çalışmasına ait otomasyon süreleri

Servisler	Adımlar												
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
Frontend	1	2	0	26	4	2	16	9	19	31	12	9	1
Authentication	1	1	1	19	3	1	13	8	13	31	12	10	0
User	1	2	0	15	4	2	14	8	58	31	13	9	1
Movie	1	1	1	14	2	1	10	39	34	31	15	11	0
Series	1	2	0	14	2	1	10	33	41	31	13	11	0
Search	2	1	1	16	7	0	19	37	14	31	23	12	1
Chatbot	1	2	0	21	9	7	12	4	4	31	3	24	0
ML	2	1	1	15	3	2	16	13	15	31	15	15	1
Producer	2	2	0	15	4	0	25	12	12	31	3	24	1
Consumer	2	2	0	16	5	0	19	22	12	31	16	35	0

Otomasyonun ikinci çalıştırılmasında, toplam süre ilk çalıştırmaya kıyasla yaklaşık %33 oranında azaltılmıştır. Bu azalma, özellikle tekrar eden işlemler için önbelleklerin (cache) kullanılması sayesinde sağlanmış ve adımların çok daha hızlı tamamlanmasına olanak tanımıştır.

Aşağıdaki tablo, servis otomasyon süreçlerinin tek başına çalıştırıldığında bile ölçülebileceğini göstermektedir. Başka bir deyişle, bir otomasyon yalnızca tüm servislerin ardışık olarak zincirlenmesiyle değil, gerektiğinde her servisin bağımsız olarak dağıtılmasıyla da ilerleyebilir. Bu durum, DevOps'un esnek yapısının en önemli göstergelerinden biridir: ekipler hem toplu dağıtımları hem de yalnızca tek bir servise özgü güncellemeleri gerçekleştirebilir ve her iki senaryoda da otomasyonun gücünden faydalanabilir.

Tablo 6.7 Servislerin ayrı ayrı güncellenmesi

Servisler	Adımlar												
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13
Frontend	2	1	0	20	3	1	13	9	9	31	13	10	1
Authentication	2	1	1	14	9	7	12	1	3	31	2	24	1
User	1	1	1	15	4	2	15	10	24	31	13	10	0
Movie	2	1	1	23	5	9	19	24	18	31	18	24	1
Series	2	1	0	16	4	0	26	12	13	31	3	25	1
Search	2	1	1	16	3	2	14	15	18	21	15	17	1
Chatbot	1	2	0	16	2	1	12	36	51	31	15	11	1
ML	2	1	1	23	12	0	23	43	30	31	26	13	1
Producer	1	1	1	14	2	0	12	37	49	31	16	10	0
Consumer	2	1	1	24	5	3	17	9	57	31	16	10	0

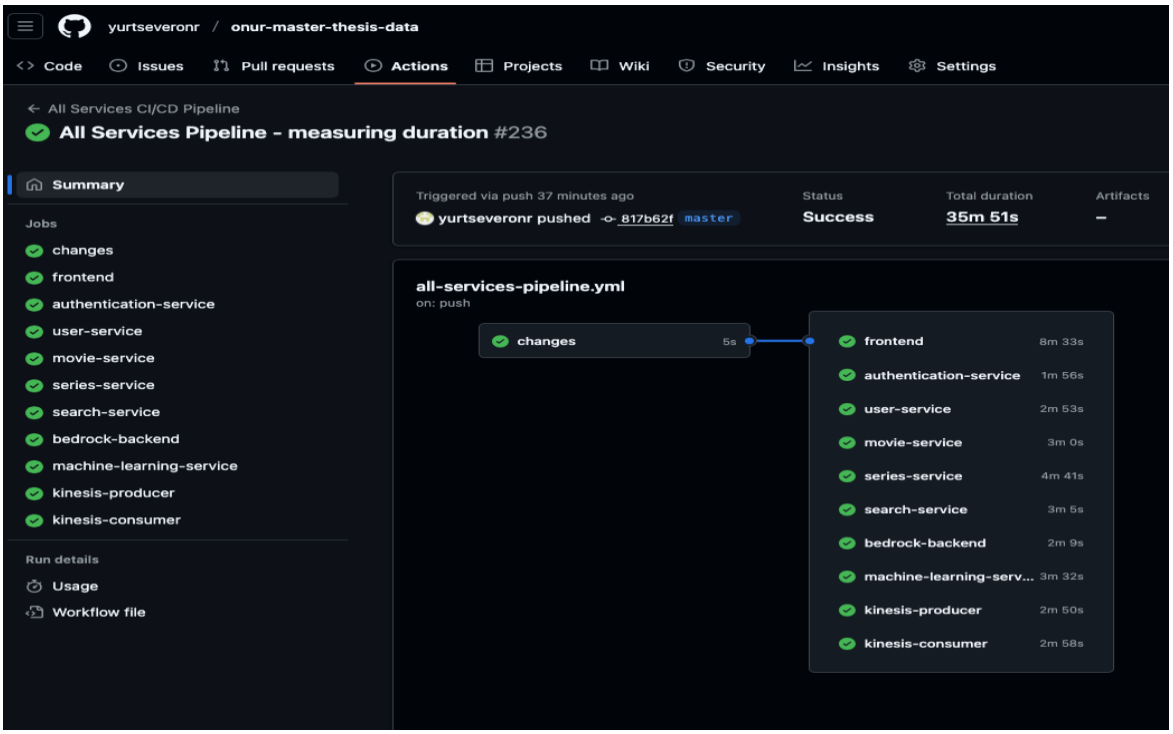
Mikroservis mimarisi ile birleştiğinde bu yaklaşım, her servisin bağımsız olarak geliştirilip test edilmesine ve yalnızca ilgili bileşenin canlıya alınmasına imkân tanır. Böylece tüm sistemi etkilemeden hızlı güncellemeler yapılabilir. Tablo, yalnızca tek bir servisin canlıya alınmasının da hızlı ve güvenilir bir şekilde yapılabildiğini, tüm testler uygulanmasına rağmen dağıtım sürelerinin kontrol altında kaldığını ortaya koymaktadır. Sonuç olarak, DevOps ve mikroservis mimarisinin birleşimi, hem bütünsel pipeline'lar hem de bireysel servis dağıtımları için hızlı geri bildirim, güvenilir testler ve kısa canlıya geçiş süreleri sağlayarak geliştirme ekiplerinin çevikliğini en üst düzeye çıkarmaktadır.

Firefox'un DevOps öncesi döneminde büyük sürümler 1+ yıl geliştirme süreci sonrası manuel işlemlerle canlıya çıkarılıyordu. Hızlı sürüm modeline geçişte bile minimum 18 hafta alıyordu ve tüm süreç otomasyonsuz, elle yönetiliyordu[156] .

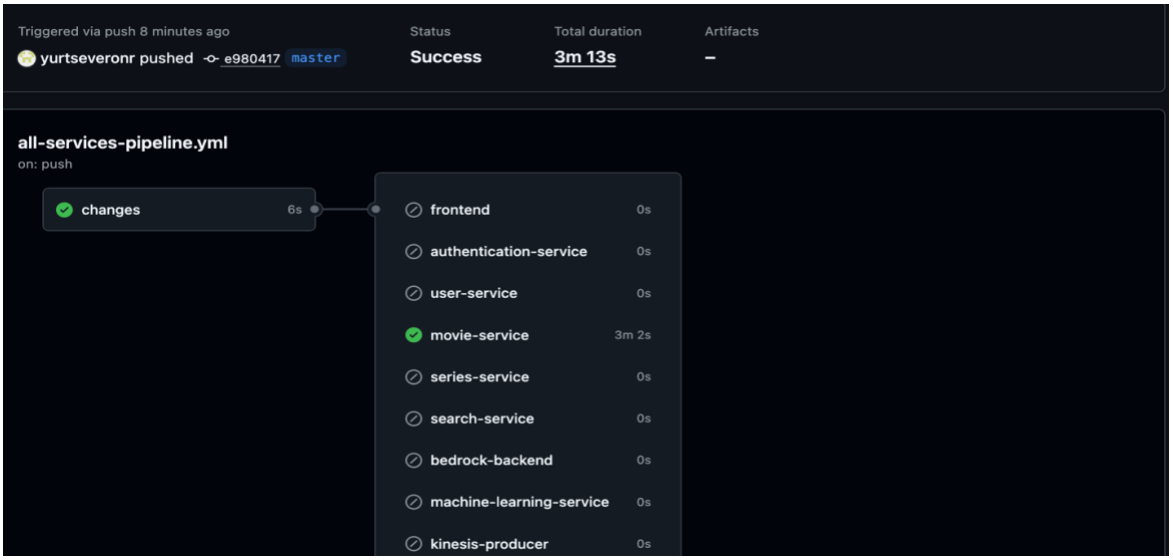
DevOps'un en büyük gücü, uçtan uca yazılım geliştirme süreçlerini otomatikleştirme becerisinde yatmaktadır; bu da hem hız hem de güvenilirlik sağlar. Bir otomasyon çalışmaya başladığında, testler, güvenlik kontrolleri ve dağıtım adımları sırasıyla otomatik olarak uygulanır ve manuel müdahale ihtiyacı ortadan kalkar. Bu sayede hatalar daha erken tespit edilir, ekipler her commit'ten sonra hızlı geri bildirim alabilir ve canlıya geçiş süresi önemli

ölçüde kısalır. Sonuç olarak DevOps, yalnızca geliştirme hızını artırmakla kalmaz, aynı zamanda yazılımın kalite ve güvenilirliğini de yükselterek sürekli teslimat kültürünün benimsenmesini mümkün kılar.

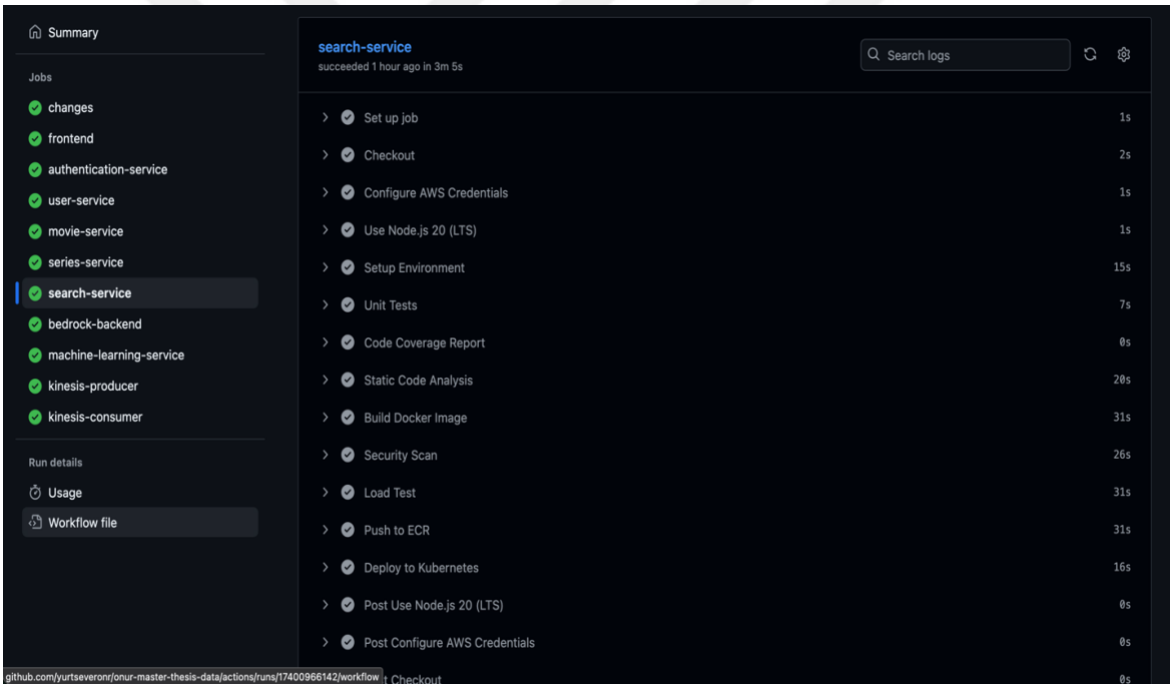
Aşağıdaki ekran görüntüleri bu otomasyonlara ait ekran görüntüleri sunmaktadır.



Şekil 6.16 İlk otomasyona ait ekran görüntüsü



Şekil 6.17 Film servisine ait otomasyon



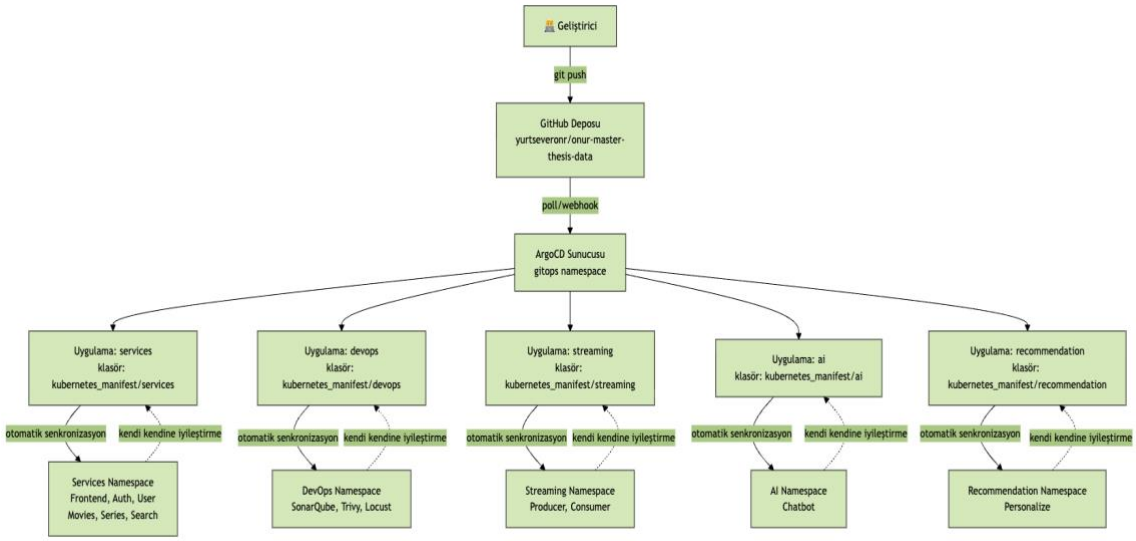
Şekil 6.18 Arama servisine ait otomasyon

6.11. GitOps Otomasyonu

GitOps yaklaşımı, altyapı ve uygulama yönetimi için versiyon kontrol sistemlerini kullanarak operasyonel süreçleri otomatikleştirir. Bu modelde, bir Git deposunda yapılan değişiklikler otomatik olarak algılanır ve Argo CD gibi araçlar sayesinde sürekli senkronizasyon ile Kubernetes ortamına uygulanır. Bu otomasyon, manuel müdahale ihtiyacını azaltır, dağıtım

süreçlerini hızlandırır, hataları en aza indirir ve güvenilirliği artırır. Standardize edilmiş ve tekrarlanabilir süreçler, ekiplerin daha hızlı geri bildirim almasını ve canlı ortama yapılan değişikliklerin hızlanmasını sağlar.

Aşağıdaki şekil GitOps mimarisini göstermektedir:



Şekil 6.19 GitOps mimarisi

GitOps uygulaması, her birinin belirli bir servisten sorumlu olduğu birden çok klasörü sürekli olarak izleyen Argo CD ile modüler bir şekilde yapılandırılmıştır. Main klasörü ortak servislerin güncellenmesine ayrılmışken, Services klasörü uygulamaya özgü servisleri yönetir. DevOps klasörü, DevOps ile ilgili araçların her zaman senkronize edilmesini sağlarken, Streaming klasörü akış servislerine yönelik güncellemeleri yönetir. Benzer şekilde Recommendation klasörü, öneri tabanlı bileşenlerden sorumludur ve AI klasörü yapay zekâ ile ilgili servislerin güncellemelerini uygular. Bu mimari sayesinde her işlevsel alan bağımsız olarak gelişebilirken, ArgoCD Git deposuna işlenen her değişikliğin ilgili Kubernetes kaynaklarına otomatik olarak uygulanmasını garanti eder. Bu modüler tasarım yalnızca karmaşık sistemlerin yönetimini basitleştirmekle kalmaz, aynı zamanda ekiplerin tek tek alanları güncellemesine ve ölçeklendirmesine olanak tanıyarak tüm sistemi etkilemeden dağıtımları hızlandırır.

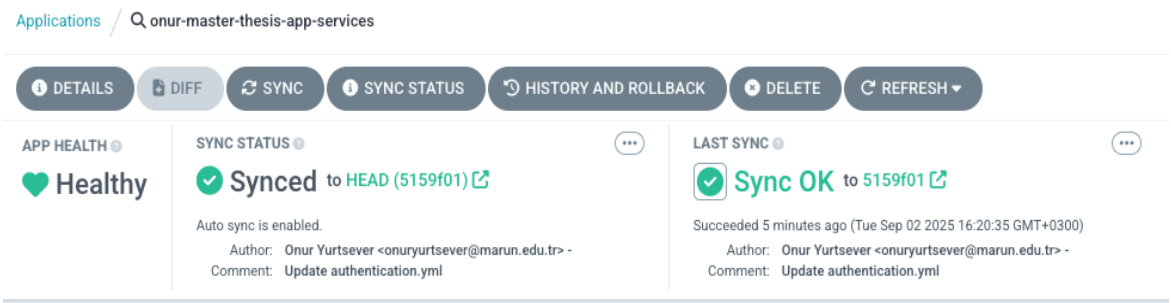
GitOps sürecini göstermek amacıyla bir kullanım senaryosu sunulmaktadır.

Bu GitOps kullanım örneğinde, süreç bir Kubernetes bildiriminde yapılan bir değişiklikle, özellikle authentication.yml dosyasındaki pod sayısının güncellenmesiyle başlar. 5159f01 olarak tanımlanan commit, 16:19:59 UTC+0300'de oluşturuldu ve işlem hattının başlangıç noktasını işaret etti.



Şekil 6.20 GitOps sürecinin ilk aşaması – Güncellemenin yapılması

Değişiklik depoda doğrulandıktan sonra, Argo CD'si değişimi otomatik olarak algılar ve kümeyi istenen yeni duruma senkronize eder. Argo CD'si kontrol paneli, uygulama sağlığını sağlıklı ve senkronizasyon durumunu aynı taahhüt revizyonuna senkronizasyon olarak onaylar ve 16:20:35 UTC+0300'de tamamlanır.



Şekil 6.21 GitOps sürecinin ikinci aşaması – Uygulamanın senkronize edilmesi

Son olarak, Kubernetes tarafındaki doğrulama, güncellemeyi (commit) doğrudan canlı ortama bağlayan zaman damgası 2025-09-02T13: 20: 36Z ile yeni bir kimlik doğrulama bölmesi oluşturulduğunu göstermektedir.

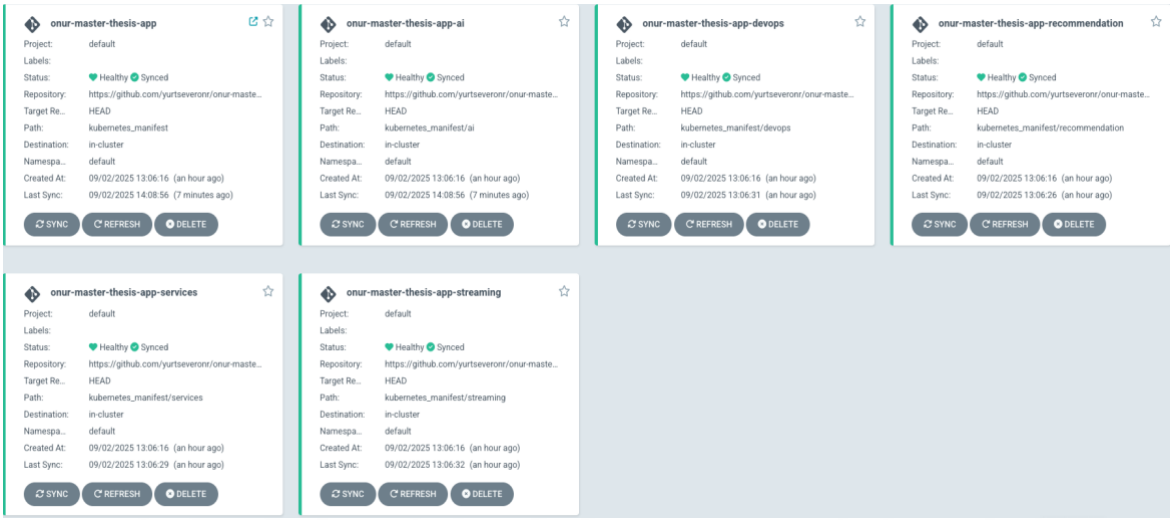


Şekil 6.22 GitOps sürecinin üçüncü adımı – Yeni podların ayağa kalkması

GitOps, yazılım geliştirme ekipleri tarafından yapılan her değişikliğin doğrudan canlı ortama hızlı ve güvenilir bir şekilde yansıtılmasını sağlayan bir yaklaşımdır. Bir geliştirici kodda bir güncelleme yaptığında veya tezahür ettirdiğinde, bu değişiklik otomatik olarak algılanır ve

senkronize edilir ve sistemi anında güncellemek için Kubernetes ortamında yeni baklalar oluşturulur. Bu şekilde, taahhüt anından itibaren yapılan herhangi bir değişiklik, dakikalar beklemek yerine saniyeler içinde yaşar. GitOps'un en önemli yararı, manuel işlemleri ortadan kaldırması, süreci hızlandırması, hataları azaltması ve ekiplerin sürekli güncellenmesini sağlamasıdır. Bu, hem geliştirme hem de operasyon tarafları üzerinde daha çevik bir çalışma sağlar ve serbest bırakma işlemi güvenli ve hızlı bir şekilde tamamlanır.

Şekil 6.23 ArgoCD'den ekran görüntülerini göstermektedir.

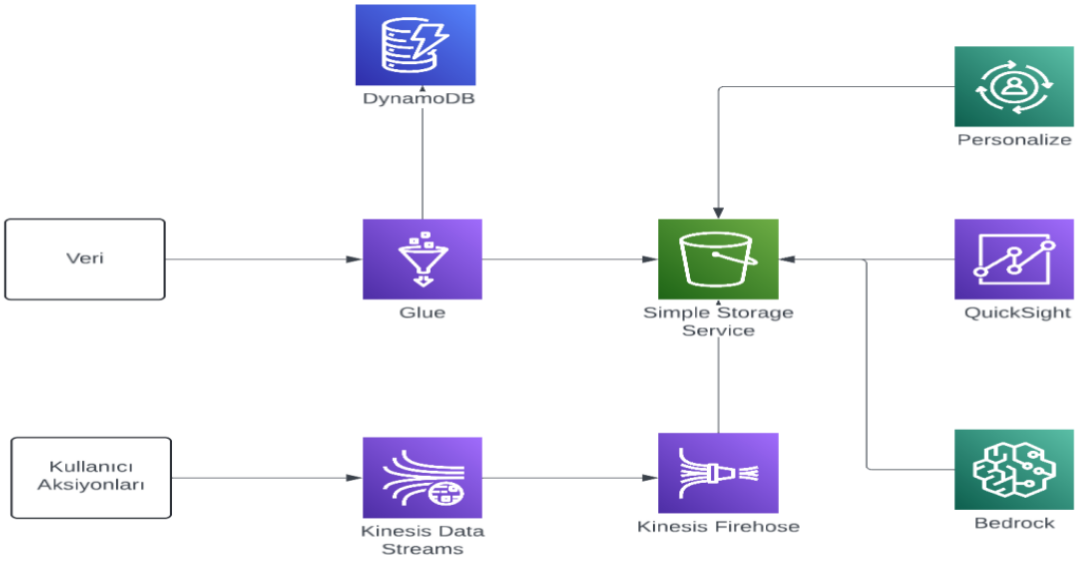


Şekil 6.23 ArgoCD ekran görüntüsü

6.12. Veri ve Yapay Zeka Servislerinin Kurulumları

Şekil 6.24, sistemin veri katmanı ile yapay zekâ uygulamalarının tamamını göstermektedir. Çözüm, bulut bilişim ortamında sunucusuz (serverless) hizmetler kullanılarak geliştirilmiştir. Böylece kullanıcılar, sunucu ve altyapı yönetimi ayrıntılarına hâkim olmadan, yalnızca bulut hizmetleri bilgisiyle ölçeklenebilir bir veri ve yapay zekâ altyapısı

kurup işletebilmektedir.



Şekil 6.24 Veri ve Yapay Zeka mimarisi

AWS Glue Sunucusuz ETL Pipeline, 834.018 ham film kaydını 314.270 üretime hazır girdiye ve 152.937 dizi kaydını 64.196 optimize edilmiş veri setine dönüştürerek olağanüstü bir veri işleme verimliliği sergilemektedir. Bu %62'lik veri optimizasyon oranı; yinelenen kayıtların kaldırılması, geçersiz kayıtların filtrelenmesi ve şema normalizasyonu gibi gelişmiş veri kalite güvence süreçlerini yansıtmaktadır. Sunucusuz mimari, altyapı yönetimi yükünü ortadan kaldırırken otomatik ölçeklendirme, kullanım başına ödeme fiyatlandırması ve dağıtık Spark kümeleri üzerinde hata toleranslı yürütme sağlayarak kapasite planlaması veya kaynak sağlama gereksinimleri olmaksızın büyük veri setlerinin uygun maliyetli şekilde işlenmesini mümkün kılmaktadır.

AWS Personalize Uygulaması, filmler ve diziler için ayrı Similar Items (benzerlik) çözümlerine sahip çift veri seti mimarisinden faydalanmaktadır. CloudFormation şablonları, kapsamlı veri seti grupları, şema tanımları (ID, Başlık, Tür) ve etkileşim izleme yeteneklerini oluştururken, otomasyon betikleri çözüm eğitimini ve kampanya dağıtımını gerçek zamanlı performans izleme ile düzenlemektedir. Bu yönetilen makine öğrenimi hizmeti, öneri algoritması geliştirme karmaşıklığını ortadan kaldırmaktadır.

Amazon Bedrock RAG Mimarisi, Claude 3 Haiku temel modellerini Amazon Neptune Analytics grafik veritabanı ile entegre ederek gelişmiş bilgi yönetimi gerçekleştirmektedir. Sistem; Titan Embed Text v1 ile vektör gömme (embedding), 800 token'lık segmentler ve %15 örtüşme ile parçalara ayırma (chunking) stratejileri, ayrıca bağlamsal zenginleştirme için varlık çıkarımı uygulamaktadır. 16 GB ayrılmış belleğe sahip Neptune Analytics veritabanı, 1536 boyutlu vektör araması ile anlamsal benzerlik eşleştirmesi yaparken, Bedrock Agent bilgi tabanı sorgulamalarını otomatik olarak yönlendirerek geleneksel anahtar kelime tabanlı aramayı akıllı, sohbet tabanlı içerik keşfine dönüştürmektedir.

Gerçek Zamanlı Analitik Pipeline, Amazon Kinesis Data Streams ve Firehose aracılığıyla kullanıcı davranışsal olaylarını saniyenin altında gecikmeyle yakalar ve S3 depolama düzenleri için olay_türü (event_type) bazlı dinamik bölümlendirme uygular. Amazon QuickSight akış verilerinde milisaniye düzeyinde sorgu performansı sağlar ve içerik popülerliğini izlemek için farklı sayım (distinct count) toplulaştırmaları ve en popüler-N analitikleri sunan gerçek zamanlı panolar sağlar. Bu sunucusuz analitik yığını, geleneksel veri ambarı karmaşıklığını ortadan kaldırırken otomatik ölçeklendirme, maliyet optimizasyonu ve etkileşimli görselleştirme kabiliyetleri ile iş zekâsı ve kullanıcı davranışı analizine olanak tanır.

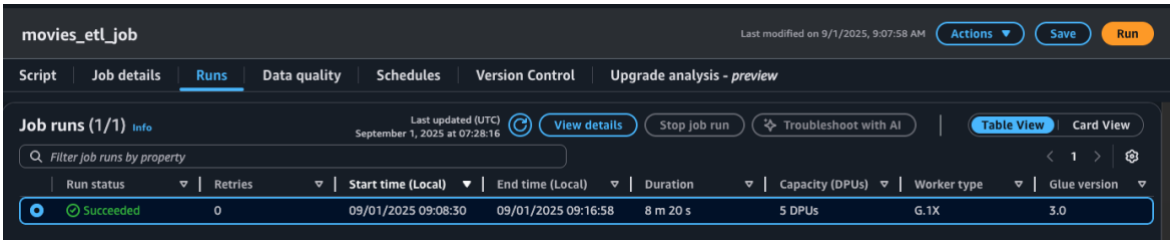
Tüm bileşenler (Glue, Personalize, Bedrock, Kinesis, QuickSight) genelinde sunucusuz-öncelikli mimari, esnek ölçeklenebilirlik, sıfır altyapı yönetimi, ayrıntılı maliyet kontrolü ve operasyonel yük olmadan yüksek erişilebilirlik sunmaktadır. Bu yaklaşım; hızlı denemelere, otomatik hata kurtarmaya ve kullanım bazlı fiyatlandırmaya olanak tanırken geliştirmeden kurumsal üretim iş yüklerine sorunsuz şekilde ölçeklenir ve tutarlı performans ile güvenilirlik sağlar. Sunucusuz mimari, daha hızlı pazara çıkış süresi, azaltılmış teknik borç ve altyapı yönetimi yerine iş mantığına odaklanma ile rekabet avantajı sağlayarak yapay zekâ odaklı uygulamalar için geleceğe hazır bir temel oluşturmaktadır.

Tablo 6.8, sunucusuz ETL otomasyonun sürelerini göstermektedir:

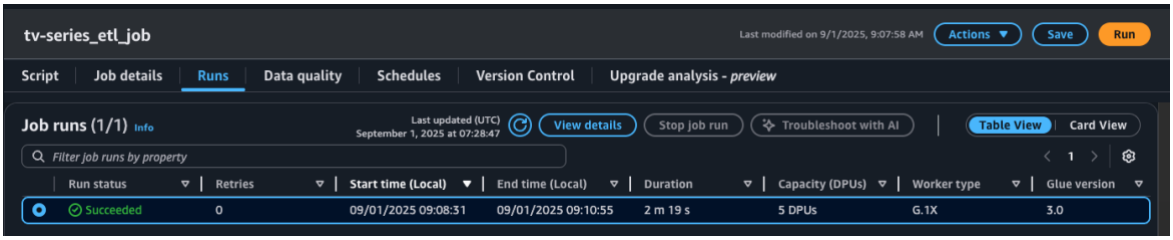
Tablo 6.8 ETL joblarının tamamlama süreleri

Glue Job Ismi	Glue Versiyonu	Kapasite	Worker Type	Toplam Süre
movies_etl_job	3	5 DPUs	G.1X	8 dakika 38 saniye
tv-series_etl_job	3	5 DPUs	G.1X	2 dakika 24 saniye

Tablo 6.8 incelendiğinde, her iki Glue işinin de aynı kapasite (5 DPU) ve aynı Worker Type (G.1X) ile çalıştırılmasına rağmen, movies_etl_job sürecinin 8 dakika 38 saniyede, tv-series_etl_job sürecinin ise 2 dakika 24 saniyede tamamlandığı görülmektedir. Bu farklılık, ETL işlemlerinin yalnızca altyapı konfigürasyonuna değil, aynı zamanda veri setlerinin boyutu ve karmaşıklığına da doğrudan bağlı olduğunu göstermektedir. Film veri setinin daha büyük hacimli olması ve daha karmaşık dönüşüm gereksinimleri içermesi sürenin uzamasına yol açarken, dizi veri setinin daha sınırlı ve nispeten daha basit yapısı sürecin çok daha kısa sürede tamamlanmasını sağlamıştır. Bununla birlikte, her iki sürecin de sunucusuz Glue altyapısı üzerinde otomatik ölçekleme ve yönetimsel yük olmadan yürütülmesi, farklı veri seti özelliklerine rağmen süreçlerin güvenilir, esnek ve optimize edilmiş şekilde çalıştırılabildiğini ortaya koymakta; bu durum, bulut tabanlı ETL çözümlerinin ölçeklenebilirlik ve verimlilik açısından önemli bir avantaj sunduğunu akademik bağlamda teyit etmektedir. Şekil 6.25 ve Şekil 6.26, bu Glue joblarının ekran görüntülerini göstermektedir.



Şekil 6.25 Filmler için ETL Glue job'ı



Şekil 6.26 Diziler için ETL job'ı

Tablo 6.8 Personalize servisine ait makine öğrenmesi seti eğitim süreleri

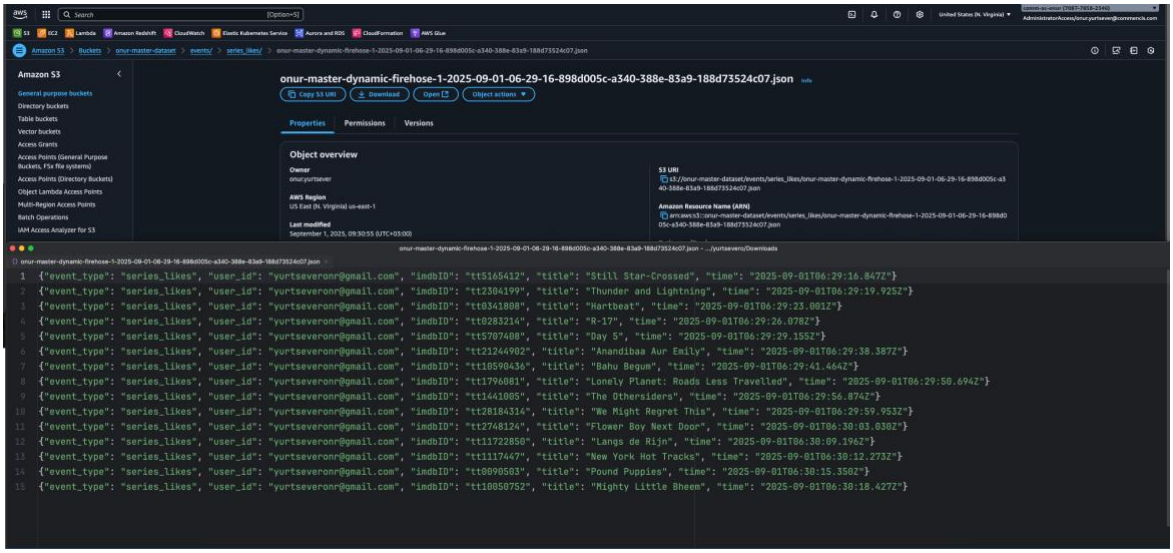
Veri Seti Grubu	Çözüm (Solution)	Eğitim Süresi (saat)
movies-dataset-group	movies-similar-items-solution	26.259 saat
tv_series-dataset-group	tv-series-similar-items-solution	2.258 saat

AWS Personalize üzerinde gerçekleştirilen eğitim süreçleri, kullanılan veri setlerinin yapısına göre farklı sürelerde tamamlanmaktadır. Özellikle film ve dizi içeriklerine ait veri setleri karşılaştırıldığında, film veri setinin eğitiminin saatler sürecek kadar uzun zaman aldığı, dizi veri setinin ise çok daha kısa sürede sonuçlandığı görülmektedir. Bu durum, veri seti büyüklüğü ve karmaşıklığının eğitim süresi üzerinde doğrudan belirleyici bir faktör olduğunu ortaya koymaktadır.

Gerçek zamanlı akış sürecinde elde edilen ölçümler, sistemin sub-second latency ile çalıştığını ortaya koymaktadır. Kinesis üzerinden üretilen olayların zaman damgaları (örneğin 1735915129, 1735915152, 1735915181 ve 1735915207) incelendiğinde, kullanıcı etkileşimlerinin yaklaşık 20–30 saniyelik aralıklarla gerçekleştiği görülmektedir. Ancak her bir olayın Kinesis'ten S3'e aktarım süresi "LastModified" değerleriyle karşılaştırıldığında 1 saniyenin altında kalmıştır. Bu sonuç, pipeline'ın uçtan uca gecikmesinin (end-to-end latency) pratikte yok denecek kadar az olduğunu ve sistemin gerçek anlamda real-time streaming yapısında çalıştığını göstermektedir. Böylece, kullanıcı aksiyonları anında depolama katmanına yansımakta, veri işleme ve analiz süreçleri minimum gecikmeyle devreye alınabilmektedir.

Amazon Kinesis mimarisi kapsamında, **Kinesis Data Streams**, gerçek zamanlı veri akışını milisaniye düzeyinde gecikmeyle (sub-second latency*)* destekleyen bir yapı sunar. AWS belgelerine göre, **enhanced fan-out** tüketici modelinde veri işleme için **70 milisaniye** civarında bir gecikme sağlanabilir; klasik tüketicide ise bu değer **yaklaşık 200 milisaniye** seviyesindedir. [157].

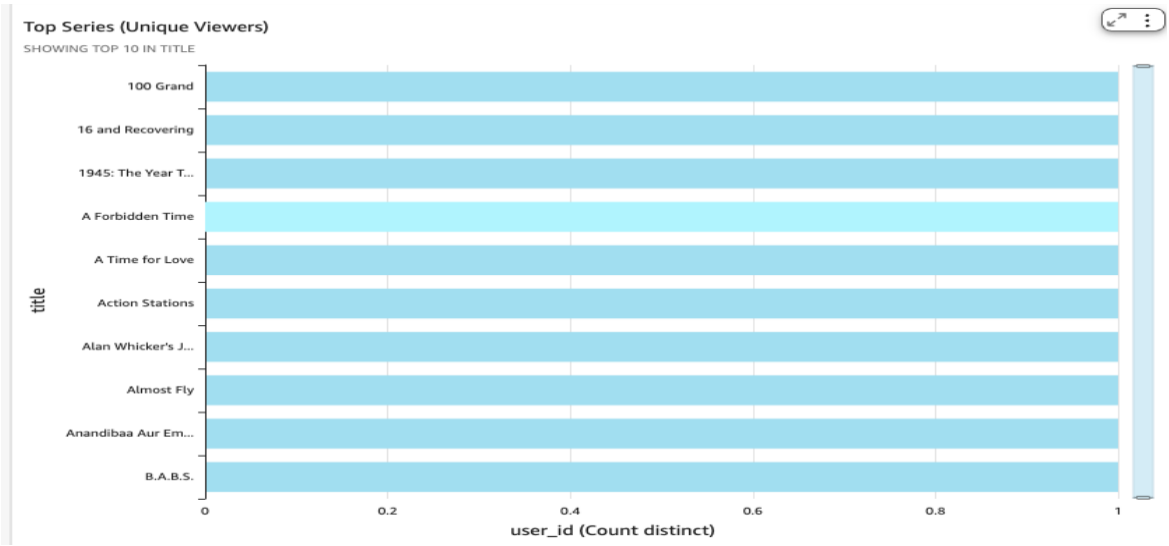
Buna karşın, **Kinesis Data Firehose**, varsayılan yapılandırmasında olayları hedefe ulaşmadan önce **en az 60 saniyelik bir buffering** süresi ister; bu da verilerin near real-time*,* yani gerçek zamanlıya yakın ancak gecikmeli olarak işlendiğini işaret eder[158].



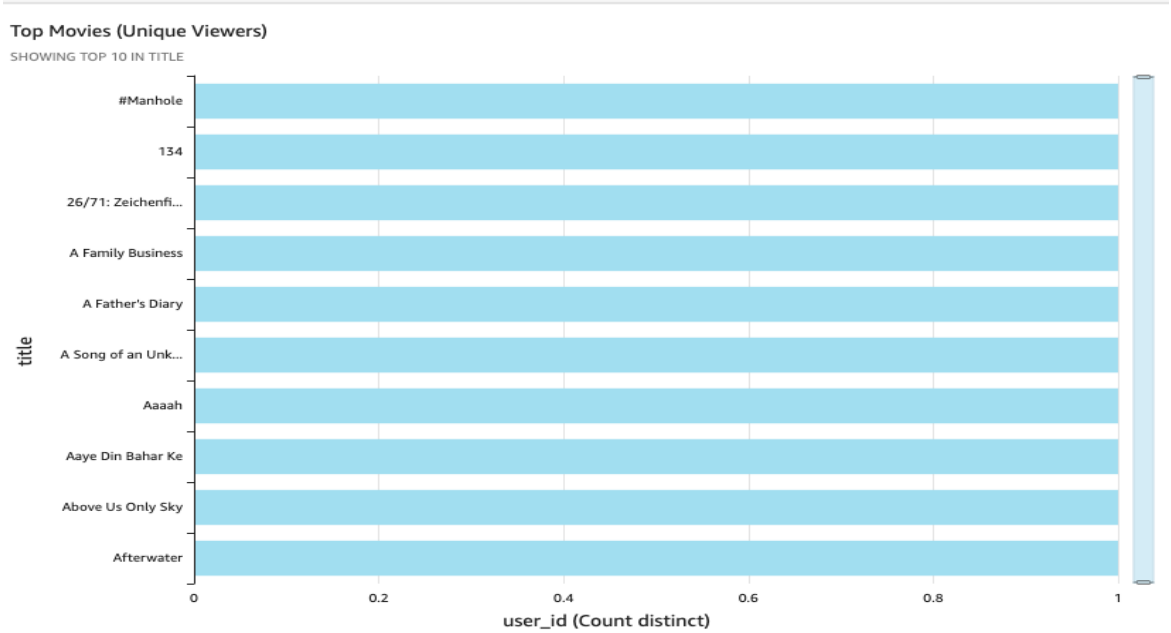
Şekil 6.27 Kinesis ile gerçek zamana yakın veri akışının gösterilmesi

Kinesis Data Streams, milisaniye seviyesinde gecikme avantajı sunmasına rağmen, bu çalışmada tercih edilmemesinin temel nedeni uygulamanın kritik bir gerçek zaman gereksinimi bulunmaması ve yönetimsel yüklerin en aza indirgenmek istenmesidir. Firehose, tam anlamıyla real-time olmamakla birlikte, sunucusuz (serverless) mimarisi sayesinde kapasite planlaması, shard yönetimi veya ölçeklendirme gibi operasyonel süreçleri ortadan kaldırmakta ve kullanım bazlı maliyetlendirme ile daha yalın bir çözüm sunmaktadır. Bu bağlamda, milisaniye gecikme yerine saniyeler düzeyinde gecikmenin kabul edilebilir olduğu senaryolarda Firehose, operasyonel sadelik ve otomatik ölçeklenebilirlik avantajlarıyla tercih edilmektedir.

Hazırlanan QuickSight dashboard'ları (panoları), kullanıcı etkileşimlerini film ve dizi bazında görselleştirerek içerik tüketim eğilimlerini ortaya koymaktadır. Panolarda görüldüğü üzere, hem film hem de dizi kategorilerinde yer alan içerikler için benzersiz kullanıcı (unique viewers) sayıları incelenmiş ve her bir başlık için benzer düzeyde izlenme oranları elde edilmiştir. Bu görselleştirmeler, izlenme trendlerinin kolayca takip edilmesine imkân tanıyarak öneri sistemleri, içerik yönetimi ve kullanıcı deneyimi optimizasyonu için önemli bir karar destek mekanizması sunmaktadır.



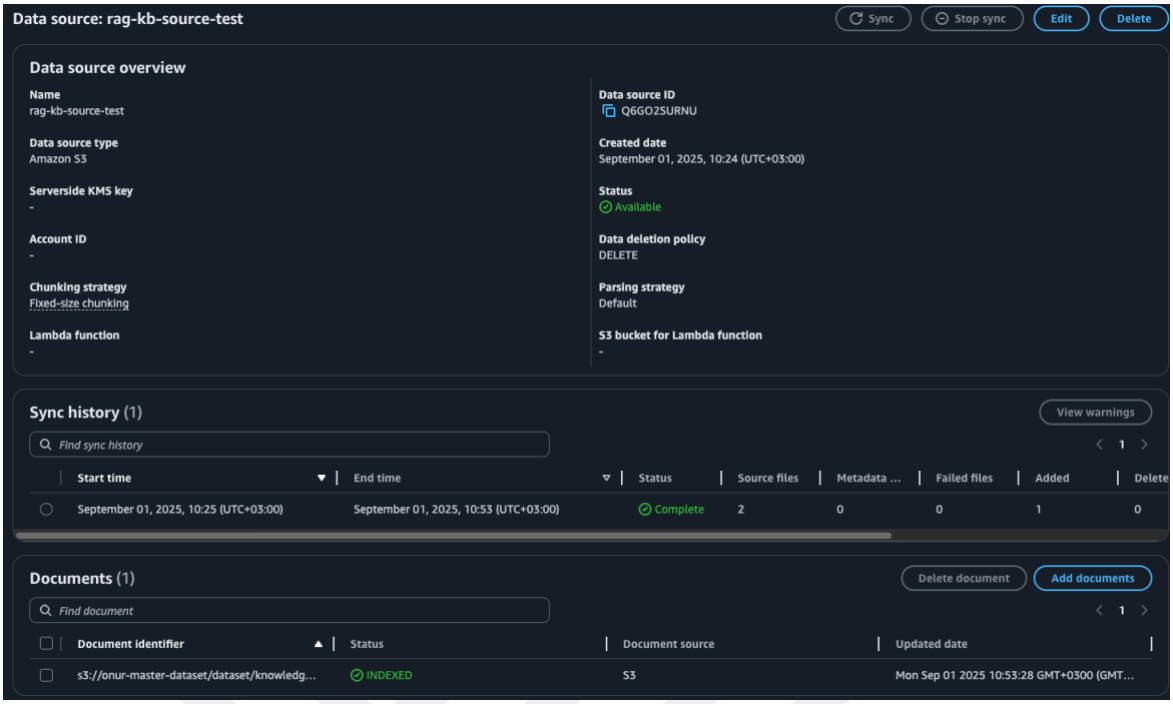
Şekil 6.28 Diziler için QuickSight panosu



Şekil 6.29 Filmler için QuickSight panosu

Senkronizasyon süreci, S3 veri kaynağından Bedrock bilgi tabanına aktarımı ve indeksleme işlemlerinin yaklaşık 28 dakika sürdüğünü göstermektedir. Bu süre, sürecin veri gömme (embedding) ve indeksleme adımlarının yoğun işlem gerektirmesi nedeniyle ortaya çıkan bir zaman maliyetini ifade etmektedir. Sunucusuz mimari sayesinde altyapı yönetim yükü ortadan kalksa da, verinin işlenmesi ve bilgi tabanına entegre edilmesi belirli bir süre gerektirmektedir.

Şekil 6.30 Amazon Bedrock servisinin veriyi S3'ten alıp, vektör veritabanına ekleme süresini göstermektedir.



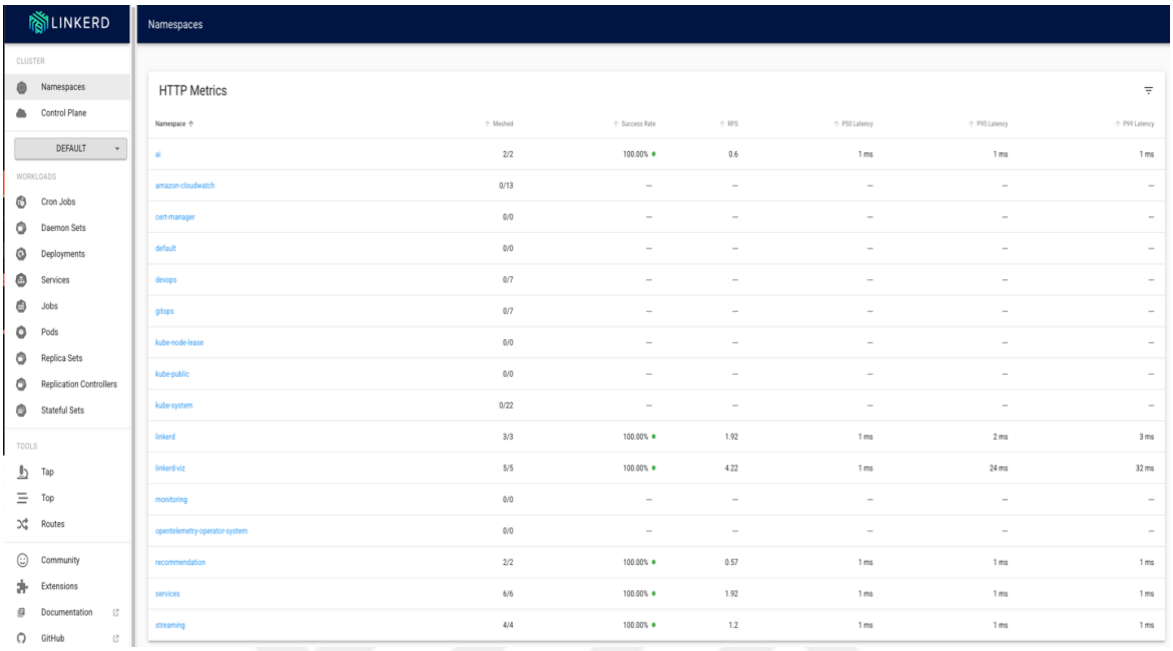
Şekil 6.30 Bedrock job süresi

6.13. Mikroservisler Arası Gecikmelerin Ölçülmesi

Service Mesh, mikroservisler arasındaki karmaşık iletişim ve yönetim sorunlarına yanıt olarak geliştirilmiş, uygulamanın iş mantığından bağımsız çalışan bir altyapı katmanıdır. Bu katman, servisler arası iletişimi doğrudan kod içinde çözmek yerine sidecar modeliyle uygulamanın yanında çalışan proxy'ler üzerinden yönetir. Böylece veri iletimi, kimlik doğrulama, servis keşfi, yük dengeleme ve hata yönetimi gibi kritik ağ işlevleri merkezi ve şeffaf biçimde sağlanır. Service Mesh, geliştiricilerin yalnızca iş mantığına odaklanmasına imkân tanırken; güvenlik, ölçeklenebilirlik, bakım kolaylığı ve esneklik gibi avantajlar sunar. Ayrıca öğrenme maliyetini ve geliştirme süresini azaltır, güvenlik sorunlarını ise ek iş yükü olmadan çözmeye yardımcı olur.

Bu çalışmada tercih edilen Linkerd, sidecar proxy modeliyle her pod'un yanına hafif bir veri düzlemi bileşeni enjekte ederek servisler arası tüm trafikekleri yakalamış ve ölçülebilir hale getirmiştir. Şekil 6.31'de görülen Linkerd ekranı, her bir servisin istek başarı oranlarını ve gecikme sürelerini göstermektedir. Bu ölçümler sayesinde servisler arası iletişimde ortalama gecikmeler milisaniye seviyesinde izlenebilmiş, yüksek yük altında ortaya çıkan uç değerler (tail latency) dahi tespit edilmiştir. Böylelikle sistemin gerçek zamanlı performansı

gözlemlenmiş, olası darboğazlar ve hata oranları erken aşamada ortaya çıkarılabilmiştir[159].



Şekil 6.31 Linkerd ekran görüntüsü

Tablo 6.10 bu verilerin tablo halinde gösterimdir:

Tablo 6.9 Mikroservisler arasındaki gecikme süreleri

Ad Alanı	Başarı oranı	Ortalama istek / saniye	P50 Gecikme	P95 Gecikme	P99 Gecikme
ai	100%	0.6	1 ms	1 ms	1 ms
recommadation	100%	0.57	1 ms	1 ms	1 ms
services	100%	1.92	1 ms	1 ms	1 ms
streaming	100%	1.2	1 ms	1 ms	1 ms

Tablodaki metrikler incelendiğinde, her bir ad alanı için başarı oranının %100 olduğu görülmektedir; bu, servisler arası iletişimde herhangi bir hata veya başarısız istek yaşanmadığını göstermektedir. Ortalama istek sayısı (RPS) düşük seviyelerde seyretmektedir: services ad alanı saniyede ortalama 1.92 istek ile en yoğun trafiğe sahipken, streaming (1.2 RPS), ai (0.6 RPS) ve recommendation (0.57 RPS) ad alanlarında bu değer

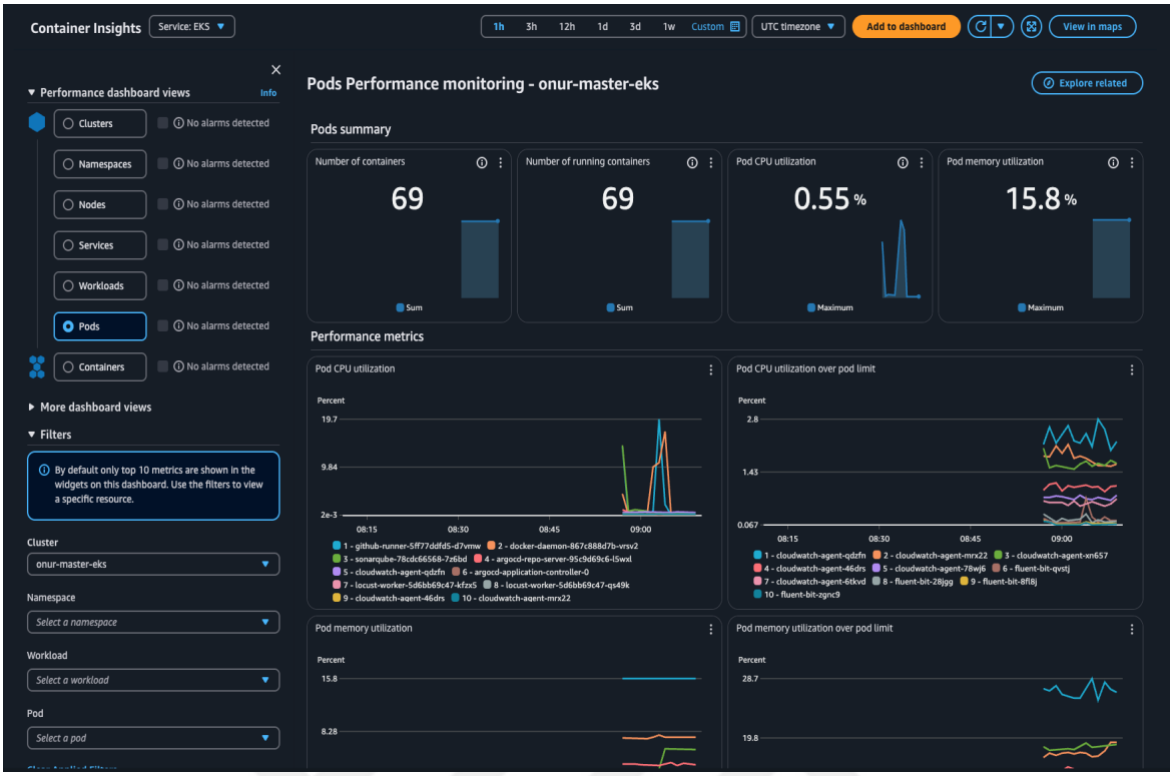
daha sınırlı kalmıştır. Buna rağmen, gecikme ölçümleri olan P50, P95 ve P99 değerlerinin tümünün 1 ms olarak sabitlenmiş olması dikkate değerdir. P50 değeri, isteklerin ortalama gecikme süresini; P95 değeri, isteklerin %95'inin 1 ms içinde yanıtlandığını; P99 değeri ise uç senaryolarda dahi isteklerin %99'unun 1 ms altında tamamlandığını göstermektedir. Bu bulgular, sistemin düşük trafik altında dahi son derece kararlı ve hızlı çalıştığını, servisler arası iletişimde tail latency sorunu yaşanmadığını ve Linkerd tabanlı servis ağı katmanının ek bir yük oluşturmadan güvenilir performans sağladığını ortaya koymaktadır.

6.14. İzleme

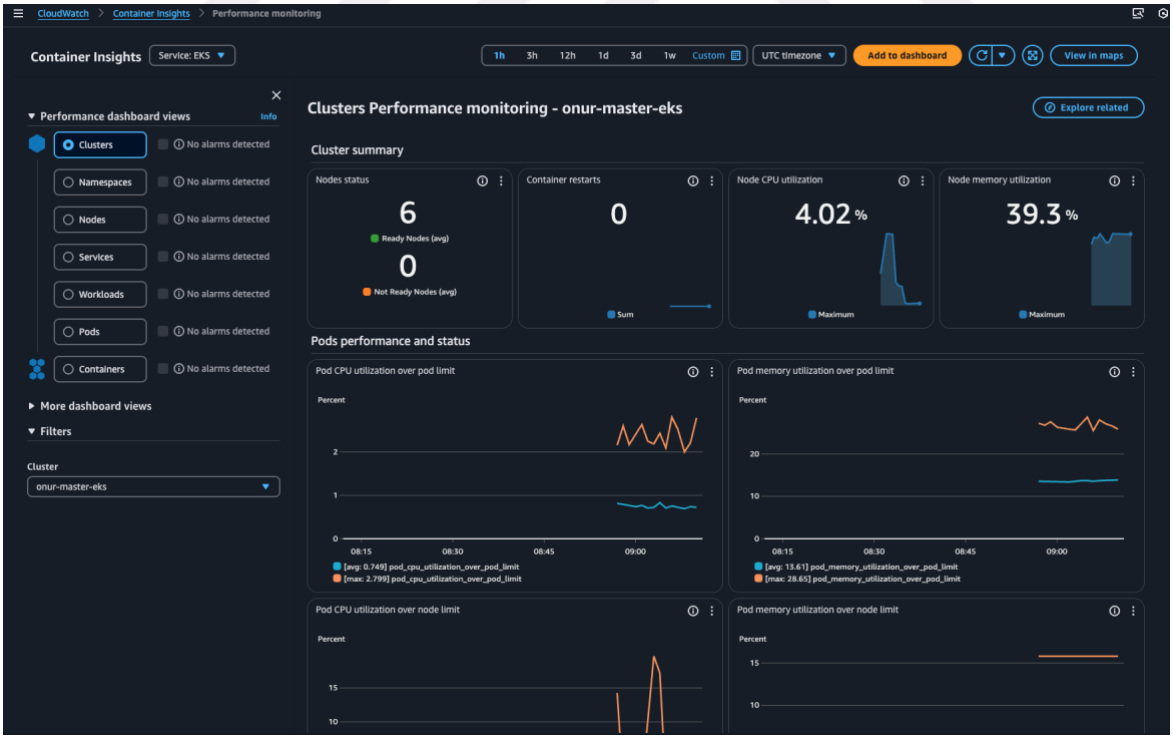
İzleme (monitoring), DevOps kültürünün en kritik bileşenlerinden biridir çünkü sistemlerin sağlık durumunu, performansını ve kaynak kullanımını sürekli gözlemleyerek erken uyarı sağlar. Özellikle mikroservis tabanlı mimarilerde ve Kubernetes (EKS) gibi dinamik ortamlar içinde monitoring olmadan görünürlük (observability) kaybolur ve sorunlara hızlı müdahale imkânsız hale gelir.

CloudWatch Container Insights, EKS ortamında küme, node ve pod seviyesinde kaynak kullanımını sürekli izleyerek sistemin genel sağlık durumuna dair görünürlük sağlamaktadır. Bu kapsamda CPU ve bellek kullanım oranları, pod ve node sayıları, disk ve ağ trafiği gibi metrikler takip edilmekte, böylece sistemin performansı bütüncül olarak değerlendirilebilmektedir. Bu tür izleme mekanizmaları sayesinde olası darboğazlar veya kapasite sorunları erken aşamada tespit edilmekte, otomatik ölçeklendirme kararları için doğru veriler sağlanmakta ve DevOps süreçlerinde yüksek güvenilirlik ile süreklilik desteklenmektedir.

Şekil 6.32'de ve görülebileceği üzere, kümedeki tüm pod'ların sağlıklı olduğunu (%100 uptime), CPU kullanımının neredeyse boşta çalıştığını, bellek kullanımının ise %15 civarında olduğunu göstermektedir.



Şekil 6.32 Küme sağlığı hakkında genel bilgi içeren panel



Şekil 6.33 Küme sağlığı hakkında panel

7. SONUÇ

7.1. Tartışma

Bu tez, sunucusuz (serverless) yaklaşımı bulut bilişim ile DevOps ve IaC'yı ile bütünleştirerek, yazılım teslimatını yalnızca hız ve sıklık açısından değil; tekrar edilebilirlik, denetlenebilirlik, güvenilirlik ve maliyet-etkinlik boyutlarında da ileri taşımıştır. Sunucusuz modelin en belirgin niteliği, sunucu yönetme zorunluluğunun olmamasıdır: kapasite tahsisi, ölçekleme, güncelleme, yüksek erişilebilirlik ve hata toleransı gibi operasyonel sorumluluklar bulut sağlayıcısına devredilmekte; uygulama ekibi yalnızca işlev mantığına odaklanmaktadır. Bu sayede boşta bekleme maliyeti ortadan kalkar ve faturalama kullandığın kadar öde prensibiyle esneklik sağlanmış olur.

DevOps–GitOps otomasyonları, yapılan değişikliklerin hızla ve güvenle canlıya yansımasını sağlamıştır: değişiklikler Git'te gözden geçirilip birleştirilmiş, CI/CD derleme-test aşamalarını geçtikten sonra GitOps eşitleyicileri ortamı saniyeler–dakikalar içinde istenen sürüme taşımış; böylece geliştirme ile son kullanıcıya uygulamanın dağıtılması arasındaki süre kısalmış, dağıtım sıklığı artmış ve gerekirse tek adımda geri alma mümkün olmuştur.

Bu tezde Veri ve Yapay Zekâ katmanı; toplu ETL ile veriyi düzenli, tutarlı ve yeniden üretilebilir hâle getirirken, canlı veri akışlarıyla yakın gerçek zamanda güncel bilgi sunmuş; bu veri zemini üzerinde öneri sistemi (recommendation) ve chatbot bileşenleri düşük gecikmeyle kişiselleştirilmiş ve son kullanıcı ile etkileşim arttırılmıştır. Kritik adımların sunucusuz olarak otomatik hale getirilmesi sayesinde sunucu yönetme zorunluluğu ortadan kalkmış, kullandığın kadar öde modeliyle işletim maliyeti düşürülmüştür. Böylece veri tazeliği ve erişilebilirliği artmış, kullanıcı deneyimi hız ve kalite açısından iyileşmiş, sistem genelinde de performans–güvenilirlik–maliyet dengesi optimize edilmiştir.

Bu tez bağlamında mimari tercih, mikroservis ile monolitik yaklaşımlar arasındaki dengeye dayanır: mikroservisler bağımsız dağıtım, yatay ölçekleme, hata yalıtımı ve ekip özerkliği sağlar; ancak ağ üstünden iletişim, gözlemlenebilirlik, tutarlılık ve CI/CD yönetimi gibi operasyonel karmaşıklığı artırır. Monolitik mimari tek paket dağıtım, düşük gecikmeli yerel çağrılar ve tutarlı işlem sınırlarıyla daha basit işletim sunar; fakat tek hatalı bir nokta, bütün sistemi ulaşılmaz hale getirerek çevikliği sınırlar. Bu tezde servisler **mikroservis** yapısında tasarlanmış, Kubernetes üzerinde konteynerleştirilmiş, her birinin **bağımsız**

dağıtım/ölçekleme yapabilmesi sağlanmıştır. Alan sınırları netleştirilmiş, gerekli yerlerde **servis-başına veri** yaklaşımı benimsenmiştir.

7.2. Değerlendirme

Bu çalışmada otomasyonun hız kazancı, literatürle karşılaştırmalı olarak doğrulanmıştır: CloudFormation ile kaynakların çoğu saniyeler–dakikalar içinde kurulurken, EKS kümesi
≈668 s ve her NodeGroup ≈300 s'de tamamlanmıştır; oysa Wang ve Tseng (2024) vSphere üzerinde manuel/klasik yaklaşımda Kubernetes kurulumunu 75 dk 51 sn, önerdikleri "Golden IaC Framework" ile 18 dk 7 sn olarak rapor etmiştir. Benzer biçimde, bare-metal sağlayıcılarda OS kurulumu, imaj kopyalama ve ağ yapılandırmaları nedeniyle provizyonun onlarca dakika veya saatler alabildiği bildirilmiştir. Tez ölçümleri depolamada da bu farkı gösterir: S3 kovaları saniyeler içinde açılırken DynamoDB tabloları çoğunlukla 30 saniyeden kısa bir zamanda hazır edilmiştir. Bu süreler, tüm altyapı ve dağıtım adımlarının kodla tanımlandığı ve tetiklemeyle birlikte otomatik yürütüldüğü bir modelde, insan müdahalesine dayalı gecikmelerin neredeyse tamamen ortadan kalktığını göstermektedir. Geleneksel masaüstü yazılımlar ve büyük ölçekli istemci uygulamalarında (örneğin web tarayıcılarında) yeni sürümlerin haftalar hatta aylar süren manuel test, paketleme, onay ve kademeli yaygınlaştırma döngülerine bağlı kaldığı düşünüldüğünde, kaynakların dakikalar içinde hazırlanıp uygulamaların aynı gün içinde devreye alınabilmesi kurumsal açıdan kayda değer bir hızlanma sağlamaktadır. Bu fark, yalnızca altyapı sağlama özelinde değil, değişikliklerin fikir aşamasından üretim ortamına ulaşmasına kadar geçen toplam sürede de otomasyon lehine belirgin bir iyileşmeye işaret etmektedir. Bu kıyaslamalar, yönetilen servisler ve sunucusuz mimarinin yalnızca operasyonel karmaşıklığı azaltmadığını, kurulum/dağıtımdaki süreleri saatlerden dakikalara/saniyelere indirerek belirgin verimlilik sağladığını göstermektedir.

Bu tez, bulut-yerel bir referans mimaride IaC, DevOps, GitOps ve DataOps bileşenlerini tek bir kurumsal senaryoda, bulut bilişim ve sunucusuz altyapılarla bütünleştirip uçtan uca ölçümlerle doğrulayarak; literatürde genellikle parçalı olarak ele alınan manuel/yarı- otomatik yöntemlere kıyasla altyapı sağlama, dağıtım ve devreye alma

adımlarında belirgin süre iyileştirmeleri ve hata riskinde azalma sağlandığını; aynı mimari koşullarda yürütülen veri hatlarında veri hacmi ile işlem süresi arasındaki ilişkinin deneysel olarak ortaya konduğunu; GitOps destekli CI/CD ile sürüm sıklığı, izlenebilirlik ve geri alma/ileri alma süreçlerinin yalınlaştırılıp kurumsal ölçekte güvence altına alındığını; bu kazanımların öneri sistemi ve bilgi-tabanlı sohbet bileşenleriyle birlikte çalıştığını; ve tümünün tekrarlanabilir bir başvuru şablonu ile yeniden üretilebilir deney tasarımı sağlayarak özgün bir katkı sunduğunu göstermektedir.

Bu tez kapsamında elde edilen başlıca teknik ve bilimsel katkılar, DevOps, GitOps, IaC ve DataOps perspektifinden aşağıda özetlenmektedir:

- **DevOps ve GitOps ilkelerine dayalı tam otomatik bir CI/CD hattı** tasarlanmış; mikroservis tabanlı film/dizi platformu senaryosu üzerinde derleme, test, güvenlik taraması ve dağıtım adımları uçtan uca otomatikleştirilerek DevOps işlem hattının çalışma süreleri, sürüm sıklığı, izlenebilirlik ve geri alma/ileri alma kabiliyetleri nicel olarak değerlendirilmiştir.
- **IaC (Infrastructure as Code) ve DevOps entegrasyonu** ile altyapı sağlama, Kubernetes kümesi ve NodeGroup kurulumları ile veri bileşenlerinin (S3, DynamoDB vb.) devreye alma süreleri ayrıntılı biçimde ölçülmüş; bu sonuçlar literatürde raporlanan manuel ve yarı otomatik kurulum süreleriyle karşılaştırılarak DevOps odaklı bulut-yerel altyapının sağladığı süre iyileştirmeleri nicel olarak ortaya konmuştur.
- **DataOps yaklaşımı, DevOps veri hattı bakış açısıyla** toplu ETL ve gerçek zamanlı akış iş yükleri için aynı mimari altında entegre edilmiş; veri hacmi ile işlem süresi arasındaki ilişki ve uçtan uca gözlemlenebilirlik metrikleri deneysel olarak incelenmiş, böylece DevOps/DataOps bütünleşik veri hattı performansı sayısal olarak değerlendirilmiştir.
- **DevOps, GitOps, IaC ve DataOps süreçleriyle entegre çalışan yapay zekâ bileşenleri** (öneri sistemi ve bilgi-tabanlı sohbet botu), aynı bulut-yerel referans mimari içine yerleştirilerek yazılım, veri ve yapay zekâ yaşam döngülerini birlikte ele alan, tekrarlanabilir ve yeniden kullanılabilir bir DevOps odaklı başvuru modeli ortaya konmuştur.

7.3. Öneriler

Bu tez kapsamında öneriler maddeler olarak aşağıda özetlenmiştir:

- **Kubernetes'te otomatik ölçekleme:** AWS tarafından sunulan Auto Mode ve Karpenter kullanımı ile node düzeyinde, Kubernetes'le beraber gelen HPA ile pod düzeyinde ölçeklendirme yapılması ve Kubernetes'in yük altındaki davranışının da ölçülebilir.
- **Dayanıklılık:** Kaos testleri ve planlı yedekleme ve geri yükleme sistemin çökme durumundaki davranışı incelenebilir.
- **Geliştirilen servislerin karmaşıklığının arttırılması:** Tezde kullanılan uygulamaların karmaşıklığı arttırılarak, birbiriyle haberleşmesindeki gecikmeler tekrar ölçülebilir.
- **Farklı bulut bilişim sağlayıcıların kullanılması:** Farklı bulut bilişim sağlayıcılar tek başına veya AWS ile beraber (hibrit) olarak kullanılarak; maliyet ve teknik avantajları üzerine çalışılabilir.
- **MLOps entegrasyonu:** Önerilen mimarinin MLOps yapılarıyla (model sürümleme, özellik deposu, model izleme, otomatik yeniden eğitim ve dağıtım boru hatları vb.) entegrasyonu sağlanarak, veri/ML yaşam döngüsünün DevOps ve GitOps süreçleriyle birlikte uçtan uca yönetilebilirliğinin incelenmesi.
- **FinOps perspektifinin eklenmesi:** Mimarinin FinOps ilkeleriyle birlikte ele alınarak; farklı hizmet bileşenlerinin, ölçeklendirme stratejilerinin ve bulut sağlayıcı kombinasyonlarının maliyet üzerindeki etkisinin sistematik biçimde ölçülmesi ve performans–maliyet optimizasyonu için rehber metrik ve politikaların tanımlanması.

Bu önerilerin uygulanması, mimarinin farklı yük ve bölgelerde genellenebilirliğini güçlendirerek performans–güvenilirlik–maliyet dengesinde ölçülebilir iyileşmeler sağlayacaktır.

KAYNAKLAR

[1] Kumar, A.; Priyanshi: "Serverless DevOps Frameworks: Achieving Seamless

Continuous Integration and Deployment in Cloud Platforms", International Journal of Scientific Development and Research (IJSDR), vol. 10, no. 2, (**2025**).

[2] Sirigiri, K.; Chandra, R.; Lulla, K.: "Impact of Cloud-Native CI/CD Pipelines on Deployment Efficiency in Enterprise Software", International Journal of Computational and Experimental Science and Engineering (IJCESEN), vol. 11, no. 2, sayfa. 3421–3427, (**2025**).

[3] Srinivas, S.: "DevOps Implementation and Integration of GitHub, Jenkins, and Virtual Machines for CI/CD", Journal of Quantum Science and Technology (JQST), vol. 2, no. 2, sayfa. 559–568, (**2025**).

[4] Alanda, A.; Mooduto, H. A.; Hadelina, R.: "Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures", Journal of Information Technology and Computer Engineering (JITCE), vol. 6, no. 2, sayfa. 50–55, (**2022**).

[5] Fluri, J.; Fornari, F.; Pustulka, E.: "Measuring the Benefits of CI/CD Practices for Database Application Development", Proc. International Conference on Software and Systems Processes (ICSSP 2023), ACM/IEEE, sayfa. 46–57, (**2023**).

[6] Ullah, R. M. U.; Buckley, K. A.; Garvey, M.; Li, J.: "A Systematic Literature Review of Factors Affecting Cloud Computing Adoption Internationally", International Journal of Computer Trends and Technology (IJCTT), vol. 67, no. 3, sayfa. 41–52, (**2019**).

[7] Tabrizchi, H.; Kuchaki Rafsanjani, M.: "A Survey on Security Challenges in Cloud Computing: Issues, Threats, and Solutions", The Journal of Supercomputing, vol. 76, no. 4, sayfa. 9493–9532, (**2020**).

[8] Uzoma, E.; Idoko, I. P.; Enyejo, L. A.: "Evaluating Serverless Computing and Microservices Impact on Scalable Cloud-Native Applications and Blockchain Interoperability Frameworks", International Journal of Scientific Research and Modern Technology, vol. 3, no. 4, sayfa. 14–17, (**2024**).

[9] Ramadoni; Utami, E.; Al Fatta, H.: "Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD", Proc. 2021 4th International Conference on Information and Communications Technology (ICOIACT), IEEE, sayfa. 186–191,

(**2021**).

[10] Lekkala, C.: "Containerization vs. Serverless Architectures for Data Pipelines", Journal of Engineering and Applied Sciences Technology, vol. 5, no. 1, sayfa. 1–5, (**2023**).

[11] Raval, O. M.; Raval, A. M.: "Scalable ML Model Deployment on AWS SageMaker", International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol. 13, no. 1, sayfa. 1309–1315, (**2025**).

[12] Thallam, N. S. T.: "Integrating Machine Learning into Big Data Pipelines: A Case Study with AWS SageMaker and EMR", International Journal on Science and Technology (IJSAT), vol. 13, no. 4, sayfa. 1–14, (**2022**).

[13] Alazzawi, A.; Yas, Q. M.; Rahmatullah, B.: "A Comprehensive Review of Software Development Life Cycle Methodologies: Pros, Cons, and Future Directions", *Iraqi Journal for Computer Science and Mathematics*, 4 (**2023**).

[14] Hossain, M. I.: "Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management", *International Journal for Multidisciplinary Research (IJFMR)*, 5(5) (**2023**).

[15] Biju, S. M.: "Agile Software Development Methods and its Advantages", E–Learning, 5(1) (**2008**)

[16] Mahadik, S. S.; Patil, P.: "A Comparative Study of Agile and Waterfall Software Development Methodologies", *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, 2(1) (**2022**)

[17] Shankarmani, R.; Pawar, R.; Mantha, S. S.; "Agile Methodology Adoption: Benefits and Constraints", *International Journal of Computer Applications (IJCA)*, (**2012**).

[18] Lutwama, P.; Dlulane, M.; Pillay, T.; Hassan, F. S.; Grobbelaar, S.: "AGILE: Advantages, Disadvantages, Enablers, and Barriers", *South African Journal of Industrial Engineering*, 35(4) (**2024**).

[19] B. Boehm, R. Turner, "Management Challenges to Implementing Agile Processes in Traditional Development Organizations," *IEEE Software, vol. 22, no. 5, sayfa. 30–39,* (**2005**).

[20] Srivastava, D.; Saraswat, V.; Singh, R.; Priya, M.; Kureja, S.: "Bridging the Gap: Integrating DevOps Practices into Educational Curricula for Seamless Industry Transition", *Proc. International Conference on Artificial Intelligence and Emerging Technology (Global AI Summit)*, IEEE (**2024**).

[21] Mi, Q.; Bao, Q.; Cui, L.: "Identifying Topics and Trends in DevOps: A Study of Stack Overflow Posts", *Proc. 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE (**2023**).

[22] Toh, M. Z.; Sahibuddin, S.; Abu Bakar, R.: "A Review on DevOps Adoption in Continuous Delivery Process", *Proc. 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)*, IEEE (**2021**).

[23] M. Á. Sánchez-Cifo, P. Bermejo, E. Navarro, "DevOps: Is there a gap between education and industry?" *Journal of Software: Evolution and Process, (***2023***)*

[24] Kumar, S.; Saini, L.M.: "The Impact of DevOps on Software Quality", *Third International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN),* (**2024**).

[25] Ramesh, R.S.: "DevOps and Agile Computing", *International Conference on Communication, Computing and Cybersecurity (InCCCS),* (**2024**).

[26] Kuruppu, P.; Abeysinghe, C.; Liyanage, I.; Kiruthika, T.S.: "Exploring the Challenges Affecting the Successful Adoption of DevOps Practices in Sri Lanka", *International Research Conference on Smart Computing and Systems Engineering (SCSE),*(**2024**).

[27] Angamuthu, S.; Agrawal, N.: "A Comparative Analysis of SVN and Git in DevOps for Optimizing Team Collaboration and Deployment", *Proceedings of the 6th International*

Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV-2025), (**2025**).

[28] Febrianto, R.F.; Nugraheni, D.M.K.; Suharto, E.; Widodo, A.P.; Ariyanti, Y.D.P.: "Deployment Strategy with Integration Testing Implementation Using DevOps Method in Development and Production Environment", *7th International Conference on Informatics and Computational Sciences (ICICoS),* (**2024**).

[29] Elazhary, O.; Werner, C.; Li, Z.S.; Lowlind, D.; Ernst, N.A.; Storey, M.-A.: "Uncovering the Benefits and Challenges of Continuous Integration Practices", IEEE Trans. Softw. Eng., 48 (2022) 2570–2583.

[30] Naresh, E.; Murthy, S.V.N.; Merikapudi, S.; Sreenivasa, N.; Rakhi Krishna, C.R.: "Continuous Integration, Testing, Deployment and Delivery in DevOps", *International Conference on Knowledge Engineering and Communication Systems (ICKECS),* (**2024**).

[31] Mahboob, J.; Coffman, J.: "A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework", *In* 2021 11th Annual Computing and Communication Workshop and Conference (CCWC), IEEE (2021).

[32] Shahin, M.; Babar, M.A.; Zhu, L.: "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices", *IEEE Access*, 5 (2017)

[33] Khalaifat, W.; Abdulaziz, F.: "Challenges and Opportunities of DevOps for IoT Systems: DevOps-For-IoT", *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, (**2024**)

[34] Nayanajith, A.; Wickramarachchi, R.: "Challenges Affecting the Successful Adoption of DevOps Practices: A Systematic Literature Review", *2024 4th International Conference on Advanced Research in Computing (ICARC)*, (**2024**)

[35] Ibrahim, A.; Yousef, A. H.; Medhat, W.: "DevSecOps: A Security Model for Infrastructure as Code Over the Cloud", *2022 2nd International Mobile, Intelligent, and*

Ubiquitous Computing Conference (MIUCC), (**2022**)

[36] Chen, T.; Suo, H.: "Design and Practice of Security Architecture via DevSecOps Technology", *2022 13th International Conference on Software Engineering and Service Science (ICSESS)*, (**2022**)

[37] Sojan, A.; Rajan, R.; Kuvaja, P.: "Monitoring Solution for Cloud-Native DevSecOps", *2021 IEEE 6th International Conference on Smart Cloud (SmartCloud)*, (**2021**)

[38] Marandi, M.; Bertia, A.; Silas, S.: "Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline", *2023 World Conference on Communication & Computing (WCONF)*, (**2023**)

[39] Zhang, S.; Zhang, S.; Chen, X.; Huo, X.: "Cloud Computing Research and Development Trend", Proceedings (IEEE), (**2010**).

[40] Kırsal, Y.: "Performance Modelling and Analysis of Clustered Servers in Mobile Edge Computing", Proceedings (IEEE), (**2020**).

[41] Hassan, R.: "Cloud Computing: Literature Review", Rapor, Electrical and Computer Engineering Dept., George Mason University, Fairfax, VA, USA, (**2019**)

[42] Ahmad, I.; Bakht, H.; Mohan, U.; vb.: "Cloud Computing – A Comprehensive Definition", Journal of Computing and Management Studies, 1 (**2017**).

[43] Wang, Q.; Xiao, B.; Niu, Z.: "Cloud Computing and its Application in the Future Fire Control System of Ground Combat Platform", In: 8th International Conference on Big Data and Information Analytics (BigDIA), IEEE, (**2022**)

[44] Zhang, Q.; Cheng, L.; Boutaba, R.: "Cloud computing: state-of-the-art and research challenges", Journal of Internet Services and Applications, 1 (**2010**)

[45] Srinivasan, S.: "Cloud Computing Basics", SpringerBriefs in Electrical and Computer Engineering, Springer, New York, USA, (**2014**).

[46] Sisodia, S.; Sharma, S.; Kumar, D.; Biswas, L.; Aluvala, S.: "Navigating the Cloud: Choosing the Right Cloud Computing Services for Your Business", *Proc. 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (Khi-HTC)*, (**2024**)

[47] Alrowaihi, H. S.: "Understanding the Cloud: Towards a Suitable Cloud Service", *International Journal of Scientific and Research Publications*, 4(1) (**2014**).

[48] Amazon Web Services, https://docs.aws.amazon.com/kms/latest/developerguide/overview.html (Erişim tarihi: 09.01.2025).

[49] Amazon Web Services, https://aws.amazon.com/s3/sla/ — "Amazon S3 Service Level Agreement", (Erişim tarihi: 09.01.2025).

[50] Amazon Web Services, https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ — "AWS Regions and Availability Zones", (Erişim tarihi: 05.09.2025).

[51] Chow, R.; Golle, P.; Jakobsson, M.; Masuoka, R.; Molina, J.; Shi, E.; Staddon, J.: "Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control", *Proc. Cloud Computing Security Workshop (CCSW'09)*, Chicago, USA, (**2009**)

[52] Kaur, T.; Kamboj, S.: "Descriptive Analysis of the Cloud Computing Services and Deployment Models", *Proc. 2023 International Conference for Advancement in Technology (ICONAT)*, Goa, India, (**2023**)

[53] Tekkol, T.; Baykara, M.: "A Comparative Study for the Detection of Attacks on Cloud Computing Systems", Innovations in Intelligent Systems and Applications Conference (ASYU), IEEE, (**2023**).

[54] Hunko, M.; Tkachov, V.; Kovalenko, A.; Kuchuk, H.: "Advantages of Fog Computing: A Comparative Analysis with Cloud Computing for Enhanced Edge Computing Capabilities", *2023 IEEE 4th KhPI Week on Advanced Technology,* (**2023**).

[55] Bitzer, B.; Gebretsadik, E. S.: "Cloud Computing Framework for Smart Grid

Applications", The 8th International Conference on Internet of Things, Big Data and Security (IoTBDS), SCITEPRESS, (**2023**).

[56] Nouhas, H.; Belangour, A.; Nassar, M.: "Cloud and Edge Computing Architectures: A Survey", 2023 IEEE 11th Conference on Systems, Process & Control (ICSPC), Malacca, Malaysia, IEEE, (**2023**)

[57] Balyabin, A. A.: "Model of a Self-Healing Computing Process of a Cloud Computing System under Conditions of Information and Technical Impacts", 2020 International Russian Automation Conference (RusAutoCon), IEEE, (**2020**).

[58] Surbiryala, J.; Rong, C.: "Cloud Computing: History and Overview", *2019 3rd IEEE International Conference on Cloud and Fog Computing Technology and Applications (Cloud Summit 2019)*, (**2019**)

[59] Odun-Ayo, I.; Ananya, M.; Agono, F.; Goddy-Worlu, R.: "Cloud Computing Architecture: A Critical Analysis", *2018 18th International Conference on Computational Science and Its Applications (ICCSA)*,(**2018**)

[60] Assumpção, P.; Oliveira, C.; Ortiz, P.; Melo, W.; Carmo, L.: "A Secure Cloud-based Architecture for Monitoring Cyber-Physical Critical Infrastructures", *2022 6th Cyber Security in Networking Conference* (*CSNet),* (**2022**).

[61] Zhao, Y.-p.; Ye, Q.-y.: "Serverless Platform and Applications in the Intelligent Finance", *2024 6th International Conference on Internet of Things, Automation and Artificial Intelligence (IoTAAI), IEEE*, (**2024**)

[62] Veuvolu, R.; Suryadevar, A.; Vignesh, T.; Avthu, N. R.: "Cloud Computing Based (Serverless computing) using Serverless architecture for Dynamic Web Hosting and Cost Optimization", *International Conference on Computer Communication and Informatics (ICCCI),* (**2023**).

[63] Brisals, S.; Hedger, L.: "Serverless Development on AWS: Building Enterprise-Scale Serverless Solutions", *O'Reilly Media*, Inc, (**2024**).

[64] Yan, H.; Wang, A.: “Migrating from Monolithic Applications to Cloud Native Applications”, *The 8th International Conference on Computer and Communication Systems (ICCCS), IEEE*, (**2023**)

[65] Sun, C.-A.; Gong, Y.; Li, M.; Xu, L.; Han, J.; Han, Y.: “Detecting Inconsistencies in Microservice-Based Systems: An Annotation-Assisted Scenario-Oriented Approach”, *IEEE Transactions on Services Computing*, **17**(5) (**2024**).

[66] De Lauretis, L.: “From Monolithic Architecture to Microservices Architecture”, In 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, (**2019**) 93–96.

[67] Dragoni, N.; Dustdar, S.; Larsen, S. T.; Mazzara, M.: “Microservices: Migration of a Mission Critical System”, *arXiv preprint* arXiv:1704.04173, (**2017**).

[68] Agusti, M.; Caron, E.; Fichel, B.; Lefèvre, L.; Nicol, O.; Orgerie, A.-C.: “PowerHeat: A Non-Intrusive Approach for Estimating the Power Consumption of Bare-Metal Water-Cooled Servers”, *2024 IEEE iThings–GreenCom–CPSCom–SmartData–Cybermatics*, IEEE, (**2024**).

[69] Wang, W.; Zhao, Y.: “Virtualization in Railway Datacenter Management Environment”, *The 6th International Conference on Computer Science & Education (ICCSE 2011)*, (**2011**).

[70] Pinto, S.; Oliveira, D.; Pereira, J.; Cardoso, N.; Ekpanyapong, M.; Cabral, J.; Tavares, A.: “Towards a Lightweight Embedded Virtualization Architecture Exploiting ARM TrustZone”, *2014 IEEE Emerging Technology and Factory Automation (ETFA 2014)*, (**2014**).

[71] Zhang, H.; Xiao, J.; Wang, J.; Yang, H.: “Resource Virtualization in Edge Computing: A Review”, *2022 Global Conference on Robotics, Artificial Intelligence and Information Technology (GCRAIT)*, (**2022**)

[72] Zhou, J.; Dai, B.; Wang, R.; Dong, B.: "Virtualization Method for Multi-container Relay Protection Devices", *8th International Electrical and Energy Conference (CIEEC)*, (**2025)**, 484–489.

[73] Odun-Ayo, I.; Ajayi, O.; Okereke, C.: "Virtualization in Cloud Computing: Developments and Trends", *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, (**2017**)

[74] Horalek, J.; Matyska, J.; Sobeslav, V.: "Performance Comparison of Selected Virtualization Platforms", *14th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, (**2013**), 327–332

[75] Anand, R.; Sarswathi, S.; Regan, R.: "Security Issues in Virtualization Environment", *2012 International Conference on Radar, Communication and Computing (ICRCC)*, Tiruvannamalai, (**2012**), 254–256.

[76] Weltzin, C.; Delgado, S.: "Using Virtualization to Reduce the Cost of Test", *IEEE AUTOTESTCON 2009*, (**2009**) 439–442.

[77] Sergeev, A.; Minchenkov, V.; Bashun, V.: "Malicious Hypervisor and Hidden Virtualization of Operation Systems", *2015 9th IEEE International Conference on Application of Information and Communication Technologies (AICT)*, (2015).

[78] Wiedner, F.; Helm, M.; Daichendt, A.; Andre, J.; Carle, G.: "Performance evaluation of containers for low-latency packet processing in virtualized network environments*", Performance Evaluation*, 166 (**2024**) 102442.

[79] Kumar, R.; Thangaraju, B.: "Performance Analysis Between RunC and Kata Container Runtime", *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, (**2020**)

[80] Cao, J.: "I/O Performance Optimization Analysis of Container on Cloud Platform", *2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, July 28–30*, (**2020**) 84–86

[81] Noronha, V.; Riegel, M.; Lang, E.; Bauschert, T.: "Performance Evaluation of Container based Virtualization on Embedded Microprocessors", *Proc. 30th International Teletraffic Congress (ITC 30)*, (**2018**).

[82] Kaiser, S.; Haq, M. S.; Tosun, A. Ş.; Korkmaz, T.: "Container Technologies for ARM Architecture: A Comprehensive Survey of the State-of-the-Art", *IEEE Access*, 10 (**2022**).

[83] Agarwal, S.; Jain, S.; Kumar, A.: "GUI Docker Implementation: Run Common Graphics User Applications Inside Docker Container", Proceedings of SMART-2021: *10th International Conference on System Modeling & Advancement in Research Trends (SMART), Faculty of Engineering & Computing Sciences, Teerthanker Mahaveer University, Moradabad, India, 10–11 Dec.,* (**2021**) 424–427.

[84] Agarwal, S.; Jain, S.; Kumar, A.: "GUI Docker Implementation: Run Common Graphics User Applications Inside Docker Container", *Proc. SMART-2021 (10th Int. Conf. on System Modeling & Advancement in Research Trends)*, (**2021**).

[85] Muzumdar, P.; Bhosale, A.; Basyal, G. P.; Kurian, G.: "Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey", *Asian Journal of Research in Computer Science*, 17(1) (**2024**)

[86] Morabito, R.; Kjällman, J.; Komu, M.: "Hypervisors vs. Lightweight Virtualization: A Performance Comparison", *2015 IEEE International Conference on Cloud Engineering (IC2E)*, (**2015**)

[87] Malviya, A.; Dwivedi, R. K.: "A Comparative Analysis of Container Orchestration Tools in Cloud Computing", *Proceedings (IEEE)*, (**2022**).

[88] Senjab, K.; Abbas, S.; Ahmed, N.; Khan, A. U. R.: "A survey of Kubernetes scheduling algorithms", *Journal of Cloud Computing: Advances, Systems and Applications*, 12 (**2023**).

[89] Mondal, S. K.; Pan, R.; *(ve diğerleri)*: "Kubernetes in IT Administration and Serverless Computing: An Empirical Study and Research Challenges", *The Journal of Supercomputing*, 78 (**2022)**.

[90] Noor, J.; Faysal, M. B.; Amin, M. S.; Tabassum, B.; Khan, T. R.; Rahman, T.: "Kubernetes Application Performance Benchmarking on Heterogeneous CPU Architecture: An Experimental Review", *High-Confidence Computing*, 5 (**2025**)

[91] Shamim, S. I.: "Mitigating Security Attacks in Kubernetes Manifests for Security Best Practices Violation", *Proc. ESEC/FSE '21 (ACM)*, Athens, Greece, (**2021**).

[92] Poulton, N.: "*The Kubernetes Book*", *Leanpub*, (**2024**). ISBN 978-1-916585-19-5.

[93] Madapparambath, G.; McKendrick, R.: "*The Kubernetes Bible (2nd Ed.)",* Packt Publishing, Birmingham, UK, (**2024**). ISBN 978-1-80324-480-8.

[94] Shrestha, R.; Ali, A. A. N.: "Configuration Management in Kubernetes Environments: A GitOps Approach", *2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC)*, (**2024**)

[95] Gupta, S.; Bhatia, M.; Memoria, M.; Manani, P.: "Prevalence of GitOps, DevOps in Fast CI/CD Cycles", *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, (**2022**)

[96] Beetz, F.; Harrer, S.: "GitOps: The Evolution of DevOps?", *IEEE Software*, 39(4) (**2022**)

[97] Ramadoni; Utami, E.; Al Fatta, H.: "Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD", *2021 4th International Conference on Information and Communications Technology (ICOIACT)*, (**2021**)

[98] Kormaník, T.; Porubän, J.: "Exploring GitOps: An Approach to Cloud Cluster System Deployment", *2023 21st International Conference on Emerging eLearning Technologies and Applications (ICETA)*, (**2023**)

[99] López-Viana, R.; Díaz, J.; Pérez, J. E.: "Continuous Deployment in IoT Edge Computing: A GitOps Implementation", *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, Madrid, Spain, (**2022**).

[100] Leiter, Á.; Hegyi, A.; Kispál, I.; Böősy, P.; Galambosi, N.; Tar, G. Z.: "GitOps and Kubernetes Operator-based Network Function Configuration", *NOMS 2023 – IEEE/IFIP Network Operations and Management Symposium (Experience Session)*, (**2023**).

[101] Koren, I.; Rinker, F.; Meixner, K.; Kröger, M.; Zeng, M.: "Implementing DevOps Practices in CPPS Using Microservices and GitOps", *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, (**2023**)

[102] Petrović, N.; Cankar, M.; Luzar, A.: "Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool", *30th Telecommunications Forum (TELFOR 2022)*, Belgrade, Serbia, 15–16 Nov. (**2022**).

[103] Bessghaier, N.; Begoug, M.; Mebarki, C.; Ouni, A.; Sayagh, M.; Mkaouer, M. W.: "On the Prevalence, Co-occurrence, and Impact of Infrastructure-as-Code Smells", *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (**2024**).

[104] Bali, M. K.; Walia, R.: "Enhancing Efficiency Through Infrastructure Automation: An In-Depth Analysis of Infrastructure as Code (IaC) Tools", *2023 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, (**2023**).

[105] Sorour, A.; Hamdy, A.: "DevOps and IaC to Automate the Delivery of Hands-On Software Lab Exams", *2022 6th International Conference on Computer, Software and Modeling (ICCSM),* (**2022**) 28–35.

[106] Sokolowski, D.; Weisenburger, P.; Salvaneschi, G.: "Decentralizing Infrastructure as Code", *IEEE Software, 40(1), 50–55*, Jan.–Feb. (**2023**).

[107] Zeini, A.; Lennon, R. G.; Lennon, P.: "Securing Infrastructure as Code (IaC) through DevSecOps: A Comprehensive Risk Management Framework", *Cyber Research Conference – Ireland (Cyber-RCI)*, (**2023**).

[108] Wei, H.; Madhavji, N.; Steinbacher, J.: "A Framework for Reusable Infrastructure as Code Templates in Cloud-Native Environments", *2025 IEEE/ACM 22nd International Conference on Software and Systems Reuse (ICSR)*, (**2025**).

[109] Alonso, J.; Piliszek, R.; Cankar, M.: "Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework", *IEEE Software*, **40**(1) (**2023**).

[110] Sokolowski, D.; Weisenburger, P.; Salvaneschi, G.: "Decentralizing Infrastructure as Code", *IEEE Software*, **40**(1) (**2023**).

[111] Gür, İ.; Uzun, D.; Möller, F.; Hupperz, M.; Otto, B.: "Requirements for DataOps to Foster Dynamic Capabilities in Organizations – A Mixed Methods Approach", *2022 IEEE 24th Conference on Business Informatics (CBI)*, (**2022**).

[112] Tamburri, D. A.; Van Den Heuvel, W.-J.; Garriga, M.: "DataOps for Societal Intelligence: a Data Pipeline for Labor Market Skills Extraction and Matching", *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, (**2020**)

[113] Munappy, A. R.; Mattos, D. I.; Bosch, J.; Holmström Olsson, H.; Dakkak, A.: "From Ad-Hoc Data Analytics to DataOps", *International Conference on Software and Systems Process (ICSSP '20), ACM*, Seoul, Republic of Korea, (**2020**)

[114] (Yazar bilgisi görünmüyor): "Research on DataOps Capability – Practice and Development", *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, (**2023**).

[115] Singhal, B.; Aggarwal, A.: "ETL, ELT and Reverse ETL: A Business Case Study", *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing and Communication Engineering (ICATIECE)*, (**2022**)

[116] Vilela, F. de A.; Ciferri, R. R.: "A Novel Solution to Perform Real-Time ETL Process Based on Non-Intrusive and Reactive Concepts*", 2021 International Conference on Computational Science and Computational Intelligence (CSCI),* (**2021**) 556–561

[117] Elmas, T.; Fabian, A. C.; Varol, C.; Varol, A.: "Hardware Performance Evaluation of Freeware ETL Tools", *2025 13th International Symposium on Digital Forensics and Security (ISDFS), IEEE*, (**2025**)

[118] Dash, B.; Swayamsiddha, S.: "Reverse ETL for Improved Scalability, Observability, and Performance of Modern Operational Analytics – A Comparative Review", *2022 OITS International Conference on Information Technology (OCIT), IEEE*, (**2022**)

[119] Vijayalakshmi, M.; Minu, R. I.: "Incremental Load Processing on ETL System through Cloud", *2022 International Conference for Advancement in Technology (ICONAT), IEEE*, Goa, India, Jan 21–22, (**2022**)

[120] Sreemathy, J.; Brindha, R.; Selva Nagalakshmi, M.; Suvekha, N.; Karthick Ragul, N.; Praveennandha, M.: "Overview of ETL Tools and Talend-Data Integration", *2021 7th International Conference on Advanced Computing & Communication Systems (ICACCS), IEEE*, (**2021**)

[121] Munawar: "Extract Transform Loading (ETL) Based Data Quality for Data Warehouse Development", *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI), IEEE*, Jakarta, Indonesia, (**2021**)

[122] Lal, D. K.; Suman, U.: "Towards Comparison of Real Time Stream Processing Engines", *2019 IEEE Conference on Information and Communication Technology (CICT)*, (**2019**).

[123] Lal, D. K.; Suman, U.: "Towards comparison of real time stream processing engines", *2019 IEEE Conference on Information and Communication Technology (CICT), IEEE,* (2019).

[124] Jiang, Z.; Liu, K.: “Real Time Interpretation and Optimization of Time Series Data Stream in Big Data”, *2018 3rd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA 2018), Chengdu, China, Apr. 20–22*, (**2018**) 243–247

[125] Dou, L.; Liu, S.; Wang, X.; Wu, L.: “The Application of Big Data Real-Time Stream Processing Technology in Earthquake Rapid Report”, *2021 7th International Conference on Hydraulic and Civil Engineering & Smart Water Conservancy and Intelligent Disaster Reduction Forum (ICHCE & SWIDR)*, (**2021**)

[126] Liao, X.; Gao, Z.; Ji, W.; Wang, Y.: “An Enforcement of Real Time Scheduling in Spark Streaming”, *2015 6th International Green and Sustainable Computing Conference (IGSC),* (**2015**).

[127] Raza, S.; Rahman, M.; Kamawal, S.; Toroghi, A.; Raval, A.; Navah, F.; Kazemeini, A.: “A Comprehensive Review of Recommender Systems: Transitioning from Theory to Practice”, *arXiv preprint*,(**2024**).

[128] Xu, X.; Zhang, D.; Liu, Q.; Lu, Q.; Zhu, L.: “Agentic RAG with Human-in-the-Retrieval”, *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*, IEEE, (**2025**)

[129] Feuerriegel, S.; Hartmann, J.; Janiesch, C.; Zschech, P.: “Generative AI”, *arXiv preprint*, (**2023**).

[130] Github/**yurtseveronr**: “onur-master-thesis-data (GitHub deposu)”, GitHub, URL: https://github.com/yurtseveronr/onur-master-thesis-data (Erişim tarihi: **05.09.2025**).

[131] Azizah, N. N.: “Cloud Computing Adoption Trends: A Systematic Literature Review of Organizational Perspectives”, *Journal of Computer Science, Information Technology and Telecommunication Engineering (JCoSITTE)*, **5(**1) (**2024**).

[132] Dangi, C. S.; Dhariwal, S.: “Exploring Cloud Deployment Services through Machine Learning: A Focus on AWS”, *2024 International Conference on Augmented Reality, Intelligent Systems, and Industrial Automation (ARIIA), IEEE*, (**2024**).

[133] Gohil, R.; Patel, H.: “Comparative Analysis of Cloud Platform: Amazon Web Service, Microsoft Azure, and Google Cloud Provider – A Review”, *The 15th IEEE International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE*, IIT Mandi, India, (**2024**).

[134] Dubey, P.; Tiwari, A. K.: “A Survey on Use of Amazon Web Services by Engineering Students”, *2023 IEEE Fifth International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, (**2023**).

[135] Piper, B.; Clinton, D.: “AWS Certified Cloud Practitioner Study Guide: Foundational (CLF-C02) Exam, Second Edition”, *John Wiley & Sons (Sybex)*, Hoboken, NJ, USA, (**2024**). ISBN 978-1-394-23563-6.

[136] Montgomery, T.: “AWS Certified Advanced Networking Study Guide: Specialty (ANS-C01) Exam, Second Edition”, *John Wiley & Sons (Sybex)*, Hoboken, NJ, USA, (**2024**).

[137] Mathieu, L.: “Mastering AWS Security, Second Edition”, *Packt Publishing*, Birmingham, UK, (**2024**).

[138] Ismail, J. A.: “Architecting Multisite Database Proxy using ProxySQL”, *Proc. 2nd International Conference on Technology Innovation and Its Applications (ICTIIA)*, IEEE, (**2024**)

[139] Devi, K.; Parthasarathy, R.; Unnamalai, K.; Perinba Athiban, S.; Manikandan, K.; Ramya, N.: “Fisherman Location Store, Updating, and Alert System using Cloud AWS with Satellite”, *2024 International Conference on Electronic Systems and Intelligent Computing (ICESIC)*, IEEE, (**2024**).

[140] Uthej, K.; Beena, B. M.; Nagandla, K. S. K.; Musunuru, N. K.: “Cloud-Infused AWS Services: Automobile Database Management System”, *2nd International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, IEEE, (**2024**).

[141] Rahman, A.; Kumar, D.: “Serverless Web Application”, *International Conference on Electrical, Electronics and Computing Technologies (ICEECT)*, IEEE, (**2024**)

[142] Pandey, M.; Pranave, K. C.; Kunda, N. S. S.; Dinesh Kumar, K.: "*Creating a Virtual Literary Hub: Leveraging AWS for Deploying an Application in the Cloud*", *3rd International Conference on Smart Systems for applications in Electrical Sciences (ICSSES)*, IEEE, (**2025**)

[143] Gulabani, S.: "Practical Amazon EC2, SQS, Kinesis, and S3: A Hands-On Approach to AWS", Apress, New York, USA, (**2017**)

[144] Pandey, M.; KC, P.; Kunda, N. S. S.; Kumar, K. D.: "Creating a Virtual Literary Hub: Leveraging AWS for Deploying an Application in the Cloud", *3rd International Conference on Smart Systems for Applications in Electrical Sciences (ICSSES),* (**2025**).

[145] Gupta, A.; Dhanda, N.; Gupta, K. K.: "Ingest and Visualize CSV Files using AWS Platform for Transition from Unstructured to Structured Data", *2023 11th International Conference on Emerging Trends in Engineering & Technology – Signal and Information Processing (ICETET-SIP), IEEE*, (**2023**).

[146] Ding, S.; Raman, V.: "Harness the Power of Generative AI in Healthcare with Amazon AI/ML Services", *2024 IEEE 12th International Conference on Healthcare Informatics (ICHI)*, (**2024**).

[147] Remya, S.; Abburi, S. K.; Maddala, H. S. M. K.; Marasani, J.; Samudrala, Y.: "Examining Amazon Customer Reviews using PySpark and AWS: A Data Lake Approach", *14th IEEE International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IIT Delhi, India, (**2023**).

[148] Srivastava, M.; Yadav, P.: "Build a Log Analytic Solution on AWS", *2021 5th International Conference on Information Systems and Computer Networks (ISCON), IEEE*, (**2021**).

[149] Kodali, R. K.; Karthik, T.: "Water Quality Prediction using AWS", *14th IEEE International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IIT Delhi, India, (**2023**).

[150] Tirtakusuma, D. I.; Sukaridhoto, S.; Budiarti, R. P. N.; Mulyo, D.; Alfarezi, F.; Aminputra, B. B.; Hakim, O. S.: "*Real-time Monitoring Using AWS Cloud Platform for Agriculture Soi*l", *2024 IEEE International Symposium on Consumer Technology (ISCT)*, (**2024**).

[151] de Carvalho, L. R.; de Araujo, A. P. F.: "Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators", *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, (**2020**)

[152] Arrais, M.; Martinez Bravet, R.; Ciccone, L.; Nobre Cocharero, A.; Kurauchi, É.; Rozestraten, H.: "AWS Certified Database Study Guide — Specialty (DBS-C01) Exam", *John Wiley & Sons, Hoboken, NJ, USA*, (**2023**).

[153] Orr, M.; Cao, Y.-X.: "Mastering Elastic Kubernetes Service on AWS", *Packt Publishing*, (**2023**)

[154] Wang, C.-C.; Tseng, C.-C.: "Provisioning a Reliable Kubernetes Cluster with CI/CD", *2024 International Computer Symposium (ICS)*, IEEE, (**2024**)

[155] Mohan, A.; Turk, A.; Gudimetla, R. S.; Tikale, S.; Hennesey, J.; Kaynar, U.; Cooperman, G.; Desnoyers, P.; Krieger, O.: "M2: Malleable Metal as a Service", arXiv preprint arXiv:1801.00540, (**2018**).

[156] Khomh, F.; Adams, B.; Dhaliwal, T.; Zou, Y.: "Understanding the Impact of Rapid Releases on Software Quality — The Case of Firefox", *Empirical Software Engineering*, 20 (**2015**) 336–373.

[157] Amazon Web Services, https://aws.amazon.com/pm/kinesis/ , Erişim Tarihi: **01.09.2025**

[158] Amazon Web Services, https://aws.amazon.com/tr/firehose/, Erişim Tarihi: **01.09.2025**

[159] Qi, B.; Yu, Y.; Li, X.: “The Utilisation and Implementation of the Istio Framework in the Architectonic Progression of Online Learning Platform”, *2024 13th International Conference of Information and Communication Technology (ICTech)*, (**2024**) 182–186.

ÖZGEÇMİŞ

Onur YURTSEVER, 2018 yılında Kocaeli Üniversitesi'nde Elektrik Mühendisliği lisans bölümünü tamamlamıştır, 2019 yılında Sistem ve Ağ Uzmanlığı eğitimi almıştır. 2021 yılından itibaren Commencis Teknoloji A.Ş firmasında Bulut Bilişim Uzmanı olarak görev yapmaktadır.