KURUMSAL DÜZEYDE DEVOPS KÜLTÜRÜNÜN ENTEGRASYONU VE ALTYAPI OTOMASYONUNUN MODERNİZASYONU[1]

Bayraktar KARAGÖZ[2]

Ümit Çiğdem TURHAL[3]

1. GİRİŞ

Günümüzde kurumların rekabet gücünü sürdürebilmesi, teknolojik değişimlere hızlı uyum sağlama becerilerine bağlıdır. Dijital dönüşüm, yalnızca bilgi teknolojileri (IT) altyapısının yenilenmesini değil; iş süreçlerinin yeniden yapılandırılmasını ve çevik organizasyonel yapıların kurulmasını da gerektirir. Yazılım sistemleri bu dönüşümün merkezinde yer almakta; otomasyon, çeviklik ve entegrasyon odaklı yeni yazılım geliştirme yöntemlerini beraberinde getirmektedir.

Geleneksel yazılım modelleri hiyerarşik yapıları nedeniyle esnek değildir ve değişen ihtiyaçlara yeterince hızlı yanıt veremez. Ayrıca geliştirici ve operasyon ekiplerinin ayrı çalışması, iletişim eksikliklerine ve sistem sürekliliğini tehdit eden sorunlara yol açar. Bu sorunları aşmak amacıyla geliştirilen DevOps yaklaşımı, yazılım geliştirme ve operasyon süreçlerini entegre ederek iş birliğini, otomasyonu ve hızlı teslimatı

---

[1] Bu kitap bölümü Bilecik Şeyh Edebali Üniversitesi (BSEU) Lisansüstü Eğitim Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı'nda yapılmış olan tez çalışmasından üretilmiştir.

[2] Bilecik Şeyh Edebali Üniversitesi, Lisansüstü Eğitim Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı, 5740301@ogrenci.bilecik.edu.tr, ORCID:0009-0006-5391-3382

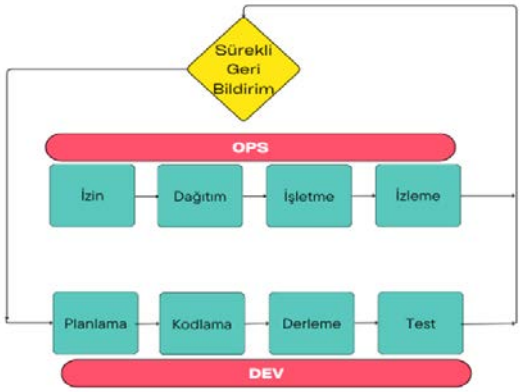
[3] Doç. Dr., Bilecik Şeyh Edebali Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü, ucigdem.turhal@bilecik.edu.tr, ORCID:0000-0003-2387-1637

mümkün kılar (Istifarulah ve Tiaharyadini, 2023). DevOps, yazılım yaşam döngüsündeki tüm aktörler arasında sürekli iletişim ve geri bildirim sağlayarak güvenilir ve sürdürülebilir bir geliştirme ortamı sunar.

DevOps'un uygulanabilirliğini somutlaştıran en önemli unsurlardan biri, yazılım geliştirme sürecini döngüsel ve otomasyona dayalı bir yapıda tanımlayan DevOps yaşam döngüsüdür (Şekil 1.1). Bu yaşam döngüsü; planlama, kodlama, oluşturma (build), test etme, yayınlama (release), dağıtım (deploy), işletme (operate) ve izleme (monitor) gibi sekiz temel aşamadan oluşur. DevOps yaşam döngüsü, yazılım projelerinin fikir aşamasındaki planlama ve tasarımdan başlayarak, kod geliştirme ve sürüm yönetimi süreçlerine; ardından otomatik testler, sürekli entegrasyon (CI) ve konteyner tabanlı paketleme adımlarına geçişi kapsayan bir akış sunar (Alhamidi, 2017). Her bir aşamada otomasyon boru hatları (pipeline) devreye girerek kod değişikliklerinin hızlıca derlenmesi, test edilmesi ve bir sonraki ortama taşınması sağlanır. Onaylanan sürümler, sürekli dağıtım (CD) mekanizmalarıyla güvenli ve tekrarlanabilir şekilde canlı ortama aktarılır. Bu süreçte canlıya geçen uygulamalar; performans, erişilebilirlik ve güvenlik metrikleri açısından kesintisiz olarak izlenir. İzleme sonucunda elde edilen veriler, geri bildirim döngüsüyle bir sonraki planlama evresine taşınır. Bu yapı, yalnızca yazılımın daha hızlı ve hatasız şekilde üretime alınmasını değil; aynı zamanda sistem kararlılığının sağlanmasını, kullanıcı memnuniyetinin artırılmasını ve organizasyonel çevikliğin sürekli desteklenmesini mümkün kılar.

Bu bağlamda DevOps'un yalnızca bir teknoloji yaklaşımı değil, aynı zamanda organizasyonel dönüşüm aracı olduğu kabul edilmektedir. Günümüz dijital ekonomisinde, kurumsal ölçekli yazılım geliştirme ve IT operasyonları sadece teknik yeterlilikle değil; aynı zamanda çevik iş kültürünün

kuruma entegre edilmesiyle sürdürülebilir başarıya ulaşabilmektedir. Bu kültürel dönüşümün başarısı, genellikle altyapı otomasyon süreçlerinin modernizasyonu ile doğrudan ilişkilidir. Infrastructure as Code (IaC) gibi yaklaşımlar sayesinde altyapı; tanımlanabilir, versiyonlanabilir ve tekrarlanabilir hâle gelirken, sistemlerin esnekliği ve ölçeklenebilirliği önemli ölçüde artmaktadır. Dolayısıyla, kurumsal yapılarda DevOps'un etkili entegrasyonu, ancak altyapı süreçlerinin otomasyon odaklı yeniden yapılandırılmasıyla mümkün hâle gelmektedir.



Şekil 1.1. DevOps Yaşam Döngüsü

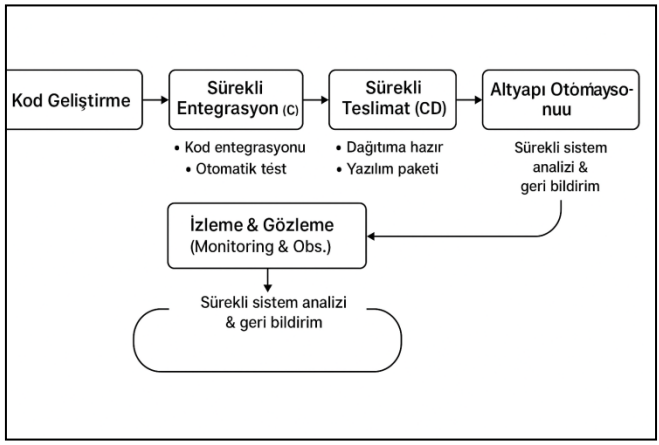
CI, CD, otomatik testler ve izleme araçları gibi bileşenlerle desteklenen DevOps, dijital dönüşüm hedeflerine ulaşmak isteyen kurumlar için stratejik bir zorunluluk hâline gelmiştir (Kim vd., 2016).

1.1. DevOpsBileşenleri, Araçları ve Uzantıları

DevOps yaklaşımının temelini oluşturan dört ana teknik bileşen; CI, CD, altyapı otomasyonu ve izleme-gözleme süreçleridir (Şekil 1.2).

CI, geliştiricilerin yaptığı kod değişikliklerinin merkezi bir sürüm kontrol sistemine sık aralıklarla entegre edilmesini ve

her entegrasyonun otomatik testlerle doğrulanmasını sağlar; böylece hatalar erken tespit edilerek geliştirme sürecinin sürekliliği korunur. Bu süreci takip eden CD aşamasında, testleri başarıyla geçen kod her an dağıtıma hazır olacak şekilde paketlenir ve üretim ortamına geçiş için manuel müdahaleye gerek kalmaksızın teslimata uygun biçimde bekletilir. Dağıtım sürecinin bu denli otomatik ve güvenilir hâle gelebilmesi, büyük ölçüde altyapının IaC yaklaşımıyla tanımlanarak otomatik yönetilmesini mümkün kılan altyapı otomasyonuna dayanır. Bu yaklaşım, sistem kaynaklarının tutarlı, tekrarlanabilir ve versiyonlanabilir olmasını sağlar. Tüm bu süreçlerin güvenli ve sağlıklı ilerleyebilmesi ise izleme ve gözleme mekanizmaları ile desteklenir; sistem performansı, kaynak kullanımı ve hata oranları sürekli takip edilir, elde edilen veriler analiz edilerek sistem davranışları hakkında içgörü üretilir ve olası sorunlara erken müdahale imkânı sağlanır.



Şekil 1.2. DevOps Temel Bileşenleri

DevOps uygulamalarının etkin ve sürdürülebilir bir şekilde yürütülmesi, kapsamlı bir araç ekosistemiyle desteklenmektedir. Bu araçlar, yazılım geliştirme sürecinin her aşamasında otomasyonu, şeffaflığı ve izlenebilirliği artırarak DevOps'un teknik altyapısını oluşturur. CI/CD süreçlerinde en

çok tercih edilen araçlar arasında Tekton, Jenkins, GitLab CI, CircleCI ve ArgoCD yer almaktadır. Bu araçlar sayesinde kod entegrasyonu, test otomasyonu ve dağıtım adımları belirli tetikleyicilerle başlatılarak yazılımın sürekli olarak güncellenmesi sağlanır. Özellikle Jenkins ve Tekton, modüler yapısı ve geniş eklenti desteği ile kurumsal düzeyde özelleştirilmiş CI/CD hatları oluşturulmasına imkân tanımaktadır.

DevOps'un temel bileşenlerinden biri olan konteyner teknolojileri, uygulamaların taşınabilirliğini ve ortamdan bağımsız çalışmasını kolaylaştırmaktadır. Bu alanda öne çıkan araçlardan Docker, uygulamaların tüm bağımlılıklarıyla birlikte konteyner içinde çalışmasını sağlarken, Kubernetes bu konteynerlerin ölçeklendirilmesi, dağıtılması ve yönetilmesini üstlenmektedir. Kubernetes'in sunduğu otomatik yük dengeleme, yeniden başlatma ve kendi kendini iyileştirme gibi yetenekler, büyük ölçekli sistemlerde kararlılığı artırmakta ve operasyonel yükü azaltmaktadır.

İzleme ve analiz süreçleri ise sistemin performansını gerçek zamanlı olarak izlemeye ve potansiyel sorunları önceden belirlemeye olanak tanır. Bu amaçla kullanılan araçlar arasında Prometheus ve Grafana ön plana çıkmaktadır. Prometheus, zaman serisi verilerini toplayarak sistemin kaynak kullanımını izlerken; Grafana, bu verileri anlamlı paneller ve grafikler hâlinde sunarak operasyonel ekiplerin sistem durumu hakkında anlık içgörüler elde etmesini sağlar. Bu araçlar, hata tespiti, kapasite planlaması ve hizmet düzeyi anlaşmalarının (SLA) takibi açısından kritik rol oynamaktadır.

DevOps'un evrilmesiyle birlikte güvenlik, operasyonel zekâ ve sürüm yönetimi süreçleri de daha derin entegrasyon gerektiren alanlara dönüşmüştür. Bu bağlamda ortaya çıkan DevSecOps yaklaşımı, güvenlik kontrollerinin yazılım

geliştirme yaşam döngüsünün (SDLC) her aşamasına entegre edilmesini hedefler. Geleneksel olarak sürecin sonuna bırakılan güvenlik testleri, DevSecOps sayesinde otomatikleştirilmiş araçlarla sürekli olarak yürütülür. Bu alanda öne çıkan araçlar arasında SonarQube, kod kalitesini ve güvenlik açıklarını analiz ederken; Clairve AquaSecurity, açık kaynak kütüphaneler ve konteyner güvenliği için otomatik tarama ve raporlama işlevleri sunar. Bu araçlar, sadece uygulama katmanını değil, aynı zamanda yapılandırma dosyalarını ve altyapı tanımlarını da analiz ederek geniş kapsamlı bir güvenlik sağlar.

Öte yandan, GitOps yaklaşımı, Git deposunu hem kodun hem de altyapının "tek doğruluk kaynağı (singlesource of truth)" olarak konumlandırır. Bu modelde, Kubernetes gibi orkestrasyon sistemleri, doğrudan Git deposundaki manifest dosyalarına göre sistem durumunu senkronize eder. GitOps'un temel araçlarından biri olan ArgoCD, Git tabanlı dağıtım süreçlerini gerçek zamanlı izleyerek sürüm geri alma, fark analizi ve erişim denetimi gibi işlevleri otomatikleştirir. GitOps, dağıtım süreçlerinde hem şeffaflığı artırmakta hem de manuel müdahaleye gerek bırakmaksızın sistem tutarlılığını sağlamaktadır.

Bunun yanı sıra AIOps (ArtificialIntelligencefor IT Operations), DevOps ortamlarında üretilen büyük hacimli sistem verilerinin yapay zekâ ve makine öğrenmesi teknikleriyle analiz edilmesini amaçlar. AIOps araçları, sistem loglarını, performans metriklerini ve kullanıcı etkileşimlerini analiz ederek anormallikleri tespit eder, otomatik kök neden analizi (rootcauseanalysis) gerçekleştirir ve bazı durumlarda öngörüye dayalı aksiyonlar alır. Bu kapsamda Moogsoft, Dynatrace ve Splunk ITSI, AIOps uygulamaları arasında en bilinen çözümlerden olup, operasyon ekiplerine proaktif müdahale ve karar destek sistemi sunmaktadır.

2. KURUMSAL UYGULAMALARDA DEVOPS TEMEL PRENSİPLERİNE BAKIŞ

Kurumsal organizasyonlarda DevOps dönüşümü, teknik araçların ve otomasyonun ötesinde, köklü bir kültürel değişimi gerektirmektedir. Forsgren ve diğerleri (2018)'nin araştırmaları, başarılı DevOps uygulamalarının temelinde yatan en önemli faktörün organizasyonel kültür olduğunu ortaya koymaktadır. Bu kültürel dönüşüm; ekipler arası iş birliği, sürekli öğrenme, deneyimleme ve hata toleransı gibi temel değerleri içermektedir. Özellikle, PuppetLabs'inState of DevOps Report 2021 raporuna göre, güçlü bir DevOps kültürüne sahip organizasyonların yazılım dağıtım sürelerini %96'ya varan oranlarda azaltabildikleri ve değişiklik başarı oranlarını %80'in üzerine çıkarabildikleri gözlemlenmiştir.

Modern yazılım geliştirme süreçlerinde DevOps uygulamaları; CI/CD gibi otomasyona dayalı pratikleri, altyapının kod olarak yönetimini IaC ve mikroservis mimarilerini içermektedir. Humble ve Farley (2010)'e göre bu pratikler, organizasyonların daha hızlı, güvenilir ve öngörülebilir yazılım teslimatı yapmalarını sağlamaktadır. Bununla birlikte, Lwakatare ve diğerleri (2019) tarafından vurgulandığı üzere, bu teknik pratiklerin başarısı, organizasyonun DevOps kültürünü ne ölçüde benimsediğiyle doğrudan ilişkilidir.

Bu bağlamda kurumsal düzeyde DevOps'un başarıyla uygulanabilmesi, yalnızca teknik altyapının değil; aynı zamanda ekip yapılanmasının ve süreç yönetiminin de DevOps ilkeleri doğrultusunda yeniden şekillendirilmesini gerektirir. DevOps'un temel prensipleri; iş birliği, otomasyon, sürekli geri bildirim, ölçülebilirlik, paylaşılan sorumluluk ve sürekli iyileştirme başlıkları altında toplanabilir. Bu ilkeler, yazılım geliştirme ve operasyon ekipleri arasındaki yapay sınırların kaldırılmasını ve

süreçler arası etkileşimin artırılmasını hedefler. İş birliği odaklı bu yapı, silo mantığının yerini şeffaflık ve ortak hedefe yönelimle değiştirirken; otomasyon, hız ve doğruluk kazandırır. Sürekli geri bildirim mekanizmaları sayesinde sistem kararlılığı korunur, kullanıcı odaklılık artar. Ayrıca tüm ekiplerin yazılım kalitesi ve sistem sürekliliği üzerinde ortak sorumluluk taşıması teşvik edilir. Bu yapı, özellikle sürekli gelişimi ve organizasyonel çevikliği teşvik eden kültürlerde sürdürülebilir başarı sağlar.

DevOps'un kurumsal uygulamalardaki önemi, dijital dönüşüm sürecinin hızlanmasıyla birlikte daha da artmıştır. Gartner'ın (2023) araştırmasına göre, Fortune 500 şirketlerinin %75'i DevOps pratiklerini kurumsal stratejilerinin merkezine yerleştirmiş durumdadır. Bu dönüşüm, organizasyonların sadece teknolojik altyapılarını değil, aynı zamanda iş yapış şekillerini ve organizasyonel kültürlerini de derinden etkilemektedir. Willis ve Edwards (2022)'in çalışması, başarılı DevOps implementasyonlarının organizasyonlarda inovasyon kültürünü güçlendirdiğini ve çalışan memnuniyetini artırdığını göstermektedir.

Kurumsal uygulamalarda DevOps adaptasyonu, beraberinde bir dizi zorluğu da getirmektedir. Özellikle büyük ölçekli organizasyonlarda mevcut süreçlerin ve kültürel yapının değişimi önemli bir mücadele alanı oluşturmaktadır. Rodriguez ve diğerleri (2020)'nin araştırması, kurumsal DevOps dönüşümlerinde karşılaşılan en büyük zorlukların başında değişim direnci, teknik borç yönetimi ve ölçeklenebilirlik konularının geldiğini ortaya koymaktadır. Bu zorlukların üstesinden gelmek için organizasyonların sistematik bir yaklaşım benimsemeleri ve değişim yönetimi süreçlerini etkin bir şekilde yürütmeleri gerekmektedir.

DevOps'un sunduğu değer önerisi, modern iş dünyasının temel gereksinimlerini karşılamada kritik öneme sahiptir. Sürekli entegrasyon ve teslimat pratikleri, organizasyonların pazar değişimlerine hızlı yanıt verebilmelerini sağlarken; otomasyon ve standardizasyon uygulamaları operasyonel verimliliği artırmaktadır. Sharma ve Coyne (2021)'nin çalışması, DevOps olgunluk seviyesi yüksek organizasyonların, rakiplerine kıyasla %200'e varan oranlarda daha hızlı yenilik yapabildiğini ve müşteri memnuniyetinde önemli artışlar sağladığını göstermektedir.

Güvenlik perspektifinden bakıldığında ise DevOps'un "DevSecOps" evrimiyle birlikte güvenlik pratiklerinin yazılım geliştirme yaşam döngüsüne entegre edilmesi büyük önem kazanmıştır. Bass ve diğerleri (2021)'nin araştırması, güvenlik kontrollerinin otomatikleştirilmesi ve sürekli izleme pratiklerinin, organizasyonların güvenlik açıklarını %70 oranında azaltabildiğini göstermektedir. Bu entegrasyon, modern kurumsal uygulamaların karşı karşıya kaldığı siber güvenlik tehditlerinin proaktif bir şekilde yönetilmesini sağlamaktadır.

3. KURUMSAL DEVOPS DÖNÜŞÜMÜNE İLİŞKİN BİR UYGULAMA VE DEĞERLENDİRME

Kurumsal yapılarda DevOps kültürüne geçiş, mevcut sistemin analizini, eksikliklerin belirlenmesini ve kuruma özel çözüm önerilerinin geliştirilmesini içeren bütüncül bir dönüşüm sürecidir. Bu bölümde, DevOps entegrasyonunun başlangıç aşamasında mevcut yapının durumu ve geçiş öncesi karşılaşılan sorunlar ele alınmakta; DevOps altyapısına geçmek isteyen kurumlar için rehber niteliğinde bir model sunulmaktadır.

Kurum genelinde konteyner ve küme yönetimi teknolojileri aktif olarak kullanılmamaktadır. Oysa konteyner

mimarisi, yazılımların farklı ortamlarda tutarlı şekilde çalışmasını sağlayarak geliştirme süreçlerinde çeviklik ve hız kazandırır. Bazı uygulamalar test ortamında Docker ile çalışsa da, bu mimari henüz kurum genelinde yaygınlaşmamıştır.

Şekil 3.1'den görüldüğü üzere, kurum içerisinde canlı sistemde CI/CD sürecinde çalışan bir uygulama bulunmamaktadır. Oysa bir DevOps altyapısında CI/CD süreçlerinin kurulmuş olması kritik öneme sahiptir. Sürekli entegrasyon ve sürekli dağıtım süreçleri, günümüz yazılım dünyasında yazılım kalitesinin, güvenliğinin ve sürdürülebilirliğinin sağlanmasında temel rol oynamaktadır.

Kurumda statik kod analizi için Fortify kullanılmakta olup, pipeline entegrasyonu ile yeni bir araca daha ihtiyaç duyulmaktadır. Ayrıca kurum genelinde merkezi bir loglama altyapısı bulunmamaktadır. Uygulama loglarının yeterli düzeyde alınamaması, performans ve hata takibini zorlaştırmakta; bu durum, sistemlerde yaşanan sorunların kaynağının tespitini güçleştirmektedir. Bu nedenle kapsamlı bir izleme ve merkezi loglama altyapısının kurulması gerekmektedir.

Tespit Edilen Sorun	Sorunun Kaynağı
Java Versiyonunun Güncel Olmaması	Java 8 desteği yakında sonlanacak
Loglama ve İzleme Eksikliği	hata durumlarının analiz edilmesi ve genel akışın takibi açısından kritik öneme sahiptir
Veritabanı Değişikliklerine Yönelik Versiyon Kontrol Eksikliği	Veritabanı Devops sürecine entegre değil
Kod Kalite Analizi Eksikliği	Fortify, daha çok güvenlik katmanına odaklanır.
CI/CD Süreci Eksikliği	Uygulamalarda CI/CD süreci yoktur
Standart Yazılımsal Çözümlerin Eksikliği	bir standard olmaması benzer işlere her projede efor harcanmasına sebep olmaktadır.
REST API'lar İçin Standart Eksikliği	Rest API'ları için Daire Başkanlıkları genelinde bir standart yok
İçerik Yönetimi (Content Manager) olamaması	dosya yükleme ihtiyaçları için dosya sunucuları kullanılmaktadır.
Uygulama Testlerinin Eksikliği	uygulamalarda birim test, entegrasyon testi gibi testler bulunmamaktadır

Şekil 3.1. DevOpsa geçiş sürecinde tespit edilen sorun ve kaynağı

Mevcut uygulamaların çoğu web uygulama sunucularında (WAS) çalışmakta, bu da yüksek maliyet ve eski Java sürümlerine bağımlılık yaratmaktadır. Daha esnek ve güncel bir yapı için sunucu altyapısının yeniden düzenlenmesi önerilmektedir. Kurumda test süreçleri yetersizdir; oysa CI/CD hatlarında sağlıklı bir dağıtım için testler kritik önemdedir. Ayrıca merkezi bir depolama altyapısının bulunmaması, sistem bileşenlerinin yönetimini ve güvenliğini zorlaştırmaktadır.

Yukarıda belirtilen DevOps'a geçişle ilgili eksikliklere ek olarak, Şekil 3.1'de görülen yazılıma ilişkin bazı standartizasyon süreçlerinin de DevOps altyapısı kurulurken entegre edilmesinin süreçleri kolaylaştırdığı ve yazılım ekiplerinin DevOps kültürüne adaptasyonunu desteklediği gözlemlenmiştir.

3.1. Örnek Kurumsal DevOps Dönüşüm Senaryosu

Senaryo kapsamında, geleneksel yazılım geliştirme ve dağıtım süreçlerine sahip bir kamu kurumunun, mevcut IT altyapısını DevOps prensiplerine uygun hâle getirme süreci ele alınmıştır. Kurumun ilk aşamada karşılaştığı temel zorluklar şunlardır:

- Geliştirme ve operasyon ekipleri arasında iletişim eksikliği,
- Dağıtımların manuel yapılması nedeniyle yüksek hata oranı,
- Geribildirim mekanizmalarının eksikliği,
- Altyapı konfigürasyonlarının dokümante edilmemesi.

Dönüşüm süreci üç faza ayrılmıştır (Şekil 3.2):

1. Planlama ve Kültürel Hazırlık:Ekipler arası ortak hedef tanımı, eğitimler ve iş birliği kültürünün oluşturulması.
2. Teknik Dönüşüm:

- Tekton ile CI süreçlerinin kurulması,
- SonarQube ve Clair ile otomatik testlerin entegrasyonu,
- Docker ile uygulama konteynerleştirme,
- Kubernetes ile uygulama orkestrasyonu,
- Prometheus, Grafana ve Jeager ile sistem izleme altyapısının kurulması.

3. Operasyonel Otomasyon:
    - Webhook ve trigger(tetikleyiciler) ile kod değişikliklerinde altyapının otomatik çalışması,
    - ArgoCD ile GitOps tabanlı dağıtım süreçlerinin başlatılması,
    - Vault ile güvenli yapılandırma yönetimi.

Dönüşüm sonucunda, uygulama dağıtım süreleri %85 oranında azalmış, hata oranı belirgin şekilde düşmüş ve geri bildirim süreleri önemli ölçüde kısalmıştır. Kurum, manuel süreçlerin otomasyona geçirilmesiyle birlikte yazılım geliştirme verimliliğini artırmış, operasyonel yükü azaltmış ve kurumsal çevikliği güçlendirmiştir.

Şekil 3.2. DevOps dönüşüm yolculuğu

4. DEVOPS METODOLOJİSİNİN UYGULANMASI, DEVOPS KÜLTÜRÜNÜN BENİMSENMESİ VE DEĞERLENDİRİLMESİ

Bu bölümde, kurumda DevOps altyapısının uygulanışı, CI/CD süreçlerinin işleyişi, kullanılan araçlar, DevSecOps entegrasyonu ve DevOps kültürünün yaygınlaştırılması ele alınmaktadır. Mevcut projede, manuel süreçlerin otomasyona geçirilmesi ve DevOps prensiplerinin benimsenmesi amaçlanmaktadır.

Kurulan yapıda GitLab, kod yönetimi ve otomatik pipeline tetikleyici olarak merkezde yer almaktadır. Tekton ile oluşturulan pipeline; Maven ile derleme, SonarQube ile kod analizi ve Kaniko ile konteyner imajı oluşturma adımlarını içermektedir. Oluşan imajlar Quay'e aktarılmakta ve Clair ile güvenlik taramasından geçirilmektedir.

4.1. Devops Süreçlerinde Altyapı Otomatizasyonu

DevOps uygulamalarında altyapı otomasyonu, yazılım geliştirme ve dağıtım süreçlerinin hızlandırılmasında, insan hatalarının azaltılmasında ve tekrarlayan işlemlerin güvenilir biçimde yürütülmesinde kritik bir rol üstlenir. Ebert ve diğerleri (2016), "DevOps uygulamalarında altyapı otomasyonu, tekrarlanabilir ve güvenilir dağıtım süreçlerinin anahtarıdır" sonucuna varmışlardır. Bu kapsamda Tekton–GitLab entegrasyonu, DevOps altyapısında tam otomatik bir iş akışı sunarak önemli bir örnek teşkil eder. Benzer şekilde kurumlar, GitLab–Jira gibi araçlar arasında da otomasyon temelli bağlantılar kurgulayarak süreçlerini uçtan uca entegre edebilirler.

Tekton, Kubernetes üzerinde çalışan açık kaynaklı bir CI/CD framework'ü olup, sürekli entegrasyon aşamasında etkin biçimde kullanılırken; GitLab, kod yönetimi, sürüm takibi ve proje organizasyonu için merkezi bir platform işlevi görür. Bu

iki sistem arasındaki otomatik veri akışı webhook'lar aracılığıyla sağlanır; örneğin bir merge isteği (mergerequest) oluşturulduğunda GitLab tarafından tetiklenen webhook, Tekton'da tanımlı pipeline sürecini devreye sokar.

Otomatikleştirilen bu pipeline, Maven ile derleme, SonarQube ile kod analizi, Kaniko ile konteyner imajı oluşturma ve bu imajın Quay gibi bir containerregistry'ye gönderilmesi gibi adımlardan oluşur. Ardından Argo CD, GitOps prensipleri çerçevesinde Quay'deki yeni sürümü algılar ve Helm şablonlarını kullanarak Kubernetes ortamına parametrik olarak dağıtımı gerçekleştirir. Helm, dağıtım yapılandırmalarının farklı ortamlar için esnek biçimde uyarlanmasını sağlar.

Test ortamlarında bu iş akışı uçtan uca otomatik ilerlerken, canlı sistemlerde güvenlik ve kararlılık gereksinimleri nedeniyle belirli adımlar manuel onaya tabi tutulur. Üretim ortamına geçişler genellikle kontrollü şekilde elle tetiklenir ve bir dizi ek güvenlik doğrulamasıyla desteklenir. Tekton–GitLab iş birliği; test süreçlerinde hız ve doğruluk sağlarken, üretim sistemlerinde güvenlik ve denetim gereksinimlerini karşılayan esnek bir altyapı otomasyonu modelini temsil eder. Bu aşamada Clair aracı ile güvenlik taraması gerçekleştirilmekte, olası açıklar ve bağımlılık sorunları proaktif şekilde tespit edilmektedir. İzleme ve gözleme süreçlerinde Loki logların toplanması, Grafana ise görselleştirme amacıyla kullanılmaktadır. JeagerAPM uygulama performansını izlerken, Prometheus ve Alertmanager sistem kaynaklarının takibi ve alarm üretimi için devreye girmektedir.

Dağıtım aşamasında Argo CD ile uygulama Kubernetes ortamına kontrollü ve otomatik şekilde yerleştirilmektedir. Helm kullanılarak dağıtılan uygulamalar, gerekli bileşenlerle birlikte paketlenerek kolayca yönetilebilir hâle getirilmektedir. Bu yapı,

GitOps mantığını desteklemekte ve uygulama dağıtımlarında güvenilirlik ve tutarlılık sağlamaktadır.

"DevOps kültürünün kuruma entegrasyonu sırasında en büyük kazanımlardan biri, geliştirme ve operasyon ekipleri arasındaki silo yapılarının ortadan kaldırılması olmuştur"(Gupta vd., 2024) Proje bazlı yaklaşımdan ürün bazlı yaklaşıma geçilmesiyle birlikte ekipler, yazılım yaşam döngüsünün tüm aşamalarında daha fazla sorumluluk almış, dağıtım sıklığı artarken hata oranı azalmıştır. Yapılan iç değerlendirmelerde, yazılım teslim sürelerinde %80'e varan kısalma, sistem erişilebilirliğinde ise %95'in üzerinde süreklilik elde edildiği raporlanmıştır.

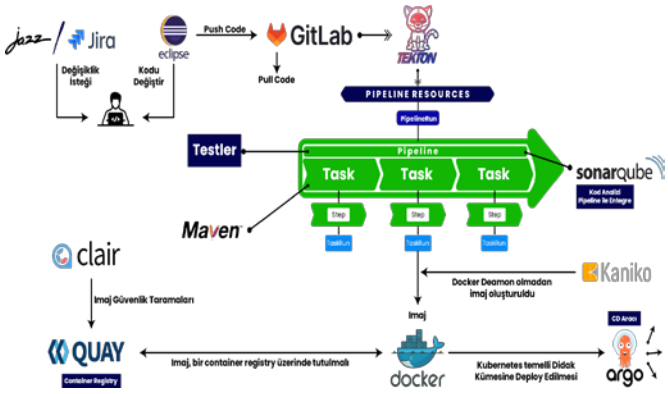
Kurum içerisinde DevOps kültürünün benimsenmesi için çeşitli iç eğitim programları, rehber dokümantasyonlar ve teknik mentorluk faaliyetleri yürütülmüştür. Ayrıca sistem performansı, dağıtım başarısı ve kullanıcı memnuniyeti gibi göstergeler periyodik olarak ölçülerek sürekli iyileştirme döngüsü desteklenmiştir.

Bu dönüşüm sonucunda, yazılım geliştirme yaşam döngüsünün tüm aşamalarında otomatizasyon sağlanmış, manuel müdahalelerden kaynaklanan hata riskleri minimize edilmiş ve geliştirme süreçlerinin verimliliği artırılmıştır. Ayrıca, ekipler arası iş birliği güçlenmiş ve değişikliklerin canlı ortama aktarılması daha güvenli ve hızlı hâle gelmiştir. Sonuç olarak, DevOps metodolojisinin uygulanması sadece operasyonel verimlilik açısından değil; organizasyonel çeviklik, kalite odaklılık ve teknolojiye hızlı adaptasyon açısından da önemli bir değer yaratmıştır. DevSecOps paradigması, güvenliği yazılım geliştirme yaşam döngüsü boyunca temel bir bileşen olarak yerleştirerek, riskleri erken tespit eder ve hızlı geliştirme süreçleriyle dengeler (Chittibala, 2023).Önümüzdeki süreçte, bu yapının DevSecOps açısından var olanlara ek olarak daha

gelişmiş güvenlik politikalarıyla genişletilmesi ve otomasyon kapsamının derinleştirilmesi hedeflenmektedir.

Mevcut yazılım projesinde, geliştirme ve dağıtım süreçlerinin modernizasyonu için kapsamlı bir dönüşüm yapılmıştır. Şu anda manuel yürütülen süreçlerin otomatikleştirilmesi ve DevOps pratiklerinin benimsenmesi hedeflenmektedir. Bu dönüşümün temelinde, CI/CD yaklaşımının uygulanması yer almaktadır. CI/CD yöntemleri, iş birliği ve otomasyonu teşvik ederek yüksek kaliteli yazılımın hızlı teslimini sağlar; bu sayede geliştirme döngüleri kısalır ve hata oranları düşer (Gupta vd., 2024).

Kurulan yapıda, kaynak kod yönetimi için tercih edilen GitLab deposu merkezi rol oynayacaktır. Geliştirme ekiplerinin kod değişiklikleri, otomatik tetiklenen bir pipeline sürecini başlatacaktır. Bu pipeline'ı oluşturacak Tekton altyapısı, konteyner tabanlı mimarisi sayesinde yüksek ölçeklenebilirlik ve esneklik sunacaktır (Şekil 4.1). Derleme sürecinde Maven kullanılarak projenin inşası gerçekleştirilecek, bu aşamada birim testlerin koşturulması ve kod kalitesinin değerlendirilmesi sağlanacaktır. Kod kalitesi kontrolü için entegre edilecek SonarQube aracı, yazılım kalite metriklerini analiz ederek potansiyel sorunları erken aşamada tespit edecektir.

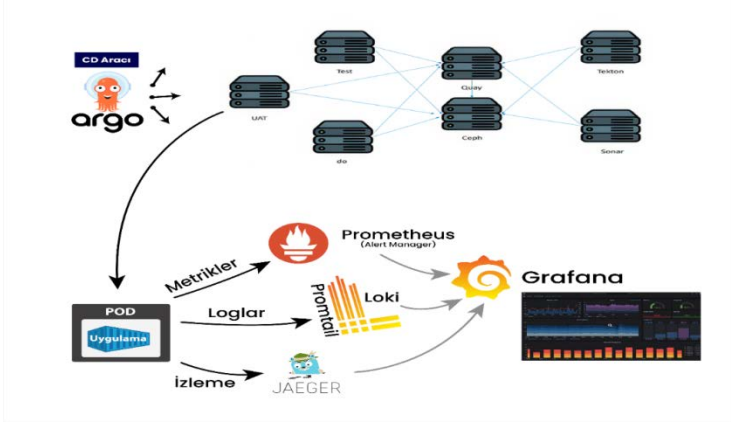


Şekil 4.1. CI Süreci

Başarılı derleme sonrasında, uygulamanın konteyner imajı Kaniko aracılığıyla oluşturulacaktır. Üretilen imajlar, merkezi bir depo olan Quay'de saklanacak ve bu aşamada Clair aracı kullanılarak güvenlik taramasından geçirilecektir. Bu tarama, olası güvenlik açıklarının ve bağımlılık sorunlarının proaktif olarak belirlenmesini sağlayacaktır.

Monitoring kısmında ise Loki, logların toplanması amacıyla Grafana ise logların görüntülenmesi ve veri görselleştirilmesi amacıyla kullanılması planlanmaktadır. İzleme-takip etme aracı olarak JeagerAPM, uyarı yöneticisi olarak ise Prometheus kullanılacaktır.

Şekil 4.2 de görüldüğü gibi dağıtım aşamasında Argo CD kullanılarak, uygulamanın Kubernetes ortamına kontrollü ve otomatik bir şekilde yerleştirilmesi gerçekleştirilecektir. Uygulamalar Helm adı verilen paket yöneticisi vasıtasıyla dağıtılacaktır. Uygulamanın clusterda çalışabilmesi için gerekli olan tüm bileşenleri bir paket haline getirir ve bu şekilde kolayca ve denetlenebilir ve GitOps mantığını destekler şekilde imajı dağıtır. Bu yaklaşım, uygulama dağıtımlarının tutarlı ve güvenilir bir şekilde yapılmasını garanti altına alacaktır.



Şekil 4.2. CD Süreci

Bu dönüşüm sonucunda, yazılım geliştirme yaşam döngüsünün tüm aşamalarında otomatizasyon sağlanacak, manuel müdahalelerden kaynaklanan hata riskleri minimize edilecek ve geliştirme süreçlerinin verimliliği artırılacaktır. Ayrıca, ekipler arası işbirliği güçlenecek ve değişikliklerin canlı ortama aktarılması daha güvenli ve hızlı hale gelecektir.

5. SONUÇ

Bu çalışma, DevOps metodolojisinin kurumsal düzeyde entegrasyonunu hem teorik hem de pratik yönleriyle ele almıştır. DevOps yaşam döngüsünün tüm aşamaları detaylı biçimde incelenmiş; özellikle CI ve CD süreçlerinin yazılım kalitesi, çeviklik ve teslim süresine etkileri vurgulanmıştır.

Kurum içinde Tekton, GitLab, Argo CD gibi açık kaynak araçlarla kurulan otomatik pipeline yapısı sayesinde manuel işlemler en aza indirilmiş, yazılım süreçleri hızlanmış ve kalite kontrolleri güvenilir hâle getirilmiştir. Tüm bileşenlerin uyumlu ve maliyetsiz olması, kamu kurumları ve sınırlı bütçeye sahip organizasyonlar için uygulanabilir bir model sunmaktadır.

Araştırma bulgularına göre DevOps'un başarısı üç temel faktöre bağlıdır:

1. Kültürel dönüşüm: Ekipler arası iş birliği ve sorumluluk paylaşımı,
2. Süreç otomasyonu: Hataların azaltılması ve verimliliğin artırılması,
3. Teknolojik altyapı: Modern, entegre araçların etkin kullanımı.

Sonuç olarak, DevOps yalnızca teknik bir model değil; kurum genelinde verimlilik, kalite ve sürdürülebilir dönüşüm sağlayan stratejik bir yaklaşımdır. Gelecekte, özellikle

KOBİ'lerde ve büyük kurumlarda DevOps etkinliğinin ve DevSecOps uygulamalarının yaygınlaştırılması üzerine çalışmalar yapılması önerilmektedir.

KAYNAKÇA

Alhamidi. (2017). Membangun Sistem Aplikasiuntuk Seleksi Calon Mahasiswa Undanganpada Tingkat Sekolah Menengah Atas. Jurnal JClick; Vol 3 No 2 (2016): J-Click. http://ejurnal.jayanusa.ac.id/index.php/JClick/article/view/26.

Bass, L., Weber, I., &Zhu, L. (2021). DevOps: A software architect's perspective on security integration. IEEE Security &Privacy, 19(1), 41-49.

Chittibala, D. R. (2023). DevSecOps: Integrating Security in to the DevOps Pipeline. International Journal of Science and Research, 12(12), 2074–2077.

Ebert, C., Gallardo, G., Hernantes, J., &Serrano, N. (2016). DevOps. IEEE Software, 33(3), 94-100.

Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations. IT RevolutionPress.

Gupta, M. L., Puppala, R., Vadapalli, V. V., Gundu, H., & Karthikeyan, C. V. S. S. (2024). Continuous Integration, Delivery and Deployment: A SystematicReview of Approaches, Tools, Challenges and Practices. In G. Paidi, S. V. Gangashetty, & A. K. Varma (Eds.), Recent Trends in AI Enabled Technologies (pp. 76–89). Springer, Cham.

Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional.

Istifarulah, M. H. R. I., & Tiaharyadini, R. (2023). DevOps, Continuous Integration and Continuous Deployment

Methods for Software Deployment Automation. JISA (Jurnal Informatika dan Sains), 6(2), 116-123.

Puppet Labs. (2021). State of DevOps Report 2021. Puppet Labs Research.

Rodriguez, P., Mikkonen, K., Kuvaja, P., Oivo, M., & Garbajosa, J. (2020). Advances in DevOps: A systematic literature review. IEEE Access, 8, 85727-85744.

Sharma, S., & Coyne, B. (2021). DevOps adoption patterns and pathways to success. International Journal of Information Management, 58, 102324.

Willis, J., & Edwards, D. (2022). Theevolution of DevOpspractices in enterpriseorganizations. Journal of Software: EvolutionandProcess, 34(3), e241