T.C.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

KURUMSAL UYGULAMALARDA DEVOPS KÜLTÜRÜNÜN BENİMSENMESİ VE ALTYAPI OTOMASYONU SÜREÇLERİNİN İYİLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

BAYRAKTAR KARAGÖZ

TEZ DANIŞMANI

DOÇ. DR. ÜMİT ÇİĞDEM TURHAL

BİLECİK, 2026

10788022

T.C.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

KURUMSAL UYGULAMALARDA DEVOPS KÜLTÜRÜNÜN BENİMSENMESİ VE ALTYAPI OTOMASYONU SÜREÇLERİNİN İYİLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

BAYRAKTAR KARAGÖZ

TEZ DANIŞMANI

DOÇ. DR. ÜMİT ÇİĞDEM TURHAL

BİLECİK, 2026

10788022

BEYAN

"**Kurumsal Uygulamalarda DevOps Kültürünün Benimsenmesi ve Altyapı Otomasyonu Süreçlerinin İyileştirilmesi**" başlıklı yüksek lisans tezinin hazırlık ve yazım aşamasında bilimsel araştırma ve etik kurallarına uyduğumu, Yükseköğretim Kurumları Bilimsel Araştırma ve Yayın Faaliyetlerinde Üretken Yapay Zekâ Kullanımına Dair Etik Rehberine uygun olarak tez/dönem projemi hazırladığımı, başkalarının eserlerinden yararlandığım bölümlerde bilimsel etik kurallarına uygun olarak atıfta bulunduğumu, kullandığım verilerde herhangi bir tahrifat yapmadığımı, çalışmamın herhangi bir kısmının başka bir tez/dönem projesi olarak sunulmadığını, aksinin tespit edilmesi durumunda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Bayraktar KARAGÖZ

.../..../2026

İmza:

ÖN SÖZ

Çalışmalarımda bana desek veren ve katkıda bulunan değerli danışman hocam Doç.Dr.Ümit Çiğdem TURHAL'a, Mühendislik fakültesi dekanı Prof.Dr.Mehmet KURBAN hocama, Dr. Öğr. Üyesi Nazım İMAL hocama, Dr.öğr.üyesi Mehmet FİDAN hocama, hiçbir zaman desteğini esirgemeyen bilişim uzmanı abim Serbülent KARAGÖZ'e , Daire Başkanı Nusret CİVELEK'e, Melike UYSAL'a, iş arkadaşım bilişim uzmanı Duygu Canpolat TAYDAŞ'a ve Dr.Ayşegül Özkaya EREN ve İbrahim Haluk AVCI'ya teşekkür ederim.

Bayraktar KARAGÖZ
2026

ÖZET

Bu tez çalışması, dijital dönüşümün hız kazandığı günümüz yazılım ekosisteminde, geleneksel yazılım geliştirme ve dağıtım yaklaşımlarının yetersizliklerini ele alarak, kurumsal ölçekte modern yazılım mimarilerine uyumlu bir DevOps altyapısının nasıl tasarlanıp uygulanabileceğini incelemektedir. Çalışma, DevOps yaklaşımını yalnızca teknik bir araçlar bütünü olarak değil, aynı zamanda organizasyonel kültür, süreç yönetimi ve otomasyonun bütünleştiği bir dönüşüm modeli olarak ele almaktadır. Araştırma kapsamında, yazılım yaşam döngüsünün otomasyonu için GitLab (kaynak kod yönetimi), Tekton (sürekli entegrasyon – CI) ve Argo CD (sürekli teslimat – CD) araçları entegre edilmiştir. GitLab üzerinde yönetilen projeler, Tekton aracılığıyla otomatik olarak derlenmiş, test edilmiş ve kalite kontrollerinden geçirilmiştir. Bu süreçte Maven ile bağımlılık yönetimi sağlanmış, Kaniko ile Docker imajları oluşturulmuş, SonarQube ile kod kalitesi analiz edilmiş ve Helm chart kullanılarak dağıtım parametreleri tanımlanmıştır. Ardından Argo CD ile uygulamalar Kubernetes ortamına otomatik olarak dağıtılmıştır. Tez kapsamında geliştirilen CI/CD mimarisi; modüler, yeniden kullanılabilir ve standartlaştırılmış bir süreç hattı modeli olarak tasarlanmıştır. Bu yaklaşım, yeni projeler için ek yapılandırma ihtiyacını ortadan kaldırarak operasyonel verimliliği artırmıştır. GitOps prensipleri doğrultusunda Argo CD'nin Git deposu ile canlı sistemler arasında sürekli senkronizasyon sağlaması sayesinde sistem durumu anlık olarak izlenebilmiş, yapılandırma sapmaları otomatik şekilde tespit edilip giderilmiştir. Böylece yazılım yaşam döngüsünde şeffaflık, izlenebilirlik ve tam otomasyon sağlanmıştır. DevOps ekosisteminin izleme ve gözlemlenebilirlik katmanı da kapsamlı biçimde ele alınmıştır. Prometheus sistem metriklerinin toplanması için, Grafana bu metriklerin görselleştirilmesi için, Loki kayıt yönetimi için; Jaeger ve Sentry ise dağıtık izleme, performans analizi ve hata takibi amacıyla yapılandırılmıştır. Bu sayede sistemin hem operasyonel izleme hem de hata ayıklama gereksinimleri bütüncül bir şekilde karşılanmıştır. Sonuç olarak bu çalışma, tamamen açık kaynaklı ve lisans maliyeti olmayan teknolojiler kullanılarak, kurumsal ölçekte maliyet etkin, sürdürülebilir ve güvenli bir DevOps altyapısının nasıl kurulabileceğini ortaya koymaktadır. Elde edilen bulgular, DevOps'un yalnızca teknik yetkinliklerle değil; süreçsel olgunluk, kültürel adaptasyon ve sürekli iyileştirme anlayışıyla birlikte ele alınması gerektiğini göstermektedir. Tez, CI/CD süreçlerinin inşasında sektörel ihtiyaçlara cevap verebilecek, yol gösterici bir uygulama modeli sunmaktadır.

Anahtar Kelimeler: Açık Kaynak, CI/CD, DevOps, GitOps, Kubernetes

ABSTRACT

This thesis examines how a DevOps infrastructure compatible with modern software architectures can be designed and implemented at the enterprise level by addressing the limitations of traditional software development and deployment approaches in today's rapidly evolving digital transformation landscape. The study considers DevOps not merely as a set of technical tools, but as a comprehensive transformation model integrating organizational culture, process management, and automation. Within the scope of the research, GitLab (source code management), Tekton (continuous integration – CI), and Argo CD (continuous delivery – CD) were integrated to automate the software development lifecycle. Software projects managed in GitLab were automatically built, tested, and subjected to quality controls through Tekton. During this process, dependency management was handled using Maven, Docker images were created with Kaniko, code quality was analyzed using SonarQube, and deployment parameters were defined through Helm charts. Subsequently, applications were automatically deployed to a Kubernetes environment using Argo CD. The CI/CD architecture developed in this thesis was designed as a modular, reusable, and standardized pipeline model, thereby eliminating the need for project-specific configurations and improving operational efficiency. In line with GitOps principles, Argo CD ensured continuous synchronization between the Git repository and live environments, enabling real-time system monitoring and automatic detection and resolution of configuration drifts. This integration provided transparency, traceability, and full automation throughout the software lifecycle.

The monitoring and observability layers of the DevOps ecosystem were also comprehensively addressed. Prometheus was configured for collecting system performance metrics, Grafana for visualizing these metrics, Loki for log management, and Jaeger and Sentry for distributed tracing, performance monitoring, and error analysis. This approach enabled a holistic fulfillment of both operational monitoring and debugging requirements. In conclusion, this study demonstrates how a cost-effective, sustainable, and secure DevOps infrastructure can be established at the enterprise level using entirely open-source and license-free technologies. The findings indicate that DevOps adoption should be approached not only through technical competencies but also through process maturity, cultural adaptation, and continuous improvement. The thesis presents a guiding application model capable of addressing sectoral needs in the construction of CI/CD processes.

Keywords: CI/CD, DevOps, GitOps, Kubernetes, Open Source

İÇİNDEKİLER

Sayfa

TABLOLAR LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

Sayfa

KISALTMALAR VE SİMGELER LİSTESİ

API: Application Programming Interface(Uygulama Programlama Arayüzü)

BT: Bilgi Teknolojileri

CD: Continuous Delivery (Sürekli Teslim)

CI: Continuous Integration (Sürekli Entegrasyon)

DAST: Dynamic Application Security Testing (Dinamik Uygulama Güvenlik Testi)

DevOps: Development and Operations (Geliştirme ve Operasyonlar)

ELK: Elasticsearch, Logstash, Kibana (Log ve izleme teknolojileri yığını)

GitLab: GitLab yazılım platformu

IaC: Infrastructure as Code (Kod Olarak Altyapı)

IDE: Integrated Development Environment (Tümleşik Geliştirme Ortamı)

IT: Information Technology (Bilgi Teknolojileri)

OWASP ZAP: OWASP Zed Attack Proxy (Web Uygulama Güvenlik Tarama Aracı)

Pipeline: Yazılım teslim sürecini ifade eden otomasyon hattı

RAD: Rapid Application Development (Hızlı Uygulama Geliştirme)

SAST: Static Application Security Testing (Statik Uygulama Güvenlik Testi)

SDLC: Software Development Life Cycle (Yazılım Geliştirme Yaşam Döngüsü)

VS Code: Visual Studio Code (Kod geliştirme editörü)

YAML:Yet Another Markup Language (Yapılandıma ve İşaretleme dili)

1.GİRİŞ

Dijital dönüşüm çağında, organizasyonların rekabet gücünü koruyabilmeleri ve sürdürülebilir büyüme sağlayabilmeleri için teknolojik adaptasyon yetenekleri kritik önem taşımaktadır. Modern iş dünyasında yazılım sistemleri, kurumsal süreçlerin merkezinde yer alırken, bu sistemlerin geliştirilmesi ve işletilmesi süreçlerinde çeviklik, güvenilirlik ve sürekli iyileştirme ihtiyacı ön plana çıkmaktadır.

Bu bağlamda DevOps, geleneksel yazılım geliştirme ve operasyon süreçlerini dönüştüren, organizasyonel kültür değişimini tetikleyen önemli bir yaklaşım olarak karşımıza çıkmaktadır. DevOps, yazılım geliştirme (Development) ve sistem operasyonları (Operations) arasındaki geleneksel bariyerleri ortadan kaldırarak, bu iki kritik fonksiyonu entegre eden bir kültür, felsefe ve uygulama metodolojisidir. Kim vd. (2016)'nin belirttiği gibi, DevOps yaklaşımı, organizasyonların yüksek hızda yazılım teslimi yapabilme ve aynı zamanda sistem güvenilirliğini koruma yeteneklerini geliştirmelerini sağlamaktadır.

Forsgren vd. (2018)'e göre, DevOps kültürünün başarılı bir şekilde benimsenmesi için organizasyonların teknoloji, süreç ve kültür boyutlarında kapsamlı bir dönüşüm geçirmesi gerekmektedir. Kurumsal organizasyonlarda DevOps dönüşümü, teknik araçların ve otomasyonun ötesinde, köklü bir kültürel değişimi gerektirmektedir. Bu kültürel dönüşüm, ekipler arası işbirliği, sürekli öğrenme, deneyimleme ve hata toleransı gibi temel değerleri içermektedir.

Bununla birlikte, Lwakatare vd. (2019) tarafından yapılan araştırmada, DevOps'un başarısının yalnızca teknik araçların değil, aynı zamanda kültürel dönüşümün benimsenme düzeyine bağlı olduğu vurgulanmıştır. Araştırmacılar, kurumsal DevOps uygulamalarını beş temel boyutta incelemiş ve olgunluk seviyelerini artırmaya yönelik öneriler geliştirmiştir.

DevOps dönüşümünde insan faktörü, eğitim ve yetkinlik geliştirme önemli rol oynamaktadır. Chen ve Kim (2023), Zhang vd. (2023), Anderson ve Lee (2021) bu sürecin kritik bileşenlerini tartışmışlardır.

Modern yazılım geliştirme süreçlerinde, DevOps uygulamaları CI/CD gibi otomasyona dayalı pratikleri, altyapının kod olarak yönetimini ve mikroservis mimarilerini içermektedir. Humble ve Farley (2010)'e göre, bu pratikler organizasyonların daha hızlı, güvenilir ve öngörülebilir yazılım teslimatı yapmalarını sağlamaktadır.

Bass vd. (2020), DevOps'un kurumsal mimarideki rolünü inceleyerek, mikroservis mimarilerinin ve container teknolojilerinin DevOps pratiklerinin uygulanmasındaki etkisini

ortaya koymuşlardır. Chen vd. (2020) ise test ve deployment otomasyonunun kurumsal DevOps başarısına etkisini detaylandırmıştır.

Jenkins, GitLab, Docker, Kubernetes gibi araçlar DevOps'un teknik altyapısını oluşturmaktadır. Fitzgerald ve Stol (2017), bu araçların manuel müdahaleleri azaltmadaki rolünü açıklamıştır. Google Cloud'un "State of DevOps 2023" raporu ise DORA metriklerinin etkin kullanımıyla dönüşüm başarısının arttığını göstermiştir.

Puppet Labs'in "State 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.

DevOps'un sunduğu değer önerisi, modern iş dünyasının temel gereksinimlerini karşılamada kritik öneme sahiptir. CI ve CD pratikleri, organizasyonların pazar değişimlerine hızlı yanıt verebilmelerini sağlarken, otomasyon ve standardizasyon uygulamaları operasyonel verimliliği artırmaktadır. Sharma ve Coyne (2021)'nin çalışması bu konuyu ampirik verilerle desteklemektedir.

Güvenlik perspektifinden bakıldığında, DevOps'un "DevSecOps" evrimiyle birlikte, güvenlik pratiklerinin yazılım geliştirme yaşam döngüsüne (SDLC) entegre edilmesi önem kazanmıştır. Rodriguez vd. (2021) ile Bass vd. (2021), güvenlik kontrollerinin otomatikleştirilmesinin sağladığı katkıları ortaya koymuştur.

Kurumsal uygulamalarda DevOps adaptasyonu, beraberinde bir dizi zorluğu da getirmektedir. Rodriguez vd. (2020), Smeds vd. (2022) ile Kim vd. (2021) gibi araştırmacılar; değişim direnci, geleneksel yapıların yarattığı engeller ve teknik borç gibi unsurlara dikkat çekmektedir.

DevOps'un kurumsal uygulamalardaki önemi, dijital transformasyon sürecinin hızlanmasıyla birlikte daha da artmıştır. Gartner'ın 2023 yılı araştırmasına göre, Fortune 500 şirketlerinin %75'i DevOps pratiklerini kurumsal stratejilerinin merkezine yerleştirmiştir.

Bu çalışmada, DevOps'un kurumsal uygulamalarda ne olduğu, gerekliliği ve yazılım süreçlerine sağladığı katkı ele alınacak; altyapısı olmayan kurumların karşılaştığı zorluklar ile eski teknolojilere bağlı değişim gereklilikleri irdelenecektir. DevOps metodolojisinde kullanılan yöntemler ve araçlar incelenerek, seçilen bir kurumsal uygulamanın bu altyapıda nasıl çalıştırıldığı, DevOps'a nasıl entegre edildiği ve altyapının nasıl otomatikleştirildiği ayrıntılı olarak gösterilecektir. Araştırma, teorik çerçeveye ek olarak pratik uygulama örnekleri ve vaka çalışmalarıyla desteklenerek, kapsamlı bir rehber oluşturmayı hedeflemektedir.

Çalışmanın en önemli ayırt edici özelliği, ele alınan tüm araç ve yöntemlerin açık kaynaklı ve ücretsiz olması; ayrıca bunların geniş ölçekli bilgi işlem altyapılarında uygulanabilirliğidir. Sonuç bölümünde, organizasyonların DevOps dönüşüm süreçlerini daha etkin yönetmelerine ve bu dönüşümden en yüksek değeri elde etmelerine yardımcı olacak somut öneriler sunulacaktır.

Dijital dönüşüm sürecinde, SDLC'nin otomasyonu, çeviklik ve sürdürülebilirlik hedefleri doğrultusunda kritik bir gereklilik haline gelmiştir. Bu doğrultuda hazırlanan tez çalışmasında, kurumsal yazılım projelerinde GitOps yaklaşımı temel alınarak bir CI/CD altyapısı tasarlanmış ve uygulamaya alınmıştır. Çalışmada GitLab üzerinden kaynak kod yönetimi sağlanmış (Gitlab, 2026), Tekton ile CI süreçleri otomatikleştirilmiş (Tekton, 2026); kodun derlenmesi, test edilmesi, kalite analizlerinin yapılması ve konteyner imajlarının oluşturulması merkezi bir süreç hattı yapısı ile standardize edilmiştir. Süreç devamında ArgoCD aracı ile Git tabanlı CD süreçleri yönetilmiş (ArgoCD, 2026), Helm chart altyapısı kullanılarak (Helm, 2026) uygulamaların Kubernetes ortamlarına (Kubernetes, 2026) otomatik ve sürüm kontrollü olarak dağıtılması sağlanmıştır. Böylece hem operasyonel süreçler sadeleştirilmiş, hem de uygulama dağıtımlarında tutarlılık ve izlenebilirlik güvence altına alınmıştır. Bu tez, DevOps kültürünün yazılım süreçlerine entegrasyonuna dair uygulanabilir ve bütünleşik bir model sunarak, kurumlar için sürdürülebilir ve güvenilir bir çözüm önerisi ortaya koymaktadır.

2. DEVOPS ve DEVOPS SÜREÇLERİ

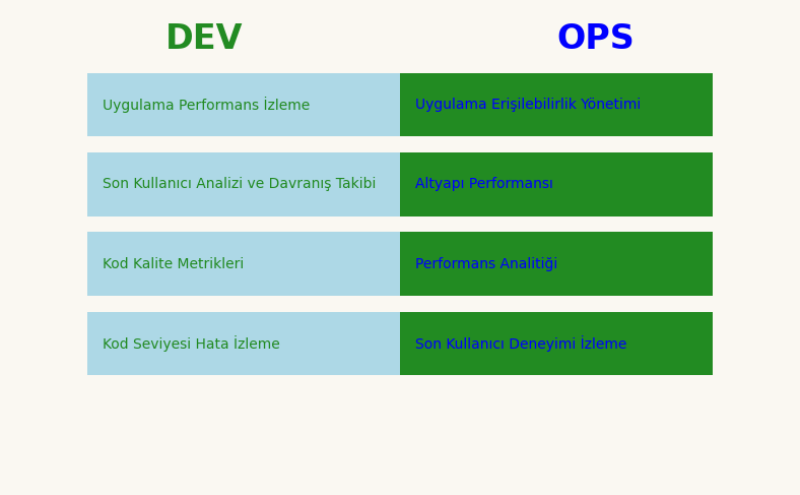
2.1. DevOps Nedir?

DevOps, kurumsal yazılım geliştirme süreçlerinde yazılım geliştirme (Development) ve sistem operasyonları (Operations) ekipleri arasında geleneksel olarak var olan ayrımı ortadan kaldırmayı amaçlayan bütüncül bir kültür, felsefe ve metodoloji olarak tanımlanmaktadır. Bu yaklaşım, yazılımın daha hızlı, güvenli ve sürdürülebilir şekilde teslim edilmesini sağlamak üzere disiplinler arası iş birliğini, süreç entegrasyonunu ve teknik otomasyonu esas almaktadır.

Geleneksel yazılım geliştirme modellerinde geliştirici ekipler kod üretiminden, operasyon ekipleri ise üretilen bu kodun canlı ortama aktarımından ve işletiminden sorumlu tutulmaktadır. Bu ayrım, çoğunlukla ekipler arası iletişim kopukluklarına, süreçsel uyumsuzluklara ve zaman kayıplarına yol açmakta; yazılım teslimat sürecinin etkinliğini azaltmaktadır. DevOps yaklaşımı ise bu iş bölümü anlayışını dönüştürerek, ekiplerin yazılım yaşam döngüsü boyunca birlikte çalışmasını ve ortak hedefler doğrultusunda hareket etmesini teşvik etmektedir.

DevOps metodolojisi, yazılım geliştirme ve operasyon süreçlerinin baştan itibaren birlikte planlanmasını; roller arası bilgi paylaşımının artırılmasını ve otomasyon teknolojilerinin yoğun biçimde kullanılmasını öngörmektedir. CI ve CD yaklaşımları bu kapsamda temel uygulamalardır. CI/CD süreçleri sayesinde geliştirilen kod, otomatik test senaryolarından geçirilerek hızlı ve güvenli biçimde üretim ortamına aktarılmakta; böylece yazılım güncellemelerinin daha kısa sürede ve daha düşük hata oranlarıyla son kullanıcıya ulaşması mümkün hale gelmektedir.

Şekil 1.1'de kavramsal olarak gösterildiği üzere, geliştirici ekipler proje gereksinimlerini planlamak, kod yazmak, derleme işlemlerini yürütmek ve kalite güvencesi sağlamak amacıyla test süreçlerini gerçekleştirmekle sorumludur. Operasyon ekipleri ise test edilmiş uygulamaların üretim ortamına entegrasyonu, sunucu ve ağ altyapısının yönetimi, sistemin kesintisiz hizmet sunmasının sağlanması ve performans/hata izleme faaliyetlerinden sorumludur. Ancak DevOps çerçevesinde bu roller arasında kesin sınırlar yerine, süreç bütünlüğünü esas alan bir iş birliği modeli benimsenmektedir.



Şekil 1.1. Dev ve Ops ekiplerinin görev dağılımı

DevOps uygulamaları, yalnızca yazılım üretim süreçlerini değil, aynı zamanda sistem altyapısının planlanması, kurulumu, otomasyonu ve izlenmesini de kapsamaktadır. Yazılım bileşenleri, otomatik test ve dağıtım süreç hatlarına entegre edilerek manuel müdahalelerin en aza indirilmesi hedeflenmektedir. Bu sayede hem insan kaynaklı hata olasılığı azaltılmakta hem de operasyonel verimlilik artırılmaktadır. Süreç boyunca kullanılan merkezi izleme araçları, dağıtım sırasında meydana gelen hataları erken evrede tespit edebilmekte; kayıt ve metrik analizleri ile sorunların kökenine hızla inilebilmektedir.

Güvenlik boyutu da DevOps metodolojisinin ayrılmaz bir parçası haline gelmiştir. SDLC'nin her aşamasına entegre edilen güvenlik katmanları sayesinde erişim kontrolleri sıkılaştırılmakta, kod kalitesi ve mevzuata uyumluluk denetimleri sistematik hale getirilmektedir. Bu bütünleşik yaklaşım, "DevSecOps" kavramının temelini oluşturmaktadır.

DevOps uygulamalarının kurumsal organizasyonlarda benimsenmemesi ya da eksik uygulanması durumunda; süreçler manuel yürütülmek zorunda kalmakta, bu da yapılandırma hatalarının, dağıtım problemlerinin ve performans aksaklıklarının artmasına neden olmaktadır. Ayrıca müşteri geri bildirimlerinin ürüne entegrasyon süresi uzamakta; bu durum kurumların pazardaki çevikliklerini ve rekabet güçlerini olumsuz etkilemektedir.

DevOps yaklaşımının kurumsal düzeyde etkin biçimde uygulanabilmesi, yazılım geliştirme ve dağıtım süreçlerinin uçtan uca otomasyonuna olanak tanıyan bütünleşik bir altyapının kurulmasını gerektirmektedir. Bu altyapı, manuel müdahaleleri en aza indirerek yazılım geliştirme yaşam döngüsünde tutarlılığı, hız ve güvenliği artırmayı hedeflemektedir. Geliştiriciler tarafından yapılan kod değişiklikleri, CI/CD süreç hatları aracılığıyla otomatik olarak test edilmekte ve güvenli biçimde üretim ortamına aktarılmaktadır. Sürecin her

aşamasında entegre edilen güvenlik kontrolleri sayesinde potansiyel açıklar erken evrede tespit edilmekte, risk seviyesi asgariye indirilmektedir.

Uygulama bileşenleri, çalışma zamanında ihtiyaç duyulan tüm dosyalar, bağımlılıklar ve konfigürasyonlar ile birlikte izole edilmiş paketler hâlinde konteyner yapıları içerisinde sunulmaktadır. Bu yaklaşım, uygulamaların farklı ortamlarda tutarlı bir şekilde çalıştırılabilmesini sağlamakta ve sistem bağımlılıklarından kaynaklanabilecek uyumsuzlukları ortadan kaldırmaktadır. Konteynerlerin yaşam döngüsü ve dağıtımı ise Kubernetes gibi modern konteyner orkestrasyon platformları tarafından yönetilmektedir. Kubernetes, küme yapısı içinde pod'lar aracılığıyla hizmetlerin hangi zamanda, hangi düğümde ve kaç örnekle çalıştırılacağını belirlemekte; kaynakları dinamik olarak ölçeklendirmekte ve sistem performansını optimize etmektedir. Olası bir konteyner arızasında yeni bir örnek otomatik olarak devreye alınmakta, hizmet sürekliliği garanti altına alınmaktadır.

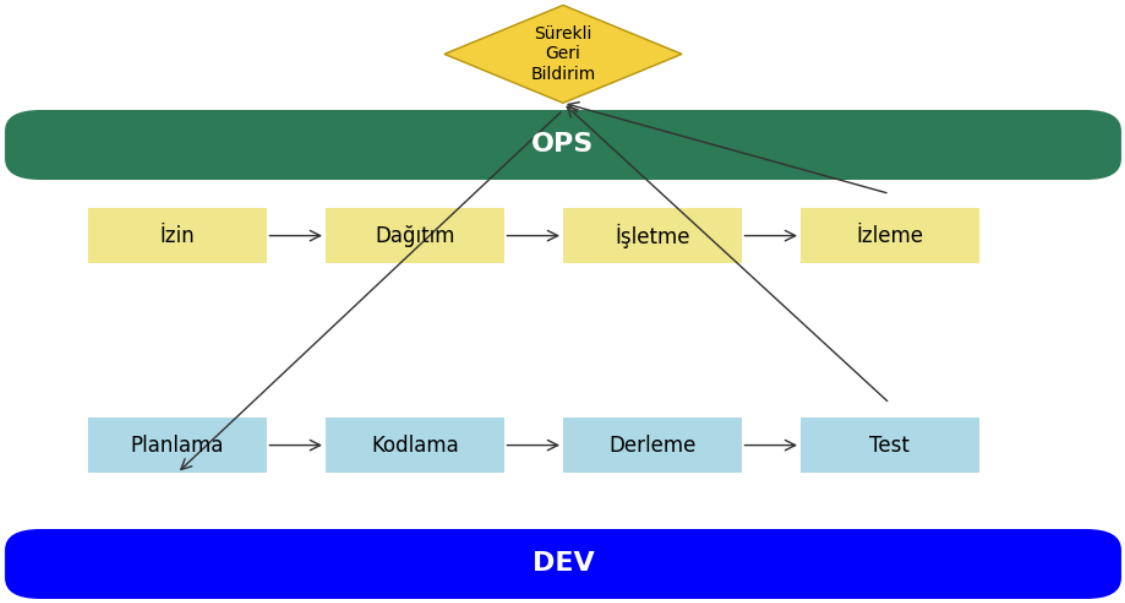
Ayrıca, sistem üzerindeki yük ve kullanıcı talepleri otomatik ölçekleme mekanizmalarıyla dengelenmekte; böylece performans istikrarı korunmaktadır. Süreç boyunca kullanılan merkezi izleme ve alarm sistemleri, anomalileri gerçek zamanlı olarak tespit ederek ilgili ekipleri uyarmakta ve hızlı müdahale olanağı sunmaktadır. Bu yapı, DevOps'un temel ilkeleri olan çeviklik, otomasyon ve güvenilirliği sistemsel düzeyde destekleyen bir teknolojik temel teşkil etmektedir.

2.2 DevOps Yaşam Döngüsü

DevOps yaşam döngüsü, yazılım projelerinin fikir ve gereksinim analizinden başlayarak, planlama, geliştirme, test etme, sürümleme, dağıtım ve izleme aşamalarını kapsayan kesintisiz ve döngüsel bir süreçtir. Bu döngü, geleneksel yazılım geliştirme modellerinden farklı olarak, tüm aşamaların birbirine entegre biçimde işlemesini ve sürekli geri bildirim mekanizmaları ile optimize edilmesini esas alır. Şekil 1.2'de kavramsallaştırıldığı üzere, DevOps yaşam döngüsü bir bütün olarak ele alınmakta ve süreçlerin birbirini tetikleyen, yinelemeli yapılarla ilerlediği bir model sunmaktadır.

Yaşam döngüsünün ilk aşamasında proje hedefleri, iş gereksinimleri ve sistem tasarımı planlanmakta; ardından yazılım geliştirme faaliyetleri gerçekleştirilerek kaynak kodlar versiyon kontrol sistemleriyle entegre edilmektedir. Bu noktadan itibaren DevOps'un temel yapıtaşlarından biri olan otomasyon devreye girmektedir. Kod değişiklikleri CI süreçleri kapsamında otomatik olarak derlenmekte, test senaryolarından geçirilmekte ve bir sonraki ortama aktarılmak üzere hazır hâle getirilmektedir. Onaylanan yazılım sürümleri, CD

mekanizmaları aracılığıyla tekrarlanabilir, güvenli ve kontrol edilebilir biçimde üretim ortamına aktarılmaktadır.



Şekil 1.2. DevOps yaşam döngüsü

Uygulamanın canlı ortama geçişinin ardından, sistemin çalışabilirliği, performansı, erişilebilirliği ve güvenlik düzeyi çeşitli metriklerle sürekli olarak izlenmektedir. Bu izleme faaliyetleri yalnızca anlık sistem sağlığını değerlendirmekle kalmayıp, aynı zamanda yaşam döngüsünün başlangıç aşamasına veri sağlayan önemli bir geri bildirim kaynağı işlevi görmektedir. Böylece planlama ve tasarım kararları, gerçek zamanlı kullanıcı verileri ve sistem içgörüleri temelinde yeniden değerlendirilerek, çevik bir geliştirme süreci desteklenmektedir.

Bu bütüncül ve döngüsel yapı, ekipler arası iş birliğini güçlendirmekte; hataların erken aşamada tespit edilmesini ve iyileştirme adımlarının daha etkin biçimde uygulanmasını mümkün kılmaktadır. Neticede, yazılım ürünleri yalnızca daha hızlı değil, aynı zamanda daha güvenilir, sürdürülebilir ve kullanıcı odaklı biçimde sunulabilmektedir.

2.3 Sürekli Entegrasyon (CI) ve Sürekli Teslimat (CD) Kavramları

CI ve CD, DevOps yaşam döngüsünün teknik temelini oluşturan iki kritik DevOps mühendisliği pratiğidir. Bu yaklaşımlar, yazılım geliştirme sürecinde otomasyonu artırarak teslimat sürelerini kısaltmak, kaliteyi artırmak ve hata oranlarını minimize etmek amacıyla sistematik olarak uygulanmaktadır.

CI, yazılım geliştiricilerin gerçekleştirdiği kod değişikliklerini sık aralıklarla merkezi bir versiyon kontrol sistemine (örneğin Git) göndermesi ve bu değişikliklerin otomatik olarak derlenip birim testlerine tabi tutulması sürecini ifade etmektedir. Her entegrasyon işlemi, sistemin bütünlüğünü koruyacak şekilde otomatik olarak doğrulanır. Bu sayede kod tabanındaki olası hatalar ve entegrasyon çakışmaları erken evrede tespit edilmekte; geliştirici ekipler arasında uyumsuzlukların ortaya çıkması önlenmektedir. Erken hata tespiti, geriye dönük düzeltme maliyetlerini azaltarak yazılım geliştirme sürecinin çevikliğini artırmaktadır.

CD ise CI süreçlerinden başarıyla geçen kod bileşenlerinin, üretim ortamına aktarılmaya hazır hâle getirilmesini kapsayan bir süreçtir. Bu süreçte, uygulama bileşenleri üretim benzeri ortamlarda entegrasyon testlerine, güvenlik taramalarına ve kullanıcı kabul kriterlerine tabi tutulur. Bu kontrollerin başarıyla tamamlanmasının ardından yeni yazılım sürümleri, manuel müdahaleye gerek kalmaksızın güvenli, tekrarlanabilir ve izlenebilir şekilde canlı ortama geçebilecek düzeye getirilir.

CI ve CD süreçlerinin bütünleşik bir yapı içerisinde uygulanması, yazılım teslimatında otomasyonun etkinliğini maksimize etmektedir. Bu yapı sayesinde geliştiriciler küçük ve bağımsız kod değişikliklerini kısa sürelerde sisteme entegre edebilmekte; bu değişiklikler otomasyon süreç hatları (CI/CD pipeline hattı) aracılığıyla sırasıyla derlenmekte, test edilmekte, paketlenmekte ve dağıtıma hazırlanmaktadır. Sürecin tüm aşamaları izlenebilir olduğu için, ortaya çıkan sorunlara ilişkin geri bildirimler anlık olarak sistemin erken safhalarına iletilebilmekte ve düzeltici eylemler hızla alınabilmektedir.

Sonuç olarak, CI/CD pratikleri yazılım geliştirme süreçlerinin daha çevik, sürdürülebilir ve güvenilir olmasını sağlamaktadır. Organizasyonlar bu sayede yalnızca yazılım teslim sürelerini kısaltmakla kalmamakta, aynı zamanda inovasyon kapasitesini artırmakta, sistem güvenliğini güçlendirmekte ve müşteri memnuniyetini önemli ölçüde yükseltmektedir.

2.4 Altyapı Otomasyonu Kavramı

Altyapı otomasyonu (Infrastructure as Code – IaC), DevOps metodolojisinin temel yapıtaşlarından biri olup, bilgi işlem kaynaklarının (sunucular, ağlar, depolama birimleri, hizmet yapılandırmaları ve dağıtım süreçleri) manuel müdahale gerektirmeden, kod aracılığıyla tanımlanarak yönetilmesini mümkün kılmaktadır. Bu yaklaşım, kurulum ve yapılandırma işlemlerinde insan hatasını ortadan kaldırmak, sistem tutarlılığını artırmak ve altyapıyı daha sürdürülebilir bir biçimde yönetmek amacıyla uygulanmaktadır.

Kod tabanlı altyapı yönetimi, genellikle Terraform, Ansible, Puppet ve Chef gibi araçlarla oluşturulan şablonlar veya betikler aracılığıyla gerçekleştirilir. Bu araçlarla tanımlanan altyapı bileşenleri, versiyon kontrol sistemlerinde izlenebilir hâle getirilmekte; böylece yapılan her değişiklik kayıt altına alınmakta ve gerektiğinde geriye dönük olarak incelenebilmektedir. Otomasyon sayesinde altyapının kurulumu, güncellenmesi ve yeniden yapılandırılması hızlı, tekrarlanabilir ve ölçeklenebilir biçimde yürütülebilmektedir.

Altyapı otomasyonu, CI/CD süreçlerine entegre edildiğinde, yazılım geliştirme ekipleri altyapı değişikliklerini kod güncellemeleriyle aynı akış içinde yönetebilmektedir. Bu bütünleşik yapı sayesinde yazılım kodundaki her değişiklik, tetikleyici koşullarla ilişkilendirilerek yeni bir test ortamının otomatik olarak oluşturulmasına; testlerin tamamlanmasının ardından başarılı sürümün üretim ortamına aktarılmasına olanak tanımaktadır. Tüm bu süreç, herhangi bir manuel müdahaleye gerek kalmaksızın otomatik biçimde tamamlanmaktadır.

Bu yaklaşım, yalnızca teslimat sürecinin hızlandırılmasına değil, aynı zamanda hizmet sürekliliğinin artırılmasına, altyapının daha esnek ve dinamik hâle getirilmesine ve operasyonel maliyetlerin azaltılmasına katkı sağlamaktadır. Ayrıca, kaynakların talep doğrultusunda dinamik biçimde ölçeklenebilmesi ve dağıtım sırasında sistemin güvenilirliğinin korunması, kurumsal bilgi sistemlerinin dayanıklılığını önemli ölçüde artırmaktadır.

2.5 DevOps Kültürü ve Sürekli Geri Bildirim Kavramları

Sürekli DevOps metodolojisi yalnızca teknik araç ve süreçlerden ibaret olmayıp, aynı zamanda organizasyonel kültürün dönüşümünü de içeren kapsamlı bir paradigmadır. CI, CD ve IaC gibi teknik uygulamalar, DevOps kültürünün başarılı bir şekilde hayata geçirilmesinde temel birer yapı taşı işlevi görmektedir. Ancak bu teknik bileşenlerin etkinliği, onları çevreleyen kültürel yapının nitelikleriyle doğrudan ilişkilidir.

DevOps kültürünün temelinde, ekipler arası şeffaflık, iş birliği, ortak sorumluluk ve sürekli iyileştirme ilkeleri yer almaktadır. Otomasyon pipeline hatları aracılığıyla yürütülen CI/CD süreçleri ve kod temelli altyapı yönetimi, yazılım geliştirme ve operasyon ekipleri arasında bilgi akışını hızlandırmakta, süreçlerin şeffaf ve izlenebilir hâle gelmesini sağlamaktadır. Bu şeffaf yapı, ekip üyelerinin birbirlerinin çalışmalarını anlık olarak takip edebilmesine ve birlikte karar alma süreçlerine etkin katılım sağlamasına imkân tanımaktadır.

Sürekli geri bildirim mekanizmaları ise bu iş birliğinin dinamik yapısını destekleyen en önemli unsurlardan biridir. Sistem izleme araçları, kayıtlama çözümleri, test sonuçları ve kullanıcı geri bildirimleri aracılığıyla elde edilen veriler, yazılım yaşam döngüsünün her

aşamasında kullanılmak üzere merkezi biçimde paylaşılmakta ve değerlendirilmekte; böylece her sürümde daha güvenilir, daha işlevsel ve kullanıcı odaklı ürünler geliştirilebilmektedir.

DevOps kültürünün önemli ilkelerinden biri olan "blameless postmortem" yaklaşımı da bu süreçte belirleyici rol oynamaktadır. Bu yaklaşım, sistemsel aksaklıkların veya başarısızlıkların bireyler yerine süreçler üzerinden değerlendirilmesini teşvik eder. Hatalar birer suçlama aracı değil, öğrenme ve iyileştirme fırsatı olarak ele alınır. Bu kültürel duruş, organizasyon içerisinde psikolojik güvenliği artırmakta ve inovatif düşünceyi desteklemektedir.

Sonuç olarak, DevOps'un teknik bileşenleri olan CI/CD ve IaC, güçlü bir kurumsal kültür ve etkin geri bildirim sistemleriyle bütünleştirildiğinde; yalnızca yazılım teslimat sürecinin değil, aynı zamanda organizasyonel öğrenmenin ve çevikliğin de sürdürülebilir şekilde artmasına olanak tanımaktadır. Bu bütüncül yapı, DevOps'u yalnızca bir teknik yaklaşım değil, aynı zamanda bir dönüşüm modeli olarak konumlandırmaktadır.

2.6 DevSecOps: Güvenliğin Yazılım Yaşam Döngüsüne Entegrasyonu

DevOps'un yazılım geliştirme ve operasyon süreçlerini bütünleştiren yapısı, hız ve çevikliği ön plana çıkarırken; güvenliğin bu sürecin ayrılmaz bir parçası olarak ele alınması ihtiyacını da beraberinde getirmiştir. Bu gereksinim doğrultusunda ortaya çıkan DevSecOps (Development–Security–Operations) yaklaşımı, güvenlik uygulamalarının yazılım yaşam döngüsünün tüm aşamalarına sistematik olarak entegre edilmesini hedeflemektedir. Böylece güvenlik, yalnızca sonradan eklenen bir kontrol mekanizması değil, sürecin başlangıcından itibaren yerleşik bir bileşen hâline gelir.

DevSecOps'un temel prensibi, güvenlik süreçlerini geliştirmenin ve dağıtımın hızını kesmeden entegre etmektir. Bu bağlamda, "Shift-Left Security" (güvenliğin sola kaydırılması) ilkesi devreye girmekte; güvenlik kontrolleri planlama, geliştirme ve test aşamalarında devreye alınarak, potansiyel açıkların mümkün olan en erken evrede tespit edilmesi sağlanmaktadır. Böylece güvenlik problemleri, canlıya geçiş sonrasında değil, daha kod yazımı sırasında ele alınmakta; hem maliyetli düzeltme işlemleri azaltılmakta hem de sistem bütünlüğü korunmaktadır.

DevSecOps uygulamalarında yaygın olarak kullanılan teknik çözümler şunlardır:

- **Statik Kod Analizi (SAST):** Kaynak kod üzerinde gerçekleştirilen taramalarla potansiyel güvenlik açıklarının erken evrede tespit edilmesi sağlanır.

- **Dinamik Uygulama Güvenlik Testi (DAST):** Uygulama çalışır hâlde iken dışardan gelen saldırı senaryoları üzerinden güvenlik açıkları analiz edilir.
- **Açık Kaynak Güvenlik Taramaları:** Bağımlılık yönetimi araçları (ör. OWASP Dependency-Check, Snyk) ile kullanılan kütüphane ve paketlerdeki bilinen zafiyetler kontrol edilir.
- **Gizli bilgi sızıntısı tespiti (secret scanning):** Versiyon kontrol sistemlerinde (ör. Git) yer alabilecek API anahtarları, parolalar ve erişim belirteçlerinin taranarak ifşa riski önlenir.

Bu araçlar genellikle CI/CD pipeline hatlarına entegre edilerek, yazılım her derlendiğinde otomatik olarak taranır ve olası güvenlik ihlallerine karşı anlık uyarılar üretir. Böylece geliştirici ekipler, güvenlik geri bildirimlerine anında erişebilir ve düzeltici eylemleri kod düzeyinde gerçekleştirebilir. DevSecOps süreçlerinde her güvenlik kuralı veya politika, versiyonlanabilir ve kodla ifade edilebilir; bu da denetlenebilirlik ve izlenebilirlik açısından önemli bir avantaj sunar.

DevSecOps'un başarısı yalnızca araçlarla değil, aynı zamanda organizasyonel kültürle de doğrudan ilişkilidir. Güvenliğin tüm ekiplerin ortak sorumluluğu olarak görülmesi, geliştirici, operasyon ve güvenlik birimleri arasında etkin iş birliğini gerektirir. Geleneksel güvenlik ekiplerinin dışlayıcı veya sonradan müdahale eden yapısı yerine, güvenlik uzmanlarının CI/CD süreçlerine dâhil edilmesi ve geliştiricilerle birlikte çalışması esastır.

Uygulamalar, genellikle konteyner tabanlı olarak uygulama kodu, uygulama hizmetleri kodu, altyapı olarak kod (infrastructure as code), politika olarak kod (policy as code) ve gözlemlenebilirlik olarak kod (observability as code) olmak üzere beş kod türüne ayrılabilir. DevSecOps, sürekli entegrasyon, sürekli teslimat ve sürekli dağıtım (CI/CD) boru hatları gibi ilkelere dayanarak bu uygulamalar için uygun bir geliştirme, dağıtım ve çalışma zamanı paradigması sağlar. Bu sayede yüksek güvenlik güvencesi sağlanırken "continuous authority to operate" (C-ATO) yetkinliği mümkün hale gelir." (Chandramouli, 2022).

Sonuç olarak, DevSecOps yaklaşımı, SDLC'nin hız, esneklik ve güvenlik boyutlarını dengeli biçimde bütünleştirmektedir. Geliştirme sürecinin doğal bir parçası hâline gelen güvenlik uygulamaları, yalnızca sistemsel açıkların azaltılmasını değil; aynı zamanda kurumların yasal uyumluluk gereksinimlerini karşılamasını, müşteri güvenini pekiştirmesini ve sürdürülebilir yazılım kalitesini sağlamasını mümkün kılmaktadır.

2.7. DevOps Olgunluk Modelleri ve Başarı Metrikleri

DevOps uygulamalarının kurumsal düzeyde benimsenmesi, organizasyonların teknik, kültürel ve süreçsel dönüşüm düzeyine bağlı olarak farklılık göstermektedir. Bu bağlamda, DevOps'un organizasyonel yapıya ne ölçüde entegre edildiğini değerlendirmek amacıyla olgunluk modelleri geliştirilmiştir. Bu modeller, kurumların mevcut durumlarını analiz etmesine, zayıf yönlerini belirlemesine ve stratejik gelişim planları oluşturmasına olanak tanır.

2.7.1. DevOps Olgunluk Düzeyleri

DevOps olgunluk modelleri genel olarak beş aşamalı bir çerçeve üzerinden kurgulanır:

1. **Başlangıç:** DevOps kavramı henüz kurumsal düzeyde tanınmamıştır. Geliştirme ve operasyon ekipleri birbirinden kopuktur; manuel süreçler ağırlıktadır.
2. **Gelişen:** Bazı otomasyon araçları kullanılmaya başlanmıştır; CI/CD deneysel olarak uygulanmaktadır. Kültürel dönüşüm sınırlıdır.
3. **Tanımlı:** DevOps süreçleri dokümante edilmiştir; otomasyon süreç hatları kurumsal uygulamalara entegre edilmiştir. Ekipler arası iş birliği artmıştır.
4. **Yönetilen:** CI/CD, altyapı otomasyonu ve izleme araçları sistematik olarak kullanılmaktadır. Sürekli geri bildirim süreçleri yerleşmiştir.
5. **Optimum:** Tüm süreçler veriye dayalı biçimde iyileştirilmekte; güvenlik, kalite ve performans metrikleri sürekli izlenmektedir. Organizasyon çevik ve kendi kendini iyileştiren bir yapıya ulaşmıştır.

Her bir düzey, teknik araçların kullanımı kadar, organizasyonel kültür, liderlik desteği ve ekipler arası etkileşim açısından da farklı gelişim göstergeleri sunar.

2.7.2. Başarı Metrikleri (DORA Göstergeleri)

DevOps uygulamalarının başarısını nicel olarak değerlendirebilmek için en yaygın kullanılan metrik çerçevesi DORA (DevOps Research and Assessment) göstergeleridir. Google tarafından desteklenen DORA araştırmaları, yüksek performans gösteren ekiplerin ortak operasyonel özelliklerini tanımlamış ve dört temel gösterge önermiştir:

- **Dağıtım Sıklığı:** Yazılımın canlı ortama ne sıklıkla gönderildiği.
- **Değişiklik Başına Teslim Süresi:** Bir kod değişikliğinin üretime ulaşma süresi.
- **Değişiklik Başarısızlık Oranı:** Üretime alınan değişikliklerin ne sıklıkla hata ürettiği.

- **Ortalama Kurtarma Süresi:** Bir sistem arızasından sonra hizmetin normale dönme süresi.

Bu göstergeler, teknik performansı ölçmenin yanı sıra ekip çevikliği, sistem kararlılığı ve süreç optimizasyonu hakkında da önemli içgörüler sağlar. DORA metriklerini düzenli izleyen organizasyonların, DevOps dönüşümünde daha hızlı ilerlediği ve müşteri memnuniyetinde anlamlı artışlar elde ettiği araştırmalarla ortaya konmuştur.

2.7.3. Metrik Odaklı Sürekli İyileştirme

DevOps olgunluğu, yalnızca belirli bir seviyeye ulaşmakla değil; aynı zamanda bu seviyenin veriye dayalı iyileştirme döngüleriyle sürdürülebilir kılınmasıyla ilgilidir. Bu nedenle, organizasyonların kendi metriklerini periyodik olarak analiz etmesi, darboğazları tanımlaması ve bu doğrultuda kültürel, süreçsel ve teknik aksiyonlar alması gerekmektedir. Olgunluk değerlendirme anketleri, süreç haritalamaları ve performans tabloları bu izleme faaliyetlerinde sıkça başvurulan yöntemlerdir.

2.8. DevOps'un Kurumsal Dönüşümdeki Rolü

Dijital dönüşüm çağında, yalnızca teknik yeterlilikler değil; aynı zamanda organizasyonların yapısal esnekliği, çevik karar alma süreçleri ve sürdürülebilir inovasyon yetenekleri de rekabet avantajı açısından belirleyici hâle gelmiştir. Bu bağlamda DevOps, yalnızca yazılım geliştirme süreçlerini optimize eden bir yöntem değil, aynı zamanda kurumların bütüncül dijital dönüşüm stratejilerinin merkezinde yer alan bir kültürel ve operasyonel dönüşüm aracıdır.

DevOps'un kurumsal düzeyde benimsenmesi, geleneksel hiyerarşik yapılarla çevik yönetim modelleri arasında bir köprü kurarak, organizasyon içi silo yapılarını ortadan kaldırmayı ve ekipler arası iş birliğini artırmayı hedeflemektedir. Yazılım geliştiricileri, operasyon ekipleri, güvenlik uzmanları ve test mühendisleri gibi farklı disiplinlerden gelen aktörlerin ortak bir yaşam döngüsü çerçevesinde senkronize çalışması, hem ürün kalitesini hem de teslim sürelerini doğrudan etkilemektedir.

Kurumsal dönüşüm bağlamında DevOps'un sunduğu katkılar aşağıdaki temel başlıklarda toplanabilir:

- **Yalınlaştırılmış Süreçler ve Operasyonel Verimlilik:** Otomasyon, CI/CD ve altyapı kodlaması sayesinde tekrar eden manuel işlemler ortadan kaldırılarak, süreçlerin standardizasyonu ve hızlandırılması sağlanır.

- **İnovasyonun Hızlandırılması:** Kod teslimat sürelerinin kısalması ve geri bildirim döngülerinin etkinleşmesi, organizasyonların yeni fikirleri daha hızlı test etmesine ve pazara sunmasına olanak tanır.
- **Kültürel Dönüşüm ve Psikolojik Güven:** DevOps'un şeffaf, sorumluluk paylaşımına dayalı ve "blameless" yaklaşımı, çalışan bağlılığını artırır ve öğrenen organizasyon yapısının gelişmesine katkı sunar.
- **Sistemsel Dayanıklılık:** DevOps, izleme ve hızlı müdahale kapasitesiyle operasyonel kırılganlığı azaltır. Arıza sonrası toparlanma sürelerinin düşmesi, hizmet sürekliliğini doğrudan olumlu etkiler.

Bu çerçevede DevOps, yalnızca yazılım ekiplerinin değil; aynı zamanda strateji, insan kaynakları, iş geliştirme ve üst yönetim birimlerinin de ortak sorumluluğunu gerektiren bir dönüşüm inisiyatifidir. Başarılı bir DevOps adaptasyonu, liderlik desteği, uygun kaynak tahsisi, eğitim programları ve kurumsal iç iletişim stratejilerinin birlikte yönetilmesini zorunlu kılar.

DevOps, uçtan uca bir yaklaşım sunarak iş, BT ve operasyon ekipleri arasındaki siloları yıkar. Bir kültürel paradigma olarak, süreklilik ve otomasyon getiren sürekli entegrasyon, sürekli teslimat ve sürekli dağıtım uygulamaları ile istikrarlı değişim sağlar." (Kiran, 2022)

Araştırmalar, DevOps uygulamalarını kurumsal düzeyde başarılı bir biçimde entegre eden organizasyonların; pazara çıkış sürelerinde %50'ye varan azalma, müşteri memnuniyetinde belirgin artış ve çalışan devir oranlarında düşüş gibi çok boyutlu kazanımlar elde ettiğini ortaya koymaktadır (Willis ve Edwards, 2022; Gartner, 2023).

Sonuç olarak, DevOps yalnızca teknik bir paradigma değil; kurumsal öğrenme, süreç yeniliği ve dijital olgunluk eksenlerinde stratejik bir dönüşüm motorudur. Organizasyonlar, DevOps'u sadece bir yazılım geliştirme metodolojisi olarak değil, aynı zamanda kültürel ve yönetsel bir değişim felsefesi olarak benimsemelidir.

3. KURUMSAL YAPILARDA DEVOPS UYUMLULUK ANALİZİ: EKSİKLİKLERİN TESPİTİ VE ALTYAPI DÖNÜŞÜMÜNE YÖNELİK ÖNERİLER

Kurumsal organizasyonlarda DevOps kültürüne geçiş süreci, yalnızca teknolojik bir dönüşüm değil; aynı zamanda mevcut yapıların sistematik bir biçimde değerlendirilmesini gerektiren stratejik bir dönüşüm sürecidir. Bu kapsamda, yazılımsal sistemlerin durumu analiz edilmeli, operasyonel darboğazlar ve otomasyon eksiklikleri tespit edilmelidir. DevOps uyumluluğunu engelleyen yapısal eksikliklerin belirlenmesi, etkili bir dönüşüm planlamasının ön koşuludur. Tablo 3.1'de bu eksikliklere ve kaynaklarına ilişkin sistemli bir değerlendirme sunulmaktadır.

Uygulamada DevOps benimsemesinin önündeki teknolojik, kültürel ve organizasyonel engeller arasında farkındalık eksikliği, değişime direnç, eski sistemlerle entegrasyon sorunları, sürekli test eksikliği ve beceri boşlukları yer alır. Başarı için çapraz fonksiyonel iş birliği, otomasyon, eğitim ve lider desteği kritik önemdedir (Haddad, 2024).

Modern DevOps altyapılarında konteyner tabanlı mimariler, uygulamaların taşınabilirliğini ve tutarlılığını garanti altına almak açısından kritik rol oynamaktadır. Konteynerlerin küme bazlı yönetimini sağlayan Kubernetes gibi orkestrasyon sistemleri, yük dengeleme, yüksek erişilebilirlik ve hata toleransı sunarak sistem sürekliliğini güçlendirmektedir.

CI ve CD süreçleri, DevOps uygulamalarının merkezinde yer almaktadır. Kurumlar içerisinde canlı sistemde CI/CD süreçlerinin aktif olarak çalıştırılmaması, yazılım teslimat sürecinin manuel müdahalelere ve hata riskine açık olduğunu göstermektedir. Bu eksiklik, geliştirme çevikliğini ve operasyonel güvenilirliği ciddi ölçüde sınırlamaktadır.

Kod kalitesi ve güvenlik taramaları, yazılım yaşam döngüsünün ilk aşamalarında yapılmalı ve CI/CD süreç hatlarına entegre edilmelidir. Kurumlarda genelde yalnızca Fortify kullanımı ile sınırlı kalan bu süreç, genişletilerek SonarQube gibi araçlarla desteklenmeli (SonarQube, 2026); statik analizlerin yanı sıra güvenlik testleri de dahil edilmelidir. Ayrıca, otomatik testlerin yetersizliği nedeniyle yazılım güvenilirliği zayıflamaktadır. Test süreçlerinin genişletilmesi ve geliştirici eğitimleriyle desteklenmesi gereklidir.

Uygulama ve sistem kayıtlarının merkezi olarak toplanmaması, hata ayıklama ve performans yönetimi süreçlerini güçleştirmektedir. Modern izleme çözümleri Prometheus

(Prometheus, 2026), Grafana (Grafana, 2026), ELK Stack vb. devreye alınarak sistemin bütünsel olarak gözlemlenmesi ve anlık müdahale kabiliyetinin artırılması sağlanmalıdır.

Tablo 3.1. DevOpsa geçiş sürecinde tespit edilen sorun ve kaynağı

Tespit Edilen Sorun	Sorunun Kaynağı
Java Versiyonunun Güncel Olmaması	Java 8 desteği yakında sonlanacak
Kayıtlama 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 standardın olmaması benzer işlere her projede efor harcanmasına sebep olmaktadır
REST API'lar İçin Standart Eksikliği	REST API'lar için 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

Konteyner mimarisinde veri yönetimi özel çözümler gerektirmektedir. Kurumlarda veya şirketlerde kalıcı ve merkezi bir depolama altyapısının olmaması, veri kaybı ve performans düşüklüğü risklerini artırmaktadır. Verilerin güvenli saklanmasını ve hızlı erişilebilirliğini sağlayacak merkezi bir depolama altyapısı kritik önemdedir.

REST API'ler ve içerik yönetimi gibi alanlarda kurumlar genelinde yazılım geliştirme standartlarının eksikliği tespit edilmiştir. Bu durum, projeler arasında entegrasyon zorluklarına ve kaynak israfına neden olmaktadır. Standart tanımlamalar kurumsal DevOps kültürünün yerleşmesi açısından zorunludur.

Tablo 3.2'de listelenen modern DevOps araçları, kurumların ve şirketlerin yazılım geliştirme, dağıtım, gözlemleme ve güvenlik süreçlerini otomatikleştirmek için önerilmektedir. Tekton, GitLab, Kubernetes, Helm, Terraform, Grafana, Prometheus ve OWASP ZAP gibi

araçların doğru yapılandırma ile entegre edilmesi, kurumsal dönüşümde büyük fayda sağlayacaktır.

Yukarıda detaylandırılan analiz bulguları, mevcut durumunun DevOps ilkeleriyle uyumlu hâle getirilmesi için hem teknik hem de kültürel alanda kapsamlı bir dönüşüm ihtiyacı olduğunu ortaya koymaktadır. Bu bağlamda yapılacak yatırımlar; yazılım teslim sürelerinde azalma, operasyonel hata oranlarında düşüş, ekipler arası iş birliğinde artış ve sürdürülebilir dijital altyapının inşası gibi somut kazanımlar sağlayacaktır.

DevOps süreçlerine güvenliğin entegre edilmesi anlamına gelen DevSecOps, güvenlik kontrollerinin yalnızca son aşamalarda değil; kodun yazıldığı ilk andan itibaren tüm yaşam döngüsüne yayılmasını sağlar. Geleneksel sistemlerde güvenlik testleri, geliştirme tamamlandıktan sonra manuel olarak uygulanırken; DevSecOps yaklaşımında güvenlik açıkları, otomatik statik analizler, açık kaynak bileşen taramaları ve konteyner güvenlik denetimleriyle erken aşamada tespit edilir.

Fortify gibi araçların sınırlı düzeyde kullanılması, güvenlik sürecinin bütünsel DevOps yaşam döngüsüne entegre edilmediğini göstermektedir. SonarQube, Snyk, OWASP ZAP, Trivy,Clair gibi açık kaynaklı veya ticari araçlarla yapılacak entegrasyonlar, hem kaynak kodu güvenliğini hem de dağıtımda kullanılan konteynerlerin güvenliğini sağlamaya katkı sunacaktır. Ayrıca, otomatik güvenlik testlerinin CI/CD süreç hatlarına dahil edilmesi, sistemin kesintisiz güvenliğini mümkün kılar.

Bu nedenle kurumsal dönüşüm stratejisinin bir parçası olarak DevSecOps kültürünün benimsenmesi; hem yasal uyumluluk açısından hem de müşteri verilerinin korunması bağlamında büyük önem arz etmektedir.

Tablo 3.2. Kullanılan bazı DevOps araçları

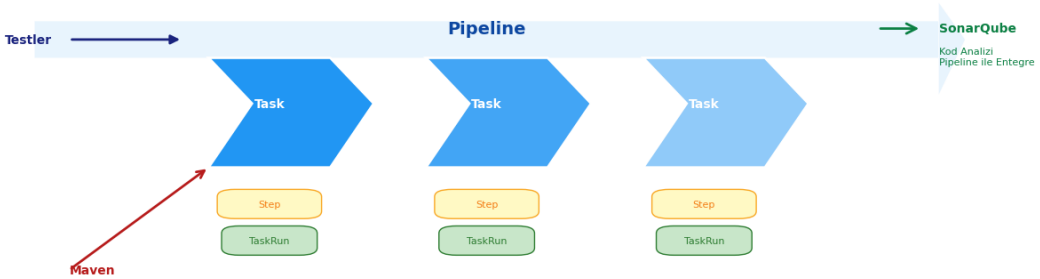
Araç	Kategori	Kısa Açıklama
GitLab	Kod Reposu	GitLab içinde yazılımcıların kodları ve Helm chartlar tutulur.
Tekton	CI(SürekliEntegrasyon)	Açık kaynaklı, Kubernetes üzerinde çalışan, esnek ve modüler yapısıyla otomatik derleme, test ve dağıtım olanağı sağlar. Pipelinelar buradan tetiklenir ve maven sonarqube gibi adımlar pipelinenın task adımı olarak tanımlanır
Maven	Build & Dependency Management	Java projelerinde bağımlılık yönetimi ve proje derleme sürecini otomatikleştiren, XML tabanlı yapılandırma kullanan araç.
SonarQube	Kod Kalitesi & Güvenlik	Statik analizle kod kalitesi ve güvenlik zafiyetlerini tespit eder; CI pipeline hattı entegrasyonu sunar.
Kaniko	Konteynerizasyon	Docker olmadan, Kubernetes veya bulut ortamlarında güvenli imaj oluşturmayı sağlayan araç.
Docker	Konteyner Platformu	Uygulamaları bağımlılıklarıyla birlikte konteyner olarak paketleyerek her ortamda tutarlı çalışmasını sağlar.
Quay	Konteyner Kayıt Deposu (Registry)	Güvenli, ölçeklenebilir konteyner imaj depolama ve dağıtım hizmeti sunar.
Clair	Güvenlik Tarama (Container)	Konteyner imajlarında bilinen güvenlik açıklarını tarayan açık kaynaklı araç.
Argo CD	GitOps Dağıtım	Kubernetes üzerinde GitOps yaklaşımıyla sürekli dağıtım yapmayı sağlayan deklaratif araç.
Kubernetes	Konteyner Orkestrasyonu	Pod, Deployment, Service tanımları üzerinden ölçeklenebilir, self-healing ve otomatik yeniden başlatma sunar.
Helm	Kubernetes Paket Yönetimi	Kubernetes üzerinde uygulama paketlerini (Charts) kolayca kurup yönetmeye ve rollback yapmaya imkân tanır.
Prometheus	İzleme & Uyarı	Zaman serisi veritabanı ve alerting motoru; Kubernetes ile sıkı entegrasyon sunar.
Loki	Kayıt Yönetimi (Log Aggregation)	Hafif, ölçeklenebilir log toplama sistemi; Grafana ile entegre çalışır.
Jaeger	Dağıtık İzleme (Tracing)	Mikroservis mimarisinde isteklerin uçtan uca izlenmesini sağlayan açık kaynaklı dağıtık izleme aracı.
Grafana	Görselleştirme & Dashboard	Metrik, log ve izleme verilerini farklı kaynaklardan çekip özelleştirilebilir panolar oluşturur.
Ceph	Dağıtık Depolama Sistemi	Nesne, blok ve dosya depolamayı destekleyen, yüksek erişilebilirlik ve ölçeklenebilirlik sunan açık kaynaklı depolama çözümü.

4. KURUMSAL UYGULAMALARDA DEVOPS GEÇİŞ SÜRECİ: MEVCUT DURUM ANALİZİ VE DÖNÜŞÜM YOL HARİTASI

DevOps entegrasyonunun ilk adımı, kod ve görev yönetiminin kurumsal düzeyde organize edilmesidir. Bu amaçla GitLab üzerinde her bir mikroservis için ayrı birer proje oluşturulmuş; bu projelere ait kodlar, issue board üzerinden tanımlanan görevlerle ilişkilendirilmiştir. Geliştiriciler, kendilerine atanan görevleri gerçekleştirdikçe bu görevleri planlama tahtasında ilerleterek görev takibini görsel olarak sürdürmüştür. Şekil 4.1'de bu planlama sürecinin örnek bir görünümü sunulmaktadır.

Şekil 4.1 Kurumsal DevOps metodolojisinde adımlar

Kod depolarına her bir commit işlemi gerçekleştirildiğinde, GitLab da webhook tarafından tanımlanan CI/CD süreç hattı otomatik olarak tetiklenmektedir. Bu süreç hattı, derleme, test, sonarqube scan, dockerize, push, deploy aşamalarından oluşmakta ve dağıtımın her adımı otomasyon ile yürütülmektedir. CI/CD süreci içerisinde webhookda gerekli parametreler tanımlanarak süreç hattı'ın tetiklenme koşulları özelleştirilmiştir. Şekil 4.2'de örnek bir CI/CD süreç hattı akışı görülmektedir.

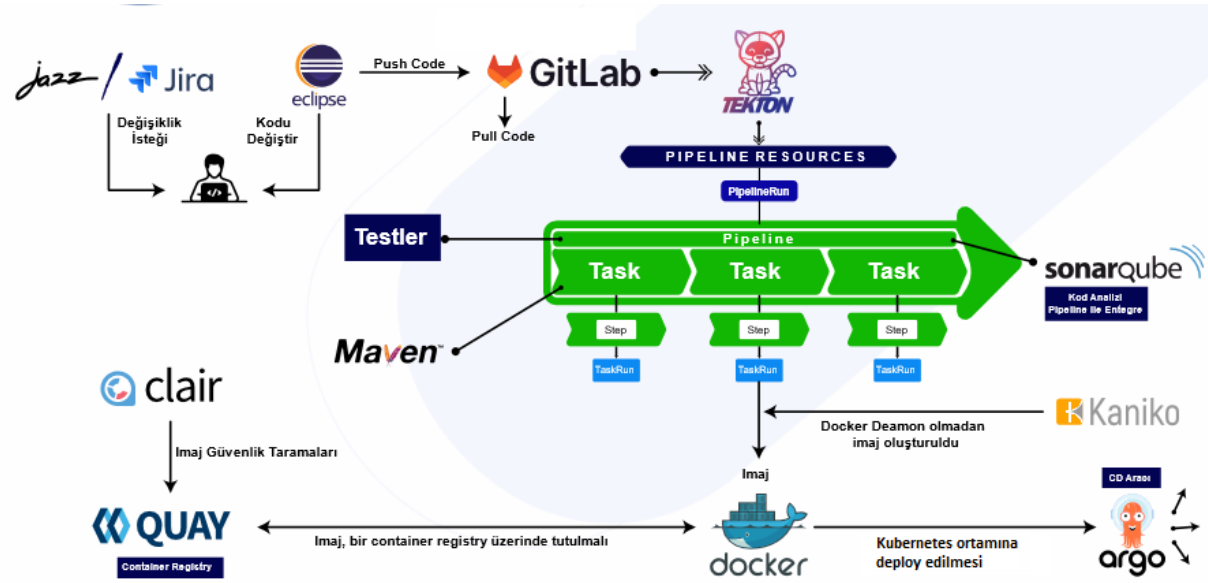


Şekil 4.2. DevOps CI Süreç Hattı akışı

Java tabanlı mikroservislerin geliştirilmesinde kullanılan Maven, hem bağımlılık yönetimi hem de proje yapılandırması açısından kritik bir rol oynamaktadır (Maven, 2026). Her bir proje için pom.xml dosyası üzerinden tanımlanan bağımlılıklar, CI sürecinde otomatik

olarak çözülmekte ve derleme işlemi tamamlanmaktadır. Maven ayrıca statik analiz öncesi derleme aşamasında kullanılmaktadır ve unit testlerin tetiklenmesinde de kullanılabilmektedir

Yazılımın kalitesinin sürekliliği açısından SonarQube ile statik kod analizleri yapılmış, OWASP ZAP entegrasyonu ile güvenlik açıkları test edilmiştir. Bu testler CI süreç hattının bir parçası hâline getirilerek geliştirici müdahalesi olmaksızın çalıştırılmıştır. Böylece kod birleştirme işlemi öncesinde kalite denetimleri tamamlanmış olur. Şekil 4.3'te bu entegrasyon şematik olarak gösterilmektedir.



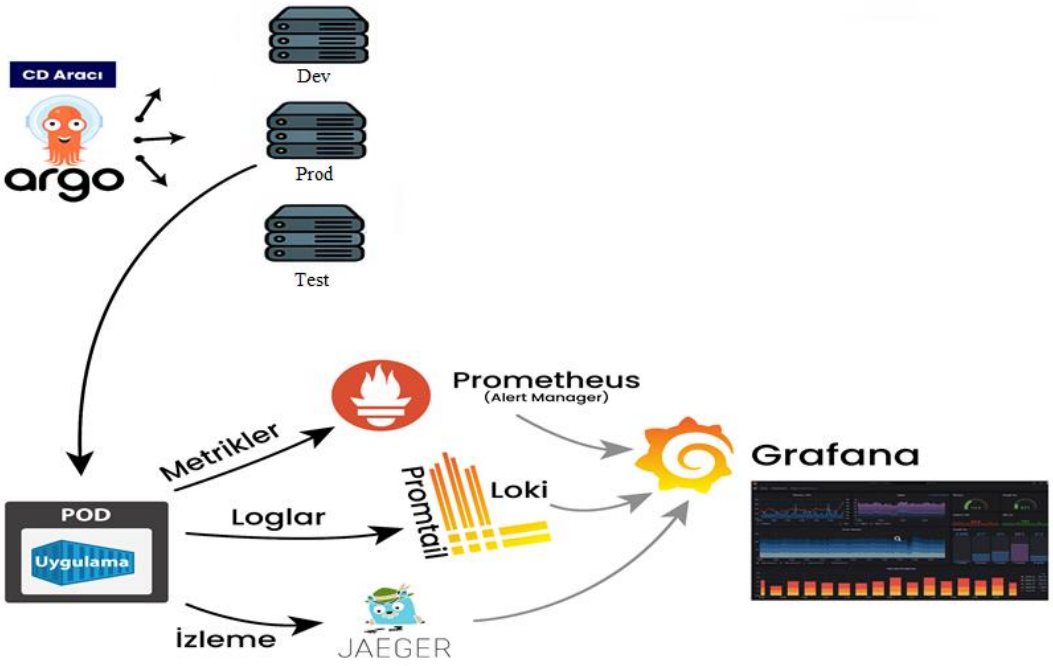
Şekil 4.3. CI süreci ve kullanılan araçlar

CI sürecinin sonunda uygulama bileşenleri güvenli bir şekilde Kaniko kullanılarak imaj hâline getirilir (Kaniko, 2026). Kaniko, Docker Daemon gerektirmeden konteyner imajlarını oluşturur ve bu sayede güvenli, izole derleme ortamları sunar. Bu yapı, Kubernetes-native süreçlerde imaj oluşturmayı mümkün kılar (Docker, 2026) .

Oluşturulan Docker imajları Quay sistemine yüklenmekte ve burada merkezi olarak yönetilmektedir (Quay, 2026). Quay, versiyonlama, erişim denetimi ve imaj tarama özellikleriyle kurumsal güvenlik gereksinimlerini karşılamaktadır. Bu imajlar, Clair güvenlik aracıyla otomatik olarak taranarak bilinen zafiyetlere karşı korunma sağlanır.

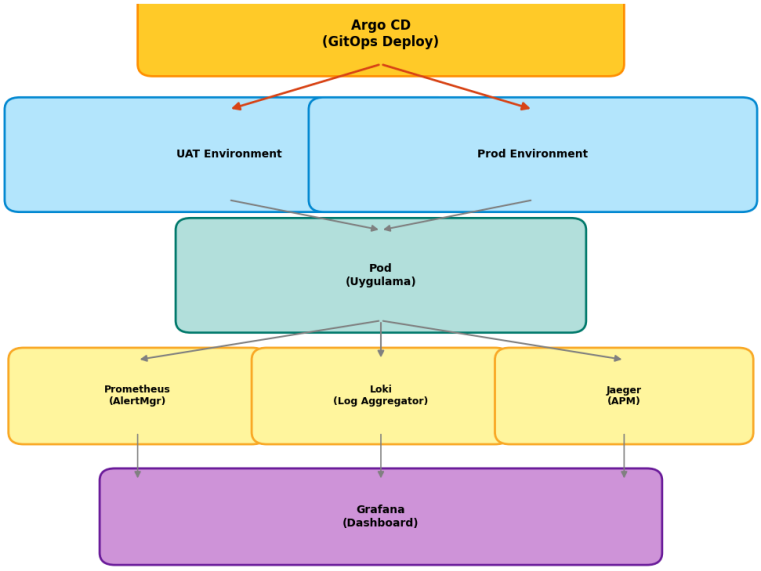
Quay'de saklanan imajların dağıtımının sağlanması için Kubernetes manifest dosyaları Helm ile Git deposunda tutulmakta ve Argo CD bu manifest dosyalarını sürekli izleyerek canlı ortamdaki uygulamalarla uyumlu olup olmadığını denetlemektedir. İstenen durum ile mevcut

durum arasında fark tespit edildiğinde otomatik olarak senkronizasyon sağlanır. Argo CD, görsel arayüzü ve rollback desteği ile dağıtım sürecini daha yönetilebilir kılar.



Şekil 4.4. CD süreci ve kullanılan izleme araçları

Canlı ortamdaki uygulamaların anlık izlenmesi amacıyla Prometheus ajanları kullanılarak metrikler toplanmış ve Grafana arayüzü ile görselleştirilmiştir. Sistemin CPU kullanımı, bellek tüketimi, istek yanıt süreleri gibi performans göstergeleri bu paneller aracılığıyla anlık olarak takip edilmiştir. Şekil 4.4'te örnek bir Prometheus izleme ekranı yer almaktadır.



Şekil 4.5. GitOps tabanlı CD ve çalışma zamanı izleme mimarisi

Dağıtık yapıdaki mikroservislerin kayıtları merkezi bir Loki yapısıyla toplanmış ve analiz edilmiştir (Loki, 2026). Bu sayede uygulamalarda oluşan hatalar veya olağandışı davranışlar tek bir merkezden izlenebilmiştir. Kayıt verileri indekslenerek geriye dönük hata analizi ve kök neden incelemesi yapılabilir hâle getirilmiştir.

Yazılımın performansını uçtan uca takip edebilmek amacıyla APM araçları entegre edilmiştir. Bu sistem, özellikle yüksek trafik altında darboğazların, yavaş sorguların ve sistemsel hataların tespitini kolaylaştırmakta ve geliştirme sürecine geri bildirim sağlamaktadır.

Hâlen WebSphere gibi geleneksel sunucular kullanılmakla birlikte, modern mikroservis mimarilerine uygun olarak Spring Boot ile bağımsız çalışan servisler geliştirilmektedir. Ayrıca ters proxy işlevi için NGINX kullanılmış, yük dengeleme ve yönlendirme işlemleri bu yapı üzerinden gerçekleştirilmiştir.

Konteyner tabanlı yapılarda kalıcı veriye ihtiyaç duyan servisler için S3 uyumlu bir nesne depolama çözümü olan Ceph tercih edilmiştir (Ceph, 2026). Uygulama içerisindeki medya dosyaları, kullanıcı yüklemeleri ve yedeklemeler Ceph üzerinde güvenli ve erişilebilir biçimde saklanmaktadır.

CI/CD altyapısında Tekton kullanılarak Kubernetes pipeline hatları oluşturulmuştur. Tekton; Task, Pipeline ve Trigger gibi bileşenlerle CI/CD iş akışlarını tanımlamaya olanak tanır ve tam konteyner uyumluluğu sağlar. Bu sayede CI/CD süreci doğrudan Kubernetes üzerinde yönetilebilir hâle gelmiştir.

Bu bölümde açıklanan DevOps uygulamaları sayesinde otomasyon, güvenlik, izleme, dağıtım ve veri yönetimi uçtan uca entegre edilmiştir. Argo CD, Kaniko, Tekton gibi modern çözümlerle desteklenen bu yapı; güvenli, sürdürülebilir ve ölçeklenebilir bir yazılım geliştirme ekosistemine zemin hazırlamıştır.

5. KURUMSAL UYGULAMANIN DEVOPS ALTYAPISINA ALINMASI VE DEVOPS METODOLOJİSİNİN UYGULANMASI

Bu bölümde, DevOps süreçlerinin kurumsal bir uygulamaya entegrasyonunu destekleyen proje hedefleri ve planlama araçları ele alınmaktadır. Projeye yönelik iş hedefleri ile birlikte teknik gereksinimler tanımlanmış, görev yönetimi için IBM Jazz aracı tercih edilmiştir. Alternatif olarak Jira da değerlendirilmiş; Jazz'ın esnek yapılandırma imkânları ve Jira'nın kapsamlı entegrasyon yetenekleri kıyaslanarak en uygun çözüm belirlenmiştir.

Uygulama, Java tabanlı Spring Boot çatısı altında geliştirilmiş ve Open Liberty sunucusu üzerinde çalışacak biçimde yapılandırılmıştır. Geliştiriciler RAD, Eclipse, IntelliJ IDEA veya VS Code gibi çeşitli IDE'ler kullanarak kodlama sürecini gerçekleştirmektedir. Kod bağımlılıkları ve derleme işlemleri için Maven tercih edilmiş; ön yüz katmanı ise Docker konteyneri içinde NginX ile paketlenmiştir.

Kod yönetimi ve sürüm kontrolü için GitLab kullanılmaktadır. Merge-request ve kod inceleme süreçleri GitLab üzerinden yönetilirken, CI/CD süreç hattı Tekton temelli olarak yapılandırılmıştır. Her yeni commit veya merge işlemi bir süreç hattı tetiklemekte, bu veri hattı içerisinde derleme, test ve dağıtım işlemleri sıralı olarak yürütülmektedir. Argo CD aracıyla birlikte GitOps prensiplerine uygun biçimde manifest dosyaları izlenmekte ve dağıtım süreçleri test ve üretim ortamlarında otomatik olarak gerçekleştirilmektedir.

İmaj oluşturma sürecinde Kaniko kullanılmakta; elde edilen imajlar Quay imaj deposunda saklanmaktadır. Güvenlik taramaları ise Clair aracı aracılığıyla gerçekleştirilmektedir. Bu yapıyla birlikte, CI/CD sürecinin sonunda oluşturulan uygulama imajları güvenli biçimde dağıtıma hazır hale getirilmektedir.

Test süreçlerinin otomatikleştirilmesi ile yazılım kalitesi garanti altına alınmaktadır. Java projelerinde birim testler için JUnit kullanılırken, CI pipeline hattına entegre edilen SonarQube aracıyla statik kod analizleri gerçekleştirilmekte; potansiyel hatalar, kod kokuları ve güvenlik açıkları erken safhada tespit edilmektedir.

Sistem metriklerinin ve performans göstergelerinin izlenmesi amacıyla Prometheus kullanılmakta; uyarı yönetimi Alertmanager ile gerçekleştirilmektedir. Loki aracı, uygulama ve altyapı kayıtlarını merkezi olarak toplamaktadır. Uygulama performans yönetimi için Jaeger tercih edilmiş (Jaeger, 2026), alternatif olarak Sentry aracı da değerlendirilmiştir (Sentry, 2026). Tüm veriler Grafana panolarında görselleştirilerek sistem durumu gerçek zamanlı olarak izlenmektedir.

5.1. Analiz, Planlama, Proje Yönetimi ve Görev Takibi

DevOps metodolojisinin kurumsal ölçekte etkin biçimde uygulanabilmesi için, proje yönetimi, analiz ve görev takibi süreçlerinin bütünleşik bir yaklaşımla ele alınması gerekmektedir. Bu kapsamda kullanılan araçlar, ekipler arası koordinasyonu artırmanın yanı sıra, yazılım yaşam döngüsünün ilk aşamalarını sistematik biçimde yönlendirmeyi amaçlamaktadır.

Proje yönetimi ve görev takibi süreçlerinde IBM Rational Team Concert (Jazz) ve Jira gibi araçlar, iş akışlarının planlanması ve takibi açısından önemli işlevler sunmaktadır. Jazz, IBM tarafından kurumsal çözümler için geliştirilen web tabanlı bir araç olarak, proje yönetimi gereksinimlerini karşılayan geniş bir fonksiyon setine sahiptir. Jazz'ın dikkat çeken yönlerinden biri, iş akışlarının ekiplerin ihtiyaçlarına göre özelleştirilebilmesine imkân tanımasıdır. Bu yönüyle özellikle küçük ve orta ölçekli projeler için sade, kullanıcı dostu ve erişilebilir bir platform sunmaktadır.

Jira ise daha geniş ölçekli projelerde esnek yapılandırma seçenekleri, güçlü entegrasyon yetenekleri ve gelişmiş raporlama özellikleriyle öne çıkmaktadır. Jira'nın sunduğu zengin eklenti ekosistemi, proje takibini özelleştirilebilir hale getirirken; DevOps araç zinciriyle olan doğal uyumu da entegrasyonu kolaylaştırmaktadır. Pan ve ark. (2025), Jira Yazılımı, planlama, izleme ve sürüm yönetiminde ekip işbirliğini kolaylaştırma yetkinliğiyle sorunsuz işbirliğini kolaylaştırmada yüksek beğeni toplamaktadır.

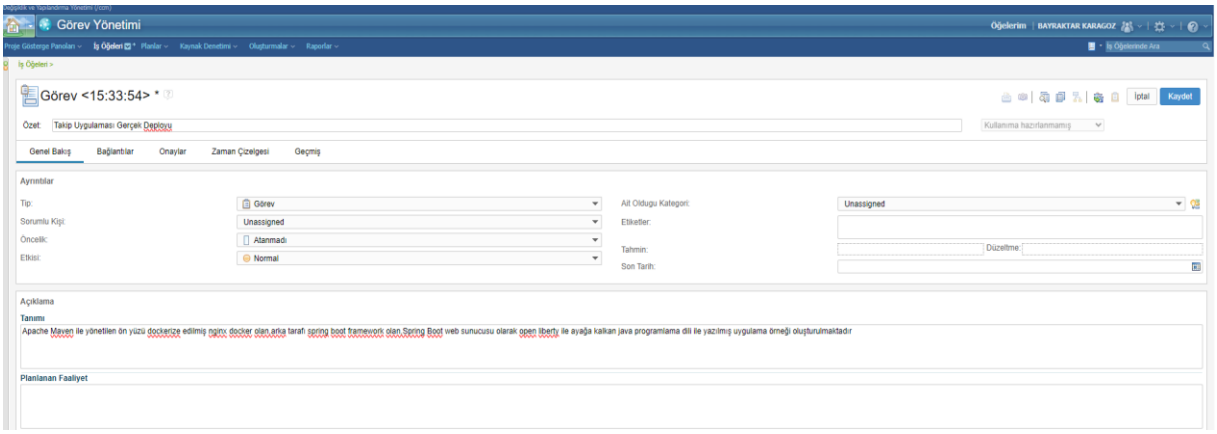
Bu tez kapsamında, öncelikli olarak Jazz kullanımı benimsenmiş; ancak ilerleyen süreçte açık kaynak kodlu alternatiflerden biri olan Open Project değerlendirmeye alınmıştır. Ayrıca karşılaştırmalı analiz amacıyla Jira'nın potansiyeli de incelenmiş; Jazz'ın sade kullanım arayüzü ile Jira'nın genişletilebilir yapısı arasında kurumsal ihtiyaçlara göre bir denge kurulması hedeflenmiştir.

Bu doğrultuda gerçekleştirilen analiz ve planlama süreci, DevOps altyapısına geçişte ihtiyaç duyulan tüm görevlerin açık şekilde tanımlanmasını, önceliklendirilmesini ve izlenmesini sağlamıştır. Proje takibi ile görev yönetimi arasındaki entegrasyon sayesinde, planlama aşamasında tanımlanan gereksinimler kodlama, test ve dağıtım aşamalarına eksiksiz biçimde yansıtılabilmektedir.

Bu aşamada, geliştiricilere sistematik görev atamaları gerçekleştirilerek DevOps sürecinin ilk adımı somutlaştırılır. Örneğin, Şekil 5.1'de gösterildiği gibi, Jazz platformu

üzerinden bir iş maddesi (work item) oluşturularak, belirli bir görev ilgili yazılımcıya veya ekip üyelerine atanır. Bu yapı, hem görevlerin takibini kolaylaştırmakta hem de proje yönetimi açısından izlenebilirlik ve hesap verebilirlik sağlamaktadır.

Görev atamasıyla birlikte DevOps süreci etkin biçimde başlamış olur. Geliştirici, tanımlanan iş gereksinimlerini yerine getirmek amacıyla ilgili kodları geliştirir, gerekli testleri planlar ve sistemin CI/CD altyapısına uyumlu hale getirilmesini sağlar. Bu noktada, kurumların mevcut sistem mimarisi de dönüşüme tabi tutulabilir. Örneğin, geleneksel bir WebSphere Application Server (WAS) üzerinde çalışan bir uygulamanın, daha hafif, konteyner tabanlı ve modern mimarilerle uyumlu bir platform olan Open Liberty veya Apache Tomcat gibi sunuculara taşınması gerekebilir. Bu geçiş, yalnızca platform bağımlılıklarını azaltmakla kalmaz, aynı zamanda konteynerleştirme süreçlerine daha uygun bir yapı sunarak, DevOps entegrasyonunun sürdürülebilirliğini artırır.



Şekil 5.1. Jazz da iş maddesi oluşturma süreci

Aynı şekilde, yalnızca yazılım geliştirme görevleri değil; sistem yöneticileri, test mühendisleri veya DevOps ekip üyeleri için tanımlanması gereken işler de bu platform aracılığıyla atanmaktadır. Böylece projenin her aşamasında ilgili sorumluluklar sistemli biçimde paylaştırılmakta ve süreç boyunca kurumsal görünürlük sağlanmaktadır.

Jazz'ın öne çıkan özelliklerinden biri de, yazılım geliştirme süreçlerinde oldukça kritik olan kod gözden geçirme süreçlerine entegre destek sunmasıdır. Ekipler, geliştirdikleri kodlar üzerinde doğrudan platform üzerinden değerlendirme yapabilir, yapıcı geri bildirimlerde bulunabilir ve revizyon süreçlerini kayıt altına alabilirler. Bu da sürüm kalitesini artırmakta ve hataların erken aşamada tespit edilmesine olanak tanımaktadır.

Buna ek olarak, Jazz'ın derleme yönetimi özellikleri, yazılım üretim sürecinin temel yapıtaşlarından biri olan CI adımlarını da desteklemektedir. Kod gönderiminden sonra otomatik

derleme ve temel testlerin başlatılması gibi işlemler, geliştiricilerin manuel müdahaleye gerek kalmadan akışı sürdürebilmesini sağlamaktadır.

Ancak tüm bu güçlü yönlerine rağmen, Jazz platformunun modern DevOps ekosistemindeki bazı araçlarla olan entegrasyonlarında kısıtlılıklar yaşanabilmektedir. Özellikle konteyner tabanlı dağıtım, mikrosistem mimarileri ve dinamik CI/CD süreç hatları gibi güncel DevOps yapı taşlarıyla doğrudan ve esnek entegrasyon yeteneklerinin sınırlı olması, bazı durumlarda alternatif ya da tamamlayıcı araçların kullanımını gerektirebilmektedir.

5.1.1. IBM Rational Team Concert (Jazz)

DevOps süreçlerinin projeye uygulanması kapsamında halihazırda Jazz aktif olarak kullanılmaktadır. Jazz, IBM'in kurumsal çözümlerine entegre biçimde çalışabilen, web tabanlı bir proje ve görev yönetim aracıdır. Bu platform, proje yönetimi faaliyetlerinde ekiplerin iş akışlarını kendi gereksinimlerine göre özelleştirmesine olanak tanıyan esnek bir yapı sunmaktadır.

Jazz, küçük ve orta ölçekli projeler için yeterli seviyede işlevsellik sağlarken, anlaşılır kullanıcı arayüzü sayesinde proje paydaşlarının sisteme hızlı biçimde adapte olmasına katkı sağlar. Jazz platformu üzerinde görev atamaları, "Create Work Item" arayüzü üzerinden gerçekleştirilmekte olup; bu arayüzde görev türü (örneğin geliştirme talebi, hata kaydı), öncelik düzeyi, hedef teslim tarihi gibi bilgilerin yanı sıra açıklamalar ve teknik gereksinimler gibi detaylar da eklenebilmektedir.

Şekil 5.1’de görüldüğü üzere, yazılımcıya ait bir görev tanımlanmış ve bu görev çerçevesinde uygulamada yapılması gereken değişiklikler belirtilmiştir. IBM Jazz platformu üzerinde yapılan bu tür atamalar, hem bireysel hem de ekip düzeyinde izlenebilirlik sağlar. İş maddelerine ekip üyelerinin atanabilmesi ve ilerlemelerin sistem üzerinden takip edilebilmesi, proje yönetiminde şeffaflık ve koordinasyonu güçlendirmektedir.

Ayrıca Jazz, kod inceleme ve derleme süreçlerini de desteklemektedir. Kod değişiklikleri platform üzerinden değerlendirilerek geliştirici ekipler arasında etkileşim artırılabilir. Derleme yönetimi bileşeni ise, CI süreçlerinin temellerini Jazz içerisinde oluşturmaya imkân tanır. Bununla birlikte, modern DevOps uygulamalarının gerektirdiği tam otomasyon ve üçüncü taraf araçlarla geniş entegrasyon açısından bazı sınırlılıklar mevcuttur.

Jazz, lisans maliyetleri açısından daha uygun bir çözüm sunmakta ve özellikle IBM ürünlerini kullanan kurumlar için cazip bir seçenek olarak değerlendirilmektedir. Ancak, kullanıcı arayüzünün görece daha az gelişmiş olması ve sınırlı entegrasyon yetenekleri, bazı

ekipler için dezavantaj teşkil edebilmektedir. Bu bağlamda, Jazz daha çok maliyet duyarlılığı olan ve IBM ekosistemine entegre çalışan kurumlar için uygun bir tercih olarak öne çıkmaktadır.

Dijital dönüşüm hedefleri kapsamında birçok kurum ve şirket, mevcut Jazz yapısını modern ve entegre bir altyapıyla güçlendirmeyi hedeflemekte, bu sürecin ilerleyen aşamalarında ise Jira'ya geçişi planlamaktadır. Bu geçiş süreci, daha güçlü entegrasyon olanakları ve gelişmiş kullanıcı deneyimi sağlayan modern DevOps araçlarının benimsenmesini hedeflemektedir. Ancak geçiş tamamlanana kadar proje yönetim süreçleri Jazz üzerinden sürdürülecektir.

5.1.2. Jira

Jira, Atlassian ekosisteminin temel bileşenlerinden biri olarak, modern yazılım geliştirme süreçlerine yüksek uyum sağlayan gelişmiş bir proje yönetim aracıdır. DevOps süreçlerine tam entegrasyon sunan yapısıyla, otomatikleştirilmiş iş akışlarının kurulmasına, ekipler arası koordinasyonun artırılmasına ve CD döngüsünün yönetilmesine olanak tanır.

Jira'nın kullanıcı dostu arayüzü üzerinden, yeni bir iş kaydı "Create Issue" butonu aracılığıyla kolaylıkla oluşturulabilir. Bu formda, iş kaydının türü (görev, hata, iyileştirme vb.), başlığı, açıklaması, öncelik düzeyi, sorumlu kişi ve diğer ilgili bilgiler detaylı biçimde tanımlanabilir. Bu yapı, ekiplerin görev takibini verimli ve şeffaf bir biçimde yönetmesine imkân tanır.

Jira, Jenkins, GitLab CI, CircleCI gibi popüler CI/CD araçlarıyla doğrudan entegre olabilmekte, webhook desteği sayesinde kod değişikliklerinin otomatik olarak iş akışlarına yansımasını sağlamaktadır. Ayrıca Bitbucket ve Confluence gibi Atlassian ürünleriyle sağladığı bütünleşik yapı, dokümantasyon, kod versiyonlama ve bilgi paylaşımı süreçlerini tek platformdan yürütmeye olanak tanır.

Agile proje yönetimi özellikleri açısından da güçlü bir altyapı sunan Jira, Scrum ve Kanban gibi metodolojileri destekleyen şablonlar ve panolarla sprint planlama, backlog yönetimi ve retrospektif süreçlerini etkin biçimde yürütmeyi mümkün kılar.

Gelişmiş raporlama yetenekleriyle proje yöneticilerine karar destek sağlayan Jira; özelleştirilebilir dashboard'ları, filtrelenebilir veri panelleri ve görsel analiz araçları ile ekiplerin performans değerlendirmesini ve proje izlenebilirliğini artırmaktadır.

Ancak Jira’nın gelişmiş özellikleri, beraberinde daha yüksek maliyetleri de getirmektedir. Kullanıcı başına lisanslama modeli ve ücretli eklentiler, özellikle geniş kullanıcı tabanına sahip kurumlar için önemli bir maliyet kalemi oluşturabilir. Ayrıca kurulum ve yapılandırma süreci, Jazz'a kıyasla daha karmaşık olup ileri düzey teknik bilgi gerektirebilir.

Sonuç olarak, Jira modern DevOps pratiklerine daha uygun, güçlü entegrasyon olanaklarına sahip bir çözümdür. Ancak bu kapsamlı yapının getirdiği lisans maliyeti ve kurulum karmaşıklığı, kurumların bütçe ve insan kaynağına göre değerlendirilmelidir. Jazz, düşük maliyetli ve IBM sistemleriyle entegre çalışan kurumlar için makul bir çözüm sunarken; Jira, geniş ölçekli, yüksek entegrasyon gereksinimi olan ve çevik yöntemleri benimseyen organizasyonlar için daha uygun bir platformdur.

5.1.3. Geçiş Stratejisi: Jazz’tan Jira’ya Dijital Dönüşüm Perspektifiyle

Günümüzde birçok kurumda proje yönetimi ve görev takibi için Jazz kullanılmakla birlikte, modern DevOps uygulamalarının daha etkin entegre edilebilmesi, CI/CD otomasyonunun kesintisiz sürdürülebilmesi ve geliştirici deneyiminin iyileştirilmesi amacıyla Jira’ya geçiş planlanmaktadır. Bu geçiş süreci, yalnızca bir araç değişimi değil, aynı zamanda kurumların dijital dönüşüm hedeflerine ulaşması için gerekli yapısal dönüşümün bir parçası olarak değerlendirilmektedir. Jazz ile yürütülen proje yönetimi faaliyetleri, belirli eşiklerin aşılmasıyla birlikte Jira ortamına aktarılabilir. Bu geçiş süreci aşamalı, kontrollü ve geri izlenebilir biçimde tasarlanmalıdır.

5.2. KOD

5.2.1. Uygulama Sunucusu: Open Liberty

DevOps tabanlı modern yazılım geliştirme süreçlerinde uygulama sunucularının esneklik, hız, uyumluluk ve kaynak verimliliği gibi kriterleri karşılaması büyük önem taşımaktadır. Bu bağlamda, Open Liberty, mikroservis mimarisi ve bulut yerel uygulamalar için optimize edilmiş, hafif ve modüler bir Java uygulama sunucusudur.

Open Liberty, Jakarta EE ve Eclipse MicroProfile gibi modern Java teknolojileriyle tam uyumluluk sunmakta ve geliştiricilere yalnızca ihtiyaç duyulan modülleri seçerek çalıştırma olanağı sağlamaktadır. Bu modüler yapı, başlatma süresini kısaltmakta, bellek kullanımını optimize etmekte ve yüksek erişilebilirlik sağlayan sistem mimarilerinin kurulmasına katkıda bulunmaktadır.

Ayrıca, Open Liberty'nin kesintisiz güncelleme özelliği sayesinde, çalışan uygulamalar üzerinde yapılan değişiklikler canlı ortamda uygulanabilmekte, bu da CD süreçlerini kesintiye uğratmadan sürdürmeyi mümkün kılmaktadır. Bu özellik, özellikle Argo CD gibi GitOps araçlarıyla yapılan dağıtımlarda uygulamaların versiyonlanması ve güvenli biçimde canlıya alınması sürecini kolaylaştırmaktadır.

Konteyner teknolojileriyle yüksek düzeyde uyumlu olan Open Liberty, Docker ve Kubernetes ortamlarına kolaylıkla entegre edilebilmekte, bu sayede uygulama dağıtımları hızlı, taşınabilir ve ölçeklenebilir hale gelmektedir. Bu yapı, CI/CD veri hatlarına doğrudan entegre edilerek sürekli entegrasyon ve dağıtımın kesintisiz ve otomatize bir şekilde yönetilmesini desteklemektedir.

Bellek ve kaynak yönetimi açısından ise, sadece aktif kullanılan bileşenlerin yüklenmesini sağlayan dinamik servis yönetimi, mikroservis mimarilerinde düşük kaynak tüketimi ile yüksek performans elde edilmesini mümkün kılmaktadır. Ayrıca, Open Liberty'nin kademeli sürüm geçişlerine olanak tanıyan mimarisi, sürüm yönetimi süreçlerinde esneklik sağlamakta ve büyük güncellemelerden kaynaklanan riskleri azaltmaktadır.

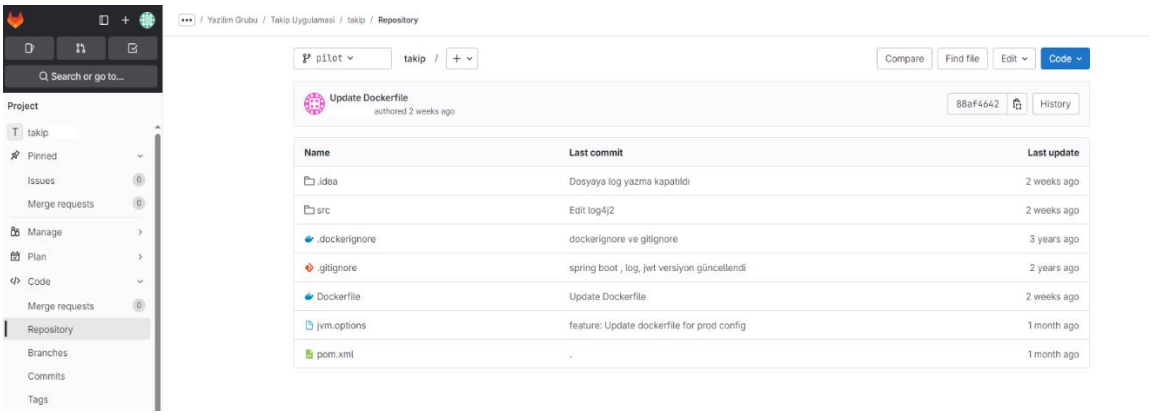
Ticari bir çözüm olan IBM WebSphere Application Server (WAS)'ın maliyetli lisans yapısına karşın, Open Liberty açık kaynak kodlu yapısı ve IBM desteği ile kurumlara hem teknik hem de ekonomik açıdan avantaj sunmaktadır. Bu bağlamda, birçok kurumda hâlihazırda WebSphere Application Server (WAS) üzerinde çalışan bazı uygulamaların Open Liberty'ye taşınmasına yönelik çalışmalar başlatılmakta ve geliştiricilere bu geçişlerde teknik destek sağlanmaktadır..

Bahsedilen DevOps altyapısına entegre edilen uygulamalar genellikle Open Liberty üzerinde çalıştırılmakta olup, yazılımcılarla koordineli biçimde, geleneksel sunucu mimarilerinden daha modern ve bulut uyumlu Open Liberty platformuna geçişler sağlanmaktadır.

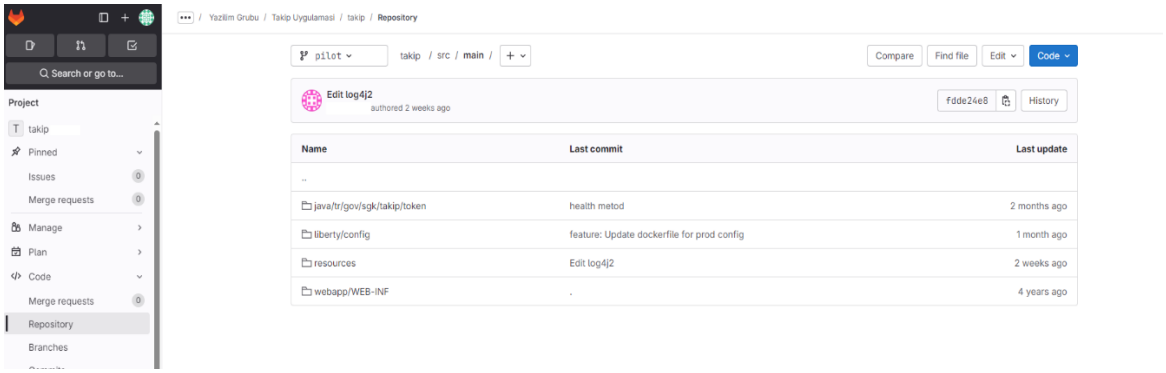
Uygulamanın yapılandırma detayları, Şekil 5.2, 5.3 ve 5.4'te sunulmuştur.

- Şekil 5.2, uygulamanın GitLab üzerindeki kod deposunda yer alan dizin yapısını ve kritik dosyaları göstermektedir. Bu yapıda, uygulama bağımlılıklarının tanımlandığı pom.xml ve uygulamanın konteyner imajına dönüştürülmesini sağlayan Dockerfile, ana dizin altında yer almaktadır.
- pom.xml, Maven aracılığıyla Java sürümü, bağımlılıklar, paketleme türü (JAR/WAR), test yapılandırmaları ve kaynak dizinleri gibi konfigürasyonları yönetmekte; derleme sürecinde ihtiyaç duyulan bileşenlerin teminini sağlamaktadır.
- Dockerfile ise, Maven ile oluşturulan derlenmiş uygulamayı bir konteyner imajına dönüştürmekte kullanılır. Bu iki dosya birlikte, hem derleme hem de konteynerizasyon sürecinin temel bileşenlerini oluşturmaktadır.
- Şekil 5.3 ve 5.4, uygulamanın dizin yapısını daha ayrıntılı biçimde sergilemekte ve Open Liberty sunucusuna özel yapılandırma detaylarını içermektedir.
- Özellikle Şekil 5.4'te yer alan server.xml dosyası, sunucunun çalışma davranışını tanımlayan temel yapılandırma bileşeni olup, uygulamanın çalışma portları, kaynak bağlantıları ve özellik profillerini içermektedir.

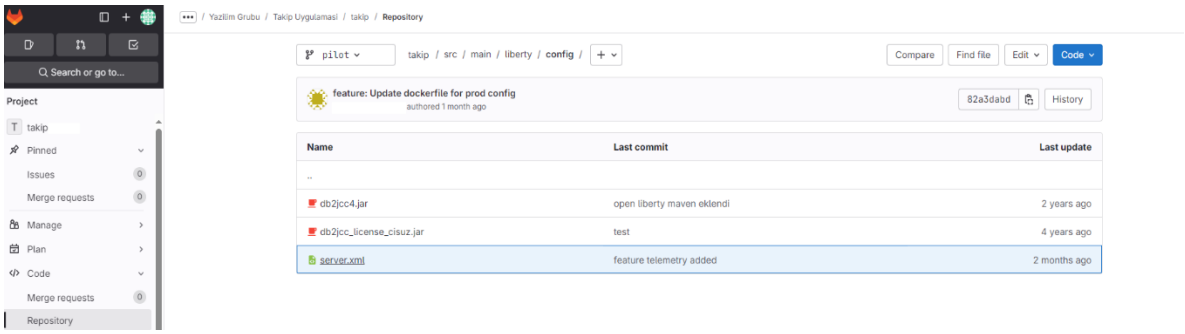
Sonuç olarak, Open Liberty; DevOps altyapısına entegre çalışan yazılım projelerinde, yüksek performans, kolay bakım, modüler yapı, otomasyon dostu dağıtım ve lisans maliyeti avantajı gibi kriterler çerçevesinde ihtiyaçları etkili şekilde karşılayan bir uygulama sunucusu olarak konumlandırılmıştır.



Şekil 5.2. Projenin Open Liberty yapısı



Şekil 5.3. Projenin Open Liberty dizin yapısı



Şekil 5.4. Projenin Open Liberty dizin yapısı 2

5.2.2. Tomcat Uygulama Sunucusu

Apache Tomcat, Java tabanlı web uygulamaları için geliştirilmiş açık kaynaklı bir uygulama sunucusudur ve özellikle HTTP protokolü üzerinden web servisleri sağlamak üzere tasarlanmıştır. Jakarta Servlet, JSP ve WebSocket gibi temel Java web teknolojilerini destekleyen bu platform, özellikle kurumsal ölçekteki dinamik web uygulamalarının barındırılması için hafif ve etkili bir çözüm sunmaktadır.

Tomcat'in mimarisi üç temel bileşen üzerine kuruludur: iletişim katmanı, uygulama yönetim birimi ve oturum denetleyici.

- İletişim katmanı, gelen istemci isteklerini karşılayarak uygun işleyici birimlere yönlendirir ve güvenli bağlantılar ile HTTP protokolünün yönetimini sağlar.
- Uygulama yönetim birimi, web uygulamalarının yaşam döngüsünü yönetir, eşzamanlı istekleri dengeleyerek sistem kaynaklarının etkin kullanımını sağlar ve dinamik içerik üretimini destekler.
- Oturum denetleyici ise kullanıcı oturumlarının takibini gerçekleştirir ve kullanıcıya özel verilerin yönetimini koordine eder.

Tomcat'in özelleştirilebilir yapısı, çeşitli proje gereksinimlerine göre yapılandırılmasına imkân tanır. Güvenlik açısından; kimlik doğrulama, SSL şifreleme ve erişim kontrolü gibi özellikleri destekler. Bu yönüyle Tomcat, özellikle güvenliğin ön planda olduğu uygulamalarda yaygın biçimde tercih edilmektedir.

Ayrıca, Docker ve Kubernetes gibi konteyner teknolojileri ile sorunsuz entegrasyonu, Tomcat'i DevOps mimarileri için de cazip kılmaktadır. Hafif yapısı sayesinde CI/CD süreç hatlarında daha hızlı derleme ve dağıtım süreçlerine olanak sağlar.

Kurumsal DevOps uygulamalarında, ticari WebSphere Application Server (WAS) gibi alternatiflerin yüksek lisans maliyetleri göz önüne alındığında, Tomcat ekonomik ve etkin bir çözüm olarak ön plana çıkmaktadır. Nitekim bazı uygulama örneklerinde, Tomcat'in entegrasyon sürecinde Open Liberty'ye kıyasla daha çevik bir yapı sergilediği ve birçok projede daha hızlı devreye alınabildiği gözlemlenmiştir.

Bununla birlikte, tez kapsamında ele alınan örnek kurumsal uygulamada tercih edilen sunucu yapısı Open Liberty'dir. Ancak Tomcat, sistem mimarisinin ihtiyacına bağlı olarak alternatif bir uygulama sunucusu seçeneği olarak her zaman değerlendirilmektedir.

5.3. İmaj Reposu: Quay

Konteyner teknolojilerinin modern yazılım geliştirme süreçlerinde yaygınlaşmasıyla birlikte, imajların güvenli biçimde saklanması ve yönetilmesi kritik bir ihtiyaç haline gelmiştir. Bu kapsamda kullanılan Quay, konteyner imajlarının yaşam döngüsünü uçtan uca yöneten, Kubernetes uyumlu, kurumsal düzeyde bir imaj deposudur.

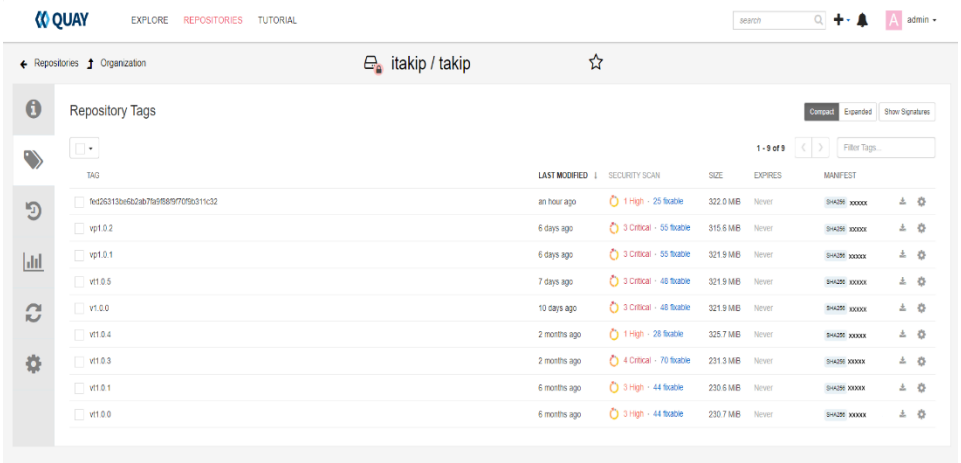
Quay'in temel işlevleri arasında şunlar yer almaktadır:

- **İmaj Güvenliği:** İmajların bütünlüğü, gelişmiş şifreleme algoritmaları ve kapsamlı erişim denetimi ile korunur. Sisteme imaj yükleme ve sistemden çekme işlemleri güvenlik politikalarıyla uyumlu şekilde gerçekleştirilir.

- **Ölçeklenebilirlik:** Yüksek trafikli ortamlarda dahi hizmet kesintisi yaşanmadan çalışabilen mimarisi sayesinde Quay, kurumsal projelerde sürdürülebilir bir altyapı sağlar. Yük dengeleme ve otomatik ölçeklendirme gibi yetenekleri, kaynak kullanımını optimize eder.
- **Güvenlik Tarama Entegrasyonu:** Sisteme yüklenen imajlar sürekli olarak güvenlik açıklarına karşı taranır. Özellikle Clair aracıyla gerçekleştirilen bu taramalar, potansiyel tehditlerin erken evrede tespit edilmesini ve gerekli önlemlerin alınmasını sağlar.
- **Gelişmiş İzleme ve Kayıt Tutma:** İmajların sürüm geçmişi, erişim kayıtları ve değişiklik geçmişi detaylı biçimde kaydedilir ve analiz edilebilir hale getirilir. Bu sayede hem yasal denetim hem de operasyonel takip kolaylaşır.

Şekil 5.5'de görüldüğü üzere, Tekton süreç hattı'ı tarafından oluşturulan imajlar, otomatik olarak Quay deposuna gönderilmektedir. Depoya gönderilen imajların sürüm geçmişi korunmakta, geçmiş imajlar erişilebilir durumdadır. Sürecin devamında Argo CD, bu imajları çekerek hedef Kubernetes kümelerine dağıtım işlemini gerçekleştirmektedir.

Quay, bu bağlamda yalnızca bir saklama alanı değil, DevOps yaşam döngüsünün güvenlik, izlenebilirlik ve ölçeklenebilirlik boyutlarında merkezi bir rol üstlenmektedir. Bu proje kapsamında da Quay, CI sürecinin son aşamasında imajların merkezi olarak saklandığı ve dağıtıma hazırlandığı ana yapı olarak konumlandırılmıştır. İmajların versiyonu gitlabdaki commit id olarak da ayarlanabilmektedir.



Şekil 5.5. Quay kullanımı

5.3.1. Clair: Konteyner Güvenlik Taraması

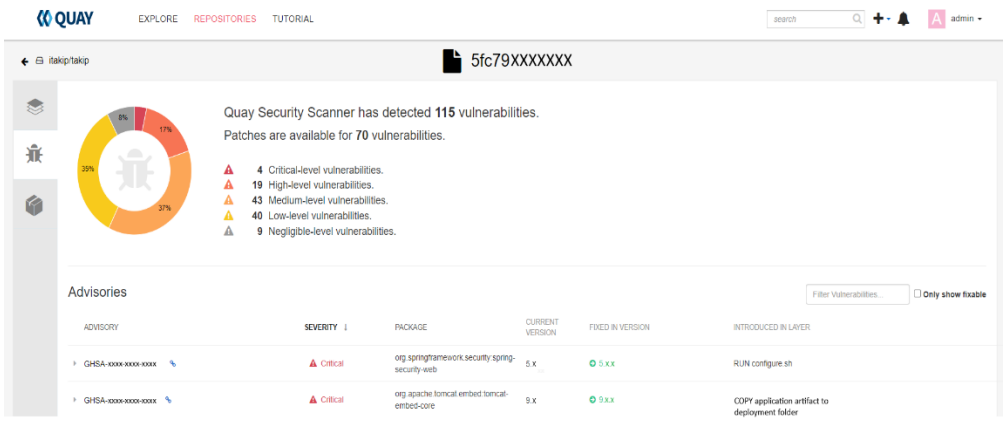
Konteyner güvenliğini sağlamak amacıyla kullanılan Clair, imajların güvenlik analizini gerçekleştiren açık kaynaklı bir tarama sistemidir. Quay ile entegre biçimde çalışan Clair, konteyner imajlarının içeriğini analiz ederek, bilinen güvenlik açıklarını tespit eder ve detaylı raporlar üretir. Bu sistem, güvenlik odaklı yazılım geliştirme ve dağıtım süreçlerinin ayrılmaz bir bileşeni haline gelmiştir.

Clair'in çalışma mekanizması, güncel ve sürekli genişleyen bir güvenlik zafiyeti veritabanına dayanmaktadır. Sistemin temel işlevleri şu şekilde özetlenebilir:

- **Katman Bazlı Tarama:** Konteyner imajları, bileşenlerine ayrılarak katman katman analiz edilir. Her bir katmanda yer alan yazılım paketleri ve sürümleri incelenerek, olası güvenlik açıkları belirlenir.
- **Zafiyet Tespiti:** Bilinen Yaygın Güvenlik Açıkları ve Maruziyetler (CVE) kayıtlarıyla karşılaştırma yapılır. Bu sayede, imaj içinde yer alan yazılım bileşenlerinin hangi güvenlik açıklarına karşı savunmasız olduğu belirlenir.
- **Otomatik Güvenlik Süreçlerine Entegrasyon:** Clair, CI/CD veri hatlarına entegre edilebilir. Böylece derleme aşamasında otomatik taramalar yapılarak, sorunlu imajların üretim ortamına geçmesi engellenebilir.
- **Raporlama:** Tespit edilen açıklar sistem yöneticilerine ve geliştiricilere detaylı şekilde sunulur. Güvenlik seviyesi, açıklığın ciddiyeti (örneğin: kritik, yüksek, orta, düşük) ve çözüm önerileri gibi bilgiler bu raporlarda yer alır.

Şekil 5.6'da görüldüğü üzere, Quay'a yüklenen bir imaj, Clair tarafından otomatik olarak taranmış ve mevcut güvenlik açıkları raporlanmıştır. Bu yapı sayesinde güvenlik, sadece manuel test süreçlerine değil; otomatik ve sürekli analiz mekanizmalarına da entegre edilmiştir.

Clair, proje kapsamında kullanılan DevOps altyapısında, konteyner imajlarının güvenliğini denetlemek amacıyla aktif olarak yapılandırılmış ve kullanılmıştır. Bu sayede yazılımın güvenliğine ilişkin süreçler, dağıtım öncesi aşamalarda kontrol altına alınarak, üretim ortamına potansiyel tehditlerin taşınması engellenmiştir.



Şekil 5.6. Clair raporu ve güvenlik bugları kullanımı

5.4. Test Otomasyonu: SonarQube ile Statik Kod Analizi

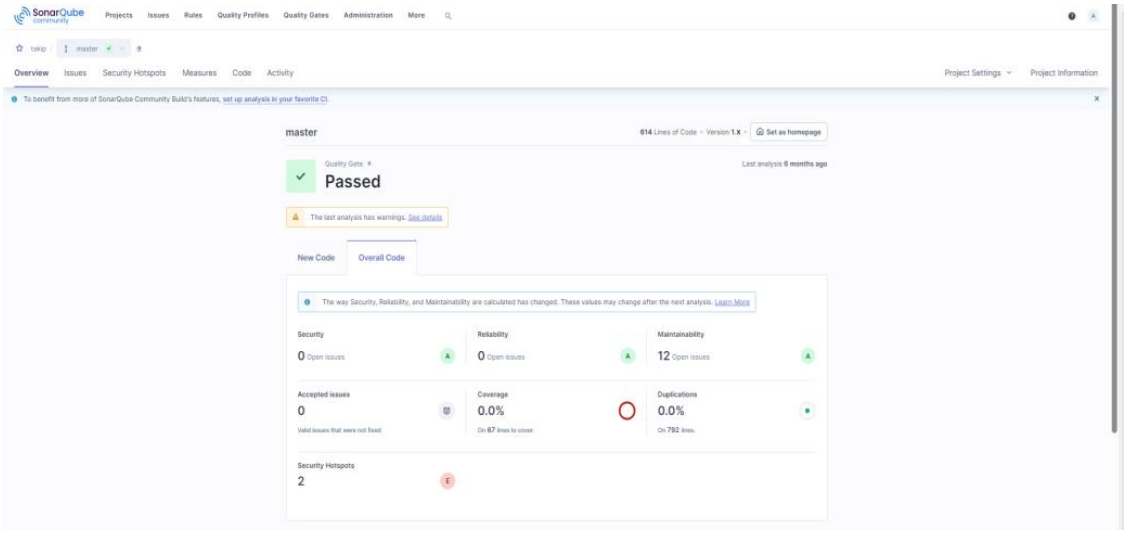
DevOps süreçlerinde yazılım kalitesinin sürekliliğini sağlamak amacıyla otomatik test sistemlerinin yanı sıra statik kod analizi araçları da yoğun şekilde kullanılmaktadır. Bu bağlamda, proje kapsamında tercih edilen araçlardan biri olan SonarQube, kaynak kodun kalite, güvenlik ve sürdürülebilirlik açısından sürekli denetlenmesini sağlayan kapsamlı bir platformdur.

SonarQube, farklı programlama dillerini destekleyerek, geniş bir teknoloji yelpazesine hitap eden projelerde etkin olarak kullanılabilmektedir. Sistem, geliştirilen yazılımları; potansiyel hatalar, güvenlik açıkları, kod tekrarları ve teknik borç unsurları yönünden analiz eder. Bu analizlerin sonuçları, geliştiricilere detaylı raporlar halinde sunularak kod kalitesinin artırılması yönünde rehberlik sağlar.

Platformun DevOps ortamlarında öne çıkan avantajlarından biri, CI/CD veri hatlarına kolayca entegre edilebilmesidir. Yapılandırılan otomasyon sayesinde, her kod gönderimi sonrasında SonarQube tarafından tetiklenen taramalar ile kod kalitesi gerçek zamanlı olarak izlenebilir. Bu sayede geliştiriciler, kodlarının sürdürülebilirlik, güvenlik ve hata potansiyeli açısından hangi düzeyde olduğunu erken aşamada görebilirler.

Şekil 5.7'de görüldüğü üzere, analiz edilen örnek kodda kalite kapısı başarıyla geçilmiş; güvenlik ve güvenilirlik açısından kritik bir açık tespit edilmemiştir. Ancak, sürdürülebilirlik başlığı altında 12 hata saptanmış, test kapsama oranı %0 ve kod tekrar oranı da %0 olarak raporlanmıştır. Bu, testlerin uygulanmadığını ve kodda gereksiz çoğaltmalara yer verilmediğini göstermektedir. Ayrıca iki adet “security hotspot” tespit edilmiştir; bu, potansiyel güvenlik riski taşıyan kod parçalarının dikkatle incelenmesi gerektiğine işaret etmektedir.

SonarQube'un sunduğu bu ayrıntılı analizler, yalnızca mevcut hataların tespiti açısından değil, aynı zamanda geliştirici becerilerinin gelişimi ve kod yazma alışkanlıklarının iyileştirilmesi açısından da önemlidir. Proje kapsamında bu sistem, CI/CD süreçlerine entegre edilerek kod kalitesinin otomatik ve sürekli izlenmesi sağlanmıştır. Bu yaklaşım, yazılımın sürdürülebilirliğini artırmış, bakım maliyetlerini düşürmüş ve güvenli geliştirme prensiplerinin benimsenmesine katkı sağlamıştır.



Şekil 5.7. Sonarqube kullanımı

5.5. İçerik Yönetimi: Rook-Ceph ile Dağıtık Depolama

Modern yazılım altyapılarında veri yönetimi ve kalıcı depolama çözümleri, özellikle Kubernetes tabanlı sistemlerde büyük önem taşımaktadır. Bu kapsamda, proje mimarisine entegre edilen Rook-Ceph, dağıtık ve esnek bir depolama altyapısı sunarak içerik yönetiminin güvenli, ölçeklenebilir ve yüksek erişilebilirlik ilkesine uygun şekilde gerçekleştirilmesini sağlamaktadır.

Ceph, nesne depolama, blok depolama ve dosya sistemi hizmetlerini aynı platformda bir araya getirerek çok katmanlı bir veri yönetimi çözümü sunar. Veriler, sistem genelinde dengeli bir şekilde dağıtılır ve çoğaltılır. Bu dağıtım, özel algoritmalar ile gerçekleştirilerek yüksek erişilebilirlik, veri bütünlüğü ve hata toleransı sağlanır.

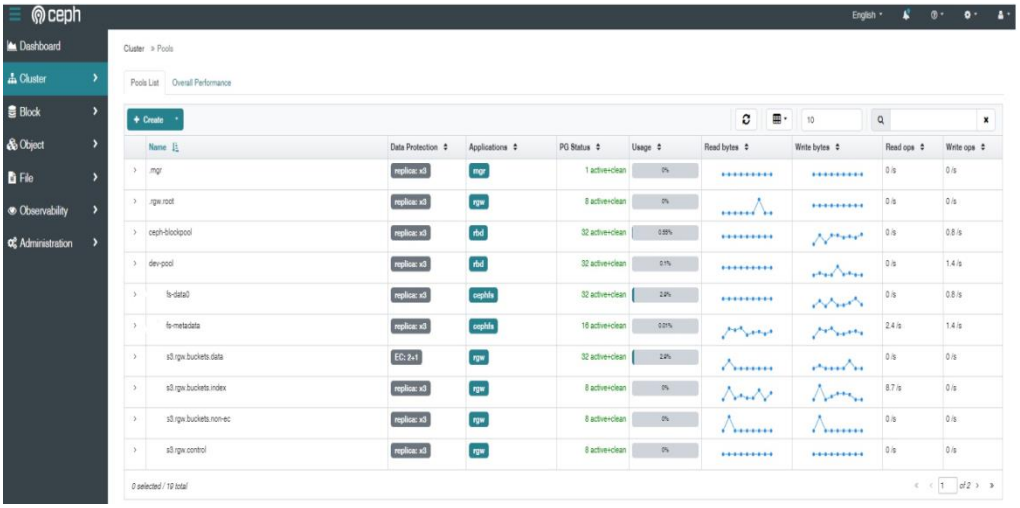
Dinamik ölçeklenebilirlik, Ceph'in öne çıkan yeteneklerinden biridir. Sisteme ihtiyaç duyuldukça yeni düğümler ya da diskler eklenebilir veya mevcut bileşenler çıkarılabilir. Bu özellik, uygulama trafiğinin arttığı durumlarda kaynakların otomatik olarak ölçeklenmesini ve depolama kapasitesinin ihtiyaca göre şekillendirilmesini mümkün kılar.

Veri güvenliği sistemin merkezinde konumlandırılmıştır. Ceph, verileri farklı fiziksel konumlara yayarak yüksek yedeklilik sağlar ve veri parçalarının bütünlüğünü düzenli olarak kontrol eder. Bu sayede, kritik verilerin kaybı en aza indirilir.

Kubernetes ortamı ile entegre çalışan Rook Operatörü, bu karmaşık altyapının yönetimini basitleştirir. Operatör, Ceph kaynaklarının Kubernetes içinden yönetilmesini sağlar ve sistem kaynakları ile Ceph depolama bileşenleri arasında doğrudan bir köprü kurar. Uygulamalar, Kalan Hacim (PV) ve Sürekli Talep Hacmi (PVC) kaynakları üzerinden Ceph depolamasına erişebilir, böylece konteyner tabanlı sistemlerde verilerin kalıcı olarak saklanması sağlanır.

Şekil 5.8'de Ceph arayüzü görülmektedir. Burada yer alan "Pools" ekranı, sistem yöneticilerine her havuzun durumu hakkında hızlı bilgiler sunar: Her bir havuzda verinin hangi kodlama yöntemiyle saklandığı, doluluk oranı ve o havuzdaki okuma/yazma işlemleri raporlanmaktadır. Bu ekran, sistem performansını ve güvenliğini izlemek için temel bir araçtır.

Proje kapsamında Rook-Ceph sistemi, içerik yönetimi altyapısı olarak konumlandırılmış ve uygulamalarda kalıcı veri ihtiyacını karşılamak üzere yapılandırılmıştır. Sistem, yüksek performans, veri güvenliği ve operasyonel esneklik sağlayarak DevOps süreçlerinin sürdürülebilirliğine katkıda bulunmaktadır.



Şekil 5.8. Ceph kullanımı

5.6. İzleme ve Kayıt Yönetimi

5.6.1. Prometheus

Günümüzün karmaşık ve dağıtık sistem altyapılarında, sistem sağlığının izlenmesi ve performans göstergelerinin sürekli takibi kritik bir gereklilik haline gelmiştir. Bu ihtiyaca yönelik olarak geliştirilen Prometheus, Cloud Native Computing Foundation (CNCF) çatısı altında sürdürülen, açık kaynak kodlu ve yüksek performanslı bir izleme ve uyarı sistemidir. Prometheus, zaman serisi veri modeliyle çalışmakta olup, sistemlerden topladığı metrik verileri analiz ederek yazılımın davranışlarını görünür kılmayı amaçlamaktadır.

Sistemin temel çalışma mantığı, izlenmesi hedeflenen bileşenlerden HTTP protokolü aracılığıyla belirli aralıklarla veri çekilmesine dayanır. Özellikle geçici ve kısa ömürlü süreçlerden metrik toplamak amacıyla geliştirilen Push Gateway bileşeni, bu tür verilerin sisteme entegre edilmesini mümkün kılar. Elde edilen tüm metrik veriler, Prometheus tarafından lokal olarak depolanmakta ve bu veriler üzerinde gelişmiş sorgulamalar gerçekleştirilebilmektedir.

Prometheus'un öne çıkan özelliklerinden biri, verileri etiket-değer çiftleri şeklinde saklamasıdır. Bu esnek veri modeli sayesinde, metriklerin çok boyutlu olarak kategorize edilmesi ve detaylı analizlerin yapılması mümkün hale gelir. Prometheus'un özgün sorgulama dili olan PromQL (Prometheus Query Language), bu metrik verileri üzerinde güçlü ve özelleştirilebilir sorgular gerçekleştirilmesine imkân tanır.

Uygulama seviyesinde detaylı izleme sağlayabilmek için Prometheus, yaygın programlama dillerine yönelik istemci kütüphaneler sunmaktadır. Java, Go, Python ve Ruby gibi diller için geliştirilen bu istemciler, uygulamalardan Prometheus formatında metrik verilerin sağlanmasına olanak tanır. Böylece geliştiriciler, uygulama içi davranışları detaylı biçimde gözlemleyebilir ve performans iyileştirmeleri için gereken içgörüleri edinebilir.

Uyarı yönetimi, Prometheus ekosistemindeki Alertmanager bileşeni tarafından yürütülmektedir. Alertmanager; uyarıların gruplanması, tekrarlayan bildirimlerin filtrelenmesi ve kritik olayların farklı iletişim kanallarına (örneğin e-posta, PagerDuty, OpsGenie vb.) yönlendirilmesi gibi işlevleri üstlenir. Bu yapı sayesinde, yalnızca sistem durumu hakkında bilgi vermekle kalmaz; aynı zamanda olaylara hızlı müdahale edilebilmesi için kapsamlı bir uyarı altyapısı sağlar.

Sonuç olarak, Prometheus; sistem kaynaklarının izlenmesi, performans takibi, olay analizi ve olaylara müdahale süreçlerinin otomasyonu açısından DevOps odaklı mimarilerde vazgeçilmez bir bileşen haline gelmiştir.

Şekil 5.9'da görüldüğü üzere, Grafana'nın *Explore* ekranında son bir saatlik zaman dilimi incelendiğinde, özellikle node_memory_MemAvailable_bytes metriğinin yaklaşık 227 GiB seviyesinden başlayarak 225 GiB düzeyine kadar düştüğü ve ardından kısmi bir toparlanma eğilimi sergilediği gözlemlenmektedir. Bu durum, sunucunun muhtemelen artan işlem yükü veya önbellek kullanımı nedeniyle geçici bir bellek tüketimi artışı yaşadığını, ancak kritik eşiklere ulaşmadığını göstermektedir.

Aynı zaman dilimi içerisinde ALERTS grafiğinde sabit "1" değeri dikkat çekmektedir; bu da aktif durumda bir uyarının hâlen çözülmediğini ve sistemin bu durumu izlemeye devam ettiğini göstermektedir. Diğer yandan aggregator_discovery_aggregation_count_total metriği, dakikada yaklaşık 0.06 işlem/saniye hızında düzenli bir şekilde artarak Prometheus'un API servis keşfi işlemlerini kesintisiz ve kararlı bir biçimde gerçekleştirdiğini ortaya koymaktadır.

Şekil 5.9. Prometheusun kullanımı

aggregator_unavailable_apiservice metriğinin sıfır değeri ise, sistemde hiçbir API servisinin "unavailable" (erişilemez) duruma geçmediğini ve Kubernetes içi servis iletişiminin sağlıklı bir biçimde sürdüğünü teyit etmektedir. Ayrıca alertmanager_alerts metriğinde gözlenen sabit grafik çizgisi, Alertmanager bileşenine iletilen uyarı sayısında herhangi bir değişiklik olmadığını, dolayısıyla uyarı entegrasyon hattının beklenen şekilde çalıştığını göstermektedir.

Bu gözlemler ışığında sistem performansında belirgin bir darboğazın oluşmadığı, ancak artan bellek kullanımı ve çözülmemiş uyarının dikkatle izlenmesi gerektiği anlaşılmaktadır.

Gerekli durumlarda, kaynak tahsisi yeniden değerlendirilerek müdahale edilmesi önerilmektedir.

Prometheus'un Kubernetes ortamlarıyla doğal entegrasyonu, konteyner tabanlı mimarilerde sistem izleme süreçlerini oldukça etkili kılmaktadır. Ancak yalnızca altyapı seviyesinde değil, uygulama düzeyinde de ayrıntılı izleme sağlanabilmesi için ilgili programlama dili kütüphanelerinin entegrasyonu gerekmektedir.

Genel olarak değerlendirildiğinde, Prometheus; mikroservis mimarileri ve bulut tabanlı sistemler gibi modern yazılım altyapılarının karmaşık izleme gereksinimlerine karşılık verebilecek güvenilir, ölçeklenebilir ve sürdürülebilir bir çözüm sunmaktadır. Bu bağlamda, proje kapsamında Prometheus; sistem performansının izlenmesi ve uyarı yönetiminin otomasyonu amacıyla DevOps altyapısına entegre edilmiştir.

5.6.2. Prometheus Ekosistemi: Grafana, Alertmanager, Loki ve Jaeger Entegrasyonu

Prometheus'un güçlü metrik toplama altyapısı, diğer açık kaynaklı araçlarla birlikte kullanıldığında daha kapsamlı bir gözlem mimarisi oluşturmaktadır. Bu bağlamda Grafana, Alertmanager, Loki ve Jaeger gibi sistemlerle olan entegrasyonlar, izleme, uyarı, kayıt yönetimi ve iz sürme gibi farklı katmanlarda tam kapsamlı bir görünürlük sağlar.

Grafana ile Görselleştirme

Prometheus'un topladığı metrik veriler, görselleştirme ve kullanıcıya anlamlı dashboard'lar sunma açısından tek başına yeterli değildir. Bu ihtiyacı karşılamak üzere kullanılan Grafana, zaman serisi verileri interaktif panolar aracılığıyla sunan güçlü bir arayüz sağlar. Kullanıcılar, PromQL ile yazılan sorgularla özel grafikler oluşturabilir, sistem performansını gerçek zamanlı olarak izleyebilir ve kritik eşik değerleri temelinde uyarılar tanımlayabilir. Proje kapsamında Grafana, hem operasyonel metriklerin izlenmesi hem de karar destek mekanizmalarının görsel olarak desteklenmesi amacıyla Prometheus ile birlikte entegre biçimde yapılandırılmıştır.

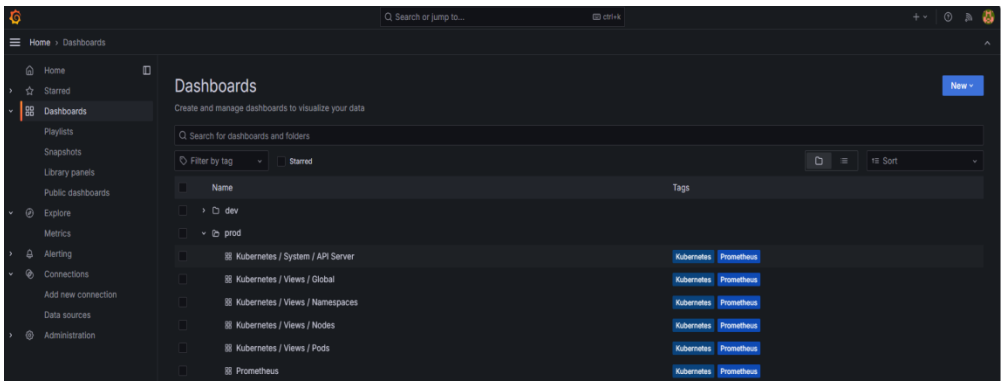
"Grafana, dağıtık sistemlerde gerçek zamanlı verileri görselleştirmek ve analiz etmek için yaygın olarak kullanılan açık kaynak bir platformdur. Kullanıcıların çeşitli veri kaynaklarından (örneğin veritabanları, sunucular, bulut hizmetleri) gelen verileri dashboard'lar aracılığıyla izleyip potansiyel sorunları kolayca teşhis etmelerini sağlar… Desteklediği veri kaynakları arasında InfluxDB, Prometheus, Graphite ve Elasticsearch yer alır." (Sawant, 2025)

Şekil 5.10'da proje kapsamında kullanılan Grafana panosunun örnek gösterimi yer almaktadır. Panelde, Prometheus'tan toplanan metriklerin görsel olarak sunumu, sistem kaynak kullanımı ve uyarı durumları takip edilebilmektedir.

Grafana, açık kaynaklı ve web tabanlı bir veri görselleştirme aracıdır. Temel olarak zaman serisi verilerini net, anlaşılır grafikler ve tablolar haline getirmeye olanak tanır. Farklı türdeki veri kaynaklarını bir araya getirerek kullanıcıların tek bir panel üzerinde verileri analiz etmelerini kolaylaştırır.

Grafana, küçük çaplı uygulamalardan geniş kurumsal sistemlere kadar farklı büyüklüklerdeki projelerde verimli bir biçimde kullanılabilir. Kullanıcıların verileri daha iyi yorumlaması ve anlamlandırması için çok sayıda grafik, tablo ve çizelge türünü desteklemektedir. Ayrıca, kullanıcıların kendi ihtiyaçlarına uygun özel paneller hazırlaması mümkündür. Bunun yanında, açık kaynak topluluğu tarafından önceden oluşturulmuş hazır paneller de kullanılabilir.

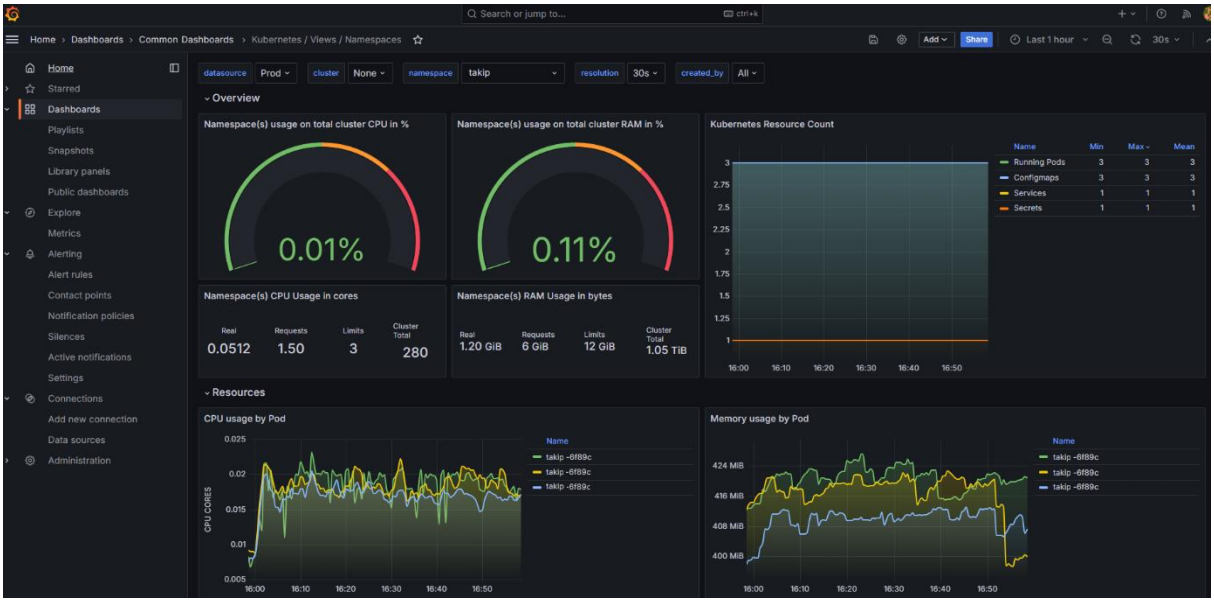
Grafana'nın en güçlü yönlerinden biri esnekliği ve genişletilebilir yapısıdır. Birçok farklı eklenti ve modül ile işlevleri zenginleştirilebilir. Kullanıcı yönetimi konusunda da gelişmiş yeteneklere sahiptir. Farklı kullanıcılar ve kullanıcı grupları oluşturularak, bu kullanıcılara ayrı ayrı yetkilendirmeler yapılabilir ve panellere erişim seviyeleri düzenlenebilir.



Şekil 5.10. Grafana Dashboard yapısı

Şekil 5.11'de görüldüğü üzere, ilgili pod mevcut durumda yapılandırılmış kaynak sınırlarının oldukça altında bir yükle çalışmaktadır. CPU ve bellek talepleri ile limitleri arasında geniş bir esneklik aralığı tanımlanmış olup, herhangi bir kaynak yetersizliği ya da kısıtlama/durdurma durumu gözlemlenmemektedir. Bu durum, pod'un istikrarlı ve verimli bir biçimde çalıştığını ortaya koymaktadır.

Bu proje kapsamında Grafana; sistem kayıtlarının analizi, performans göstergelerinin takibi ve farklı veri kaynaklarından elde edilen metriklerin görsel olarak sunulması amacıyla Prometheus ile birlikte entegre edilerek etkin biçimde kullanılmıştır.



Şekil 5.11. Uygulamanın Grafanada gözlenmesi

Alertmanager ile Uyarı Yönetimi

Prometheus tarafından tespit edilen kritik durumlar, Alertmanager bileşeni aracılığıyla yönetilir. Alertmanager, benzer nitelikteki alarmları gruplar, tekrar eden bildirimleri bastırır ve belirli kurallara göre yönlendirerek uç noktalara (örneğin e-posta, Slack, OpsGenie, PagerDuty) iletir. Böylelikle sistem yöneticileri, yalnızca anlamlı ve eyleme geçirilebilir uyarılarla muhatap olur. Bu bileşenin yapılandırılması, izleme sisteminin proaktif olmasını sağlar ve sistem sağlığına ilişkin olayların daha kontrollü bir şekilde ele alınmasına olanak tanır.

Loki ile Kayıt Yönetimi

Prometheus metrik verilerini toplarken kayıt verilerine doğrudan erişim sağlamaz. Bu boşluğu doldurmak amacıyla geliştirilen Loki, kayıt verilerinin indekslenmesini değil, etiketlenmiş olarak saklanmasını esas alan modern bir kayıt yönetim sistemidir. Loki, Prometheus ile aynı etiketleme mantığını kullandığından, metriklerle kayıt verileri arasında doğrudan ilişkilendirme yapılabilir. Bu sayede bir performans düşüşü veya uyarı durumunda, zaman damgası üzerinden ilgili kayıtlara doğrudan erişilerek kök neden analizleri kolaylaştırılır. Projede Loki, sistem davranışlarının kayıt bazlı analizi ve olay korelasyonu için Prometheus ekosistemi içinde kullanılmıştır.

Günümüz dijital altyapılarında kayıt yönetimi için geliştirilen Loki, verimli kaynak kullanımı ve ölçeklenebilir yapısıyla öne çıkan modern bir kayıt toplama çözümüdür. Geleneksel sistemlerden farklı olarak, kayıt içeriklerinin tamamını değil, yalnızca tanımlayıcı etiketleri dizinleyerek yenilikçi bir yaklaşım sunar.

Loki'nin benzersiz mimarisi, kayıt verilerini sıkıştırılmış biçimde saklarken, yalnızca etiket bilgilerini dizinler. Bu akıllı tasarım, sistem kaynaklarının optimize kullanımını sağlarken, depolama maliyetlerini önemli ölçüde düşürür. Veriler, yerel disk alanında saklanabildiği gibi, bulut tabanlı depolama hizmetlerinde de (örneğin AWS S3, GCS gibi) tutulabilir.

Sistemin omurgası üç temel bileşenden oluşur: veri toplama ajanı, merkezi işlem birimi ve görselleştirme arayüzü. Veri toplama aşamasında özelleştirilmiş bir ajan yazılımı (örneğin Promtail) kullanılır. Bu ajan, sistemlerden kayıt verilerini toplar, etiketler ekler ve düzenli akışlar halinde merkezi sunucuya iletir.

Veri toplama işlemlerinde tercih edilen bu özel ajan yazılımı, hedef sistemleri otomatik keşfetme özelliğine sahiptir. Konteyner ortamlarında yan uygulama olarak ya da sistem genelinde servis şeklinde çalışabilir. Toplanan veriler üzerinde filtreleme, dönüştürme ve etiketleme işlemleri gerçekleştirir.

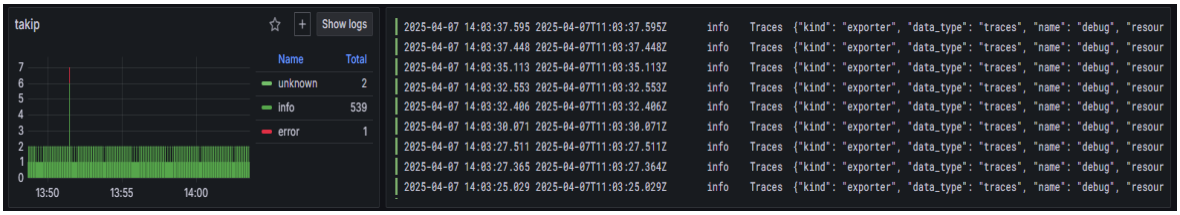
Merkezi işlem birimi, kayıt verilerinin alınması, işlenmesi ve sorgulanmasından sorumludur. Geleneksel yaklaşımların aksine, kayıt içeriklerini indekslemek yerine, akış bazlı gruplama ve etiket dizinleme stratejisi kullanır. Bu yöntem, kayıtlara neredeyse anlık erişim sağlarken, sistem kaynaklarını verimli kullanır.

Depolama altyapısı, zaman serisi veritabanı prensiplerine dayanır. Veriler, küçük parçalar halinde organize edilir ve seçilen depolama sisteminde saklanır. Bu yapı, sistemin yatay büyümesine ve kesintisiz hizmet sunmasına olanak tanır.

Kayıtların görüntülenmesi ve analizi için genellikle Grafana arayüzü kullanılır. Bu görsel arayüz sayesinde kullanıcılar, kayıt verilerini zaman aralıklarına ve etiketlere göre sorgulayabilir, analiz edebilir ve gerekli içgörüleri elde edebilir. Loki’nin Prometheus ile birlikte çalışabilirliği sayesinde kayıtlar ve metrikler bütüncül şekilde değerlendirilebilir, bu da sistem davranışlarının daha etkin analizini mümkün kılar.

Şekil 5.12’de görüldüğü gibi kayıtlar, sisteminizin izleme bileşeninin debug seviyesindeki trace verilerini düzenli olarak işlediğini ve çoğunlukla sorunsuz çalıştığını göstermektedir. Toplam 542 kayıttan’dan 539’u “info” seviyesinde; bu da veri akışının

kesintisiz sürdüğünü ve her bir trace paketi hakkında ayrıntılı bilgilendirme yapıldığını ortaya koymaktadır. Sadece tek bir "error" kaydı, potansiyel bir aksaklığı işaret etmektedir. Kayıtların çok kısa aralıklarla (milisaniye mertebesinde) oluşturulması, süreç hattı'ın yüksek hacimli veri işlediğini göstermektedir. Bu teknoloji, özellikle büyük ölçekli sistemlerde kayıt yönetimini basitleştiren, kaynak dostu ve maliyet etkin bir çözüm sunmaktadır. Modern uygulama altyapılarının karmaşık kayıt yönetimi gereksinimlerini karşılamak üzere tasarlanmıştır. Proje kapsamında Loki, kayıtların toplanması ve analiz edilmesi amacıyla Prometheus ekosistemi içinde başarıyla kullanılmıştır.



Şekil 5.12. Uygulamanın kayıtlarının görüntülenmesi

5.6.3. Jaeger ile Dağıtık İzleme ve Performans Takibi

Mikroservis mimarilerinde, kullanıcı isteklerinin sistem boyunca nasıl aktığını anlamak için yalnızca metrik ve kayıt verileri yeterli olmamaktadır. Bu noktada devreye giren Jaeger, açık kaynaklı ve yüksek performanslı bir uygulama izleme çözümüdür. Özellikle dağıtık sistemlerde isteklerin uçtan uca takibini sağlar. Her mikroservisin işleme süresi, gecikme noktaları ve çağrı zincirleri ayrıntılı biçimde analiz edilerek performans darboğazları tespit edilebilir.

Mikroservis sistemlerinin karmaşıklığı, izleme ve test süreçlerinde özel çözümler gerektirir (Waseem vd., 2021).

Jaeger, yalnızca işlem sürelerini izlemekle kalmaz; aynı zamanda hata teşhisi, servisler arası bağımlılıkların analizi ve kök neden analizinin yapılmasına da olanak tanır. Tracing yaklaşımı sayesinde, herhangi bir kullanıcı talebinin başlangıcından sonlanmasına kadar geçen süreç detaylandırılabilir; işlem süresi, gecikme noktaları ve sorun yaşanan adımlar net biçimde ortaya konabilir.

Grafana ve Prometheus gibi izleme araçlarıyla entegre çalışabilen Jaeger, gözlemlenebilirliği artıran bütüncül bir izleme ekosistemi sunar. Prometheus ile elde edilen metriklerin, Jaeger aracılığıyla işlem bazlı izlenmesi; performans analizlerinde derinlemesine içgörülerin elde edilmesini mümkün kılar.

Bu proje kapsamında Jaeger, özellikle kullanıcı deneyimini etkileyen gecikmelerin tespiti ve servisler arası ilişkilerin görselleştirilmesi amacıyla devreye alınmıştır. Proje mimarisi içerisinde dağıtık izleme çözümü olarak konumlanmış; sistem genelindeki işlemlerin akışını ayrıntılı biçimde ortaya koyarak operasyonel karar süreçlerine katkı sağlamıştır.

5.6.4. Sentry ile Alternatif İzleme Yaklaşımı

Jaeger, dağıtık sistemlerde uçtan uca işlem izleme işlevi ile mikroservis çağrıları arasındaki bağıntıları görselleştirme konusunda güçlü bir araçtır. Ancak Sentry, yalnızca dağıtık izleme yapmakla kalmaz; aynı zamanda gerçek zamanlı hata tespiti, istisna yönetimi ve kullanıcı davranışlarının izlenmesi gibi bütüncül bir gözlemleme deneyimi sunar.

Sentry, uygulama içi SDK'ları aracılığıyla hem istemci hem de sunucu tarafında meydana gelen hataları otomatik olarak toplar ve bu hataları issue başlıkları altında gruplandırır. Bu özellik, özellikle tekrar eden istisnaların merkezi bir noktadan yönetilmesini ve takip edilmesini mümkün kılar.

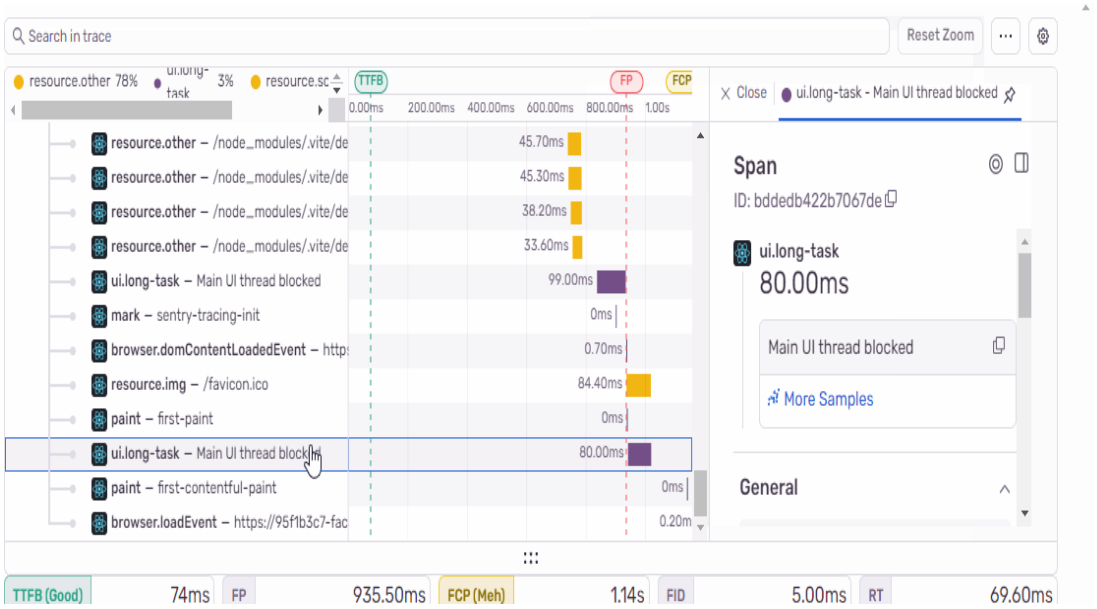
Sentry'nin güçlü taraflarından biri olan İzler özelliği, kullanıcı etkileşimlerini ve sistem davranışlarını zaman sıralı biçimde kaydederek, geliştiricilere hatanın oluşma sürecini tüm detaylarıyla sunar. Kaydedilen bağlam verileri arasında console.log çıktıları, HTTP istekleri, DOM olayları ve kullanıcı tıklamaları yer alır. Bu yaklaşım, hata kaynaklarına ulaşmada klasik kayıt analizlerine kıyasla çok daha yüksek doğruluk ve hız sağlar.

Ayrıca Sentry, performans izleme kapsamında span sayısı, gecikme süreleri, işlem zincirleri ve özel tanımlı metrikleri grafiksel olarak sunar. Geliştiriciler, özellikle kullanıcı deneyimini etkileyen yavaş yüklemeler, UI tıkanmaları ve JavaScript darboğazlarını ayrıntılı olarak analiz edebilir.

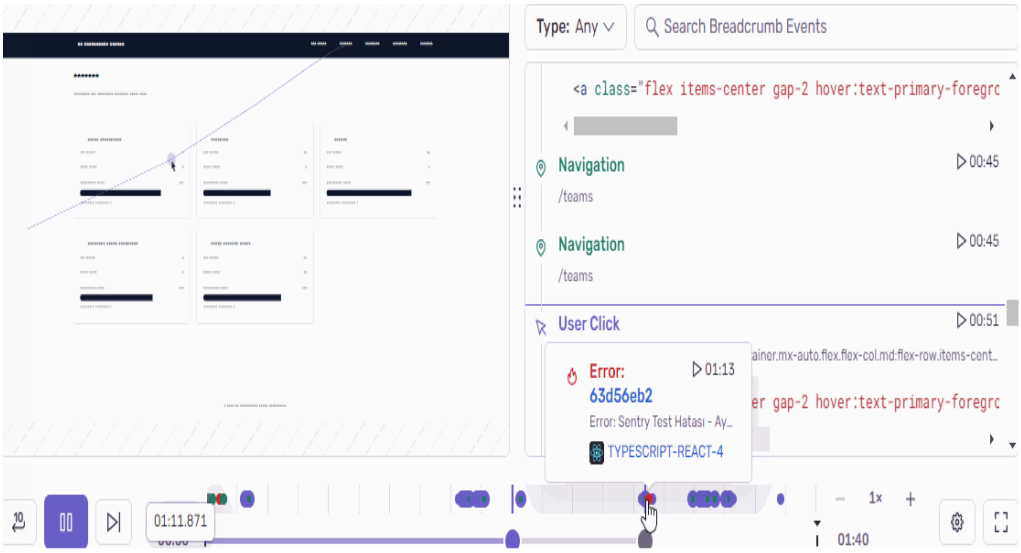
Şekil 5.13'de verilen Sentry arayüzünde kullanıcı arayüzü performansı örneği görülmektedir Bu ekranda, tarayıcının ana iş parçacığının "ui.long-task" olarak işaretlenen 80 ms'lik bir işlem nedeniyle meşgul olduğu ve bu durumun sayfanın boyama ve yüklenme sürecini geciktirdiği görülmektedir. Bu tür ayrıntılar, performans darboğazlarının görsel olarak izlenmesini ve hedefe yönelik optimizasyon yapılmasını mümkün kılar.

Sentry'nin öne çıkan özelliklerinden biri de Oturum Tekrarı yeteneğidir. Kullanıcı oturumlarının video benzeri bir akış halinde oynatılmasına olanak sağlayan bu özellik sayesinde, hataların oluştuğu andaki kullanıcı etkileşimleri ayrıntılı olarak izlenebilir. Şekil 5.14'de görüldüğü gibi, bu oturum kaydı özelliğiyle kullanıcı hareketleri yeniden

görüntülenebilir; böylece hatanın nedeni doğrudan gerçek kullanıcı deneyimi üzerinden analiz edilebilir.



Şekil 5.13. Sentry performans izleme paneli



5.14. Sentry video gibi hata görüntüleme özelliği

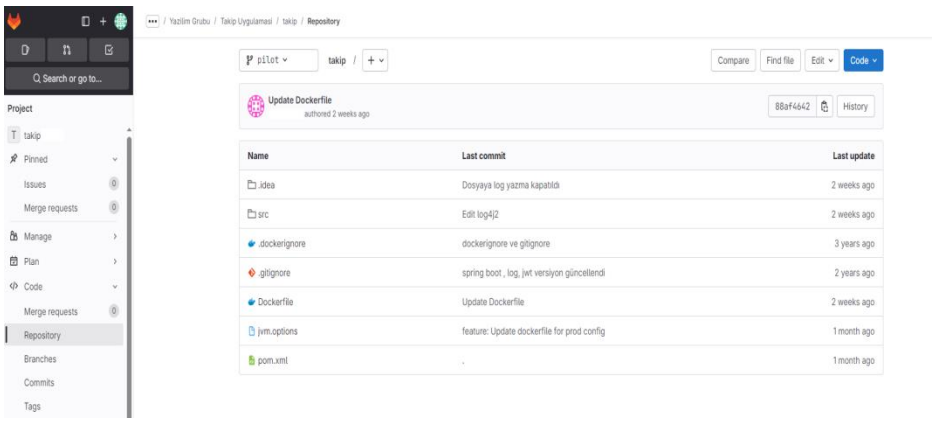
Ayrıca, Sentry'nin Slack, Jira ve e-posta gibi araçlarla entegrasyonu sayesinde, tespit edilen kritik hatalar otomatik olarak ilgili ekip üyelerine yönlendirilerek çözüm süreci hızlandırılır. Bu özellikler, Sentry'yi yalnızca bir tracing aracı olmaktan çıkarıp, aynı zamanda uygulama güvenilirliği, performans analizi ve hata yönetimi platformu haline getirir.

Her ne kadar bu tez kapsamında incelenen kurumsal uygulamada henüz Sentry entegrasyonu yapılmamış olsa da, yürütülen AR-GE çalışmaları çerçevesinde Sentry'nin performansı test edilmiş ve özellikle hataların erken tespiti ve kullanıcı etkileşimlerinin

bağlamsal analizi açısından önemli avantajlar sunduğu gözlemlenmiştir. Proje bazlı yetki tanımlamalarının yapılabilmesi ve kurumsal giriş sistemleri (örneğin LDAP) entegrasyonun sağlanabilmesi sayesinde; yazılımcılara projelerinin davranışlarını doğrudan gözlemleme imkânı tanınmakta, bu da DevOps kültürünün benimsenmesini ve yaygınlaşmasını hızlandırmaktadır.Bu bağlamda Sentry, DevOps mimarisi içinde Jaeger'a alternatif veya tamamlayıcı bir izleme çözümü olarak değerlendirilebilir.

6. KAYNAK KOD YÖNETİMİ VE SÜREÇ ENTEGRASYONU

GitLab, yazılım geliştirme süreçlerinde kod yönetimi, sürüm kontrolü ve ekip işbirliği için etkin bir platform sunar. Proje kapsamında, uygulamanın kaynak kodları GitLab üzerinde yönetilmekte, sürümler buradan takip edilmekte ve konteynerizasyon süreçleri bu yapı üzerinden sürdürülmektedir. Şekil 6.1'de görüldüğü gibi, uygulamanın kök dizininde pom.xml, Dockerfile, .gitignore gibi kritik yapı dosyaları yer almakta; ayrıca kaynak kodun bulunduğu src klasörü ve yapılandırma ayarlarını içeren jvm.options dosyası da dizin yapısında açıkça görülmektedir. Bu yapı, uygulamanın Maven tabanlı derlenmesini ve Docker imajına dönüştürülmesini mümkün kılmaktadır.



Şekil 6.1 Projede Gitlab kullanımı

6.1. GitLab ile Sürüm Kontrolü

Mevcut yazılım projesinde, modern yazılım geliştirme ve dağıtım süreçlerinin etkinliğini artırmak amacıyla kapsamlı bir dönüşüm planı yürütülmektedir. Bu kapsamda, sistem mimarisi; ön yüzde Nginx tabanlı Docker konteyner, arka planda ise Open Liberty uygulama sunucusu kullanılarak yapılandırılmıştır. Bu yapı, hem modülerlik hem de konteyner tabanlı taşınabilirlik açısından modern yazılım mimarisi prensiplerine uygun olarak tasarlanmıştır.

6.2. Kod Yönetimi, Bağımlılık Yapısı ve Konteynerizasyon

Yazılım geliştirme sürecinde kod yönetimi ve sürüm kontrolü amacıyla GitLab tercih edilmiştir. GitLab'ın sunduğu merkezi kod deposu, ekipler arası iş birliğini kolaylaştırmakta; kod değişikliklerinin takibi, dallanma stratejilerinin yönetimi ve kod inceleme süreçleri bu platform üzerinden verimli biçimde yürütülmektedir. Bu yapı, kod kalitesinin sürekli kontrolünü ve geliştirme sürecinin standardizasyonunu sağlamaktadır.

Uygulamanın ön yüz bileşeni, Nginx web sunucusunu içeren bir Docker konteyneri içinde çalıştırılmaktadır. Nginx, yüksek performanslı HTTP işleme kabiliyeti ve etkili yük dengeleme özellikleriyle tercih edilmiştir. Docker konteynerizasyonu sayesinde, uygulama geliştirme, test ve üretim ortamlarında tutarlılık, taşınabilirlik ve ölçeklenebilirlik kazanmaktadır.

Arka planda, Java EE tabanlı uygulamaların çalıştırılması için Open Liberty uygulama sunucusu tercih edilmiştir. Open Liberty'nin seçiminde etkili olan başlıca faktörler; hafif mimarisi, hızlı başlatma süreleri ve mikroservis mimarileriyle uyumlu esnek yapısıdır. Geleneksel Tomcat altyapısı yerine Open Liberty'nin kullanılması, bulut-yerel uygulamalar için daha modern ve sürdürülebilir bir ortam sunmaktadır. Ayrıca, modüler yapısı sayesinde yalnızca gerekli bileşenlerin yüklenmesiyle sistem kaynakları daha verimli kullanılmakta, uygulama performansı optimize edilmektedir.

Proje bağımlılıklarının yönetimi, açık kaynaklı bir araç olan Apache Maven aracılığıyla yürütülmektedir. Maven, XML tabanlı konfigürasyon sistemi sayesinde projenin ihtiyaç duyduğu kütüphaneleri otomatik olarak indirip yönetebilmekte; böylece bağımlılık yönetimi, derleme ve paketleme süreçleri standartlaştırılmış bir yapı içerisinde gerçekleşmektedir. Bu yapı, geliştirici ekiplerin manuel müdahaleye gerek kalmaksızın hızlı ve tutarlı geliştirme süreçleri yürütmesini sağlamakta, projenin bakım ve sürdürülebilirliğine önemli katkı sunmaktadır.

6.3. CI/CD Süreçleri ve Geliştirme Ortamı

CI/CD süreçleri, Gitlab webhook yapısı kullanılarak tamamen otomatize edilmiştir. Bu yapı sayesinde, geliştiriciler tarafından yapılan kod değişiklikleri, sürüm kontrol sistemine aktarıldığı anda otomatik olarak test edilmekte, derlenmekte ve ilgili hedef ortamlara dağıtılmaktadır. CI pipeline hatları, aşağıdaki aşamaları içerecek şekilde yapılandırılmıştır:

- Kod kalitesi kontrolü
- Statik analiz ve güvenlik taramaları
- Birim ve entegrasyon testleri
- Otomatik dağıtım ve imaj üretimi

Geliştirme ortamı olarak, geliştiriciler Eclipse, IntelliJ IDEA gibi modern entegre geliştirme ortamlarını (IDE) kullanmaktadır. Bu IDE’ler, kod yazımı, hata ayıklama, derleme ve versiyon kontrol sistemleriyle entegrasyon açısından büyük kolaylık sağlamaktadır.

Bu bütünleşik yapılandırma sayesinde yazılım geliştirme süreçleri çok daha verimli, güvenilir ve ölçeklenebilir hâle getirilmiş; manuel işlemlerden kaynaklanan hatalar minimize edilmiş ve ekip içi iş birliği ile teslimat süreleri önemli ölçüde iyileştirilmiştir.

6.3.1. Tekton ile CI/CD Süreçlerinin Yönetimi

Proje kapsamında Tekton, yazılım geliştirme süreçlerinin otomatikleştirilmesi amacıyla CI için kullanılmaktadır. Kaynak kodun derlenmesi, test edilmesi ve dağıtılması gibi temel aşamalar, Tekton tarafından tanımlanan özelleştirilmiş iş akışları aracılığıyla yürütülmektedir. Platformun sunduğu esneklik ve ölçeklenebilirlik, CI/CD süreçlerinin etkinliğini ve güvenilirliğini önemli ölçüde artırmaktadır.

Kubernetes ekosisteminde CI/CD süreçlerini düzenlemek amacıyla geliştirilen Tekton, yenilikçi ve esnek bir çerçeve sunmaktadır. Bu sistem, Kubernetes'in Özel Kaynak Tanımlama (CRD) mimarisi üzerine inşa edilmiştir ve işlem adımlarını bağımsız, tekrarlanabilir ve konteyner tabanlı görevler hâlinde yapılandırarak yönetir.

Tekton'un en önemli avantajı, CI/CD süreç hatlarını tamamen Kubernetes yerel bileşenleri üzerinden tanımlama ve çalıştırma yeteneğidir. YAML tabanlı tanımlamalarla sürüm kontrolü kolaylaşırken, platform bağımsızlığı ve taşınabilirlik elde edilir. Ayrıca GitOps yaklaşımıyla da kolayca bütünleşerek modern yazılım teslimat süreçlerinde etkin bir rol oynar.

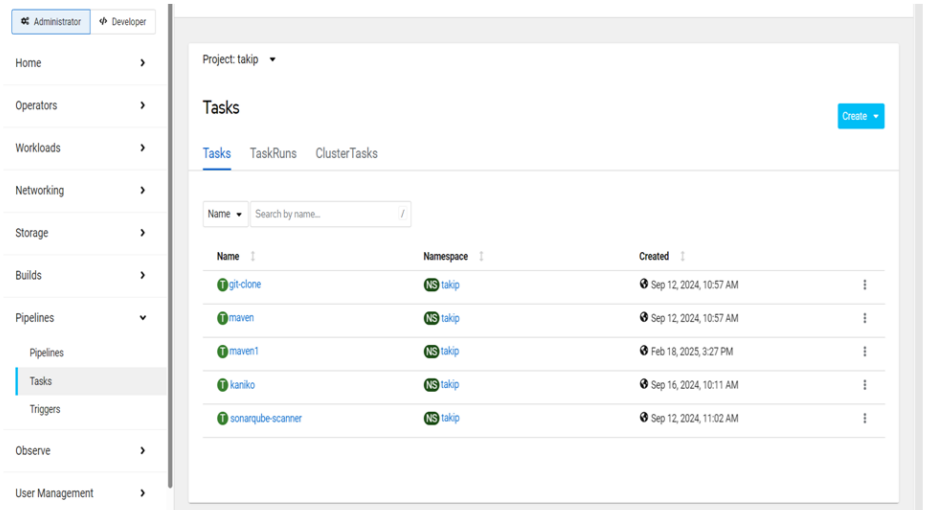
Sistemin temel yapı taşları şu şekilde özetlenebilir:

- **Konteynerleştirilmiş İş Birimleri:** Her bir görev, Kubernetes ortamında bağımsız pod'lar olarak çalıştırılmakta; bu yapı, işlem yalıtımı ve kaynak optimizasyonu sağlamaktadır.
- **Özelleştirilmiş Kaynak Yönetimi:** Süreçler, Kubernetes'in CRD mekanizması üzerinden YAML formatında yapılandırılır. Bu sayede geliştiriciler, CI/CD akışlarını programatik biçimde yönetebilir.
- **Bütünleşik Sistem Desteği:** Tekton; GitHub Actions, Jenkins ve benzeri yaygın CI/CD araçlarıyla entegre çalışarak kurulu sistemlerin dönüşümünü kolaylaştırır.
- **Olay Temelli İş Akışları:** Kod deposundaki değişiklik gibi olaylara bağlı olarak otomatik süreç hattı tetiklemeleri sağlanabilir. Bu yapı, CI/CD süreçlerini daha duyarlı ve reaktif hâle getirir.
- **Yeniden Kullanılabilir Bileşenler:** Tanımlanan görevler modülerdir ve farklı iş akışlarında tekrar kullanılabilir, bu da bakım kolaylığı ve verimlilik sağlar.

- **Güvenlik ve İzleme:** Tekton, Kubernetes'in sunduğu RBAC gibi güvenlik yapılarını kullanarak süreçleri yetkilendirme ve denetim açısından güvenli kılar. Ayrıca izleme araçlarıyla süreçlerin izlenebilirliği sağlanır.

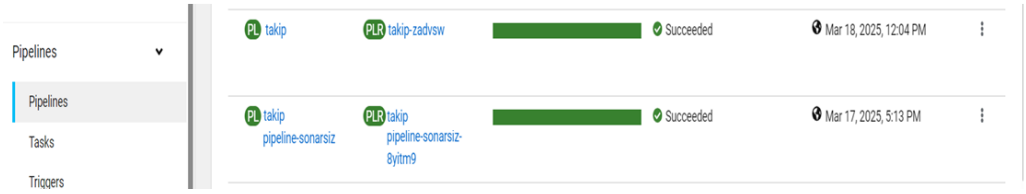
Proje kapsamında Tekton, yazılım geliştirme süreçlerinin otomatikleştirilmesi amacıyla CI aracı olarak konumlandırılmıştır. Bu yapı içerisinde kaynak kodun derlenmesi, test edilmesi ve konteyner imajına dönüştürülmesi gibi adımlar ardışık olarak yürütülmektedir. Tekton'un sunduğu CRD tabanlı yapı sayesinde, bu adımlar bağımsız görevler şeklinde tanımlanarak bir süreç hattı yapısı altında birleştirilmiştir.

Şekil 6.1'de görüldüğü üzere, CI süreci git-clone, maven, kaniko ve sonarqube-scanner gibi görevlerden oluşan bir yapı ile organize edilmiştir. Bu görevler sırasıyla GitLab üzerinden kodların çekilmesi, Maven ile derleme edilerek war dosyasının oluşturulması, Kaniko kullanılarak Docker imajına dönüştürülmesi ve son olarak SonarQube ile kod kalitesinin analiz edilmesi adımlarını kapsamaktadır. Bu yapı ile birlikte, kodlar otomatik olarak işlenmiş, konteyner imajına dönüştürülmüş ve Quay gibi bir imaj deposuna gönderilerek CI süreci tamamlanmıştır.



Şekil 6.1. Tekton Görev yapısı

Şekil 6.2'de ise tamamlanan pipeline süreç hattı çalışmaları ve bu çalışmaların başarılı sonuçlandığı görülmektedir. Bu yapı sayesinde CI süreci manuel müdahaleye gerek kalmaksızın tekrarlanabilir ve izlenebilir bir hale getirilmiştir. Böylece, oluşturulan konteyner imajları CD aşamasına hazır hale getirilmiştir.



Şekil 6.2. Tekton Pipeline Hattı çalışma sonuçları

6.3.2. Merkezi Süreç hattı

Bu merkezi yapı sayesinde, projeye özgü CI/CD süreç hatlarının ayrı ayrı tanımlanmasına duyulan ihtiyaç ortadan kaldırılmakta; yeniden kullanılabilir görev tanımları aracılığıyla yazılım geliştirme süreçlerinin standartlaştırılması sağlanmaktadır. Böylece yazılım yaşam döngüsünde süreklilik, güvenilirlik ve izlenebilirlik önemli ölçüde artırılmaktadır.

Merkezi süreç/pipeline hattı; kaynak kodun alınması, Maven aracılığıyla derlenmesi, SonarQube ile kod kalitesinin analiz edilmesi, Kaniko kullanılarak Docker imajının oluşturulması ve bu imajın container registry'e (örneğin Quay) aktarılması gibi temel adımlardan oluşacak şekilde yapılandırılmıştır. Her bir adım, Tekton süreç hattı içerisinde birer görev olarak tanımlanmış ve bu görevler arasındaki bağımlılıklar belirlenerek sıralı, otomatik bir işlem akışı oluşturulmuştur.

Söz konusu yaklaşım yalnızca manuel işlem yükünü azaltmakla kalmayıp; aynı zamanda insan kaynaklı hata riskini en aza indirerek kod kalitesi denetimlerini sistematik hâle getirmekte ve yazılım güvenliği ile sürdürülebilirliğini desteklemektedir. Ayrıca merkezi pipeline hattı yapısı, kurumsal düzeyde CI/CD standartlarının tanımlanmasına ve farklı projeler arasında yapısal tutarlılığın sağlanmasına önemli katkı sunmaktadır.

Şekil 6.3'te görüldüğü üzere, CI/CD süreci belirli adımlar çerçevesinde otomatik olarak ilerlemektedir. İlk olarak, GitLab üzerindeki kaynak kodlar sistem tarafından çekilmekte; ardından Maven ile derleme işlemi gerçekleştirilmekte ve proje çıktısı elde edilmektedir. Derlenen kodlar, SonarQube analizine tabi tutularak kod kalitesi değerlendirilmektedir. Analiz sonucunun olumlu olması durumunda, Kaniko aracılığıyla oluşturulan imaj, quay-task üzerinden Quay container registry'ye aktarılmaktadır.

Devamında, cleanup-task çalıştırılarak sistemde geçici kaynakların temizlenmesi sağlanmakta ve Helm Chart güncellenmektedir. GitOps yaklaşımı doğrultusunda yapılandırılmış olan ArgoCD, Helm Chart üzerindeki değişiklikleri algılayarak hedef ortama—bu ortam test veya üretim (prod) olabilir—otomatik dağıtım gerçekleştirmektedir.

Ancak, eğer sonarqube-task kapsamında yapılan kod kalitesi analizinden olumsuz bir sonuç elde edilirse, süreç durdurulmakta ve yeni imajın ortama dağıtımı engellenmektedir. Bu yapı, yalnızca kaliteli ve onaylı kodların canlı sistemlere aktarılmasını garanti altına alarak yazılım güvenliği ve sürdürülebilirliğini desteklemektedir.

Şekil 6.3. Merkezi süreç-pipeline hattı yapısı

6.3.3. Sürekli Teslimat: ArgoCD ve GitOps Yaklaşımı

GitOps yaklaşımını benimseyen ve Kubernetes ortamlarında yazılım dağıtımını yöneten ArgoCD, modern yazılım geliştirme ve teslimat süreçlerinde merkezi bir rol üstlenmektedir. Bu sistem, yazılım dağıtımlarının yönetiminde Git depolarını tek kaynaklı doğruluk noktası olarak benimsemekte ve Kubernetes kümeleriyle sürekli senkronizasyon sağlamaktadır.

ArgoCD'nin çalışma prensibi, Git depolarında saklanan Kubernetes yapılandırma dosyalarını temel alarak bu yapılandırmaları istenen ortama otomatik olarak uygular. YAML formatındaki bu konfigürasyonlar, sistemin hedef ortamla eşleşmesini sağlar. Git üzerinde yapılan her değişiklik, otomatik olarak izlenir ve gerektiğinde Kubernetes kümesine entegre edilir. Böylece dağıtım süreçleri tamamen sürüm kontrollü, geri alınabilir ve tekrarlanabilir hale getirilir.

Temel Bileşenler ve İşlevler

ArgoCD aşağıdaki temel bileşenlerden oluşur:

- Application Controller: Git deposu ile Kubernetes kümesindeki gerçek durum arasındaki farkları sürekli olarak izler. Fark algılandığında, Kubernetes ortamını Git'teki istenen duruma getirir.
- Repository Server: Helm, Kustomize ve benzeri araçlarla tanımlanmış kaynakları Git deposundan okuyarak Kubernetes'e uygulanabilir manifest dosyalarına dönüştürür.

- API Server ve Web UI: Kullanıcıların uygulamaları yönetmesini, durumlarını takip etmesini ve manuel müdahalelerde bulunmasını sağlar.

Bu yapıların sağladığı yeteneklerle sistem, şu işlevleri yerine getirir:

- Merkezi Yapılandırma Yönetimi:
    - Dağıtım yapılandırmaları Git üzerinden kontrol edilir.
    - Kubernetes ortamlarına otomatik dağıtım gerçekleştirilir.
    - Sistem durumu sürekli olarak izlenir ve raporlanır.
- İzleme ve Kontrol Mekanizmaları:
    - Web arayüzü üzerinden görsel takip ve kontrol sağlanır.
    - Komut satırı araçlarıyla derinlemesine yönetim imkanı sunar.
    - Sistem sapmaları anlık olarak tespit edilir ve raporlanır.
- Otomasyon ve Güvenlik:
    - Kod değişiklikleri otomatik olarak ortama yansıtılır.
    - Geri alma işlemleri hızlı ve güvenli şekilde yapılabilir.
    - RBAC ve Git geçmişi sayesinde denetlenebilirlik ve erişim kontrolü sağlanır.

Kurumsal Düzeyde Sağladığı Katkılar

ArgoCD'nin sağladığı başlıca avantajlar şunlardır:

- Süreç İyileştirmeleri:
    - Dağıtım süreçlerinde tutarlılık ve tekrarlanabilirlik sağlanır.
    - Sürekli dağıtım iş akışları otomatize edilir.
    - Ekip verimliliği artırılır.

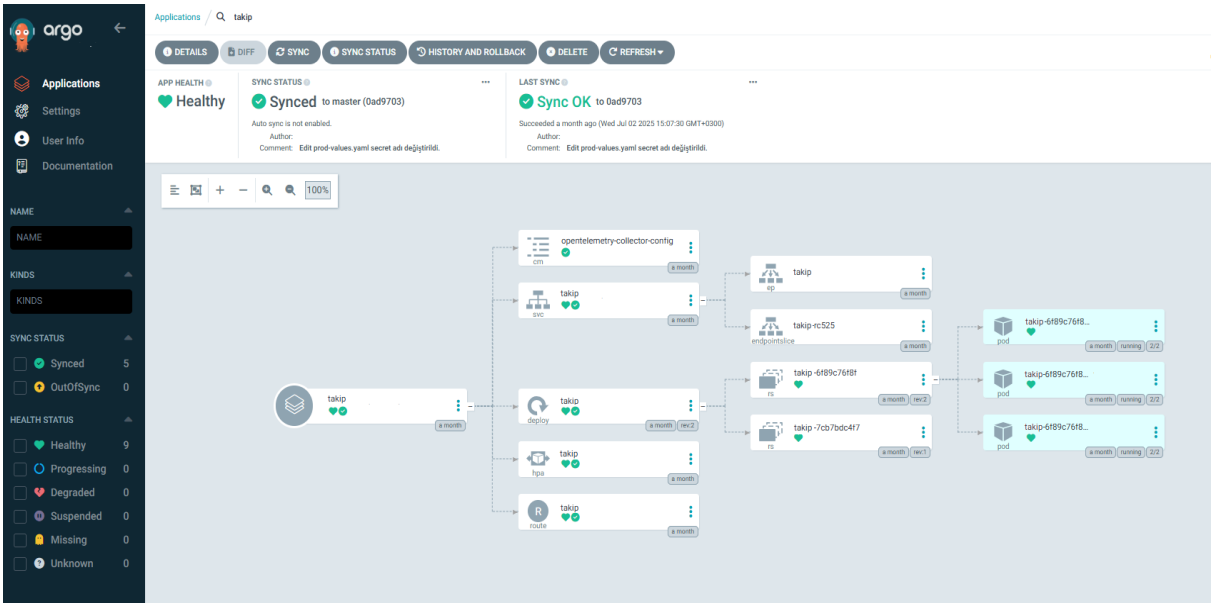
- Kalite ve Güvenilirlik:
    - Kod kalitesi ve sistem kararlılığı korunur.
    - Dağıtımlar standartlara uygun biçimde gerçekleştirilir.
    - Yapılandırma sapmaları minimize edilir.

- İş Birliği ve Şeffaflık:
    - Geliştirici ve operasyon ekipleri arasında entegrasyon artar.
    - Yapılan her değişiklik Git üzerinden izlenebilir kılınır.
    - Süreçler geriye dönük olarak denetlenebilir.

Bu Projedeki Kullanım Senaryosu

Bu proje kapsamında ArgoCD, web arayüzü üzerinden sürekli teslimat süreçlerinin yönetimi için aktif olarak kullanılmaktadır. GitLab üzerinde yapılan her kod değişikliği sonrası güncellenen Helm Chart'lar, ArgoCD tarafından izlenmekte ve değişiklikler tespit edildiğinde hedef Kubernetes ortamına otomatik olarak dağıtım gerçekleştirilmektedir. Bu sayede, dağıtım süreci manuel müdahale olmadan gerçekleşmekte; sistem güvenliği, izlenebilirlik ve operasyonel verimlilik en üst düzeye çıkarılmaktadır.

Şekil 6.4'te Argo CD arayüzü üzerinden gerçekleştirilen GitOps tabanlı sürekli teslimat süreci görselleştirilmiştir. Bu yapı, Argo CD'nin Git deposundaki Helm chart'ları ve GitLab üzerindeki proje repolarını izleyerek yapılandırma değişikliklerini anlık olarak algılayabilmesini sağlamaktadır. Projede herhangi bir değişiklik yapıldığında, Argo CD bu değişikliği kısa sürede tespit etmekte ve tanımlı işlem adımlarının ardından güncellenmiş Docker imajının ilgili Kubernetes ortamına dağıtımını otomatik olarak gerçekleştirmektedir.



Şekil 6.4. ArgoCD ile GitOps Tabanlı Sürekli Teslimat süreci

Görselde, Argo CD üzerinden yönetilen uygulamanın durumunu göstermektedir. Uygulama Healthy (sağlıklı) ve Synced (senkronize) durumdadır, yani Git deposundaki tanımlar ile Kubernetes ortamındaki kaynaklar uyumludur. Hiyerarşik görünümde ConfigMap, Service, Deployment, HPA ve Route gibi Kubernetes kaynakları ile bu kaynaklara bağlı çalışan pod'lar gösterilmektedir. Uygulama şu anda 3 pod üzerinde çalışmakta olup, tüm pod'lar sorunsuz ve yeşil durumdadır. Bu yapı, Argo CD'nin GitOps yaklaşımıyla versiyon kontrolünden Kubernetes ortamına dağıtımı ve kaynak durumunun anlık izlenmesini sağlamaktadır.

7. DEVOPS SÜREÇLERİNDE ALTYAPI OTOMASYONU

Modern yazılım geliştirme ve dağıtım süreçlerinde DevOps uygulamaları, hız, güvenlik ve sürdürülebilirlik açısından önemli kazanımlar sunmaktadır. Bu yaklaşımların merkezinde ise IaC yer almaktadır. Altyapının manuel işlemlerden arındırılması, sistemlerin tekrarlanabilirliğini, izlenebilirliğini ve sürdürülebilirliğini artırmaktadır. Bu bölümde, örnek bir DevOps mimarisi üzerinden Tekton ve GitLab entegrasyonu ile oluşturulan altyapı otomasyonunun detayları sunulmaktadır.

Altyapı otomasyonu, manuel yapılan yapılandırma ve dağıtım işlemlerinin, betikler veya otomasyon araçları ile standart hale getirilerek sistematik olarak yürütülmesini sağlar. Bu kapsamda temel hedefler şunlardır:

- Zaman kazanımı: Tekrarlayan görevlerin otomasyonu, geliştirme süresini önemli ölçüde azaltır.
- Hata riskinin azaltılması: İnsan hatasına dayalı yapılandırma sorunları en aza indirilir.
- Versiyonlanabilirlik: Altyapı tanımları kod olarak saklanarak geçmiş değişiklikler izlenebilir hale gelir.
- Çeviklik: Değişen iş gereksinimlerine hızlı adaptasyon sağlanır.

Bu bağlamda, Tekton ve GitLab gibi araçların entegre kullanımı, bu hedeflere ulaşmak için ideal bir yapı sunmaktadır.

Tekton, Kubernetes ortamları üzerinde çalışan, açık kaynaklı bir CI/CD çözümüdür. Bulut tabanlı mimarilerde çalışabilen, esnek, yeniden kullanılabilir süreç hattı bileşenleriyle dikkat çeken bu sistem, YAML tabanlı tanımlamalar ile görevlerin otomasyonunu sağlar.

GitLab ise kaynak kod yönetiminden, proje planlamaya ve CI/CD süreçlerine kadar birçok özelliği barındıran kapsamlı bir platformdur.

Bu iki sistemin entegrasyonu, webhook mekanizması aracılığıyla sağlanmaktadır. Webhook, belirli bir olay (örneğin merge isteği) gerçekleştiğinde, Tekton pipeline hattı'ını tetiklemek amacıyla GitLab tarafından HTTP tabanlı bir istek gönderen bir mekanizmadır.

Proje kapsamında geliştirilen otomasyon süreci aşağıdaki adımları içermektedir:

1. Kod güncellemesi ve Merge Request: Geliştirici, GitLab üzerinde kod güncellemesini tamamlar ve birleştirme isteği (MR) oluşturur.
2. Webhook tetiklemesi: Merge işlemi sonrası webhook, Tekton süreç hattı'ını başlatır.

3. Süreç hattı görevleri:
    - Gitlab Pull Task: Kodlar çekilir
    - Maven Task: Java projesi derlenir.
    - Sonarqube Task:Kod kalitesi analiz edilir
    - Quay Task: Derlenen uygulama için kaniko ile Docker imajı oluşturulur. Quay'a saklanması için gönderilir.
    - Update Helm Chart Task: Oluşan imaja göre Helm Chart değiştirilir.
4. Container imajının yüklenmesi: Oluşturulan Docker imajı, Quay gibi bir container registry sistemine otomatik olarak aktarılır.
5. Argo CD ile Dağıtım:
    - Argo CD, GitOps yaklaşımıyla container imajındaki güncellemeyi algılar.
    - Helm chart'lar aracılığıyla Kubernetes ortamına parametrik dağıtım yapılır.
    - İlgili servis otomatik olarak güncellenir.

Oluşturulan yapı, iki temel ortama hizmet vermektedir:

- Test Ortamı: Pipeline hattı uçtan uca otomatik çalışmakta, tüm aşamalar herhangi bir insan müdahalesi olmadan tamamlanmaktadır.
- Canlı Ortam: Kritik operasyonel gereklilikler nedeniyle, bazı kontrol noktaları manuel olarak kurgulanmıştır. Bu kapsamda:
    - Merge işlemi sonrası tetikleme kullanıcı onayına bağlanır.
    - Sonuçlar manuel olarak doğrulanır.
    - Dağıtım süreci sorumlu ekip tarafından izlenir ve gerekirse geri alınabilir.

Bu farklılaşma, hem sistem güvenliğini hem de operasyonel sürekliliği sağlamayı hedeflemektedir.

Tekton–GitLab entegrasyonu ile oluşturulan bu otomasyon yapısı, sadece yazılım geliştirme süreçlerini hızlandırmakla kalmamakta; aynı zamanda hata oranlarını azaltmakta, güvenlik açıklarını minimize etmekte ve CI/CD süreçlerinin sürdürülebilirliğini artırmaktadır.

Test ortamlarında tam otomasyon, canlı ortamlarda ise kontrollü süreçlerin benimsenmesi, organizasyonların hem çeviklik hem de denetim gereksinimlerini aynı anda karşılayabilmesini mümkün kılmaktadır.

Bir projenin DevOps ortamına alınabilmesi için çok sayıda ayar ve konfigürasyonun yapılması gerekir. Bu süreçte farklı araçlarda gerekli token'ların tanımlanması ve YAML dosyalarının hazırlanması manuel yürütüldüğünde ciddi emek ve zaman gerektirmektedir. Bu çalışmada önerilen yaklaşım, hazır/standart bir ürün kullanmaktan ziyade, kurumun ihtiyaçları ve mevcut (on-premises) altyapı kısıtları doğrultusunda kodlanarak geliştirilen bir **"**DevOps Yöneticisi**"** aracı üzerinden ilerlemektedir.

Şekil 7.1'de özetlenen bu yapı; ilgili API'lere erişerek gerekli token'ları otomatik olarak temin etmekte, Argo CD ve Tekton bileşenleri ve ilgili ortamlar için gereken YAML tanımlarını hedef ortama uygun şekilde üretmekte ve mevcut şablonlar üzerinde gerekli düzenlemeleri yaparak süreçleri standartlaştırmaktadır.

Şekil 7.1. DevOps Yöneticisi uygulaması ile DevOpsa alma süreci

Böylece, yazılımcı tarafından kod seviyesinde DevOps'a uygun hale getirilen proje, otomatikleştirilen bu adımlar sayesinde yaklaşık 5–10 dakika gibi kısa bir sürede DevOps ortamına alınabilmektedir.

8. DEVOPS KÜLTÜRÜNÜN YAZILIMCILARA AKTARILMASI

DevOps yalnızca bir araç seti ya da “pipeline kurma” işi değildir; yazılım geliştirme (Dev) ve operasyon (Ops) ekipleri arasındaki hedefleri, iletişimi ve sorumluluk paylaşımını yeniden tanımlayan bir çalışma biçimidir. Bu nedenle DevOps, belirli teknolojilerden bağımsız olarak; ortak sahiplenme (end-to-end ownership), sürekli geri bildirim, otomasyon, ölçümleme ve sürekli iyileştirme gibi davranış kalıplarını merkeze alan bir kültür olarak değerlendirilir. Araçlar değişebilir; ancak ekiplerin birlikte çalışma yaklaşımı, karar alma süreçleri ve kalite/teslimat hedefleri DevOps’un sürdürülebilirliğini belirleyen asıl unsurdur..

DevOps kültürünün yazılımcılara aktarılması, teslimat süreçlerinin yalnızca bir “Ops sorumluluğu” olarak görülmesini engeller ve yazılımın yaşam döngüsünü uçtan uca hızlandırır. Yazılımcı DevOps prensiplerini benimsediğinde; uygulamanın çalışabilirliği, izlenebilirliği, konfigürasyon yönetimi, dağıtım stratejileri ve hata senaryoları gibi konular geliştirme sürecine daha erken aşamada dahil edilir. Bu durum hem üretim ortamında yaşanan problemlerin azalmasını sağlar hem de geliştirme–dağıtım döngüsünü kısaltır. Kısacası DevOps, “sonradan eklenen bir operasyon katmanı” değil, yazılımın tasarımından itibaren gözetilen bir yaklaşım olduğunda verim üretir.

Kurumlarda DevOps dönüşümünün en zorlandığı alanlardan biri, eski teknolojiyle geliştirilmiş ve uzun süredir yaşayan projelerin DevOps süreçlerine taşınmasıdır. Bu tür projelerde çoğu zaman kod seviyesinde refaktör, bağımlılıkların güncellenmesi, yapılandırmanın ayrıştırılması, çalışma zamanı parametrelerinin düzenlenmesi ve deployment yaklaşımının yeniden kurgulanması gibi adımlar gereklidir.

Pratikte ise DevOps Engineer rolü çoğu kurumda altyapı, CI/CD altyapısının kurulması, ortam standartları, güvenlik politikaları, izleme/metric altyapısı ve operasyonel süreklilik gibi konulara odaklanır. Bu nedenle DevOps Engineer’ların uygulama koduna doğrudan müdahalesi genellikle sınırlıdır. Zorunlu hallerde Dockerfile hazırlanması, temel containerizasyon düzenlemeleri veya merge gibi operasyonel müdahaleler yapılabilse de; uygulamanın iç mimarisi ve kod tabanının DevOps’a uyumlu hale getirilmesi çoğunlukla yazılımcıların sorumluluğunda kalır.

Öte yandan Ops ekibinin de kendi asli sorumluluk alanları (sistem sürekliliği, erişilebilirlik, güvenlik, kapasite, incident yönetimi vb.) bulunduğundan, eski bir uygulamanın DevOps’a uyarlanması için gereken geliştirme odaklı işleri üstlenmesi beklenmez. Sonuç olarak, DevOps’a geçmek isteyen ancak eski teknolojiye sahip uygulamaları bulunan

yazılımcılar, gerekli bilgi ve rehberlik yeterince sağlanmadığında sürecin "nereden başlanacağı" ve "hangi standartların sağlanacağı" noktasında zorluk yaşayabilmektedir. DevOps Engineer'ların koda sınırlı müdahalesi ve Ops ekiplerinin farklı önceliklere sahip olması, kurum içinde bu dönüşümde bir ara katman ihtiyacı oluşturur. Bu ara katman ihtiyacını kapatmak için kurumlarda iki tamamlayıcı yaklaşım öne çıkmaktadır:

1. Dönüşüm Komisyonu / Mentorluk Yapısı:
   Daha önce benzer projeleri DevOps süreçlerine taşımış, uygulama dönüşüm deneyimi olan geliştiricilerden (ve gerekiyorsa mimari/DevOps temsilcilerinden) oluşan küçük bir komisyon kurulabilir. Bu yapı; proje bazlı yönlendirme, teknik standartların netleştirilmesi, örnek şablonların paylaşılması ve "uygulama kabul kriterleri"nin belirlenmesi gibi görevler üstlenir. Böylece yazılımcı ekip, belirsizlik yaşamadan dönüşüm adımlarını planlayabilir ve DevOps Engineer ekibi de altyapı standartlarını koruyarak süreci ölçekleyebilir.
2. Analiz Araçlarının Kullanımı:
   Projenin kod tabanını ve konfigürasyon yapısını analiz ederek eksik gereksinimleri (ör. yapılandırma ayrıştırma, environment değişkenleri, loglama standardı, health-check uçları, imaj şablonları vb.) raporlayan araçlar kullanılması, dönüşümü sistematik hale getirir. Bu tür araçlar, manuel kontrol yükünü azaltır; aynı zamanda projelerin DevOps'a alınma sürecinde ortak bir kontrol listesi yaklaşımı oluşturur.

Bu iki yaklaşımın birlikte uygulanması, eski teknolojili projelerin DevOps'a taşınmasını kişisel çabalara bağlı olmaktan çıkarıp kurumsal bir standarda dönüştürür. Sonuç olarak, projelerin DevOps ortamına alınma hızı artar, ekipler arası bağımlılıklar azalır ve dönüşüm süreci daha öngörülebilir hale gelir.

Ek olarak, APM (Application Performance Monitoring) araçlarının sağladığı görünürlük ve ölçülebilirlik, DevOps'a geçişi hedefleyen yazılımcılar için güçlü bir motivasyon oluşturur. Canlı ortamda gecikme, hata oranı, işlem izleri (trace) ve bağımlılık ilişkileri gibi metriklerin izlenebilmesi; performans iyileştirme ve hata giderme süreçlerini daha sistematik hale getirir. Bu sayede DevOps'un yalnızca dağıtım otomasyonu değil, aynı zamanda izlenebilirlik ve sürekli iyileştirme yaklaşımı olduğu daha net anlaşılır. Özellikle eski teknolojiyle geliştirilen projelerde APM kullanımının artması, projelerin DevOps ortamına taşınmasına yönelik istek ve talebin yükselmesine katkı sağlar.

9. SONUÇ VE ÖNERİLER

Bu tez çalışması, modern yazılım geliştirme süreçlerinde DevOps metodolojisinin kurumsal entegrasyonu ve uygulamadaki yansımalarını çok yönlü olarak ele almıştır. DevOps'un teorik çerçevesinden başlayarak, uçtan uca otomasyon senaryoları üzerinden pratik uygulamalara uzanan geniş kapsamlı bir analiz sunulmuştur.

Araştırma kapsamında, DevOps yaşam döngüsünün tüm aşamaları ayrıntılı biçimde incelenmiş ve bu aşamaların kurumsal yapılara entegrasyon sürecinde karşılaşılan zorluklar değerlendirilmiştir. Özellikle CI ve CD süreçlerinin, organizasyonel dönüşümdeki stratejik rolü vurgulanmıştır. DevOps'un sadece teknik bir metodoloji değil, aynı zamanda kültürel bir dönüşüm aracı olarak işlev gördüğü ortaya konmuştur.

Çalışma boyunca uygulamaya yönelik bir yaklaşım benimsenmiş; DevOps prensiplerinin gerçek hayat uygulamalarına nasıl adapte edilebileceği detaylı biçimde ele alınmıştır. Bu bağlamda, organizasyonların dijital dönüşüm süreçlerini destekleyecek stratejik öneriler sunulmuş, kullanılan açık kaynak teknolojilerin (Tekton, GitLab, Argo CD vb.) verimlilik ve kaliteye katkısı analiz edilmiştir.

Uygulanan çözüm mimarisi sayesinde, yazılım yaşam döngüsünün her aşamasında otomasyon sağlanmış, manuel müdahaleye duyulan ihtiyaç minimuma indirilmiş ve sistematik bir iş akışı oluşturulmuştur. Otomatik test süreçleri, kod kalite kontrolleri ve versiyonlama sistemleri, geliştirme süreçlerinin güvenilirliğini artırmış; hataların erken aşamada tespit edilmesine olanak tanımıştır.

Özellikle kullanılan tüm araçların açık kaynak ve ücretsiz olması, DevOps altyapısının lisans maliyeti olmadan da sürdürülebilir şekilde kurulabileceğini göstermektedir. Bu yönüyle çalışma, farklı ölçeklerdeki organizasyonlara uygulanabilir bir model sunmakta; maliyet-etkin bir DevOps altyapısının kurulabilirliğini pratik bir örnekle ortaya koymaktadır.

Araştırma sonucunda, DevOps uygulamalarının başarıyla hayata geçirilebilmesi için üç temel faktör öne çıkmaktadır:

1. Kültürel Dönüşüm: Ekipler arası iletişimin ve iş birliğinin güçlendirilmesi, DevOps'un sürdürülebilir başarısı için kritik öneme sahiptir. Geleneksel silo yapıların ortadan kaldırılarak ortak sorumluluk kültürünün teşvik edilmesi gerekmektedir.

2. Süreç Optimizasyonu: Sürekli iyileştirme ve otomasyon uygulamaları, yazılım teslim süreçlerinde verimliliği ve kaliteyi artırmaktadır.
3. Teknolojik Altyapı: Modern CI/CD araçlarının ve gözlemlenebilirlik bileşenlerinin entegre bir yapı içerisinde çalışması, otomasyonun başarısını doğrudan etkilemektedir. Seçilen teknolojilerin birbiriyle yüksek entegrasyon uyumu göstermesi önemlidir.

DevOps uygulamaları, yalnızca yazılım geliştirme sürecini hızlandırmakla kalmamakta; aynı zamanda müşteri geri bildirimlerine daha hızlı yanıt verilmesini ve sürekli iyileştirme döngüsünün etkin bir şekilde işletilmesini sağlamaktadır. Bu da organizasyonlara hem operasyonel verimlilik hem de rekabet avantajı sunmaktadır.

Bu çalışmada geliştirilen mimari, DevOps yaklaşımlarının orta ve büyük ölçekli kurumsal yapılarda nasıl uygulanabileceğine dair örnek bir yapı sunmaktadır. Ancak, gelecekteki çalışmalarda, özellikle küçük ve orta ölçekli işletmelerde (KOBİ) DevOps uygulamalarının etkinliği, sürdürülebilirliği ve sektörel farklar bağlamında adaptasyonu daha detaylı şekilde araştırılabilir.

Sonuç olarak, başarılı bir DevOps altyapısı için teknik yeterliliklerin yanında, organizasyonel olgunluk, kültürel adaptasyon ve sürekli öğrenme ilkeleri ön planda tutulmalı; teknolojik yatırımlar bu değerlerle bütünleşik şekilde planlanmalıdır.

KAYNAKÇA

Apache Maven (2026). Apache Maven Project. [Erişim: 10.02.2026, https://maven.apache.org/].

Argo CD (2026). Argo CD Documentation. [Erişim: 10.02.2026, https://argo-cd.readthedocs.io/].

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

Ceph (2026). Ceph Documentation. [Erişim: 10.02.2026, https://docs.ceph.com/].

Chandramouli, R. (2022). Implementation of DevSecOps for a microservices-based application with service mesh (SP 800-204C). National Institute of Standards and Technology.

Chen, L. ve Kim, M. (2023). Delivering effective DevOps culture: A systematic literature review. Journal of Systems and Software, 195.

Clair (2026). Clair Documentation. [Erişim: 10.02.2026, https://quay.github.io/clair/].

Docker (2026). Docker Documentation. [Erişim: 10.02.2026, https://docs.docker.com/].

Fitzgerald, B. ve Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123.

Forsgren, N., Humble, J. ve Kim, G. (2018). Accelerate: The science of lean software and DevOps: Delivering and scaling high performing technology organizations. IT Revolution Press.

GitLab (2026). GitLab Documentation. [Erişim: 10.02.2026, https://docs.gitlab.com/].

Grafana (2026). Grafana Documentation. [Erişim: 10.02.2026, https://grafana.com/docs/].

Grafana Loki (2026). Loki Documentation. [Erişim: 10.02.2026, https://grafana.com/docs/loki/latest/].

Haddad, R. A. A. (2024). Overcoming challenges in DevOps adoption: Insights from case studies. Journal of Electrical Systems, 20(10s).

Helm (2026). Helm Documentation. [Erişim: 10.02.2026, https://helm.sh/docs/].

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

Jaeger (2026). Jaeger Documentation. [Erişim: 10.02.2026, https://www.jaegertracing.io/docs/].

Kaniko (2026). Kaniko Documentation. [Erişim: 10.02.2026, https://github.com/GoogleContainerTools/kaniko].

Kim, G., Debois, P., Willis, J. ve Humble, J. (2016). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press.

Kiran, P. (2022, 1 Temmuz). The role of DevOps in digital transformation. [Erişim: 10.02.2026, https://www.opensourceforu.com].

Kubernetes (2026). Kubernetes Documentation. [Erişim: 10.02.2026, https://kubernetes.io/docs/home/].

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., ... ve Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. Information and Software Technology, 114.

Pan, J.-S., Dao, T.-K., Nguyen, T.-T., Nguyen, T.-X.-H. ve Weng, S. (2025). Jira's collaboration: Survey insights on optimizing team planning, tracking, and release management. Advances in Smart Vehicular Technology, Transportation, Communication and Applications, 429, Springer.

Prometheus (2026). Prometheus Documentation. [Erişim: 10.02.2026, https://prometheus.io/docs/introduction/overview/].

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

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

Sawant, N. C. (2025). Grafana: A monitoring tool. International Research Journal of Engineering and Technology (IRJET), 12(1).

Sentry (2026). Sentry Documentation. [Erişim: 10.02.2026, https://docs.sentry.io/].

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

SonarQube (2026). SonarQube Documentation. [Erişim: 10.02.2026, https://docs.sonarsource.com/sonarqube/].

Tekton (2026). Tekton Documentation. [Erişim: 10.02.2026, https://tekton.dev/docs/].

Waseem, M., Liang, P., Shahin, M., Di Salle, A. ve Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. [Erişim: 10.02.2026, https://arxiv.org].

Willis, J. ve Edwards, D. (2022). The evolution of DevOps practices in enterprise organizations. Journal of Software: Evolution and Process, 34(3).