T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK BANKACILIK SİSTEMLERİNDE MONOLİTİK MİMARİDEN MİKROSERVİS MİMARİYE GEÇİŞ

FATİH DERVİŞİ

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Tez Danışmanı: Dr. Öğr. Üyesi Cem TAŞKIN

EDİRNE-2023

FATİH DERVİŞİ'nin hazırladığı **"AÇIK BANKACILIK SİSTEMLERİNDE MONOLİTİK MİMARİDEN MİKROSERVİS MİMARİYE GEÇİŞ"** başlıklı bu tez, tarafımızca okunmuş, kapsam ve niteliği açısından **Bilgisayar Mühendisliği** Anabilim Dalında bir **Yüksek lisans tezi** olarak kabul edilmiştir.

Jüri Üyeleri (Ünvan, Ad, Soyad): İmza

Dr. Öğr. Üyesi Cem TAŞKIN……. ……

Dr. Öğr. Üyesi Derya ARDA……. ……

Dr. Öğr. Üyesi Selçuk YAZAR…. ……

……. ……

……. ……

Tez Savunma Tarihi: 17 / 04 / 2023

Bu tezin Yüksek Lisans tezi olarak gerekli şartları sağladığını onaylarım.

İmza

Dr. Öğr. Üyesi Cem TAŞKIN

Tez Danışmanı ……

Trakya Üniversitesi Fen Bilimleri Enstitüsü onayı

……..

Prof. Dr. Hüseyin Rıza Ferhat KARABULUT

Fen Bilimleri Enstitüsü Müdürü

T.Ü.FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI YÜKSEK LİSANS PROGRAMI

DOĞRULUK BEYANI

Trakya Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında, tüm verilerin bilimsel ve akademik kurallar çerçevesinde elde edildiğini, kullanılan verilerde tahrifat yapılmadığını, tezin akademik ve etik kurallara uygun olarak yazıldığını, kullanılan tüm literatür bilgilerinin bilimsel normlara uygun bir şekilde kaynak gösterilerek ilgili tezde yer aldığını ve bu tezin tamamı ya da herhangi bir bölümünün daha önceden Trakya Üniversitesi ya da farklı bir üniversitede tez çalışması olarak sunulmadığını beyan ederim.

17 / 04 / 2023

Fatih Dervişi

İmza

Yüksek Lisans Tezi

Açık Bankacılık Sistemlerinde Monolitik Mimariden Mikroservis Mimariye Geçiş

T.Ü. Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

ÖZET

Açık bankacılık şirketlerinin var olan sistemlerini ya da üçüncü parti hizmet sağlayıcıların sağladığı hizmetleri kullanarak son kullanıcıya daha kolay ve hızlı erişim sağlayan bir sistemdir. Günümüzde hem yazılım endüstrisi hem de akademik anlamda mikroservis mimarisi en çok tercih edilen mimari haline gelmiştir. Mikroservis mimariler, monolitik yapılardan farklı olarak birbirinden bağımsız yazılım servislerine sahip olan, temel olarak bir yazılım uygulamasında belirli özellik ya da fonksiyonu sağlayan tek bir amaca hizmet eden servislerdir.

Tezin amacı, monolitik yazılım geliştirme servislerinden mikroservis yapılı yazılım servislerine geçilme sürecinde karşılaşılacak olan sorunlar için çözüm önerileri sunmaktır. Tez süresince yapılan mevcut literatür bilgilerinin incelenmesiyle, mikroservis mimarisine mevcut veri ve veri tabanının taşınma süreci hakkında detaylı bilgi sunulmuştur. Ayrıca araştırmalar sonucunda literatürdeki bazı eksiklikler ve çelişkiler nedeniyle, bu tez mikroservis mimari ve monolitik mimarinin kullanımı ve taşınması hakkında daha fazla araştırma yapmaya yöneltmiştir.

Yıl : 2023

Sayfa Sayısı : 67

Anahtar Kelimeler : Açık Bankacılık, Mikroservis Mimari, Monolitik Mimari.

Master's Thesis

Migration from Monolithic Architecture to Microservice Architecture in Open Banking Systems

Trakya University Institute of Natural and Applied Sciences

Departament of Computer Engineering

ABSTRACT

Open banking is a system that provides easier and faster access to the end user by using the existing systems of open banking companies or by third party services provided by service providers. Moving data from a monolithic application to a microservice structure is not impossible, but involves many complexities. Microservice architectures, unlike monolithic structures, are services that have independent software services, basically serving a single purpose, providing a specific feature or function in a software application.

The aim of the thesis is to offer solutions for the problems that will be encountered in the process of transitioning from monolithic software development services to microservice structured software services. By examining the existing literature information made during the thesis, detailed information about the migration process of the existing data and database to the microservice architecture is presented. In addition, due to some deficiencies and contradictions in the literature as a result of research, this thesis has led to more research on the use and migration of microservice architecture and monolithic architecture.

Year : 2023

Number of Pages : 67

Keywords : Open Banking, Microservice Architecture, Monolithic Architecture.

TEŞEKKÜR

Tez çalışmamın hazırlanmasında bana yol gösteren, bilgi ve tecrübelerini benden esirgemeyen, bana her zaman destek olan danışman hocam Sayın Öğr. Üyesi Cem TAŞKIN'a ve Bilgisayar Mühendisliği bölümü saygıdeğer hocalarıma teşekkürlerimi sunarım.

Hayatım boyunca desteklerini esirgemeyen ve her zaman yanımda olan sevgili aileme teşekkür ederim.

İÇİNDEKİLER

ÖZET iv
ABSTRACT v
TEŞEKKÜR vi
İÇİNDEKİLER vii
SİMGELER VE KISALTMALAR DİZİNİ ix
ŞEKİLLER DİZİNİ x
BÖLÜM 1 GİRİŞ 1
BÖLÜM 2 YAZILIM MİMARİLERİ 5
2.1. Monolitik Mimari 5
2.1.1. Teknolojik Açıdan Monolitik Mimarisi 9
2.1.2. Monolitik Mimarinin Dezavantajları 9
2.2. Mikroservis Mimari 10
2.2.1. Teknolojik Açıdan Mikroservis Mimarisi 14
2.3. Monolitik Problemlerin Mikroservis ile Çözümü 15
2.4. Mikroservis ve Monolitik Mimarileri Arasındaki Farklar 16
2.5. Mikroservis Mimarisinin Avantajları 17
2.5.1. Ölçeklendirebilirlik 18
2.5.2. Yüksek Kullanılabilir Olması ve Hata Toleransı 18
2.5.3. Dağıtımın Kolaylaştırılmış Olması 19
2.5.4. Farklı Teknolojilerle Kullanılabilirliği 19
2.6. Mikroservis Mimarisinin Dezavantajları 19

2.7. Mikroservis Mimarisinin Güvenilirliği 20
2.8. Monolitik Mimariden Mikroservis Mimariye Taşınma Süreci 21
2.9. Mikroservis Haberleşme Mekanizmaları 25
2.9.1. API Gateway 25
2.9.2. Servisler Arası Haberleşme Mekanizmaları 27
2.9.3. Asenkron Mesaj Tabanlı İletişim 28
BÖLÜM 3 MİKROSERVİS BİLGİ İŞLEM PLATFORMLARI 29
3.1. Konteyner Tabanlı Mikroservisler 29
3.2. Fonksiyon Tabanlı Mikroservisler 31
BÖLÜM 4 MATERYAL VE METOD 32
4.1. Monolitik Mimariden Mikroservis Mimarisine Modüler Migrasyon 32
4.2. Monolitik Mimariden Mikroservis Mimariye Geçiş Senaryosu 35
4.3. Monolitik ve Mikroservis Yapılarda Operasyonel Karşılaştırma 39
BÖLÜM 5 SONUÇLAR 48
BÖLÜM 6 TARTIŞMA 51
KAYNAKLAR 53
ÖZGEÇMİŞ 54

SİMGELER VE KISALTMALAR DİZİNİ

API:	Application Programming Interface (Uygulama Programlama Arayüzü)
AWS:	Amazon Web Services (Amazon Web Hizmetleri)
CI:	Continuous Integration (Sürekli Entegrasyon)
CD:	Continuous Delivery (Sürekli Teslimat)
HTTP:	Hypertext Transfer Protocole (Hipermetin Aktarım Protokolü)
IPC:	Inter Process Communication (Ara İlerişim)
REST:	Representational State Transfer (Temsili Durum Aktarımı)
SOA:	Service Oriented Architecture (Servis Odaklı Mimari)
URL:	Uniform Resource Locator (Tekdüzen Kaynak Bulucu)
VM:	Virtual Machine (Sanal Makine)

ŞEKİLLER DİZİNİ

Şekil 2.1. Basit bir monolitik mimari şeması..... 6

Şekil 2.2. Monolitik uygulamanın basit gösterimi..... 7

Şekil 2.3. Basit bankacılık sisteminde mikroservis mimarisi ile servislerin kullanımı. ... 9

Şekil 2.4. Mikroservis mimarinin farklı şekilde gösterilmesi..... 12

Şekil 2.5. Mikroservis mimari örneği. 13

Şekil 2.6. Monolitik ve mikroservis mimarilerinin şekilsel gösterimi. 16

Şekil 2.7. Farklı kullanıcılardan uygulamaya gönderilen istekler. 26

Şekil 2.8. API Getaway kullanarak kullanıcıların hizmetlere ulaşımının sağlanması.... 27

Şekil 3.1. Her konteyner için bir servis örneği. 30

Şekil 4.1. Monolitik yapıdan mikroservis yapıya geçiş..... 34

Şekil 4.2. Senaryosunun kurulduğu monolitik yapıdaki sistemler. 36

Şekil 4.3. CI/CD sürecinin AWS ortamında örneği..... 38

Şekil 4.4. Monolitik ve mikroserviste yapılan istek sayısına karşı çalışma süresi. 40

Şekil 4.5. Monolitik ve mikroserviste eş zamanlı istek süresine karşı çalışma süresi.... 41

Şekil 4.6. Monolitik ve mikroserviste istek zincir derinliğine karşı güvenilirlik. 43

Şekil 4.7. Monolitik ve mikroserviste örnek sayısına karşı RAM kullanımı. 44

Şekil 4.8. Monolitik ve mikroserviste çalışma süresine karşı dolar bazında harcama. .. 46

Şekil 4.9. Monolitik ve mikroserviste geliştirici ekip sayısına karşı iletişim kanalları sayısı. 47

BÖLÜM 1

GİRİŞ

Özellikle son yıllarda yazılım şirketlerinin hızlı bir şekilde büyümesi web servislerinin geliştirilmesi sırasında yazılım mimarilerine ihtiyaç duyulmaya başlanmıştır. Teknolojinin hızla değişimi ve gelişimi beraberinde birçok sorun getirmekle beraber aynı zamanda ekonomik hayatın da daha fazla gelişmesene ve değişmesine neden olduğunu söyleyebiliriz. Bunun sonucunda banka sistemlerinde kullanılan ürünlerin ve sistemlerin büyük bir kısmının teknolojinin büyük bir hızla gelişmesiyle değiştirilmesi zorunluluğu oluşmuştur. Teknolojinin bu denli hızlı gelişimi banka sistemlerinin kararlılık elde etmesini ve sistemlerini daha fazla müşteri odaklı ve yeni teknoloji odaklı geliştirmesini sağlamıştır.

Açık bankacılık tanımında kullanılan açık ifadesine bakıldığında hedef odaklı oluşturulmuş herhangi bir yenilik ve aynı zamanda esnek olarak koordine edişmiş olan yeniliklerden bahsedilebilir. Bu tanımdan da anlaşılacağı üzere açık bankacılık sistemleri hedef odaklı ve yenilikçi sistemler olduğu sonucuna varılabilir (Levine & Prietula, 2014). Açık bankacılık sistemleri kullanıcılara daha yenilikçi sonuç ve ürünler sunulabilmesi amacıyla geliştirilen sistemlerdir. Bu sistemler genel olarak müşterilerin verilerinin bankalar tarafından API'ler aracılığı ile irin ve hizmet geliştiriciler ile paylaşılması şeklinde tanımlanabilir. Basit olarak açık bankacılık yenilikçi veya inovasyondan türetilen bir kavramdır. Açık bankacılık, yukarıdaki tanımlardan da anlaşılacağı üzere müşteri verilerinin ürün ve hizmet geliştirici kurumlar işe paylaşılması gerektiği kavramını benimser (Heinrich vd., 2017).

Geleneksel eski bankacılık yöntemleri ile yenilikçi açık bankacılık yöntemini karşılaştırdığımızda ise, açık bankacılık sistemlerinde bankalar ve misteriler arasında üçüncü parti hizmet sağlayıcılar olduğu görülür. Geleneksel bankacılık sistemlerinde ise. Müşteri ve banka arasında başka kurum ve hizmetler yer almaz

Teknoloji ve yazılımın bu denli hızlı bir şekilde gelişmesi ile günümüzde hem akademik hem de yazılım açısından kabul edilmiş mikroservis mimarisinin diğer mimarilere tercih edilmesi artmıştır. İşletme ve bankaların dijitalleşmesi yazılım geliştirme için mükemmel denilebilecek farklı çözümler arama ihtiyacı sunmuştur.

Açık bankacılık sistemlerinde uygulama geliştiricileri tarafından hem uygulamanın daha hızlı ve verimli şekilde gelişmesi hem de ortaya çıkabilecek hataların en kısa sürede ortadan kaldırması için en fazla kullanılan yazılım geliştirme mimarileri monolitik mimari ve mikroservis mimaridir. Literatüre bakıldığında ise, monolitik mimarilerin mikroservis mimarilerine dönüştürülmesi için birçok kayda değer nedenlere rastlanmaktadır. Fakat özellikle mevcut monolitik mimariden verilerin mikroservis mimarisine nasıl taşınacağına dair literatür bilgileri sınırlıdır.

Monolitik mimari, yazılım geliştirme mimarilerinin geleneksel yapısı olarak tanımlanabilir. Monolitik mimariye alternatif bir yaklaşım olarak mikroservis mimarisi teknolojik açıdan ortaya sunulmuştur. Daha küçük ölçekli yeni bir uygulama geliştirilmeye başlanıldığı zaman genellikle yazılım ekipleri tarafından monolitik yaklaşım seçilir. Monolitik mimari ile tasarım yapmak geliştiricilere hız kazandırmaktadır fakat ilerleyen dönemlerde uygulama daha büyük ölçekli hale geldiğinde yeni teknolojilere adaptasyon ve her servisin bağımsız bir şekilde güncelleyememek gibi farklı sorunları ortaya çıkabilir. Mikroservis mimarisi ise, ölçeklendirebilir olması, dağıtım ve bakımının kolay olması, arızaya karşı dayanıklı olması ve pazara çok daha hızlı sunulması gibi özellikleri sayesinde son zamanlarda çok fazla kullanılan bir mimari hizmete dönüşmüştür. Sayılan özellikleri sayesinde birçok yazılım şirketi yaptıkları çalışmalardaki mimarileri monolitik mimariden mikroservis mimariye dönüştürme eğilimi göstermeye başlamıştır. Bu dönüşüm sürecinde ise yazılım ekipleri, mevcut uygulamadaki kod tabanlarının yeniden kullanılmasının önemli olduğu büyük yazılım uygulamalarında, bu kodların yeni mimariye taşınmasında büyük zorluklarla karşı karşıya kalırlar (Salonen & Deleryd, 2011).

Mikroservis mimarileri diğer mimari yöntemler ile karşılaştırıldığında yapı olarak daha az karmaşık ve daha kolay kullanılabilir bir yapıya sahiptir bu yüzden oluşturulan servisler de karmaşıklığı az ve daha anlaşılırdır. Aynı zamanda mikroservis mimarisinde oluşturulan modeller tek bir işlev içereceği için hem güncelleme hem de hataların bulunmasında monolitik uygulamaya göre daha az maliyetli bir yapıya sahip olacaktır. Mikroservis mimarisi ile geliştirilen bir sistem diğer mimari yöntemleri ile karşılaştırıldığında, yüksek ölçeklendirilebilirlik, sürdürülebilirlik ve çok dilli geliştirme sağlaması gibi avantajlardan bahsedilebilir (Knoche & Hasselbring, 2019). Mikroservis hizmetler mimarisi, akademik olarak ve aynı zamandaş endüstride de ilgi görüyor ve monolitik mimariye tercih ediliyor. Günümüzde, birçok büyük şirket uygulamalarını ve sistemlerini buluta taşımıştır. Bulut bilişim sistem modeli, bu şirketlerin bilgiişlem kaynakları kullanımlarına göre daha kolay ölçeklendirme yapmalarını sağlamaktadır. Mikro sistemlerde, tek bir uygulama olarak çalışan birçok küçük hizmete sahip sistemler HTTP kaynak API'si gibi küçük mekanizmalar yardımıyla birbiri arasında iletişim kurar ve her hizmet kendi sürecinde bağımsız olarak çalışır.

Mikroservis hizmetler bağımsız olarak dağıtılabilen modüler hizmetler oldukları için sıklıkla ortaya çıkan sorun ve bağımlılıklardan kaçınmak için önemli bir yeniliktir. Sistem içerisinde daha az bağımlılık, sistemde değişiklik yapmayı kolaylaştırır ve veri tabanında değişiklikler gerçekleştirirken sistemin bozulma riskini azaltır. Mikroservis mimarisinin kullanımının daha yüksek maliyet getirmesi ile ilgili yapılan araştırmada ise, sistemin boyutuna ve karmaşıklığına bağlı olarak değişeceği gözlemlenmiştir. Genel olarak, mikroservis mimari monolitik mimarinin sınırlarını ele almayı ve çözmeyi amaçlar. Monolitik sistemdeki uygulama işlevsel hizmetlere bölünmüş, her hizmetin kendi arasında yüksek uyumu vardır ve çoğu uygulamadaki sistemlerden daha küçüktür.

Bu tez çalışmasında, monolitik bir yapıda bulunan mevcut verilerin mikroservis tabanlı bir uygulamaya veri taşınma sorunu ele alınmış, taşınma nedenleri ve sonuçları açıklanmıştır. Tez çalışmasında, açık bankacılık sisteminde monolitik mimaride bulunan yapının mikroservis mimariye taşınma sürecine ortaya çıkabilecek sorunlar ele alınmış ve çözüm önerileri sunulmuştur. Mikroservis mimarisi, yazılım geliştirmek için yaygınlaşan yeni bir yaklaşım olduğu için eski sistemlerin bu mimariye dönüştürülmesine önemli bir kılavuz sunulmaktadır. Bu tez çalışması boyunca, mimariler hakkında önemli bilgiler toparlanmış ve mimarilerin taşınma süreci açıklanmıştır. Bu çalışma için yapılan

araştırmada, mikroservis ve monolitik mimari karşılaştırılmakta ve değerlendirilmektedir. Başka bir deyişle, mikroservis kullanımı ile devamlılık, tekrar kullanım, ölçeklendirilme ve otomatik devreye alma gibi özelliklerden dolayı süreçler kolaylaşacağından diğer mimari yöntemlerine tercih edilmeye başlanmıştır. Mikro hizmetlerin temel avantajlarından birincisi teknoloji heterojenliğine güvenilirliği yani her hizmet için farklı teknoloji kullanma imkanıdır. Mikro hizmetlerin önemli diğer bir avantajı ise, sistemin bir bölümünde oluşan arızanın sistemin geri kalanını etkilememesidir. Bu avantaja dayanıklılık adı verilmektedir (Newman, 2015). Monolitik mimaride bir kod tabanında sıkıştırılmış onlarca ya da yüzlerce farklı hizmete rastlanabilir. Bu durum da aynı ortamda çalışan ekipler için birçok farklı sorun yaratabilir. Bundan dolayı, küçük ya da büyük birçok şirket geliştirme ekiplerinin birbirleri arasında daha kolay koordine olmaları ve daha iyi bir performans için mimarilerinde değişime başvurmuşlardır.

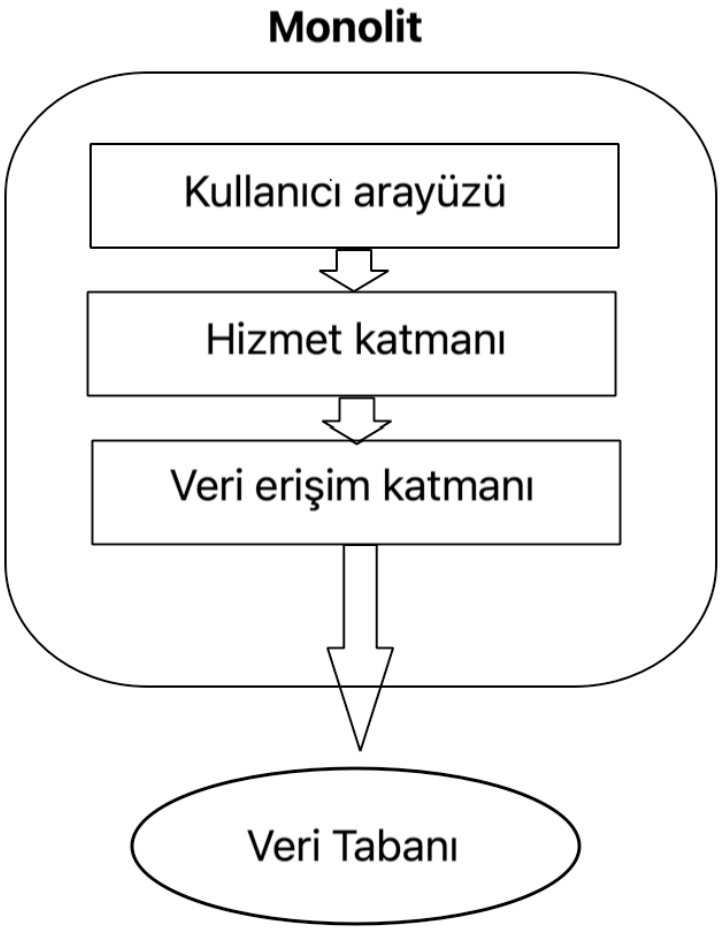
Tezin amacı ise, monolitik yazılım geliştirme servislerinden mikroservis yapılı yazılım servislerine geçilme sürecinde karşılaşılacak olan sorunlar için çözüm önerileri sunmaktır. Yapılan kaynak ve mevcut literatür bilgilerinin incelenmesiyle, mikroservis mimarisine mevcut veri ve veri tabanının taşınma süreci hakkında detaylı bilgi sunulmuştur. Ayrıca araştırmalar sonucunda literatürdeki bazı eksiklikler ve çelişkiler nedeniyle, bu tez mikroservis mimari ve monolitik mimarinin kullanımı ve taşınması hakkında daha fazla araştırma yapmaya yöneltiştir.

BÖLÜM 2

YAZILIM MİMARİLERİ

2.1. Monolitik Mimari

Monolitik mimari, geliştirilen bütün parçaların aynı proje içinde olduğu geleneksel ve geniş bir topluluk tarafından kullanılan mimari çeşididir. Monolitik bir uygulama çok fazla farklı hizmete API ve HTML sayfaları gibi farklı arabirimlere hizmet vermek için sadece tek bir kod tabanını kullanan mimarilerdir. Uygulama geliştirmeye başlarken monolitik mimari standart yol olarak kullanılmaktadır. Monolitik yaklaşım ile oluşturulan mimaride kullanılan tek bir kod tabanı, uygulama boyutu küçük olduğu süre boyunca uygulamanın devreye alınmasını ve ölçeklenmesini kolaylaştırır. Monolitik mimaride genellikle tek bir veri tabanı kullanılır ve bu veri tabanı parçalama yoluyla farklı bölümlere ayrılsa da aynı şema kullanılır. Yapılan birçok uygulama başlangıçta basit bir yapıyla başlar ve giderek zorlaşır bu durumda monolitik mimari ile yapılacak en önemli çözüm uygulamayı parçalara bölmektir (Fowler, 2002). Bu şekilde katmanlı bir yaklaşım çok yaygın olarak kullanılmaktadır. Şekil 2.1.'de de gösterildiği gibi katmanlı monolitik mimari, birçok geliştirici tarafından kullanılan en yaygın mimari şemasıdır. Son aşamadaki veri erişim katmanı daha sonra bu uygulamayla ilgili tüm verileri işleyen bir veri tabanına erişir.



Şekil 2.1. Basit bir monolitik mimari şeması.

Monolit kelimesinin sözlük anlamına bakıldığında ‘sütun şeklinde büyük taş’, ‘değişmek istemeyen büyük ve güçlü sistem’ gibi anlamlara geldiği görülür. Sözlük anlamına da bakıldığında bir durumu veya bir nesneyi tanımlamak için monolit kelimesini ‘tek veya bütün’ olarak kullanmak anlam bütünlüğü sağlayacaktır. Bu nedenle sözlük anlamından da görüldüğü üzere monolitik mimari, gerekli hizmetleri sağlamak için tek ve bütün olarak oluşturulmuş uygulamayı tanımlamaktadır. Yazılım mimarilerinde monolitik mimari, sistemi oluşturan tüm bileşenlerin tek bir kod tabanında toplanması olarak tanımlanır (Candela, Bavota, Russo& Oliveto, 2016).

Gerçek anlamda, monolitik mimariden bahsedildiğinde mimarinin ihtiyaç duyduğu bütün işlevleri bir araya getiren ve tek bir uygulama tarafından çalışan mimarilerden bahsedilebilir (Gökalp, 2016). Monolitik mimari hızlı geliştirebilme ve ilk adaptasyonunun kolay olması özelliklerinden dolayı küçük uygulamalar ile yapılan

çalışmalarda tercih edilmesini sağlar. Fakat bu özelliklerinin dışında kod sahiplik derecesinin düşük olması ve kurulum maliyetinin yüksek olması dezavantajları arasındadır. Aynı zamanda yapılan tüm değişimler tüm mimari yapısını da etkiler.

Terim olarak 'monolitik' bağımsız olan ve uzun süreli sürüm güncellemelerinde bakımı daha karmaşık hale gelen büyük ana bilgisayar uygulamaları tanımlamak için kullanılır (Sun, Nanda & Jaeger, 2015). Daha geniş olarak, bir monolit uygulama birden fazla işlevi içeren kod bloğu olarak tanımlanabilir. Bir uygulamanın ilk sürümü onlarca işlevi içerirken diğer ileri sürümleri yüzlerce farklı işlev içerebilecek yetkiye sahip olarak büyüyebilir (Richardson & Smith, 2016).



Şekil 2.2. Monolitik uygulamanın basit gösterimi.

Monolitik mimariyle yazılmış bir uygulamanın temel özellikleri şunlardır;

- Tek bir programlama dili ile yazılmıştır,
- Tek kaynak kod deposuna sahiptir,
- Uygulama tek bir çalışma zamanına derlenmiştir,
- Yüksek bağlantı.

Monolitik mimariler uygulama geliştirmenin geleneksel eski yolu olarak kabul edilmektedirler. Bu uygulama şekli büyük bir kod tabanına sahip ve modülerlikten yoksul olan mimari olarak tanımlanabilir. Monolitik mimaride, uygulamaya geliştiren geliştirici bir değişiklik yapmak istediğinde her şeyi güncellemesi gerekir. Değişim yapmak için aynı kod tabanına ulaşan geliştirici bu sayede tüm yığında bir kerede değişiklik yapar. Bu durum ise geliştirici için birçok zorluk sağlar. Monolitik yaklaşım, uygulama oluşturmanın standart yolu olduğu için herhangi yeni bir mühendis ekibi kolaylıkla bu

mimari şeklini uygulayabilir, çok gelişmiş ve üstün bilgi becerilerine sahip ekiple çalışmayı gerektirmez.

Hızlı bir teknolojinin başlarında iken monolitik mimariyle sistemi oluşturmak mantıklı olabilir. Monolitik mimariyle oluşturulmuş bir sistemin diğer mimarilerle oluşturulan sistemlere göre avantajları şöyle sıralanabilir;

- Test edilmesi daha basittir

Monolitik sistemlerde uygulamada otomatikleştirilmiş testler ve uçtan uca terslerle çalışma daha kolay olacağından dolayı test edilmesi de diğer mimarilere göre daha kolaydır.

- Kurulum basitliği

Monolitik bir uygulamada, kapsayıcıya kolayca dağıtılabilen tek bir koşullandırılabilir dosya ile paketlenmektedir.

- Basit geliştirilebilir olması

Uygulama bütün şeklinde olduğu için daha kolay geliştirilebilir tek tek yapılmasına gerek yoktur.

Bununla birlikte ekip ve uygulamanın büyümesiyle monolitik yapıdaki sistemin dezavantajları ortaya çıkacaktır. Sistemin küçük bir bölümünde yapılan değişiklik tüm sistemi etkileyeceğinden, tüm monolitik yapının yeniden oluşturulması ve test edilmesi gerekir. Sürekli dağıtımı zor olduğundan, sık küçük değişiklikler yapmaya cesaret edilemez. Zamanla, şirketin bünyesine katılan birçok yeni geliştirici ile kod tabanını modüler tutmak zorlaşır, kodu anlamayı zorlaştırır ve işi daha karmaşık hale getirir. Tüm monolitik mimariyi proje ortamına yüklemek projeyi yavaşlatabilir. Bu durum hem geliştiriciyi hem de dağıtımı etkiler. Monolitik sistemde bileşenlerin genellikle birden fazla gereksinimleri olabilir fakat bunları tek tek bağımsız olarak ölçeklendirmek imkânsız olduğu için bu durum geliştirici için bir dezavantajdır.

Şirket ve proje büyüdüğünde, tüm ekipler ek bir kod tabanı üzerinde çalınıyorsa onları bağımsız ve her alana bağlı olan ekiplere ayırmak daha mantıklı olacaktır. Monolitik mimaride her ekip tek bir işlevsel alana odaklanamaz tüm ekipler tek bir kod tabanı üzerinde çalışmak zorunda kalır. Monolitik bir sistem tek bir teknoloji yığınını ele

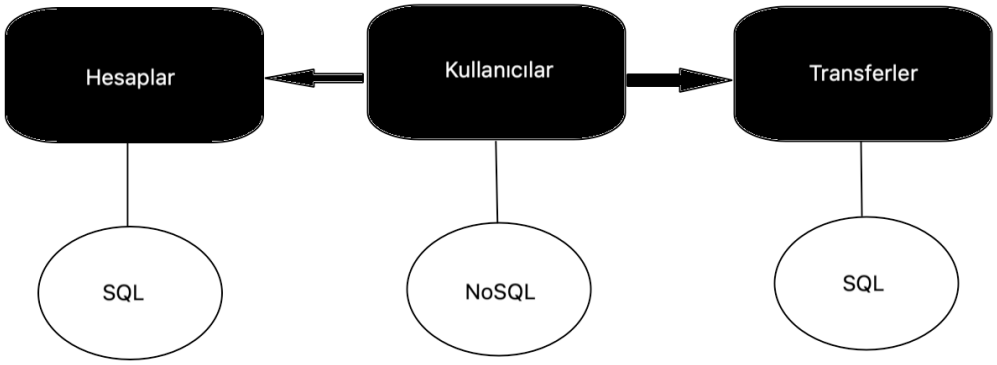
alır ve bu teknoloji ile çalışır. Sistemin farklı bileşenleri farklı programlama dilleri ile yazılamayacağı için teknolojiden bağımsız bir sistem olduğunu söyleyemeyiz.

2.1.1. Teknolojik Açıdan Monolitik Mimarisi

Mikroservis mimarisi ile geliştirilen yazılım servislerinin her birinin otonom olması her bir servis için farklı teknolojilerin kullanılabileceği anlamına gelir. Bu da yazılım geliştiriciler için her bir servise uygun programlama dili seçmelerine olanak sağlar.

Örnek olarak bir banka uygulamasını aldığımız zaman, oluşturulan mikroservis yapıda genel olarak kullanıcılar, transferler ve hesaplar olmak üzere üç adet servis bulunur. Mikroservis yapıdaki kullanıcı servisi aynı anda hem transferlere hem de hesaplara erişimi olması gereklidir. Bu iletişim yapısı veri tabanı olarak birbirinden bağımsız olacağı için mikroservis hizmetinin seçilmesi uygundur. Kurulması istenilen yapı mikroservis ilkeleri ile kolaylıkla uygulanacaktır (Bennett & Rajlich, 2000).

Kullanıcılar NoSQL tabanlı veri tabanını kullanırken, hesaplar ve transferler servisleri ise SQL tabanlı ver tabanını kullanabilirler.



Şekil 2.3. Basit bankacılık sisteminde mikroservis mimarisi ile servislerin kullanımı.

2.1.2. Monolitik Mimarinin Dezavantajları

Yazılım mühendisleri ve şirketleri kullandıkları teknolojiyle ilgili sürekli bir endişeleri ve dezavantajları ile ilgili korkuları mevcuttur. Bahsedilen dezavantajlar

doğrudan hem uygulamayı hem de yazılımı etkileyeceğinde bu dezavantajların en minimuma indirgendiği sistem ve mimariler tercih edilir. Geleneksek bir yöntem olan monolitik mimarinin en belirgin dezavantaj ve sorunu olarak tek bir kod tabanında tüm bileşenlerin paylaşıldığından bahsedilebilir. Aynı zamanda diğer bir önemli sorun da monolitik mimarinin bakımıdır. Mimarinin bakımı sadece uzun bir süreç değil aynı zamanda maliyeti yüksek de bir süreçtir (Bennett & Rajlich, 2000). Bahsedilen bakım süresinin zorluğu yazılım şirketinin pazara sunduğu ürünün güncellemesi gereken zaman ile bağlantılı olduğu için kullanıcıya olumsuz etki etmektedir. Bu durum, rekabet dünyasında önemli bir dezavantaj sağlamaktadır.

Teknik açıdan mimarinin oluşturduğu dezavantaja bakıldığında ise, yeni bir bileşen eklemek veya tek bir bileşenin bakımını yapmak tüm uygulamayı etkilemektedir. Aynı zamanda mimarinin boyutunun büyük olması uygulamayı daha karmaşık hale getirir ve değişikliklerin hızlı ve doğru bir şekilde yapılmasını zorlaştırır. Mimarinin tek bir tabanlı olması da uygulamanın başlatma süresini yavaşlatabilir. Bu durum, kullanıcı için önemli bir dezavantajdır. Monolitik uygulamalar ile ilgili diğer bir sorun da güvenliktir. Herhangi bir modülde oluşan hata tüm süreci etkileyebilir ve güvenlik açığı sağlayabilir. Ayrıca oluşan hata tüm uygulamanın kullanımını etkileyebilir.

Monolitik yaklaşım ile yapılan uygulamalarda yazılımda gelişen yeni teknolojilerin takibi daha zordur çünkü yazılım dilleri değişiklikleri sistemin tümünü etkileyeceğinden maliyet ve değişikliği uygulama zorluğu açısından son derece sıkıntılar yaşatır (Salonen & Deleryd, 2011).

Monolitik mimari yapılar ile geliştirilen sistemlerin bakım sorunu geliştiriciler ve kullanıcılar için önemli ölçüde endişe uyandıran bir sorundur. Bir şirketin yazılım ya da güncellemeyi pazara sunması için gereken ve harcanan süre bu bağlantının ana sebebini oluşturur. Monolitik mimarideki bakım sorunu yenilikçi ve sürekli gelişen teknoloji dünyasında büyümek için harcanan sürede birincil endişe kaynaklarındandır.

2.2. Mikroservis Mimari

SOA çalışma prensibine bakıldığında yazılım sistemlerine bütün uygulamayı oluşturan parçalara ayrı bir hizmet parçası olarak yaklaşır, fakat diğer mimarilerden farklı olarak bu ayrı ele alınan sistemler yerel olarak bulunabilen ayrı birimlerdir. Temel olarak,

SOA sistemlerine bakıldığında teknolojik olarak yazılım gereksinimlerini karşılamak amacıyla ayrı Web hizmetlerini bağlayarak geliştirilmesi olarak tanımlanabilir. Mimarinin içerisinde bulunan her hizmet ayrı ayrı diğer hizmetlerden bağımsız olarak farklı teknolojiler tarafından oluşturulabilir.

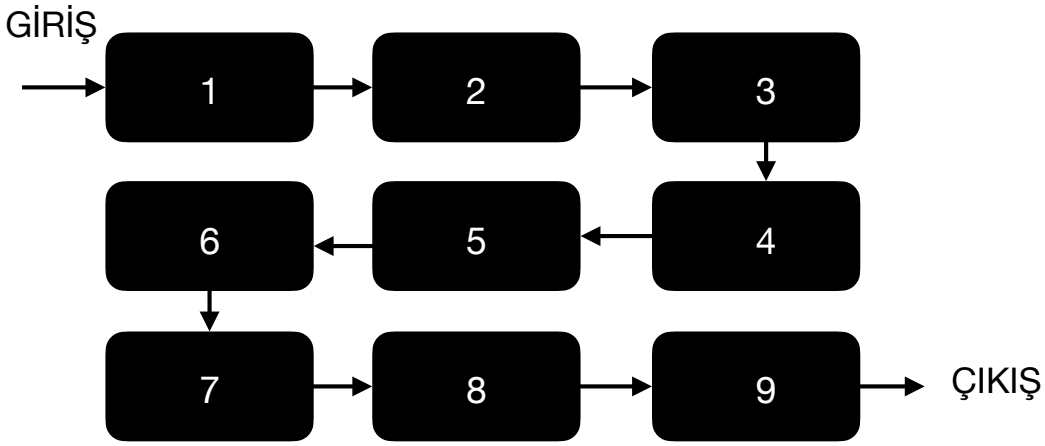
Mikroservis mimariler, monolitik yapılardan farklı olarak birbirinden bağımsız yazılım servislerine sahip olan, temel olarak bir yazılım uygulamasında belirli özellik ya da fonksiyonu sağlayan tek bir amaca hizmet eden servislerdir. Mikroservis mimarisindeki her hizmet ayrı ayrı olarak bağımsız bir şekilde bakımı yapılabilir ve dağıtılabilir yapıya sahip olur. Bu önemli özelliği sayesinde de son zamanlarda çok tercih edilen bir sistem olmuştur. Mikroservis mimari temelinde SOA yaklaşımından ortaya çıkmış, fakat bu yaklaşımın daha geliştirilmiş mimari halidir. SOA' dan farklı olarak mikroservis mimarideki servisler daha küçüktür. Bu küçük mimari yapıların her biri kendi sorumluluk alanındaki işleri gerçekleştirir, esnek ve ölçeklendirilebilir bir yapıya sahiplerdir (Lewis & Fowler, 2014). Aynı zamanda SOA bağımsız dağıtım birimleri ve ilgili oluşan sonuçlara odaklanmaz sadece işletmeler arasında iletişim için kullanılan bir yaklaşımdır.

SOA ve Mikroservis mimarisi arsındaki farklara göz atılacak olunursa, en belirgin fark mikroservis mimarisinde her hizmetin kendine ait veri tabanının mevcut olmasıdır. Farklı bir deyişle, Servis Odaklı Mimari'de görülen veri tabanı paylaşımı, mikroservis mimarisinde görülmemektedir. Mikroservis mimarisinde hizmet için gereken veri hizmetin kendine ait veri tabanından sağlanmaktadır.

Terim olarak 'Mikro' yanıltıcıdır çünkü çoğu geliştirici yapılan hizmetli mümkün olduğunca küçük hale getirmeye eğilim gösterirler. SOA bağlamında hizmetler genellikle birkaç düzine işlem desteğiyle monolitik küreler olarak uygulanır dolayısıyla SOA benzeri hizmetleri mikro hizmetler olarak yeniden tasarlamak geliştiriciye mikro hizmet mimarisi kadar olanak sağlamayacaktır. 'Mikro Hizmetler' terimi ise ilk olarak 2011 yılında modellenmesi gereken işlevler için uygulamak ve uygulanan işlevleri ideal bir şekilde mesajlar yoluyla ileten bağımsız bir süreci tanıtmıştır (Dragoni, Giallorenzo, Lafuente, Mazzara, Montesi, Mustafin & Safina, 2016).

Mikroservis yapısındaki mikro hizmetler yapıda seri bir şekilde çalıştırılması gerekmez. Mikro hizmetlerle, veriler kolayca kopyalanabilir ve daha sonra ver hattının

bir parçası olarak farklı bir konuma dağıtılabilir. Bu özelliği sayesinde mimari, birden çok yol alabilir ve Şekil 2.4'de de gösterildiği gibi farklı şekillerde işlenebilir. Bahsedilen özellik, kullanıcıya veri hattını nasıl tanımlayacağı konusunda çok esneklik sağlarken aynı zamanda akış içerisinde daha fazla çıktı elde etmesini sağlar. Mimarinin genişletilmesi gerektiği durumlar için ise çok daha basit uygulama kolaylığı elde edilmiş olunur.



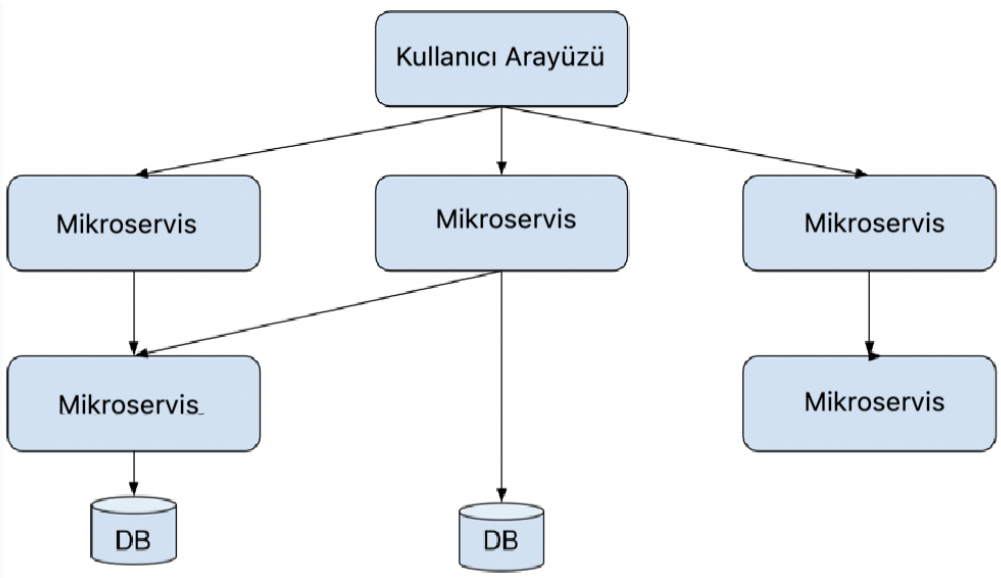
Şekil 2.4. Mikroservis mimarinin farklı şekilde gösterilmesi.

Mikroservis mimarisinin yaygınlaşmasıyla birçok teknoloji şirketi bu mimariyi kullanarak çözümler bulmaya başlamıştır. Son yıllarda geliştirilen önde gelen yazılım firmalarına bakıldığında, şirketlerin genel olarak daha üretken olmasına ve daha gelişmiş ürünler ortaya çıkarması için mikroservis mimari yapıları ile geliştirilen yaklaşımlar çok yararlıdır. Geleneksel yazılım işleri yapan birçok kuruluş da bu mimari yapısını desteklemiş ve sistemlerini bu yapıya göre düzenlemeye başlamıştır (Dragoni, Giallorenzo, Lafuente, Mazzara, Montesi, Mustafin & Safina, 2016).

Mikroservis mimarisindeki yapıların her biri kendi süreçlerinde çalışır ve genelde bir HTTP kaynak API olan mekanizmalarla iletişim kurar. Farklı programlama dilleriyle yazılabilen bu küçük küçük hizmetler aynı zamanda farklı veri depolama teknolojilerini de kullanabilirler (Lewis & Fowler, 2014). Mikroservis mimarisi düşünüldüğünde aslında

bütün halinde geliştirilmesi mümkün olan bir uygulamanın küçük farklı bölümlere ayrılarak daha kolay geliştirilmesinin sağlanması olarak açıklanabilir.

Bir mikro hizmet mimarisinin stili, Şekil 2.5.'de gösterildiği gibi bağımsız, küçük ama tek amaçlı hizmete uygun bir uygulama geliştirir. Her hizmet, diğer dağıtılmış mimarilerden daha küçüktür ve tek işleve odaklanır (Ford & Richards, 2020). Bu hizmetler gevşek bir şekilde birleştirilmiş, iyi tanımlanmış bir arabirim aracılığı ile hafif protokoller üzerinde gerçekleşir.



Şekil 2.5. Mikroservis mimari örneği.

Yazılım uygulamaları genellikle istemci tarafı olan kullanıcı birimi, veri tabanı ve sunucu olmak üzere üç ana bölümden oluşmaktadır. Sistemde yapılması istenen değişiklikler sunucunun yeni bir sürümünü oluşturmayı ve dağıtmayı içerir. Monolitik uygulamalarda yapılan küçük bir değişiklik sistemin tamamındaki monolitlerin yeniden oluşturulmasını gerektirir, fakat mikroservis yapıları kendi başlarına farklı yapılar oldukları için yapılan değişiklikler sistemin tümünü etkilemez.

Mikroservis hizmetinin kullanılmasının amacı birbirinden izole edilmiş otonom birimleri kullanmak ve bunları dağıtılmış bir yapıya entegre etmektir. Buna örnek olarak konteyner tabanlı teknoloji olan Dockler verilebilir. Mikroservis mimarisinin

kullanılması için, sistemde bir değişikliğin uygulanması ile bu yapılan değişikliğin kullanıcı ortamına aktarılması arasındaki süreyi azaltan DevOps gibi çevik uygulamaların da benimsenmesi gerekmektedir (Aderaldo, Mendonc, Pahl & Jamshidi, 2017).

Mikroservis mimarisi, kendine ait avantajları sayesinde yazılım geliştiriciler tarafından aranan bir çözüm halini almıştır.

2.2.1. Teknolojik Açıdan Mikroservis Mimarisi

Mikroservis hizmetlerin dağıtımı, sunucuyu aşırı yüklemeden hızlı erişim sağlayan birkaç hizmet halindedir. Mikro hizmetler, birbiriyle iletişim kuran birçok küçük uygulamaya sahip özel bir yapıya sahiptir. Bu uygulamalar farklı teknoloji ve programlama dilleri ile geliştirilebilir, böylece geliştiricilerde değişim yapılması durumunda uygulama kolaylıkta ölçeklendirilebilir (Jarman, 2017).

Mikroservis sisteminin yapısına bakıldığında küçük hizmetlere ayrılmış ve kendi kendini yönetebilme üzerine kurulu bir sistem ortaya çıkar. Mikroservis mimarisi Servis Platformu bulutları için bir uygulama mekanizması olarak görev yapabilecek ve bulutların içerisinde sanallaştırılmayı sağlayabilecek bulut bilişiminde konteyner teknolojisine paralel olarak ortaya çıkmıştır. İdeal bir mikroservisin bulut yoluyla paketlenip dağıtıldığı bu bilgiden yola çıkılarak söylenebilir (Söylemez & Tarhan, 2018). Bulut sistemlerinde mikroservis kullanımının yaygınlaşmasındaki ana sebeplerden biri yazılımın dağıtımının ve güncellenmesinin diğer mimari hizmetlere göre daha kolay ve daha az maliyetli olmasıdır. Mikroservis yapısının temel amacı uygulamadaki karmaşıklığın mümkün olduğunca azaltılması ve her biri birbirinden bağımsız halde tasarlanmış deploye edebilen hesaplama birimlerine dağıtılmasıdır. Bahsedilen küçük hizmetler de birbirlerine gevşek bağlar ile bağlıdırlar.

Kurulacak sistemi, mikroservis mimarisi hizmetleri bileşimiyle oluşturmak her hizmette farklı teknolojilerin kullanılmasını sağlar. Sistemin bütününü küçük etki alanı olan parçalara bölmek performans iyileştirme ve daha hızlı geliştirme adına verimi teknolojilerin seçilmesine olanak sağlar. Sistemin farklı bölümlerinin teknolojik açıdan farklı beklentileri olabilir. Örnek olarak yapay zekâ işlemini tanımlamak için programlama dili ve belge tabanlı veri tabanı olarak Python kullanmak mantıklı olacaktır. Diğer yandan aynı mimari içerisinde farkı veri tabanının kullanılması gereken bir işlem

de olabilir. Bu hizmeti geliştirmek için programlama dili olarak Java veya C# kullanılabilir (Montesi & Weber, 2016). Mikro hizmetler içerisinde barındırdığı küçük bileşenler sayesinde teknolojideki yeni gelişimlerin sistemi nasıl etkilediğinin görülmesini ve yeni teknolojilerin sistemlere uygulanmasını sağlar. Mikroservis mimarilerinin çok dilli yapısı sayesinde sistemin tamamını etkilemeden yeni programlama dilleri ve yeni veri tabanları deneyebilir ve gelişimleri gözlemlenebilir.

Mikroservis hizmetleri ve veriler arasındaki dağıtım karmaşık olup, ara katman yazılımı çözümleri kullanılarak bu karmaşıklıkla başa çıkılır. Mikroservis gibi karmaşık yapıların güvenlik açısından hassas ve zayıf noktaları ortaya çıkabilir. Bir mimarinin içerisinde çok fazla küçük yapının bulunması birçok taraftan arıza ve saldırıların olabileceği anlamına gelir.

2.3. Monolitik Problemlerin Mikroservis ile Çözümü

Son yıllarda monolitik mimari ile yapılan çalışmalarda meydana gelen sorunların üstesinden gelebilmek için mikro hizmet yapıları orta çıkmıştır. Monolitik sistemlerde, kullanılan tüm bileşenler yapı içerisinde tek bir ünitede birbirine bağlı şekilde tasarlanır. Sürekli büyüyen monolitik mimari ile tasarlanmış bir sistemde uygulamanın test edilmesi ve geliştirilmesi çok zordur. Aynı zamanda monolitik sistemler düzgün ölçeklenememek ve modülerliği azaltmak gibi dezavantajlara sahip olduğu için bu sistemlerin mikroservis mimarisi ile çözümünün yapılması gerekebilir (Namiot, Dmitry & Manfred, 2014).

Monolitik mimaride genelde paylaşılan bileşenlerin sayısı çoktur faka bu durum mikroservis mimarisine geçildiğinde minimumuma indirgenir. Mikro hizmetler bilindiği üzere bağımsız ve gevşek bağlı bileşenlerin bütünü olduğundan diğer bileşenler etkilenmeden ayrı ayrı dağıtım işlemi gerçekleştirilebilir. Hizmetlerin birbirine bağlı olmaması mikroservisteki güncelleme ve test sorunlarını da ortadan kaldırır.

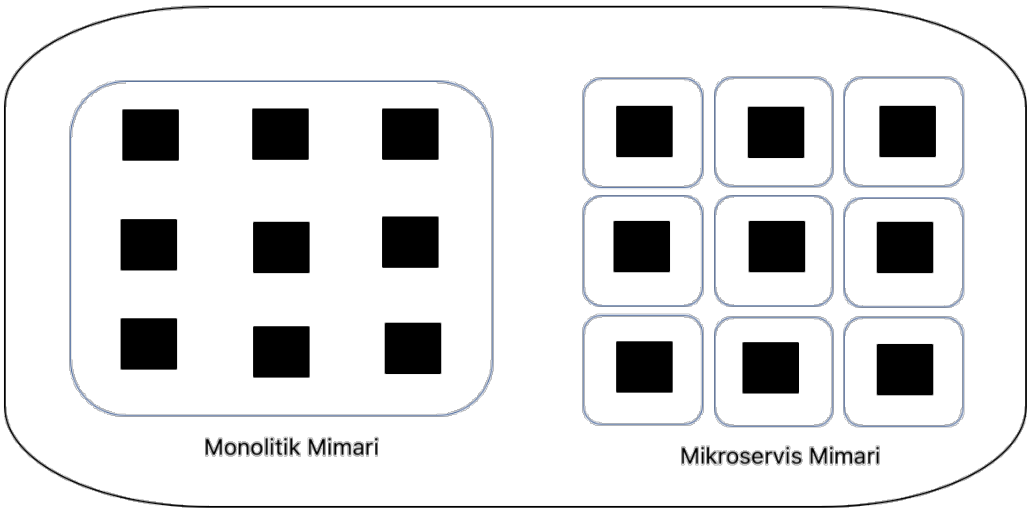
Mikroservis yapıları yazılım ekibinin kontrol edemedikleri sisteme giden tüm yolların kapatılmasına olanak sağlar. Yazılım ekibi veri tabanları ve bağlantı nokralarını yani tüm altyapıyı Docker ile kontrol ettiği için, istenmeyen tüm uygulama erişimleri ortadan kaldırılabilir. Banka ve benzeri uygulamalar işin bu olanak doğrudan veri tabanı sorunlarının ortadan kalkmasını sağlar. Monolitik mimariden mikroservis

mimariye geçişte yazılım ekibi tek bir teknolojiye bağlı kalmaktan kurtulur çok dilli bir mimari yapı ile uygulama geliştirebilir (Dragoni, Dustdar, Larsen & Mazzara, 2017).

2.4. Mikroservis ve Monolitik Mimarileri Arasındaki Farklar

Mikroservis hizmetler monolitik mimarinin zıt yaklaşımı olarak ortaya çıkmıştır. Uygulama geliştirmede odak noktası sadelik ve daha basit bir süreç oluşturmaksa monolitik mimari uygun olacaktır. Uygulamanın boyutu büyüdüğünde, uygulamanın geliştirme ekibinin de büyümesi gerekir. Bu durumdan sonra uygulamanın karmaşıklığı artar ve ilerleme zorlaşır. Mevcut olan geniş kod tabanının düşük anlaşılırlığı ve düşük kod kalitesi de eklendiğinde üretkenlik önemli ölçüde azalır. Bu durumda mikroservis ve monolitik mimarilerinin aralarındaki farklar birçok uygulayıcıyı monolit uygulamadan mikroservise geçmeye itmiştir. Bu durum, mikroservis mimarisinin son yıllarda popülerliğinin arttığının göstergesidir.

Çoğu teknolojik kararlarda olduğu gibi monolitik ve mikroservis mimarileri arasında seçim yapmak zorunda kalınlığında birçok faktör göz önünde bulundurulmalı ve kullanımdaki kolaylıklar, zorluklar, maliyet gibi farklı durumlar analiz edilmelidir. Bu farklar analiz edildiğinde projenin çıkarlarına göre en uygun mimari yapı seçilerek işleme başlanılır.



Şekil 2.6. Monolitik ve mikroservis mimarilerinin şekilsel gösterimi.

Geliştiriciler mikroservis mimari yapısının mevcut kullanılan bir monolitik mimariye tercih edilmesinin doğru olup olmadığını birkaç soruya cevap bularak değerlendirirler. Monolitik mimari basit uygulamalar için daha uygun olup, uygulama karmaşık ve gelişen bir yapıya sahip ise mikroservis mimarisi daha uygundur. Monolitik yapılar aşağıdaki durumlarda mikroservis yapıya dönüştürülmesi gerekir;

1. Monolitik mimari ile geliştirilmiş yapının bakımı ve mimarinin genişlettirilmesi zorluklar yaratmaya başladıysa,
2. Uzun vadede mikroservis mimarisinin kullanımı faydalar sağlayacaksa,
3. Modülerlik uygulamada önemli bir yer aldıysa
4. Yazılım ekibinin büyüyen proje ile genişlettirilmesi gerektiyse monolitik yapı mikroservis yapıya dönüştürülebilir (Dehghani, 2018).

2.5. Mikroservis Mimarisinin Avantajları

Uzun yıllar boyunca teknolojinin de gelişmesinin etkisiyle endüstride kolaylık sağlayabilen ve küçük parçalar halinde kullanılabilen yaklaşımlar arattırılmış ve geliştirilmiştir. Tek bütün halinde bulunan bir yapıda tek bir kod parçasında yapılan değişiklikler yapının tümünü etkileyeceğinden çok tercih edilmeyen sonuçlara yol açabilir. Monolitik mimariler de bu yapıda bulundukları için yapının içeriğinden tek bir satır kod değişikliği bile kodun ve yapının tamamını etkileyebilir. Mikroservis mimarisi ise bu ve bunun gibi sorunlara kesin bir çözüm oluşturmamakla birlikte. Hata ve arızaları e-minimuma indiren özellikleri barındıran mimari yapıdır.

Mikroservis mimarisi birçok alanda çeşitli alanlarda avantajlara sahiptir. Dağıtımların bağımsız oluşu, sistemlerin ölçeğini ve sağlamlığını geliştirmek için yeni modeller ve teknolojileri karşılaştırıp seçilmesine olanak tanır. Hizmetler arasında paralel olarak çalışma olanağı olduğu için geliştiriciler kendi sorun ve sistemleriyle ilgilenmeleri daha kolay olur. Mikroservis mimarileri, geliştiricilere her alanda esneklik sağlar. Uygulanan sistemde farklı programlama dillerinin, programlama stillerinin, dağıtım platformları veya veri tabanlarının kullanabilme olanağı sistemin farklı teknolojik açılardan gelişmesini sağlar.

Mikroservis mimarisi DevOps gibi tamamlayıcı yaklaşımlar sunmaktadır ve bu yapıda konteyner temelli sanallaştırma gibi önemli teknolojiler kullanılır. Bu teknolojiler

sayesinde bulut platformları da kullanılır ve mikroservislere yüksek derecede ölçeklendirilebilir, kullanılabilir ve taşınabilir bir yapı oluşturur. Bütün bu özellikler performansta yüksek derecede bir artış sağlamaktadır (Heinrich vd., 2017).

2.5.1. Ölçeklendirebilirlik

Monolitik sistemlerdeki en büyük kusurlardan biri en küçük değişikliklerin bile tüm sisteme dağıtılmasının gerekmesidir. Bu sayede sistem çok iyi ölçeklenmemektedir (Namiot, Dmitry & Manfred, 2014). Mikroservis sistemlerinde, her bir sistem bağımsız olarak ölçeklendirilebilme özelliğine sahiptir. Bu nedenle tüm uygulamanın birden ölçeklendirilmesi gerekmez, küçük parçalar halinde birleştirilmiş mimarinin her bölümü ayrı ayrı ölçeklendirilebilir. Bu durum da hem zaman hem de maliyet açısından önemli avantaj sağlar.

Sistemin büyük ve monolitik bir yapıya sahip olması sistemdeki her şeyi birlikte ele alınması gerektiğini gösterir. Büyük yapıda monolitik bir sistemde, sistem içerisinde yer alan en küçük bir yapının performans sorunlarına yol açması tüm sistemin bütünsel olarak yapılandırmasını gerektirir. Mikroservis yapısı ile ele alınmış dağıtılmış sistemlerde ise tek bir yapıdaki performans bozukluğu sadece o yapıyı baz alarak giderilmesi mümkün olan sorunlardır.

2.5.2. Yüksek Kullanılabilir Olması ve Hata Toleransı

Hizmetlerin kullanıcıya gecikmeden sunulması için yüksek kullanılabilir olma özelliği mikroservis hizmetler için çok önemli bir özelliktir. Mikroservis mimarisi sayesinde kullanıcıya hizmetler gecikmeden sunulur. Mikro hizmetler, gevşek bağlantı ile karakterize edildiği için kullanıcı uygulamanın hata gösterdiği durumda bile yazılım ürünü ile etkileşim halinde olabilir ve ürünü kullanabilir. Aynı zamanda mikroservis mimarisi ile kurulmuş bir sistem hataya dayanıklı yazılım sistemi olarak geliştirilir. Örnek olarak bir web platformu yüzlerce isteği aynı anda işlemesi gerektiğinde bile ağır yükler halinde mikroservis yapısından dolayı işleyebilir. Platformun sorunsuz bir şekilde işlemesiyle kullanıcı istediği işlemi kolayca gerçekleştirebilir.

Mikroservis mimarisi yapısı gereği esnek ve hızlı bir yapıya sahiptir. Hata toleransı sayesinde yapı içerisinde oluşan hatalar bu mekanizma sayesinde çok çabuk

tespit edilir ve konteyner izolasyonu şeklinde servisin diğer bölümlerine etki etmez. Bu yüzden de monitoring yani izleme anlamına gelen sistem kurularak mikroservis yapılarında meydana gelen hatalar çok çabuk farkına varılır ve ortadan kaldırılır (Hasselbring, 2016).

2.5.3. Dağıtımın Kolaylaştırılmış Olması

Mikroservis yapısından da anlaşılacağı üzere yapının küçük küçük bileşenlerden oluşmasından dolayı her servis ayrı şekilde dağıtılır ve servisler dağıtırken büyük sistem hala çalışmaya devam eder. Büyük firmaların yapılarına bakıldığı zaman (örneğin; Amazon) yapılarının mikroservis mimari ile oluşturulduğu görülür. Mikroservis mimari ile oluşturulmuş yapının en önemli avantajı da bu gibi büyük şirketlerin yeniş gelişmelerinde kesintisiz şekilde hizmet sağlamalarıdır. Mikroservis mimarisinin kullanılmadığı durumlar için, yazılımda tek bir satırda yapılan değişiklikler bile sistemdeki diğer tüm kodlara etki edeceği için bu kabul edilemez zorlukları ortaya çıkarır. Bu durum da uygulama için büyük risk oluşturur.

2.5.4. Farklı Teknolojilerle Kullanılabilirliği

Teknolojik yapılandırmalarda, ekipler geliştirdikleri servislerin teknolojilerini zamanla değiştirebilecekleri servis teknolojileri seçebilirler. Örnek olarak Java programlama dil ile geliştirilmiş bir sistemin yanına Python dili ile geliştirilmiş başka bir sistem eklenebilir. Mimari yapıdaki her bir sistem için farklı teknoloji veya veri tabanı kullanılabilir.

Mikroservisler ile yeni teknolojiler daha kolay bir şekilde benimseneceği gibi aynı zamanda bu teknolojilerin projeye nasıl etki edeceğini de görebiliriz. Monolitik sistemlerde sistemin bir parçasının farklı bir programlama dili, veri tabanı veya framework denenmesi istenirse yapılacak herhangi bir değişiklik tüm sistemi olumsuz şekilde etkileyebilir.

2.6. Mikroservis Mimarisinin Dezavantajları

Monolitik mimari ile karşılaştırıldığında mikroservis yapıları daha karmaşık bir yapıya sahiptir. Mikroservis yapının ne kadar karmaşık olduğu yazılım servisi tarafından

kullanılan mikro hizmetler ile doğru orantılıdır. Bu hizmetler ne kadar artarsa karmaşıklık o kadar artar.

Mikroservis mimarisinde çok sayıda servisin olması yazılım uygulaması için büyük güvenlik endişesi oluşturur. Mikroservis mimarisi kullanılarak oluşturulan yazılımların test edilmesi daha karmaşık bir süreç haline gelebilir. Çok sayıda hizmet söz konusu olduğundan dolayı bu sistemleri test etmek ve yönetmek zor bit süreç olabilir (Rahman & Gao, 2015). Bir hizmeti benzer bir test için, o hizmeti ve bağlı olduğu hizmetleri başlatmanız gerekir. En azından test edilmesi gereken hizmetin bağlı olduğu hizmetleri için taslaklar yapılandırılması gerekir. Birden çok hizmeti kapsayan değişiklikleri uygulamak daha zordur. Bir mikro hizmet mimarisinde hizmetlerin her birine değişikliğin uygulanması ve bu değişiklik ile kullanıma sunulması için yapılacak işlemin dikkatli bir şekilde planlanması ve uygulanması gerekir.

2.7. Mikroservis Mimarisinin Güvenilirliği

Mikroservis sistemlerinde bölümler arasındaki iletişim artık monolitik sistemlerdeki gibi işlev çağrıları yerine ağ çağrıları şeklinde gönderilmeye başlar. Bu durum da mikroservis sistemleri için güvenlik açıkları oluşturur. Güvenlik riskini ortadan kaldırmak için önlemler uygulanmazsa uygulama kolay bir şekilde kesilebilir ve hata riski artabilir.

Mikro hizmetler üstesinden gelinmesi gereken çok sayıda güvenlik sorunlarına sahiptir. Mikroservisler ile oluşturulmuş basit bir uygulama birçok kişinin kullandığı geniş bir uygulama alanına dönüştüğünde güvenlik açıkları da artar.

Bulut sistemine geçerken verilen kararlar her zaman sorgulanmalıdır. Sistemin boyutu ve kalitesi ne olursa olsun nasıl güvenilir bir yapıya sahip olduğu önemlidir. Yani sadece büyük sistemlerde değil daha küçük ölçekli sistemlerde de güvenlik önemli bir unsurdur. Belirli bir uygulamanın alt etki alanının güvenliği ihlal edilirse ve bu etki alanına herhangi bir içerik sunulursa kullanıcıların kimlik veya verilerine ulaşılabilir (Sun, Nanda & Jaeger, 2015). Güvenli geliştirme, bir geliştirme ekibinin en çok çaba ve özveri göstermesi gereken alandır.

2.8. Monolitik Mimariden Mikroservis Mimariye Taşınma Süreci

Monolitik yapı ile geliştirilmiş bir yazılımdan mikroservis yapıya geçiş hızlı bir şekilde gerçekleştirilemez. Mevcut bir monolitik sistemi, mikroservis mimariye dönüştürmek birçok yenileme ve yüksek milliyet gerektirir (Fowler, 2002). Mikroservis mimari yapılar yeni bir mimari tarz olduğu için eski monolit yapıdan yeni yapıya geçişin tek bir yolu yoktur, birkaç farklı yol ile monolitik mimariden mikroservis mimariye taşınma planlanabilir (Mishra, Kunde & Nambiar, 2018).

Mikroservis hizmetleri sıkı bağlantıya ve çok sayıda bağımlılığa sahip monolitlerin aksine tüm uygulamanın yeniden dağıtılmasına gerek kalmadan bağımsız olarak dağıtılabilir. Bu şekilde geliştirme ve dağıtma, esnek olmayan mimarilerden kaçarak tasarım ilkelerine uygun bir şekilde mikroservis mimarisi geliştirilerek etkin kullanım fırsatları yaratılabilir. Mikroservis mimarisi, daha hızlı test edilip geliştirmeye hazır hale getirilerek daha yüksek bir dağıtım oranı sağlanır. Çevik metodoloji ve mikroservisin avantajları birleşerek ürün geliştirme süresi azaları ve yenilikçi ürünlerin pazara çıkması kolaylaşır (Blanch, 2017).

Geçiş süresindeki en önemli zorluk ise monolitik hizmetler ile tasarlanmış kod tabanından mikro yapıya geçişin sağlanmasıdır. Monolitik mimariden mikroservis mimariye geçiş tasarlandığı zaman iki farklı yapıdan bahsedilebilir:

1. Yeniden inşa (rebuilding)

Tüm yapının monolitik mimariden, mikroservis mimariye taşınması bazen uygulamayı düzenlemek yerine yeniden oluşturarak yapılması daha az maliyetli ve hızlı bir işlemdir (Linthicum, 2017). Fakat çok eski diller ile yazılmış uygulamaların ya da kötü bir tasarıma sahip uygulamaların yeniden aynı şekilde oluşturulması önerilmez. Aynı zamanda eski bir veri tabanına sahip ve veri tabanına sıkıca bağlı uygulamaların yeniden oluşturulması önerilmeyen durumlardandır.

2. Yeniden düzenleme (refactoring)

Eski uygulamanın modernizasyonu ile eski uygulamayı mikro hizmetlere bölerek tek bir platformda bağlamak amacıyla uygulama yeniden düzenlenebilir (Linthicum, 2017). Monolit hizmetleri mikro hizmetlere ayırmanın birçok farklı yolu vardır. Kullanılan her bir yolun avantajları ve dezavantajları mevcuttur. Bu yöntemlerden

bazıları çok daha genel birçok uygulamada kullanılabilirken bazıları ise spesifik olarak sadece birkaç uygulama için kullanılabilir.

Alessandra Levcovitz ve arkadaşları tarafından 2015 yılında monolitik sistemlerin mikroservis sistemlere taşınması teknik tanımlanmıştır. Bu teknik aşağıdaki adımlardan oluşmaktadır; (Levcovitz, Terra & Valente, 2015)

- Veri tabanı ayrıştırma
- İş metotları ve veri tabanları arasında bağımlılık grafiğinin oluşturulması
- Bağımlılık grafiğine bağlı olarak veri tabanı tablolarının eşleştirilmesi, alt sistemlerin eşleştirilmesi
- Mikro hizmetlere dönüştürülecek yapılar belirlenir, bu yapılardan ver tabanından giden yoldaki kodları incelenir
- Mikro hizmetlere geçiş için API ağ geçişleri oluşturulur, bu ağ geçişi istemci ve sunucu tarafı arasında bir ara katman oluşturur.

Oluşturulan sistem yardımıyla, tüm alt sistemler tanımlanabilir ve sınıflandırılabilir. Aynı zamanda yalnızca veri tabanı tabloları değerlendirilerek kaynak kodlarına erişilebilir ve bağımlılık grafiği oluşturulup analiz gerçekleştirilebilir.

Rui Chen ve arkadaşları tarafından 2017 yılında ise veri akışına dayalı bir algoritma geliştirerek bir yaklaşım önermişlerdir. Bu yaklaşım aşağıdaki üç adımdan oluşmaktadır; (Chen & Liand, 2017)

- Yazılım uzmanları iş gereksinimi analizleri yapar ve iş mantığıyla ayrıntılı bir veri akış diyagramı oluşturur
- Algoritmada aynı tür çıktılar ve aynı işlemler sanal bir veri akışı ile birleştirilir
- Algoritmada tanımlanan mikro hizmet adayları için ayrı ayrı modüller çıkartılır.

Veri akışına dayalı olan bu yaklaşımda, bir iş mantığı içinde veri işlemi açısından en ayrıntılı mikro hizmetler bulunur. Önerilen yaklaşım büyük ölçekli sistemlerde sıkça kullanılmış ve verimli sonuçlar elde edilmiştir.

Holger Knoche ve arkadaşları tarafından önerilen yaklaşım ise beş adımdan oluşmaktadır. Bu yaklaşımda mikro hizmetlere uygulamanın ayrılması için geçici bir süreç önerilmiştir. Modernizasyon süreci aşağıdaki aşamalardan oluşur; (Knoche & Hasselbring, 2018)

- İstemci sistemlerinin gerektirdiği işlevselliği iyi tanımlamış bir harici hizmet cephesi tanımlama
- Hizmet cephesinin uyarlanması
- Yeni oluşturulan harici hizmet cephesi ara yüzlerinin kullanılmaya başlanması
- Uygulamaların dahili olarak yeniden yapılandırılması
- Hizmet uygulamalarının mikro hizmetlere değiştirilmesi – monolitik hizmetlerin yerini mikro hizmetler alır.

Bu yaklaşımın avantajı uygulamanın temel etki alanına dayalı platformdan bağımsız arabirimler oluşturmaktır. Bu sayede uygulamanın geliştirilebilirliği iyileşmektedir.

Zhamak Dehghani tarafından ise monolitik sistemden mikroservise geçiş için aşağıdaki noktalardan oluşan bir sistem önerildi; (Chen & Liand, 2017)

- Yapının monolite olan bağımlılığı en aza indirgenir
- Birbirine bağlı yapılar ayrılır
- Ayrıma dikey olarak gerçekleştirilir
- Gerekli iş için neyin önemli olduğuna karar verilir
- Yetki ayrılır, kod değil
- Önce makro yapıya sonra da mikro yapıya geçilir.

Her bir monolit uygulama kendisine özel karmaşıklık, teknoloji yığını, ekip boyutu ve becerileri gibi farklılıkları vardır. Bu yüzden, monolitten mikro hizmetlere geçmenin en iyi ve tek bir yolu yoktur. Monolitik mimariye oluşturulmuş bir yapıdan mikroservis mimariye geçiş çok kolay bir iş değildir. Geçişin sağlanması için yeniden düzenlemek mi yoksa yeniden oluşturmak mı gerektiği sorulmalı ve cevap yönünde devam edilmelidir. Her kuruluşun kendine ait nedenleri ve hedefleri vardır bundan dolayı monolit hizmetten mikroservise geçişte bu hedeflere odaklanılmaktadır.

Mikroservis hizmetler ile geliştirilecek sistem için teknoloji seçimi söz konusu olduğunda çeviklik, daha kısa geliştirme süresi, ölçeklendirebilir olması ve esneklik sunar fakat bu sistemin yeni yeteneklerden tamamıyla faydalanması için kuruluş kaynaklarının uyarılması gerekir. Monolitik mimariye sahip kuruluşlar yeni bileşenler geliştirmek için genel olarak plan odaklı geliştirme sistemini kullanır. Bu bileşenler büyüyebilir ve

birbirleriyle birleştirilebilir. Büyük bileşenlerde yapılan herhangi bir değişiklik uygulamanın yeniden başlatılması ve yeniden test edilmesi gerekir. Sistemde yapılan entegrasyon testleri ve kullanıcı kabul testleri çok fazla zaman harcanmasını sağlar. Megargel & Shankararaman tarafından yapılan araştırmada ise geliştirme sırasında ortaya çıkan sorunların %80-90'ı yanlış test edilen değişikliklerden kaynaklanıyor (Megargel & Shankararaman, 2020). Araştırmada ise bunun nedeninin sıkı bağlantılı büyük sistemler olduğu açıklanmıştır. Kritik sorunların oluşmaması için büyük kuruluşlar (bankalar gibi), monolitik sistemlerinin yeniden koşullandırmasını ayda bir kez yaparlar. Bu kuruluşlarda kaynakların bu şekilde çalışmasıyla verimli bir sonuç elde edilemez ve ortama yayınlanma süresi oldukça uzar.

Mikroservis mimarisi yalnızca bireysel mikro hizmetler oluşturmak için değil aynı zamanda bu hizmetler arasında iletişim sürecini mümkün olduğunca güvenilir ve hatasız hale getirmektedir. Bunu başarmak için ise servis kaydı gereklidir. Servis kaydı, servisleri kaydeden ve diğer hizmetler arasında aranmasına ve bağlanmasına izin veren merkez depo olarak tanımlanabilir.

Mikroservis mimarisini oluşturmanın temel ilkeleri şunlardır;

İş kavramları etrafında modellenmiş servis – mikro hizmetler sınırlı bağlamlar etrafında yapılandırılmaktadır.

Bağımsız dağıtılabilir olması – her mikro hizmet bağımsız olarak dağıtılabilir olması gereklidir. Bu durum ekiplerin bağımsız olarak da çalışmalarına olanak sağlamaktadır. Ekip çalışanları üretim dağıtımlarını sürekli olarak düzenlemek durumunda kalmayacağı için daha az zamanda daha fazla iş yaparlar.

Gizli uygulama ayrıntılarının bilinmemesi – mikroservis işlemlerde veri tabanı da dahil olmak üzere uygulama ayrıntıları güvenlidir ve gizlenmektedir.

Tam otomasyona öncelik tanıması – otomatikleştirilmiş testler mikroservislerin en önemli kullanım alanıdır. Tam otomasyon ve sürekli teslimat olmadan altyapının kontrol edilmesi oldukça zor ve zaman alan bir işlemdir.

Arızaların kolay bulunabilir hale getirilmesi – her servis için oluşabilecek arızalar vardır ve bunlar için plan yapılıp, servis çağrılarının bazen başarısız olabileceğini

varsayılmalıdır. Zaman aşımının makul olduğu durumlarda bile servis arızalarının kolay bulunabilmesi ve müdahale edilmesi gerekir.

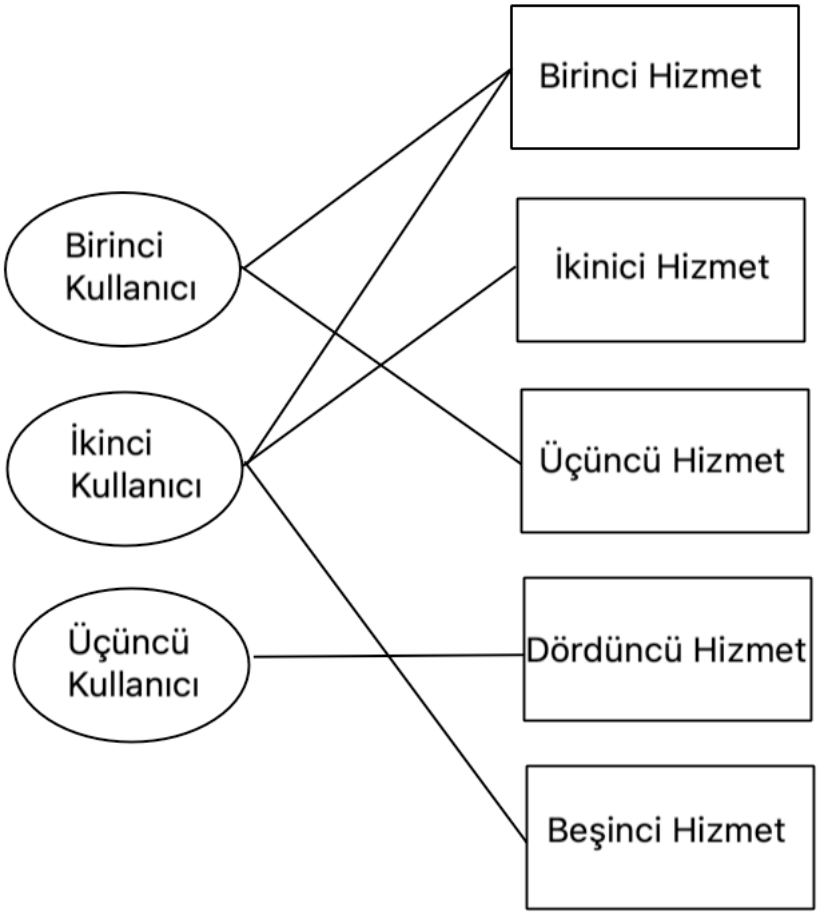
İzleme – sistemin düzgün çalıştığını anlamak için sistemin bütününü değil de tek tek servislerinin çalışıp çalışmadığını izlemek daha yararlı olacaktır.

2.9. Mikroservis Haberleşme Mekanizmaları

Uygulama ve kullanıcı arasında mikro hizmetler arasındaki iletişim, monolitik sistemlere göre farklılıklar göstermektedir. Bu iletişim ve haberleşme mekanizmalarını birkaç ana başlık altında toplayabilir ve kullanılan haberleşme biçimlerini aşağıdaki noktalar ile açıklayabiliriz.

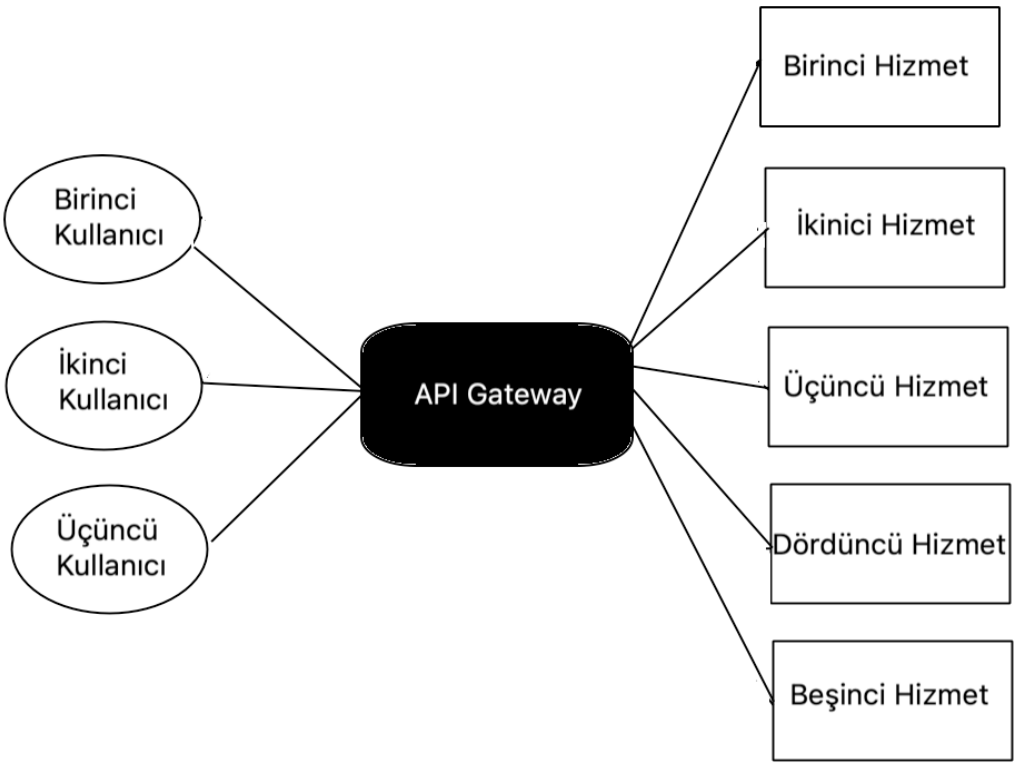
2.9.1. API Gateway

Monolitik olarak, istemci ve uygulama sistemleri, uygulamanın aynı örneğine bir yük dengeleyici kullanarak HTTP istekleri göndermektedir. Mikro hizmetlerde ise hizmet kümesi monolitlerin yerini alır. Şekil 2.7.'de görüldüğü gibi farklı kullanıcılardan farklı hizmetler için bir veya birden fazla istekler gönderilir. Bu istekler Şekil 2.7.'de RESTful HTTP hizmetleri olarak gönderilmektedir.



Şekil 2.7. Farklı kullanıcılardan uygulamaya gönderilen istekler.

Fakat aynı durum için Şekil 2.8.'ye bakıldığında kullanılan API Gateway ile istemciler tarafından istenen veriler arasında önemli bir düzeyde farklılık görülür. İlk örnekte tek bir sayfayı görüntülemek için yüzlerce servisin aranması gerekebilir bu da performansı düşürür. Bu bant genişliğini önlemek için istemci sayfa başına az istek yapmalıdır bu durum da yaklaşımı destekleyen API ile gerçekleştirilir. Bu yaklaşımda, istemciler ve mikro hizmetler arasına API ağ geçidi kurulur. Bu sayede kullanıcılar bekleme veya performans düşüklüğü yaşamadan istedikleri sayfaya ulaşabilirler.



Şekil 2.8. API Getaway kullanarak kullanıcıların hizmetlere ulaşımının sağlanması.

API ağ geçidi istemcilerden gelen isteklere işlemek için yüksek performanslı ağa sahip LAN kullanmaktadır. Bu sayede farklı hizmetlerle yapılan istekler mikroservis mimarisinde kullanılabilir hale gelirler. Ayrıca hizmetlerde yapılan değişiklikler yalnızca API güncellenmesiyle yapılabilir (Pachghare, 2016).

2.9.2. Servisler Arası Haberleşme Mekanizmaları

Mikroservisler monolitik hizmetlerden farklı olarak yapısında çalışan farklı servisler vardır. Mikroservis yapısında servisler arasında iletişim için IPC kullanılır. Bir servis için servisler arasındaki iletişim mekanizmasını seçmeden önce servislerin etkileşiminin bilinmesi gerekir. İstemci ve servisler birbirleriyle bire bir ya da bire çok şeklinde iletişime girebilirler. Bire bir mekanizmalarda istemciden gelen istek bir örnek tarafından işlenirken bire çok mekanizmalarda birden çok örnek tarafından işlenmektedir.

Diğer bir haberleşme mekanizması ise Senkronize HTTP veya Asenkrondur. Bu mekanizmada, IPC için REST gibi HTTP tabanlı senkron bir iletişim yaklaşımı kullanılır.

Ağ üzerinde kolayca çalışabilir bir yapıya sahiptir. HTTP, REST tarafından URL kullanarak kaynakları değiştirmek için kullanılır.

2.9.3. Asenkron Mesaj Tabanlı İletişim

Bu sistemde, süreçler birbirleriyle asenkron olarak mesaj alışverişi yaparak iletişim kurmaktadırlar. İstemci tarafından hizmet için yapılan istekte hizmet istemciye ayrı bir mesaj gönderir. Sunucu yanıt beklemeden asenkron iletişim yöntemini kullandığından, istemci yanıtın daha sonra geleceğini varsayarak mesaj iletmeye devam eder. Kanallar mesaj alışverişi için kullanılır. Noktadan noktaya bir kanal, bir mesajı kanaldaki yalnızca bir alıcıya dağıtır (Pachghare, 2016).

BÖLÜM 3

MİKROSERVİS BİLGİ İŞLEM PLATFORMLARI

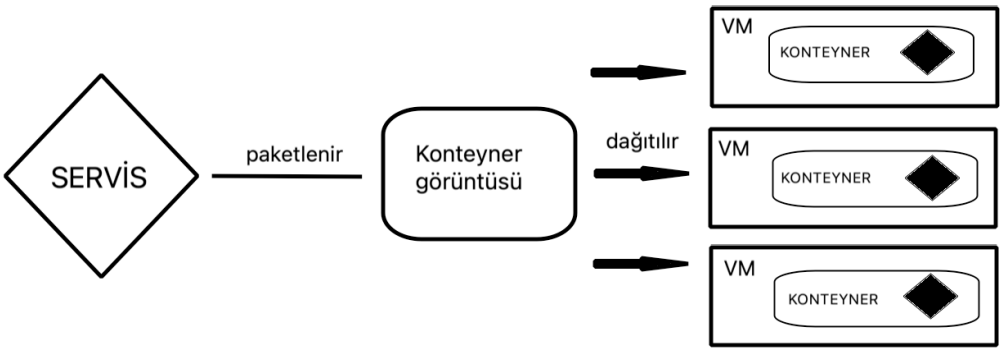
3.1. Konteyner Tabanlı Mikroservisler

Yeni teknolojinin gelişmesiyle yeni yaklaşımlar ile veri merkezli monolitik yapılardan daha kolay ölçeklenebilir, esnek ve kolay değişim yapılabilen yeni yapı olan mikroservis yapıya yönlenilmiştir.

Mikroservis mimari yapısı açısından DevOps ve CD gibi tamamlayıcı yaklaşımlar sunmaktadır. Bu mimar yapı konteyner temelli sanallaştırma ve konteyner orkestrasyon çözümler gibi önemli deployment teknolojileri ortaya çıkarır. Bu teknolojiler bulut platformlarının da etkili bir şekilde kullanılmasına izin vermektedir. Aynı sunucuda çalışan çoklu servislerin oluşturduğu dağıtım problemini ortadan kaldırabilmek adına her sunucu için bir servis örneği metodu önerilmiştir. Çoklu servislerdeki izolasyon problemi her bir servisi kendi sunucusuyla ayrı ayrı çalıştırılarak çözülmesi amaçlanmıştır. Bahsedilen bu model, her VM başına servis örneği ve her konteyner başına servis örneği olarak iki şekilde özelleştirilip kullanılabilir.

Konteyner başına servis desenini kullandığımız zaman her hizmet örneği kendi konteynerinde çalışır. Bu durumda konteynerler işletim sistemi düzeninde bir sanallaştırma mekanizması görevini taşır. Bir konteyner, sanal alanda çalışmakta olan bir işlemden oluşabilirken birden daha fazla işlemden de oluşabilir. Konteyner teknolojisi kullanılarak servis örnekleri birbirinden ayrılıp izole edilir bu sayede konteyner tarafından tüketilen kaynaklarına kolayca erişim sağlanılır. Konteynerler esnek teknolojilerdir bundan dolayı konteyner görüntülerini oluşturmak diğer teknolojilere göre

çok daha hızlı bir işlemdir. Ayrıca uzun bir işletim sistemi önyükleme mekanizması olmadığından dolayı konteynerler çok hızlı bir şekilde başlar. Şekil 3.1.'de her konteyner için servis örneği gösterilmektedir. Bu örnekte, her konteyner kendine ait port ad alanı ve kök dosya sistemleri vardır. Ayrıca konteynerler de VM'ler gibi servislerin teknolojilerini sarmalamakta ve güvenli bir yapı haline getirmektedirler (Richordson, 2016).



Şekil 3.1. Her konteyner için bir servis örneği.

Büyük şirketler üzerinde gerçekleştirilen mikroservis hataları yanı buglar spesifik bileşenler üzerinde izole edilmeleri sağlanarak hata oranı azaltılır aynı zamanda hata ayıklama performansı basitleştirilir. Mikroservisler konteyner tabanlı model uygulayan veri merkezlerinde çok iyi uyum sağlamakta ve bu uyum sayesinde oluşturulan kaynak taleplerine uygun ölçeklendirilebilir durumdadırlar.

Mikroservis yapıları otomasyon ve hızlı elastik bir yapıda olup hata tolerans mekanizmaları sayesinde oluşan hatalar konteyner izolasyonu sayesinde diğer servisleri etkilemez, sadece hatanın olduğu servisi etkiler. Bu durum da mikroservis yapısında gerçekleşen hataların çok çabuk bulunmasına ve hemen düzeltilmesine olanak sağlar. Mikroservislerde oluşabilecek hataları minimuma indirgeyebilmek için izleme (monitoring) sisteminin kurulması gereklidir. Bu sayede bir şeyler yanlış giderse geliştiriciler çok çabuk uyarılır ve hatanın çözümü büyümeden yapılır.

3.2. Fonksiyon Tabanlı Mikroservisler

Sunucusuz dağıtım modeli olan örnek olarak verilebilecek AWS Lambda yeni bir yaklaşımdır. Bu yaklaşım Java, Node.js ve Python gibi servisleri de desteklemektedir. Mikroservislerin dağıtımı için çok uygun ve hızlı bir yöntem olan sunucusuz dağıtım yöntemi geliştiricinin uygulamayı geliştirmeye odaklanmasını sağlar. Bu teknoloji sayesinde geliştiriciler teknolojik altyapıdan sorumlu olmazlar.

Sunucusuz kaynaklar kullanılarak mikro hizmetler geliştirildiğinde sorunların çözülmesi daha kolay hale gelir ve bazı durumlarda sorunlar tamamen ortadan kaybolur. Bu mimaride, uygulama bileşenlerinin her biri ayrıştırılır ve bağımsız olarak dağıtılarak çalıştırılır (Richordson, 2016).

Günümüzde yüksek düzeyde işlemler artık bulut üzerinden gerçekleştirilmekte olup bunun en önemli sebebi kullanıcı için kaynak esnekliğine sahip olmasıdır. Kaynak esnekliğinin asıl amacı kullanıcıların talep ettikleri anda istedikleri kaynaklara ulaşabilme kolaylığının sağlanmasıdır. Bu sayede bulut hizmetleri kullanıcılara çok katlı bir altyapı sunarak maliyet etkinliği oluşturur.

Mikroservis kaynak verimliliği ve maliyet alanında yapılan çalışmalarda ise işletim sistemine göre monolitik yaklaşımda genellikle depolama ve network tarafından çıkarken mikroservis yaklaşımında API'lerden sağlanıyor.

BÖLÜM 4

MATERYAL VE METOD

4.1. Monolitik Mimariden Mikroservis Mimarisine Modüler Migrasyon

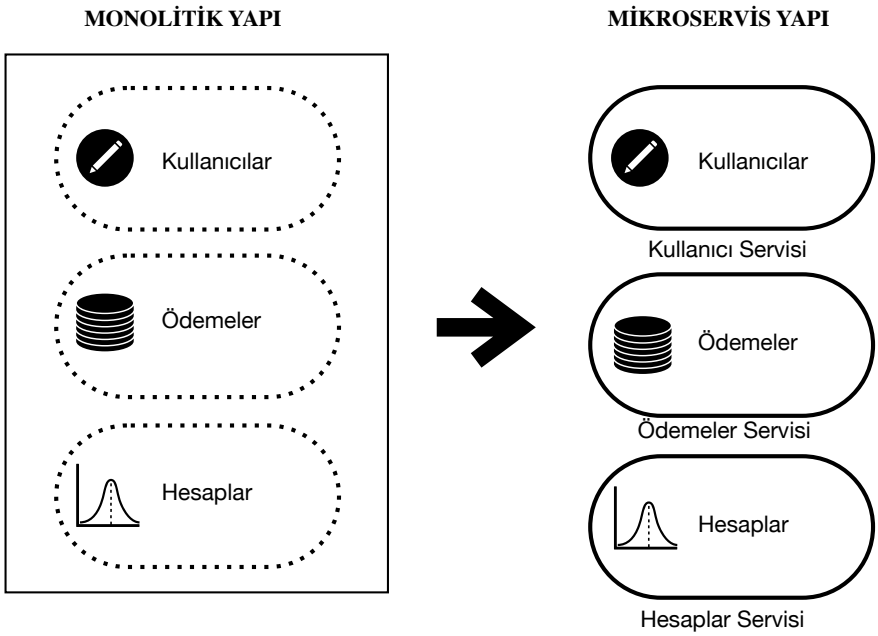
Monolitik yapıdaki yazılımlarda genellikle yazılım geliştirme, dağıtım, test ve ölçeklendirme gibi süreçler daha karmaşık ve daha yüksek maliyet getiren süreçlerdir. Monolitik mimarilerde kod tabanı genellikle mikroservise göre daha büyüktür, bu durum yapım zamanı (build time), test ve dağıtım gibi süreçleri zorlaştırır. Monolitik mimarilerde yazılım ekibi tek tabanlı kod yapısında çalışır ve tek bir yazılım diline bağlı kalır. Bu durumda yazılım yapısı bütün olarak çalıştığı için hem test aşamasında hem de yeni özelliklerin eklenmesinde yapıdaki diğer parçaların etkilenmediğinden emin olmak gerekir. Bu yüzden, bahsedilen her bir aşamanın daha kolay ve hızlı gerçekleşmesi için monolitik mimari yerine mikroservis mimarisini kullanımı yaygınlaşmıştır.

Monolitik yapıdaki bir mimariden mikroservis yapıya geçiş modül olarak ve planlayarak yapılmalıdır. Mikroservis yapıya geçişin gerçekleştirilmesinde, geçiş sürecinin etki edeceği alanlar, sistemin veya yazılımın geçiş boyunca kapalı kalmaması ve ölçeklendirebilir olması en çok dikkat edilmesi gerekilen noktalardır. Mikroservis mimarisine geçiş süreci planlanırken iş, ekip, bilgi / birikim, zorluk ve ölçeklendirilebilirdik faktörleri göz önünde bulundurularak sistemin hangi önemli parçalarının önce değerlendirilmesi gerektiğine karar verilir. Geçiş süreci planlanmasında servislerin öncekileri belirlendikten sonra, bunların departman ve süreç olarak da planlanması gerekir. Oluşturulan bu planlamalar sayesinde geliştirme, test ve altyapı gibi departmanların geçiş süreci boyunca nasıl bir yol izleyecekleri belirlenir.

Geçiş süreci tasarlanırken yapılan en büyük yanlışlardan bir tanesi sistemin baştan yazılıp oluşturulmasıdır. Bu karar işletmeleri veya projeleri zor durumda bırakacak bir durumdur. Mikroservis mimarisinde geçiş aşamasında düzgün planlama yapılmadığı takdirde işin zorluk kısmı tamamen anlaşılamayacağı gibi çoğu zaman başarısız sonuçlara varılır. Bu durum aynı zamanda maliyet ve zaman açısından da zarar görülecek sonuçlar ortaya çıkarır.

Monolitik mimarilerde kod yapısı tek tabanlı bir altyapıda geliştirildiği için bu yapıyı bir anda mikroservise taşımak hem yanlış hem de tehlikeli bir yaklaşımdır. Bu nedenle, geçişi sağlamak için modüler geçişin yapılması çok daha uygun olacaktır. Modüler geçişlerde öncelikleri belirlenen servisler, tek tek taşınarak mikroservis yapıları oluşturulur. Bu oluşturulan servisler her biri ayrı bir ortamda koşar ve ayrı ortamda test edilir.

Servislerin modüler geçiş süresi boyunca yapıdaki kullanıcılar, ödemeler, hesaplar gibi servisler ayrı ayrı incelenip taşınma süreci başlatılır. Geçiş süreci, servislerin öncelik ve önem sırasına göre planlanır. Servislerin taşınma sürecinde yazılım ekibinin bilgi sahibi olduğu yazılım dili ve teknolojileri kullanılır. Servislerin monolitik yapıdan mikroservis yapısına taşınma süresinde yapının tekrar yazılması beklenemez, fakat bu servislerde problem ve çalışma mantık hataları gözlemlenirse o zaman öncelik belirlenip planlama yapıldıktan sonra sistemler üzerinde sistemler üzerinde hangi değişikliklerin yapılması gerektiğine karar verilir.



Şekil 4.1. Monolitik yapıdan mikroservis yapıya geçiş.

Monolitik yapıdan mikroservis yapıya taşınma süresince sistemin etkileneceği yerler dikkat edilmediği takdirde büyük ve geriye dönülmeyecek yanlışlar oluşabilir. Modüler geçiş boyunca ise taşınan servislerin çalışmaması ya da düzgün test edilmemesi sadece o servisi etkiler. Bir servisin çalışmama ya da kapalı kalma durumu sistemin geri kalanını etkilemez, bu durum geçiş süresi boyunca hem zaman açısından ekip ve projeler için daha verimlidir hem de daha az maliyetlidir.

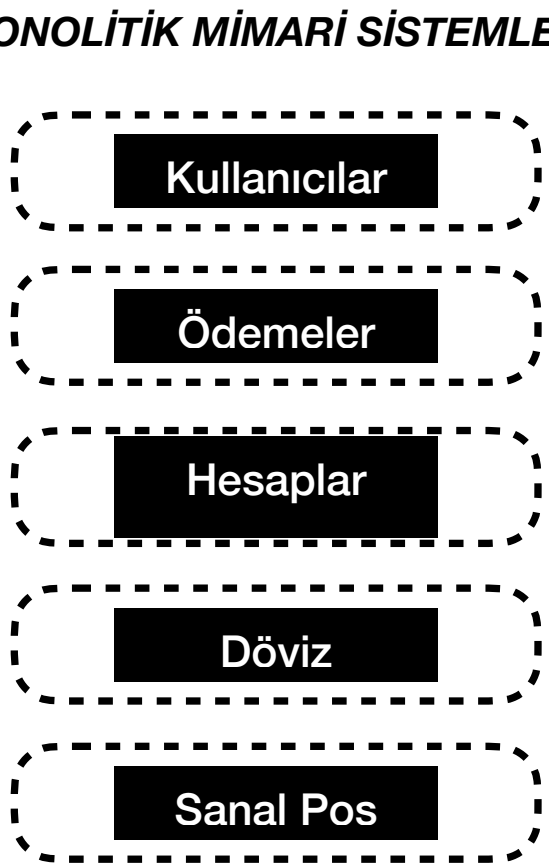
Monolitik yapıdan mikroservis yapısına geçilirken önemli parçalardan bir tanesi altyapı (infrastructure) olayıdır. Genellikle projenin ya da şirketin ihtiyacına ve bütçesine bağlı olarak hangi altyapının kullanılması gerektiği belirlenir. Altyapı belirlenirken önemli etkenlerden bir tanesi bulut tabanlarıdır. Bulut tabanlarının seçilmesi ve karar verilmesi, bulut sağlayıcıların projenin ihtiyacına ve bütçesine göre uygun olup olmadığı araştırılır. Altyapılar genellikle uzman kişiler tarafından projeye uygun olarak geliştirilir.

Modüler geçiş süreci yapıdaki sırasıyla en ilk ve en son öncelikli servislerin tek tek taşınmasıyla tamamlanır ve mikroservis yapısına geçiş sağlanır. Planlanarak yapılan geçişler genellikle ay, yıl gibi uzun sürelerde sağlanır. Geçişlerin sağlıklı ve verimli olabilmesi için ekibin buna hazır olması ve eğitim alması gereklidir.

4.2. Monolitik Mimariden Mikroservis Mimariye Geçiş Senaryosu

Bu senaryoda, monolitik yapıdaki var olan bir sistemden mikroservis yapıya geçiş süreci aşamalar ile anlatılmıştır. Yazılan senaryoda bir banka sisteminde var olan veya geliştirilen örnek servisler ele alınmış ve açıklanmıştır. Bu senaryoya dayanarak, eski, değişmemiş ve ölçeklendirmesi zor olan monolitik yapıdaki sistemden örnek alınarak mikroservis yapısına geçişte nelere dikkat edilmesi gerektiği anlatılmıştır. Senaryoda kullanılan bulut hizmet sağlayıcısı AWS'dir.

Monolitik mimarisinde var olan servisler sadece routing ile ayrılmaktadır ve bütün servisler tek bir veritabanı ile çalışmaktadır. Monolitik mimarideki servislerimiz, tek bir kod tabanında çalışmakta ve sistemi her çalıştırmak istediğimizde bütün servis ve sistemi ayağa kaldırabilmemiz için bekleme süremiz çok uzun olmaktadır. Geliştirme sürecinde ise, sisteme yeni özellik eklenmek istendiğinde her uygulama çalıştırıldığında ve uygulama çalıştıktan sonra testlerin koşması için harcanan zaman ekibin zamanının önemli bir kısmını alır. Monolitik yapıdaki bu gibi sorunlar, proje için hem maliyet hem de zaman sorunu yarattığı için yapının mikroservise dönüştürülmesine büyük etki sağlar.



Şekil 4.2. Senaryosunun kurulduğu monolitik yapıdaki sistemler.

Geliştirme süresi boyunca, sistemde var olan özelliklerin dikkate alınması ve kod kısmında değiştirilecek her parçanın test edilmesi gerekmektedir. Değişikliklerin sadece o servise değil bütün sisteme test edilmesi gerekir aksı takdirde yeni eklenen özellik bütün sistemi olumsuz etkileme olasılığı vardır. Monolitik sistemlerdeki diğer bir sorun da ölçeklendirilebilir olması sorunudur. Ölçeklendirilebilirlik monolitik mimaride dikey düzlemde gerçekleşir ve bu da altyapının değiştirilmesi ve sunucuların güçlendirilmesi anlamına gelir. Örnek olarak kurduğumuz bu senaryoda ödeme işlemleri ve döviz işlemleri çoğunlukla kullanılıyorsa biz sadece bu iki sistemi değil sistemin tümünü ölçeklendirmemiz gerekir. Bu durum da işletme ve projeye çok büyük maliyet ve zaman kaybettirir. Monolitik mimaride otomatik ölçeklendirme olsa bile, bunun için çok büyük tecrübe ve zaman gerekmektedir. Bir diğer önemli durum da geliştirilen özelliğin

çalışmaması durumları sistemin tamamının çalışmamasına neden olur. Örneğin kullanıcı servisinde bir hata varsa bu olay bütün sistemlerin çökmesine neden olacaktır, sanal pos veya döviz gibi işlemlerin gerçekleştirilmesi mümkün olmayacaktır. Bu ve bunun gibi birçok sorundan dolayı monolitik mimarideki sistemlerin, modern ve ölçeklendirilebilir bir mikroservis mimarisine geçişimi inceleyeceğiz.

Monolitik yapıdan geçiş için ilk önce var olan servis ve veri tabanlarını inceleyip sıralarını önem, risk ve yaşanan sorun sayısına göre listeleyip planlama kısmına geçilmesi gerekir. Planlama yapılırken sistemin çökme veya kapanma gibi senaryoları önceden gözlemlenip bu olasılığın minimuma inebilmesi için çalışmalar yapılır.

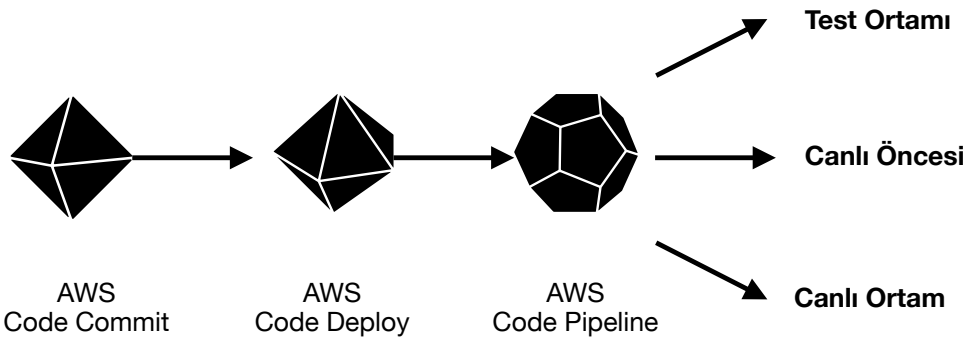
Bizim senaryomuzda ödemeler servisinde çok trafik yaşandığından sisteme etkisi çok büyüktür bu yüzden ödemeler servisi modüler geçişte ilk sırayı alır. Servisin kod tarafının geçişinden önce altyapısının hazırlanması gerekir, servislerin geçişinde kullanılan veri tabanı ortak olduğundan altyapı hazırlanırken bu altyapı içerisinde servise özel bir veri tabanı kurulması gerekir. Altyapı kurulduktan sonra ödeme servisinin yeni altyapıya geçebilmesi için ilk önce verilerin taşınması gerekilmektedir. Veriler taşındıktan sonra servisin kod tarafında ayrılması ve kendi başına çalışabilmesi gerekir. Servisin kendi kod tabanı oluşturulurken, gerekli olan bağımlı kütüphaneler ve servisler incelenerek taşınması sağlanır. Taşınma gerçekleştirildikten sonra servis aynı işlemler ve özellikleri sağlayıp sağlamadığı test edilip onaylanır.

Modern yazılım mimarilerinde altyapılar genellikle birden fazla oluşturulur. Bu altyapılar; test ortamı, geliştirme ortamı, canlıdan önce ve canlıdan sonraki ortam olarak adlandırılan altyapılardır. Birden fazla altyapının oluşturulması genellikle ortamların ve verilerin gerçek olan ortamdan ayrıştırılması ve geliştiriciler için test ortamının oluşturulmasını sağlar. Test ve geliştirici ortamı genellikle yazılımcıların ve testerlerin kullandığı eklenen yeni özelliği ilk test etmek için kullanılan ortamlardır. Canlında önce (Staging) ve Canlı ortam (Production) birbirleriyle aynı olmalıdır, bu da canlından önceki ortamda verilerin gerçek olmaması ama altyapının aynı olması ekibi canlı gibi gerçek bir altyapıda sistemi test etmelerine yardımcı olur.

Yeni geliştirilen ödemeler servisinin test ortamın gönderilerek test edildiği süre botunca monolitik sistem ile çalışır. Servislerin birer birer modüler bir sistem olarak taşınması sistemin kapanma riskini azaltır ve geliştirmeyi kolaylaştırır.

Servislerin kodlarının sunuculara taşınması ve bu kod paketlerinin hazırlanıp sunuculara gönderilmesi ve çalıştırılması için kullanılan araçlar olan CI ve CD araçlarının kurulması çok önemlidir. Bu kurulumlar üçüncü parti servisler yardımıyla gerçekleştirilir. Bu süreçte ilk önce kod verilerinin saklanacağı servis kurulumu gerçekleştirilir. Senaryomuza Git sunucusu kullanılmıştır. AWS servisi Code Commit yardımıyla kurulan depolarda (repo) kod bloklarımız saklanmaktadır. Bu repoları pipeline yardımıyla sunucuya taşınması sağlanmaktadır. Bu süreçlerin her kod değişikliğinde yapılması hem zaman hem maliyet açısından büyük olduğu için otomatikleştirmek gerekir. Sürecin otomatikleştirilmesi için genellikle var olan AWS servisleri kullanılmaktadır. Geliştiriciler kod değişikliklerini Git sunucularına taşıdığında otomatik olarak süreç arka planda başlatılır ve sistem güncellenir.

CI\CD süresi boyunca testler çalışabilir ve testlerin başarıyla tamamlanması durumda değişiklikler belirlenmiş olan sunuculara taşınır.



Şekil 4.3. CI/CD sürecinin AWS ortamında örneği.

Mikroservis yapıdaki bu süreçler çok daha basittir çünkü her bir servisin kendi pipeline yapısı olduğundan her küçük değişiklik test edilip kendi sunucu ortamına gönderilir. Mikroservis mimarilerinde altyapılar oluşturulurken bunların güncel kalması ve bakımı sistem büyüdükçe zorlaşmaktadır bu yüzden genellikle böyle durumlarda altyapı güncellemelerine otomatikleştirmeye gidilmekte, altyapı güncelleme ve geliştirme için üçüncü parti servisler kullanılır. Örneğin AWS CloudFormation, Terraform. Bu süreçler Infrastrukcture as a Code (IaaC) olarak bilinir. Bu araçların amacı

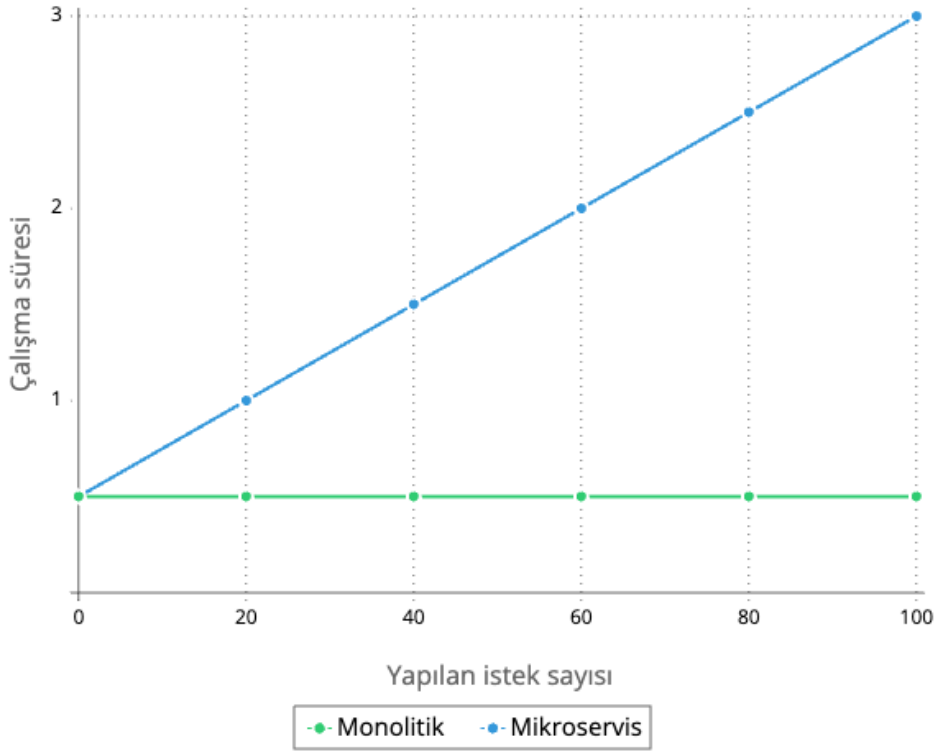
geliştiricilerin işini kolaylaştırmak ve kod üzerinde altyapı yönetimini sağlamaktadır. Bu araçlar ile yazılan kodlar yardımıyla altyapıların oluşturulması, güncellenmesi ve bakımı otomatikleştirilir aynı zamanda kolaylaşır.

4.3. Monolitik ve Mikroservis Yapılarda Operasyonel Karşılaştırma

Monolitik mimarilere karşıt olarak oluşturulmuş mikroservis mimarisi, iş sürecini birden çok bağımsız hizmete böler. Örnek olarak bir uçak bileti rezervasyonunda bilet rezervasyon işlemi bir dizi süreç içerir. Bir mikro servis mimarisinde, bireysel süreçler bağımsız hizmetlere bölünür. Bu örneğe bağlı olarak, hizmetler bilet rezervasyonu, kartla ödeme ve onay olabilir. Bu durumda bağımsız servisler tanımlanmış ara yüzler sayesinde birbirleriyle iletişim kurar. İki yazılım mimarisini aşağıdaki özellikleri ile karşılaştırıldığında sonuç olarak bir kazanan belli olacaktır.

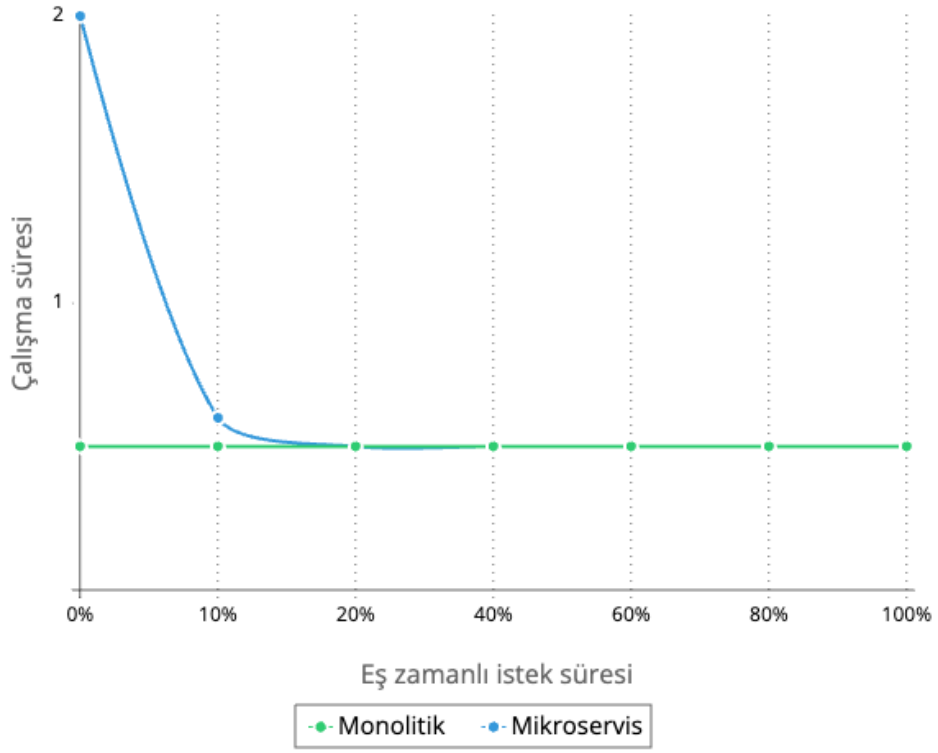
- Gecikme

Mikro hizmetler söz konusu olduğunda temel bir yasa olan ağ üzerinden herhangi bir hizmet arandığında, ağ üzerinden baytlar gönderilir. Bu işlem baytları elektrik sinyallerine, elektrik sinyallerini tekrar baytlara dönüştürmeyi içerir. Bu baytlara göre bir mikro hizmetin gecikme süresi yaklaşık 24 ms'dir. Toplam işlem süresi de Şekil 4.4'te gösterildiği gibidir;



Şekil 4.4. Monolitik ve mikroserviste yapılan istek sayısına karşı çalışma süresi.

Ideal olarak, tüm çağrılar aynı anda gerçekleşebilir ve birbirine bağlı değildir. Buna ise fan-out modeli denilmektedir. Şekil 4.5'te ise aynı anda daha fazla çağrı istenirken toplam sürenin nasıl azaldığı gösterilmektedir.



Şekil 4.5. Monolitik ve mikroserviste eş zamanlı istek süresine karşı çalışma süresi.

Tüm çağrıların paralel olarak yürütülmesi, hizmetin en uzun çağrı bittikten sonra tüketiciye geri döneceği anlamına gelir. Bu durumlarda monolitik mimarilerde gecikme yoktur. Sistem tamamen paralel hale getirildiğinde bile monolitik yapılar yine daha hızlı olacaktır. Bu durum grafiklerde de gördüğümüz gibi monolitik yapılar için açık bir üstünlüktür.

- Karmaşıklık

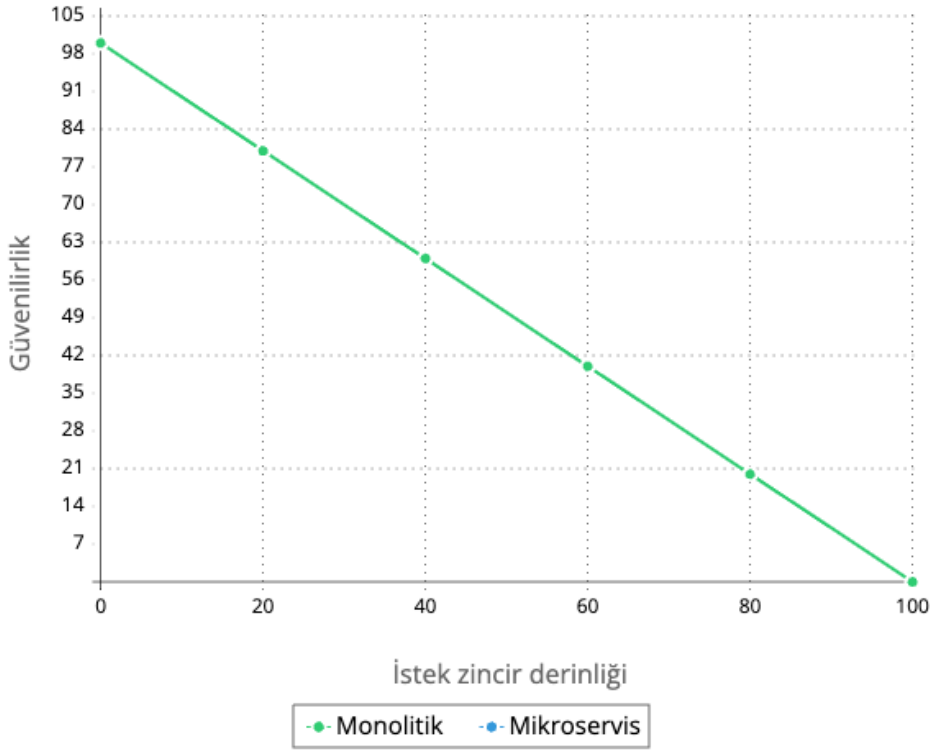
Sistemleri karşılattırırken karmaşıklık etkeninde gelişmenin karmaşıklığı, yazılımın karmaşıklığı gibi birçok farklı faktör mevcuttur. Geliştirmenin karmaşıklığı nedeniyle mikro hizmetlerde yazılım oluşturulurken kod tabanının boyutu hızla büyüyebilir. Bu sistemlerde farklı diller kullanılan birden fazla kaynak söz konusudur. Mikro hizmetlerin birbirinden bağımsız olması gerektiğinde genellikle kod tekrarı olacaktır.

Bir monolitte günlük kayıt tek bir günlük dosyasını incelemek kadar basittir. Bu durum mikroservislerde ise birden fazla günlük dosyanın kontrol edilmesini içerir. Mikro hizmetler her çağrı için farklı aralık kullanır. Bu durumda Elasticsearch gibi araçlarla hizmetlerin tüm günlük çıktılarının bulunması sağlanır. Jaeger gibi araçlar ise tüm mikro hizmetlerdeki çağrıları izleyebilir ve profil oluşturabilir.

Mikro hizmetler bir Kubernetes kümesinde çalıştırılırken karmaşıklık daha da artar. Kubernet'let otomatik ölçeklendirme gibi kolaylıklar sağlasa da yönetimi çok kolay olmayan karmaşık bir sistemdir. Monoliti başlatmak ya da durdurmak için genellikle basit bir komut yeterli olurken, mikro hizmetlerde o kadar basit değildir. Karmaşıklık için yapılan karşılaştırmada tekrar monolitik mimarinin üstün çıktığı görülmüştür.

- Güvenilirlik

Monolit yapılardaki istekler lokal olduğundan ağ arızası olasılığı çok düşüktür. Mikro hizmetler ile karşılaştırdığımızda ise mikroservis uygulamalarda ağ üzerindeki bir hizmeti %99,9 olasılıkla güvenli hale getirmek demek 1000 aramadan birinin başarısız olabileceği anlamına gelir. Şimdi o servis başka bir servisi çağırırsa %99,8 güvenlik elde ederiz. Büyüklüğü 10 çağrı olan bir sistem zincirinde %99 oranında güvenilirliğe sahip olunur. Bu da 100 çağrıdan sadece 1 çağrının başarısız olabileceği anlamına gelir.



Şekil 4.6. Monolitik ve mikroserviste istek zincir derinliğine karşı güvenilirlik.

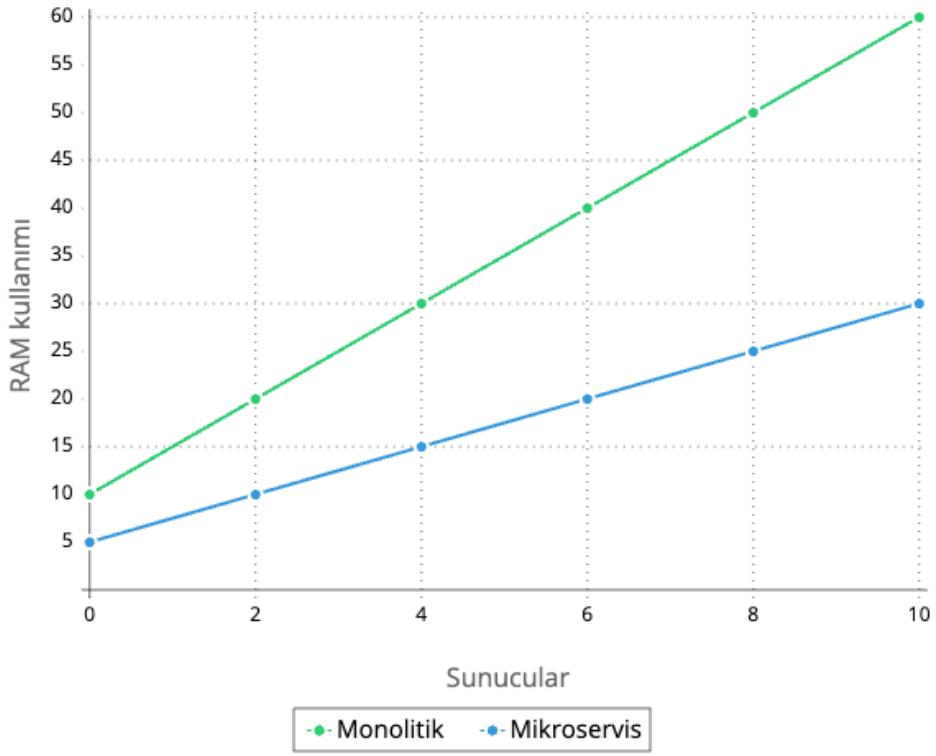
Bir mikroservis hizmet kümesinde tek bir hizmet başarısız olduğunda küme yöneticisi yalnızca yenisini getirir. Bu durum mimariyi oldukça esnek hale getirir. Mikroservis mimarileri tasarlanırken ağın bir noktada kesileceğini varsaymak önemlidir. Mikroservis sistemleri bu sorunu çözmek için bazı çözümler sunar. Ağ sorunları mikro hizmet mimarilerinde daha sık meydana gelir ancak bu beklendiğinden sorunları çözmek için tasarlanmıştır. Bunun dışında ağ arızaları düzenli olarak meydana geldiğinden mikro hizmetler ağ sorunlarının çözümlerini çok hızlı bir şekilde gerçekleştirir. Bu durumda ağ güvenilirliği söz konusu olduğunda mikroservis mimarisi kullanımı daha uygundur.

- Kaynak kullanımı

Bir mikro hizmet çağrısı aynı algoritmada aynı işi yaparsa her zaman monolitik servisten daha fazla kaynak kullanır. Docker ve sanal makineler monolitik hizmetine daha fazla yük getirir.

Başka bir kıyaslamada ise bir Docker konteynır kullanıldığında bağlantı sayısında yaklaşık %8 oranında bir düşüş olduğu gözlemlenir. Monolitik yapı kullanımında günlük toplama ve izleme gibi durumlarda da kaynaklar tüketilecektir.

Mikro hizmetler kullanılarak hazırlanmış sistemlerde ise kaynak kullanımı daha akıllıca tasarlanmıştır. Küme yöneticisi kaynakları gerektiği düzeyde getirebilme özelliğine sahip olduğu için gerçek kaynak kullanımı çok daha düşük bir oranda olacaktır. Kodun %20 'sinde için %80 'inin yapıldığı monolitik bir yapıya baktığımızda, kodun işlevselliğinin en yoğun olduğu noktada ölçeklendirebilirsek ne olacağını görebiliriz. Örneğin; monolitik yapının bir örneği 8 GB kullanıyorsa, iki örneği 16 GB'lık bir alan kullanır ve bu şekilde örmek sayısı arttıkça artmaya devam eder. Yapının %20 'sinin daha ağır yüklü işlemleri yapmak için kullanıldığını varsayarsak 8GB olan örneğimizin RAM kullanımının %20 'si olan 1.6 GB mikro hizmet ile kullanımı mümkündür. Şekil 4.7'de kaynak kullanımındaki farklılığın her işi sistemde nasıl gerçekleştiğini görebiliriz.

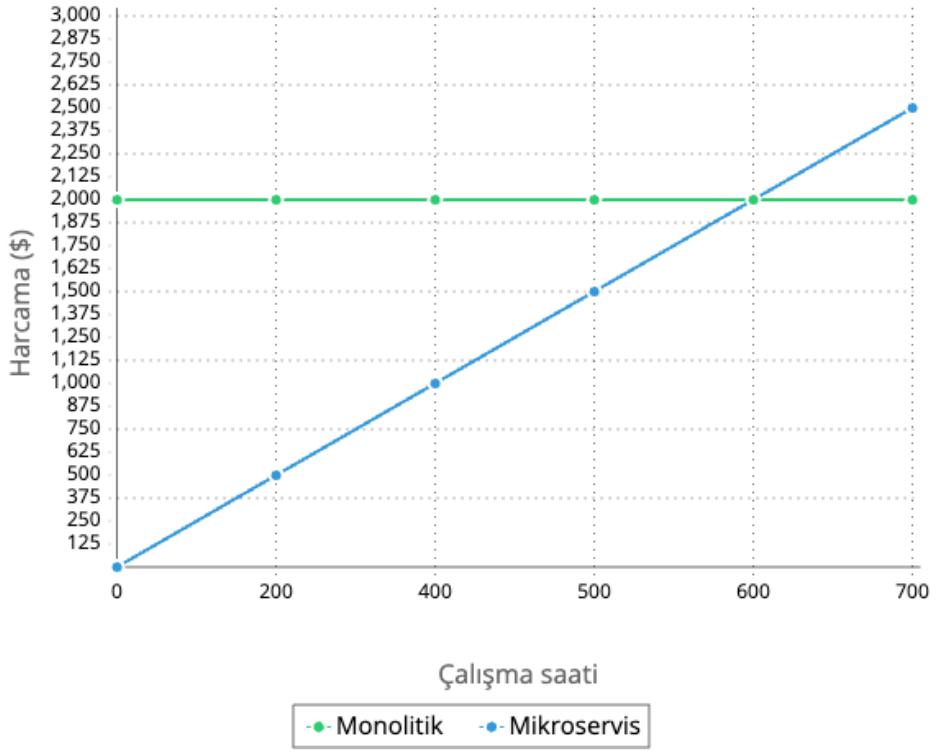


Şekil 4.7. Monolitik ve mikroserviste örnek sayısına karşı RAM kullanımı.

Monolitik yapılar Şekil 4.7'de de görüldüğü üzere çok uç durumlarda mikro hizmetlerden daha iyi performans gösterir. Ancak çoğu senaryo için normal kullanım koşullarında mikroservis hizmetlerinin kullanımı daha düşük olduğu için daha kazançlıdır.

- Ölçeklenebilirlik

Bir monoliti ölçeklendirmenin yollarından biri, birden fazla örnek çalıştırılabilir ve istekler buna göre yönlendirilebilir. Mikroservis mimarilerinde ise kaynak kullanımında görüldüğü gibi daha az kaynak kullanılabilir. Bir monolit tüm kaynakları kullanıyorsa daha fazla bağlantı kurmanın kolu ikinci bir örnek oluşturmaktır. Tek bir mikro hizmet de tüm kaynakları kullanıyorsa yalnızca bu hizmetin daha fazla örneğe ihtiyacı olacaktır. Ölçeklendirme daha kolay ve kesin olduğundan yalnızca gerekli miktarda kaynak kullanıldığı anlamına gelir. Örneğin Amazon Web Services bulut sunucusundan bir monolitin çalıştığını varsayalım. Bulut sunucusu yapısında 96 CPU ve 384 GB RAM sunar. 7/24 çalışan bir sistem için aylık çok yüksek değerlerde harcama elde edilir. Şekil 4.8'de, 7/24 yerine belirli bir süre çalışan sistemin çalıştırdıktan sonra bu 12 küçük bulut sunucusuna ne kadar mal olduğunu gösteren bir örneği vardır.

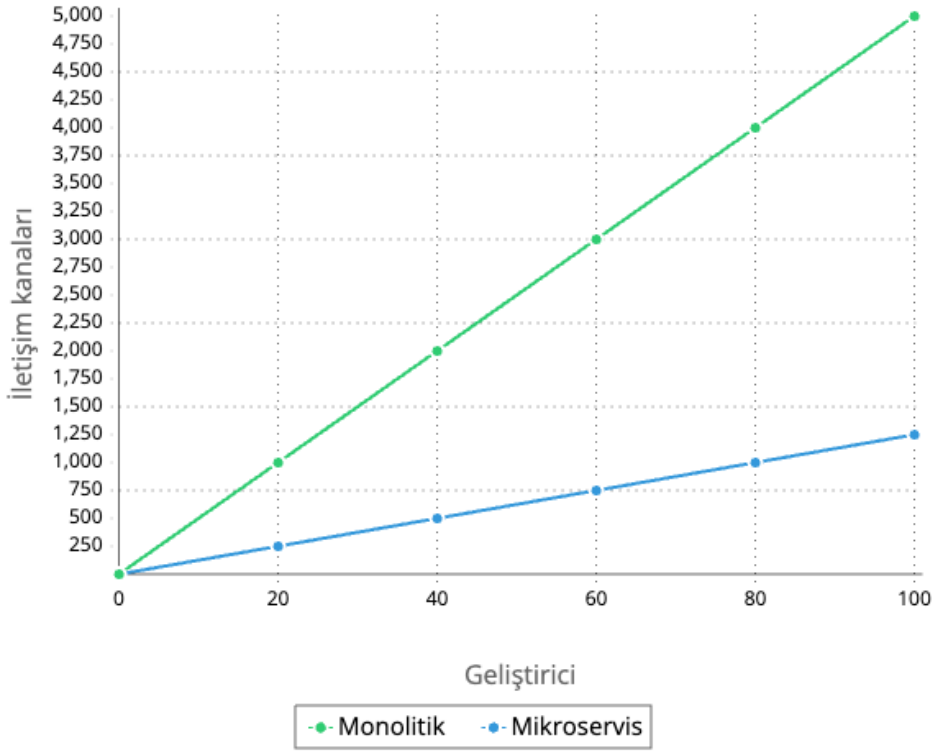


Şekil 4.8. Monolitik ve mikroserviste çalışma süresine karşı dolar bazında harcama.

Mikro hizmet mimarileri daha ayrıntılı olduğundan bu hizmetleri ölçeklendirmek de daha ayrıntılıdır. Hassas ölçeklendirme ve daha iyi kaynak kullanımı için mikroservis hizmetleri daha avantajlı kulanım sunar.

- İletişim

Mikroservis hizmetleri genellikle geliştirme ekibinin boyutuna göre tanımlanır. 20 kişiden oluşan bir ekip, 190 olası iletişim kanalına sahiptir. Bu geliştiricileri iki ekibe ayırmak iletişim kanallarının sayısını önemli ölçüde azaltır. 20 geliştiriciden oluşan örnek ekibi her biri 5 kişiden oluşan 4 mikro hizmet ekibine ayırırsak ekip başına 10 iletişim kanalı oluşturulur. Toplam iletişim kanalı 46 olup, 20 kişilik ekibin dörtte biri kadardır. Şekil 4.9'da mikro hizmet ekiplerine karşı iletişim kanalları sayısı gösterilmektedir.



Şekil 4.9. Monolitik ve mikroserviste geliştirici ekip sayısına karşı iletişim kanalları sayısı.

50 geliştiriciden oluşan bir ekipte iletişim kanallarının sayısı neredeyse 10 katıdır. Bu büyüklükteki bir ekipte çalışmak iletişim için çok zaman harcamak anlamına gelmektedir. 10 geliştiricinin üzerindeki herhangi bir geliştirme projesi, onu daha küçük ekiplere bölerek iyi bir şekilde geliştirmenin yapılmasını sağlar. Hizmet sınırları boyunca iyi tanımlanmış ara yüzlere sahip oldukları için mikroservis hizmetleri iletişim için ideal kullanımı sağlar.

BÖLÜM 5

SONUÇLAR

Bu tezin ana hedefi mikroservis mimarisinin monolitik mimariye kıyasla artılarının ve eksilerinin açıklanması ve sunulmasıdır. Bir uygulama ve servisin monolitik sistemden mikroservis sisteme geçiş süreci ve hızlandırılmasının açıklanması ile bu geçişin kolaylaştırılmasıdır. Monolitik mimariler, birçok büyük kuruluş gibi banka sistemlerinde de test edilmiş, onaylanmış ve çok uzun yıllar boyunca kullanılmıştır. Bölüm 4'te de açıklandığı gibi, monolitik mimari sistemlerinin güncellenmesi daha zor ve riskli bir durumdur. Bununla birlikte uygulama büyüdükçe ve ekipteki çalışanların sayısı arttıkça monolitik mimari sistemlerinde daha birçok sorunla karşılaşılır. Sistemin küçük bir altkümesinde yapılan bir değişiklik sistemin tamamını etkileyeceğinden tüm monolitik sistemin yeniden oluşturulması, test edilmesi ve yeniden koşullandırılması gerekmektedir. Monolitik sistemlerde karşılaşılan diğer bir sorun da zamanla şirketin bünyesine karılan birçok yeni geliştiriciyle kod tabanını modüler halde tutmak zorlaşır ve hataya yatkın hale gelir.

Bu tezde anlatılan ve incelenen geçişler monolitik mimaride bulunan bir sistemden mikroservis yapıdaki bir sisteme geçişin nasıl daha kolay ve az maliyetli oluşunu incelemek, mevcut bilgileri bir ana başlık altında toplamaktır. Monolitik mimariden, mikroservis yapısına modüler geçişler hem sistemin anlık çalışmasına yardım eder hem de yazılım ekiplerinin daha kolay bir şekilde yönetilmesine ve geliştirilmesine imkânı sağlar. Büyük yazılım kuruluşlarında ve mimarilerde bu geçişlerin düzgün planlanması, ortaya çıkabilecek hata ve sonuçların öngörülmesi ve planlanması gerekmektedir. Yapılan raporlandırmalarda genellikle monolitik mimariden mikroservis

mimarisine geçişin ne kadar sisteme kolaylık, ölçeklendirilebilirlik ve daha az maliyet sağladığı hesaplanır.

Mikroservis mimarisi monolitik mimari modelinin sınırlamalarını ele almayı ve düzeltmeyi amaçlar. Mikroservis mimarisindeki uygulama işlevsel hizmetlere bölünmüş haldedir ve her hizmetin kendi arasında yüksek uyumu vardır. Adından da anlaşılacağı gibi mikro hizmetler küçük ve birbirinden bağımsız hizmetlerdir (örn. Ödeme hizmeti, kullanıcı hizmeti gibi). Mikroservis mimarisine geçiş sürecinde tüm aşamaları açıkça bilinse de bazı durumlarda şirket bünyesindeki çalışan ekiplerin buna hazırlanması için dışardan mikroservis alanında tecrübesi olan farklı kurum ve kişilerden yardım alınması gerekebilir. Mikroservis mimarisi tecrübe gerektiren bir yapıdır ve geçiş sürecinin yönetilmesi karmaşık ve zor durumlara yol açabilir. Bu yüzden genellikle birçok kurumsal şirket uzmanlardan yardım alarak bu geçiş sürecini hızlandırır.

Mikroservis yapısı tanımı gereği de küçük yapılardan oluştuğu için sistemin yapılarını yeniden yazmak çok daha kolaydır. API değişmediği sürece, sorumlu geliştirme ekibi uygulama ayrıntılarını kolaylıkla değiştirme özgürlüğüne sahiptir. Bu sayede ekipler yeni çıkan teknolojileri kolaylıkla bünyelerine kayabilir ve bu teknolojilerden faydalanabilirler. Mikro hizmetlerin hata izolasyonundan bahsedilecek olunursa, mikro hizmetler ile tüm sınırlar izole edilir ve bu da olası problemlerin de izole edildiği anlamına gelir. Bu durum arıza olasılığını oldukça fazla azaltır ve sistemin kolay test edilmesini aynı zamanda kolay bakımının yapılmasını sağlar. Mikroservis yapısındaki bir hizmetin kolay dağıtılabilir olması tüm ekiplerin bağımsız olarak hizmet geliştirebileceği ve dağıtabileceği anlamına gelir. Bu durum da ekiplere büyük kolaylık sağlar ve yeni sürümlerin sık sık dağıtılmasını kolaylaştırır. Her hizmet tek bir sorumluluğa sahip olduğu için hizmetler aralarında yüksek uyum içerisindedirler. Oluşturulan mikro hizmetler yapılarından dolayı geliştirme ortamını yavaşlatamaz ve geliştiricilerin daha üretken olmalarına olanak sağlar. Aynı zamanda her mikro hizmet teorik olarak farklı teknolojik yapıya sahip olabileceği için birbirinden farkı teknoloji yığınları kullanılabilir.

Mikroservis yapısına geçilmek istendiğinde, ayrıca bulut tabanlı kaynakların da temin edilmesi gerekmektedir. Bu bulut tabanlarının kurulumu genellikle ayrı bir ortamda gerçekleştirilir. Oluşturulan yeni yapının çalıştırılması, geliştirilmesi ve test edilmesi oldukça uzun ve zaman alan bir süreçtir. Bu süreç bittikten sonra ise oluşturulan yeni

mikroservis tabanlı yapı, eski yapının yerine geçebilir ve geliştirme ortamında çalışmaya başlayabilir. Yeni yapıya geçmeden önce dikkat edilmesi gerekilen en önemli unsur, yapının büyük işlem ve büyük trafikle test edilmesidir. Mikroservis yapılarının sahip oldukları birçok avantajdan sonra oluşabilecek birkaç dezavantajı da anlamak önemlidir. Dağıtılmış bir sistem doğası gereği daha karmaşık ve kolay anlaşılabilir olmayan bir yapıya sahiptir. Dağıtılmış mikroservis tabanlı bir uygulamada, uygulamanın farklı sürümlerinin birbiri ile uyumunu test etmek ve bu uyumdan emin olmak daha karmaşıktır. Bu sistemlerde entegrasyon ve uçtan uca testlerini çalıştırmak daha fazla dikkat gerektirir.

BÖLÜM 6

TARTIŞMA

Mikroservis hizmetler yapıyı uygulayan her şirketin kod tabanındaki değişikliği detaylandıran ve anlaşılması kolay hale getiren sistemler olduğu için son zamanlarda kullanımı yaygınlaşmış yapılardır. İşlevsel ayrıştırma ve ölçeklendirme yetenekleri otonom geliştirme ile birleştiğinde mikroservis yapısını geleneksel monolitik yapıdan daha değerli kılar. Bu tez, mühendislik kavramlarını ele alarak monolitik tabanlı bir yapıdan mikroservis tabanlı bir yapıya geçiş sürecinde geliştiricilere yardımcı olan tüm arka plan ve ilkeleri birleştirerek, geliştiriciler için yararlı kapsamlı bilgi sunmaktadır. Bu bağlamda yapılan açıklamalarda mimariler arasındaki geçiş önemi ve gereksinimi açıklanırken geliştiricilerin hangi mimariye uygun olduklarını seçmeleri amaçlanmıştır. Tez süresince önerile mikroservis mimarisine geçi birçok avantaj sağlar. Aynı sisteme entegre olan farklı servislerin geliştirilen otonom ekipler varlığında karmaşık uygulamaların geliştirilmesinde kalite ve hız artışı sağlanır. Yeni teknolojiler kullanılırken uygulamalar arasındaki heterojenlik her hizmetin kendi sınırlarının olması ve farklı hizmetlerin farklı programlama dilleriyle yazılabilir olması hem kuruma hem de geliştiricilere büyük avantajlar sağlamaktadır. Ayrıca, bir hizmeti farklı bir dil ile gerektiğinde yeniden geliştirme özelliği sayesinde yen teknolojinin uygulanması çok daha basittir.

Mikroservis mimarisi, monolitik mimariye önemli bir alternatiftir. Bu tür mimari, gerekli iş mantığı uygulandığında çok sayıda karmaşık veriyi işleyen bir yapıda bulunmaktadır. Yapılan araştırmalarda da görüldüğü gibi bir mimari yapıya başlamadan önce sistemin karmaşıklığının ve zorluğunun keşfedilmesi gerekmektedir. Monolitik

yapıdaki bir sistem, mikroservis yapısın tarafından sağlanan nitelikleri gerektirdiğinde, bu tür mimariye geçişin yapılması gerekmektedir. Geçiş sırasında ise kademeli ve kontrollü bir geçişin sağlanması ve modülerlik ve işlevselliği yöneterek en kısa zamanda geçişin sağlanması gerekir.

Mikroservis yapılar yeniden kullanılabilir, incelenebilir ve yönetilebilir olmasından dolayı geniş alanlarda kullanım kolaylığı sağladığı söylenebilir. Aynı zamanda bu yöntemle yapılan projeler, yazılımın test sürecinde ve projenin müşteriye daha kaliteli ve uygun sonuç vermesini sağladığı görülmüştür. Son zamanlarda yapılan birçok çalışmada eski monolitik yapılar yerine esnek olarak tanımlayabileceğimiz mikroservis yapılar kullanılmaya başladığı ve çok iyi sonuçlar verdiğini görebiliyoruz. Geliştiriciler açısından monolitik yapıdan mikroservis yapıya geçiş zorlu ve karmaşık bir süreçtir. Yapılan araştırma çalışmasıyla inanıyoruz ki araştırma ve geliştirme yapan kişilere için bilgilere kolayca ulaşabilme şansı doğar.

KAYNAKLAR

Aderaldo, C. M., Mendonc, N. C., Pahl, C., & Jamshidi, P. (2017). Benchmark Requirements for Microservices Architecture Research. In *1st Interna- tional Workshop on Establishing the Community-Wide Infrastructure for Architecture- Based Software Engineering (ECASE)*, 8–13.

Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution. In *Proceedings of the conference on The future of Software engineering - ICSE '00*, 34, 120-134.

Blanch R., (2017). Microservices: Strategies for Migration in a Brownfield Environment.

Candela, I., Bavota, G., Russo, B., & Oliveto, R. (2016). Using Cohesion and Coupling for Software Remodularization. *ACM Transactions on Software Engineering and Methodology*, 32, 117-124.

Chen, R. & Liand, S. (2017). *From Monolith to Microservices: A Dataflow- Driven Approach, 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing*, 466–475.

Dragoni N., Dustdar, S., Larsen S. T. & Mazzara, M. (2017). *Microservices: Migration of a Mission Critical System, Technical University of Denmarkand Orebro University, Sweden.*

Dehghani Z., (2018). How to break a Monolith into Microservices. [Online]. Available: https://martinfowler.com/articles/break-monolith-into- microservices.html

Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L., (2016). *Microservices: yesterday, today, and tomorrow.* arXiv preprint arXiv:1606.04036.

Ford, N. & Richards, M. (2020), *Fundamentals of Software Architecture*, O'Reilly Media, Inc.

Fowler, M., (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Gökalp, G., (2016), Monolithic ve MicroService Architecture'a Genel Bir Bakış, http://www.gokhan-gokalp.com/monolithic-ve-microservice-architecturea- genel-bir bakis.

Hasselbring, W. (2016). Microservices for Scalability: Keynote Talk Abstract. Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, 33-134.

Heinrich, R., Hoorn, A., Knoche, H., Li, F., Lwakatere, L., Pahl, C. & Wettinger, J. (2017). Performance Engineering for Microservices: Research Challenges and Directions.

Jarman, P. (2017). *Microservice – A New Application Paradigm, Infosys.*

Knoche, H. & Hasselbring, W. (2018). Using Microservices for Legacy Software Modernization, IEEE Software, vol. 35, no.3, 44–49.

Knoche, H. & Hasselbring, W. (2019). Drivers and Barriers for Mikroservis Adoption - A Survey among Professionals in Germany. *Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling*, *14*(1), 1–35.

Lewis, J & Fowler, M. (2014). Microservice. http://martinfowler.com/articles/mikroservices.html.

Levine, S. S. & Prietula, M. J. (2014). Open Collaboration for In-novation: Principles and Performance. Organization Science, 25(5), 1414-1433.

Linthicum, D., (2017). From containers to microservices: Modernizing legacy applications. [Online]. Available: https://techbeacon.com/enterprise- it/containers-microservices-modernizing-legacy-applications

Levcovitz, A,. Terra, R. & Valente, M. T. (2015). *Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM),* 97–104.

Montesi, F. & Weber, J. (2016). *Circuit Breakers, Discovery and API Gateways in Microservices*, 9-21.

Mishra, M., Kunde, S. & Nambiar, M. (2018). *Cracking the Monolith: Challenges in Data Transitioning to Cloud Native Architectures. ECSA '18 Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, 1*–4.

Megargel, A., & Shankararaman, V. (2020). *Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example.* Software Engineering in the Era of Cloud Computing. 85-108. Research Collection School Of Information Systems.

Namiot N., Dmitry S., & Manfred S., (2014). *On micro-services architecture. International Journal of Open Information Technologies,* 2-9.

Newman, R. (2015). Building microservices: designing fine-grained systems, First Edition. Beijing Sebastopol, CA: O'Reilly Media.

Pachghare, V. (2016). *Microservices Architecture for Cloud Computing, Journal of Information Technology and Sciences.*

Rahman, M., & Gao, J. (2015). *A reusable automated acceptance testing architecture for microservices in behavior-driven development, EEE Symposium on Service-Oriented System Engineering*, 21-325.

Richardson, C. & Smith, F., (2016). *Microservices: From design to deployment*, *Nginx Inc,* 24-31.

Richordson, R., (2016). Microservices From Design to Deployment. https://www.nginx.com/blog/ microservices-from-design-to-deployment-ebook-nginx/.

Sun, Y., Nanda, S., & Jaeger, T., (2015). Security-as-a-service for microservices-based cloud applications, *IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, 50-57.

Salonen, A., & Deleryd, M. (2011). Cost of poor maintenance. *Journal of Quality in Maintenance Engineering*, 24(2), 233-246.

Sun, Y., Nanda, S. & Jaeger, T. (2015). Security-as-a-Service for Microservices-Based Cloud Applications. IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom).

Söylemez, M., & Tarhan, A. (2018). Mikroservis Mimarisi ve Mimari Faktörleri Üzerine Endüstriyel Bir İncelem. http://ceur-ws.org/Vol-2201/UYMS_YMO_2018_paper_106.pdf

ÖZGEÇMİŞ

1994'de Kosova'da Prizren ilinde doğdu. İlk ve orta öğrenimini "Fadil Hisari" ilkokulunda, lise öğrenimini "Gjon Buzuku" fen lisesinde tamamladı. 2013 yılında Trakya Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümünde okumaya hak kazandı. 2017 yılında Lisans eğitimini tamamladı. Eylül 2018'da Trakya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans eğitimine başladı.