Mikroservis Mimarisi ve Mimari Faktörleri Üzerine Endüstriyel Bir İnceleme

Mehmet Söylemez[1] ve Ayça Tarhan[2]

[1] TÜBİTAK-BİLGEM-YTE
Yazılım Teknolojileri Araştırma Enstitüsü, Ankara, Türkiye
[2] Hacettepe Üniversitesi, Bilgisayar Mühendisliği Bölümü, Ankara, Türkiye

1 mehmet.soylemez@tubitak.gov.tr
2 atarhan@hacettepe.edu.tr

Özet. Mikroservis mimarisi, servis odaklı yazılım endüstrisinde baskın bir mimari tarz haline gelmiştir. Mikroservis mimarisi, sistemi küçük hizmetlere ayırmayı vurgulayan bir mimari tarzdır ve geleneksel hizmet odaklı mimari tarzın bir evrimidir. Bu tarzın yaygın olarak kabul edilen yararları arasında; çeviklikteki artış, geliştirici verimliliği, esneklik, ölçeklenebilirlik, güvenilirlik, süreklilik, ilgilerin ayrılması (seperation of concerns) ve dağıtım kolaylığı sayılabilir. Bu faydaların yanında Mikroservis mimarisi; ağ üzerindeki hizmetlerin keşfedilmesi, güvenlik yönetimi, iletişim eniyileme, veri paylaşımı ve performans değerlendirme gibi bazı gerekleri da beraberinde getirmektedir. Bu gerekler düzgün bir şekilde ele alındığında, mikroservis mimarisi, yazılım sisteminin yukarıda belirtilen faydalardan yararlanmasını sağlar. Literatürde mikroservis mimarisi için yapılan çalışmalar dağınık olarak bulunmaktadır ve hangi çözümlerin hangi sorunlar için önerildiğini belirten bir çalışmaya rastlanmamıştır. Bu çalışmada, mikroservis dünyasında başı çeken teknoloji şirketlerinin mikroservis mimarisinin güçlü ve zorlu yönleri baz alarak geliştirdikleri çözüm önerileri, belirlemiş olduğumuz araştırma soruları dâhilinde değerlendirilerek incelenmiştir.

Anahtar Kelimeler: Mikroservis mimarisi, yazılım mimarisi, teknoloji araştırması

CEUR-WS.org/Vol-2201/UYMS_YMO_2018_paper_106.pdf

An Industrial Investigation Into Microservice Architecture and Its Factors

Mehmet Söylemez[1] ve Ayça Tarhan[2]

[1] TÜBİTAK-BİLGEM-YTE
Software Technologies Research Institute, Ankara, Turkey
[2] Hacettepe University, Computer Engineering Department, Ankara, Turkey

1 mehmet.soylemez@tubitak.gov.tr
2 atarhan@hacettepe.edu.tr

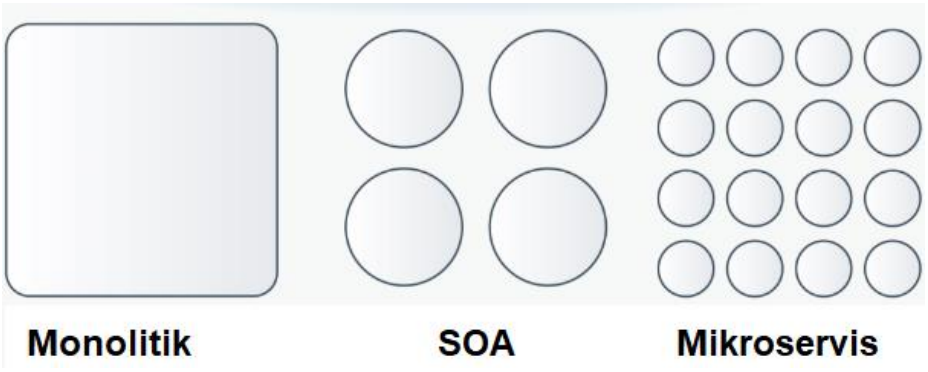
Abstract. Microservice architecture has become a dominant architectural style in the service oriented software industry. Microservice architecture is an architectural style that emphasizes allocating the system to small services and is an evolution of traditional service-oriented architectural style. Commonly accepted benefits of microservice architectural style include; agility increase, developer productivity, flexibility, scalability, reliability, continuity, seperation of concerns, and ease of deployment. In addition to these benefits, microservice architecture also brings some requirements such as network discovery, security management, communication optimization, data sharing and performance evaluation. However, when properly implemented, the microservice architecture approach allows the system to take advantage of the above mentioned benefits. Studies in microservice architecture are scattered in the literature and we could not come across with a study indicating which solutions are proposed for which problems. In this study, the suggestion of the solution developed by the technology companies, which are leading in microservice architecture world, based on strong and difficult aspects of microservice architecture has been examined within the research questions we have determined.

Keywords: Microservice architecture, software architecture, technology research

1 Giriş

Mikroservisler, Servis Odaklı Mimari (*Service Oriented Architecture – SOA*) yaklaşımından ortaya çıkan, kendi kendini yönetme ve hafifliğe vurgu yapan bir mimari tarzdır. Şekil 1'de gösterildiği gibi mikroservis mimari tarzında, SOA'dan farklı olarak servisler daha küçük, sınırları alana özgü ve ihtiyaca göre iyi belirlenmiş olarak tasarlanmaktadır. Mikroservis mimari tarzının SOA'dan diğer bir farkı ise merkezi bir mesajlaşma sisteminin yer almamasıdır. Bu mesajlaşma sistemi yerine, daha hafif ve basit mesajlaşma teknikleri kullanılır.

Mikroservisler tasarlanırken fonksiyonel ayrıştırmaya dikkat etmek gereklidir. Her mikroservis kendi sorumluluk alanındaki işler ile ilgilenmeli ve otonom bir yapı sunmalıdır. Mikroservis mimarisi ile geliştirilen sistemlerin, monolitik bir mimari ile geliştirilen bir sisteme göre daha çevik, esnek ve ölçeklenebilir olduğu gözlemlenmiştir [1,2]. Bu mimari bir ilke olmakla birlikte, Servis Platformu (*Platform as a Service* – PaaS) bulutları için bir uygulama paketleme mekanizması olarak görev yapabilecek hafif sanallaştırma mekanizması sağlamak için, bulut bilişimdeki konteyner teknolojisine paralel olarak ortaya çıkmıştır. Mikroservisler, ideal olarak bulut yoluyla paketlenebilir, tedarik edilebilir ve düzenlenebilir [3].



Şekil 1. Monolitik, SOA ve Mikroservis Mimarileri

Son on yılda, önde gelen yazılım danışmanlığı firmaları ve ürün tasarım şirketleri, ekiplerin ve yazılım kurumlarının genel olarak daha üretken olmasına ve daha başarılı yazılım ürünleri oluşturmasına olanak veren cazip bir mimari olması açısından, mikroservis yaklaşımını faydalı bulmuştur. Geleneksel yazılım geliştirme işinin dışındaki birçok kurum da bu mimari tarzı deneyip test etmiş ve oldukça faydalı bulmuştur. Netflix ve SoundCloud gibi şirketler buluttaki mikroservis tarzını benimseyip bundan çeşitli faydalar sağlamıştır [4][5].

Mikroservis mimarisinin yaygınlaşması ile birçok teknoloji şirketi, dağıtık ortam ve mikroservis mimarisi özelinde çözümler geliştirmeye başlamıştır. Birbirinin yerine tercih edilebilecek çözümler de bir hayli fazladır, fakat bu çözümlerin neler olduğunu ve ne gibi faktörleri adreslediğini bir arada anlatan kaynak sayısı azdır. Bu boşluktan hareketle bu çalışmada, yaygın bilinen teknoloji şirketlerinin ilgilendiği faktörleri ve

önerdikleri çözümleri inceleyerek mikroservis mimarisi çözümlerine endistriyel bir bakış sunmak hedeflenmiştir. Çalışmada, aşağıdaki iki araştırma sorusu (AS) ile teknoloji şirketlerinin mikroservis mimarisine katkıları tartışılacaktır.

- AS-1. Teknoloji şirketleri, mikroservis mimarisi çerçevesinde hangi faktörler ile ilgileniyorlar?
- AS-2. Teknoloji şirketleri bu faktörler için ne gibi çözümler öneriyorlar?

Bu sorulara cevap verebilmek için öncelikle, literatür taraması ile mikroservis mimarisi için önemli görülen faktörler belirlenmiştir. Daha sonra, teknoloji şirketlerinin bu faktörlerden hangileri ile ilgilendikleri araştırılmış ve son aşamada ise ilgilendikleri alanlar için ne gibi çözümler ürettiklerini tespit edilmiştir.

Bildirinin izleyen kısımları şu şekilde devam edecektir: İkinci bölümde mikroservis mimarisi hakkında bilgi verilecektir. Üçüncü bölümde ilişkili çalışmalar özetlenecektir. Dördüncü bölümde mikroservis mimarisine ilişkin belirlenen faktörler anlatılacaktır. Beşinci bölümde araştırma sorularına yanıtlar paylaşılacaktır. Altıncı ve son bölümde ise çalışmanın sonuçları verilecektir.

2 Ön Bilgi

2.1 Alan-Odaklı Tasarım (Domain-Driven Design – DDD)

Alan-Odaklı Tasarım (*Domain Driven Design* – DDD), ilk olarak Eric Evans tarafından önerilmiştir. Temel amacı uygulamanın temel iş kavramlarını bir modele derinlemesine bağlayarak karmaşık ihtiyaçlara yönelik yazılım geliştirme çözümü sunmaktır. Bu teori şu üç maddeden oluşmaktadır [6].

- Projenin birincil odağını ana alana ve etki alanı mantığına yerleştirin.
- Karmaşık tasarımları bir modele oturtun.
- Teknik ve alan uzmanları arasında, problemin kavramsal tabanını daha da incelemek üzere bir işbirliği başlatın.

DDD, yeni beceriler, disiplin ve sistematik bir yaklaşım gerektirir. DDD, bir teknoloji veya metodoloji değildir. DDD, karmaşık alanlarla uğraşan yazılım projelerine odaklanan ve yazılım projelerini hızlandıran tasarım kararları vermek için bir uygulama ve terminoloji yapısı sağlar [6].

DDD, mikroservis mimarisi için kritik öneme sahiptir. Büyük ve karmaşık bir alan için yazılım geliştirilirken bu alanı, farklı alt alanlara ayırmak ve buradan mikroservislerin sınırlarını doğru bir şekilde çizmek, DDD'nin sağladığı bakış açıları ile mümkün olabilmektedir.

2.2 Mikroservis Mimarisi

Mikroservis mimari tarzı, her biri kendi sürecinde çalışan ve genellikle bir HTTP kaynak Uygulama Programlama Arayüzü (*Application Programming Interface – API)* olan hafif mekanizmalarla iletişim kuran küçük bir servis paketi olarak tek bir uygulamanın geliştirilmesine yönelik bir yaklaşımdır [1]. Bu hizmetler, iş mantığı etrafında yapılandırılmıştır ve tam otomatik dağıtım makineleri tarafından bağımsız

olarak konuşlandırılabilir. Farklı programlama dillerinde yazılabilen ve farklı veri depolama teknolojileri kullanabilen bu hizmetlerin olabildiğince az merkezi yönetim ihtiyacı vardır [1].

Mikroservis mimari tarzını açıklamaya başlamak için, onu monolitik mimari tarzla karşılaştırmak yararlı olacaktır. Aralarındaki temel fark, Şekil 1'de de gösterildiği gibi, geliştirilen uygulamaların boyutlarıdır. Mikroservis mimari tarz, monolitik mimari tarz ile bir bütün olarak geliştirilecek bir uygulamanın, daha küçük uygulamalar olarak geliştirilmesi olarak düşünülebilir. Kurumsal uygulamalar genellikle üç ana bölümden oluşur: Bir istemci tarafı kullanıcı arabirimi (HTML sayfaları ve kullanıcının makinesinde bir tarayıcıda çalışan javascript içerir), bir veritabanı (ortak ve genellikle ilişkisel, veritabanı yönetimine eklenen birçok tablodan oluşur) ve sunucu tarafında bir uygulama. Sunucu tarafındaki uygulamada, HTTP istekleri işlenir, iş mantığı yürütülür, veritabanından verileri alıp güncelleyecek ve tarayıcıya gönderecek HTML görünümleri hazırlanır. Bu yapı monolitler için iyi bir örnektir. Sistemdeki herhangi bir değişiklik, sunucu tarafı uygulamasının yeni bir sürümünü oluşturmayı ve dağıtmayı içerir. Değişim döngüleri birbirine bağlıdır. Uygulamanın küçük bir kısmında yapılan bir değişiklik, tüm monolitin yeniden oluşturulmasını ve konuşlandırılmasını gerektirir [1].

Mikroservis mimarisinin ise monolitik mimariden farklı olarak ortak bazı özellikleri vardır. Bunlar şu şekildedir [7]:

- Servisler ile bileşenleştirme
- İş yetenekleri etrafında organize edilme
- Akıllı arayüzler ve basit iletişim
- Merkezi olmayan yönetişim
- Merkezi olmayan veri yönetimi
- Altyapı otomasyonu
- Başarısızlık için tasarım

3 İlişkili Çalışmalar

Literatürde sayıca az olmakla birlikte, mikroservisleri farklı araştırma soruları ile değerlendiren çalışmalar mevcuttur. Bu bölümde bu çalışmalara kısaca değinilecektir.

Vural ve arkadaşlarının yaptığı çalışmada [14], mikroservisler ile ilgili çalışmalar incelenmiş ve bu çalışmalar neticesinde mikroservis alanında yapılan çalışmaların, araştırma türlerinin ve motivasyonlarının neler olduğu belirlenmiştir. Ayrıca çalışmalarda sıklıkla kullanılan araçlar da belirlenmiştir.

Pahl ve arkadaşlarının yaptığı çalışmada [15] ise mikroservis mimarisi tasarım tipleri, geliştirmelerde hangi metot ve tekniklerin kullanıldığı ve şu anda hangi alanlarda araştırma boşluklarının bulunduğu belirtilmiştir.

Richardson'ın yaptığı çalışmada [7], mikroservis mimarisinin hangi alt çalışma alanlarından oluştuğu ve bunlar arasındaki ilişkiler belirtilmiştir.

Bu çalışmada ise diğer çalışmalardan farklı olarak mikroservis mimarisinin hangi faktörlerden oluştuğu, akademik ve akademik olmayan literatür sentezlenerek belirlenmiş ve faktörler için ne gibi çözümlerin olduğu araştırılmıştır.

4 Mikroservis Mimarisi Faktörleri

Tüm mimari tarzların faydaları ve zorlukları (maliyetli oldukları noktalar) vardır. Mimari şekillendirilirken tüm bu noktaların göz önünde bulundurulması mimarinin geleceği için kritik öneme sahiptir. Mikroservis mimarisinin getirdiği faydalar gibi zorluklar da vardır. Bu maddeler ışığında mikroservis mimarilerinin ana faktörleri şekillenmiştir.

4.1 Mikroservis Mimarisi Faydaları

- Büyük, karmaşık uygulamaların sürekli teslimatı ve dağıtımını sağlar [1, 2, 5, 7, 8, 9, 10, 11].
- Daha kaliteli test edilebilirlik sağlar. Test işlemleri daha küçük ve hızlıdır [1, 2, 5, 7, 11].
- Daha iyi konuşlanma – servisler bağımsız olarak dağıtılabilir [1, 2, 5, 7, 8, 9, 10, 11].
- Birden fazla takım etrafında geliştirme aşamalarını organize etmenizi sağlar. Her ekip bir veya daha fazla hizmetten sorumludur. Her ekip, diğer tüm takımlardan bağımsız olarak kendi hizmetlerini geliştirebilir, dağıtabilir ve ölçeklendirebilir [1, 2, 5, 7, 8, 9, 10].
- Her bir mikroservis nispeten küçüktür [1, 2, 5, 7, 8, 9, 10, 11].
- Geliştirici tarafından daha kolay anlaşılır ve daha hızlı adaptasyon sağlar [7].
- Uygulama daha hızlı başlar, bu da geliştiricilerin daha üretken olmasını sağlar ve dağıtımları hızlandırır [7].
- Bir teknoloji kümesine olan uzun vadeli taahhüdü ortadan kaldırır. Yeni bir servis geliştirirken yeni bir teknoloji seçilebilir. Benzer şekilde, mevcut bir hizmette büyük değişiklikler yaparken, yeni bir teknoloji kullanarak yeniden yazılabilir [7].
- Daha iyi ölçeklendirilebilir sistemler tasarlamamıza olanak sağlar [1, 2, 5, 7, 8, 9, 10, 11].

4.2 Mikroservis Mimarisi Zorlukları

- Geliştiriciler, dağıtılmış bir sistem oluşturma ek karmaşıklığı ile uğraşmak zorundadırlar [1, 2, 5, 7, 8, 9].
- Geliştirici araçları (*IDE*), monolitik uygulamalar oluşturmaya yöneliktir ve dağıtılmış uygulamalar geliştirmek için açık destek sağlamaz [7].
- Test yapmak daha zordur [7].
- Dağıtılmış işlemler kullanmadan birden fazla hizmeti kapsayan kullanım durumlarını uygulamak zordur [7].
- Birden fazla hizmeti kapsayan kullanım durumlarını uygulamak, ekipler arasında dikkatli bir koordinasyon gerektirir [7].
- Üretimde, birçok farklı hizmet türünden oluşan bir sistemin dağıtımı ve yönetilmesinin operasyonel karmaşıklığı da vardır [1, 2, 5, 7, 8, 9, 10, 11].
- Sistemi izleme (*monitoring*) zorlaşmaktadır [1,2,7,10,11].

- Ağ kesintileri ve yavaşlıklarını ele almak gerekir [7].
- Servis sınırlarını iyi belirlemek gerekir. Bunun için DDD yaklaşımı önerilmektedir [1,2,5,6,7].

4.3 Mikroservis Mimarisi Faktörleri

Yaptığımız çalışmalar sonucunda mikroservis mimari çözümlerini değerlendirmek için aşağıdaki (Tablo 1) faktörleri belirledik.

Tablo 1. Mikroservis Mimarisi Faktörleri

1	**Dağıtım kalıpları** [7] (*Deployment Pattern*)	Servislerimizi nasıl konuşlandıracağız?
2	**İletişim şekli** [7,8,10] (*Communication Style*)	Servisler kendi aralarında ve dış dünya ile nasıl bir iletişim yöntemi izleyecektir?
3	**Harici API** [7,9] (*External API*)	Dış istemciler servisler ile nasıl iletişim kuracaklar?
4	**Servis keşif** [7,8,9,10,11] (*Service Discovery*)	İstemciler, servislerin ağ adreslerini nasıl bulacaklar?
5	**Sunucu olmadan geliştirme – Servis olarak fonksiyon** [9,10,11] (*Serverless Computing, Function as a Service - FaaS*)	DevOps maliyeti azaltarak ve sunucu yönetimi yapmadan, esnek ölçeklenebilir bir uygulamayı nasıl geliştiririm?
6	**Güvenilirlik** [7,9,10,11] (*Reliability*)	Girdi aldığım diğer servislerin çökmesi durumunda, ağ ve servis çökmelerini nasıl önleyeceğiz?
7	**Veri yönetimi** [7,11] (*Data Management*)	Veri tutarlığını nasıl sağlayacağız? Birden çok servis ile ilgili sorgularımızı nasıl yapacağız?
8	**Güvenlik** [7,10,11] (*Security*)	İstek sahibinin kimliği, isteği yerine getiren hizmetlere nasıl iletilir?
9	**Test** [7,11] (*Testing*)	Mikroservis bileşenlerinin testleri nasıl yapılır? Servislerin entegrasyon testleri nasıl yapılır?
10	**Gözlenebilirlik** [7,8,9,11] (*Observability*)	Uygulamamızda ortaya çıkan problemleri nasıl gözlemleriz? Uygulamamızın davranışını nasıl takip ederiz? *Scale-up* ve *Scale-out* işlemlerini nasıl gerçekleştireceğiz?
11	**Yük Dağılımı** [7,8,9,10,11] (*Load Balancing*)	Sunuculara gelen istekleri, sistemin güvenirliği sağlamak ve ağ trafiğini dengelemek için verimli bir şekilde nasıl ele alabiliriz?

5 Araştırma Sorularına Yanıtlar

5.1 AS-1. Teknoloji şirketleri, mikroservis mimarisi çerçevesinde hangi faktörler ile ilgileniyorlar?

Yaptığımız literatür taraması sonucunda mikroservis mimarisi faktörlerini çıkarmış ve bunu bir önceki bölümde açıklamıştık. Bu mimari faktörler aynı zamanda, teknoloji şirketlerinin şu anda ilgilendikleri alanlardır. Şirketler bu alanlarda, çeşitli çözümler üretmek amacıyla çalışmalar yapmaktadırlar.

5.2 AS-2. Teknoloji şirketleri bu faktörler için ne gibi çözümler öneriyorlar?

Birçok teknoloji şirketi bu alanda yenilikçi çözümler önermektedir. Bu çözümleri öneren şirketler birden çok faktöre odaklandığı gibi birkaç faktör üzerine de odaklanabilmektedir. Birden çok faktöre odaklanan şirketlerden Amazon AWS, ölçekleme, yük veya karmaşıklıktan bağımsız olarak, herhangi bir uygulama mimarisini destekleyen bütünleşik bir yapı sağlamaktadır [11]. Diğer bir kuruluş olan Apache'nin misyonu kamu yararına yazılım sağlamaktır. Yazılım dünyasındaki problemli noktalara çözümler üreten takımlar oluşturulur ve açık kaynak kodlu olarak yayınlanır [12]. Netflix ise dünyadaki en büyük medya sağlayıcı şirketlerdendir. Kendi bünyesinde büyük veri, konuşlandırma, güvenlik, performans, güvenilirlik gibi konularda açık kaynak kodlu çözümler üreten bir yazılım ekibi bulundurmaktadır [13]. Bu üç şirket dışında diğer bazı şirketler de çeşitli çözümler önermektedir. Tablo 2'de hangi faktör için ne gibi çözümlerin getirildiği belirtilmiştir. Ayrıca bu tabloda belirtilen çözümler ile nelerin amaçlandığı incelenmiş ve Tablo 3'te açıklanmıştır.

Tablo 2. Mikroservis Mimarisi Faktörleri İçin Çözümler

Faktör	AWS	Apache	Netflix	Diğer
Dağıtım kalıpları	EC2 Container Service	Marathon, Mesos		Docker, Kubernetes
İletişim şekli	Amazon SNS, Amazon SQS	Apache Kafka		RabbitMQ, REST, Linkerd
Harici API	AWS API Gateway	Apache Http Server	Zuul	Nginx, Spring Cloug Gateway, Linkerd, Google Cloud Endpoints
Servis keşfi	Amazon ECS Service Discovery	Apache Zookeper	Eureka	Kubernetes, Consul
Güvenilirlik			Hystrix	Spring Security, Linkerd
Veri yönetimi				SAGA, Event Sourcing, CQRS
Güvenlik	AWS Security, AWS Single Sign- On			JWT, CAS
Test				Spring Cloud Test
Gözlenebilirlik	AWS Cloud Watch	Marathon		Zipkin, Kubernetes
Yük dağılımı	Amazon EC2 Elastic Load Balancer	Apache Http Server	Ribbon client, Zuul	Nginx
Sunucu olmadan işlem	AWS Lambda			Google Cloud Function, Microsoft Azure Funciton

Tablo 3. Mikroservis Mimarisi Çözümleri ve Tanımları

Faktör	**Çözüm**	**Açıklama**
Dağıtım kalıpları	EC2 Container Service	Amazon EC2 Container Service, Docker kapsayıcılarını destekleyen ve Amazon EC2 örneklerinin yönetilen kümeleri üzerinde uygulamalarınızı kolayca çalıştırabilmenizi sağlayan yüksek düzeyde ölçeklenebilir, yüksek performanslı bir kapsayıcı yönetimi hizmetidir. Amazon EC2 Container Service, basit API çağrılarıyla kapsayıcı özellikli uygulamaları başlatmanızı ve durdurmanızı sağlar.
	Marathon	Marathon, Mesosphere'in DataCenter İşletim Sistemi (*Datacenter Operating System – DC/OS*) ve Apache Mesos için bir konteyner orkestrasyon platformudur.
	Mesos	Apache Mesos, makineden (fiziksel veya sanal) uzaktaki CPU, bellek, depolama ve diğer bilgi kaynaklarını soyutlar ve hata toleranslı ve elastik dağıtılmış sistemlerin kolayca oluşturulmasını ve çalıştırılmasını sağlar.
	Docker	Konteynır olarak da bilinen işletim sistemi düzeyinde sanallaştırmayı gerçekleştirir. Docker'ın sanallaştırma yapısı, klasik bilinen sanal makinelerden farklı olarak bir Hypervisor katmanına sahip değildir. Bunun yerine Docker, Docker Engine üzerinden, işletim sistemine erişmekte ve sistem araçlarını paylaşımlı olarak kullanmaktadır.
	Kubernetes	Kubernetes, konteynerli uygulamaların dağıtımını, ölçeklendirilmesini ve yönetimini otomatikleştirmek için açık kaynaklı bir sistemdir.
Gözlene-bilirlik	AWS Cloud Watch	Amazon Cloud Watch, AWS bulut kaynakları ve AWS'de çalıştırılan uygulamalar için bir izleme servisidir.
	Zipkin	Zipkin dağıtılmış bir izleme sistemidir. Microservice mimarilerindeki gecikme sorunlarını gidermek için gereken zamanlama verilerini toplamaya yardımcı olur. Bu verilerin toplanmasını ve aranmasını yönetir. Zipkin'in tasarımı Google Dapper çalışmasına dayanmaktadır.
Güveni-lirlik	Hystrix	Hystrix, uzak sistemlere, hizmetlere ve 3. parti kitaplıklarına erişim noktalarını ayırmak, basamaklı başarısızlığı durdurmak ve başarısızlığın kaçınılmaz olduğu karmaşık dağıtılmış sistemlerde esnekliği sağlamak için tasarlanmış bir gecikme ve hata tolerans kütüphanesidir.
	Spring Security	Spring Security güçlü ve son derece özelleştirilebilir bir kimlik doğrulama ve erişim kontrol çerçevesidir. Spring tabanlı uygulamaların güvenliğini sağlamak için bir standarttır.
Güvenlik	AWS Security	AWS, gizliliği artırmak ve ağ erişimini kontrol etmek için çeşitli güvenlik özellikleri ve hizmetler sunar.
	Json Web Token (JWT)	JSON Web Token'ları, haberleşen iki sistem arasında kullanıcı doğrulama, kullanıcı tanıma, veri bütünlüğünü ve bilgi güvenliğini koruma gibi noktalarda kullanılmaktadır.
	AWS Single Sign-On	AWS Tek Oturum Açma (SSO), çoklu AWS hesaplarına ve iş uygulamalarına SSO erişimini merkezi olarak yönetmeyi kolaylaştıran bir bulut SSO hizmetidir.
	Central Authenticati on Service (CAS)	Web için kurumsal çok dilli tekli oturum açma çözümüdür ve kimlik doğrulama ve yetkilendirme ihtiyaçlarınız için kapsamlı bir platformdur. CAS açık ve iyi dokümante edilmiş bir kimlik doğrulama protokolüdür.
Harici API	AWS API Gateway	Amazon API Gateway, geliştiricilerin herhangi bir ölçekte API oluşturmasını, yayınlamasını, sürdürmesini, izlemesini ve güvenliğini sağlamasını kolaylaştıran, tamamen yönetilen bir servistir. AWS

		Yönetim Konsolu'nda birkaç tıklamayla, Amazon Elastic Compute Cloud (Amazon EC2) üzerinde çalışan iş yükleri, AWS Lambda üzerinde çalışan kod veya herhangi bir web uygulaması gibi arka uç hizmetlerinden veri, iş mantığı veya işleve erişmek için uygulamalar için, "ön kapı" olarak işlev gören bir API oluşturabilirsiniz.
	Apache Http Server	Apache HTTP Sunucusu Projesi, UNIX ve Windows dâhil olmak üzere modern işletim sistemleri için açık kaynaklı bir HTTP sunucusu geliştirmek ve sürdürmek için gerçekleştirilmiş bir projedir. Bu projenin amacı, mevcut HTTP standartlarıyla senkronize olarak HTTP hizmetleri sağlayan güvenli, verimli ve genişletilebilir bir sunucu sağlamaktır.
	Zuul	Zuul, cihazlardan ve web sitelerinden gelen tüm isteklerin servislere iletir. Yük dengeleme özelliği de bulunmaktadır
	Nginx Api Gateway	Nginx, ters proxy, yük dengeleyici, e-posta proxy ve HTTP önbelleği olarak da kullanılabilen bir web sunucusudur.
	Spring Cloud Gateway	Spring Model – Görünüm – Kontrolör (*Model View Controller – MVC*) 'nin üzerinde bir API Gateway oluşturmak için bir kütüphane sunmaktadır. Spring Cloud Gateway, API'lere yönlendirmek ve onlara güvenlik, izleme/metrikler ve esneklik gibi çapraz kaygılar sunmak için basit ama etkili bir yol sağlamayı amaçlamaktadır.
	Google Cloud Endpoints	NGINX tabanlı bir proxy ve dağıtık mimari ile, performans ve ölçeklenebilirlik sağlar. API geliştirmenin her aşaması için ihtiyaç duyulan araçları sağlar ve Google Cloud Monitoring, Cloud Trace, Google Cloud Logging ve Cloud Trace ile entegredir.
İletişim şekli	Amazon SNS	Amazon Basit Bildirim Servisi (*Simple Notification Service – SNS),* mesajların teslimatını koordine etmek için, esnek, tamamen yönetilen bir mesajlaşma ve mobil bildirimler servisidir.
	Amazon SQS	Amazon Basit Kuyruk Hizmeti (*Amazon Simple Queue Service – SQS*), mikroservisleri, dağıtılmış sistemleri ve sunucu olmayan uygulamaları ayırmayı ve ölçeklemeyi kolaylaştıran tamamen yönetilen bir mesaj kuyruklama hizmetidir.
	Apache Kafka	Kafka, gerçek zamanlı veri pipeline'ları ve akış uygulamaları oluşturmak için kullanılır. Yatay olarak ölçeklendirilebilir, hataya dayanıklıdır. Mesajlaşma için kullanılır.
	RabitMQ	Bir mesaj atayıcıdır. Çoklu mesajlaşma protokollerini destekler. RabbitMQ, yüksek ölçekli, yüksek kullanılabilirlik gereksinimlerini karşılamak için dağıtılmış ve birleşik konfigürasyonlarda dağıtılabilir.
	REST	Web üzerinde bilgisayar sistemleri arasında standartların sağlanması için bir sistemdir ve sistemlerin birbirleriyle iletişim kurmasını kolaylaştırır.
	Linkerd	Linkerd, cloud-native uygulamalara güvenilirlik, güvenlik ve görünürlük sağlayan bir hizmet ağıdır (*service mesh*).
Servis keşfi	AWS ECS Service Discovery	Amazon ECS entegre servis keşiflerini içerir. Bu, bir ECS servisinin kendisini Amazon Route 53'te öngörülebilir bir Alan Adı Sistemi (*Domain Name System – DNS)* adıyla otomatik olarak kaydetmesini mümkün kılar. Servisler yük veya konteyner sağlığına yanıt olarak yukarı veya aşağı ölçeklendiğinde, Route 53 barındırılan bölgesi güncel tutulur.
	Apache Zookeper	ZooKeeper, yapılandırma bilgilerini korumak, adlandırma, dağıtılmış eşitleme ve grup hizmetlerini sağlamak için merkezi bir hizmettir. Bütün bu servisler dağıtılmış uygulamalar ile bir şekilde kullanılmaktadır.

		Apache ZooKeeper, son derece güvenilir dağıtılmış koordinasyon sağlayan açık kaynaklı bir sunucu geliştirmek ve sürdürmek için kullanılır.
	Eureka	Eureka, orta düzey sunucuların yük dengelemesi ve yük devretme amacıyla hizmetlerin yerini almak için öncelikle AWS bulutunda kullanılan bir REST tabanlı hizmettir.
Sunucu olmadan işlem	AWS Lambda	AWS Lambda, kod hazırlama veya sunucular yönetmeden kod çalıştırmaya izin verir. Yalnızca tüketilen hesaplama zamanı için ödeme yapılır, kodunuz çalışmadığında ücret alınmaz.
	Google Cloud Function	Google tarafından geliştirilen olay tabanlı sunucusuz işlem platformudur.
	Microsoft Azure Function	Microsoft tarafından geliştirilen olay tabanlı sunucusuz işlem platformudur.
Test	Spring Cloud Contract	Spring Cloud Contract, kullanıcıların Tüketici Güdümlü Sözleşmeler yaklaşımını başarılı bir şekilde uygulamalarına yardımcı olan bir projesidir.
Veri yönetimi	SAGA	Mikroservis dünyasında veri tutarlılığını sağlamak için geliştirilen bir örüntüdür. *BASE* veri tabanı işlem özelliklerini temel alır.
	Event Sourcing	Event Sourcing'in temel fikri, bir uygulamanın durumundaki her değişikliğin bir olay nesnesinde yakalanmasını sağlamaktır ve bu olay nesneleri, uygulama durumuyla aynı yaşam süresi boyunca uygulandıkları sırada saklanır.
	Command Query Separation (CQRS)	İyi tasarlanmış bir nesnenin, komutlar veya sorgular olan yöntemlere sahip olması gerektiğini belirtir. Bir komut, bir nesnenin durumunu değiştirir, ancak herhangi bir veri döndürmez, sorgu bir veri döndürür ve herhangi bir durumu değiştirmez. Yöntemleri bu iki kategoriye bölerek, sisteminizin durumunu neyin değiştirip neyin değiştirmediğini daha iyi anlatmaya çalışan bir örüntüdür.
Yük dağılımı	Amazon EC2 Elastic Load Balancer	Elastik Yük Dengeleme, gelen uygulama trafiğini Amazon EC2 örnekleri, konteynerleri ve IP adresleri gibi birden çok hedefe otomatik olarak dağıtır.
	Ribbon Client	Yük dengeleme için Eureka servis keşfini kullanır. Hata toleransı vardır ve birçok protokolde çalışır (HTTP, UDP, TCP).

Faktör bazında yaptığımız çözümlerin araştırılması sonucunda hemen hemen her faktör için çözümlerin var olduğu ve ekiplerin bunlara kolayca adapte olabileceği değerlendirilmiştir. Veri yönetimi ve test konusunda yeterli çözümün olmaması, bu faktörlerin gelişmeye açık birer alan olduklarını göstermektedir. Yazılım sistemimizi birden çok mikroservis olarak geliştirdiğimizde işlem bütünlüğü kritik önem kazanmaktadır. Farklı mikroservisleri ilgilendiren bir süreçte, işlem bütünlüğünü koruyarak veri yönetimi sağlanmalıdır. Benzer şekilde, bu mikroservislerin biribiri ile iletişimlerinde bir sıkıntı olup olmadığını da gerçek ortama çıkmadan test etmek gerekmektedir. Mikroservislerin iletişimlerinin test edilmesi, bu noktada zorlu bir aşama olarak görülmektedir.

6 Sonuçlar

Bu bildiri kapsamında mikroservis mimarisi ve temel faktörleri incelenmiştir. Daha çok teknoloji şirketlerinin öncülüğünde gelişen bu dünyada mimarinin gerekleri ortaya konmuş ve monolitik mimariye göre faydaları ve zorlukları açıklanmıştır. Mikroservis mimarisini kullanacak ekiplerin faydalanabileceği faktörler literatürdeki birçok kaynak ve proje incelenerek belirlenmiştir. Bu mimari faktörler için teknoloji firmalarının ne gibi proje veya ürünler geliştirdikleri araştırılmış ve bu proje veya ürünler hakkında bilgi verilmiştir. Yaptığımız çalışmanın sonuçlarından biri de teknoloji firmalarının son yıllarda bu alana büyük önem verdiği ve birçok proje geliştirdikleridir. Ayrıca bulut teknolojisinin gelişmesiyle, mikroservis dünyasındaki gelişmeler hızlanmış ve teknoloji firmalarının işbirlikleri artmıştır. Bu amaçla, birlikte teknoloji geliştirme yöntemleri benimsenmiş ve topluluklar oluşmuştur. *Cloud Native Computing Foundation* buna bir örnektir. Sonuç olarak, teknoloji firmalarının mikroservis mimarisi faktörlerinin çoğuyla ilgilendikleri ve bunlar için kapsamlı çözümler ürettikleri gözlemlenmiştir.

Bu çalışmanın diğer bir katkısı da mikroservis mimarisini kullanacak ekipler için, mimariye hızlı adaptasyon sağlamak adına, her bir faktör için ne gibi çözümlerin var olduğunun belirlenmiş olmasıdır. Bu şekilde ekipler kendileri için uygun olan faktörleri belirleyip çözüm alternatiflerini değerlendirebilir. Ekiplerin güçlü ve zayıf yönleri görerek daha hızlı karar vermelerini destekleme amacıyla, çözümlerin faktör bazında karşılaştırmalı incelemesi ise gelecek çalışma olarak planlanmıştır.

Kaynaklar

1. Lewis, J and Fowler, M. Microservice. http://martinfowler.com/articles/mikroservices.html, March 2014.
2. Newman, S. Building Microservices. O'Reilly. 2015.
3. Pahl, C. Containerization and the paas cloud. IEEE Cloud Computing, 2(3):24–31, 2015.
4. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L., Microservices: yesterday, today, and tomorrow.arXiv preprint arXiv:1606.04036, 2016.
5. Johannes T. Microservices. IEEE Software, 32(1):116–116, 2015.
6. Evans E. Domain-Driven Design: Tacking Complexity in the Heart of Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
7. Microservice.io Anasayfası, http://microservices.io/
8. Thoughtworks Mikroservis Anasayfası: https://www.thoughtworks.com/insights/microservices
9. Infoq Mikroservis Anasayfası, https://www.infoq.com/microservices
10. Pivotal Mikroservis Anasayfası, https://pivotal.io/microservices
11. AWS Microservice Anasayfası, https://aws.amazon.com/tr/microservices/
12. Apache Anasayfası, http://www.apache.org/
13. Netflix OSS Anasayfası, https://netflix.github.io/

14. Vural H., Koyuncu M., Guney S. (2017) A Systematic Literature Review on Microservices. In: Gervasi O. et al. (eds) Computational Science and Its Applications – ICCSA 2017. ICCSA 2017. Lecture Notes in Computer Science, vol 10409. Springer, Cham
15. Pahl, C, Jamshidi,P . 2016. Microservices: A Systematic Mapping Study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2 (CLOSER 2016), Jorge Cardoso, Donald Ferguson, Víctor Méndez Muñoz, and Markus Helfert (Eds.), Vol. 1 and 2. SCITEPRESS - Science and Technology Publications, Lda, , Portugal, 137-146. DOI: https://doi.org/10.5220/0005785501370146