T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

MİKROSERVİS MİMARİSİNİN TEMELLERİ

ARAŞTIRMA SERİSİ - SAYI 10

BİLGEM

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
HTTP	Hyper Text Transfer Protocol (Hiper Metin Transfer Protokolü)
REST	Representational State Transfer (Temsili Durum Transferi)
SOAP	Simple Object Access Protocol (Basit Nesne Erişim Protokolü)
SOA	Service Oriented Architecture (Servis Odaklı Mimari)

Yazar
Gökçenur ÇINAR

Yayın Koordinatörü
Elif ŞENYİĞİT

Editörler
Özay DUMAN
Sevinç KARAKAŞ
Tuğçe YILMAZ

Tasarım
Şeyma KOÇER

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte/

İçindekiler

Önsöz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

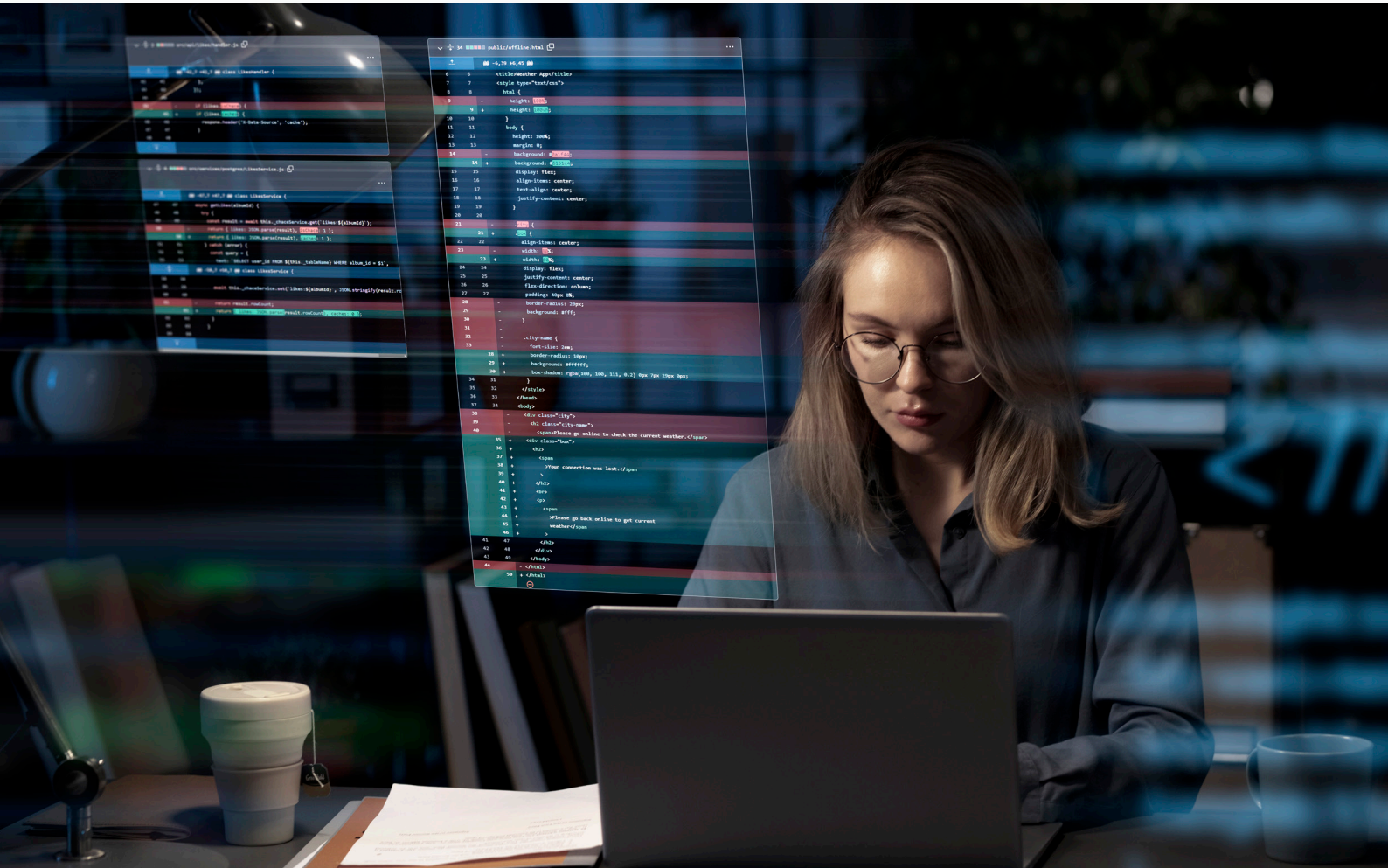
TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.

Giriş

Mikroservis mimarisi, büyük ve karmaşık yazılım uygulamalarını geliştirme ve yönetme yaklaşımında önemli bir dönüşümü temsil eder. Bu yaklaşım, yazılım uygulamalarını küçük, bağımsız hizmetlere böler ve her birini ayrı bir mikroservis olarak çalıştırır. Her mikroservis, özel bir işlevi yerine getirir, veri tabanına erişir ve kullanıcı arayüzünü yönetir. Mikroservisler, genellikle hafif konteyner teknolojileri kullanılarak dağıtılır ve bu sayede hızlı bir şekilde ölçeklendirilebilir.

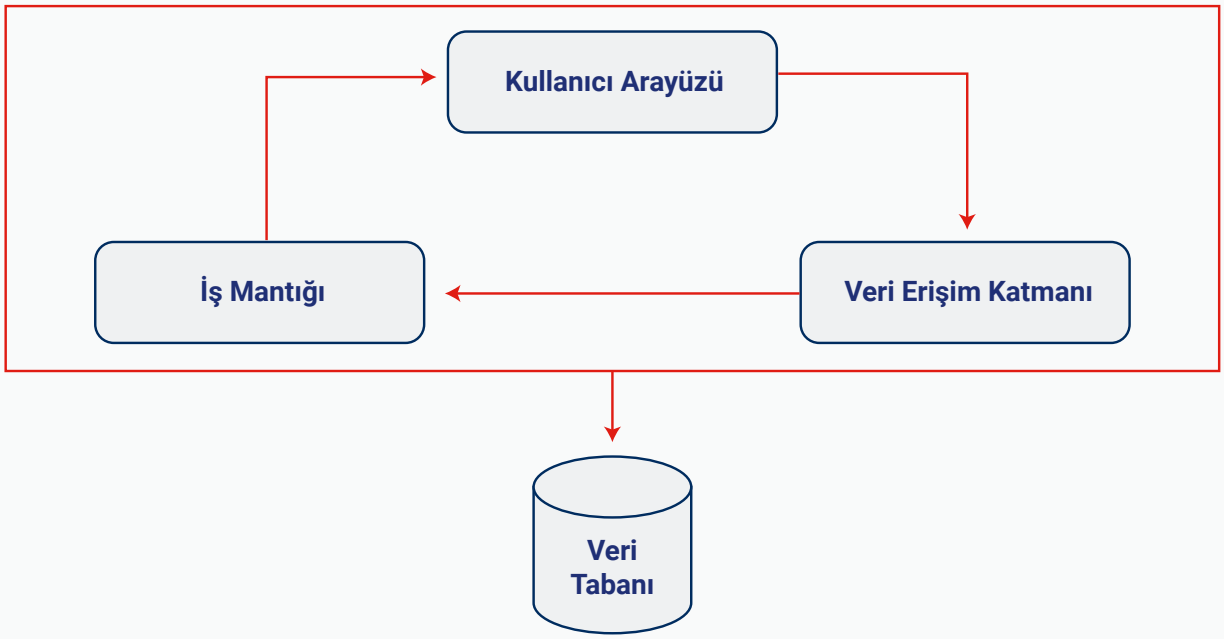
Bu yaklaşımın temel amacı, büyük monolitik uygulamaların karmaşıklığını azaltmak, geliştirme süreçlerini hızlandırmak ve bakım maliyetlerini düşürmektir. Mikroservisler, farklı teknolojilere ve dil seçimlerine izin verir, bu da ekiplerin kendi alanlarında uzmanlaşmalarını sağlar. Ayrıca, her mikroservisin bağımsız olarak dağıtılabilmesi, hataların izolasyonunu kolaylaştırır ve hizmet kesintilerini minimize eder.

Ancak, mikroservis mimarisi getirdiği bu avantajlarla birlikte yönetim zorlukları ve iletişim maliyetleri de beraberinde getirebilir. Her bir mikroservisin koordinasyonu ve hata toleransı gereksinimleri, doğru bir şekilde ele alınmalıdır. Sonuç olarak, mikroservis mimarisi, büyük ölçekli ve ölçeklenebilir yazılım projeleri için güçlü bir alternatif sunar, ancak dikkatli bir planlama ve yönetim gerektirir. Bu çalışma kapsamında mikroservis mimarisine ilişkin temel bilgiler, avantajları, maliyeti ve uygulanması ele alınmaktadır.

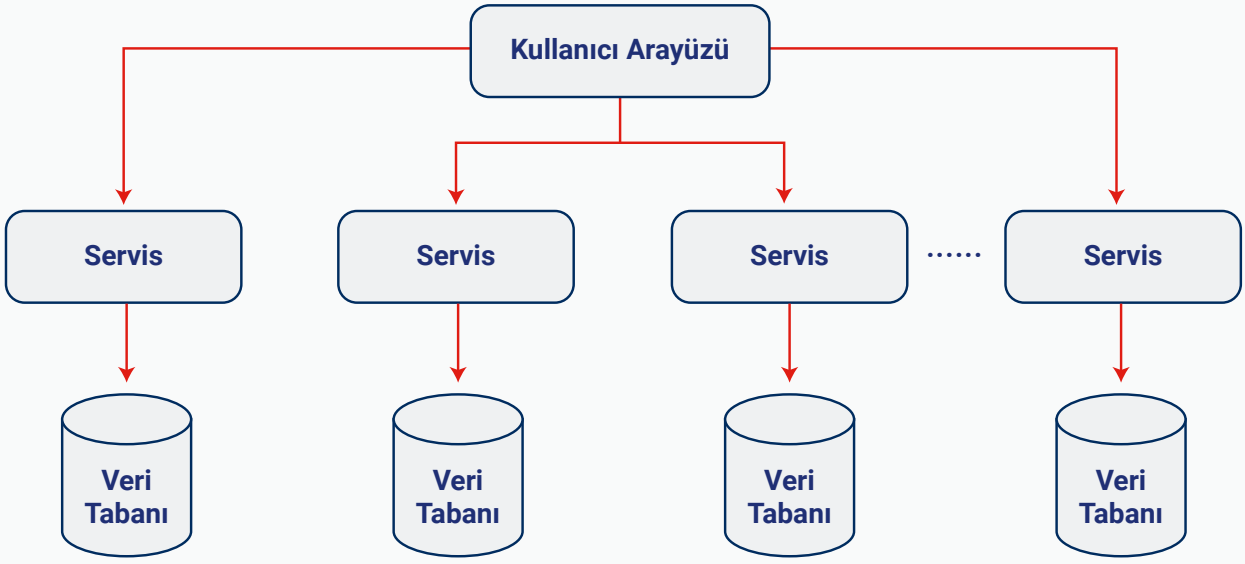


Monolitik Uygulamalar

Monolitik uygulamalar, çoklu işlevleri yerine getirmek için tasarlanmış genellikle servislerin birbirine bağımlı olduğu işlevleri kapsayan karmaşık uygulamalardır. Monolitik mimarilerde, uygulama ölçeği büyüdükçe, geliştirme süreci, yazılım testi, kurulum ve ölçeklendirme süreçleri de buna bağlı olarak zorlaşmakta ve belirli zaman sonra kontrol edilmesi güç uygulamalara dönüşmektedir. Büyük ölçekli uygulama geliştirirken karşılaşılan bu sorunlar için önerilen en önemli çözüm yönteminden biri mikroservis mimarisidir. Şekil 1'de klasik uygulama yapısı ile Şekil 2'de mikroservis mimari yapısı genel olarak gösterilmiştir.



Şekil 1. Klasik Monolitik Uygulama Yapısı



Şekil 2. Mikroservis Mimarisi Temel Yapısı

Monolitik mimari kullanılarak geliştirilen uygulamalar içermiş olduğu servis ve bileşenlerle birbirine bağlı olarak çalışan büyük bir sistem olarak tasarlanır. Genellikle tek bir kod tabanı barındırır ve ön yüz ile arka yüz genelinde aynı teknolojiler ve programlama dili kullanılmaktadır. Bu durum da bağımsız ölçeklendirme veya kod sürdürülebilirliği gibi amaçlarla servislerin ayrıştırılmasını zorlaştırmaktadır. Uygulama içerisinde kullanılan teknolojiyi, dili veya çerçeveyi değiştirmek son derece zordur. Mikroservis mimarisi, iş işlevselliğine dayalı küçük ve bağımsız modüler olarak oluşturulmaktadır. Mikroservis uygulamalarında her proje ve servis kod düzeyinde birbirinden bağımsızdır. Bu nedenle, tamamen yapılandırılması, devreye alınması ve ihtiyaçlar doğrultusunda ölçeklendirilmesi kolaydır. Mikroservis yazılım mimarisi, bir sistemin birçok sayıda küçük, bireysel ve bağımsız hizmete bölünmesine olanak tanır. Her servis esnek, sağlam ve düzenlenebilir durumdadır. Servis içerisindeki süreçler otonom olarak işletilir ve tanımlanmış API'ler aracılığıyla iletişim kurarlar. Her mikroservis, farklı bir platformda farklı bir programlama dili ile oluşturulabilmektedir. Mikroservislerin aksine, monolitik mimari, kod bileşenlerinin aynı bellek alanını paylaşan tek bir bütünleşik birim olarak birlikte çalışabilir tasarlandığı anlamına gelir. Monolitik yaklaşım kullanılarak oluşturulan uygulamaların bileşenleri birbirine bağımlıdır. Monolitik sistemde herhangi bir değişiklik veya güncelleme yapılmak istendiğinde, sürüm paketlerinin bir kerede oluşturulması ve dağıtımının yapılması gerekmektedir. Ölçeklenebilirlikte de aynı şey geçerlidir. Yalnız sorumlu olduğu servis değil, bütün sistem birlikte ölçeklenir. Monolitik mimari ile yeni bir teknolojinin uygulanması zor olabilir ve yeni bir platform veya çerçeve kullanılmak istendiğinde tüm sistemin yeniden tasarlanıp geliştirilmesi gerekebilir.

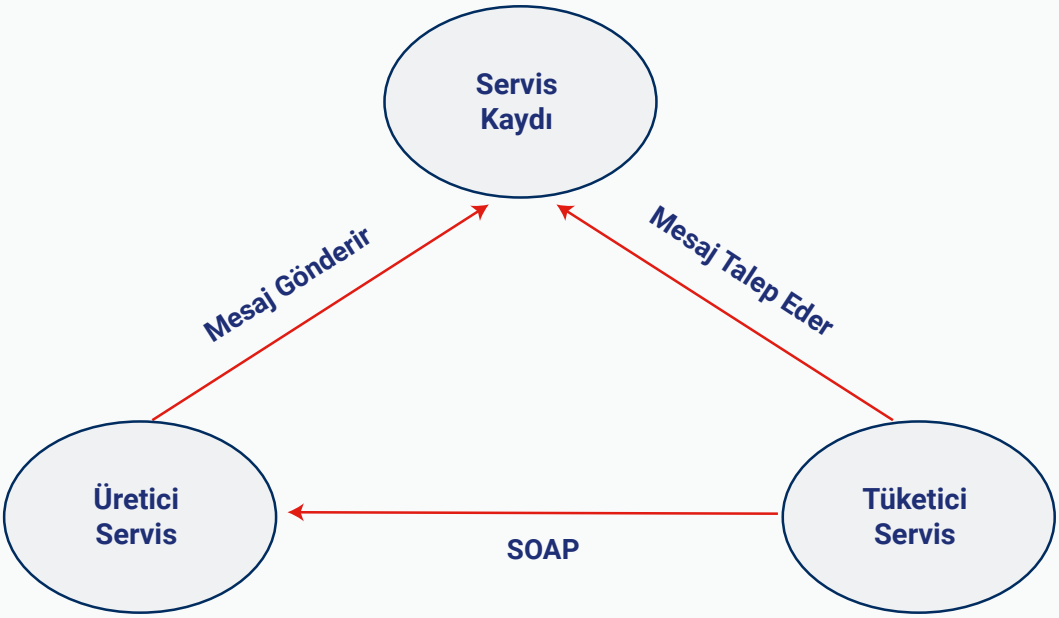
Mikroservis Mimarisi

Mikroservis, tanımı gereği küçük, geliştirilmesi ortalama iki üç hafta süren, bağımsız, diğer mikroservis bileşenleri ile sıkı sıkıya bağımlılığı bulunmayan, tek başına çalışabilen, kendine ait veri tabanı olan, geliştirme sürecinden kuruluma kadar bağımsız olan, yatayda ve dikeyde kendi başına ölçeklenebilen uygulamalardır.

Modüler yapıya sahip bu mimari de birbirlerine olan bağımlılıkları az, her bir parça kendi içerisinde veri tabanı, mesaj iletim mekanizması ayrı olarak çalışabilen farklı servisler bir araya gelir.

1. SOA'dan Farkları

Mikroservis mimarisi birbirinden bağımsız uygulamalar geliştirmek için kullanılan SOA üzerine kurulmuş dağıtık sistemler olmasına rağmen farklılıklar içermektedir. SOA, büyük ve karmaşık uygulamalar için uygulanabilir bir mimari yöntemdir. Birçok farklı uygulama ile bütünleşme gerektiren ortamlar için kolaylıkla uygulanabilir bir mimaridir. Ancak, iyi tanımlanmış bir işlem akışına sahip iş akışı tabanlı uygulamaların SOA mimari modellerinin yardımıyla uygulanması zordur. Bu nedenle küçük uygulamalar, ara yazılım mesajlaşma bileşenlerine ihtiyaç duymadıklarından SOA için de ideal değildir. Mikroservis modeli, daha küçük ve iyi bölümlenmiş web tabanlı sistemler için uygundur (Richards, 2015). Şekil 3'te SOA yapısı gösterilmiştir.



Şekil 3. SOA Yapısı

SOA ile Mikroservis Mimarisinin Temel Farkları:

- SOA modeli, bir uygulama içerisinde kullanılan bütün servislerde tek bir veri depolama katmanına sahiptir. Fakat mikroservis mimarisinde uygulama içerisinde ihtiyaç duyulan veri hizmetleri bir veya birden fazla veri tabanı üzerinde ayrışmaya gider. Böylelikle servisler veri katmanında da birbirlerine daha az bağımlı olurlar.
- SOA modelinde, ağırlıklı olarak SOAP tabanlı iletişim yöntemleri kullanılırken mikroservis mimarisinde servisler arasındaki iletişim için HTTP RESTful API'ler veya RPC'ler kullanılmaktadır.
- SOA modeli kullanılarak geliştirilen uygulamalar servisler üzerindeki yeniden kullanılabilirliği sağlamayı temel amaç olarak görürken, mikroservis mimarisinde ağırlıklı olarak servisler arasındaki bağımlılığı azaltma ve ayrışma hedeflenmektedir.
- SOA modelinde servis üzerinde yapılacak olan değişikliğin uygulamanın bütününü etkilerken, mikroservis mimarisinde sadece hizmeti sağlayan servis üzerinde değişiklik yapılabilir olup büyük değişiklikler yeni servis oluşturulması ile var olan servisin çalışabilirliğini etkilemeden karşılanabilmektedir.
- SOA modeli, servisler arasında sistem kaynaklarının paylaşılması üzerine oluşturulmuştur. Uygulama verileri servisler arasında ortak alanda tutularak paylaşılır. Fakat mikroservis mimarisi birbirinden bağımsız kaynaklarla çalışabilen servislerin geliştirilmesi üzerine tasarlanmıştır. Sistem bileşenlerinin paylaşımı yerine her bir servis, kendi sistem bileşenlerini ve depoladıkları verileri bağımsız olarak kullanmaktadır.
- SOA modeli, çoklu uygulamaları desteklemesine rağmen genellikle üç veya dört servisten oluşmaktadır. Fakat mikroservis mimarisinin kullanıldığı uygulamalar ile çok daha fazla servisin bir araya gelerek sistemin tasarlanmasına olanak sağlamaktadır.

- SOA modeli ile geliştirilmiş uygulamalarda yazılımın boyutu mikroservislere göre daha büyüktür. Mikroservis mimarisi birbirinden bağımsız küçük boyutlu servis parçalarından oluşmaktadır.

Mikroservis Mimarisinin Avantajları

Mikroservisler basit yapılara sahiptir ve temel ilkesi basitliktir. Daha küçük, bir araya getirilebilir parçalara bölündüklerinde uygulamaların oluşturulması ve bakımı daha kolay hale gelir. Her mikroservis ayrı bir kod parçası olarak geliştirildiği için kodun yönetilmesi de daha az maliyetli hale gelir. Mimari tasarım yapılırken uygulama içerisinde temeli oluşturan iş mantığı servis bazında ayrıştırılmaktadır. Her servis kendi içerisinde tanımlı olan işlevi yerine getirmektedir.

Mikroservisler birbirlerinden bağımsız yapıya sahiptir. Servisler birbirlerinden ayrı sunucular üzerinde çalışır. Uygulamalar mikro arka yüz ve mikro önyüzlere sahiptir. Bir servisin hizmet veremiyor olması durumunda diğer servisler işleyişlerine devam eder. Geliştiriciler, servisin API ile belirlenmiş şartlara uygun olması koşuluyla, uygulanabilir farklı teknolojileri farklı servisler için seçebilirler. Ayrıca sürüm dağıtımları da servisler arasında bağımsız olarak yürütülür. Bir serviste gerçekleşmiş olan değişimin dağıtımı için diğer servislerin beklenmesine gerek duyulmamaktadır. Mikroservisler farklı programlama dilleri, veri tabanları ve yazılım ortamları kullanılarak uygulanabilir. Bu, her servisin bağımsız olarak dağıtılmasına, yeniden oluşturulmasına, yeniden konumlandırılmasına ve yönetilmesine olanak tanır. Örneğin, bir mikroservis çok fazla bellek kullanırsa veya işlemciye ağır bir yük bindirirse, yalnızca bu hizmeti etkiler. Genel olarak konuşursak, bir mikroservisle ilgili herhangi bir sorun tüm sistemi etkilemez ve tek tek mikroservislerin arızası nispeten hızlı bir şekilde telafi edilebilir.

Mikroservis mimarisi, uygulamaları, geliştirme süreci daha hızlı olan yönetilebilir servislere ayırarak üretkenlik ve hız sorununu çözer. Geliştirme ekipleri, farklı ekiplerde devam eden geliştirme süreci çalışmalarını beklemek zorunda kalmadan aynı anda farklı bileşenler üzerinde çalışabilmektedir. Bu tür bir mimari, kalite güvence süreçlerini hızlandırmak için de çok kullanışlıdır, çünkü her mikroservis ayrı ayrı ve henüz geliştirme süreci tamamlanmamış olsa bile test edilebilmektedir (Soldani, Tamburri ve Heuvel, 2018).

Mikroservisler sürdürülebilirdir. Yazılım geliştirme ve değişiklik yönetimi süreçleri hızlıdır. Yazılım geliştirme, doğrulama ve geçerleme süreçleri mikroservisler arası bağımsız olarak ilerler. Değişikliklerin uygulanması bu nedenle kolaydır. Mikroservis geliştirilmeye başlandığında kullanılan teknolojiler, geliştirme süreçlerinin ilerleyen zamanlarında değiştirilebilir. Servisler yönetilmesi kolay yapıda olduğundan, teknoloji değişimleriyle eski bir servisi yeniden yazabilmek mümkün hale gelmektedir.

Yukarıda ifade edilen bilgiler doğrultusunda mikroservis mimarileri, geliştirici ekipleri bağımlı kod yazmak yerine işlevselliğe odaklanmaya olanak sağlıyor. Aynı servis ihtiyaçlara bağlı olarak birden fazla iş sürecinde veya farklı iş kanallarında tekrar kullanılabilmektedir.

Mikroservis Mimarisinin Maliyeti

Mikroservis mimarisi kullanılarak geliştirilen uygulamalarda servisler arasındaki iletişim karmaşıktır. Mimari içerisindeki bileşenler bağımsız servisler olduğundan, parçalar arasında gönderilip alınan mesaj ve taleplerin karşılanamaması durumunun veri bütünlüğünü koruyacak şekilde yönetilmesi gerekmektedir. Örneğin, bir servisin göndermiş olduğu eş zamanlı olmayan bir mesajın sistem içerisinde kaybolmaması veya mesajın tüketilememesi durumunda gerekli çözüm yoluna hızlı bir şekilde gidilmesi gerekmektedir. Yönetimi sağlanamamış bir mesaj servisler arası veri tutarsızlıklarına veya mesaj kanallarında yığılmalara neden olarak isteklere cevap verememe durumuna neden olabilmektedir. Buna benzer durumların önüne geçilebilmesi için geliştirici ekiplerin daha fazla kod yazmak zorunda kalabilmektedir.

Mikroservis mimarisinde servis sayısındaki artışa bağlı olarak sistem kaynak tüketimi de oldukça artmaktadır. Birden çok veri tabanı ve işlem yönetimi bu açıdan maliyetli olabilmektedir. Her mikroservis genellikle kendi bağımsız veri tabanına sahiptir. Bu durum; veri tabanı lisansları, depolama maliyetleri ve veri tabanı yönetimi için ekstra harcamaları gerektirir. Bu maliyetin yönetilebilmesi için de aynı veri tabanında her servis için ayrı bir şema oluşturularak çözüm sağlanabilmektedir.

Mikroservis mimarisinde sistemin genel bir testini yapmak zordur. Monolitik bir yaklaşımda, bir sunucuda uygulamanın başlatılması ve temeldeki veri tabanıyla bağlantısının sağlanması yeterlidir. Fakat mikroservislerde, bütün servislerin ortak testlerinin gerçekleşebilmesi için ayrı ayrı hazır durumda bulunmaları gerekmektedir. Her bir mikroservisin geliştirilmesi, test edilmesi ve bakımı ekstra kaynaklar gerektirir. Mikroservisler arasındaki iletişimi test etmek ve sorunları çözmek için ek test süreçleri gerekebilir.

Mikroservis mimarisinde hata ayıklama sorunları daha fazla yaşanabilir. Her servisin kendi içerisinde oluşturduğu sistem günlükleri bulunmaktadır. Farklı servis günlüklerinin bir araya gelmesi sonucunda da büyük bir veri kümesi oluşmaktadır. Büyük veri kümesi içerisinden ilgili hatanın bulunması eğer etkili bir yönetim ve görselleştirme aracının bulunmaması durumunda hata çözümleme maliyetini artırmaktadır.

Mikroservis mimarisinde servis dağıtımının yönetilmesi zordur. Ayrı bir yönetim aracına ihtiyaç duyulmaktadır. Her bir mikroservisin güvenliği sağlanmalı ve izlenmelidir. Ekstra güvenlik önlemleri ve izleme araçları maliyetli olabilmektedir.

Mikroservis Mimarisinin Uygulanması

Mikroservis mimari yaklaşımı uygulanacak sistemlerde aşağıdaki temel ilkelerin göz önünde bulundurulması gerekmektedir. Merkezi olmayan veri tabanı yapısı kullanılmalıdır. Birden fazla servisin aynı veri şeması üzerinde işlem yapması durumunda, veri katmanında sıkı bir bağlantı oluşabilir. Bu bağı önlemek için, her servisin kendi veri erişim mantığı ve ayrı veri deposu olmalıdır. Mimari uygulanırken, her servise ve veri yapısına en uygun veri depolama yöntemini seçmekte özgürdür.

Servisler arası uç noktaların (end-points) iletişimi için kullanılan API'lerin güvenlik açıklarına neden olmayacak şekilde tanımlanmış olmasına dikkat edilmelidir. İletişim için her zaman HTTP üzerinden REST gibi basit protokollerin kullanılması daha güvenli olacaktır (Dragoni, Lanese, Larsen, Mazzara, Mustafin ve Safina, Mustafin, 2017).

Uzun sürecek veya veri bütünlüğünün korunmasının önemli olduğu işlevler için eş zamanlı olmayan mesajlaşma yöntemi seçilmelidir. Servisler arasında bu iletişim kullanıldığında, veri akışı diğer servisler için engellenmemektedir. Üretici servis mesajı gönderdikten sonra tüketilmesini beklemeden diğer işlemlerine devam edebilmektedir.

Servisler arasında bağlantının en aza indirilmesi gerekmektedir. Servisler gevşek bağlantıya ve yüksek işlevsel uyuma sahip olmalıdır. Bu özellik veri tabanı yapısında ve veri bütünlüğü korunmasında bahsedilmiş olan koşulların uygulanması durumunda sağlanmaktadır.

Ağ geçidinin yönlendirme ve güvenlik şartlarını (kimlik doğrulama, SSL sonlandırma) sağlayacak şekilde yapılandırılması gerekmektedir. Servisler arası paylaşılan merkezi bir kullanıcı havuzuyla iletişime geçen ve kimlik doğrulama bilgilerini alan her mikroservis düzeyinde güvenlik bileşenleri uygulamak yerine, OAuth2 ve OpenID gibi yaygın olarak kullanılan API güvenlik standartlarıyla API Ağ Geçidi düzeyinde kimlik doğrulama tercih edilmelidir. Yetkilendirme sağlayıcısından bir kimlik doğrulama belirteci aldıktan sonra, diğer mikroservislerle iletişim kurmak için kullanılabilir (Thilina, 2021).

Mikroservisler kendi içerisinde bağımsız ve kendi kendini yönetebilir olarak tasarlanmalıdır. Uygulama içerisindeki kod tabanlarının ve geliştirme ekibi üyelerinin birden çok servis veya proje kapsamında ortak kullanımından kaçınılmalıdır. Mikroservis mimarisinde, sistem birden fazla servis ve ara yazılım bileşeninden oluştuğundan, servis içerisindeki hatalar beklenmedik bir kod parçasında karşımıza çıkabilmektedir. Bu tür hataları barındıran kod parçaları için devre kesme, bölme, yeniden denemeler, zaman aşımları, hızlı hata, yük devretme önbelleği, hız sınırlayıcılar, yük ayırıcılar gibi modellerin uygulanması, büyük hata risklerini en aza indirebilir (What is event-driven architecture, 2022).

Mikroservisler, bir proje olarak değil, bir ürün olarak tasarlandığı sürece en doğru şekilde çalışacaktır. Mühendislik süreçlerinin iyi bir şekilde uygulanması ve tutarlılığının sağlanması gerekmektedir. Mimari yapının büyüklüğünden dolayı, servislerin yönetilmesi, dağıtılması ve sürekliliğinin sağlanması ancak bu şekilde mümkün olmaktadır.

1. Servisler Arası İletişim

Monolitik uygulamalarda mesaj iletişimi aynı uygulama içerisinde, tetikleyici bir mesaj gönderilerek (metot çağırma ile) veya HTTP sorguları ile kolaylıkla sağlanabilmektedir. Mikroservis mimarisi kullanılarak geliştirilen uygulamalarda, servisler birbirlerinden bağımsız olarak çalıştığı için servisler arası iletişimde farklı mimari yaklaşımlar kullanılmaktadır. Bu rapor içerisinde yaygın olarak kullanılan "Olay Tabanlı Mimari", "İstek Tabanlı Mimari" ve "Mesaj Tabanlı Mimari" yaklaşımları incelenecektir.

1.1. Olay Tabanlı Mimari

Olay (event), sistem donanımı veya yazılımı için herhangi bir önemli durum değişikliği olarak ifade edilmektedir. Bir olay, sistemin başka bir bölümüne veya servisine bu durum değişikliğinin ortaya çıktığını bildirmek için sistem tarafından gönderilen bir mesaj olan olay bildirimiyle aynı şeyi ifade etmemektedir. Bir olayın kaynağı hem sistem içerisinden hem de arayüz aracılığıyla son kullanıcı tarafından olabilmektedir.

Bir olay tespit edildikten sonra, olay üreticisinden olay tüketicilerine olay için tanımlanmış mesaj kanalları aracılığıyla iletilmektedir. Sistem olaylarının işlenmesi ve yönetimi eş zamanlı olmayan mesajlaşma ile sağlanmaktadır. Olayı dinleyecek tüketici servislerin ilgili olay kanalına tanımlanması gerekmektedir. Olay gerçekleştiği durumda da tüketiciler, bu kanal üzerinden bilgilendirilmektedir. Almış oldukları bu olayı servis bazında işleyebilir veya sadece olay kaydının tutulmasını sağlayacak mesaj tüketimini sağlayabilmektedirler.

Bir olay işleme platformu olarak hizmet verebilecek farklı ara katman olay yöneticileri bulunmaktadır. Apache Kafka, mikroservis mimari yaklaşımında yaygın olarak kullanılan bir veri akış platformudur. Olay akışlarını gerçek zamanlı olarak yayınlamaya, mesaj kanallarına abone olmaya, depolamaya ve işlemeye olanak sağlamaktadır. Apache Kafka, yüksek verim ve ölçeklenebilirliğin hayati önem taşıdığı çeşitli kullanım durumlarını destekler ve belirli uygulamalarda veri paylaşımı için noktadan noktaya bütünleşme ihtiyacını en aza indirerek gecikme süresini büyük oranda azaltmaktadır (Apache Kafka, 2022). Olay akışı işleme, olay akışlarındaki anlamlı kalıpları tespit etmek için kullanılabilir. Basit olay işleme, bir olayın olay tüketicisinde hemen bir eylemi tetiklemesidir. Karmaşık olay işleme, bir olay tüketicisinin kalıpları algılamak için bir dizi olayı işlemesini gerektirmektedir.

Olay tabanlı mimari yaklaşımı, bir üretici / tüketici modeline veya olay akışı modeline dayalı olarak oluşturulabilmektedir.

Üretici Tüketici Modeli: Olay akışına aboneliklere dayalı bir mesajlaşma altyapısıdır. Bu model ile bir olay meydana geldikten veya yayınlandıktan sonra bilgilendirilmesi gereken abonelere (tüketici), üretici servis tarafından gönderilmektedir.

Olay Akış Modeli: Bir olay akışı modeliyle, olaylar bir günlüğe yazılır. Tüketici servisler olay kanallarına abone olmak zorunda değildir. Bunun yerine, akışın herhangi bir bölümünden okuyabilir ve istedikleri zaman akışa katılabilirler.

1.2. İstek Tabanlı Mimari

Bu mimari ile REST veya RPC teknolojileri kullanılarak servisler arası mesaj iletişimi sağlanmaktadır. REST tabanlı mesajlaşma uygulanacak ise servis uç noktalarına JSON veya XML dosyalarına dönüştürülmüş mesajlar ile HTTP istekleri gönderilir. RPC ile REST içerisinde kullanılan JSON veya XML dosyaları yerine Protocol Buffers dosyaları ile ikili (binary) olarak gerçekleştirilir. İstek tabanlı mimari ile mikroservisler birbirlerine doğrudan bağlıdır ve mesajlar bir servisten diğerine eş zamanlı olarak gönderilir.

1.3. Mesaj Tabanlı Mimari

Mesaj tabanlı mimari yaklaşım ile servisler arasındaki iletişim eş zamanlı olmadan üretici – tüketici rolleri ile hazırlanmıştır. Üretici olan servis mesajı gönderdikten sonra gönderilen mesajın tüketilmiş olma bilgisini veya tüketici servis tarafından gönderilecek bir mesajı beklemek zorunda değildir. Mikroservis mimarisinde kullanılan mesaj tabanlı mimari tasarım genellikle aşağıdaki bileşenlere sahiptir.

Üretici: Mesajın üretildiği ve gönderildiği servistir.

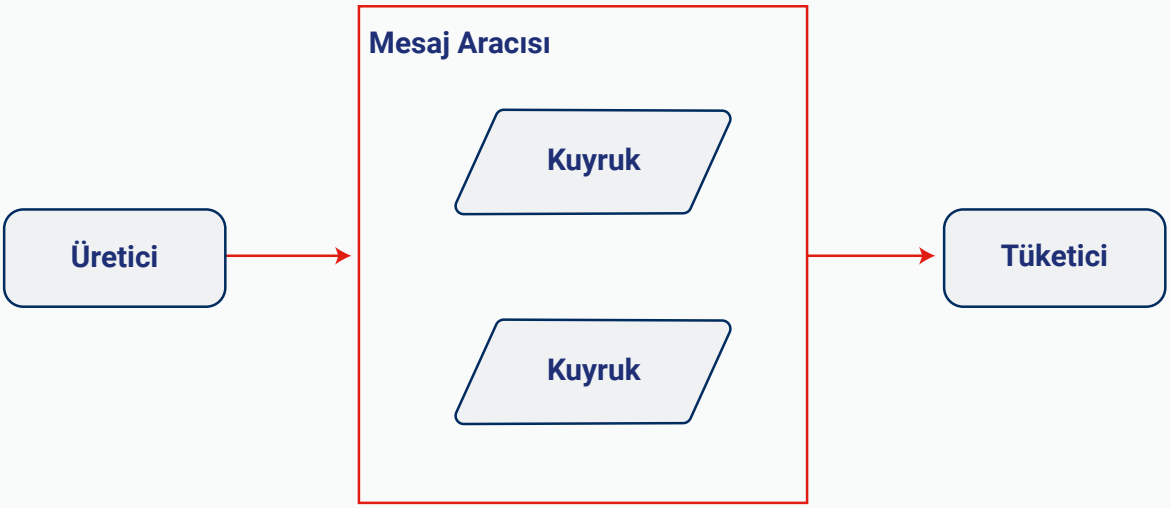
Tüketici: Belirli mesaj kanallarını dinleyerek bu mesajlara göre işlem yapan servistir.

Kuyruk (Queue): Üretici tarafından diğer servislere gönderilecek mesajların kaydedildiği ve biriktirildiği kanaldır.

Mesaj: Servisler arasında gönderilen belirli bir veri yapısına sahip içeriklerdir.

Değişim (Exchange): Mesaj aracıları (message broker) üretici mikroservis tarafından gönderilen mesajların ilk olarak karşılandığı yapılardır. Exchange içerisinde mesajlara ait iletim kuralları işletilerek her mesaj ilişkili olduğu kuyruğa gönderilir.

Mesaj Aracısı (Message Broker): Üretici mikroservislerin göndermiş olduğu mesajların tüketici servise iletilmesini sağlayan bileşendir. Mesaj aracıları içerisinde kuyruklar kullanılır. Apache Kafka, RabbitMQ ve ActiveMQ yaygın olarak kullanılmaktadır. Şekil 4'te mesaj tabanlı mimari yapısı gösterilmektedir.



Şekil 4. Mesaj Tabanlı Mimari

2. Dağıtım

Mikroservis tabanlı uygulamaları dağıtmanın en iyi yolu, süreçlere yalıtım ve temel donanım kaynaklarına özel erişim sağlayan ve sanal işletim sistemi ortamlarına sahip olan konteynerlerdir. Bu konteyner (container) çözümlerindeki en yaygın uygulamalardan biri Docker'dır.

Docker, uygulamaları geliştirmek, dağıtmak ve çalıştırmak için açık kaynak kodlu bir platformdur. Docker, uygulama yazılımlarının hızlı bir şekilde teslim edilebilmesi için uygulamaların alt yapıdan ayrılmasına olanak tanır. Docker ile sistem altyapısını, uygulamaların yönetildiği şekilde yönetilebilmektedir. Docker'ın kodu gönderme, test etme ve hızlı bir şekilde dağıtma yöntemlerinden yararlanarak, kodu geliştirme ile üretim ortamında çalıştırma arasındaki gecikme önemli ölçüde azaltılabilmektedir (Docker Overview, 2022).

Docker, bir uygulamayı konteyner (container) adı verilen, izole edilmiş bir ortamda paketleme ve çalıştırma olanağı sağlar. İzolasyon ve güvenlik özelliği, belirli bir ana sunucuda aynı anda birçok konteynerin çalıştırılmasına izin verir. Konteynerler uygulamayı çalıştırmak için gereken her şeyi içermektedir. Bu nedenle ana sunucuda şu anda yüklü olan uygulamaların kullanılması gerekmemektedir. Mikroservis uygulamaları ile çalışırken konteynerleri servisler arasında kolayca paylaşabilir ve paylaşılan servislerin aynı şekilde çalışan aynı konteynere sahip olduğunu garanti etmektedir. Docker konteynerleri, uygulama servisleri yaşam döngüsünü yönetmek için araçlar ve platform sağlamaktadır. Uygulamaların ve destekleyici bileşenlerin konteyner kullanarak geliştirilmesi yönetimi açısından büyük kolaylık sağlamaktadır. Uygulama dağıtımı yapılacağına karar verilip sürüm paketlerinin hazırlanması durumunda üretim ortamında bir konteyner veya düzenlenmiş bir servis olarak dağıtımı yapılmalıdır. Dağıtım ortamlarının farklılık içermesine rağmen Docker konteynerleri farklı ortamlar için de aynı özellikle çalışmaktadır.

Amazon Web Services (AWS) gibi alt yapı sağlayıcılarının sunmuş olduğu sanal makineler, mikroservis dağıtımları için de verimli bir şekilde çalışabilir. Kod dağıtımları, Açık Hizmet Ağ Geçidi Girişimi (OSGI) paketi kullanılarak da yapılabilmektedir.

Sonuç ve Öneriler

Mikroservis mimarisi ile büyük ve karmaşık uygulamalar verimli bir şekilde parçalı yapıya dönüştürülerek çözüm kolaylığı, yazılım geliştirme hızında artış, teknoloji seçiminde esneklik sağlanır. Her mimari yaklaşımda olduğu gibi, mikroservis mimarisinin de maliyeti bulunmaktadır. Bazen farklı diller, kütüphaneler, çatılar ve veri depolama teknolojileri kullanmak, bazı projeler için sistem yönetimi ve geliştirilmesi konusunda zorluklar yaratabilmektedir. Her tip uygulama mikroservis mimarisinin sunduğu özerklik ve bağımsızlık ile geliştirilmeye uygun olmayabilir. İş alanı ve kapsamı büyük uygulamalar için, hızlı ve otonom teslimat, doğru ölçeklendirme ihtiyacı bulunuyorsa veya sistem genelinin değil de belirli bir kod parçalarının sık sık güncellenip dağıtımının yapılması gerekiyorsa mikroservis mimarisi uygulanabilir en iyi seçeneklerden biridir.

Kaynakça

1. Richards, M. (2015) Microservices Vs. Service-oriented Architecture, O'Reilly Media
2. Soldani, Jacopo & Tamburri, Damian & Heuvel, Willem-Jan. (2018). The pains and gains of Microservices: A systematic grey literature review. journal of systems and software. 146. 10.1016/j.jss.2018.09.082.
3. N. Dragoni, I. Lanese, S.T. Larsen, M. Mazzara, R. Mustafin, L. Safina "Microservices: How To Make Your Application Scale". In: Petrenko A., Voronkov A. (eds.)
4. Thilina Ashen Gamage, Microservices Design Guide , https://medium.com/platform- engineer/microservices-design-guide-eca0b799a7e8 (Ağustos, 2022)
5. What is event-driven architecture? https://www.redhat.com/en/topics/integration/what-is-event-driven- architecture (Ağustos, 2022)
6. Apacke Kafka, https://kafka.apache.org (Ağustos, 2022)
7. Docker Overview, https://docs.docker.com/get-started/overview/ (Ağustos, 2022)

T.C. SANAYİ VE
TEKNOLOJİ BAKANLIĞI

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)