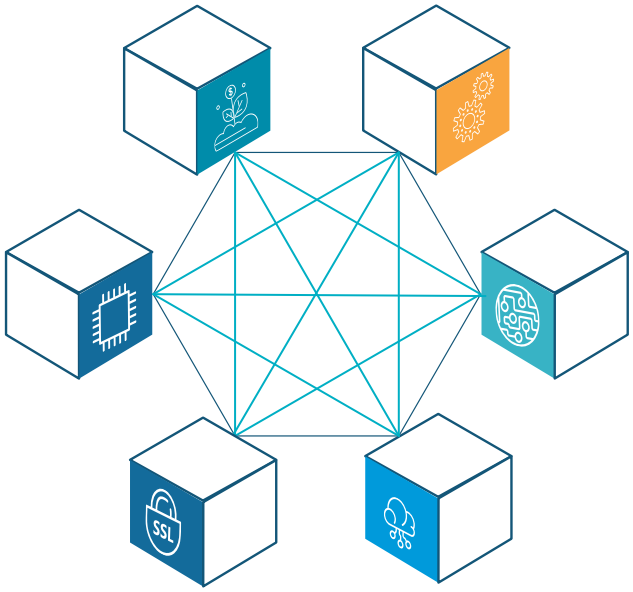
TÜBİTAK
BİLGEM
YTE

MİKROSERVİSLER



MİKROSERVİSLER GELİŞTİRME PLATFORMU

Mikroservis mimari kalıplarına uygun yazılım geliştirme ve işletme süreçlerini standartlaştırarak yazılım geliştirmeyi ve projelerin gerçekleşme sürelerini azaltmak amacıyla, Yazılım Teknolojileri Araştırma Enstitüsü tarafından açık kaynak teknolojiler üzerinde, kararlı bir yapıda hizmet sağlayan Mikroservis Tabanlı Yazılım Geliştirme Platformu geliştirilmiştir.

NEDEN MİKROSERVİSLER?

İnternetin yaygınlaşması ile birlikte, daha önce görmediğimiz bir hızda, teknolojik ve ekonomik bir değişim yaşamaktayız. Yazılımdan başka bir ürünü olmayan şirketler, 100 yıllık şirketlerden daha değerli hale gelmiş ve dünyanın en büyük şirkerleri arasındadır. Yazılımdan üretilen katma değer, her geçen gün daha yüksek seviyelere çıkmaktadır. Yazılımlar, günlük yaşamımızın bir parçası haline gelmiş, bireysel ve kurumsal olarak toplam verimliliğimiz artmış ve maliyetlerimiz de aynı şekilde düşmüştür.

Modern dünyada, her yerde ve her ortamda bir veya birden çok yazılım kullanılmaktadır. Cep telefonumuzdan, uzay araçlarına, uzak bir yerdeki sensörlerden, teknolojik silahlara, elektrik dağıtım sistemlerine kadar tüm hizmetler dijitalleşmiş, teknoloji hayatımızın bir parçası haline gelmiş, hatta hayati gereksinim haline dönüşmüştür. Yazılımlar, dünyayı hızla değişmeye ve ilerlemeye zorlamaktadır.

Firmalar ve kurumlar bu akışa ayak uydurmak için daha hızlı yenilikler yapmak zorundadır. Dijitalleşmenin temel gereksinimi olan yazılımın, daha hızlı, daha sık ve daha sağlam bir şekilde güncellenmesi gerektirmektedir. Burada mikroservisler bu ihtiyacı karşılamak için bir model olarak ortaya çıkmıştır.

> **"Mikroservisler, her uygulama işlevinin kendi başına bir hizmet olduğu ve bu hizmetlerin ayrık kapsayıcılarda dağıtıldığı ve bu kapsayıcıların API'ler aracılığıyla birbirleriyle konuştuğu mimari bir modeldir."**

> **"İş gereksinimlerini karşılamak için hızlı hareket etmek, hızlı hareket ederken de bir şeyleri bozmamak mikroservis mimarisinde prensip olmuştur."**

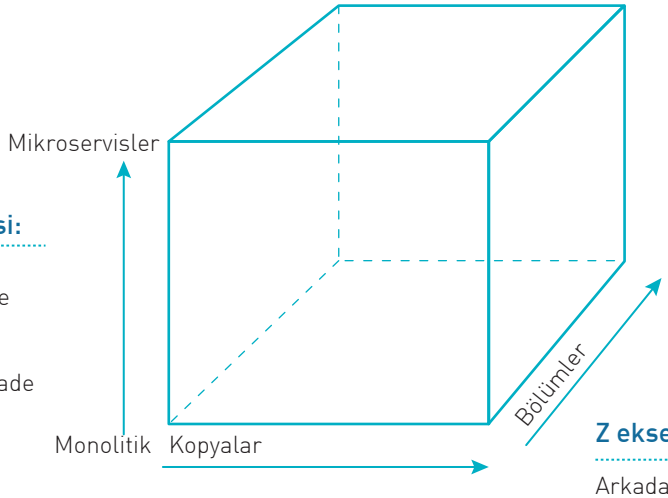
Monolitik mimarilerde, uygulamanın ölçeği büyüdükçe, geliştirme, test, kurulum ve ölçeklendirme süreçleri giderek zorlaşmakta ve belli zaman sonrasında idame edilemez hale dönüşmektedir. Büyük ölçekli uygulama geliştirmedeki belirtilen bu dar boğazlardan kurtulmanın yolu mikroservis mimarisiyle yazılım geliştirmektir.

Mikroservis mimarisini sağlam bir şekilde işletecek gerekli araç gereçler mimarinin ortaya çıkışından bu yana çok geliştirilmiş ve günümüzde büyük ve karmaşık bir uygulama geliştirilecekse mikroservis mimarisi bir zorunluluğa dönüşmüştür.

Mikroservisleri anlamak için yazılımda ölçeklendirmenin bilinmesi gerekir. Yazılımlarda ölçeklendirme, ölçek küpü yöntemiyle anlatılmaktadır.

Yazılımlarda Ölçeklenebilirlik

Yazılımlarda ölçeklendirme, ölçek küpü yöntemiyle anlatılmaktadır.



Y ekseni ölçeklendirmesi:

Uygulamanın işlevselliğine göre parçalara (farklı servislere) ayrıştırılmasını ifade eder. Bu eksen mikroservisler eksenidir.

X ekseni ölçeklendirmesi:

Birbirleri ile aynı olan servis sayısının arttırılması (klonlanarak yatayda ölçeklendirilmesi) ve bir yük dengeleyicisi üzerinden bu servislere erişim sunulması ile gerçekleştirilir.

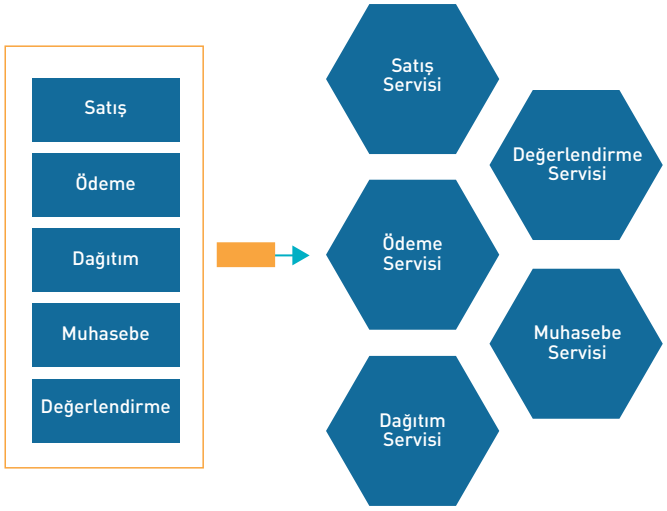
Z ekseni ölçeklendirmesi:

Arkadaki servisler birbirleri ile aynıdır fakat her bir servis, verinin ya da hizmetin belli bir altkümesinden sorumludur.

MİKROSERVİS MİMARİSİ NEDİR?

Mikroservisler bağımsız uygulamalardır, her bir servisin bir görevi vardır bu görevleri yerine getirmek için birbirleriyle gevşek bağlanmış olarak çalışmaktadırlar.

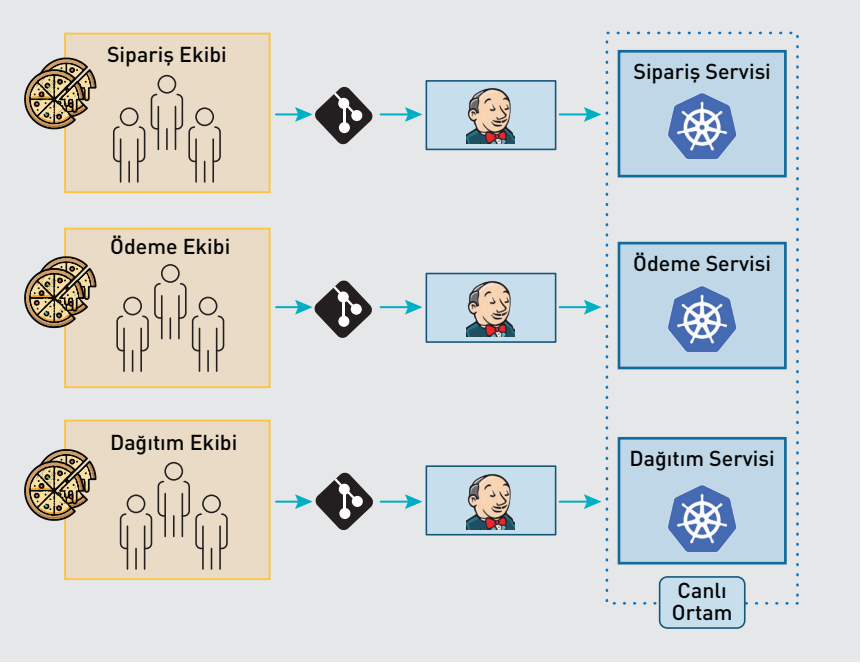
İş yeteneklerinizden mikroservisleri oluşturma



- Servisler birbirleriyle asenkron haberleşme yöntemiyle iletişim kurarlar. Tüm servislerin birbirleriyle iletişimi her birinin kendine ait APIleriyle sağlanacak ve monolit ile aynı işi yapacaklardır.
- Her bir servis otonom ve küçük takımlarla geliştirilir, büyük takımlara gerek yoktur ve tüm ekiplerin verimliliğini artırır.
- Her serviste yapılacak geliştirme hızlıca yapılır, daha az emek ve karmaşıklık içerir ve tüm uygulamada sadece o servisin etkilenmesiyle hızlıca canlıya çıkılır.
- Geliştirme ekiplerinin otonom hareket etmelerini sağlar. Bir ekip diğer servisin yapısını bilmek zorunda değildir.
- Her bir servisin verisi kendisine aittir. Ayrı tablolar, ayrı şemalar ya da ayrı veri tabanları şeklinde yapılandırılabilirler.

- Her servisin kodu diğerinden bağımsızdır, bağımsız kurulur ve yönetilir. Bağımsız olarak ölçeklenebilir.
- Her bir servis farklı bir dil ya da kütüphane kullanılarak üretilebilir. Yeni teknolojilerin kolayca denenmesini ve uygulanmasını sağlar.
- Servisler küçük olduğu için bakımı kolay yapılır. Yazılımda yeni özellikler getirme sıklığını artırır. Gereksinimlere çok hızlı tepki verilir.
- Yazılım geliştirme sürecinden testlerinin yapılması ve kurulumuna kadar istenilen ölçüde otomatikleştirmeye imkan sağlar.
- Çok iyi hata izolasyonu sağlar. Eğer bir serviste bir sorun olursa, tüm sistemi etkilemeyecek, sadece o serviste kalacaktır. Geri kalan sistem sorunsuz çalışmaya devam edecektir.

Mikroservislerde Yazılım Geliştirme Süreci



MİKROSERVİSLERİN ZOR YANLARI

Uygulamayı mikroservislere iyi ayrıştırmak gerekir. Tek seferde yapılamayabilir, yazılım yaşam döngüsü boyunca dönüşüme açık olmak gerekir.

Mikroservislerde uygulamanın tümü açısından veri dağıtık ve bazı yerlerde tekrarlı olarak tutulur. Veriyi anlamlı hale getirmek için veriyi birleştirmek ek işini gerektirir.

Uygulama testleri çok daha önemli olmaya ve daha karmaşık olmaya başlar. Otomatik testlere ve otomatik ölçeklendirme yeteneklerine sahip olmanız gerekir.

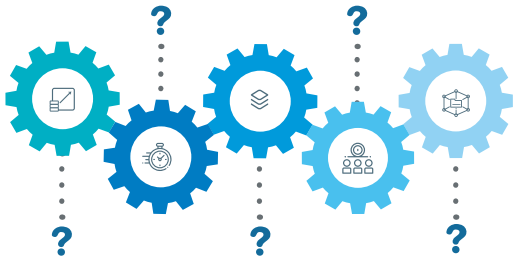
MİKROSERVİS MİMARİ KALIPLARI

Mikroserverislerin beklenen faydayı sağlayabilmesi için mikroservis dili diyebileceğimiz aşağıdaki kalıpların dikkate alınması gerekmektedir.

Bu dil size aşağıdaki konularda yardımcı olur.

- Hangi mimariyi kullanmanızın daha doğru olduğunu ölçmenize
- Her bir karar için seçeneklerinizi belirlemenize
- Her bir seçeneğin getir-götürlerini değerlendirmenize

MİKROSERVİSLER KULLANILIRKEN SORULMASI GEREKEN SORULAR



Uygulamayı Mikroservislere Nasıl Ayrıştıracaksınız?

Ayrıştırma Kalıpları

- **İş kabiliyetlerine göre ayrıştırma:** Mikroservislerimizi, var olan işi yapma şeklinize, kurumunuzun yapılanmasına göre ayrıştırmaktır.
- **Alt alanlara göre ayrıştırma:** İş kabiliyetlerine göre ayrıştırmaya benzer, fakat eğer iş kabiliyetleriniz verimli bir şekilde yapılandırılmamışsa ve verimsizlikler varsa bunu yazılıma da taşımış olacağınızdan, yazılımı, farklı ayrıştırma metedolojileriyle, olması gereken şekilde tasarlama imkanınız olacaktır.

Kodu Servislere Nasıl Kurulum Yapacaksınız?

Kurulum Kalıpları

- Veri depolamayan uygulamalarda konteyner mimarisi zorunlu ve en etkili yol olarak kabul edilmiştir ve kurumsal anlamda olgunlaşmıştır. Bu yüzden her bir konteynere bir servis ya da sadece bir fonksiyon olarak kurulabilen serverless kurulum kalıpları piyasayı domine etmektedir.
- Servisler farklı diller kullanılarak kurulabilmektedir.
- Her bir servis aynı ortamda çalışsa bile diğer servisleri etkilemeyecek şekilde sınırlandırılabilmektedir.
- Her servis izlenebilir şekilde kurulmaktadır.
- Servislerin birinde yaşanacak sorun otomatik izole edilir.
- Sadece ihtiyaç kadar kurulum yapılmakta, sistem kendi kendini otomatik olarak ölçekleyebildiğinden kaynak israfi olmamaktadır.

Servisler Arası Ortak Kullanımları Nasıl Ele Alacaksınız?

Ortak Kullanım Kalıpları

- Kimlik bilgileri ve veri tabanları ve mesaj aracıları gibi harici servislerin ağ konumları merkezi olarak çalışan ortamda saklanır.
- Uygulama log sistemi her zaman aktif çalışır. Uygulamanın ürettiği log kayıtları analiz edilebilir durumdadır.
- Uygulamanın sağlığını izleyebilmek için gerekli izleme ve sağlık kontrol hizmetleri yapılandırılır.
- Uygulamanın ne yaptığı ve nasıl performans gösterdiği hakkında sürekli ve geçmişe dönük bilgiler üreten sistemler kurulur.
- Her harici istekte, servisler arasında oluşan iletişimin performansı izlenir.

Servisler Arası Hangi İletişim Mekanizmalarını Kullanacaksınız?

İletişim Kalıpları

- **Uzaktan Prosedür Çağrısı:** Servisler arasında senkron bir iletişim ihtiyacı olduğunda kullanılır. İstek-Cevap (Request-Response) temelli yaklaşımla gerçekleştirilir. Yalnızca ve Yalnızca bir servis diğerinden sadece okunabilir (read-only) bir şey sorguluyorsa ve sadece sorgulayan tarafta veritabanı değişikliği olduğu durumlarda kullanılabilir.
- **Mesajlaşma:** Servisler arasında asenkron bir iletişim ihtiyacı olduğunda kullanılır. Yayınla-Abone ol temelli yaklaşımla gerçekleştirilir. Her iki serviste de veri tabanı değişikliği olacaksa bu iletişim biçimi kullanılır.

  Bu yöntemin gerçekleştiriminde bir mesaj kuyruğu kullanılır. İstemci bir servisteki veriyi değiştirir ve bununla ilgili o servis ortak mesajlaşma uygulamasındaki önceden tanımlı kuyruğa bir mesaj bırakır. Bu mesajı diğer servis okuyarak, kendinde yapılması gereken işlemleri yapar ve tüm uygulama tutarlı hale gelir.
- **Veri Kopyalama Yöntemi:** Bir servisin işini yapabilmesi için diğer bir servisten sürekli veri sorgulaması gerektiği ve sorgulanan verinin çok sık değişmediği durumda kullanılır. Değişiklikler asenkron olarak veriyi kopyalayan servislere bildirilir. Bu kategoride bir servis diğer servisten çok sık kullandığı veriyi sürekli sorgulamak yerine kendine kopyalar ve buradaki iletişim maliyetini ortadan kaldırır. Ayrıca servis ilgili işlemi yapabilmesi için diğer servise olan doğrudan bağımlılığını da ortadan kaldırdığı için başkasına bağımlı olmadan (autonomous) iş yapabilir.

Dış İstemciler Servislerle Nasıl İletişim Kuracaklar?

Harici API Kalıpları

- **API ağ geçidi:** Mikroservis mimarisinde istemcilerin arkadaki servislerle iletişim kurmasını sağlar. API Gateway veriyi birleştirmek, istenilen servisler için yönlendirme katmanı sunmak gibi görevleri yerine getirmektedir.
- **Ön uçlar için arka uç:** Bazen API ağ geçidi, farklı gereksinimleri olan istemcilere hizmet verdiğinden dolayı çok karmaşıklaşmakta ve yükü ağır olmaktadır. Her bir istemci tipi için genel API'yi özelleştirmek yerine, API'nin önüne gelen özel istekleri genel servise dönüştürecek daha minik bir dönüştürücü servis yerleştirmektir.

İstemci Servisin Ağdaki Yerini Nasıl Keşfedecek?

Servis Keşfi Kalıpları

Mikroservislerin sürekli kapatılıp açılabilmesi, ölçeklenebilmesi istenilen bir şeydir. Ama bu beraberinde ağda bu servislerin nasıl keşfedilebilir olacağı problemlerini doğurur.

Servis Kaydı Kalıpları

- Servis kopyası, ayağa kalktığı zaman, kendini ortak bir kayıt sistemine kaydeder.
- Ortamdaki üçüncü parti araçlar yardımıyla, ayağa kalkan uygulamanın ön tanımlı etiketlerine bakılır, görevi belirlenir ve merkezi olarak servis havuzuna eklenir.

Servis Keşfi Kalıpları

- **İstemci Tarafı Keşfi:** Bir servise bir istekte bulunurken, istemci bir hizmetin konumunu, tüm hizmet kopyalarının konumlarını bilen bir Hizmet Kaydını sorgulayarak elde eder ve isteği gönderir.
- **Sunucu Tarafı Keşfi:** Bir servise bir istekte bulunurken, istemci iyi bilinen bir konumda çalışan bir yönlendirici (örn. yük dengeleyici) aracılığıyla bir talepte bulunur. Bundan sonrasında yönlendirici, tüm hizmet kopyalarının konumlarını bilen bir Hizmet Kaydını sorgulayarak elde eder ve isteği gönderir.

Bir Ağ ya da Servis Sorununu Diğer Servislerden Nasıl Izole Edeceksiniz?

Güvenilir İletişim Kalıpları

Devre Kesici: Bazen bir istek birden çok servise gitmek zorunda olabilir. Bu istek zincirinde olan bir servisin geçici sorun yaşaması ya da önceden tanımlı kaynak sınırlarına gelmesi durumunda, toplam sistem kaynaklarını tüketmemek için önündeki servis, bağlantılı olduğu servise erişmeyi belli süre denemez ve öndeki servisi de bekletmez.

Servisler Arası Veri Tutarlılığını Nasıl Sağlayacaksınız Ve Sorgularınızı Nasıl Ölçeklendireceksiniz?

Veri Yönetimi Kalıpları

Her kurumsal uygulama en nihayetinde veriyi, bir veri tabanında saklamak zorundadır. Mikroservisler, gevşek bağımlı bir ilişki yapısına sahiptirler ve her bir mikroservisin kendine özel veri servisinin (veri tabanı) olması ve sadece kendinin erişebiliyor olması beklenir.

Mikroservis felsefesi veri servisini şu şekillerde tasarlayabileceğimizi söyler.

1. Her servise özel tablolar
2. Her servise özel şema
3. Her servise özel veri tabanı

1 ve 2 seçenekler yönetimsel olarak kolaylıklar sağlasa da, büyük mimarilerde her servise özel veri tabanı tercih edilmesinin aşağıdaki avantajları vardır:

- Tüm servisler gevşek bağlanmış olarak yapılandırıldığından bir veri tabanındaki bir değişiklik diğerlerini etkilemez.
- Her servis kendisine uygun veri tabanı yöntemini tercih edebilir. Arama için geliştirilmiş bir servis nosql doküman veri tabanı kullanabilirken, çok boyutlu ilişkiler gerektiren bir servis de grafik veri tabanı kullanabilir.

Servisler Arası Veri Tutarlılığını Nasıl Sağlayacaksınız Ve Sorgularınızı Nasıl Ölçeklendireceksiniz? devamı...

Mikroservislerde veri yönetimi zorlukları:

- Birden çok servisi ilgilendiren atomik işlemler (transactions) kolayca yapılamayacaktır. Dağıtık atomik işlemlerin kullanılmaması gerekir. En iyi çözüm **saga kalıbı**dır. Servisler verilerini güncelledikleri zaman durum/mesaj yayınlarlar ve diğer servisler bu mesajlara abone olarak tüketirler.
- Bir kaç mikroservisin verisini içeren sorgular yazmak zorlaşacaktır.
- Birden çok veri tabanı türü kullanılacaksa her birisi için ayrı ayrı uzmanlaşmak gerekecektir.

Çözümleri:

- **API Birleştirmesi:** Veri birleştirmesini (join) veri tabanı yerine uygulama yapar.
- **Veri Ortaklaştırma:** Birçok servisten gelen birleştirilmiş veri bir yada birden çok materyal gösterim içerisinde tutularak kullanılabilir.

Veri Tutarlılık Kalıpları

Her servise özel veri tabanı veri mimarisinde,

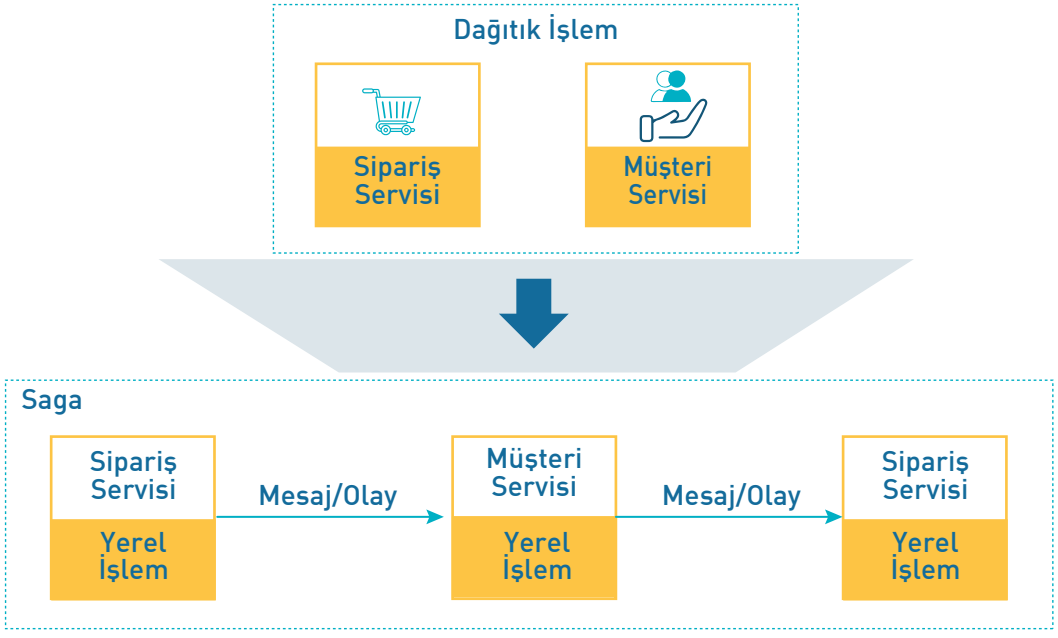
- Her servisin verisi diğerinden bağımsızdır.
- Uygulama genelinde veri tutarlılığı sağlanması gereken durumlar olabilmektedir.
- Veri tutarlılığını sağlayabilmek ve aynı zamanda veri tabanı veya serviste kilitlenmeye neden olmamak için **Saga etkileşim kalıbı** önerilmektedir.

Saga Kalıbı

> Her servis asgari kilitle kendi verisini günceller ve altyapıya bir mesaj yayınlar ya da başka bir yerel işlem tetikler. Eğer yerel atomik işlem, bir iş kuralını çiğnediği için başarısız olursa, Saga, önceki yerel atomik işlemlerin yaptığı değişiklikleri geri alacak telafi atomik işlemler yayınlaroservisler, her uygulama işlevinin kendi başına bir hizmet olduğu ve bu hizmetlerin ayrık kapsayıcılarda dağıtıldığı ve bu kapsayıcıların API'ler aracılığıyla birbirleriyle konuştuğu mimari bir modeldir.

Sagaların koordinasyonu için 2 yol vardır.

- **Koreografi:** Her yerel işlem, diğer hizmetlerdeki işlemleri tetikleyen olaylar yayınlar.
- **Orkestrasyon:** Bir merkezi yönetici (nesne) tüm taraflara hangi işlemleri gerçekleştireceğini söyler.



Testleri Nasıl Daha Kolay Yapılandıracaksınız?

Test Kalıpları

Servis Bileşeni Testi: Uygulamada mikroservisler genelde uygulama bütününde veri akışını tamamlamak için birbirlerini çağırırlar. Servisler arasındaki bu etkileşimlerde, servisin doğru şekilde davrandığı test edilmelidir. Birçok servisi başlatarak uçtan uça test işlemi yavaş ve maliyetli olacağı için, bu durum, test amaçlı "gerçek nesneleri sahte nesnelerle değiştirme" kullanılarak izolasyon içerisinde test edilir.

Tüketici Odaklı Sözleşme Testi: İki mikroservisin iletişim kurduğu noktalardaki değişiklikleri yönetebilmek için yazılan bu test tipinde tüketici sınıfı, sağlayıcı sınıftan beklentilerini içeren sözleşme yazar ve sanki sağlayıcı sınıf tarafından bu beklentiler sağlanmış gibi tasarımını ve kodlamasını yapar. Yazılan bu sözleşme, ortak bir depoda yayımlanır ve sağlayıcı tarafından çekilerek istenilenleri kendi tarafında çalıştırır.

Tüketici Tarafı Sözleşme Testi: Sağlayıcı tarafından açılan servisin, tüketici tarafındaki istemcileri ile iletişim kurabildiğini doğrulamak için yazılan testlerdir.

Uygulamanın Davranışını Nasıl Anlayacaksınız ve Sorunları Nasıl Çözeceksiniz?

Gözlemlenebilirlik Kalıpları

Tüm uygulamanın, baştan başa, ilişkili bir şekilde gözlenebilir olarak tasarlanmasıdır.

Günlük Kayıtları Merkezileştirme: Birbirleriyle bağlantılı/ortak çalışan bir çok mikroservisin ürettiği günlük kayıtlarının, önceden belirlenen ve uygulama genelinde aynı standartlarda üretildikten sonra, merkezileştiren bir toplama servisinde birleştirmek gerekir.

Uygulama Metrikleri: Tekil operasyonlar hakkında istatistik toplayan bir servisin olması gerekir.

Dağıtık İzleme: İstekler, dağıtık sistemlerde yayılırken, onları gözlemleme yöntemlerini ifade eder. Bir dizi hizmetin tekil istekleri ele almak için nasıl koordine edildiğini ortaya çıkaran bir teşhis tekniğidir.

PLATFORMDAKİ
AÇIK KAYNAK YAZILIM TEKNOLOJİLERİ

YTE tarafından Javascript React framework bileşenleri, kurumsal gereksinimlere ve standartlara göre bir araya getirilmiş ve mikroservis geliştirici yazılımcının kolay bir şekilde önyüz geliştirmesi sağlanmıştır.

Mikroservisler arası iletişimde standart haline gelen Apache Kafka için iyi pratikler etrafında kurma, ölçeklendirme ve diğer etkileşimler standartlaştırılmış ve kolaylaştırılmıştır.

Platform, mikroservislerin olmazsa olmazı olan kubernetes üzerinde çalışacak şekilde tasarlanmış, kubernetes'in mikroservis kullanımı kolaylaştıracak tüm özellikleri kullanılmıştır.

CAS Merkezi kimlik doğrulama ve yetkilendirme servisi iyi pratikler şeklinde redis dağıtık ön belleği kullanılarak hızlı kurulabilir hale getirilmiş, mikroservis kütüphaneleri cas ile kolayca entegre çalışabilecek şekilde yapılandırılmıştır.

Asenkron ve dağıtık mesajlar ve görevlerin yönetimi mikroservislerde kritik olduğundan geliştirilen platformda otomatik lider seçimi de entegre edilmiştir.

Jenkins sürekli entegrasyon ve kurulum aracıyla, mikroservislerin hızlı ve sağlam kurulumları otomatikleştirilmiş ve sağlıklı çalışması için altyapıları oluşturulmuştur.

Merkezi loglama yönetimi ile uygulama logları önceden yapılandırılmış olarak elasticstacke gidecek şekilde yapılandırmak için geliştirmeler yapılmış uygulama logları merkezi log yönetimi sisteminden izlenip analiz edilebilir hale getirilmiştir.

Redis ayrıca uygulamaların ortak durumlarını saklamak ve tüm uygulama kurulumları tarafından erişilebilir olmasını sağlamak için platforma entegre edilmiştir.

PLATFORMDAKİ
AÇIK KAYNAK YAZILIM TEKNOLOJİLERİ

Java Spring frameworkteki kütüphaneler, kurumsal yazılım gereksinimleri göz önünde bulundurularak bir araya getirilmiş, tüm yazılımcıların kolaylıkla adapte olabilecekleri kapsayıcılar geliştirilmiştir. Senkron ve asenkron iletişim kütüphaneleri eklenmiş, özelleştirilmiş hata işleme kütüphaneleri geliştirilmiştir. Ayrıca Spring Data ve hibernate kullanılarak dağıtık veri akışı ve transaction yönetimi kütüphaneleri geliştirilmiştir. Mikroservisler için olmazsa olmaz test yöntemleri platforma entegre edilmiş ve hızlıca kullanılabilir hale getirişmiştir.

> Eğer projeniz, büyük ölçekli ise veya büyük ölçekli olma olasılığı yüksekse, yukarı sayılan avantajlarından dolayı mikroservis mimarisinin kullanılması gerekli hale gelir. Monolit uygulamaların mikroservislere yavaş yavaş dönüşümü de mümkündür.

TÜBİTAK - BİLGEM
Yazılım Teknolojileri Araştırma Enstitüsü (YTE)

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA
+90 312 284 9222 +90 312 286 5222

www.yte.bilgem.tubitak.gov.tr