Monolitik Mimari Nedir? SOA Nedir? Mikroservis Mimarisi nedir?

Mikroservisler mimarisi, son birkaç on yıl içinde adından sıkça söz ettiren ve genellikle dev şirketlerin iç yapılarında tercih ettiği bir strateji olarak öne çıkıyor. Bu, bir hikaye aracılığıyla daha detaylı bir şekilde keşfedildiğinde, bir dünya turu gibi olabilir. Bir zamanlar, bir kasaba vardı. Bu kasaba, büyük şirketlerin iç dünyalarını temsil ediyordu. Kasaba, kendi içinde farklı bölümler ve yapılar barındırıyordu. Ancak, zamanla bu kasaba giderek karmaşıklaştı ve büyüdü. Bu karmaşıklık, insanların birbirleriyle iletişim kurmasını ve birlikte çalışmasını zorlaştırdı. Günün birinde kasabanın ileri görüşlü liderleri, bu karmaşıklığı ele almanın ve daha etkili bir düzen kurmanın yollarını aramaya başladılar. İnsanlar arasındaki etkileşimi kolaylaştırmak ve daha iyi bir düzen sağlamak için farklı mimarilere göz attılar. Ancak, bu araştırmaların sonunda, kasabanın geleceği için en uygun olanın “Mikroservisler Mimarisi” olduğuna karar verdiler. Mikroservisler, kasabayı küçük parçalara ayırmaya ve her bir parçayı bağımsız bir birey olarak düşünmeye dayanıyordu. Bu, her parçanın kendi görevine odaklanmasını ve diğer parçalarla daha az bağımlı olmasını sağlıyordu. Böylece, kasabanın iç yapısı daha anlaşılır ve yönetilebilir hale geldi. Bu yeni yaklaşım, kasabanın sakinleri arasında büyük bir rahatlama ve işbirliği getirdi. Her bir mikroservis, kendi görevini yerine getirirken, diğerleriyle iletişim kurabiliyor ve bir bütün olarak kasabanın daha etkili çalışmasına katkıda bulunuyordu. Mikroservislerin kasabada nasıl bir dönüşüm yarattığı, diğer kasabaların da bu stratejiyi benimsemesine öncülük etti.

Ama tabii.. Siz bu stratejeyi anlamadanöncelikle diğer mimarilere göz atmak faydalı olacaktır. Haydi birlikte bakalım !

1- Monolithic Architecture

2- SOA Architecture

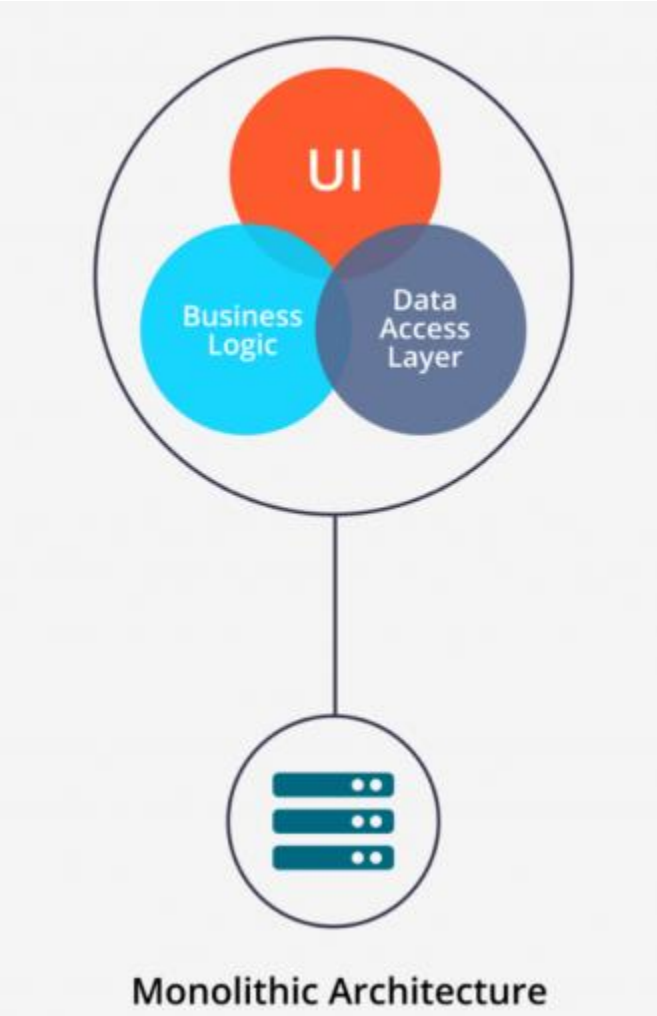
3- Microservices Architecture

Monolithic Architecture

İsimleme olarak bilmeseniz de aslında Monolitik mimariyi günlük projelerinizde kullanıyor olabilirsiniz. Müşterilerinizin olduğunu düşünün(sunucuya istekleri atan) ve bir tane de sunucunuz olsunuz. Bu sunucunuz da bir veri tabanına bağlanıyor. Örneğin; Twitter hem web sitesinde hem de mobilde çalışma düzeni olan bir uygulamanız olduğunu hayal edin. Siz bir tane gönderiyi görmek istediğinizde sunucuya bunun isteğini gönderirsiniz. Aynı şekilde resim veya video görüntülemek için de bu istekleri sunucuya gönderirsiniz. Bu sunucu üzerinde çalışan proje, bu veriyi getirebilmek için bunu veri tabanına sorması gerekir ve bu yüzden veri tabanına gider. Veri tabanından gelen datayla birlikte ekranda gönderileri,iletileri vb göstermiş olur. Bu yapı da monolitik mimari olarak geçmektedir. Bir yerlerde anımsadınız değil mi :) ? Özetle Monolithic; bir proje üzerinden tüm aksiyonları

ürettiğiniz bir proje mimarisi yapısıdır. (Tabii ki Twitter Monolitik bir mimariye sahiptir demiyoruz. Sadece örnek verdim. Aman linçlemeyin beni. :))

Monolitik mimari de sizin sunucunuzda 3 adet katman vardır.



- **Presentation Layer**

Presentation Layer katmanı adından da anlayabileceğimiz gibi sunum katmanıdır. Kullanıcının gördüğü kodları oluşturduğunuz durumdur. Yani kullanıcı arayüzüdür.

- **Business Layer**

Mantıksal işlemlerin yapıldığı yerdir. Açıkçası kodları yazdığımız bölümdür. Müşteriden bir istek geldiğinde Business Layer içerisinde işlerin yapılma işlemi gerçekleşir. Ancak Business kısmında veri tabanına bağlanmamızı gerektiren işlemler olursa bu kısımda Data Access Layer katmanına telefonla bağlanma joker hakkımızı kullanabiliriz.:) İzniniz var mıdır Kenan Bey?

- **Data Access Layer**

Veri tabanına bağlanma işlemleri için ise bu katman kullanılır. Modellerle beraber bilgiler veri tabanına aktarılır ve buradan alınan bilgiler Business Katmanı'na aktarılır. Oradan da Presentation Katmanı'na aktarılarak kullanıcıların görmeleri sağlanır.

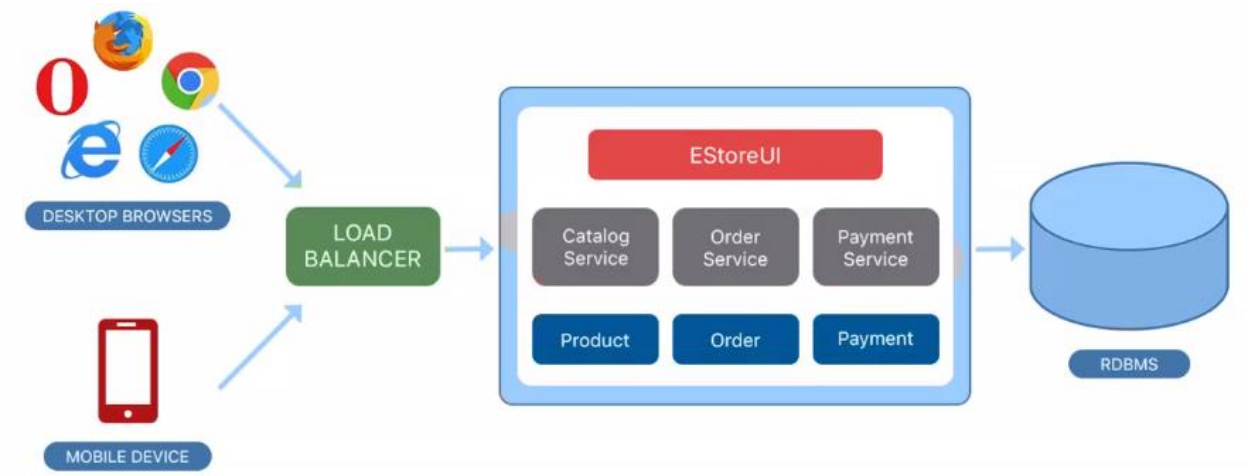
Bu kullanımda bir problem yok ama Twitter gibi bir yapıyı Monolith mimari de çalışırsan neler olur ?

Gönderi getirmek için ana sayfaya istek atmamız gerekebiliyor. Ancak bu endpoint üzerine sadece biz istek atmıyoruz. Birçok kişi anasayfaya istek atıyor ve haliyle gelen trafik artmaya başlıyor. Sizce Twitter'ın ana sayfası mı yoksa sizin hesabınızın kullanıcı adınızı güncellediğiniz kısım mı daha yoğundur? Tabii ki de ana sayfa dediğinizi duyar gibiyim. İşte bu tarz yoğunluk durumlarındabu yapıyı ölçeklendirmek istersek sunucuyu duplicate etmeniz gerekebilir.

Bırakalım Twitter'ı bir kenara, Twitter zaten mutsuz adam işi...(zaten artık Twitter bile değil, X :)). E-Ticaret sitesi örneği üzerinden ilerleyelim!

- Bir sunucunuz olduğunu düşünelim tekrardan. Bu sunucu içerisinde yukarıdaki açıkladığımız gibi katmanlarınız olsun. Monolith bir uygulamanız application server içerisinde çalışsın. (Java,PHP veya C# olabilir.)

- Bu e-ticaret sitesi örneğinde birçok servisiniz olduğunu da düşünün. (ProductService,AccountingService,PaymentService vb.). Bu servislerin içinden yoğun olarak trafik alan servis, ProductService olsun. Yoğunluk oldukça kitlenmeye başlayan bu serviste veri tabanı sorgularına yetişemez olursunuz ve haliyle kaynaklar yetersiz gelmeye başlar. Bu durumda ölçeklendirme yapmak gerekebilir. O nedenle bu kısımda sunucunun ekran görüntüsü(snapshot) alıp kopyalama işlemi gerçekleştirmek gerekir. Projeniz 8080 'de çalışıyorsa 8081'de çalışan bir sunucu daha oluşturmak gerekebilir. Peki bu durumda ölçekleme gerçekleştikten sonra buradaki yük dağılımı nasıl gerçekleşir ? İşte bu işlemi gerçekleştirmek için imdadımıza LoadBalancer yetişir. LoadBalancer müşteri üzerinden gelen istekleri duruma göre sunucular arasında paylaşmayı hedefler. Ancak buradaki örnekte dikkat ettiyseniz yoğunluğum ProductService'de olmasına rağmen tüm sunucuyu duplicate etmek gerekti. Bu da monolithic mimarideki bir dezavantajdır. :-(



Monolotik Mimari Avantajları Nelerdir ?

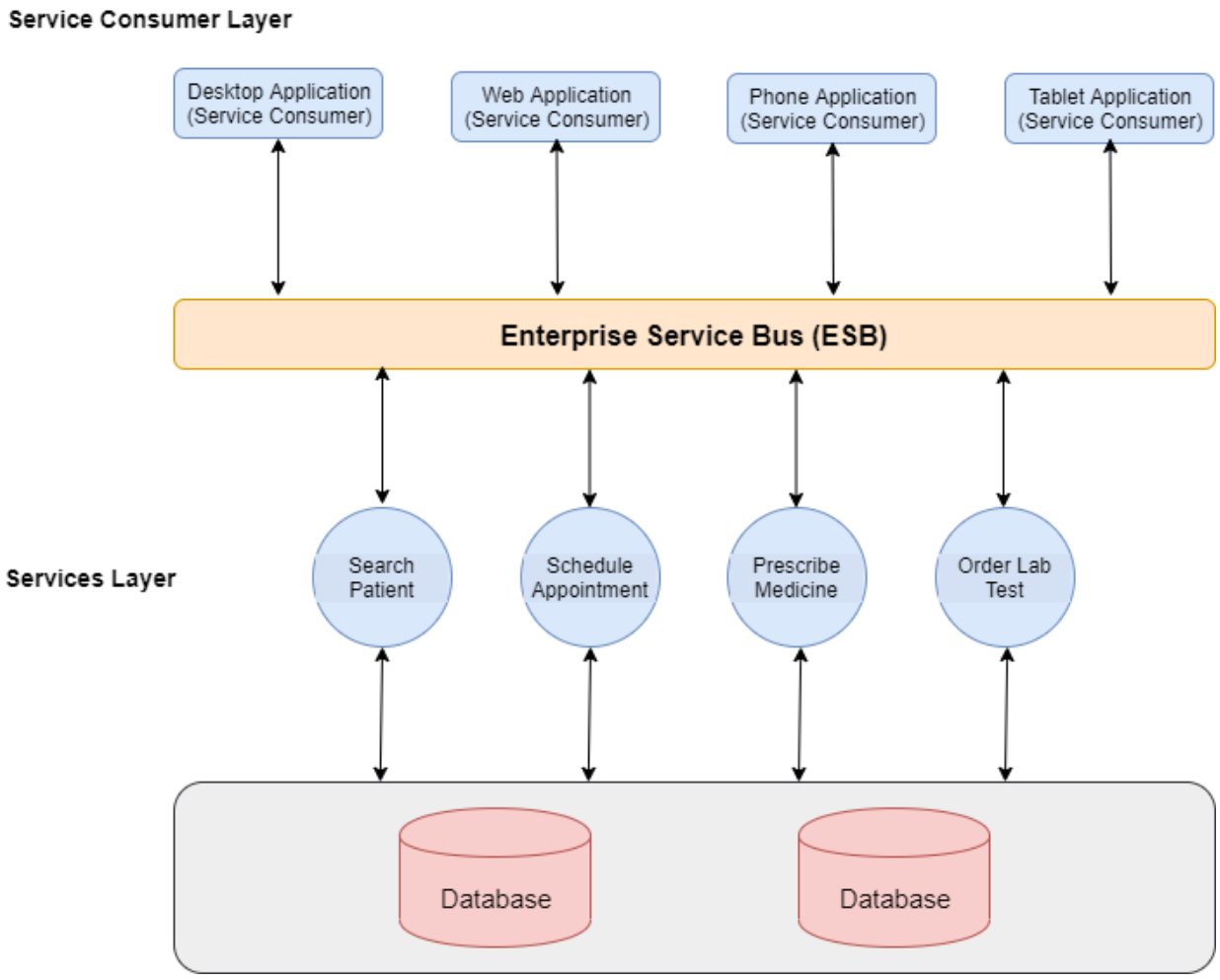
- Geliştirmesi basittir.
- Modülleri tek bir projede geliştirildiği için test edilebilirliği kolaydır.
- Tek bir projede deployment işlemi gerçekleşeceği için deployment işlemleri de oldukça kolaydır.
- Ölçeklendirme oldukça kolaydır. Yatay olarak mimarinin kopyasını Load Balancer arkasında çalıştırabilirsiniz. Yatay ölçeklendirmeyi basitçe özetleyecek olursak bir sunucunun kopyasını alıp yanına koymaktır. Dikey ise varolan sunucunun kaynaklarını arttırmaktır.(Diskini arttırmak vb.).

Monolotik Mimari Dezavantajları Nelerdir?

- Bakım proje büyüdükçe zorlaşır. Yüz tane modülünüz projeye kod yazmak yerine iki tane modüle kod yazmak arasında büyük bir fark vardır. Proje büyüdükçe batsın bu dünya, bitsin bu rüya moduna girebilirsiniz. :(
- Uygulamanın boyutu başlama süresini yavaşlatır. Monolitik bir uygulama kapandığında veya çalışmadığında tekrardan ayağa kaldırmak için belirli süreler beklemek gerekir. (Eğer proje çok büyürse.). Kimselerin durup da ince ve özel şeyleri dahi anlamaya vaktinin de sabrının da olmadığı günümüz dünyasında bu, ekstra bir dezavantajdır.
- Uygulama güncelleneceği zaman tüm uygulamayı tekrardan deploy etmek gerekir. Yani User üzerinde geliştirme yaparsanız tek User üzerinde deploy etmek mümkün değildir. Tüm projeyi canlıya çıkmak gerekir. Uygulama büyüdüğü zaman bu biraz külfet haline gelebilir.
- Monolitik uygulamalar ölçeklendirileceği zaman sorun çıkartabilir. Uygulamanızın kesinlikle bazı sistem gereksinimleri olabilir. Ölçeklendirme yapabileceğiniz sunucuya bu gereksinimleri vermezseniz yaptığınız klon istediğiniz gibi çalışmayabilir.
- Ölçeklendirme tüm proje genelinde yapılır. Bir serviste yoğunluk olduğunda tüm projeyi duplicate etmek gerekebilir. Aslında yoğun olan servisi kopyalamam yeterli olacaktı ama uygulama monolitik olduğu için maalesef bu mümkün değil :-(
- Güvenilirlik,sağlamlık ve dayanıklık kısmında sıkıntı çıkabilir. Büyük bir proje kapsamında takım arkadaşı alacağınızı düşünün. Ancak uygulamada yeterli dokümantasyon yok. O nedenle uzun süreler takım arkadaşınızın takıma uyum sağlamasını bekleyeceksiniz. Aslında bu bir nevi zaman ve nakit kaybı olabilir. Bu şekildeki handikapları içeren bir mimariyi kullanmalı mıyız diye soracak olursanız, şöyle yanıtlayabilirim. Monolitik mimari çok kötü bir mimari değil. Çünkü sektörün devleri de bir zamanlar monolitik mimariyle çalışıyordu. (Amazon, Netflix, Twitter) . Sonrasında başka başka mimarilere geçiş yaptılar. Proje büyüdükçe siz artık monolitik mimariden çıkabilirsiniz. Başlangıçta monolitik mimariyle başlayan projeler sonradan mikroservis mimarilerine evrilebilir. Bu yüzden “Monolitik kötüdür,kullanmayınız. Mikroservislere geçtim, gençleştim resmen bu kadar mı farkeder?” demem asla doğru olmaz. Her sistemin kendine göre gereksinimleri ve ihtiyaçları var. Bence bu noktada gereksinimlerimizi, ihtiyaçlarımızı belirlemek çok değerli diye düşünüyorum. Bunları belirledikten sonra kendimiz için en uygun mimariyi bulabiliriz. :)

SOA Architecture (Service Roiented Architecture)

Servislerin ayrı ayrı tasarlanıp bir yapı oluşturmasını sağlar. Monolitik mimaride bir proje içerisinde projenin içerisinde servislerimiz bulunuyordu. Bunların her biri projenin bir controller dosyası olarak geçiyordu. Bunları bunun içerisinden monolitik mimaride çıkarabilmek mümkün değildir. O yüzden ilk hedeflediği nokta “Servisleri ayırmak!” diyebiliriz.



Peki bunların kontrollerini nasıl yapacağım ?

- Özet olarak servislerin ayrı ayrı tasarlanıp bir yapı oluşturmasını sağlar.
- Yapılar birbirinden bağımsız olarak çalışabilirler. (Loose Coupling.) Loose Coupling düşük bağ anlamına gelmektedir. Birbirleriyle iletişimde olan servisler de olabilir, iletişimde olmayan servisler de olabilir.
- Birden çok sistemin yer aldığı yapılarda kullanılır.
- Kendi içerisinde birçok bileşeni vardır. (Policies, Contracts, Services ve daha fazlası.). Bu bileşenler oldukça yer kaplayabilir.
- Dağıtık yazılım sistemlerinin kalitelerini arttırmayı hedefler. (Tekrar Kullanılabirlik, Uyumluluk, Bakım Yeteneği) . Servisleri bir yapı oluşturmak için kullanılan bir mimaridir. Bu servisler kendi aralarında iletişim halinde olabilirler. Client üzerinden bir servise erişmek isterseniz direkt erişemezsiniz. Bunun için Enterprise Bus Service yapısı kullanılır. Bu yapıda müşteriden gelen isteğe göre hangi servise gitmek

isteniyorsa onu ayarlar. Kendi içerisinde ve dış dünyayla kurdukları iletişim için web servislerden yararlanılmaktadır. Bu web servisler SOAP, WDS vb. şeklinde örnek verilebilir.

SOA Avantajları nelerdir?

- Servisler tekrar tekrar kullanılabilir.(Reusable)
- Servislerin bakım ve onarım süreçler kolaydır. Çünkü koca projeyi değiştirmiyorsunuz, o yüzden monolotiğe göre daha kolaydır.
- Güvenilirlik ve dayanıklık açısından iyidir. Servisler birbirinden farklı farklı projelerle ayrılmış oldukları için böyle bir sorunla karşılaşmazsınız.
- Up time oranları yüksektir.
- Yatay ve dikey ölçeklendirme yapabilirsiniz.(Servisler ayrı olduğu için servis bazlı kopyalama yapabilirsiniz.)
- Platform bağımsızdır. (Bu mimarinin içerisindeki kullanmış olduğunuz servislerin dillerinin neyle yazılmış olduğu bağımsızdır.)
- Üretkenlik artacaktır. (Yetkinliğiniz C# ise eğer işe başlayan kişi Java biliyorsa, servisi o dille yazıp geliştirebilir. Herhangi bir sıkıntı olmaz.)

SOA Dezavantajları nelerdir?

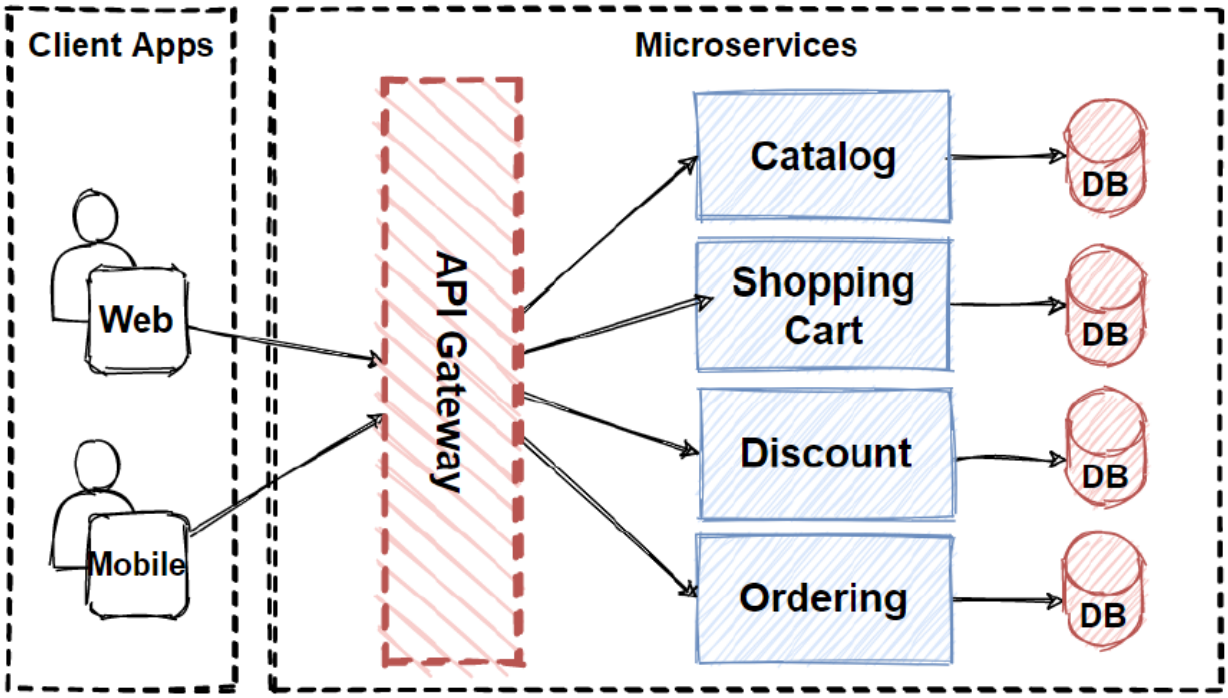
- Overload(Servisler kendi aralarında konuşurken SOAP gibi yapıları kullanırlar ve Enterprise kısmında yapılan validasyonlarda bu oldukça bir yük oluşturabilir.)
- Yüksek Maliyet
- Yüksek Bant Genişliği (Oldukça büyük bir bant genişliğine ihtiyaç olabilir, bu da sunucuyu beraberinde getirir.)

Microservices Architecture

Mikroservisler mimarisi aslında bir mimari değildir. SOA kısmında servislerin birbirinden ayrılmasını hedefleyen bir mimariden bahsetmiştik. Mikroservisin de amacı farklı değil. Mikroservislerin de amacı büyük servisleri **küçük serviscikler** halinde gerçekleşmektir. O nedenle aslında **SOA'nın** bir **yorumudur**.

- Her bir servis kendine ait bir dünyada çalışır. Soa'da bir sunucu üzerinde servisleri ayırıyorduk. Ancak mikroservislerde her bir servis kendi sunucusunda çalışacak. (Server Stack)

- Kendine ait veri tabanları vardır. (Ee yok artık, biz de abarttık ama sosunu abarttık :))
- Sadece bir küçük işi çok iyi yapması gerekir, o işi yapabilmesi yeterlidir. Diğer işlere karışmaz. (Aa sen biraz I Shaped misin acaba ? :))
- Her bir farklı sunucu olursa bu yapı nasıl iletişime geçilecek ? Tam bu kısımda **Api Gateway** burdayım be burdayım,bur-da-yım diyor ve **Api Gateway** sayesinde dış ve iç dünyaya açılma gerçekleşiyor. Gateway'i bir geçit bir kapısı olarak görebilirsiniz. Bu işlemi kod üzerinde de gerçekleştirebileceğiniz gibi **Reverse Proxy** ile de yapabilirsiniz.
- Herhangi bir teknoloji ve dile ait bir kısıtlama olunamaz. (İstersen A sunucusunda Ruby ve Mongo, diğerinde Dotnet ve Postgre kullanabilirsin.)
- Staless yapılardır. (Mikroservisin kendi sunucusu, kendi veri tabanı vardır. Sunucu üzerinde bağımlı olduğu için de sunucuya yazdığı bir dosya olmamalıdır.)
- Kolay ölçeklendirilebilir. Bu ölçeklendirme tipinde dikeye ihtiyacınız yok çünkü bir servisi gidip bir sunucu içerisind bir proje içerisine koyduğunuzda minimal anlamda bir küçük sunucu işinizi görebilir.
- Şematik olarak baktığınızda **Api Gateaway**'in yaptığı şey ona gelen istekleri alır ve uygun sunuculara gönderir. Gönderdiğiniz bir Product bilgisi ise bu bilgi JSON bilgisi ile gönderilir. Api Gateaway de alınan bu JSON bilgisini Client'a gönderir.
- **Api Gateaway** ve mikroservisler arasındaki iletişim JSON ile gerçekleşmektedir. Ancak bunu siz belirleyebilirsiniz. Daha kolay ve hızlı gerçekleştirebilirsiniz. Bu yüzden JSON döndüren servislerin hangi dilde yazıldığı ve hangi veri tabanını kullandığı hiç önemli değildir. Bu sayede hem servisleri ayırabilmiş olduk hem de farklı teknolojiler kullanabileceğini öğrenmiş olduk.



Mikroservislerin Avantajları

- Çok dilli mimari
- Kolay ölçeklendirme
- Daha iyi bir takım yönetimi
- Yenilik yapmak daha kolay (Farklı teknolojiler arası geçiş daha kolayy :) Yani yani çalışmak istemediğiniz teknolojiyle çalışırken **"Hadi yüreğim ha gayret, hele sıkı dur hele sabret. Başını eğme dik tut. "** modunda çalışmanıza gerek yok :)
- Mikroservislerin kendine ait veri tabanları vardır.
- Daha odaklı yapı ve ölçeklendirmesi bulunur. Ölçeklendirmede monolitikte olduğu gibi sunucuda ölçeklendirme yapıyorsunuz. Komple projeyi ölçeklendirmiş oluyorsunuz. Ama mikroservis projelerinde sadece ölçeklendirmek istediğim ve gerekli olan alanı ölçeklendirebiliyorum. Örneğin, Twitter'ın ana sayfası ayrı bir mikroservis, profil düzenleme kısımları ayrı bir mikroservis olsun. Ben anasayfada daha çok trafik aldığım için haliyle sadece ana sayfa alanını ölçeklendirmek isteyebilirim. Mikroservisler bunu rahatlıkla yapmanızı sağlarlar.
- Implemente edilirken diğer servisler etkilenmezler.

Mikroservislerin Dezavantajları

- Implementasyon kolay değildir. (Network Calls, Service Discovery)
- Debug kolay değildir. (Nerde hata olduğunu görebilmek için monitoring sistemler geliştirmeniz gerekebilir.)
- Fault Management(Hata yönetimi) kolay değildir. (Mikroservisin büyümesiyle beraber.)

Google,Facebook gibi şirketlerin altyapısında mikroservislerin kullanıldığını unutmayalım.

Medium Makalesi

- https://ruveydakardelcetin.medium.com/mimarilerin-gücü-adına-monolitik-soa-mikroservis-a6ee85acaee7

Kaynakça

- Building Microservices, 2nd Edition
- https://www.youtube.com/watch?v=IGUJKGskaOE
- https://www.youtube.com/watch?v=oMlzcfMPEJc