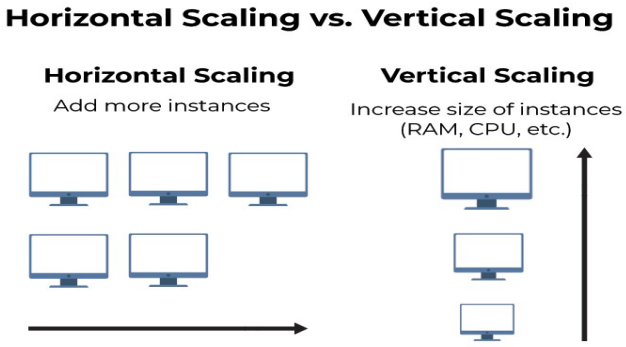
Monolith ve Microservice Mimarisi

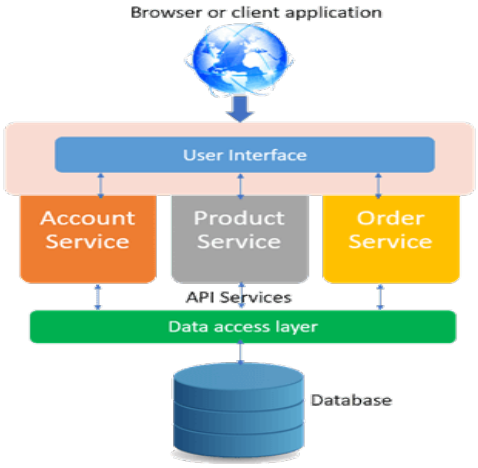
Bu yazıda Monolith ve Microservice mimarisinin genel özelikleri beraber hangi durumlarda hangi mimarinin neden kullanılması gerektiğini örnek bir senaryo üzerinden karşılaştırma yapılarak anlatılacaktır. Öncelikle Microservis ve Monolith mimarinin kısaca tanımlarından bahsedebiliriz.
Microservis mimari orta ve büyük ölçekli servisleri küçük servislere ayırmak olarak ifade edilebilir. Başlıca kullanım amacı servislerin birbirinden bağımsız olmasını sağlamaktır (loosely coupling).
Peki biz bir uygulamayı geliştirirken neden servisleri bağımsız yapmak isteriz ki? Bunun cevabını bulmak için Monolith mimariden de bahsetmemiz gerekecek.
Monolith mimaride her servisin tek bir uygulama paketi içeresinde bulunduğu ve birbirlerine yüksek bağımlıkları olan servisler olduğunu söyleyebiliriz.
Uygulamamızda kullandığımız servislerin birbirine bağımlılığı ne kadar fazla ise proje büyüdükçe eklenen her yeni bir özellik ile servislerde değişiklik yapmak giderek zor bir hal alır. Çünkü bir servis üzerinde değişiklik yapmak istediğimizde tek bir veritabanı kullanan tüm servis bileşenlerini olaya dahil etmemiz gerekecek ve çok fazla iş yükü meydana gelecektir. Uygulamalarda bu tarz sorunların meydana gelmesi Microservice mimari yaklaşımının ortaya çıkmasına sebep olmuştur.
Peki servislerin neden birbirinden bağımsız olmasını istediğimizi biraz daha detaylandırmamız gerekirse örneğin, bizim bir uygulamamız var ve uygulamamızda çok fazla trafik alan servislerimiz var. Bu servislerimizin yoğunluk durumunda gelen istekleri karşılayamadığını ve sürekli hata aldığı düşünelim. Bu durumda yapmamız gereken uygulamamızı scale etmektir. Scale kavramını dikey (vertical) ve yatay ölçeklendirme (horizontal scaling) olarak açıklayabiliriz.



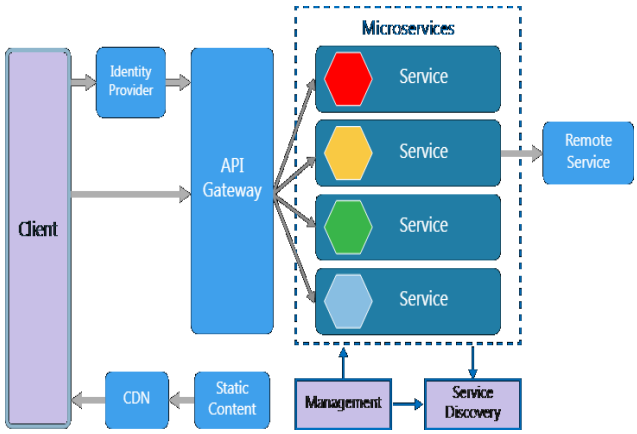
Dikey ölçeklendirmede fiziksel makinemizin yetersiz kalması durumunda yeni bir kaynak ekleyip (Ram, CPU, disk gibi) makineyi çoklama işlemi yapabiliriz ya da sanal makineler üzerinden makinemizi çoğaltabiliriz. Böylece gelen istekleri yönetmeye çalışırız. Fakat bu her zaman en iyi yöntem değildir çünkü fiziksel kaynaklarımız sınırlıdır ve bize çok fazla geliştirme maliyetine yol açabilir. Dikey ölçeklendirme yöntemi Monolith mimaride kullanılır.
Microservislerde ise scaling işlemi yatay ölçeklendirme yapılarak gerçekleştirir. Yatay ölçeklendirmede makinenin kendisini değil de uygulamada ihtiyacımız olan servisleri çoğaltırız. Böylece çok fazla istek alan servisleri birbirlerinden bağımsız yapıp gelen istekleri kolayca karşılayabiliriz.

Kavramlarım kafamızda daha iyi canlanabilmesi için hangi durumlarda Monolith ya da Microservis kullanılması gerektiğini örnek bir senaryo üzerinden anlamaya çalışalım.

Kampanya dönemlerinde çok fazla yoğunluk yaşayan e-ticaret sitelerinden biri olan N11'i hem alıcı ve hem de satıcı tarafından inceleyebiliriz. Senaryomuzda alıcı favorilerinde tuttuğu ürünü sepete eklemek istiyor ve servis geç cevap dönüyor sonunda alıcı uzun uğraşlar sonucunda sepetteki ürünün ödemesini gerçekleştiriyor fakat alıcı satın aldığı ürünü siparişlerim ekranında listelenmediğini görüyor. Burada kullanıcı servisinde bir sorun olduğu açıkça belli oluyor. Aynı şekilde alıcının gerçekleştirdiği satın alma işlemini normalde satıcının mağaza yönetim panelinde başarılı bir şekilde görmesini bekleriz. Örnek senaryomuzda satıcı, kullanıcının aldığı bu ürünü panelinde yeni bir sipariş olarak görüyor ama siparişi onayla butonu geç cevap dönüyor ve bir sonraki aşama olan kargo barkod oluşturma işleminde de hata aldığını düşenelim. Burada büyük bir senkronizasyon sorunu olduğu ve servislerin görevlerini tam olarak yerine getiremediği açıkça belli oluyor. Bu durumda N11'in hedefi sürekli istek alan servislerini tek bir uygulama içerisinde birbirlerine bağımlı Monolith yönetmek yerine, ağ trafiği yüksek servislerin bağımlılıklarını en aza indirip, ölçeklendirme yaparak Microservis mimarisini kullanması daha mantıklıdır. Örnek senaryomuzda N11, uygulamasında Monolith servis kullanmaya devam ederse bir paketin içinde bulunan sınıflara, metotlara instance yaratarak ulaşabiliyor. Uygulamasına yeni bir özellik eklemek istediğinde ise diğer servislerde de değişiklik yapması kod karmaşıklığına sebep olabiliyor. Uygulamada tek bir veritabanı üzerinden birbirine bağımlıkları yüksek servislere sahip olduğunu ve servislerin kendi networkleri içinde tutulduğundan dolayı ağ trafiğinden de bahsedemediğimizi biliyoruz.



Monolith Architecture



Microservices Architecture

Bu durumda N11'in Microservis yapıya geçmesi ile servisler birbirinden bağımsız olup, kendine ait bir sunucu ile her bir servisin ayrı bir veritabanına sahip olup aralarındaki yönlendirme, network trafiğini Gateway Api ile gerçekleştirebilir. Fakat çok sayıda küçük servislerin kullanılması durumunda servisler arasında iletişim problemi ve ağ gecikme problemi yaşanabilir. Bu durumda Load Balancer kullanarak yoğun istek alan servislerden ağ trafiğini azaltabilirler. Service Discovery sayesinde ise Feign Client kullanarak ulaşmak istedikleri servislerin hangi ağda hangi port üzerinde olduğunu görüp, dns name ile servislerin birbirini bulmasını sağlayabilirler. Aynı zamanda Hysterics gibi devre kesiciler kullanarak servisler arasındaki bağımlılığı yönetebilirler. Örneğin, sipariş listeleme servisindeki bağlantı kesilirse satın alma servisi, sipariş listeleme servisi cevap verene kadar burada gerçekleşen işlemleri gidip bir veritabanına yazar, sipariş listeleme servisi ayağa kalktığında ise buradaki işlemleri görüp gerekli işlemleri gerçekleştirebilir. Böylece bir serviste olan kesintinin diğer servisin çalışmasını etkilememeye çalışmak uygulamadaki en önemli amaçlarından biri olabilir.

Sonuç olarak, örnek senaryomuz üzerinden giderek de hangi mimari yapının hangi uygulamalarda kullanacağımızı iyi düşünüp karar vermeliyiz. Eğer ihtiyacımız yoksa Microservis mimari yapıp işlerimizi daha da zora sokmamalıyız. Çünkü Microservis mimari geliştirmenin maliyeti Monolith servise göre daha zor ve yüksektir. Eğer uygulamamızda Monolith mimaride servisler birbirine yetiyorsa, Microservis yapıya geçme düşüncesi doğru olmayabilir.

Kaynaklar

https://docs.microsoft.com/tr-tr/azure/architecture/microservices/
https://www.sourcefuse.com/blog/microservices-vs-monolithic-architecture-is-deploying-faster-and-scalable-applications-your-priority/
https://cloud.google.com/blog/topics/developers-practitioners/microservices-architecture-google-cloud
https://martinfowler.com/articles/microservices.html