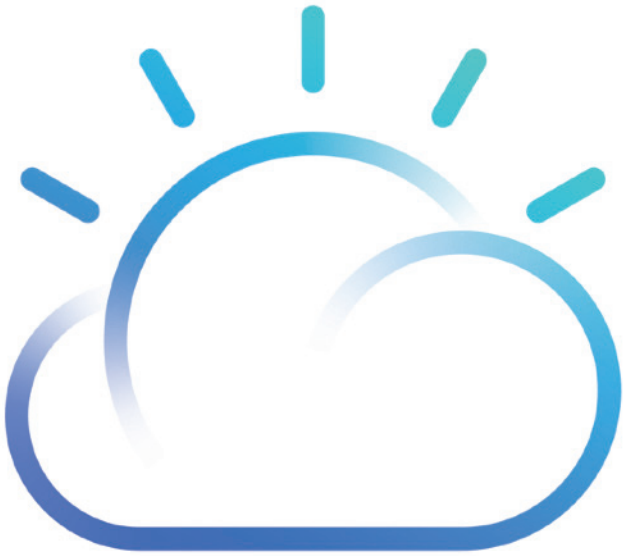
IBM Cloud

Mikro hizmetler kılavuzu

Mikro hizmetleri anlamak



Roland Barcia – IBM Kıdemli Mühendisi, CTO Microservices, NYC IBM Cloud Garage
Kyle Brown – IBM Kıdemli Mühendisi, CTO Bulut Mimarisi
Richard Osowski – IBM Kıdemli Teknik Personel Üyesi, Mikro Hizmetler Uygulaması

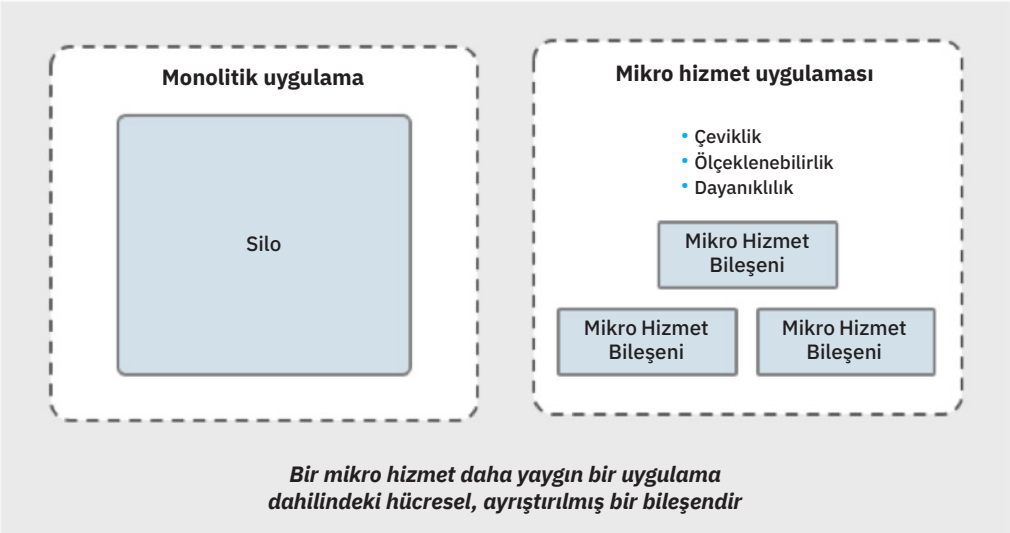
İçindekiler

Genel Bakış

Mikro hizmetler uygulama mimarisi tarzında, bir uygulama mikro hizmetler adı verilen birçok ayrı ağa bağlı bileşenden *oluşur*. Mikro hizmetler mimari tarzı, SOA (Hizmet Odaklı Mimari) mimari tarzının bir evrimidir. SOA hizmetleri kullanılarak oluşturulan uygulamalar, teknik entegrasyon sorunlarına odaklanma eğilimindedir ve uygulanan hizmetlerin seviyesi genellikle ince detaylı teknik uygulama programlama arabirimleridir (API'lar). Buna karşılık, mikro hizmetler yaklaşımı, büyük parçalı iş API'ları aracılığıyla açık iş becerileri sağlar.

İki yaklaşım arasındaki en büyük fark hizmete alınma şekilleridir. Uzun yıllardan beri uygulamalar monolitik bir şekilde paketlenmektedir; geliştiricilerden oluşan bir ekip, bir işletme ihtiyacı için gereken her şeyi yapan büyük bir uygulama oluşturur. Uygulama bir kez oluşturulduğunda birden çok kez hizmete alınır. Mikro hizmetler mimarisinde geliştiriciler, her biri uygulamanın bölümlerini uygulayan çok sayıda bağımsız küçük uygulamalar oluşturur ve paketler.

Basit bir şekilde, mikro hizmetler mimarisi büyük silo uygulamalarını daha yönetilebilir, tamamen ayrıştırılmış parçalara bölmekle ilgilidir.



Şekil 1: Monolitik uygulamalar ile mikro hizmetlerin karşılaştırması

Mikro hizmetler mimarisi kullanılarak oluşturulan uygulamaların uygulanmasını sağlayan beş basit kural vardır:

1. **Büyük monolitleri çok sayıda küçük hizmete ayırın** - Tek bir ağda erişilebilen hizmet, mikro hizmetler uygulaması için en küçük hizmete alınabilir birimdir. Her hizmet kendi sürecini yürütür. Bu kural konteyner başına bir hizmet olarak adlandırılır. Konteyner bir Docker konteynerini veya bir Cloud Foundry çalışma zamanı gibi bir hizmet mekanizmasını ifade eder.

2. **Hizmetleri tek bir fonksiyon için optimize edin** – Geleneksel bir monolitik SOA yaklaşımında, tek bir uygulama çalışma zamanı birden çok iş fonksiyonunu gerçekleştirir. Bir mikro hizmetler yaklaşımında, hizmet başına sadece bir iş fonksiyonu vardır. Bu, her hizmetin yazılmasını ve sürdürülmesini daha basit hale getirir. Bu Tek Sorumluluk İlkesi (SRP) olarak adlandırılır.

3. **REST API'ları ve mesaj aracıları ile iletişim kurun** – SOA yaklaşımının dezavantajlarından biri, SOA hizmetlerini uygulamak için çok sayıda standart ve seçenek bulunmasıdır. Mikro hizmetler yaklaşımı, bir hizmetin maksimum basitlik elde etmek için uygulayabileceği ağ bağlantı türlerini kesin olarak sınırlar. Benzer şekilde, mikro hizmetler, veri tabanı üzerinden örtülü iletişim ile sunulan sıkı bir bağlantıdan kaçınma eğilimindedir. Hizmetten hizmete tüm iletişim, hizmet API'sinden olmalı veya en azından Talep Kontrol Düzeni gibi açık bir iletişim modeli kullanmalıdır. [Hohpe ve Woolf].

4. **Hizmet başına CI/CD uygulayın** – Birçok hizmetten oluşan büyük bir uygulamada, farklı hizmetler farklı oranlarda gelişmektedir. Her hizmetin benzersiz, sürekli bir entegrasyon/dağıtım hattı doğal bir hızda ilerlemesine izin verir. Bu, sistemin her yönünün sistemin en yavaş hareket eden kısmının hızında kuvvetli bir şekilde serbest bırakıldığı monolitik yaklaşımla mümkün değildir.

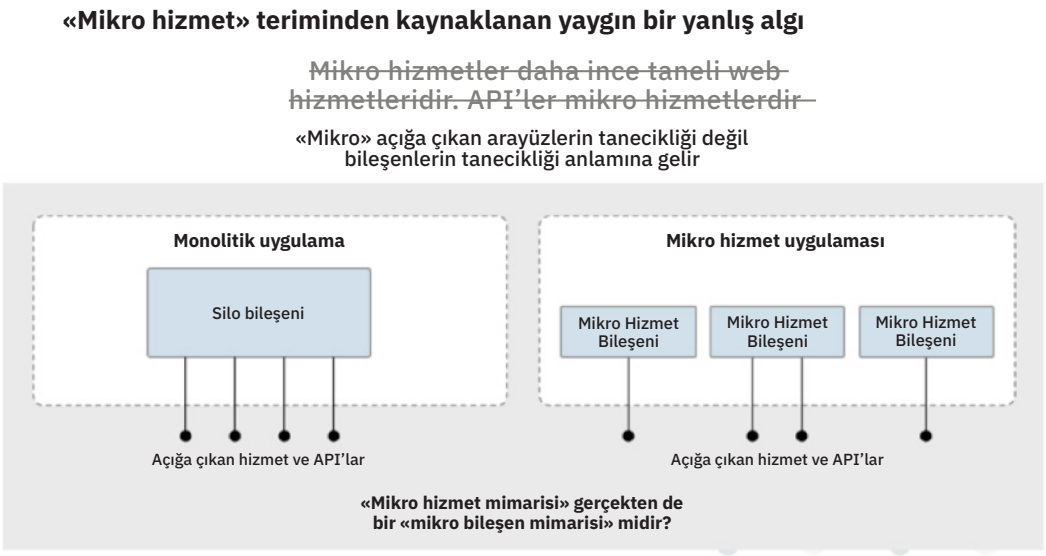
5. **Hizmet başına yüksek kullanılabilirlik (HA)/kümeleme kararları uygulayın** – Büyük sistemler oluştururken kümeleme herkese aynı beden uyar gibi bir yaklaşım değildir. Monolitteki tüm hizmetlerin aynı düzeyde ölçeklendirilmesini içeren monolitik yaklaşım, bazı sunucuların aşırı kullanılmasına ve bazılarının kullanılmamasına, hatta daha da kötüsü, iş parçacığı havuzları gibi mevcut tüm paylaşılan kaynakların ve başkaları tarafından tekeline alan bazı hizmetlerin yetersizliğine yol açar. Gerçek şu ki, geniş bir sistemde, tüm hizmetlerin ölçeklendirmesi gerekmemektedir; kaynakları korumak için bir çok hizmet en az sayıda sunucuda devreye alınabilir. Diğerleri çok büyük sayılara kadar ölçekleme gerektirir.

Bu beş kuralın bir araya getirilmesi ve yararları, mikro hizmetler mimarisinin bu kadar popüler hale gelmesinin başlıca nedenidir.

Mikro hizmetler mimarisi, büyük ölçekli ticari uygulamalar geliştirmek için fiili bir standart haline gelmiştir. Mikro hizmetlerde, bu yeni mimarisel terimin bir tanımı, Martin Fowler mikro hizmetleri tanımlıyor:

«Kısacası, mikro hizmet mimari stili, her biri kendi sürecinde çalışan ve genellikle bir HTTP kaynak API'si olan hafif mekanizmalar ile iletişim kuran küçük hizmetler paketi olarak tek bir uygulamanın geliştirilmesine yönelik bir yaklaşımdır. Bu hizmetler, iş olanakları etrafında yapılandırılmıştır ve tam otomatik hizmete alım sistemi tarafından bağımsız olarak hizmete alınabilir. Farklı programlama dillerinde yazılabilen ve farklı veri depolama teknolojileri kullanabilen bu hizmetler için asgari bir merkezi yönetim mevcuttur.»

SOA ve API'ler gibi mikro hizmetler ve paradigmalar arasındaki en önemli farklardan biri, hizmete alınan ve çalışan bileşenlere odaklanmadır. Mikro hizmetler, aşağıdaki şekilde gösterildiği gibi, arabirimlerden ziyade hizmete alınan bileşenlerin parçalarına odaklanır.



Şekil 2: Bir API ve bir mikro hizmet arasındaki farklılık

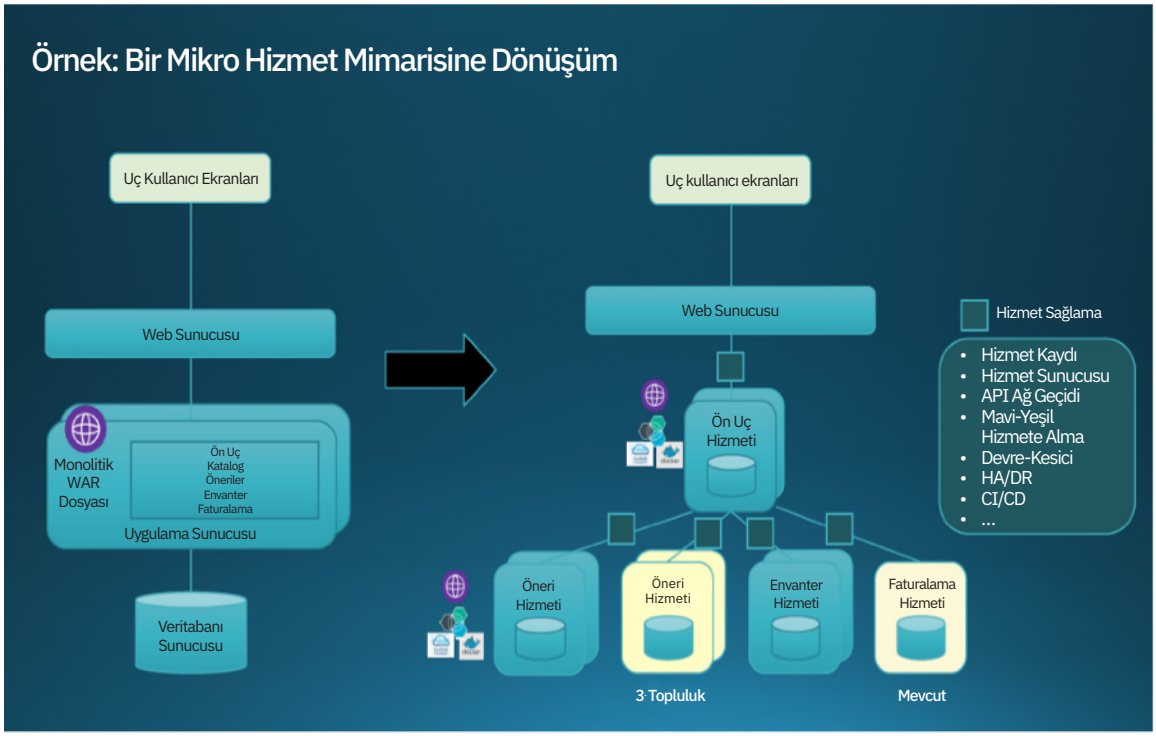
Operasyonlarla yönetilen birçok kuruluş, kullanım durumlarının ve işlevlerinin tek ve büyük bir uygulamaya yerleştirildiği monolitik uygulamalar oluşturur. Operasyonlarda bir miktar kolaylık olurken, bu uygulamaların mikro hizmetlere dayalı olarak değiştirilmesi büyük çaba gerektirir. Üç faktör mikro hizmet geliştirmeye yol açar:

1. **Geliştirme ekiplerinin nasıl organize edildiği:** Mikro hizmetlerin geliştirilmesi bir uygulamayı en iyi şekilde hizmetin tüm yaşam döngüsüne sahip olan küçük ekipler tarafından bağımsız olarak hizmete alınan ve çalıştırılan iyi tanımlanmış arabirimlere sahip tek işlevli modüllere ayırma üzerine odaklanan bir mühendislik yaklaşımı ile yapılır. Mikro hizmetler, kapsamı ve değişim riskini azaltırken, kişiler arasındaki iletişim ve koordinasyonu en aza indirerek dağıtımı hızlandırır.

2. **Uygulamaların nasıl oluşturulduğu:** Mikro hizmet tabanlı uygulamalar, oluşturuldukları ve çalıştırdıkları ortam hakkında bazı varsayımlarda bulunurlar. Ortam, genellikle bulut yerel uygulamaları *veya 12 faktör uygulamaları olarak adlandırılır*. Mikro hizmet tabanlı bir mimari, esnek ölçekleme, değişmez hizmete alım, tek kullanımlık örnekler ve daha az öngörülebilir altyapı gibi kavramları içeren standartlaştırılmış bir bulut ortamının güçlü yanlarından faydalanır ve zorluklarını barındırır.

3. **Uygulamaların nasıl sunulduğu ve çalıştırıldığı:** Konteynerlerin uygulamaları çalıştırmak için standart bir yol olarak kullanılması, mikro hizmet tabanlı uygulamaların paketlenmesini ve çalıştırılmasını sağlar. Konteynerler yeni bir teknoloji değildir. Linux konteynerleri, bir kontrol Linux ana bilgisayarında birden fazla izole edilmiş Linux sistemini (veya konteynerleri) çalıştırmayı mümkün kılan bir işletim sistemi özelliğidir. Linux konteynerleri, tam sanal makinelere hafif bir alternatif olarak hizmet eder. Konteynerler yeni bir konsept olmamasına rağmen, Docker gibi yapılar konteynerleri çalıştırmak için görüntü yaratmanın bir yolunu tasarlayarak kullanımlarını yaygınlaştırmıştır. Bu, bir uygulamayı ve tüm parçalarını paketlemenin standart bir yolunu sunar, böylece ortamlar arasında hareket ettirilebilir ve değişmeden çalışır. Cloud Foundry gibi diğer yapılar uygulamaları çalıştırmak için konteynerleri kullanır, ancak sanallaştırmayı soyutlar.

Aşağıdaki şekil, monolitik uygulama mimarisinin nasıl bir mikro hizmet tabanlı hale dönüştüğünü göstermektedir.



Şekil 3: Monolitik uygulamalar ve mikro hizmetler arasındaki farklılık

Şekil 3, bir perakende web sitesinin tüm bileşenlerini barındıran tek bir kurumsal arşivi göstermektedir. Uygulama arşivi, katalog ve envanter gibi tüm iş işlevlerinin yanı sıra web sitesi mantığı, iş mantığı ve her bileşen için süreklilik mantığını içerir. Buna ek olarak, uygulama genellikle sıkı bir şekilde birleştirilmiş veri modellerini içeren ve diğer uygulamalarla paylaşılan tek bir veritabanını paylaşır.

Görüntünün sağ tarafında, iş yeteneklerinin artık kendi verileriyle ayrı uygulamalarda hizmete alındığına dikkat edin. Bu yeni mimari tarzı, mikro hizmetler iletişimi, veri sahipliği, veri senkronizasyonu ve dayanıklılığı ile ilgili endişeleri ortaya çıkarmaktadır.

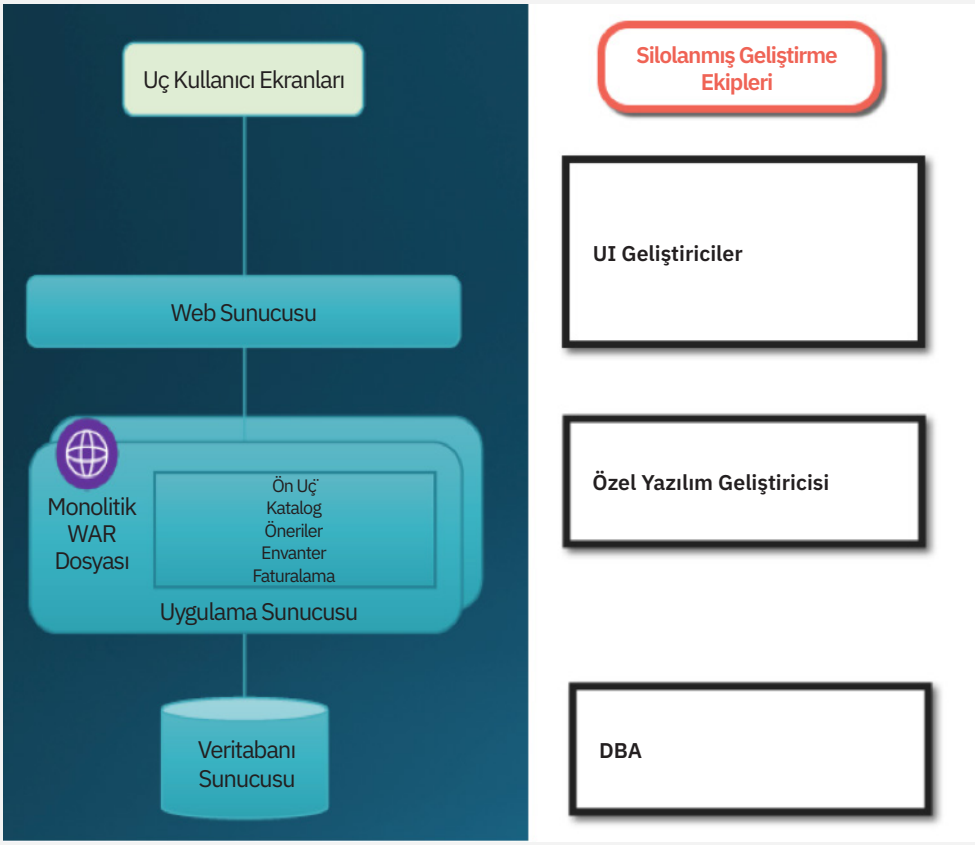
Bir Mikro Hizmet Mimarisinin Karakteristik Özellikleri bölümü James Lewis ve Martin Flower tarafından hazırlanmış olup, mikro hizmet tabanlı uygulamaların önemli unsurlarından bahsetmektedir.

Yukarıdaki referans örnek kullanılarak, içerdiği unsurların bir özeti şu şekildedir:

Hizmetler yoluyla bileşenlendirme: Bu konu bu yazıda daha önce ele alınmıştır.

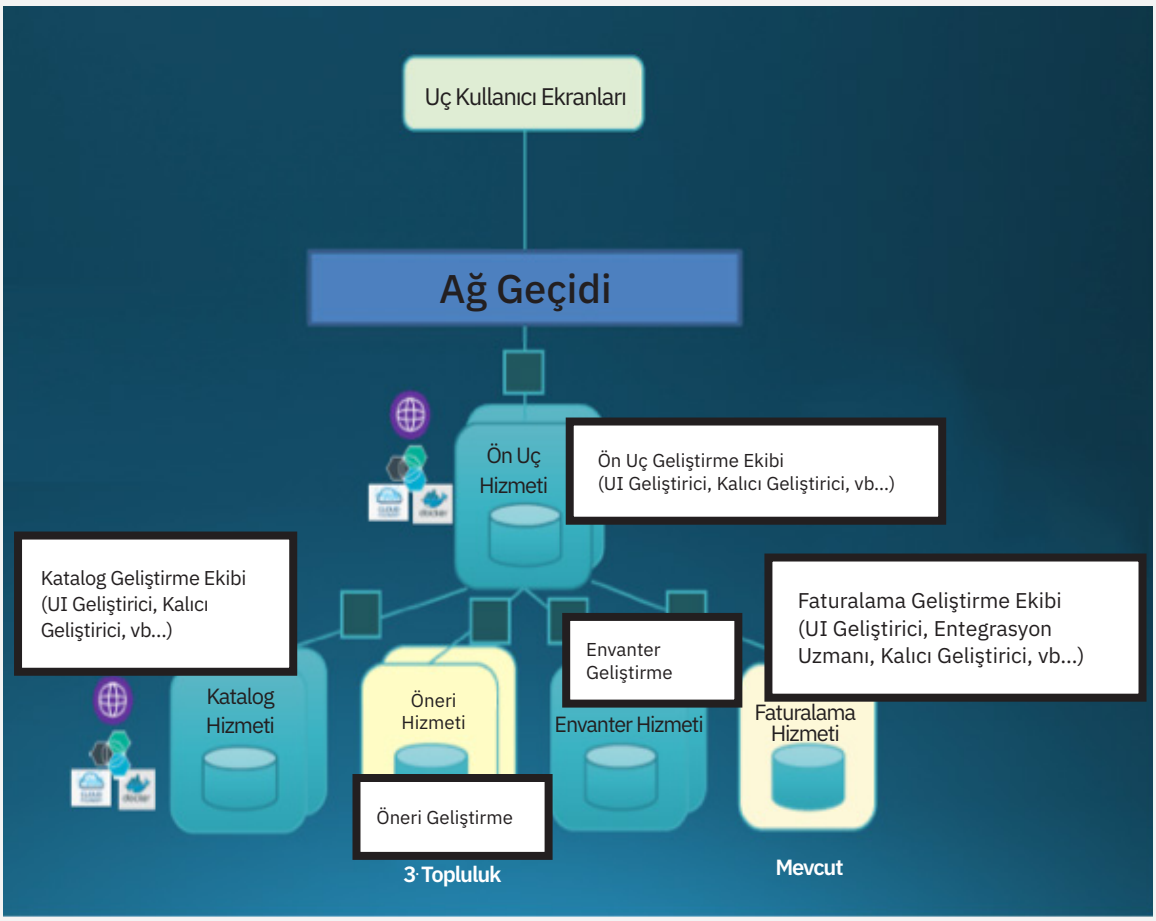
İş becerileri etrafında organize edildi: Daha önce ekip geliştirme kavramını ele aldık ve iş becerileri etrafında organize olmak geliştirme ekipleri hakkında düşüncelerimizde değişiklikler gerektirir.

Ekiplerin görevlerle aşağıda gösterildiği gibi nasıl organize edildiğini düşünün:



Şekil 4: Silolanmış geliştirme ekipleri

Bunu, şekil 5'in gösterdiği gibi, işletme becerileri etrafında organize eden bir mikro hizmetler tabanlı ekip yaklaşımıyla karşılaştırın.



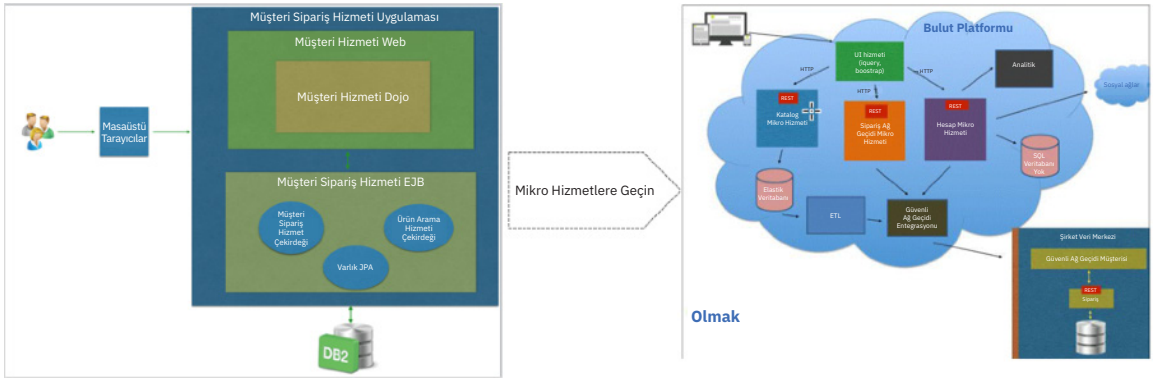
Şekil 5: İş becerileri tabanlı geliştirme ekipleri

Mikro hizmetler ile işletme sorunlarını giderin

Çoğu şirket, mikro hizmet tabanlı bir yaklaşım kullanarak yeni, yerel bulut uygulamaları oluşturur. Bununla birlikte, daha hızlı hareket etmek için birçok şirket mevcut monolitik uygulamaları yeniden düzenlemeye de ihtiyaç duymaktadır.

Monolitik uygulamaları taşıyın

Şekil 6, monolitik bir mimariden mikro hizmetlere dayalı bir mimariye geçişin bir örneğini göstermektedir. Bu durumda, bir perakendeci daha hızlı hareket etmek, müşteri hakkında daha fazla şey öğrenmek ve ödemeler gibi modern özellikleri devreye almak için mikro hizmetlere doğru yol almak ister.



Şekil 6. Bir mikro hizmetler tabanlı mimariye geçiş

Yukarıdaki senaryodaki perakendeci, monolitikten bir mikro hizmetler mimarisine geçmek için şu aşamaları izlemiştir:

1. **Kataloğun nasıl işleneceğini belirleyin.** Çözmeleri gereken ilk iş sorunu, öğelerin kataloğunun nasıl yönetileceğiydi. Müşteriler ürün verilerini bulamadılar ve perakendeci verileri başka sitelere gönderemedi. Ekip, aşağıdaki adımları kullanarak işletme kataloğu için tek bir mikto hizmet oluşturmaya karar verdi:

    - Verileri aramak ve yeni veri düzenlerini tanımlamak için yeni yollar sunmak üzere verileri bir elastik aramaya aktardılar
    - Mevcut web sitelerini yeni bir aramaya bağladılar
    - Bu pilot ile, yeni bir CI/CD modelini temel olarak sundular.

Ekip, yönetimi bir mikro hizmetler modeline geçmeye ve işi genişletmeye ikna etti.

2. **Müşteri hakkında daha fazla bilgi edinin** - Müşteri hakkında daha fazla bilgi edinmek için ekip bir hesap mikro hizmeti oluşturdu:

    - İlk olarak, işletmeyi tüketici odaklı bir yapıdan envanter odaklı bir organizasyona nasıl değiştireceklerini anladılar. Yeni bir müşteri modeli tasarladılar ve yapılandırılmamış bir veri modeli sağlayan Mongo DB veya Cloudant gibi bir NOSQL veritabanı kullandılar. Zamanla, müşteri verilerinin analitik, pazarlama ve bilişsel verilere dayalı olarak zenginleştirileceğini biliyorlardı.
    - Siparişleri izlemek için mevcut müşteri verileri kullanıldı.

3. **Yeni bir kullanıcı deneyimi oluşturun** - Ekip, mobil platformlar, web platformları ve yeni bir yerel mobil uygulama için yeni bir ön uç oluşturdu. Modern bir kullanıcı arayüzü ve katalog ile son kullanıcı için yeni bir deneyim yarattılar. Yeni katalog, temel işten oluşan ve henüz ayrılmayacak kadar karmaşık olan mevcut sipariş mantığıyla entegre edildi.

4. **Sipariş mikro hizmeti** - Ekip, mobil cihazlar için yeni sipariş API'leri oluşturmaya ve mevcut işlemlerle entegre olmaya odaklandı. İşletme, mevcut kayıt sistemine (SOR) çağrılan bir adaptör mikro hizmet oluşturmaya karar verdi. Bu adaptör katmanında yeni modern ödemelere entegre olma fırsatını yakaladılar.

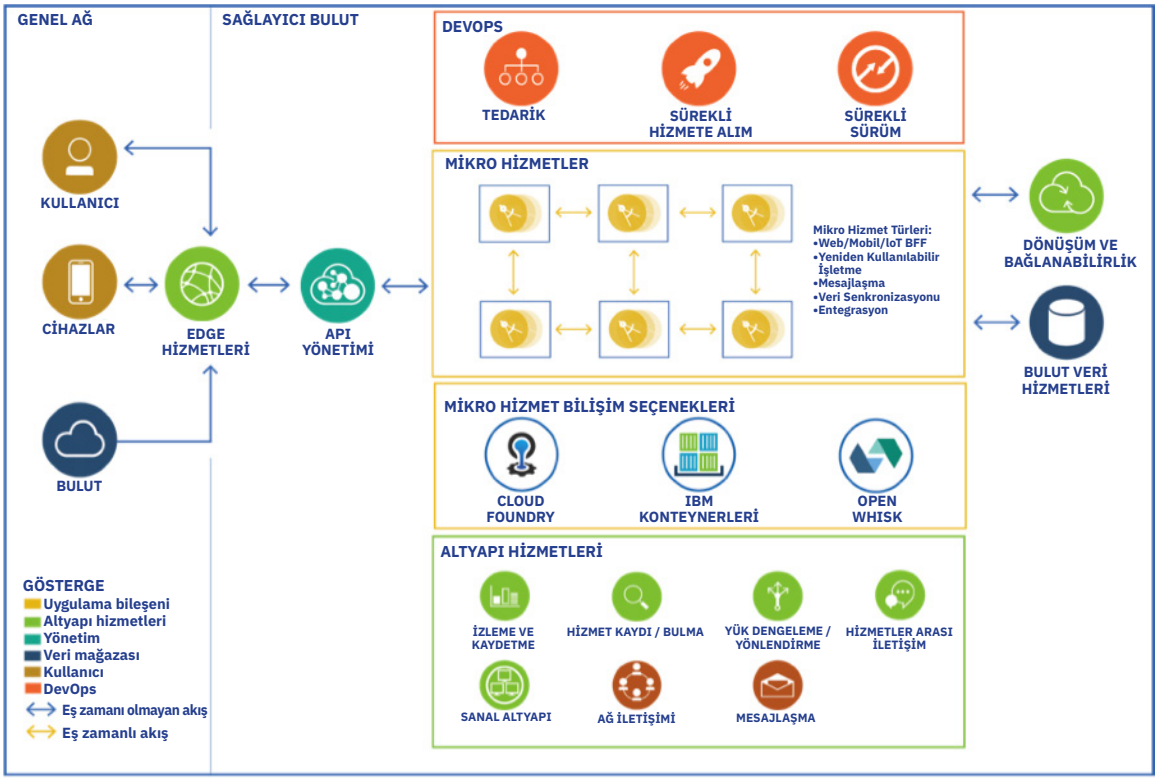
5. **İşi genişletin** - Perakendeci, yeni bir açık artırma özelliği ekleyerek yeni mikro hizmet modelini geliştirmek suretiyle iş modelini genişletmiştir.

Mikro hizmetler mimarisi

Şimdi bir mikro hizmetler tabanlı mimarinin ayrıntılarını öğrenin.

Mikro hizmetlerin becerileri

Uygulama bileşenleri, bulutta mikro hizmetler olarak çalışır ve mikro hizmet yapısı aracılığıyla birbirleriyle iletişim kurar. Farklı uygulama türleri farklı düzenler kullanır. Size daha sonra bir web/mobil örneğini göstereceğiz. Şekil 7, bir mikro hizmetler tabanlı mimarinin ihtiyaç duyduğu-becerileri göstermektedir.



Şekil 7. Bir mikro hizmet tabanlı mimarinin becerileri

Diyagramda soldan sağa doğru okunduğunda, bir sonraki adım mikro hizmet becerilerini gözden geçirmektir. Birçok mikro hizmet API'lar aracılığıyla açığa çıkarılır ve bu mikro hizmetlerin bir kısmının bir *API* Ağ Geçidi üzerinden tüketilmesi *gerekir*.

Bir API Geçidi, uç noktaların bir proxy'si kadar basit olabilir. Bazı API Ağ Geçitleri; ilk temasta güvenlik, izleme ve API versiyonlama veya üçüncü taraf tüketimi için bir geliştirici portalına sahip tam API yönetim sistemleri ile daha gelişmiş bir yapıdadır.

DevOps

Tedariği, sürekli hizmete alımı ve sürekli sürümü kapsayan başarılı bir mikro hizmet mimarisi oluşturmak için DevOps kullanarak güçlü bir otomasyon stratejisi geliştirmelisiniz. Stratejiniz şunları içermelidir:

- **Tedarik** - Başarılı bir mikro hizmet mimarisi, uygulama ortamları için otomatikleştirilmiş tedarik gerektirir. Bir Hizmet katmanı olarak Platform, bu hizmeti genellikle IBM® Cloud®'daki Hizmet olarak Konteyner (CaaS) gibi yönetilen hizmetler aracılığıyla sağlar.® Bulut®. Bir Hizmet olarak Altyapı (IaaS) katmanı üzerinde, Kubernetes veya Docker Datacenter (DDC) gibi düzenleme motorları kullanarak bir konteyner altyapısını çalıştırırsanız, bu ortamların bir sanal makine ortamında otomatik tedarik kullanılarak nasıl tedarik edileceğini ve güncelleştirileceğini otomatikleştirmelisiniz.

- **Sürekli hizmete alım** - Geliştirme ortamında Docker görüntüleri veya mikro hizmet uygulamaları için otomatik bir oluşturma ve hizmete alım süreci gereklidir. Çok bulutlu bir stratejiniz-varsa, oluşturma ve hizmete alma otomasyonunuzun, otomatik ölçekleme veya bulut politikaları gibi işleri nasıl yaptığınıza ilişkin farklılıkları içermesi gerekir.

- **Sürekli sürüm** - Bir DevOps ortamı, güçlü bir test odaklı geliştirme ve otomatik test kültürünü destekler. Bu, birim testi, otomatik fonksiyonel test ve performans test ortamı doğrulama testini içerir.

Mikro hizmet bilişim seçenekleri

Mikro hizmetleri çalıştırmak için çeşitli bilişim seçenekleri vardır:

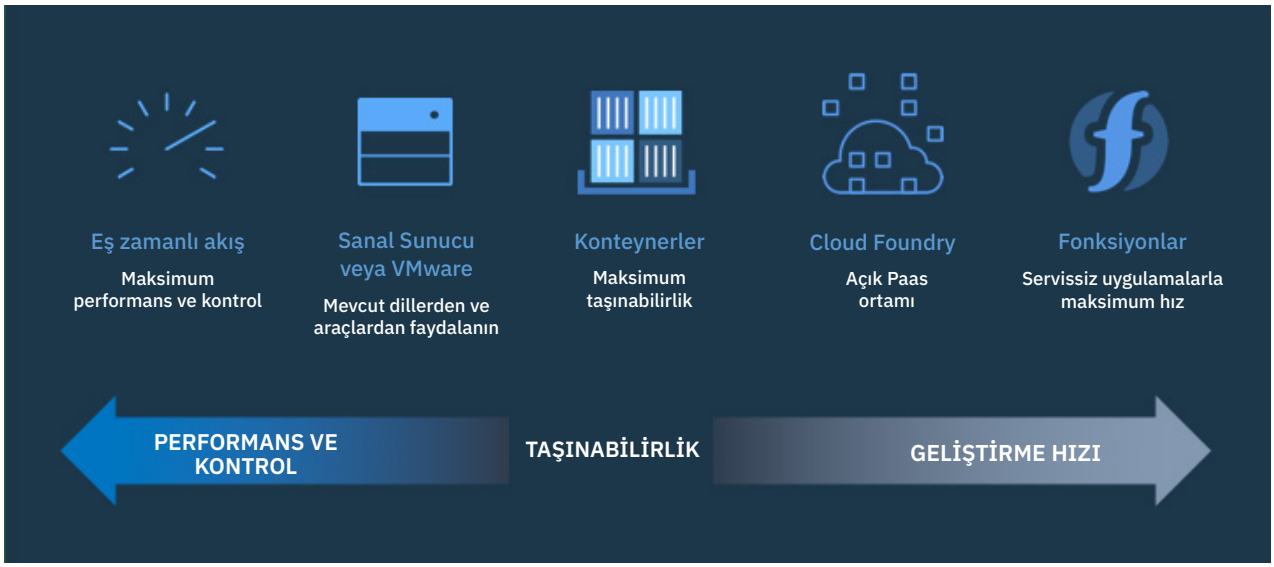
- **Docker konteynerler** - Bu konteynerler, bulutta ve tesis ortamlarında en yüksek-taşınabilirliği sağlar. Docker konteynerleri kullanmak güçlü DevOps gerektirir.

- **Cloud Foundry** - Bu açık kaynaklı PaaS, mikro hizmetlerin nasıl çalıştığı, birbirleriyle nasıl iletişim kurduğuna dair bilgiler içerirken konteynerler gibi sanallaştırma için soyutlama sağlar. Cloud Foundry, bir miktar taşınabilirlik sunarken, bulut ortamında çalışma için bir fuller stack gerektirir.

- **Fonksiyonlar** - En yüksek düzeyde soyutlama ve kullanım kolaylığı ile olay tabanlı bilişim seçeneği. Geliştiriciler, bulut bileşenlerinin merkezi bir ileti merkezine gönderdiği olayları yanıtlamak için basit olay işleyicilerini hizmete alır. Tüm sanallaştırma soyutlanır.

- **Sanal makineler veya yalın donanımlar** - Mikro hizmet tabanlı uygulamalar oluşturabilir ve bunları sanal makinelerde (VM) çalıştırabilirsiniz. Bu, daha fazla tedarik ve DevOps'un başarılı olmasını gerektirir. VM'ler ve yalın donanımlar, kolaylık pahasına maksimum esnekliği sunar.

Şekil 8 bu seçenekleri özetler:



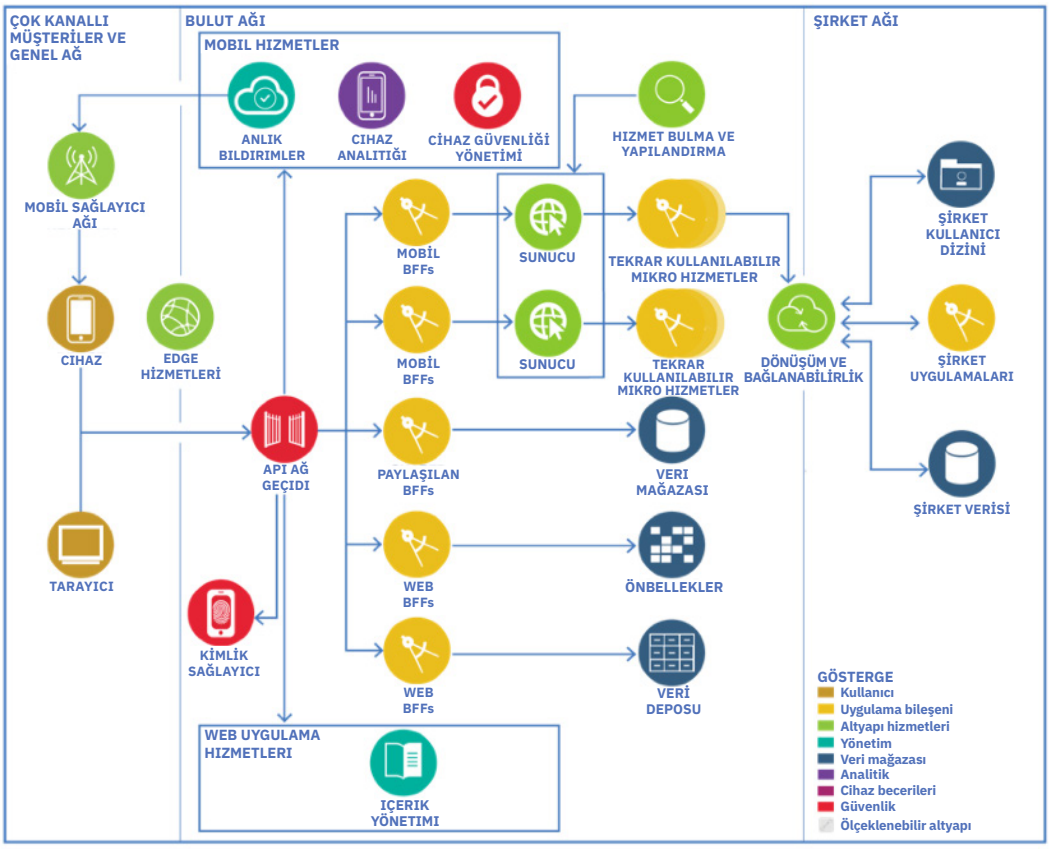
Şekil 8. Mikro hizmetler için bilişim seçenekleri

Altyapı hizmetleri

Mikro hizmetleri çevreleyen bulut ortamı; ağ oluşturma, mesajlaşma, mikro hizmet iletişimi, kaydetme ve izleme, sanallaştırma, hizmet bulma ve proxy, dayanıklılık özellikleri ve daha fazlası için gerekli olan yapı ve hizmetleri sağlar.

Uygulama mimarisi

Aşağıda bir mikro hizmet uygulama mimarisinin örneği vardır.



Şekil 9. Mikro hizmetler uygulama mimarisi

Bu mimarinin bileşenlerini özetlemek gerekirse:

- Bu örnekteki OmniChannel uygulamaları hem bir yerel iOS uygulamasını hem deAçısal tabanlı bir web uygulamasını içerir. Diyagram onları bir cihaz ve bir tarayıcı olarak tasvir eder.

- Mobil uygulamalar, işlemler ve iş için cihaz analizlerini toplamak üzere IBM Mobile Analytics for Cloud hizmetini kullanır.

- Her iki müşteri uygulaması da API Ağ Geçidi üzerinden API çağrıları yapar. API Ağ Geçidi, IBM API Connect, API güvenliğini uygulamak için bir OAuth Sağlayıcısı sunar.

- API'ler, Ön Uçlar için Arka Uç (Backend for Frontends) (BFF'ler) olarak adlandırılan Node.js mikro hizmet düzenleri olarak uygulanır. Diğer olası adlar, Experience (Deneyim) API (xAPI) veya Iteration (İterasyon) API içerir. Bu katmanda, ön uç geliştiriciler genellikle ön uçları için arka uç mantığını yazarlar. Inventory (Envanter) BFF, Express yapısı kullanılarak uygulanır. Social Review (Sosyal İnceleme) BFF, API Connect LoopBack yapısı kullanılarak uygulanır. Bu mikro hizmetler, IBM Cloud'da Cloud Foundry uygulamaları olarak çalışır.

Müşteri durum çalışmasını okuyun

Teknik ayrıntıları alın

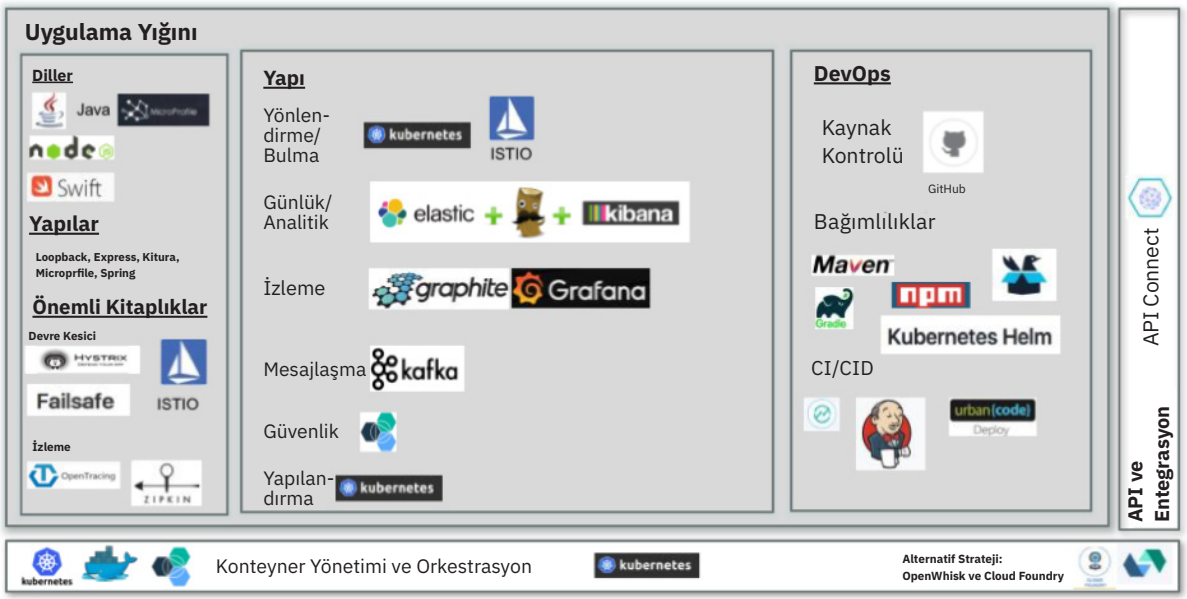
- Node.JS BFF'ler, yeniden kullanılabilir Java mikro hizmetlerinin başka bir katmanını çağırır. Gerçek dünyadaki bir projede, genellikle farklı bir ekip bunu yazacaktır. Bu yeniden kullanılabilir mikro hizmetler, Spring Boot kullanarak Java'da yazılır. Docker kullanılarak IBM konteynerler içerisinde çalışırlar.

- Düğüm BFF'leri ve Java mikro hizmetleri, bir mikro hizmet yapısını kullanarak birbirleriyle iletişim kurar. Örnekler Istio ve Netflix OSS'yi içerir.

- Java mikro hizmetleri bulut veri tabanları ve entegrasyon katmanları ile etkileşime girebilir.

Mikro hizmet istifi

Mikro hizmet yığınınızı tanımlamak önemlidir. Aşağıda, hali hazırda yapılmış bazı seçeneklere sahip bir yığın örneği verilmiştir. Uygulama, yapı ve DevOps dahil olmak üzere farklı katmanları tanımlamanız gerekir. IBM'in bu mikro hizmetleri etkinleştirme teknolojilerini nasıl belirlediğini görmek için lütfen IBM Mikro Hizmet Karar Rehberi'ne bakın.



Şekil 10. Mikro hizmet yığını

Uygulama

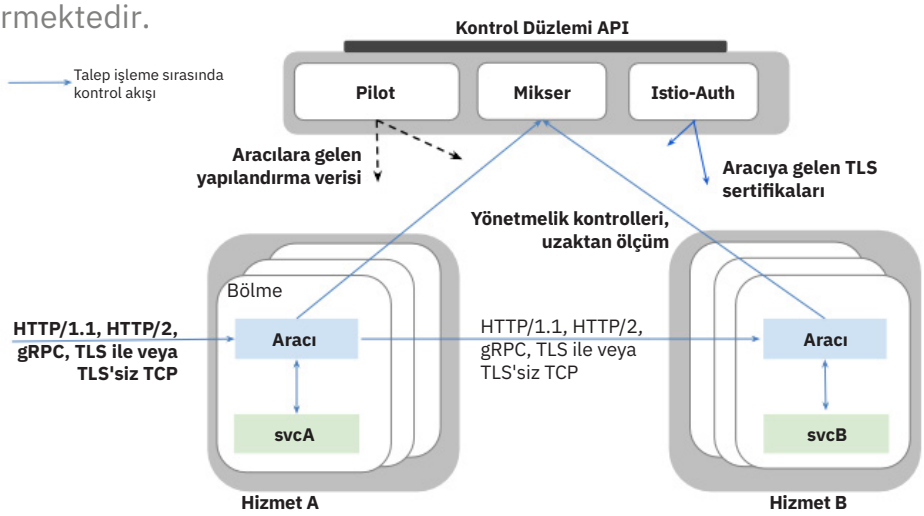
Dil ve çalışma zamanı yığını, uygulamanızı yazar. Örnekler arasında, MicroProfile, WebSphere Liberty, Spring gibi alternatif Java yığınları veya StrongLoop gibi Node.js yığınları gibi modern, hafif Java Platformu, Enterprise Edition (Java EE) yığınları bulunur.

Belirli dil kütüphanelerinin, devre kesme ve izleme gibi mikro hizmet uygulama öğelerini ele almaları beklenir.

Mikro hizmet çatısı

Mikro hizmet çatısı, bir dizi gerekli beceriyi sağlayan bir beceriler yığınıdır:

- Yönlendirme ve bulma, bulutla uyumlu, mikro hizmetler arası iletişim için temel bir özelliktir. Bir mikro hizmet örneği otomatik olarak ölçeklendirilebilir, dinamik kümelere sahip olabilir. Adından yola çıkarak mikro hizmeti bulmalı ve ardından çalışan bir örneğe yönlendirmelisiniz. Zuul/Eureka gibi çerçeveler bunu Netflix OSS yığınının bir parçası olarak sağlar. Istio, poliglot tabanlı tercih edilen bir seçenek olup, altta yatan playforma hizmet çağrı yönlendirmesinin ve kontrolünün daha hassas şekilde kontrol edilmesini sağlar. Aşağıdaki şekil Istio mimarisinin bir örneğini göstermektedir.



Şekil 11. Istio mimarisi

- Günlükler ve analitik için, akış günlüklerini yakalayan ve bunları çeşitli yerlere yönlendiren bir yığın gereklidir. Bunun için çeşitli çerçeveler vardır.

- Çalışma zamanı, uygulama ve devre kesici bilgileri dahil olmak üzere çeşitli veriler için izleme panoları gereklidir. Günlük yığını kullanarak, bu araçlar birleştirilmiş bir görünüm sağlar.

- Mesajlaşma ve güvenlik katmanları da önemlidir.

DevOps

Tedarik, düzenleme, oluşturma, hizmete alım ve operasyonlar için gereksinimleri tamamlamak üzere güçlü bir DevOps araç kümesine sahip olmanız gerekir.

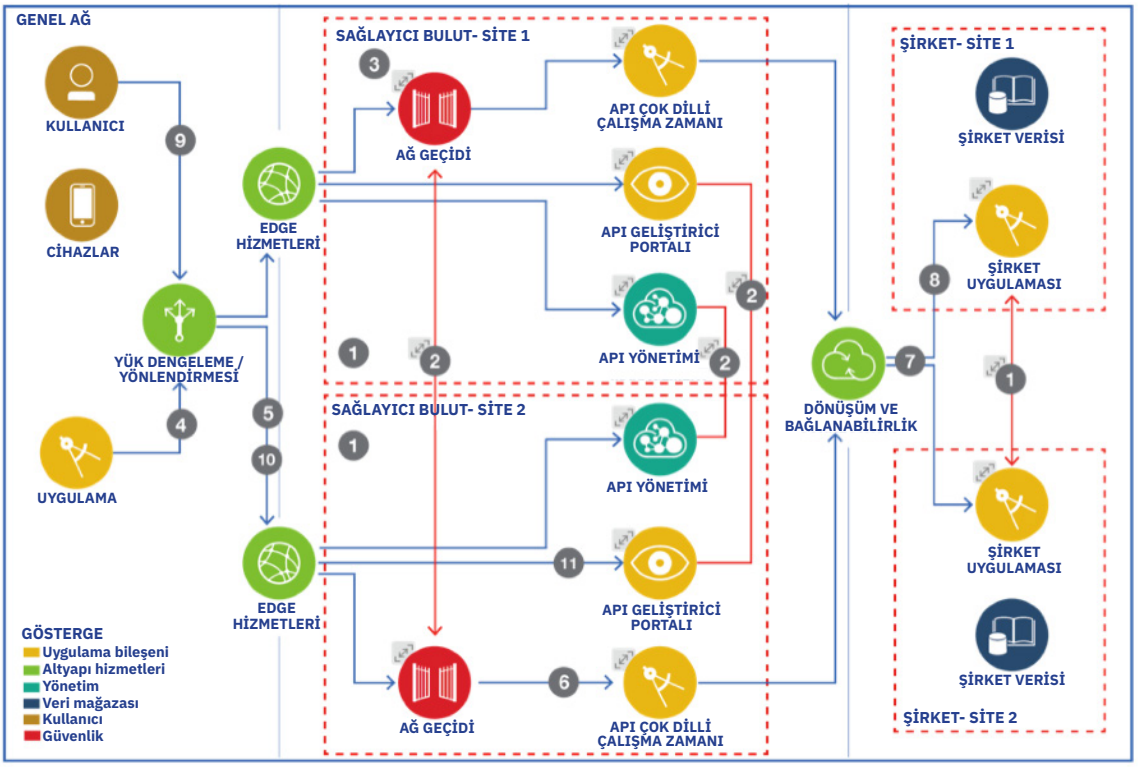
Konteyner yönetimi

Bir mikroservis kümesini desteklemek için IBM Cloud Container Service, Kubernetes veya Docker Data Center gibi bir konteyner düzenleme ve çalışma zaman ortamı gerekir.

Mikro hizmet tabanlı mimarilerde dayanıklılık

Bu tür ince taneli bileşenlerle, bir mikro hizmet mimarisindeki tüm dayanıklılık noktalarının farkında olmalısınız. Bu, yüksek kullanılabilirlik, yük devretme, DR, devre kesilmesi ve izolasyonu içerir.

Şekil 12, mikro servis tabanlı mimarilerde dayanıklılığı, bir global yük dengeleyiciyi ve çok alanlı yedeklemenin kullanımını göstermektedir.



Şekil 12. Bir mikro hizmet mimarisinde dayanıklılık

Mikro hizmetler için yüksek kullanılabilirlik ve acil kurtarma düzenleri

Dayanıklılık hususları:

- Yedek veri arka uçları
- Çoğaltılan veri arka uçları
- Bölge başına birden fazla hizmete alım
- Birden fazla bölgeye hizmete alım (üç veri merkezi)
- Global yük dengeleme

Dayanıklılık ile uğraşırken, yüksek kullanılabilirlik (HA) ve acil kurtarma (DR) arasında bazı ayrımlar yapmak önemlidir.

HA, güncellemeleri hizmete alma, barındırma, sanal makinelerini yeniden başlatma, barındırma işletim sistemine güvenlik yamaları uygulama gibi bakım faaliyetleri sistem üzerinde yapılırken hizmetlerin son kullanıcılara sunulduğundan emin olunmasını sağlar.

HA genellikle büyük güç kesintileri, depremler, ciddi donanım hataları veya bağlantı kaybı nedeniyle tüm site kaybı gibi büyük planlanmamış sorunlara işaret etmez. Bu gibi durumlarda, hizmetlerin katı hizmet düzey hedefleri (SLO) varsa, tüm uygulama yığınını (altyapı, hizmetler ve uygulama bileşenleri) en az iki farklı IBM Cloud bölgesine dayanarak yedeklemeniz gerekir. Bu tipik olarak bir Acil Kurtarma (DR) mimarisi olarak tanımlanır.

DR çözümlerini uygulamak için birçok seçenek vardır. Sadelik açısından, farklı seçenekler üç ana kategori halinde gruplandırılabilir:

Aktif/Pasif, Aktif/Bekleme ve **Aktif/Aktif.**

Aktif/Pasif

Bu seçenek, tüm uygulama yığınını bir konumda aktif tutarken, başka bir uygulama yığınını farklı bir konumda hizmete alınır, ancak boşta kalır veya kapatılır. Birincil sitenin uzun süreli kullanılamaması durumunda, yedek sitede uygulama yığını aktive edilir. Genellikle bu, birincil siteden alınan yedekleri geri yüklemeyi gerektirir. Bu yaklaşım, veri kaybı sorun yaratıyorsa veya hizmetin kullanılabilirliği kritik önem taşıyorsa tavsiye edilmez, çünkü kurtarma zamanı hedefi (RTO) birkaç saatten azdır.

Aktif/Bekleme

Bu seçenekle, tam uygulama yığını hem birincil hem de yedek konumlarda aktiftir; ancak, yalnızca birincil site kullanıcıların işlemlerine hizmet eder. Yedekleme sitesi, veritabanı çoğaltımı veya disk çoğaltımı gibi veri çoğaltım yollarıyla ana konumun bir kopyasını saklar. Birincil sitenin uzun süre kullanılmaması durumunda, tüm müşteri işlemleri yedek siteye yönlendirilir. Bu yaklaşım, iyi kurtarma noktası hedefi (RPO) ve RTO (dakikalar) sağlar, ancak çift hizmete alım nedeniyle Aktif/Pasif'ten önemli ölçüde daha pahalıdır. Kaynak israfı vardır, çünkü bekleme varlıkları ölçeklenebilirliği ve verimliliği artırmak için kullanılamaz.

Aktif/Aktif

Bu durumda, her iki bölge de aktiftir ve müşteri işlemleri çevrimsel sıralama, yük dengeleme ve konum gibi önceden tanımlanmış politikalara göre her iki bölgeye dağıtılır. Bir site başarısız olursa, diğer site tüm müşterilere hizmet verir. Bu konfigürasyonla sıfıra yakın RPO ve RTO elde etmek mümkündür. Dezavantaj ise; her iki bölgenin, her iki konumda kullanılabilir olduğunda kapasitelerinin yarısında kullanılsalar bile, tam yükün üstesinden gelmek için boyutlandırılmaları gerekliliğidir. Bu durumda, IBM Cloud hizmeti için Otomatik Ölçeklendirme, kaynakları BlueCompute örnek uygulamasına benzer şekilde gereksinimlere göre tahsis eder.

Ölçeklenebilirlik ve performans hususları

Dayanıklılık eklemek genellikle performans ve ölçeklenebilirliği geliştirmek için de kullanılabilen yedek hizmete alımlara sahip olmayı gerektirir. Yukarıdaki bölümde açıklanan Aktif/Aktif durumu için bu geçerlidir. Global uygulamalar söz konusu olduğunda, Akamai veya Dyn'in global yönlendirme çözümlerini kullanarak yanıt süresini ve gecikmeyi kısaltmak için kullanıcı işlemlerini en yakın konuma yeniden yönlendirmek mümkündür.

Mikro hizmet projelerinin uygulanması

Bir sonraki mantıklı adım, mikro hizmet tabanlı projelerin kuruluşunuzda nasıl uygulanacağını anlamaktır.

Sıfırdan veya mevcut bir monolitik sistemden bir mikro hizmet sistemi kurabilirsiniz. Çoğu IBM müşterisi mevcut monolitelerini güncelleme amacıyla mikro hizmet yolculuğuna başladıklarından, bu bölüm, mevcut bir monolitik mimariden çalışırken mikro hizmetlerin nasıl değerlendirileceğine ve uygulanacağına odaklanır.

Önceden uygulama geliştirmeksizin mikro hizmet projeleri oluşturmakla ilgili veya mikro hizmet oluştururken monolit bir ilk yaklaşım kullanın. Kısacası, öncelikle fikrinizi doğrulamak için uygulamanızı herhangi bir yolla oluşturmanız anlamına gelir.

Ardından, ilk monolitinizi bir mikro hizmet projesine ölçeklendirip dönüştürmek için bu tanıtım belgesindeki ilkeleri uygularsınız. İşletmeye değer katmayan, mimari açıdan saf mikro hizmetlerin yaratılmasında hiçbir değer yoktur.

Önce monolit yaklaşımı kullanarak bir mikro hizmet uygulamasını hayata geçirmek için mükemmel bir yeni hesap The Chronicles'daki Game On! ekibi tarafından belgelenmiştir.

Mevcut uygulamanın dönüşümü

Artık mikro hizmet tabanlı bir dönüşüm geçirirken anlamanız gereken birçok kavram olduğunu biliyorsunuz. Bu bölümde, başarılı bir mikro hizmet projesini uygulamak için anlamanız gereken üç alanı ele alıyoruz: işiniz, kültür ve beceri setiniz, teknolojiniz.

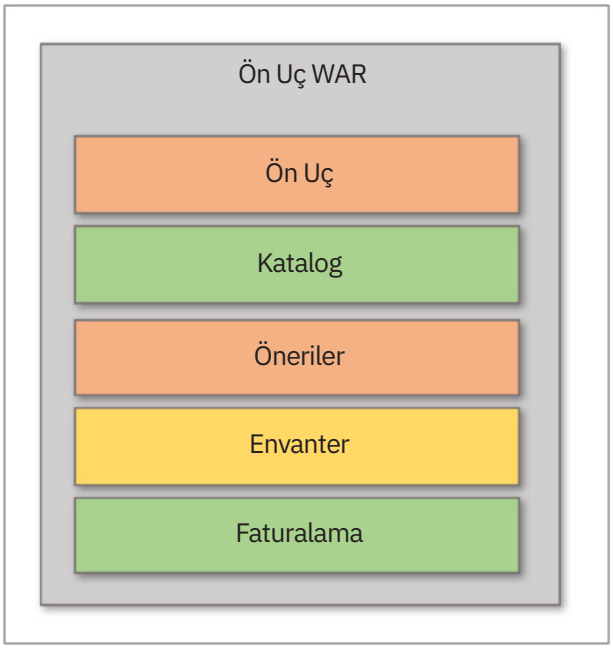
İş ihtiyaçlarınızı anlayın ve tanımlayın

Neden mikro hizmetlere geçmeyi düşünüyorsunuz? Birçok işletme için, işletmeye daha hızlı değer katmak ve daha iyi bir kullanıcı deneyimi sunmak için daha verimli yazılım geliştirme ve işletim uygulamaları gerekmektedir.

Bir mikro hizmet projesinin mevcut uygulamalarınız ve altyapınız üzerindeki etkisini anlayabilmeniz için, işletmenizin hangi bölümlerinin başarılı sonuçlar elde etmek için daha yavaş ilerlediğini anlamalısınız. Çoğu durumda, kuruluşun katılım sistemleri (SOE) yavaşlamaya neden olur. Bu sistemler, web, mobil ve API'ler dahil olmak üzere pek çok kanal üzerinden kullanılabilir. Hız eksikliği, mikro hizmet tabanlı bir mimariye geçişin temel nedenidir.

Mikro hizmet odaklı yaklaşımı benimsemeden önce, neyin yeterince hızlı pazarlanamadığını bilmelisiniz. Öncelikle uygulamanın hangi parçalarının, daha hızlı hale getirilmesi için iyileştirmelere ve değişikliklere ihtiyaç duyduğunu belirlemelisiniz. Buradan, mevcut monolitin hangi bölümlerinin mikro hizmet evrimi için hedefleneceğini tespit edebilirsiniz.

Kullanıcı deneyim akışları veya mimari diyagramları gibi tasarım ve mimari özellikler bu aşamada kullanım için değerlidir. Ekip, şekil 13'te gösterildiği gibi, monolit bölümlerini hızlı bir şekilde belirleyebilir ve önceliklendirebilir.



Şekil 13. Öncelik için Kırmızı-Sarı-Yeşil ölçek kullanan mevcut monolit problem noktaları

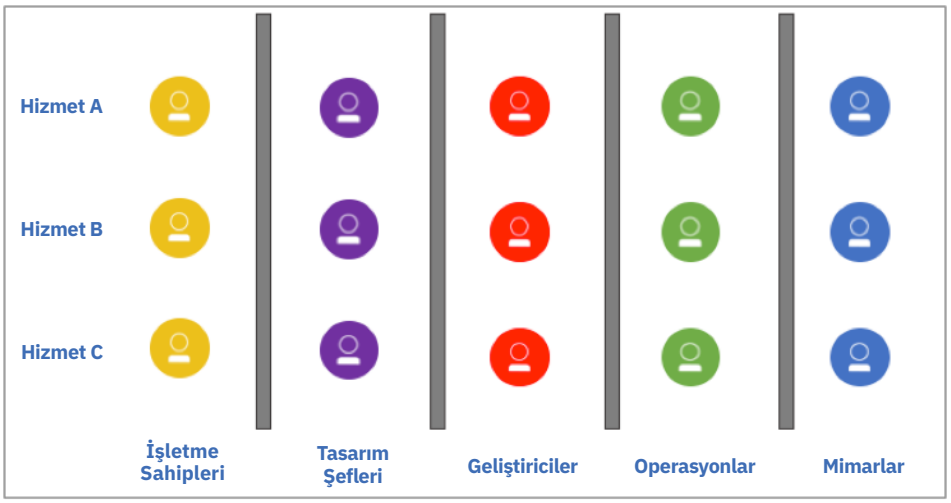
Bu tanımlama sürecinin yorucu olması ve yapı olarak yinelemeli olması gerekmez. Önemli hedefler:

- Monolitinizin sağladığı ayrı iş fonksiyonlarını tanımlamak
- Bu iş fonksiyonlarını değiştirmek için gerekli göreceli hız ve karmaşıklığı anlamak
- İşletmenin, belirli iş fonksiyonları için daha hızlı geri bildirim döngüleri görme ihtiyacını anlamak.

Kültürünüzü ve beceri setinizi anlayın

Mikro hizmetlere dayalı mimarilere özgü olmamakla birlikte, bir kuruluşun ekiplerinin, kültürünün ve beceri kümelerinin tam olarak anlaşılması, dijital bir dönüşüm sırasında kritik öneme sahiptir.

Tipik olarak, mühendislik monolitlerinde, çoğu organizasyon yazılım geliştirme yaşam döngüsü boyunca gerektiği ölçüde katılımla birlikte silolara inşa edilir. Bu genellikle kısıtlayıcı roller ve sorumluluklarla birlikte iyi tanımlanmış sınırlar oluşturur. Şekil 14, tipik bir monolitik organizasyon yapısını göstermektedir.

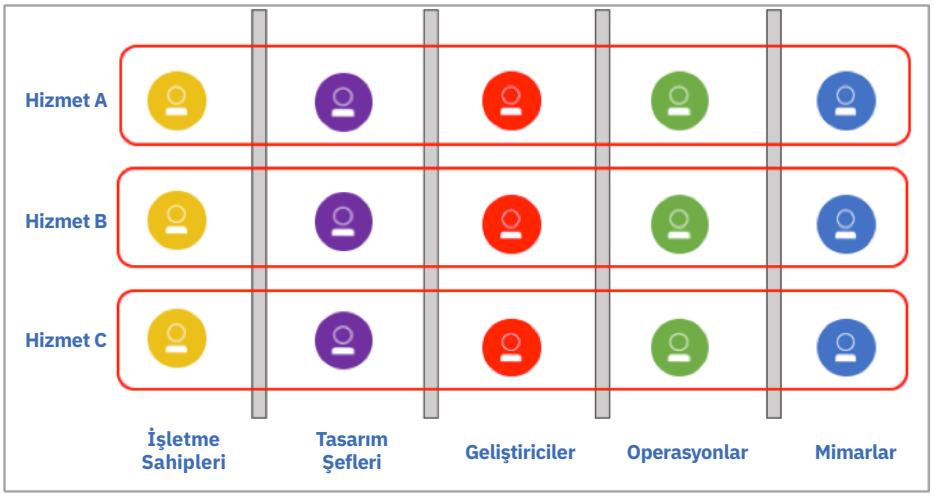


Şekil 14. Geleneksel BT organizasyon yapıları

Karşılaştırma olarak, Mikro hizmet mimarileri sadece ekipler tüm yazılım geliştirme ve operasyon yaşam döngüsüne sahip olma gücüne sahip olduklarında başarılı olabilir. DevOps yaşam döngüsünün tamamına sahip olmak için, ekiplerin farklı görev ve sorumluluklara sahip üyelere ihtiyacı vardır.

Bu çapraz fonksiyonel ekipler, mikro hizmet tabanlı mimarileri güçlendirmek için oluşturulmuştur. Silolanmış ekip üyeleri yerine tüm görev ve sorumluluklar aynı ekipte yer almaktadır. Herkes, tasarımdan geliştirmeye ve operasyonlara kadar, birbirleriyle sıkı bir şekilde çalışır ve genellikle bir arada bulunurlar. Fiziksel ekip yapıları bu yazının kapsamı dışındadır, ancak sanal ekip sınırları, çapraz işlevsel bir şekilde oluşturulduğunda işletmenin dönüştürülmesinde en fazla başarıyı sağlarlar.

Tasarım, geliştirme ve operasyon paydaşlarının tümü ekipte temsil edildiği için, işletmenin tasarım deneyimlerini net bir şekilde tanımlamasına, olası teslimat zamanlarını anlamasına ve işletme giderlerini en aza indirmesine izin verir. Şekil 15 optimal bir DevOps ekip konfigürasyonunu göstermektedir.



Şekil 15. Mikro hizmetlerin başarısını sunacak optimize edilmiş DevOps ekipleri

Çapraz fonksiyonel bir ekip, ekipteki bireysel becerilerin hızlı büyümesini de destekler. Ekip, tasarımdan operasyonlara ve çalışma zamanı verilerine kadar mikro hizmetin sorumlu olduğu her şeye sahip olduğunda, tek ekip üyeleri, tek görevlere düşürülmez. Çoğu zaman, ön uç mühendisleri veritabanı yönetim becerilerini geliştirirken, operasyon odaklı ekip üyeleri kullanıcı arayüzlerindeki farklılıklar hakkında daha fazla bilgi edinirler. Genişleyen beceri setleri, bu şekilde tüm BT organizasyonunun mikro hizmetlerle başarılı olmasına yardımcı olur—çok özel rolleri doldurmak için uzmanları ararken çok yönlü ekip üyelerinden oluşan yeni ekipler oluşturmak çok daha kolaydır.

İş sorununuzu ve ekibinizin kültür ve beceri kümelerini ele almadığınız sürece, mikro hizmet teknolojisini etkili bir şekilde uygulayamaz, aynı süreçleri ve yapıları yerinde tutarsınız.

Teknolojiyi anlayın

Mevcut teknoloji yığınlarının uygun analizi, kuruluştan kuruluşa geniş bir yelpazeye sahiptir, ancak tanımladığımız basitleştirilmiş yaklaşım, mikro hizmet projelerinizin ilk ve sürekli başarısının sağlanmasına yardımcı olur. Küçük parçalardan başlamak ve yinelemeli, ilerici başarıları tanımlamak, her şeyi bir defada dönüştür yaklaşımından çok daha kolay ve verimlidir.

Şekil 16. Mikro hizmet dönüşümlerine yinelemeli yaklaşım

Teknolojinizi anlamanın ilk aşaması, mevcut monolit içinde bulunan büyük taneli hizmetleri tanımlamaktır. Bunu genellikle, tasarım ilkeleri akılda tutularak oluşturulmamış ve mevcut uygulamalara iyi tanımlanmış tasarım ilkelerini uygulamanızı sağlayan etki alanı odaklı tasarım (DDD) prensipleriyle örtüştürebilirsiniz. Bu büyük taneli hizmetleri tanımlamak, veri yapılarının karmaşıklığını, mevcut bileşenler ile yeni büyük taneli hizmetlerden sorumlu ekipler arasındaki bağlantı seviyesini anlamanıza yardımcı olur. Başarılı bir inceleme, hem belirli bir hizmetin hem de hizmetlerin genelinde veri sınırlarının açık bir şekilde anlaşılmasını sağlar.

Büyük taneli hizmetleri belirledikten sonra, bunları ince taneli mikro hizmetlere nasıl dönüştüreceğinize dair bir plan oluşturmanız gerekir. Önceki çalışmalarınıza dayanan bu mikro hizmetlerin hepsi benzer verilerle çalışmalı, kendi verilerini yönetmeli ve diğer hizmetler için okumaları ve yazmaları gereken verileri anlamalıdır. Buradan, ince taneli mikro hizmetlerin dayanıklılığını, ölçeklenebilirliğini, çevikliğini tanımlayabilir ve uygulayabilirsiniz.

Daha önce belirtildiği gibi, API'ler ve mikro hizmetler bire bir karşılaştırma değildir; bunlar daha büyük bütünün iki parçasıdır. İnce taneli mikro hizmetlerinizi daha iyi anladıktan sonra, hangi arayüzlerin kritik yol üzerinde olduğu, hangi arayüzlerin opsiyonel olduğu ve hangi arayüzlere artık ihtiyaç olmadığı dahil olmak üzere arayüzlerinizi de daha iyi anlayacaksınız. Mevcut olan bir arayüzü veya API'yi, büyük veya ince taneli mikro hizmetlerinizden biri ile eşleştiremiyorsanız, büyük olasılıkla buna gerek yoktur.

Mikro hizmet girişiminin büyüklüğünü ölçün

Tüm analiz ve planlama çalışmaları tamamlandığında, zaman çizelgelerini, teslimat hızlarını ve beklenen sonuçları belirlemenin zamanı gelmiştir. Bunlar büyük ölçüde değişecektir, ancak mevcutdan dijital dönüşüm projelerine kadar tamamen yeni bir süreç olmayacaktır.

İşin, ekip yapısının ve teknolojinin anlaşılmasının ağır yükü, ister kavram kanıtlama kapsamı, pilot kapsam isterse büyük ölçekli evrim kapsamı olsun, organizasyonun herhangi bir monolitin mikro hizmet evriminin bütününü anlamaya hazır olmasına yardımcı olacaktır.

Mikro hizmet düzenleri

Mikro hizmetler için geliştirme düzenleri

Geliştirmede, düzenler yaygın gereksinim türlerine bilinen çözümler uygulamak için faydalıdır ve bu mikro hizmetler için de geçerlidir. Martin Fowler'ın mikro hizmet tasarım prensiplerinden biri, mikro hizmetlerin «İş becerileri etrafında organize olmasıdır».[2] Bu doğrudan bir şekilde, bir şeyi dağıtabilmenizin bunu dağıtmanız gerektiği anlamına gelmemesinden kaynaklanır.

Bu düzenler yaygın olarak mikro hizmetler için kullanılır:

- Cephe **düzeni, bir sistem** veya alt sistem için özel harici bir API tanımlar. Bu düzenin alt metni, bu API'nin işletme odaklı olmasıdır.

- **Varlık ve Küme düzenleri, iş** arayüzleri açısından tasarım yapmak için kullanılmayan geliştirme ekipleri için doğrudan mikro hizmetlere eşlenen belirli iş kavramlarını tanımlamak için kullanışlıdır.

- **Hizmetler düzeni, tek** bir varlığa veya kümeye karşılık gelmeyen işlemleri, mikro hizmetlerin gerektirdiği varlık temelli bir yaklaşımla eşleştirmenin bir yolunu sunar.

- **Bağdaştırıcı Mikro Hizmetler düzeni, birçok** durumda geliştirme ekiplerinin düzen verisi üzerinde merkezi olmayan bir kontrolünün bulunmadığı kurumsal dünyada yararlıdır. Bir Bağdaştırıcı Mikro Hizmet'te, iki farklı API arasında uyum sağlarsınız. Bir API, geleneksel bir mikro hizmetle aynı etki alanını kullanan tekniklerle RESTful veya hafif mesajlaşma teknikleri kullanılarak oluşturulan, iş odaklı bir API'dir. İkinci API, eski bir API veya geleneksel WS-* tabanlı SOAP hizmetidir.

- **Strangler Uygulama düzeni, işletmelerin** ve uygulamaların asla yeşil alanlı bir ortamda yaşamadıkları gerçeğini ele alır. Mikro hizmetlerden en fazla yararlanabilecek programlar, yeniden yapılandırılabilen büyük monolitik uygulamalardır. Düzen, yeniden yapılandırmayı yönetmek için bir yaklaşım sunar. Strangler Uygulama düzeni bu yazının ilerleyen bölümlerinde ayrıntılı olarak ele alınmıştır.

Mikro hizmetler için operasyon düzenleri

Mikro hizmetler, tek bir hizmeti değiştirmeyi ve hizmete almayı hızlandırırken, aynı zamanda, bir dizi hizmeti yönetmek ve muhafaza etmek için, ilgili bir monolitik uygulamaya kıyasla daha büyük bir çaba sarf etmeyi de gerektirir. Geleneksel uygulama yönetimi için orijinal olarak geliştirilmiş olan mikro hizmetler için bu operasyon düzenleri, DevOps'un operasyonlar tarafına uygulanır:

- **Hizmet Kayıt düzeni, akış** yönündeki mikro hizmetlerin uygulanmasını değiştirmeyi mümkün kılar ve DevOps hattınızın farklı aşamalarında değişiklik gösterecek hizmet yer seçeneği sunar. Bu, kodunuza belirli mikro hizmet uç noktalarının gömülmesi önlenerek elde edilir. Hizmet Kaydı olmadan, uygulamanız, koddaki değişikliklerin mikro hizmet çağrı zincirinden yukarı doğru yayılmaya başlamasıyla birlikte hızla bocalayacaktır.

- **Korelasyon Kimliği ve Günlük Toplayıcı düzenleri** daha iyi bir yalıtım sağlarken aynı zamanda mikro hizmet hatalarının daha kolay ayıklanmasını mümkün kılar. Korelasyon Kimlik düzeni, bir dizi farklı dilde yazılmış belli sayıda mikro hizmetle iz yayılımına izin verir. Günlük Toplayıcı modeli, bir dizi farklı mikro hizmetten günlüklerin tek, aranabilir bir depoda toplanmasını sağlayarak Korelasyon Kimliğini tamamlar. Birlikte, bu düzenler, hizmet sayısından veya her çağrı yığınının derinliğinden bağımsız olarak, mikro hizmetlerin verimli ve anlaşılabilir hata ayıklamasına olanak tanır.

- **Devre Kesici düzeni, zaten** meydana geldiğini biliyorsanız, aşağı akım arızalarını ele almada zaman kaybetmekten kaçınmanıza yardımcı olur. Bunu yapmak için, bir *alt akış hizmetinin* arızalandığını algılayan ve bunu denemekten kaçınan yukarı akış hizmet çağrılarına kodun bir devre kesici bölümünü eklersiniz. Bu yaklaşımın yararı, her aramanın hızlı bir şekilde başarısız olmasıdır. Kullanıcılarınız için daha iyi bir genel deneyim sağlayabilir ve aşağı akış çağrılarının başarısız olacağını bildiğinizde iş parçacığı ve bağlantı havuzları gibi kaynakları yanlış yönetmekten kaçınabilirsiniz.

Strangler Düzenine odaklanılması

Fowler'ın Strangler Düzeni, etrafına sarılı olduğu bir ağacı boğan bir sarmaşık analojisine dayanır. Buradaki fikir, bir uygulamayı farklı işlevsel etki alanlarına ayırmak ve her defasında bir etki alanı olmak üzere bunları yeni bir mikro hizmet tabanlı uygulama ile değiştirmek için, bir işletme etki alanının farklı unsurlarına işlevsel olarak eşlenen münferit Üniform Kaynak Tanımlayıcıları (URI'ler) ile oluşturulan bir web uygulama yapısını kullanmanızdır. Bu iki unsur, aynı URI uzayında yan yana yaşayan ayrı uygulamaları oluşturur. Zamanla, yeni yapılandırılan uygulama, monolitik uygulamayı kapatabilmenize kadar orijinal uygulamayı bozar veya değiştirir.

Strangler Düzeni adımları dönüşüm, birlikte var olma ve ortadan kaldırmadır:

- **Dönüşüm** - Örneğin, IBM Cloud'da veya var olan ortamınızda, daha modern yaklaşımlara dayanan, paralel yeni bir site oluşturun.
- **Birlikte var olma** - Mevcut siteyi bir süre için olduğu yerde bırakın. Yeni uygulanan işlevsellik için mevcut siteden yeni siteye aşamalı olarak yeniden yönlendirme yapın.
- **Ortadan kaldırma** - Eski işlevselliği mevcut siteden kaldırın veya trafik sitenin o kısmından yeniden yönlendirildiği için bunu muhafaza etmeyi bırakın.

Bu düzeni uygulamanın en önemli yanı, her şeyi tek bir büyük taşımayla yapmaya çalıştığınızdan çok daha kısa bir zaman dilimi içerisinde artan bir değer yaratmasıdır. Ayrıca, mikro hizmetlerin benimsenmesi için size artan bir yaklaşım sunar; yaklaşımın ortamınızda çalışmadığını fark ederseniz, yönü değiştirmenin basit bir yolu vardır.

Strangler uygulama düzeni ne zaman çalışır ve ne zaman çalışmaz?

- **Web veya API tabanlı monolit -** Mevcut bir Web veya API tabanlı monolitten başlamak düzenin başarılı bir şekilde uygulanması için ilk şarttır. Strangler düzeninin amacı, size yeni ve eski işlevler arasında kolayca geriye ve ileriye doğru hareket etmenin bir yolunu sunmaktır. Uygulamanız bir web uygulamasıysa, URL yapısı size sistemin nasıl ve hangi bölümlerinin uygulandığını seçmek için bir yapı sunar. Bununla birlikte, REST'e dönüştürdüğünüz SOAP API'leri veya bir mesajlaşma sisteminde uygulanan kuyruklar kümesi gibi sabit bir API kümesine dayalı herhangi bir uygulama, yapıyı uygulamanıza izin verecektir. Öte yandan, büyük müşteri uygulamaları veya çok sayıda yerel mobil uygulama, bu yaklaşıma çok uygun değildir, çünkü uygulamayı kolayca ayırmanıza izin veren bir yapıya sahip olmaları gerekmez.

- **Standartlaştırılmış URL yapısı (URL'lerin gerçek kullanımı) -** Web uygulamalarının tümü, web'in yapısının, örneğin HTTP ve HTML'nin uyguladığı standartlara göre çalışsa da, web uygulamalarını devreye almak için çok çeşitli uygulama mimarileri kullanabilirsiniz. Bu yaklaşım içerisinde uygulamayı birbirinden ayırma girişiminizi zorlaştırabilecek pek çok yol vardır. Örneğin, sunucu talepleri altında bir ara katman olduğunda, örneğin bir portal yaklaşımı gibi, uygulamayı bölmek için URL'leri kullanmada sorun yaşayabilirsiniz. Geçiş ve yönlendirme kararı tarayıcı düzeyinde değil, uygulamanın daha derinlerinde yapılır ve bu da daha karmaşık bir durumdur.

- **Meta Kullanıcı Arayüzleri -** Kullanıcı Arayüzü iş süreci temelli olduğunda veya aceleyle oluşturulduğunda, kodu kullanıcı arayüzü ve iş mantığı için farklı mikro hizmetlere ayırmak zorlaşır. Yaklaşım yine de çalışır, ancak yığın boyutu (aşağıya bakınız) daha büyüktür ve tümü aynı anda uygulanmalıdır.

Strangler Uygulama Düzeni nasıl uygulanmaz

Strangler Uygulama Düzeni her şey için bir çözüm değildir. Her durumda veya özellikle de bunlarda uygulamak istemezsiniz :

- Her seferinde bir sayfa uygulamayın. En küçük «şerit» (aşağıdaki sürüm yönetimi bölümüne bakın) tek bir mikro hizmettir. Bu mikro hizmetin eksiksiz ve kendi kendine yeterli olması ve daha da önemlisi, yönettiği verilerin tamamen sahibi olması gerekir. Tutarlılık sorunlarından kaçınmak üzere verileriniz için iki farklı erişim yöntemine sahip olmaktan kaçınmak istiyorsunuz.

- Düzenin tümünü aynı anda uygulamayın. Eğer uygularsanız, Strangler Düzenini gerçekten uygulamaz ve kendinizi tekrar Big Bang yaklaşımında bulursunuz.

Strangler Uygulama Düzeni nasıl en iyi şekilde uygulanır

Dolayısıyla, tek bir sayfa çok küçükse ve tüm uygulama çok büyükse, Strangler Uygulamasını uygulamak için doğru tanecik düzeyi nedir? Başarılı olmak için, uygulama yapılandırmanızın iki farklı yönünü birbiriyle karıştırmanız gerekir:

- Arka ucunuzu mikro hizmetler tasarımına yapılandırma (iç **kısım**)

- Ön ucunuzu mikro hizmetleri barındırmak ve yapılandırmayı yönlendiren yeni işlev değişiklikleri yapmak için yapılandırma (dış **kısım**)

İç kısım ile başlayalım:

1. Uygulama tasarımınızdaki sınırlı içerikleri belirleyerek işe başlayın. Sınırlı bir içerik Eric Evans'ın Etki Alanı Odaklı Tasarım'ından başka bir düzendir. Kitaba göre, sınırlı bir içerik, «bir modelin bünyesinde geçerli olduğu içeriği tanımlar.»[3] Sınırlı bir içerik, daha geniş bir dizi iş modelindeki bir dizi varlığın anlamını kısıtlayan ortak bir kavramsal çerçevedir. Bir havayolu uygulamasında, uçuş rezervasyonu sınırlı bir içeriktir; havayolu sadakat programı ise farklı bir sınırlı içeriktir. «Uçuş» gibi terimleri paylaşabildikleri halde, bu terimlerin kullanıldığı ve tanımlandığı şekil oldukça farklıdır.

2. Yapılandırma için en küçük, en az maliyetli sınırlı içeriği seçin. Diğer sınırlanmış içeriklerinizi karmaşıklık derecesine göre en az karmaşık olandan en karmaşık olana doğru sıralayın. Yapılandırmanızın değerini kanıtlamak için en az karmaşık sınırlı içeriklerle başlayın ve daha karmaşık ve potansiyel olarak maliyetli yapılandırma görevlerini üstlenmeden önce süreci benimsemedeki sorunları çözün.

3. Kavramsal olarak, yukarıda açıklandığı gibi Etki Alanı Odaklı Tasarım'dan Varlık, Küme ve Hizmet düzenlerini uygulayarak *içerik içindeki mikro* hizmetleri planlayın. Bu noktada, sonraki adımlarda bu yaklaşımı kullanabilmeniz için hangi mikro hizmetlerin var olabileceğini anlamaya çalışıyorsunuz.

Ardından, dış kısma geçin:

1. İlişkilerinizi mevcut kullanıcı arayüzünüzdeki ekranlar arasında analiz edin. Özellikle, birkaç ekranı birbirine sıkıca bağlayan büyük ölçekli akışlara bakın. Bir havayolu web sitesi oluşturuyorsanız, bir akış, rezervasyon işlemini tamamlamak için bilgi sağlayan çok sayıda ilgili ekrandan oluşan bir bilet rezervasyonu yapıyor olabilir. Farklı bir akış, havayolunun sadakat programına kaydolmaya yönelik olabilir. Akış dizisini anlamak, bir sonraki yapılandırma adımına geçmenize yardımcı olur.

2. Mevcut kullanıcı arayüzünü veya yeni kullanıcı arayüzünü incelerken, iç kısımda belirlenen mikro hizmetlere karşılık gelen yönleri arayın. Hangi akışların hangi mikro hizmetler ile uyumlu olduğunu belirleyin. Bu adımın çıktısı, sayfanın bir tarafındaki akışların listesi ve diğer taraftaki her akışı uygulayabilecek mikro hizmetlerin listesidir.

3. Kullanıcı Arayüzü değişikliklerinin kendi içerisinde tutarlı olması gerektiği varsayımına dayanarak yığınınızı boyutlandırın. Bir veya daha fazla akışın bir müşteriye sağlanabilecek minimum değişim büyüklüğü olduğunu, fakat yığının kendisinin bir akıştan daha büyük olabileceğini varsayalım. Örneğin, iki veya üç ayrı akıştan oluşabilmesine rağmen, müşteri sadakatinin tümünü tek bir yığın olarak kabul edebilirsiniz.

4. Tüm bir parçayı bir kerede mi yoksa bir dizi olarak mı yayınlayacağınızı seçin.

Referanslar

- **IBM Cloud Garage Methodu, Mikroservis Mimari Merkezi**
- **Uçtan Uca Referans Uygulaması (GitHub)**
- **Mikroservisler için Yeniden Düzenleme**
- **Mikroservis Yapıları IO**
- **Mikroservisleri Keşfetme: Game-On (GitBook)**

[1, 2]Martin Fowler, *Mikroservisler bu yeni mimari kavramın bir tanımı, https://martinfowler.com/articles/microservices.html*

[3]Eric Evans, *Alan Odaklı Tasarım Tackling Complexity in the Heart of Software, 2003*

Son.