Uydu Yer Yazılım Sistemleri için Servis-yönelimli Mimari'den Mikroservis Mimarisine Geçiş Stratejisi

A Service-oriented Architecture to Microservice Architecture Migration Strategy for Satellite Ground Software Systems

Ali Yamuç[1] ve Uğur Melih Sürme[2]

Türk Uzay ve Havacılık Sanayii - Turkish Aerospace Industries, Ankara
[1]ali.yamuc@tai.com.tr
[2]msurme@tai.com.tr

Özet. Bu çalışmada, Türk Havacılık ve Uzay Sanayii tarafından geliştirilmekte olan servis-yönelimli mimarideki bir uydu yer yazılım sisteminin, mikroservis mimarisine geçiş stratejisi anlatılmıştır. Mevcut mimaride karşılaşılan sorunlar ve kısa vadede mikroservis mimarisine doğru geçişte yapılması gereken taktiksel dönüşümler paylaşılmıştır. Üst düzey güvenirlik, erişilebilirlik ve performans kalite özelliklerine sahip olması gereken Uydu Komuta Kontrol Yazılımının, mikroservis mimarisiyle bu gereksinimleri nasıl karşılayabileceğimiz açıklanmıştır. Geçiş stratejisi aşamalı bir şekilde planlanmış, aşamalarda tamamlanacak unsurlar izah edilmiş ve uzun vadeli planlamalara yer verilmiştir.

Anahtar kelimeler: Servis-yönelimli Mimari, Mikroservis Mimarisi, Uydu Yer Yazılım Sistemleri.

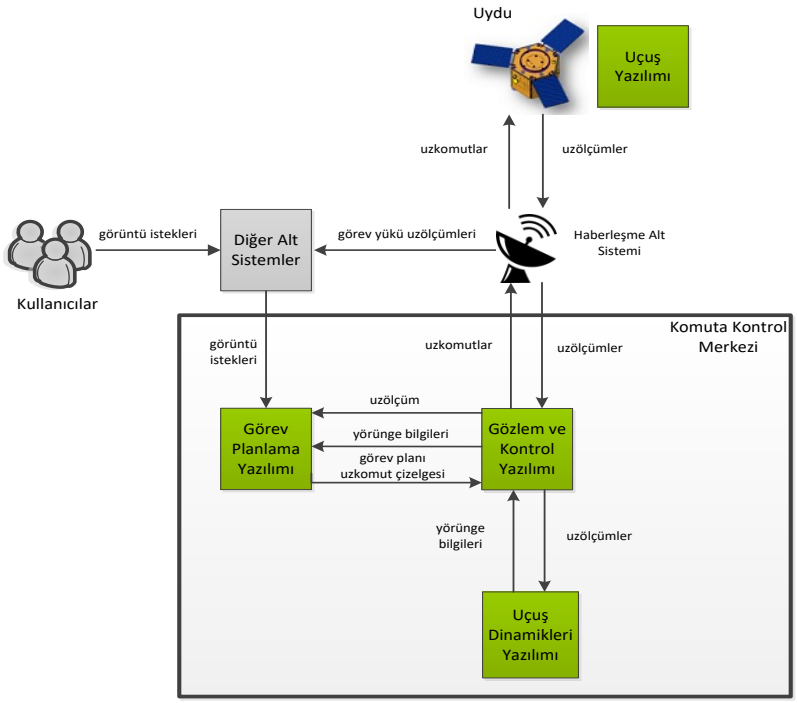
Abstract. In this study, a migration strategy of a satellite ground software system which is being developed by Turkish Aerospace Industries (TAI), in a service-oriented architecture to microservice architecture has been described. Problems in the current architecture and tactical transformations that are necessary in short term for microservice architecture migration have been shared. Satellite Command and Control Software must have high-level reliability, availability and performance quality attributes and how we will achieve these requirements via microservice architectures has been described. Migration strategy has been planned in a staged way, the elements will be completed in each stage have been explained and long-term plans have been mentioned.

Keywords: Service-oriented Architecture, Microservice Architecture, Satellite Ground Software Systems.

1 Giriş

Türk Havacılık ve Uzay Sanayii, uydu üreticisi ve entegratörü olmasından dolayı uydu komuta kontrol yer yazılımlarını geliştirmektedir. Bu yazılımların herhangi bir gözlem uydusunda kullanılabilmesini sağlayacak bir mimari tasarım doğrultusunda geliştirilmesi amaçlanmaktadır. Ayrıca bu yazılımlarda çoklu-uydu yönetimini destekleyecek mimari yapılar tercih edilmektedir. Bu uydu-bağımsız ve çoklu-uydu yönetimi yaklaşımı ile geliştirme, idame, operasyon ve operatör eğitimi maliyetlerinde önemli azalma beklenmektedir [1]. Bu gereksinimlerin yanı sıra, uzay yazılım alanından gelen üst düzey güvenilirlik standartları ve gereksinimleri [2] [3], uygulanacak mimari tasarımın şekillenmesinde önemli rol oynamaktadır.

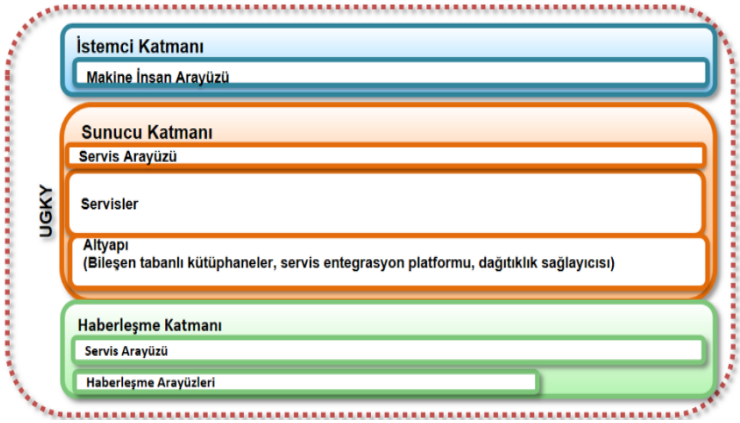
Uydu yönetim sistemlerinin genel yapısı Şekil 1'de gösterilmiştir. Uydu yönetim sistemleri uzay kesimi ve yer kesiminden oluşmaktadır. Uydunun görevini icra edebilmesi, her iki kesimde de yer alan yazılımlar ile mümkün olmaktadır. Uydu kesiminde, görev yüklerinin (Sentetik Açıklıklı Radar, elektro-optik ya da haberleşme görev yükleri gibi) ve uydunun diğer alt-sistemlerinin kontrolü Uçuş Yazılımı aracılığıyla gerçekleştirilmektedir. Uçuş Yazılımı bir görevin icrasına yönelik komutları (uzkomut) yer kesiminden alır ve sensörlerinden topladığı ölçüm bilgilerini (uzölçüm) yer kesimine gönderir. Yer yazılımları Gözlem ve Kontrol Yazılımı (GKY), Görev Planlama Yazılımı (GPY) ve Uçuş Dinamikleri Yazılımı(UDY) olmak üzere üç temel yazılımdan oluşmaktadır. GKY bir uydu görevinde uzkomut ve uzölçümleri kullanarak uydunun gerçek zamanlı olarak gözlem ve kontrol edilebildiği, uydu sağlık verileri takip edilerek problem durumlarının tespit edilebildiği, problemleri gidermek için operatörün uyduya komut gönderebildiği, GPY tarafından uydunun görev icrası için hazırlanan planları uyduya yüklemekten sorumlu yer birimidir. Ayrıca gönderilmiş uzkomutların, uzölçümlerin ve olay/ikaz verilerinin depolanması, prosedürlerin hazırlanması ve çalıştırılması ve uçuş yazılımı yönetimi işlevlerini gerçekleştirir. GPY uydunun etkili bir şekilde kullanılabilmesi için manevra, görüntüleme ve görüntü indirme aktivitelerini planlamaktadır. Yer yazılımlarının son bileşeni UDY ise uydunun konumuyla ve yörünge olaylarıyla ilgili hesaplama işlevlerini yerine getirir. Bu çalışmada, yalnızca Gözlem ve Kontrol yazılımının mimari değerlendirilmesine ve elde edilen bulgularla birlikte ileriye yönelik mimari dönüşüm stratejisine odaklanılmıştır.



Şekil 1 Uydu Yönetim Sistemlerinin Genel Yapısı

2 Mevcut Mimari Tasarıma Genel Bakış

Halen geliştirilmekte olan GKY, servis-yönelimli bir mimariye (Service-oriented Architecture - SOA) sahiptir. Bu sayede yazılıma yeni özellikler servis olarak eklenebilmektedir. GKY kapsamında geliştirilmekte olan ana uygulama monolitik bir yapıdadır. Ayrıca GKY kapsamındaki bazı modüller ayrı uygulamalar olarak geliştirilmekte ve GKY ana uygulamasına entegre edilmektedir. Sağlanan haberleşme alt yapısı ile GKY kendi iç uygulamaları arasında ve harici sistemlerle noktadan noktaya (point-to-point) ve yayınla-abone ol (publisher-subscriber) entegrasyon paternleriyle asenkron bir şekilde mesaj gönderebilmektedir. GKY istemci, sunucu ve haberleşme katmanlarına ayrılmıştır. İstemci katmanı, görsel arayüzler ile makine-insan etkileşimini sağlamaktadır. Sunucu katmanı, iş mantığının fonksiyonel olarak modülarize edildiği servislerden oluşmaktadır. Bu servisler bazı bileşen tabanlı kütüphaneleri ortak olarak yeniden kullanabilmektedirler. Aynı zamanda servisler istemci katmanına senkron veya haberleşme katmanı üzerinden asenkron mesajlar gönderebilmektedirler. İstemci katmanından sunucu katmanına senkron mesajlaşma mevcuttur. Mevcut mimari tasarım Şekil 2'de gösterilmiştir.



Şekil 2 GKY Mimarisi

3 Servis-yönelimli Mimari ve Mikroservis Mimarisi Nedir?

Servis-yönelimli mimari ve mikroservis mimarisi esasen servislerin üzerine kurulu yazılım mimarisi yaklaşımlarıdır. Paylaştığı ortak özellikleri olduğu gibi mikroservis mimarisinin ayrıştığı bazı noktalar bulunmaktadır. Mikroservis mimarisinin en temel ayrışma noktası bağımsız bir şekilde konuşlandırılabilen servislerden oluşmasıdır. Servis-yönelimli mimarilerde ise genel olarak büyük monolitik uygulamalar ve Kurumsal Veri Yolu (Enterprise Service Bus - ESB) üzerine kurulu bir haberleşme katmanı yer almaktadır. ESB üzerindeki haberleşme mekanizmalarında genellikle karmaşık iş mantıklarının işletildikleri görülmektedir. Mikroservis mimarisi, servis-yönelimli mimarinin sadece daha spesifik ve modern bir tarzı olarak düşünülse de sahip olduğu popülarite ve öne çıkan karakteristikleri dolayısıyla ayrı bir vurguyu hak etmektedir. Mikroservis mimarisini servis-yönelimli mimariden ayrıştıran başlıca karakteristikleri için: servisler ile bileşenleştirme, zeki uç noktalar ve aptal iletişim hatları, merkezi olmayan kontrol, merkezi olmayan veri yönetimi, altyapı otomasyonu ve arızalar için tasarım [4] gibi unsurlar bahsedilebilir. Bu ortak karakteristiklerinin belli başlılarına sahip olan mimariler mikroservis mimarisi olarak tanımlanabilir. Bu karakteristiklerin önemli birçoğu, GKY için de önem arz etmektedir. Mikroservis mimarisi ile daha güçlü modül sınırları, bağımsız konuşlandırma ve teknoloji çeşitliliği ile daha hızlı teknik çözüm geliştirme kabiliyeti sağlanabilecektir. Ayrıca mikroservis mimarisi ile daha güvenilir (reliable), arızalara karşı toleranslı (fault-tolerant), dirençli (resilient), ölçeklenebilir, performanslı ve erişebilirliği yüksek (high-available); uydu yer yazılım alanındaki kalite özelliklerini karşılayan bir yazılım sisteminin geliştirilmesi mümkün olabilecektir.

4 SOA'dan Mikroservis Mimarisine Geçiş Stratejisi

SOA'dan mikroservis mimarisine geçiş, aşamalı bir şekilde planlanmaktadır.

- **İlk aşamada;** uydu-bağımsız, servis-yönelimli bir mimaride yazılım sisteminin fonksiyonel gereksinimleri karşılayacak şekilde prototip olarak tamamlanması planlanmaktadır.
- **İkinci aşamada;** mevcut mimarinin güncel yaklaşımlar ve teknolojilerle modernize edilmesi, mikroservislere geçiş stratejisinin zeminini oluşturacak taktiksel dönüşümlerin gerçekleştirilmesi, yeni geliştirilecek birkaç servisin birkaç geliştirici tarafından mikroservis olacak şekilde geliştirilmesi ve bu mikroservislerin mevcut sisteme entegre edilmesi planlanmaktadır.
- **Üçüncü aşamada;** tüm servislerin alan-güdümlü tasarım (domain-driven design) prensiplerine göre ele alınarak, modül kırılımlarının yeniden gözden geçirilmesi ve tüm servislerin mikroservis mimarisinin tüm unsurlarıyla birlikte geçişin tamamlanması planlanmaktadır.
- **Dördüncü aşamada;** yazılımın hedef uydu sistemlerine göre özelleştirilebilmesi kabiliyeti sağlanabilmesi için yazılım ürün-hattı yaklaşımının mikroservis mimarisi üzerinde uygulanması ve yazılım sisteminin ürün ortamına hazır bir hale getirilmesi planlanmaktadır.

Tüm bu dönüşüm süreci bize bir takım bulgular ve çıkarılan dersler edinme fırsatı da sunacaktır. Bu yaklaşım özellikle gereksinimlerin ve arayüzlerin netleştirilmesinde oldukça faydalı olmaktadır. Martin Fowler'ın "Monolith First" [5] ve "Strangler Application" [6] yaklaşımları ve ayrıca literatürdeki mikroservislere geçiş stratejisi hakkındaki çalışmalar da [7] prototip ve monolitik bir şekilde başladığımız bu geliştirme sürecinin, aşamalı bir şekilde mikroservis mimarisine geçilmesi yönündeki stratejimizi doğrular niteliktedir. Bu çalışmada, mikroservis mimarisine geçişi sağlamak için öncelikle mevcut SOA mimarisindeki yazılım sisteminin sorunları belirtilerek, mikroservis mimarisine geçişe zemin hazırlayacak taktiksel dönüşümlerin nasıl olacağının açıklanması ve daha sonra hedeflenen nihai yazılım mimarisinin tanımlanması amaçlanmaktadır.

5 Sorunlar ve Taktiksel Dönüşümler

5.1 Entegrasyon Metodolojisi

GKY'nın GPY, UDY ve diğer başka harici sistemler ve yazılımlar ile tanımlanan harici ara yüzler üzerinden entegrasyonu sağlanmaktadır. Bunun yanı sıra bazı iç modüller de ayrı bir uygulama olarak geliştirilmektedir. Mevcut yapıda sürekli entegrasyonun sadece uygulama bazlı olması yazılım güvenirliği açısından risk oluşturmaktadır. Bundan dolayı, dış arayüzlerin uçtan-uça manuel ve/veya otomatik olarak test edilebileceği, son kullanıcıya demo yapılabileceği bir entegrasyon test ortamına (continues delivery environment) ihtiyaç duyulmaktadır.

Uydu yer yazılım alanındaki literatürdeki ampirik çalışmalar da göstermiştir ki, sık veya sürekli olmayan harici arayüzlerin entegrasyonu ve testleri, ilk entegrasyon teşebbüsünde ve ileriki safhalarda ciddi sorunlara yol açabilmektedir [8]. Ayrıca, entegrasyon test ortamı ve sürekli teslimat hattının kurulmasından sonra bağımlı uygulamaların dış arayüzlerinin süreç içerisinde değişiminin yönetilebilmesi için kontrat-tabanlı

bir entegrasyon test yaklaşımının [9] uygulanması, yazılım entegrasyonun daha problemsiz olmasını sağlayacak ve güvenilir bir yazılım elde edilmesine imkân verecektir.

5.2 Çoklu Haberleşme ve Entegrasyon Paradigmaları

Sunucu katmanında çoğunlukla soket haberleşmesinin kullanılmasının yanı sıra, bazı iç arayüzlerde RESTful servis arayüzleri de kullanılmaktadır. Ayrıca, asenkron mesajlaşma için birden fazla teknoloji aynı anda kullanılmaktadır. Bu çoklu haberleşme ve entegrasyon paradigmalarından kaynaklı sorunlar literatürde de raporlanmıştır [10]. Servis arayüzlerinin idame edilebilirliği ve yönetilebilirliği açısından; teknoloji ve dış bağımlılıklar bakımından zorunlu olan soket haberleşmesi dışında tüm iç ve dış arayüzlerin RESTful servis arayüzleriyle ile sağlanmasını mikroservislere geçişin üçüncü aşamasında planlamaktayız. RESTful servislerinin getireceği ekstra yük, sistemin konuşlanacağı ortamın kaynak büyüklüğü ve sağladığı durumsuzluk (statelessness) ile elde edilecek ölçeklenebilirlik dikkate alındığında bir kaygı olmaktan uzaktır.

5.3 Paylaşılan Bileşenler

Hem SOA'da hem Microservis mimarisinde, sistem ayrı servislere bölünmesine rağmen bazı ortak fonksiyonaliteler servisler arasında paylaşılabilmektedir. Bu ortak fonksiyonlar, kütüphane olarak ya da servis olarak bileşenleştirilerek servisler tarafından yeniden kullanılabilmektedir. Bu durumda paylaşılan bileşendeki bir güncelleme, bu paylaşılan bileşeni kullanan servislerin güncellenmesine ve kapsamlı bir şekilde tüm bu bağımlı servislerin test edilmesine neden olmaktadır. Bir bileşenin paylaşılarak yeniden kullanılmasındaki karar, güncellenme frekansına (frequency of change) ve olası güncellemelerin etki faktörünün (severity of change) analizine göre verilmelidir. Aksi takdirde servisler arasında gereksiz bir bağlaşım ve servislerin düşük bağlılığa sahip olmasına neden olur ki, bu durum yazılımın idame edilebilirliğini ve anlaşılabilirliğini önemli ölçüde azaltmaktadır. Oluşturacağı karmaşıklık ise hataların kodda saklı kalmasına neden olarak yazılımın güvenilirliğine zarar verebilmektedir. Ayrıca bağımlılıkları çok fazla olan servislerin test edilebilirliği de o ölçüde zorlaşmaktadır.

Bir ortak kütüphanenin bileşen olarak tanımlanabilmesi için bağımsız bir şekilde yeni sürümlere yükseltilebilir olması gerekmektedir. Bunun için ortak kütüphanelerin bağımlılıklarının sürüm yükseltmelerine mani olmamasının sağlanması gerekmektir. Mevcut yapıda sayıca çok ve karmaşık yapıdaki bağımlılıkların bulunması ortak bileşenlerin takibinde, hata ayıklamasında ve idamesinde zorluklar oluşturmaktadır.

Ortak kütüphaneleri bileşenleştirme için; mikroservislere dönüşümün ikinci aşamasında, ortak bileşenlerin alan analizi sonucuna göre netleştirilmesi ve mümkün olduğunca sadeleştirilmesi gerekmektedir. Gerekirse kodların tekrar etmesine izin verilmelidir. Daha sonra ortak kütüphanelerin paketlenerek bir sürüm deposunda (artifact repository) yayım ve anlık sürüm şeklinde saklanması sağlanmalıdır. Geliştirici eğer bir değişiklik yapmıyorsa bu bileşenler üzerinde, yayım sürümündeki kararlı olan paylaşılan bileşene bağımlı olmalıdır. Böylelikle paylaşılan bileşenlerdeki kırılganlıktan etkilenmeyecektir. Sürüm alınacağı zaman paylaşılan bileşenlerin de sürüm bilgisinin etiketlenerek dondurulması, sürüm deposunda saklanması ve daha sonra hedef

ortama sürüm deposundan ana uygulamalarla birlikte konuşlandırılması gerekmektedir. Günlük sürekli teslimat hatlarında ise ana uygulama kodu ve paylaşılan kütüphanelerin anlık sürümleri sürüm deposunda saklanmalı, hedef ortama konuşlandırılmalı ve koşulan entegrasyon testlerin sonucu geliştiricilere bildirilmelidir.

Bu taktiksel dönüşümün neticesinde monolitik yapı, mikroservislere bölünmek istendiğinde sınırlandırılmış bağlamlar (bounded-contexts) daha net ve sade olacağından bağımlılık analizini yapmak kolaylaşacak ve mikroservis mimarisine geçiş kolaylaşmış olacaktır.

5.4 Verinin Saklanması

Tüm türdeki verilerin saklanması için sadece tek bir ilişkisel veritabanı kullanılmaktadır. Fakat verinin türüne ve iş alan analizi yapılarak kullanım durumuna göre veritabanı teknolojilerini çeşitlendirmek gerekmektedir. İlk aşamalarda, birkaç servisin doküman-tabanlı bir veritabanına geçilmesi planlanmaktadır. Bu servislerin yıllar içerisinde biriken büyük veri üzerinde hızlı sorgu gereksinimleri vardır. Ayrıca arşivlenen bu verilerin güncellenme sıklığı çok nadirdir. Her bir mikroservisin veritabanı tipi tercihinde bulunurken bunun gibi kriterlerin göz önünde bulundurulması gerekmektedir.

Mikroservisler, servis bazlı müstakil veritabanına sahip olabilmelidir. Fakat bu durum ortak verilerin pek çok serviste çoklanması ve daha fazla entegrasyon maliyetini ortaya çıkmaktadır. Öte yandan verinin mikroservislerce çoklanması, mikroservislerin otonom ve bağımsız bir şekilde işlemlerini yapabilmesine imkân vererek ölçeklenebilir ve performanslı olmasını sağlayacaktır. Mikroservislerin iş alan analizi ve ilgili veri modelinin analizi yapılırken bu durum göz önünde bulundurulmalıdır.

5.5 Konuşlandırma

GKY'de, "12-factor app" [11] metodolojisinin 1. kuralının uygulanması planlanmaktadır. Böylelikle her bir uygulamanın kendi kod deposuna sahip olması sağlanacaktır.

Nihai yapıda, hedef ortamlardaki uygulamaların ve sistemlerin konteynerize edilerek konuşlandırılması planlanmaktadır. İlk aşamalarda ise mümkün olan monolitik yapıdaki uygulamaların konteynerize edilerek konuşlandırılması planlanmaktadır. Mikroservislere geçildiğinde ise her bir servisin konteynerize edilerek bağımsız bir şekilde konuşlandırılması planlanmaktadır. GKY sunucu uygulamasının konteynerize edilebilmesi için istemci uygulamasının ayrıştırılması planlanmaktadır. İleri aşamalarda istemci uygulamasının web uygulamasına dönüştürülmesi planlanmakta ve bu sayede istemci uygulamasını da konteynerize edebilmek mümkün olabilecektir.

5.6 Arızalar için Tasarım ve Ölçeklenebilirlik

Uygulamalar pek çok sebepten dolayı arıza yaparak servis dışı kalabilirler. Bu durumu ele alacak bir tasarımın geliştirilmesi önem arz etmektedir. Mevcut durumda yedekli bir yapı aktif/pasif üstlenme (failover) konfigürasyonunda tasarlanmıştır. Bu konfigürasyonda arıza olduğunda, pasif sunucunun ayağa kalkması ve bağlantıları

yeniden tesis ederek sistemin tekrar erişebilir olması hedeflenmektedir. Erişebilirliği daha da artırmak için aktif/aktif üstlenme konfigürasyonuna geçilmesini planlamaktayız. Bunun için bir yük dengeleyicisinin mimariye eklenmesi planlanmaktadır. Böylelikle aynı tipteki yedek sunucuların ayağa kalkarak yük dengeleyicisinden gelen istekleri karşılayabilecek ve ölçeklenebilir bir yapıya geçilebilecektir. Fakat bunu sağlamak için uygulamalar, arıza durumlarında durum bilgisini yönetebilmeli ve herhangi bir tutarsızlık sergilememelidir. Bundan dolayı "12 factor app" [11] metodolojisinin 6. kuralı ihlal edilerek ölçeklenebilirliğe mâni olan durum yönetimindeki sorunların harici bir önbellek (cache) gibi yapılar kurularak düzeltilmesi planlanmaktadır.

GKY için ölçeklenebilirlik oldukça önemlidir çünkü uygu-bağımsız ve çoklu-uydu yönetimini yapabilecek bir platform geliştirmek amaçlanmaktadır. Ölçeklenebilir bir yapı kurarken, sistemin güvenirliliğini ve erişebilirliğini de artırabilmek mümkün olabilmektedir.

6 Mikroservis Mimarisindeki Gözlem ve Kontrol Yazılımı

Gözlem ve Kontrol Yazılımının mikroservis mimarisine üçüncü aşamanın sonunda tüm unsurlarıyla geçmesi planlanmaktadır. Yeni mimarinin temel unsurları şunlar olacaktır:

- **Konteynerizasyon:** GKY'nın tüm mikroservisleri Linux Konteynerleri içerisinde konuşlandırılacaktır. Konteynerize edilmeyen hiçbir uygulama ve sistem bileşeni bulunmayacaktır. İç ağımızda kurulacak olan sürüm deposunda microservis uygulamalarını içerecek temel konteyner imajları hazır tutulacaktır. Geliştirilen uygulamalar konteynerize edilirken bu imajları temel alacaktır.
- **Otomasyon:** Tüm mimari unsurlar "infrastucture as a code" mantığıyla tek bir komut ile geliştirme, test vb. ortamlara konuşlandırılması ve ayağa kaldırılması yapılacaktır.
- **Sürekli Entegrasyon/Sürekli Teslimat Hattı:** Yazılım kaynak kodunda yapılan her bir değişiklik sonrası yazılım kodları derlenecek, statik kod analizleri yapılacak ve birim testler koşacaktır. Daha sonra veri göçü yapılacak ve entegrasyon birim testleri koşacaktır. Ayrıca tüketici-güdümlü kontrat testler (consumer-driven contract tests) sürekli entegrasyon safhasında koşacaktır. Bu sayede sorunsuz bir şekilde mikroservislerin entegre edilmesi sağlanabilecektir. Sürekli entegrasyon faaliyetlerine ilaveten mikroservisler hedef ortamlara sürekli bir şekilde konuşlandırılacak ve çalıştırılacaktır. Bunun için paketlenen uygulamalar bir konteyner içerisine konularak sürüm deposunda saklanacaktır ve sağlanan otomasyon ile konteynerize edilmiş tüm mikroservisler hedef ortamlara konuşlandırılacaktır. Hedef ortamlardan biri olarak planlanan entegrasyon test ortamında uçtan-uça ve performans testleri koşacaktır.
- **API Kapısı (API Gateway) ve Yük Dengeleyici:** Mikroservislerin birden fazla örneği aynı anda aktif bir şekilde koşabilir, yani ölçeklenebilir olması sağlanacaktır. Performans ve yedeklilik gereksinimlerine göre mikroservis örnek sayısı değişkenlik gösterebilecektir. Farklı türdeki mikroservisler aralarında haberleşirken bir

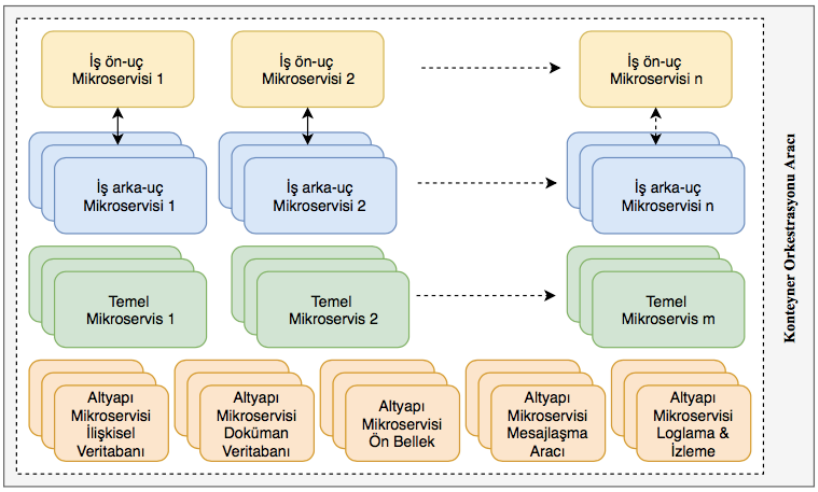
API Kapısı üzerinden mesajları gönderecektir. Mesajlar API kapısının yük dengeleyici özelliği ile en uygun olan mikroservise yönlendirilecektir.

- **Servis Keşfi:** Mikroservislerin birbirleriyle haberleşmesinde adres ve durumlarını öğrenebilmeleri için servis keşfi yapacak mikroservistir. Mikroservisler ayağa kalktığında bu mikroservise kendilerini kaydettireceklerdir.
- **Karşıt Sunucu (Reverse Proxy):** Yazılım sistemindeki tüm uygulamalar ve sistem bileşenleri bir karşıt sunucu arkasında hizmet verir olacaktır. Dış sistemler sadece bu sunucu üzerinden sisteme erişim sağlayacaktır. Bunun yanı sıra, her bir mikroservisin kısmi kullanıcı arayüzlerini birleştirecek bir kullanıcı arayüzü uygulaması ve statik içerikleri de bu karşıt sunucu üzerinde konuşlandırılacaktır. Böylelikle mikroservislerin farklı uygulamalar olarak geliştirilen kullanıcı arayüzleri, dışarıya tek bir kullanıcı arayüzü gibi gösterilebilecektir.
- **Kümelenme:** Birden fazla sanal/fiziksel makineler kullanarak entegrasyon test ortamı ve ürün ortamının kümelenmesi sağlanarak sistemin ölçeklenebilirliği ve güvenilirliği sağlanacaktır.
- **Orkestrasyon:** Konteynerler ile yapılan tüm konuşlandırmalar ve servislerin çalıştırılması ve izlenmesi bir konteyner orkestrasyonu aracıyla yapılacaktır. Mikroservislerin yedekliliği, arıza durumunda geriye alma, sistem kaynak kullanımının yönetilmesi ve otomatik ölçekleme gibi işlemler bu kapsamda yapılacaktır. Bunun yanı sıra mikroservis mimarisinde olması gereken API kapısı, karşıt sunucu, yük dengeleyici, servis keşfi ve kümelenme özellikleri orkestrasyon kapsamında da gerçeklenebilecektir ya da ayrı teknolojilerle gerçekleştirilebilecektir. Bu durum ileride alınacak tasarım kararlarında şekillenecektir.
- **Devre Kesici (Circuit Breaker):** Mikroservislerin erişebilir olmadığı ve stres altında olduğu durumlarda, mikroservislere gelen yeni istekler kesilecek ve uygun durumdaki başka bir mikroservis örneğine API kapısı tarafından istekler yönlendirilecektir [12]. Böylelikle mikroservislerin erişebilirliği artıralacak ve mikroservislerin zincirleme bir şekilde arızalanması engellenecektir.
- **Harici Ön Bellek ve Durumsuzluk:** Mikroservisler bir arıza durumunda durum bilgisini kaybetmemesi için durum bilgilerini uygulama belleğinde tutmak yerine harici bir ön bellekte saklayarak durumsuzluğu sağlayabileceklerdir.
- **Konfigürasyon Yöneticisi:** Mikroservislerin konfigürasyon ayarlarını dağıtık ortamda harici bir şekilde yönetilmesini sağlayacak bir mikroservistir. Mikroservisler devre dışı kalmadan, dinamik olarak konfigürasyon parametrelerinin değerlerinin değiştirilmesi de mümkün olabilecektir.
- **Loglama ve İzleme:** Mikroservis mimarisinde sayıca artan dağıtık uygulamalar ve sistem bileşenlerinden dolayı loglama ve izleme çok daha fazla önem arz etmektedir. Loglama, "12 factor app" [11] metodolojisinin 11. kuralına uygun olarak harici bir log yönetim sistemine olay fırlatılması ile ele alınacaktır. Yazılım üzerindeki koşulacak testlerin ve ürün ortamındaki operasyonel profilin izlenmesi için de bir alt yapı sağlanacaktır. Böylelikle bütün işlemler uçtan-uça izlenebilecektir. Yazılımın kullandığı kaynak kullanım metrikleri ve işlemler ilişkilendirilerek sistem performansı hakkında detaylı analizlerin yapılabilmesi sağlanabilecektir.
- **Aptal İletişim Hatları:** Mikroservis mimarisi, ESB gibi üzerinde pek çok dönüşümün, iş akışının ve kuralların olduğu bir iletişim hattı yerine, oldukça basit

bir şekilde mesajı iletmesinden başka iletişim hattına anlam yüklenmemesini savunur. Zekânın servislerin uç-noktalarında tanımlanması gerektiğini iddia eder. Bunun için ESB yerine sadece mesajları noktadan noktaya ve yayınla-abone ol paternleriyle asenkron bir şekilde gönderebilen güvenilir, ölçeklenebilir ve performanslı bir teknoloji tercih edilecektir.

- **Paylaşılan Bileşenler:** Servis-yönelimli mimari, bileşenlerin mümkün olduğunca paylaşılması ve yeniden kullanılabilmesini savunur. Mikroservis mimarisi ise alan-güdümlü tasarımdan gelen sınırlandırılmış bağlam kavramının öne sürdüğü mümkün olduğunca az paylaşımın yapılması gerektiği üzerine inşa edilmiştir. Kütüphane olarak paylaşılan bileşenler, yeni mimaride oldukça sınırlı düzeyde olacaktır. Ayrıca mikroservis mimarisi, kütüphane olarak bileşenleştirmeden ziyade servis olarak bileşenleştirmeyi savunur. Bu kapsamda bazı ortak özellikler ve yardımcı fonksiyonlar, sınırlı düzeyde kalmak kaydıyla mikroservisler olarak tasarlanacaktır. Kütüphane olarak paylaşılan bileşenlerin kullanılacağı istisnai durumlar daha sonra belirlenecektir. Kütüphane olarak paylaşılan bileşenler sürüm deposunda versiyonlanarak saklanacaktır.
- **Verinin Saklanması:** Mikroservisler işlem yaptığı verinin türüne göre ilişkisel veya doküman-tabanlı veritabanları kullanılabilecektir. Mikroservis-tabanlı müstakil veritabanları yeni mimaride olacaktır.
- **Mikroservisler:** Mikroservisler iş arka-uç, iş ön-uç, temel ve altyapı türlerinde olacaktır.
  - **İş arka-uç mikroservisleri**, RESTful servisleriyle arayüz sağlayacak ve iş mantığını gerçekleştirilecektir. Birden fazla uygulama örnekleri aynı anda ayağa kalkarak ölçeklenebilir bir yapıda olacaktır.
  - **İş ön-uç mikroservisleri**, sadece bağlı olduğu iş arka-uç mikroservisine RESTful istekleri atarak iletişim kuracaktır. Sadece içerdiği kullanıcı arayüzü ile makine-insan etkileşimini sağlayacaktır. Hiçbir iş mantığı içermeyecektir. Güvenlik, oturum ve durum vb. bilgiler için iş arka-uç mikroservisine bağlı olacaktır. İş ön-uç mikroservisi web uygulaması olarak geliştirilecek ve konteyner içerisindeki bir web sunucusu tarafından servis verir olacaktır.
  - **Temel mikroservisler**, mikroservis mimarisinde olması gereken karşıt sunucu, servis keşfi, konfigürasyon yöneticisi gibi temel mimari unsurları ve servis olarak bileşenleştirilen servisleri içerecektir. Birden fazla uygulama örnekleri aynı anda ayağa kalkarak ölçeklenebilir bir yapıda olacaktır.
  - **Altyapı mikroservisleri**, altyapısal sistemlerin konteynerize edilmesini ve kümelenmesini sağlayacaktır. Bu sistemlerin kendi yedeklilik mekanizmaları kullanılarak güvenilirliği ve ölçeklenebilirliği sağlanacaktır.

Yeni mimari Şekil 3'te genel olarak tarif edilmiştir.



Şekil 3 Yeni Mimari Tasarımın Genel Yapısı

7 Sonuç

GKY mimarisini SOA'dan Mikroservis mimarisine dönüştürerek, pek çok uyduya aynı anda hizmet verebilecek şekilde, güvenilir, ölçeklenebilir, dirençli, esnek ve performanslı bir platformun geliştirilmesini hedeflemekteyiz. Bu çalışmada, bu dönüşüm stratejisinin aşamaları ve nihai olarak varmak istediğimiz mimari tasarım paylaşılmıştır. GKY'nın mikroservis mimarisine geçiş sürecinin tamamlanmasından sonra bu süreçte elde edilen tecrübelerin bir çalışma olarak paylaşılması planlanmaktadır. Daha sonra ise aynı anda birden fazla uyduyu gözlem ve kontrol edebilme kabiliyeti için uydu-bazlı uyarlamaların yapılabilmesi imkân tanıyan yazılım ürün hattı yaklaşımının mikroservis mimarisi üzerinde uygulanması hedeflenmektedir. Literatürdeki çalışmalar da referans alınarak [13] [14], yazılım ürün hattını mikroservisler üzerinde uygularken bazı mikroservisleri genel (multi-tenant), bazılarını ise uyduya özel mikroservisler olacak şekilde oluşturulması planlanmaktadır. Klasik yazılım ürün hattı yaklaşımındaki, varyasyon noktalarını değiştirerek ayrı ürünler çıkarmaktan ziyade; tek bir platform üzerinde aynı ürünün varyasyonlarıyla birlikte konuşlandığı ve birlikte çalıştığı bir yazılım sistemini geliştirmeyi hedefliyoruz. Bu dönüşümde elde edilen tecrübeleri de bir sonraki çalışmamızda paylaşıyor olacağız.

Referanslar

1. Sürme, M., Çakmak, S., Erdoğan, G., Karakaya, E.: Uydu Yer Yazılımlarının Uydu Bağımsız Tasarlanması. In: 12th Ulusal Havacılık ve Uzay Konferansı - UHUK, Samsun (2018)
2. European Cooperation for Space Standardization, ECSS-E-ST-40C Space Engineering – Software (2009)

3. European Cooperation for Space Standardization, ECSS-Q-ST-80C Software product assurance (2009)
4. Martin Fowler – Microservices, https://martinfowler.com/articles/microservices.html, Son erişim 20/09/2018
5. Martin Fowler – MonolithFirst, https://martinfowler.com/bliki/MonolithFirst.html, Son erişim 20/09/2018
6. Martin Fowler – Strangler, https://www.martinfowler.com/bliki/StranglerApplication.html, Son erişim 20/09/2018
7. Carrasco, A., Bladel, B., Semeyer, S.: Migrating towards Microservices: Migration and Architecture Smells. In: Proceedings of the 2nd International Workshop on Refactoring (IwoR '18), France (2018)
8. Ciocirlan, C.: Micro-Services in Ground Control Center: Lessons Learned. In: SpaceOps Conferences, France (2018)
9. Martin Fowler – Consumer-Driven Contracts: A Service Evolution Pattern, https://martinfowler.com/articles/consumerDrivenContracts.html, Son erişim 20/09/2018
10. Dragoni, N., Dustdar, S., Larsen S.T., Mazzara, M.: arXiv:1704.04173 [cs.SE] (2017)
11. The Twelve Factor App, https://12factor.net, Son erişim 20/09/2018
12. Martin Fowler – Circuite Breaker, https://martinfowler.com/bliki/CircuitBreaker.html, Son erişim 20/09/2018
13. Furda, A., Fidge, C., Zimmermann, O., Kelly, W., Barros, A.: Migrating Enterprise Legacy Source Code to Microservices, IEEE Software (2018)
14. Tizzei, L.P., Nery, M.: Using Microservices and Software Product Line Engineering to Support Reuse of Evolving Multi-tenant SaaS, In Proceedings of SPLC, Spain (2017)