SİNEMA SEANS BİLGİ VE REZERVASYON SİSTEMİNİN MİKRO SERVİS YAKLAŞIMIYLA GELİŞTİRİLMESİ

DEVELOPMENT OF A CINEMA MOVIE TIMES AND RESERVATION SYSTEM THROUGH MICRO SERVICES APPROACH

Tolga KAPROL[1]
1. TRnet Bilgi ve İletişim Hizmetleri A.Ş.

Özetçe— Web tabanlı uygulamalar internet kullanıcı sayısının ve kullanımının sürekli olarak artmasıyla beraber giderek önem kazanmaktadırlar. Pek çok masaüstü uygulama artık bulut tabanlı olarak çalışmaya başlamış, geçmişte tek başına kapalı devre çalışan yazılımlar günümüzde birbirleri ile entegre olan, sürekli yeni teknolojilere adapte olmaya çalışan ve gerektiğinde milyonlarca kullanıcıya aynı anda cevap verebilen yapılara dönüşmüştür. İnternet devlerinin günümüzün ihtiyaçlarına uygun yazılım geliştirme metodolojisi de değişmiş, mikro servis yaklaşımı önem kazanmıştır. Bu çalışmada, Sinema Seans Bilgi ve Rezervasyon Sistemi (SSBRS), mikro servis yaklaşımı ile oluşturulan bir web tabanlı uygulama modern geliştirme teknikleri ile gerçekleştirilmiştir. Mikro servis yaklaşımı ile geliştirilen bu uygulamanın her bir alt modülü ihtiyaç gerektiğinde çoğaltılarak sistem kaynakları efektif şekilde kullanılırken yük altında olan alt bileşenler cevap verebilir aralıkta çalıştırılabilmektedir. Sonuç olarak, sistem yönetici kontrolünde ya da otomatik olarak daha az kaynak tüketerek ölçeklenebilmekte ve servis kalitesinin her yoğunlukta eşit sunulabilmesi sağlanmaktadır.

Anahtar Kelimeler—Mikro servis, ölçekleme, devops, bilgisayar mimarisi, güvenlik, servis amaçlı mimari

Abstract—Web-based applications are becoming increasingly important as the number and usage of Internet users continues to increase. Many desktop applications have now begun to work on a cloud-based basis, and in the past closed-loop software has transformed into something that is integrated with each other today, constantly adapting to new technologies and responding to millions of users when necessary. The software development methodology of the internet giants has changed according to the needs of today, micro service approach has gained importance. In this study, Cinema Movie Times Information and Reservation System (CMTIRS) was realised with modern development techniques of a web based application created with micro service approach. In Developed with a micro service approach, each submodule of this application is replicated when needed and the sub-components under load can be operated in the responsive range while system resources are used effectively. As a result, the system can be scaled by administrator control or by automatically consuming less resources, and the quality of service can be presented equally at each density.

Index Terms—Micro services, scability, DevOps, Computer architecture, Security, Service-oriented architecture

I. GİRİŞ

Mikro servis yaklaşımıyla geliştirilmiş yazılımlar Amazon [1], LinkedIn[2], Netflix [3] gibi küresel oyunculardan olduğu kadar Türkiye'de de Gitti Gidiyor [4] gibi büyük çapta trafik karşılayan İnternet portallarında kullanılmaktadır.

Bu geliştirme metodolojisi büyük, karmaşık ve yatay olarak büyütülebilir uygulamaları küçük, bağımsız ve karşılıklı haberleşme içinde bulunan geliştirme dilinden bağımsız programlama arayüzleri ile birbirine bağlamayı hedeflemektedir.

Mikro servis yaklaşımı ve bu yaklaşımla geliştirilen uygulamaların konteyner içinde çalıştırılması bulut bilişim açısından devrimsel bir gelişmedir. Mikro servisler Fowler ve Lewis'e (2014) göre bir uygulamanın her biri bağımsız ve hafif mekanizmalara sahip olacak küçük servisler grubu yaklaşımına sahip sistem mimarisidir. Mikro servis'in bugünkü anlamına en yakın çalışmalardan biri ise George (2012) Micro Services Architecture çalışmasında bahsedilmiş olan servis odaklı mimarilerin uygulanmasında yeni paradigmalar kullanılarak yenilikçi bir vizyonla geliştirilmesi konusudur.

Özellikle konteyner bazlı sanallaştırma yaklaşımları hipervizörlere yüksek performanslı bir alternatif oluşturmuştur [5]. Docker, konteyner temelli olan sanallaştırma yaklaşımlarından birini sunan çözümdür. Yapılan performans araştırmalarında bu çözümün işlemci, bellek ve I/O üzerinde çok az etkide bulunduğu [6] tespit edilmiştir. Bu yüzden de Docker kendisini "hafif sanallaştırma platformu" olarak tanıtmaktadır [7].

Mikro servis mimarisinin dağıtık yapısı, yönetilecek sistem sayısını arttırmaktadır. Alışılagelmiş sistem yönetimi pratikleriyle bu durum sistem yönetimi açısından durumu yönetilemez bir duruma gelebilmektedir. Bu sebeple sistem yönetimi de bu kavramlarla birlikte değişmekte, sistem yönetimleri de konteyner ve ilgili orkestrasyon araçlarıyla birlikte otonom hale gelmektedir [8].

Diğer bir taraftan da nesnelerin interneti kavramıyla birlikte günümüzde çok sayıda cihazdan veri akışı artmış ve mikro servis mimarisi gibi dağıtık mimarilere geçişi zorunlu kılmıştır [9].

II. GERÇEKLENEN MİKRO SERVİS UYGULAMASININ GENEL TASARIMI

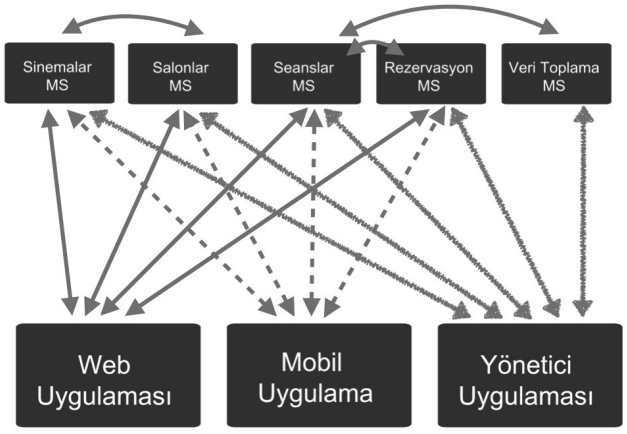
Bu çalışma kapsamında gerçeklenen SSBRS uygulaması; sinema salonları, bu salonlardaki film seansları, gösterim tarihleri ve film detaylarının paylaşıldığı aynı zamanda seanslarla ilgili rezervasyon yapılabilen bir mikro servis mimarisine sahip bir web uygulamasıdır.

SSBRS, Python 3 ile geliştirilmiş veritabanı olarak MongoDB kullanılmıştır. Uygulamadaki her bir iş bileşeni ayrı mikro servis olarak planlanmış diğer servislerden bağımsız olarak çalışabilmesi sağlanmıştır.

Mikro servisler arası haberleşme isteklerinin tasarımı uygulama geliştirilmeden önce Swagger 2.0 YAML formatında tasarlanmıştır.

Swagger formatında HTTP REST haberleşme yönetimini gerçekleştiren tüm gerekli bileşenler formatta belirtilen koşullarla tanımlanabilmektedir. Swagger'da belirtilen parametreler uçbirimin uç adresinin URL formatında ifadesi, REST servisini çalıştığı baz adres gibi temel teknik kriterlerin yanı sıra REST üzerindeki her bir HTTP isteğinin parametreleri, her isteğin kabul ettiği ve cevap olarak döndüğü veri yapıları gibi tüm teknik detaylar bu dosyada tanım olarak yer almaktadır.

Ayrıca her bir isteğin dokümantasyonu, REST uygulamasının adı, lisansı gibi kapsayıcı bilgiler de dosya içerisinde yer almaktadır.



1. SSBRS Mikro Servisleri ve Uygulamalarının Mimarisi

Tüm mikro servisler önünde son kullanıcı için web uygulaması çalışacaktır. Bu uygulama ilgili mikro servislerden gerekli sorgulamaları yaparak ön yüzde kullanıcılara sunacaktır. Bu açıdan mikro servis uygulaması n-katmanlı bir mimari özelliği göstermektedir. Mikro servisleri farklılaştıran en tipik özelliklerinden biri orta katmandaki mikro servislerin tek bir bütün olarak değil ayrı ayrı olarak tanımlanmasıdır.

III. UYGULAMANIN ÇALIŞTIRILMASI

Python programlama dili kullanılarak geliştirilmiş mikro servisleri basit HTTP web servisleri olarak çalıştırılabilmektedir. Uygulama için minimalistik bir Linux dağıtımı olan Alpine Linux[39] kullanılmıştır. Resmi Docker imajı 5 MB olan [35] Alpine imajı kullanarak uygulamanın kurulum, başlatılma ve disk üzerinde kapladığı alanın azaltılması sağlanmıştır.

Uygulama geliştirme aşamasında da geliştirme ortamı olarak bu imajlar üzerinde çalışan Python 3 yorumlayıcısı kullanılmıştır.

NoSQL veritabanı için resmi MongoDB Docker imajı [41] kullanılmıştır.

Uygulamanın geneli beş mikro servis, bir web uygulama ve bir veritabanı sunucusu olarak toplam yedi ayrı konteyner olarak çalışmaktadır.

Web uygulaması 80 portundan, mikro servisler ise 8088 ile 8091 arasındaki portlardan ağ içinde haberleşmektedirler.

Mongo DB portu olan 27017 de monitör edilme kolaylığı açısından dışarıya açılmıştır.

II. SSBRS MİKRO SERVİSLERİ VE KAPI NUMARALARI

Servis Adı	*İç Kapı Numaraları*	*Dış Kapı Numaraları*
Web Uygulaması	80	80
Sinema MS	8088	Dışarıya Kapalı
Salon MS	8089	Dışarıya Kapalı
Veri Toplama MS	8090	Dışarıya Kapalı
Seans MS	8091	Dışarıya Kapalı
Veritabanı	27017	27017

Uygulama Docker [42] ve Docker-compose [43] kurulu herhangi bir linux kutusunda Docker-compose up -d komutu ile çalıştırılabilmektedir. Karmaşık ve bir çok adımdan oluşan kurulum yönergeleri her seferinde yeniden tekrarlanabilen ve ev sahibi sistemin kurulumundan bağımsız olarak çalışabilecek duruma getirilebilmektedir. Bu durum uygulamanın taşınabilirliği konusunda sistem yöneticilerine önemli bir avantaj getirmektedir.

IV. BULGULAR VE PERFORMANS TESTİ

SSBRS'in geliştirilmesinde mikro servisin HTTP REST haberleşme metodu üzerinde cevaplayacağı alt işlemlerin önceden Swagger formatında tasarlanması geliştirici için bazı kolaylıklar sağlamıştır.

Uygulama geliştirilme aşamasından önce hazırlanan Swagger.yaml dosyası sayesinde servisin dokümantasyonu geliştirme öncesinde tamamlanmıştır.

Swagger dosyalarının REST mimarilere sahip birçok projede yer aldığı tespit edilmiştir. Swagger dosyası yorumlayıcı araçlar sayesinde uygulama geliştiriciler REST istek ve cevapları için aşina oldukları bir arayüz aracına Swagger dosyasını içe aktararak tüm özelliklere ait dokümanlara eksiksiz şekilde ulaşabilmekte ve tüm fonksiyonlara bu uygulamalar üzerinden kolaylıkla sorgu gönderebilmektedirler. Bu durum geliştiricilerin REST uygulamanın çalışmasını gerçek ortamda deneyimlemesine olanak sağlayıp geliştirme süresini kısaltabilmektedir.

Yapılan araştırmalarda Swagger formatından bir çok programlama diline sunucu ve istemci formatında çevirici uygulamalar olduğu tespit edilmiştir. Bu araçlarla dokümantasyona tam uyumlu prototip uygulamalar oluşturulabilmektedir.

Otomatik oluşturulan prototip uygulamalar uygulamanın test işlemleri için altyapı oluşturmaktadır. Swagger formatındaki uygulamalarda Test Driven Development odaklı geliştirme çalışmalarının kolaylıkla yapılabileceği tespit edilmiştir.

Mikro servis mimarisi uygulamanın bileşenlerini net bir şekilde birbirinden ayırmıştır. Uygulamanın cevap vereceği istekler ve cevap formatı öneceden belirlendiği için uygulamanın oldukça iyi tanımlanmış olduğu söylenebilir. Uygulama geliştirme sürecine her hangi bir geliştiricinin dahil olması oldukça kolaylaşmıştır. Bu durum literatürde de sıklıkla bahsedilen bir olgudur.

İhtiyaç duyulduğunda ilgili herhangi bir mikro servisin yeni baştan yazılması, yeni bir programlama diline aktarılması mümkündür.

Mikro servisler HTTP üzerinden haberleştiği için uygulama başarımı için ağ mimarisinin erişilebilirliğinin önemli bir kriter olduğu görülmüştür. Ayrıca çok fazla mikro servisin bir araya geldiği işlemlerde haberleşme monolitik bir uygulamaya göre daha yavaş olabilmektedir. Bu durumda uygulamanın ihtiyaçlarının mikro servis olup olmadığı gerçeklenecek uygulamaya göre iyi irdelenmelidir.

Uygulamanın minimalistik bir yapıda hazırlanmış olması sistemin hızlıca geliştirme ortamında çalıştırılabilmesine olanak sağlamıştır. Geliştirme ortamında yapılan geliştirmelerin bir dakikadan kısa bir sürede yeni bir Docker imajı oluşturularak çalıştırılması sağlanmıştır.

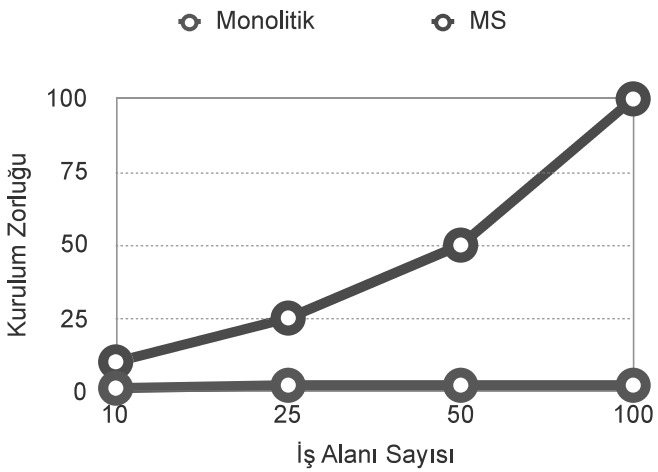
Geliştirme ortamında kullanılan sistem konfigürasyonu test ve gerçek ortamda da kullanılabilmiştir. Böylece uygulama geliştirme ortamında verdiği tepkiler, test ve gerçek ortamda da korunabilmiştir.

Geliştirme, test ve gerçek ortamda yayına alma işlemlerinin birer komuta indirgenebildiği görülmüştür, böylece bu yaklaşımın sürekli birleştirme ve sürekli yayınlama operasyonları için de kolaylık sağladığı tespit edilmiştir.

Uygulamaların yayına alınmasının otomasyonsuz zorlaştığı görülmüştür. Monolitik uygulamalarda bir uygulamanın yayına alınması yeterli iken, uygulamanın karmaşıklığına göre kurulumun zorlaştığı tespit edilmiştir. Bu zorluk oranı aşağıdaki matematiksel ifade ile gösterilebilir:

$$KZ = MSS / MUS$$

2. Mikro Servis Sayısının Kurulum Zorluğunu Arttırması



V. MİKRO SERVİS PERFORMANS TESTİ

SSBRS'deki her bir mikro servis fonksiyonel testi tasarlanan sistemin kalitesini ölçek için gerçekleştirilmiştir. Performans testi sonuçları testin gerçeklendiği sisteme, sistemin kullanım alışkanlıklarına göre değişebilmektedir.

Uygulamanın performansını ölçümleyebilmek için sistemdeki kompleks senaryolardan biri olan salon arama fonksiyonu kullanılmıştır.

Performans testi Amazon EC2 Container Service, Amazon SNS/SQS ve NoSQL veritabanı kullanılarak gerçeklenmiştir.

I. TEST İÇİN KULLANILAN ARAÇLAR VE AÇIKLAMALARI

Aracın Adı	*Açıklaması*
Apache JMeter	Java tabanlı fonksiyonel davranışları test için kullanılan ve ölçümleme yapan uygulama
influxDb	Açık Kaynaklı Zaman Serisi Veritabanı
Grafana	Açık Kaynaklı Metrik Analiz ve Görselleştirme Aracı
Cloud Watch	Amazon Web Servis Üzerinde Çalışan Servisler için Oluşturulmuş İzleme Servisi
Apache JMeter	Java tabanlı fonksiyonel davranışları test için kullanılan ve ölçümleme yapan uygulama
influxDb	Açık Kaynaklı Zaman Serisi Veritabanı

Performans testi çok farklı alanlardaki parametreleri ölçüm yapmaya olanak sağlamaktadır. Bu testler, yük testi ve stres testi gibi isimler altında ifade edilir.

VI. SONUÇ

Bu çalışma kapsamında literatürde yer alan mikro servis yaklaşımları incelenmiş ve SSBRS geliştirilmiştir. SSBRS'te her bir farklı işlev farklı mikro servisler tarafından işlenebilecek şekilde uygulama geliştirilmesi tamamlanmıştır.

SSBRS geliştirmesi mikro servis geliştirmesinin monolitik geliştirmeye göre olumlu ve olumsuz yanlarını öne çıkarmıştır. Çözülecek sorunun mantıksal parçalarının da farklı alt yazılım projeleri olarak soyutlanmış olması yazılım geliştiricilerin problemi düşünmeleri ve üzerine çözüm getirebilmeleri yönünde olumlu etkileri olduğu tespit edilmiştir. Böylece yazılım geliştiricilerin süregiden bir mikro servis yazılım projesine dahil olmalarının daha kolaylaştığı öngörülebilmektedir.

Diğer yandan mikro servislerin kurulum ve yönetim için getirdiği kolaylıklar ve ekstra yükler bulunduğu gözlemlenmiştir. Ancak mikro servis mimarisine bu alanda avantajlar getiren konteyner teknolojileriyle birlikte popülerlik kazandığı literatür araştırmaları sonucunda tespit edilmiştir. Bu paralelliğin sebeplerinden biri konteyner teknolojilerinin devrimsel nitelikteki çözümlerinin sistem yönetimi ve kurulum aşamalarına getirdiği kolaylıklardır.

SSBRS'in, mikro servis mimarisinin yanı sıra konteyner teknolojisi içinde çalışıyor olması, yazılım geliştirme sürecinden kurulum, yönetim gibi aşamalarda alışılagelmiş çözümlere göre hissedilir kolaylık getirmiştir.

Mikro servis yaklaşımı ile geliştirilen uygulamaların kolaylıkla ölçeklenebildiği de literatür araştırmaları sonucu gözlemlenmiştir. Ölçekleme konusunda da literatürde önemli çalışmalar olduğu görülmüştür. Konteyner ve konteyner orkestrasyon araçları uygulamaları otomatik olarak ölçeklendirme konusunda önemli mesafe katetmiştir.

Literatür araştırmasında da mikro servis uygulamalarının büyük çaplı internet projelerinde hızla büyüdüğü bununla paralel otomatik ölçeklendirme ve konteyner gibi kavramların da beraberinde büyüdüğü ortaya çıkmıştır. Ülkemizde de anlık binlerce işlemin gerçekleştiği özellikle elektronik ticaret sistemlerinde de bu yaklaşımın önem kazandığı ve hızla monolitik uygulamalardan mikro servislere geçiş yapıldığı görülmüştür [44].

KAYNAKLAR

1. Internet: Fulton III, S. (2015), "What led amazon to its own microservices architecture." URL: http://www.webcitation.org/query?url=https%3A%2F%2Fthenewstack.io%2Fled-amazon-microservices-architecture%2F&date=2017-07-09 Son Erişim Tarihi: 09.07.2017.
2. Gucer, V. and Narain, S. (2015). Creating Applications in Bluemix Using the Microservices Approach, ABD: IBM Redbooks.
3. İnternet: Netflix (2017). Netflix Conductor. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fgithub.com%2FNetflix%2Fconductor&date=2017-07-09 , Son Erişim Tarihi 09.07.2017
4. İnternet: Dayıbaşı, O. (2017), Gitti Gidiyor' da DevOps Süreçleri Nasıl İşletiliyor? URL: http://www.webcitation.org/query?url=https%3A%2F%2Fmedium.com%2Fcloud-and-servers%2Fgitti-gidiyor-da-devops-s%25C3%25BCre%25C3%25A7leri-nas%25C4%25B1l-i%25CC%2587%25C5%259Fletiliyor-ebee292a7545&date=2017-07-09 , Son Erişim Tarihi: 09.07.2017
5. İnternet: M. Fowler and J. Lewis. (2014) Microservices, hhttp://www.webcitation.org/query?url=http%3A%2F%2Fmartinfowler.com%2Farticles%2Fmicroservices.html&date=2017-07-09 , Son Erişim Tarihi 09.07.2017
6. İnternet: Docker Inc., Docker (2017) http://www.webcitation.org/query?url=https%3A%2F%2Fdocker.com+&date=2017-07-09 Son Erişim Tarihi: 09.07.2017
7. Villari, M., Fazio, M., (2016, December), Osmotic Computing: A New Paradigm for Edge/Cloud Integration, IEEE Cloud Computing, pp. 76-83.
8. Stubbs, J, Moreira, W. and Dooley, R.. (2015) Distributed Systems of Microservices Using Docker and Serfnode. Science Gateways (IWSG), 2015 7th International Workshop on. IEEE
9. Newman, S. (2015) Building Microservices. ABD: O'Reilly and Associates.
10. Felter, W., Ferreira, A., Rajamony, R., and Rubio, J., (2014) An Updated Performance Comparison of Virtual Machines and Linux Containers, ABD: IBM Research Division, Austin Research Laboratory, Tech. Rep.
11. Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014) Cloud Computing Patterns, ABD: Springer
12. Clark, J. K., (2016), Microservices, SOA, and APIs: Friends or Enemies?, ABD: IBM developerWorks
13. Casap, N. J . R., (2016) Microservices 101, Bolivia: Jalasoft
14. İnternet: Eventuate. (2016) Event-driven microservices., http://www.webcitation.org/query?url=http%3A%2F%2Feventuate.io%2Fwhyeventdriven.html&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
15. Clark, J. K., (2015), Microservices in Integration Architecture, MuCon 2015, Sy 14
16. Issarny, V., Georgantas and N., Hachem (2011), Service-oriented Middleware for the Future Internet: State of the Art and Research Directions, Brezilya: Brazilian Computer Society
17. Sun Y., (2013), A Low-Delay, Lightweight Publish/Subscribe Architecture for Delay-Sensitive IOT Services, 2013 IEEE 20th International Conference on Web Services
18. Lamport, L., (1978) The Implementation of Reliable Distributed Multiprocess Systems, ABD: Massachusets Computer Associates
19. Tilkov, S. (2012). Breaking the Monolith, QCon London 2012
20. Dragoni, N. (2017), Microservices: Yesterday, Today, and Tomorrow, ABD: Cornell University
21. Li, H., (2006), CiteSeerx: an architecture and web service design for an academic document search engine. Proceedings of the 15th international conference on World Wide Web. ACM
22. Villamizar, M, (2015) Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud. Computing Colombian Conference (10CCC), 2015 10th. IEEE.
23. Wolff, E. (2016) Microservices: Flexible Software Architecture. Addison-Wesley Professional
24. Zaymus, M. (2017) Decomposition of Monolithic Web Application to Microservices.
25. Boswarthick, D. , Elloumi, O. and Hersent, O, (2012) M2M communications: a systems approach. ABD: John Wiley & Sons
26. Kecskemeti, G, Marosi, A, and Kertesz A. (2016) The ENTICE Approach to Decompose Monolithic Services into Microservices. High Performance Computing & Simulation (HPCS), 2016 International Conference on IEEE.
27. Ramparany, F., (2014) Handling Smart Environment Devices, Data and Services at the Semantic Level with the FI-WARE Core Platform. Big Data, 2014 IEEE International Conference on. IEEE.
28. Martin, R C. (2002) Agile Software Development: Principles, Patterns, and Practices. ABD: Prentice Hall.
29. Wampler, D. (2007) Aspect-oriented Design Principles: Lessons from Object-oriented Design, Proceedings of the 2007 AOSD conference.
30. Sametinger, J. (1997) Software Engineering with Reusable Components. ABD: Springer Science & Business Media.
31. Nelson, B. J.(1981) Remote Procedure Call. ABD: Carnegie-Mellon University Department of Computing Sciences.
32. McCamant, S. and Ernst, M. (2003) Predicting Problems Caused by Component Upgrades. Vol. 28. No. 5.
33. Kephart, J. O. and Chess, D. (2003) The Vision of Autonomic Computing. Computer 36.1 41-50.
34. Varanasi, B. and Belida S. (2015) Documenting Rest Services. ABD: Spring REST. Apress, 91-104.
35. Balalaie, A., Heydarnoori, A. and Jamshidi P.(2015) Microservices Migration Patterns. Sharif University of Technology, Iran
36. Vresk, T. and Čavrak, I. (2016) Architecture of an Interoperable IoT Platform Based on Microservices. Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on. IEEE.
37. Daya, S. (2016) Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach. ABD: IBM Redbooks,

38. İnternet: Alpine Linux Development Team (2017) Alpine Linux, URL: http://www.webcitation.org/query?url=https%3A%2F%2Falpinelinux.org&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
39. İnternet: Docker Inc (2017) Alpine Linux Docker Image, URL: http://www.webcitation.org/query?url=https%3A%2F%2Falpinelinux.org&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
40. İnternet: Docker Inc (2017), Alpine Linux Python 3 Docker Image, URL: http://www.webcitation.org/query?url=https%3A%2F%2Fhub.docker.com%2F_%2Fpython%2F&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
41. İnternet: Docker Inc (2017), MongoDB Docker Image, URL: http://www.webcitation.org/query?url=https%3A%2F%2Fhub.docker.com%2F_%2Fmongo%2F&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
42. İnternet: Docker Inc. (2017) Install Docker. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fdocs.docker.com%2Fengine%2Finstallation%2F&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
43. İnternet: Docker Inc. (2017) Docker Compose. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fdocs.docker.com%2Fcompose%2F&date=2017-07-09, Son Erişim Tarihi: 09.07.2017
44. İnternet: Dayıbaşı, O. (2017), Gitti Gidiyor' da DevOps Süreçleri Nasıl İşletiliyor? URL: http://www.webcitation.org/query?url=https%3A%2F%2Fmedium.com%2Fcloud-and-servers%2Fgitti-gidiyor-da-devops-s%25C3%25BCre%25C3%25A7leri-nas%25C4%25B1l-i%25CC%2587%25C5%259Fletiliyor-ebee292a7545&date=2017-07-09 , Son Erişim Tarihi: 09.07.2017