Mühendislik Bilimleri ve Tasarım Dergisi
9(2), 683 – 700, 2021
e-ISSN: 1308-6693

Journal of Engineering Sciences and Design
DOI: 10.21923/jesd.929649

Araştırma Makalesi Research Article

BİLİŞİM ALTYAPISI ÜZERİNE SUNUCUSUZ MİMARİ PLATFORMU İNŞA ETME

Mete KÖSE[1*], Ecir Uğur KÜÇÜKSİLLE[2]

[1] Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği ABD, Isparta, Türkiye
[2] Süleyman Demirel Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Isparta, Türkiye

Anahtar Kelimeler
Sunucusuz Mimari, Bulut Bilişim, Sunucusuz Fonksiyon, Sunucusuz Bilişim.

Öz

Sunucusuz Mimari, kullanıcının sadece uygulamasını geliştirdiği, diğer tüm katmanların bulut bilişim ya da altyapı sağlayıcıları tarafından sunulduğu kavram olarak adlandırılmaktadır. Bu çalışmada yerinde bir veri merkezi, altyapısı bulunan kurum ve kuruluşların, hiçbir bulut sağlayıcısı kullanmadan mevcut altyapısı üzerinde örnek bir sunucusuz mimari platformunu nasıl kurulabileceği üzerinde durulmuştur. Tasarım öncesinde, sunucusuz bilişim hizmeti veren popüler bulut sağlayıcıları araştırılmıştır. Bu sağlayıcılardan en çok kullanılan ilk iki sağlayıcı olan Amazon Web Services (AWS) ve Microsoft Azure platformlarının sunucusuz mimari platformları üzerinde temel ve benzer kod içerikleri ile fonksiyonlar oluşturularak performans testleri yapılmıştır. Yapılan tasarım; bilişim katmanı, stabilite ve ölçeklendirme katmanı, ölçme ve izleme katmanı olarak üç ayrı katmana ayrılmış ve temel olarak bir mimari elde edilmiştir. Bu tasarım üzerinde popüler bulut sağlayıcıların sunucusuz bilişim platformlarında yapılan performans testlerine benzer testler yapılmıştır. AWS ve Azure üzerindeki testlerle, yapılan tasarımın performans testleri karşılaştırılmış ve üç platform üzerine koyulan, aynı görevdeki üç fonksiyonun oturum sayıları, ortalama cevap süreleri, maksimum cevap süreleri grafiklerle gösterilerek analiz edilmiştir. Sonuçta; yapılan tasarımın iyileştirme ve geliştirme noktaları, popüler bulut sağlayıcılarından geri kalan noktaları, avantajları ve dezavantajları ortaya çıkarılmış ve yerinde bir altyapısı olan kurum ve kuruluşların sunucusuz mimari ihtiyaçları için kullanımı mümkün bir tasarım elde edilmiştir.

BUILDING A SERVERLESS ARCHITECTURE PLATFORM ON INFORMATION TECHNOLOGY INFRASTRUCTURE

Keywords
Serverless Architecture, Cloud Computing, Serverless Function, Serverless Computing.

Abstract

Serverless Architecture is called the concept in which the user only develops the application and all other layers are presented by cloud computing providers or other infrastructure providers. In this study, an exemplary serverless architecture platform, which has an on-premise data center and infrastructure, can be designed without using any cloud provider, is mentioned. Prior to design, popular cloud providers providing serverless computing services were researched and performance tests were performed by creating basic and similar code contents and functions on serverless architectural platforms of Amazon Web Services (AWS) and Microsoft Azure platforms, which are the first two most used providers. The design made was detailed with three layers: the computing layer, the stability and scaling layer, the observabilty layer, and as a result, an architecture was essentially obtained. Performance tests were performed on this design, similar to the tests performed by popular cloud providers on serverless IT platforms. The performance tests of the designed architecture were compared with the tests on AWS and Azure. The session numbers, average response times and maximum response times of the three functions in the same function were analyzed by graphs. In line with the conclusion, the points of improvement and development of the designed architecture, its advantages and disadvantages were mentioned. A serverless

[*] İlgili yazar / Corresponding author: metekose@metekose.com, +90-246-211-1375

architectural design, which can be used for the serverless architectural needs of institutions and organizations with an on-site infrastructure, was obtained.

Alıntı / Cite

Köse, M., Küçüksille, E.U., (2021). Bilişim Altyapısının Üzerine Sunucusuz Mimari İnşa Etme, Mühendislik Bilimleri ve Tasarım Dergisi, 9(2), 683-700.

Yazar Kimliği / Author ID (ORCID Number)	Makale Süreci / Article Process	
M. Köse, 0000-0002-1917-8664	**Başvuru Tarihi / Submission Date**	29.04.2021
E. U. Küçüksille, 0000-0002-3293-9878	**Revizyon Tarihi / Revision Date**	27.05.2021
	Kabul Tarihi / Accepted Date	30.05.2021
	Yayım Tarihi / Published Date	20.06.2021

1. Giriş (Introduction)

Bilişim altyapılarına ait katmanların teknolojileri, yıldan yıla çok hızlı ve çeşitli yeni teknolojilerle dallanarak gelişimine devam etmektedir. Bugün hemen hemen tüm kurum ve kuruluşların; internete açık web uygulamaları, satışlarını doğrudan kişilerle buluşturan kurumların da online olarak satış yapabilen platformları, mobil uygulamaları ve ürünlerin değerlendirildiği forumları vb. bulunmaktadır.

Bilişim teknoloji altyapılarındaki hızla teknolojik değişiklikler, kurum ve kuruluşların adaptasyonunu zorlaştırmış, bu durum da büyük bilişim firmalarının sunduğu 'Bulut Bilişim' altyapılarına yönelmeye neden olmuştur. Bulut bilişimde birçok katman hazır birer servis olarak sunulmakta olduğundan, hem zaman maliyeti hem de güncelleme maliyetinden doğan işgücü açısından avantajlıdır. Aynı şekilde günümüzde, özellikle online satış yapan ya da küresel olarak PII (Personal Identical Information) olarak anılan kişisel özel veri bulunduran kurum ve kuruluşlar için güvenlik çok önemli bir konu başlığı olmuş, bu çerçevede yasalarda da zorunlu hale getirilen bazı maddeler sayesinde kurum ve kuruluşların önceliği haline gelmiştir. Güvenlik bakış açısı da bulut bilişimde büyük oranda sağlayıcı tarafından sağlandığından, bulut bilişime yönelimde güvenlik de önemli bir başlık olmuştur.

Bulut bilişim sağlayıcıları, katman-katman önce altyapıyı, sonra ilgili platformları ardından da yazılımları servis olarak sunmaya başlamış, bu da kurum ve kuruluşların inşa ettiği geleneksel bilişim teknolojilerinin büyük oranda değişimine neden olmuştur. Önce altyapılarının bir kısmını bulut bilişim sağlayıcıları üzerinden hizmete sunan şirketler, ardından azalan operasyon ve kaynak maliyeti sonrası platformlarını ve iş fonksiyonlarını da bulut bilişim altyapılarından hizmet verir hale getirmektedirler.

Sunucusuz Mimari ise, platformun hizmet olarak sunulduğu, yani kullanıcının sadece uygulamasını geliştirdiği, diğer tüm katmanların bulut bilişim sağlayıcıları tarafından sunulduğu kavram olarak adlandırılmaktadır. Bir farklı anlatımı da, backend servisleri (BaaS) ve fonksiyon servisleri (FaaS)'ni bir arada bulunduran ve bu servislerin geliştirme hizmetlerinin bulut sağlayıcısına bırakılmadan, kurum ve kuruluşların kendilerinin geliştirdiği yapıdır. Matematiksel gösterimle, (BaaS) + (FaaS) = Serverless denilebilir.

Literatürde araştırılan çalışmalarda, popüler bulut bilişim sağlayıcıları üzerinde sunucusuz bilişimin inşaları örneklendirilmiş ve bu inşaların performans testleri yapılmıştır. Sunucusuz bilişimin inşası için gereken uygulama katmanındaki değişiklikler, mikroservis yaklaşımlarına değinilmiştir. Popüler bulut sağlayıcıları üzerindeki depolama alanları kullanılarak, uygulamaların ve sunucusuz bilişimin hangi optimal konfigürasyonlarla maksimum verimde çalışabileceği ile ilgili çalışmalar yapılmıştır.

Bu çalışmanın amacı, hiçbir bulut bilişim altyapısı kullanmayan, mevcutta üzerinde, yerinde (on-premise) bir altyapı barındıran sisteme, sunucusuz mimarinin nasıl inşa edilebileceği anlatmaktır. Bir diğer deyişle, daha önce hiç sunucusuz mimari hizmeti vermemiş bir altyapının bu hizmeti nasıl verebileceğini teknik olarak anlatmaktır. Bu kapsamda önce sunucusuz mimari kavramına detaylı olarak değinilecek ve ardından günümüzde bu hizmeti sunan büyük sağlayıcıların bunu nasıl yaptıklarından bahsedilecektir. Geliştirilen Yöntem kısmında ise, bu sistemler gibi bir sistemin mevcut bir altyapıda nasıl inşa edilebileceği teknik olarak anlatılacaktır.

2. Kaynak Araştırması (Literature Survey)

Soltani vd. (2018) çalışmalarında, bulut sağlayıcı hizmetinin tek bir sağlayıcı yerine çoklu sağlayıcılar tarafından alınmasının daha çok fayda getireceğini belirtmişlerdir. Buna ek olarak sunucusuz mimarinin, gereksiz altyapı yönetimlerini gizleme ve geliştiricilerin sadece koduna odaklanması sağlama gibi kolaylıklarının yanında, sağlayıcıların kısıtladığı taraflarının da olduğunu belirtmişlerdir. Çalışmalarında çoklu bulut sağlayıcısı ile sunucusuz mimarinin kısıtlamalarının önüne geçebilecek bir mimari sunmuşlardır.

Perez vd. (2018) çalışmalarında, özellikle yeni modern bir uygulama mimarisi olan mikroservis mimarisi uygulamaları için, sunucusuz mimarinin sağlayıcıların sunduğu kısıtlardan dolayı kurulum ve yaygınlaşma gibi konularda zorluk çıkardığını belirterek, SCAR temeliyle, sunucusuz konteyner bazlı mimari oluşturmuşlardır. Bellek olarak büyük görüntü işlemelerinde, cache bazlı optimizasyonlarla, SCAR'ın maliyeti düşürdüğünü, AWS Lambda ile konteyner kullanımının uygun hale geldiği göstermişlerdir. Bu makalede, AWS Lambda platformu, geliştirilen çözüm ile bir arada performans testine sokulmuştur.

Jain vd. (2020) çalışmalarında, AWS Lambda sunucusuz bilişimin, AWS Elastic konteyner hizmetinin, AWS Fargate hizmetinin, AWS SCAR çatısının performans analizlerini yapmışlardır. Sonuç olarak kullanılacak uygulama türüne göre, kolay konfigüre edilebilecek şekilde bir çözümün kullanılmasının daha uygun olacağını belirtmişlerdir.

McGrath (2017) çalışmasında, Dotnet tabanlı, Microsoft Azure içinde, Windows konteynerlerde çalışan, performans merkezli bir sunucusuz bilişim tasarımı sunmuştur. Tasarlanan sunucusuz bilişim platformuna ait performans metrikleri önermiştir. Google, IBM, Microsoft ve AWS üzerinde eş zamanlı istek kabul etme testi yapmış ve grafiklerle göstermiştir. Bu makalede, Azure üzerindeki sunucusuz mimariye alternatif, Azure'dan bağımsız bir sunucusuz bilişim yöntemi geliştirilmiştir.

Nabell (2019) çalışmasında, bulut sağlayıcılarının sunucusuz bilişim platformları üzerinde çalışan, olay merkezli stateless uygulama fonksiyonları için, CPU ve memory gibi konfigürasyonel değerlerin optimal değerlerini bulma konusunu bir problem olarak ele almıştır. Bu problemi çözmek için, Bayesian Optimizasyonu kullanan bir çatı sunmuş ve bu çatı sayesinde optimal konfigürasyon değerlerini elde etmiştir. Bu çatı aynı zamanda, harcanan zaman ve tutarın azaltılması amaçlı istatistiksel öğrenme tekniklerini kullanmaktadır. Bu teknikler vasıtasıyla harcama tahminleri yaparak en doğru konfigürasyon değerlerinin atanmasına yardımcı olmaktadır.

Gimenez-Alventosa vd. (2019) çalışmalarında, büyük veri setleriyle kullanılan MapReduce dağıtık bilişim modeli işleri için AWS Lambda üzerinde Amazon S3 depolama alanını kullanarak yüksek performanslı bir sunucusuz mimari yaratmışlardır. Sonuçlar göstermektedir ki, AWS Lambda, çoklu ve sık dağıtık modelli işler için homojen olmaya performanslı bir bilgi işlem platformu sağlamaktadır.

Ghaemi vd. (2020) çalışmalarında, sunucusuz görevleri yürütmek için kişisel bilgisayarların hesaplama kapasitesinden yararlanan açık, genel, blok zinciri tabanlı sunucusuz bir platform olan ChainFaaS sistemini tanıtmışlardır. Bu platforma önemli sayıda kişisel bilgisayar bağlandığı takdirde, bazı görevlerin veri merkezlerinden alınabileceğini belirtmişlerdir. Önerdikleri sistemin bir prototipini üretmişler ve uygulanması halinde yeni veri merkezi inşa etme ihtiyacının azalacağını bunun da çevreye olumlu bir etki sağlayacağından söz etmişlerdir.

Eismann vd. (2020) çalışmalarında, açık kaynaklı projeler, endüstriyel kaynaklar, akademik literatür ve bilimsel hesaplamalardan 89 sunucusuz uygulamayı analiz ederek elde ettikleri sonuçları paylaşmışlardır. Bu makalede benzer performans testi yöntemi kullanılmıştır.

Bebortta vd. (2020) çalışmalarında, geospatial büyük verilerin yönetimi için Amazon Web Services (AWS) Lambda, Google Cloud Functions ve Microsoft Azure Functions gibi iyi bilinen farklı, ölçeklenebilir sunucusuz çerçevelerini incelemişlerdir. Geospatial büyük verileri analiz etmede popüler olarak kullanılan mevcut yaklaşımlardan bazılarını tartışmışlar ve sınırlamalarını belirtmişlerdir. Ayrıca bu veriler için bir çerçeve önermişler ve bu çerçevenin performansını güvenilirlik, ölçeklenebilirlik, hız ve güvenlik parametreleri açısından değerlendirerek görselleştirmişlerdir. Önerdikleri model, zamanlama kısıtlamalarını azaltmak için sunucusuz bir çerçeve entegre ettiğini ve ayrıca yüksek boyutlu hiperspektral veriler için geospatial veri işleme ile ilişkili performansı iyileştirdiğini belirtmişlerdir.

Sarkar vd. (2019) çalışmalarında, sunucusuz bilgi işlem paradigmasına dayalı hizmetleri modellemek için akıllı bir binada Nesnelerin İnterneti (IoT) cihazlarının yönetimi için bir çerçeve önermişlerdir. Önerilen çerçevenin, heterojen bir IoT ağından oluştuğunu belirtmişlerdir. IoT uyumlu bir sunucusuz paradigmanın dağıtımının, uç, sis ve bulut bilgi işlem katmanları boyunca hiyerarşik bir yapısal tasarımdan oluştuğunu ifade etmişlerdir. Sunucusuz bilgi işlem paradigmasındaki heterojen IoT ağının yönetiminin, bakımının ve kullanılabilirliğinin etkili ve verimli olmasını sağlamak için bu çerçevede bir veri dağıtım algoritması da önermişlerdir. Çalışma sonunda yaptıkları deney, bulut modeline gönderilen verilere kıyasla sis modeli için gecikmenin daha az olduğunu göstermiştir.

Gupta vd. (2019) çalışmalarında, sunucusuz sistemlerde büyük ölçekli dışbükey optimizasyon problemlerini çözmek için rastgele bir Hessian tabanlı optimizasyon algoritması olan OverSketched Newton'u geliştirmişlerdir. Yaptıkları deneylerde, AWS Lambda'da toplam çalışma süresinde, son teknoloji dağıtılmış optimizasyon şemalarına kıyasla yaklaşık %50 azalma olduğunu ifade etmişlerdir.

Singhvi vd. (2019) çalışmalarında, çok kiracılı(multi-tenant) sunucusuz bir ortamda düşük gecikmeli istek yürütmeyi sağlayan bir platform olan Archipelago'yu sunmuşlardır. Bu platform için yaptıkları testlerde, Archipelago'nun gerçekçi uygulama isteği iş yüklerinin% 99'undan fazlası için gecikme süresini karşıladığını ve son teknoloji sunucusuz platformlara kıyasla kuyruk gecikmelerini 36 kata kadar azalttığını ifade etmişlerdir.

Sreekanti vd. (2020) çalışmalarında, tahmin sunma sistemlerinin, büyük hacimli düşük gecikmeli çıkarım makine öğrenimi modelleri sağlamak için tasarlandığını belirtmişlerdir. Tanıdık bir veri akışı API'ının bu gecikmeye duyarlı göreve çok uygun olduğunu ve değiştirilmemiş kara kutu makine öğrenimi modellerinde bile optimizasyona uygun olduğunu ifade etmişlerdir. Bu API'ı sağlayan ve bunu otomatik ölçeklendirmeli sunucusuz arka uçta gerçekleştiren bir sistem olan Cloudflow'un tasarımını sunmuşlardır. Sonuçta, Cloudflow'un optimizasyonlarının sentetik iş yüklerinde önemli performans iyileştirmeleri sağladığını ve Cloudflow'un gerçek zamanlı video analizi gibi zorlu uygulamaların gecikme hedeflerini karşılayan gerçek dünya tahmin pipeline hatlarında kadar gelişmiş tahmin servis sistemlerinden 2 kat daha iyi performans gösterdiğini söylemişlerdir.

Li vd. (2019) çalışmalarında, sunucusuz bilgi işlemin performansının, birkaç popüler açık kaynaklı sunucusuz platform kullanan bir dizi tasarım sorununa bağlı olup olmadığını araştırmışlardır. Farklı açık kaynaklı sunucusuz platformlar için performansı (aktarım hızı ve gecikme) etkileyen özellikleri belirtmişlerdir. Ayrıca, yalnızca kaynak tabanlı veya iş yükü tabanlı otomatik ölçeklendirmenin sunucusuz platformların ihtiyaçlarını karşılamak için yetersiz olduğunu gözlemlediklerini ifade etmişlerdir.

Manner vd. (2019) çalışmalarında, sunucusuz işlevler için hata tespiti ve çözümünü iyileştirmek amacıyla yarı otomatik bir sorun giderme süreci sunmuşlardır. Sundukları konseptin işlem adımlarının; log kalitesini artırdığını, başarısız yürütmeleri otomatik olarak algıladığını ve log verilerinde sağlanan bilgilere dayanarak test iskeletleri oluşturduğunu ifade etmişlerdir.

Schleier-Smith vd. (2020) çalışmalarında, bulut işlevleri için paylaşılan bir dosya sistemi sunmuşlardır. Sundukları dosya sisteminin, geleneksel paylaşılan dosya sistemlerinin sunabileceği şeylerin ötesinde ölçeklenebilirlik ve performans elde etmek için bulut işlevlerinin ayırt edici yönlerinden yararlanırken tanıdık POSIX semantiği sunduğunu belirtmişlerdir.

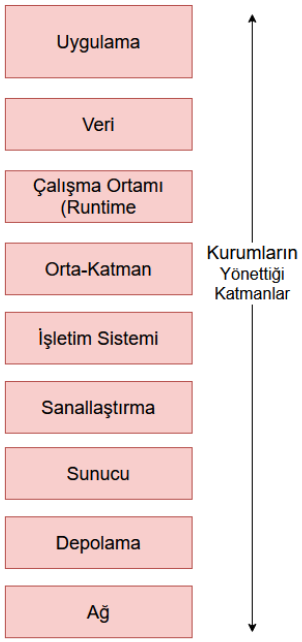
3. Materyal ve Yöntem (Material and Method)

Kurum ve kuruluşlar, bilgisayarla yapılan işlemlerin, bir insan kaynağı ile manuel olarak yapılmasına kıyasla daha hızlı ve güvenilir olduğunu fark ettikten sonra, bilişim altyapılarını kurmaya yönelmişlerdir. Başta finans kuruluşları olmak üzere dünya genelinde kuruluşların bilişim altyapısı inşaları hız kazanmıştır. 2000'li yılların ilk onluk diliminin sonuna doğru gitgide büyüyen altyapılar hem insan kaynağı hem de maliyet açısından kuruluşların yükünü artırmıştır. Yönetim ve bakım zorluğu gibi sebeplerle, altyapılar, yerini önce yönetiminin dış kaynak (out-source) firmalara devrine, ardından da hem yönetim hem de bakım maliyetlerinin bulut bilişim sağlayıcıların otomatik yönetilen sistemlerine bırakmıştır.

Bulut bilişim modeli, John McCarthy'nin 1960'larda ortaya attığı "Bir gün hesaplama işlemleri geniş kamusal ağlar üzerinde gerçekleşecek" görüşüne dayanmaktadır. Bulut kavramı gerçekte bir telekomünikasyon terimi olup servis sağlayıcı ile son kullanıcı arasında kalan ağ üzerindeki sistemi sembolize eder (Yıldırım, 2020).

3.1. Geleneksel Bilişim Sistemleri (Traditional Computing Systems)

Geleneksel altyapılar, bir diğer deyişle içerisinde hiçbir bulut sağlayıcı bulundurmayan altyapılar; tüm ağ, depolama, sunucu, sanallaştırma, işletim sistemi, orta-katman, veri katmanı ve uygulama katmanını kendisi inşa ederek, bakım ve güncelleme maliyetlerini de kurum ve kuruluşların yönettiği altyapılardır. Geleneksel altyapılarda kurumların yönetmek zorunda olduğu katmanlar Şekil 1'de gösterilmiştir.

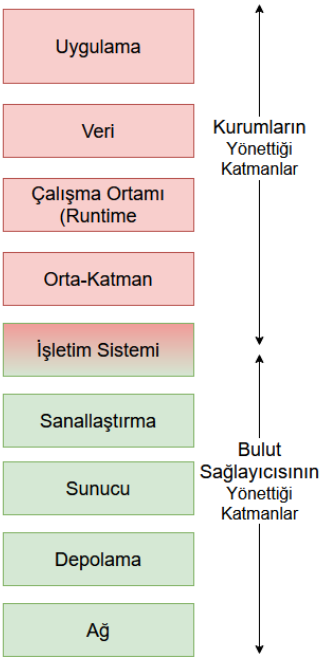


Şekil 1. Geleneksel altyapılarda kurumların yönettiği katmanlar (Layers managed by institutions on traditional infrastructures)

3.2. Altyapı Hizmeti - (Infrastructure as a Service)

Bir bulut sağlayıcısının, ilgili kurum ve kuruluşun altyapı hizmetlerindeki katmanlardan; ağ, depolama, sunucu, sanallaştırma ve işletim sistemi üzerindeki güncelleme, bakım ve onarım gibi operasyonlarına ait hizmetleri kiralayarak servis olarak sunmasına Altyapı Hizmeti (IaaS) denir. Bunun sonucunda müşteriler, bu katmanların yönetimi için insan kaynağına gerek duymazlar. Aynı şekilde, donanım satın almak zorunda da kalmamış olurlar. Bulut sağlayıcıları bu hizmeti görünmez bir şekilde, fiziksel ya da sanal olarak sunabilirler.

IaaS kapsamında müşterinin yönettiği ve bulut sağlayıcısının yönettiği katmanlara ait çizim Şekil 2'de gösterilmektedir. Bulut hizmet sağlayıcısı, bu katmanlardaki hizmetin sağlanabilmesi için gerekli fiziksel kaynakların yönetiminden sorumludur.

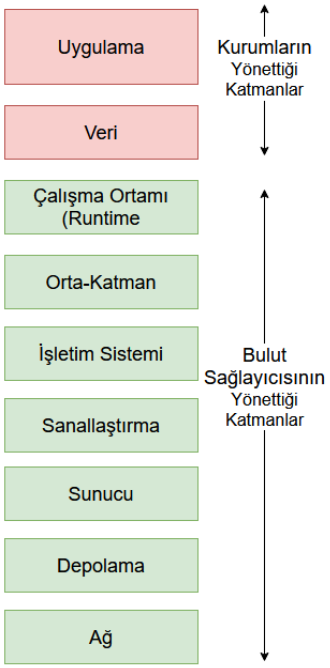


Şekil 2. IaaS kapsamında kurum ve sağlayıcının yönettiği katmanlar. (Layers managed by institutions and providers within the scope of IaaS.)

3.3. Platform Hizmeti - (Platform as a Service)

Bir bulut sağlayıcısının IaaS ile sunduğu hizmetlere ek, işletim sistemindeki tüm hizmetleri, orta-katmandaki yazılımı, orta-katman yazılımının çalıştığı platform yazılımını da sunduğu hizmet türüne platform hizmeti denmektedir. PaaS, öncelikli olarak IaaS kullanıcısı olan müşterilerin, zamanla yazılımlarını 'buluta hazır' (cloud-ready) hale getirerek, orta-katmanın da bulut sağlayıcısından hizmet alınabilir olmasına ve orta katmanın üzerinde koşacağı işletim sisteminin de kiralanabilmesi sonuçlarının doğmasına neden olmuştur. Cloud-ready kavramı ilk olarak PaaS ile ortaya çıkmıştır.

PaaS kapsamında müşterinin yönettiği ve bulut sağlayıcısının yönettiği katmanlara ait çizim Şekil 3'de gösterilmektedir. Bulut hizmet sağlayıcısı, yeşil ile gösterilen bu katmanlardaki hizmetin sağlanabilmesi için gerekli fiziksel kaynakların yönetiminden sorumludur.

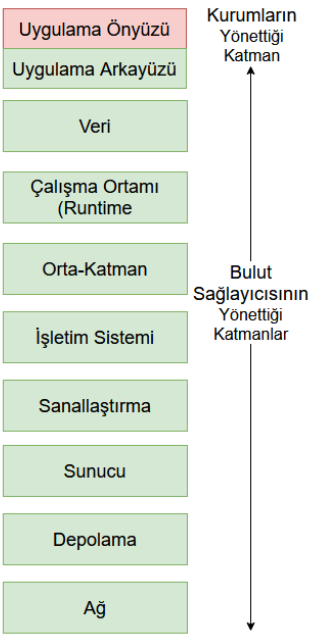


Şekil 3. PaaS kapsamında kurum ve sağlayıcının yönettiği katmanlar. (Layers managed by institutions and providers within the scope of PaaS.)

3.4. Sunucusuz Bilişim - (Serverless Computing)

Sunucusuz bilişim ya da mimari, PaaS'da işletimi orta-katman da dahil bulut sağlayıcısına bırakmanın üzerine, veri ve uygulamanın arka tarafta koşan (backend) servislerini de bulut sağlayıcısının işletimine bırakma işlemi olarak adlandırılabilir. Müşteri kodunu yükler ve bulut sağlayıcısı bu kodu çalıştırır. Sunucusuz denmesinin ana sebebi, uygulama çalışırken sunucu önemsenmemelidir' mesajı vermektir.

Şekil 4'te bir bulut sağlayıcısının ve müşterinin Sunucusuz Bilişim kullandığında hangi sistemleri yönettiği gösterilmiştir. Şekilde de görüleceği üzere, uygulama backend'inin bir kısmını sağlayıcı bir kısmını müşteri yönetebilir. Backend'in hizmet olarak sunulmasına BaaS (Backend as a Service) adı verilmektedir. Yalnızca uygulamanın son kullanıcıya açılan kısmı, bu genellikle bir koddur, müşteri tarafından yönetilmektedir. Eğer backend tarafındaki kod, sağlayıcının belirlediği bazı yazılım dillerinin çatısında (framework) hazırlanmışsa buna da FaaS (Function as a Service) denilmektedir. Bu nedenle BaaS ve FaaS'ın bir arada bulunması durumuna Sunucusuz Mimari denmektedir. İkisi aynı anda bulunmak zorunda değildir. Yalnız BaaS ya da yalnız FaaS işletimine de sunucusuz mimari işletimi denilebilir.



Şekil 4. Sunucusuz mimaride kurum ve sağlayıcının yönettiği katmanlar. (Layers managed by institutions and providers within the scope of serverless architecture.)

3.5. Popüler Bulut Sağlayıcılarının Sunduğu Sunucusuz Bilişim Hizmetlerinin Performans Testi - (Performance Test of Serverless Computing Services Served By Popular Cloud Providers)

Şekil 5'te uluslararası bir teknoloji araştırma şirketi olan Gartner'ın raporu görülmektedir. Bu rapora göre, Temmuz 2019 itibariyle kullanışlı, sektörde lider ve vizyoner sağlayıcılar gösterilmiştir. Grafik incelendiğinde, Amazon ve Microsoft sektörde lider konumdadırlar. Google da onları takip etmektedir (Gartner, 2019).

Şekil 5. Magic Quadrant, Bulut Sağlayıcıları. (Magic Quadrant, Cloud Providers).

3.5.1. Amazon Web Services Sunucusuz Bilişim / Lambda – (Amazon Web Services Serverless Computing / Lambda)

AWS, hem IaaS hem PaaS hem de Serverless hizmetleri konusunda 2020 itibariyle birçok istatistikte, pazar payında lider olarak gösterilmektedir. Sunucusuz (Serverless) mimarisini de AWS Lambda ismiyle, etklinlik odaklı (event-driven) bir yapıda Kasım 2014 yılında piyasaya sürmüştür. Daha önce Amazon Elastic Compute Cloud (EC2) ismiyle duyurduğu PaaS ve IaaS hizmetlerini Lambda ile daha mikro seviyede vermeye başlamıştır. Verilen hizmet etkinlik merkezli (event-driven) mantığında tasarlandığından, bir etkinliğin gerçekleşmesiyle beraber hızlıca, Lambda üzerinde bir servis ayağa kaldırabilecek yapıda hizmet vermektedir.

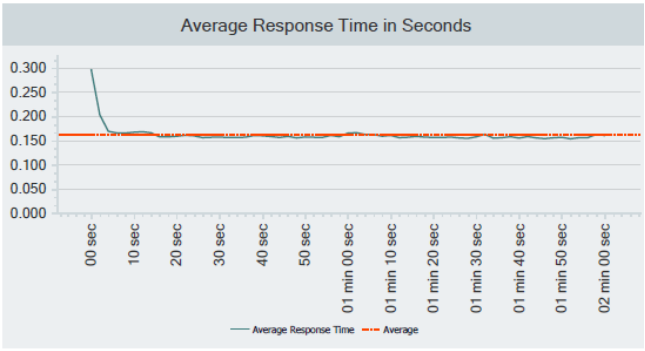
Lambda'nın performansını ölçmek amacıyla AWS üzerinde, 'sunucusuz' bir uygulama oluşturulmuştur. Uygulamanın içine de Şekil 6.'da gösterilen kod bloğu yer alan Node.js 12.0.x ile yazılmış, bir fonksiyon

yerleştirilmiştir. Bu fonksiyon için 128 MB bellek ayrılmış ve 2dk boyunca her saniye 100 sanal kullanıcının yük vermesi sağlanmıştır. Bu süre zarfında toplam 2531 oturum oluşmuştur.

```
exports.handler = function(event, context) {
  context.succeed("Hello, World!");
};
```

Şekil 6. Lambda'nın performansını ölçmek amaçlı kullanılan kod. (The code used to measure the performance of Lambda.)

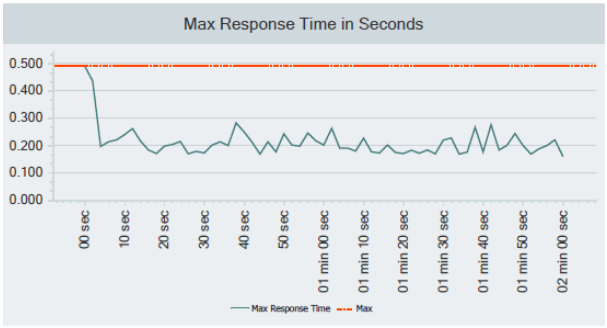
Şekil 7'de ise isteklere verilen ortalama cevap süresi yer almaktadır. İlk oturum, AWS üzerindeki konteynerin ayağa kalkmasının ardından başladığından 0,3 saniye civarındadır fakat sonrasında 2531 isteğe ortalama 0.1618 saniyede cevap verebilmiştir. Yeşil çizgi Ortalama cevap süresini, kırmızı ise genel ortalamaya etkisini göstermektedir.



Average : 0.1618

Şekil 7. Lambda performans testi ortalama cevap süresi grafiği. (Average response time graph of Lambda performance test.)

Şekil 8'de ise maksimum cevap süreleri saniye cinsinden gösterilmektedir. Yine aynı sebepten konteynerin hizmet verebilir hale gelmesi için geçen süre de dahil olunca, ilk istek cevap süresi en yüksek istektir ve bu değer 0,49 saniyedir. Yeşil çizgi maksimum cevap süresini, kırmızı çizgi ise en yüksek değeri işaretlemektedir.



Max : 0.49

Şekil 8. Lambda performans testi maksimum cevap süresi grafiği. (Maximum response time graph of Lambda performance test.)

3.5.2. Microsoft Sunucusuz Bilişim / Azure Functions – (Microsoft Serverless Computing / Azure Functions)

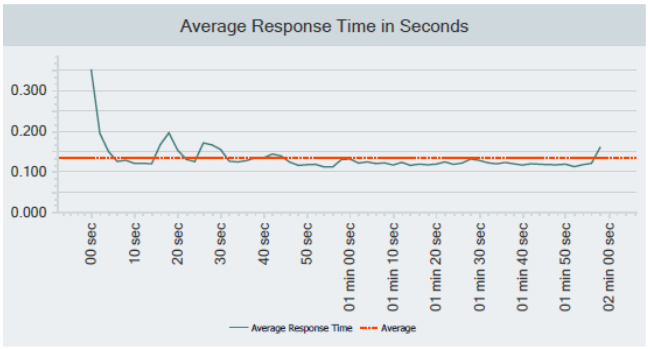
Microsoft'un Azure ismiyle sunduğu Bulut Hizmeti, ilk olarak 2008'in Ekim ayında piyasaya sürülmüştür. Dünyaca ünlü birçok araştırma şirketinin istatistiklerine göre, uluslararası boyutta en çok kullanılan ikinci bulut sağlayıcısıdır. Azure 'sunucusuz' bilişim hizmetini 'Azure Functions' ismiyle ilk olarak Mart 2016'da hizmete sunmuştur.

Azure Functions'ın performansını ölçmek amacıyla, Azure üzerinde sunucusuz bir uygulama ve içerisinde bir adet FaaS hizmeti barındıran fonksiyon oluşturuldu. Şekil 9'da bu kod bloğu yer almaktadır. Burada daha önce AWS üzerinde denenen kodun aynısı, bu sefer Windows temelli bir işletim sistemi üzerinde, yine bir Node.js 12.0.x platformu üzerinde çalıştırılmıştır. Bu fonksiyon için ayrılması gereken bellek kullanıcı tarafından atanmamış, Microsoft Azure platformu grafiklerde görüldüğü, gibi ortalama 132 MB değerinde bir belleği kendisi otomatik ölçeklendirerek atamıştır. AWS ile aynı test özellikleriyle çalıştırılmıştır. 2 dakika boyunca her saniye 100 sanal kullanıcının ilgili servise yük vermesi sağlanmıştır.

```
module.exports = async function (context, req) {
context.log('JavaScript HTTP trigger function processed a request.');
context.res = {
body: "Hello World" };};
```

Şekil 9. Azure Functions'ın performansını ölçmek amaçlı kullanılan kod. (The code used to measure the performance of Azure Functions.)

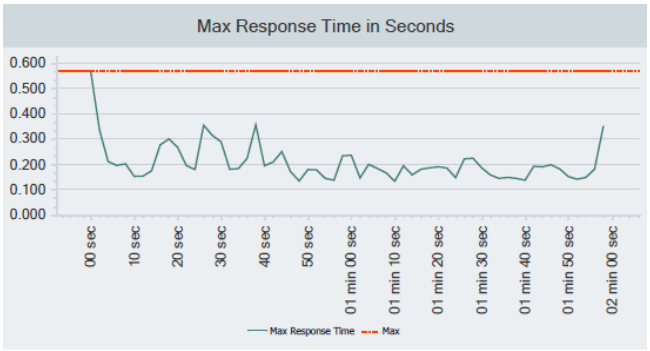
Şekil 10'da ise isteklere verilen ortalama cevap süresi yer almaktadır. İlk oturum, Azure üzerindeki sistemin ayağa kalkmasının ardından başladığından 0.4 saniye civarındadır fakat sonrasında 2548 isteğe ortalama 0.1323 saniyede cevap verebilmiştir. Yeşil çizgi ortalama cevap süresini, kırmızı ise genel ortalamaya etkisini göstermektedir.



Average : 0.1323

Şekil 10. Azure Functions performans testi ortalama cevap süresi grafiği. (Average response time graph of Azure Functions performance test.)

Şekil 11'de maksimum cevap süreleri saniye cinsinden gösterilmektedir. Yine aynı sebepten fonksiyonun Windows bazlı işletim sistemi üzerinde hizmet verebilir hale gelmesi için geçen süre de dahil olunca, ilk istek cevap süresi en yüksek istektir ve bu değer 0.568 saniyedir. Yeşil çizgi maksimum cevap süresini, kırmızı çizgi ise en yüksek değeri işaretlemektedir.



Max : 0.568

Şekil 11. Azure Functions performans testi maksimum cevap süresi grafiği (Maximum response time graph of Azure Functions performance test.)

Örnek bir matematiksel model ile formül gösterimi verilmiştir. Formüller ortalı olarak hizalanmalıdır. Formüllerin yerleştirilmesinde sorun yaşandığı takdirde çerçeve kalınlığı belli olamayan bir tablo içerisine yerleştirme yapılabilir.

4. Deneysel Sonuçlar (Experimental Results)

4.1. Geliştirilen Mimari (Developed Architecture)

Bulut sağlayıcılarının sunduğu sunucusuz bilişim mimarileri, üzerinde bulundurdukları BaaS ya da FaaS'ın çalıştırılabilmesi için öncesinde bir tetikleyiciye ihtiyaç duyar. Bu tetikleyici veri tabanına atılan bir kayıt da olabilir, sistemler üzerine bırakılan bir 'etkinlik' de olabilir bir yere atılmış bir web çatalı da olabilir. Bir diğer deyişle, özellikle başka bir amaç için de programlanabilir.

Yerinde altyapısı olan ve bu altyapısıyla geliştiricilere hizmet veren bir kurumda, aslında sistem yöneticileri ile

geliştirici ekipler arasında bir SaaS (Software as a Service) hizmeti vardır. Geliştirici ekipler yazılımı geliştirip, ilgili kurumun sistemlerine, yazılımın yaygınlaşması için talepte bulunurlar.

Bu bölümde yerinde altyapısı olan, bir diğer deyimle geleneksel bir altyapı bulunduran bir kurum ya da kuruluşun, elindeki sistemle nasıl basit bir sunucusuz mimari oluşturabileceğinden ve bu mimarinin minimum hangi bileşenleri içermesi gerektiğinden ve bu doğrultuda geliştirilen yöntemden bahsedilecektir.

Aslında temelde, müşteri elindeki SaaS platformuna bir web çatalı (webhook) atarak sunucusuz mimari ile çalıştığını ya da sunucusuz bir mimari inşa ettiğini düşünebilir. Fakat inşa edilecek bir sunucusuz mimarinin bazı temel hizmetleri sunuyor olması gerekmektedir.

Birincisi, web çatalı ile tanımlanan fonksiyon arasındaki, tetikleme mekanizmasının bağlarını yapacak bir programdır. Bu program temel olarak, web çatalı üzerinden gelen isteği, geliştiricinin tanımladığı fonksiyona iletecek, iletirken de ilişkiyi kuracak programdır. Geliştirilecek sunucusuz platform, tanımlanacak her fonksiyon için ayrı bir konteyner ya da bir web sunucusu ayağa kaldırmalıdır. Eğer aynı sunucu ya da aynı konteyner üzerinde koşacaksa bu durum kodların birbirlerinin kaynaklarını tüketmesine neden olabilir.

İkincisi, fonksiyonların geliştirilebilmesi için sunulacak programlama ortamlarıdır. Bir önceki bölümde Azure ve AWS üzerindeki sunucusuz platformları performans testine sokarken geliştirme dili olarak Node.js kullanılmıştır. Kurum da eğer kendi sunucusuz platformunu geliştirecekse, belirli diller için geliştirme ortamı sunmalıdır. Kurum kendi geliştirme politikalarına göre bu dilleri hızlıca belirleyip, bu diller için bir geliştirme ortamı sunabilir.

Üçüncüsü ise fonksiyon tanımıdır. Verilecek FaaS hizmeti için, temel bir 'fonksiyon' tanımı ve onun arkasındaki iş mantığı belirlenmelidir. SaaS içinde koşan bir web kancası tetiklendiğinde çalışacak ilk şey fonksiyondur.

Dördüncüsü, sunucusuz platform üzerinde tanımlanacak fonksiyonlar için bir API katmanıdır. Bu API katmanı tüm, yaratma, okuma, güncelleme ve silme (CRUD) işlemlerini yapabilir olmalıdır. Fonksiyonların üzerindeki işlemler bu API'larla, geliştiricinin kullanabileceği formatta olmalıdır.

Son olarak bir diğer gereksinim de yapılacak çalışmanın verimli olması amacıyla, yaratılacak olan sunucusuz bilişim platformunun iyi performanslı ve izlenebilir olmasıdır. Koyulacak fonksiyonun performans metrikleri, uygulamanın bilişim katmanında yarattığı yük, disk maliyeti, kapasite yönetim metrikleri gibi tüm izleme metrikleri kolaylıkla görüntülenebilir olmalıdır. Kuşkusuz bir FaaS zamandan kazandıracaktır fakat performansı da uygulamanın hiçbir sunucusuz hizmet almadan, eski geleneksel yapıda bağımsız şekilde çalışmasına yakın bir performansta olmalıdır.

4.1.1. Bilişim Katmanı (Computing Layer)

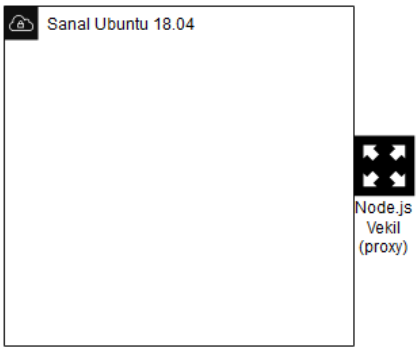
Sunucusuz Bilişim için en temelde gerekli bir donanım ve bu donanıma ait bir CPU, bellek, disk ve dışarıyla iletişimi için gerekli ağ ara birimleri bulunmalıdır.

AWS bu katmanda kendisine ait bir işletim sistemi olan Amazon Linux (AMI) kullanırken AMI'yı host olarak kabul eden konteyner çözümlerini; Azure ise bu katmanda seçimi kullanıcıya bırakarak, Windows, Linux ya da Linux tabanlı konteyner çözümlerini sunmaktadır. Bir önceki bölümde elde edilen performans verileri ise; bellek değeri otomatik olarak artan ve 130 MB civarında olan Windows çözümünün, 128 MB bellek ile AMI üzerinde oluşturulmuş sunucusuz fonksiyona göre daha hızlı çalıştığını göstermektedir.

Geliştirilen çözümde, bilişim katmanı Vmware teknolojisi ile sanallaştırılmış bir sanallaştırma katmanı üzerinde, Ubuntu 18.04 işletim sistemi, bu işletim sistemi dağıtımının minimum sistem gereksinimleri olan, 2 GHZ çift çekirdek işlemci, 2 GB bellek ve 25 GB boş hard disk ile bir VM olarak oluşturulmuştur.

Bu katman, sunucusuz mimarinin tanımında da yer alan, aslında bir sunucunun var oluşunu fakat geliştiricinin hiç düşünmediği, arka plandaki o sunucuyu temsil etmektedir.

Bu katmanın, üzerinde çalışacak fonksiyonları ve o fonksiyonların yönetimsel anlamdaki, CRUD (Create, Read, Update, Delete) işlemlerini yapabilmesi için içerisinde bir istek kabul eden ve yönlendirebilen bir kapıya (port) ihtiyacı bulunmaktadır. Eğer kurumun kendine ait bir vekil sunucusu ya da HTTP URI bazlı yönlendirme yapabilen bir yük dağıtıcısı varsa bu sunucunun önüne konumlandırabilir. Tasarlanan çözümde sunucunun dışarısında, Node.js tabanlı bir vekil sunucu konumlandırılmıştır. Şekil 12'de bu görselleştirilmiştir.

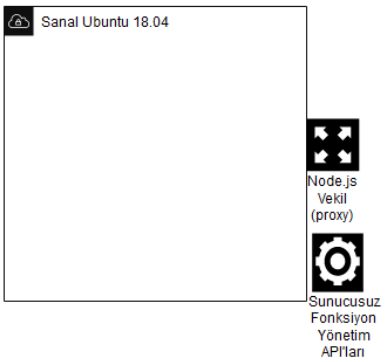


Şekil 12. Tasarlanan çözümün Node.js proxy'si ile gösterimi. (View of the designed solution with Node.js proxy.)

Node.js ile programlanan bu vekil web sunucusunun ana görevi, gelen HTTP isteklerini anlamlandırıp, isteği arkadaki Ubuntu sunucusuna yönlendirmek, dolayısıyla sanal Ubuntu sunucusunun üzerine konumlandırılacak fonksiyonlarla aradaki bağı kurmaktır.

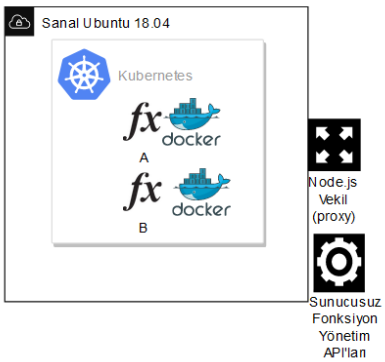
Bundan sonraki aşamada eklenecek bileşen ise, içeri koyulduğu varsayılan bir fonksiyonun yönetimsel amaçlı ihtiyaç duyacağı API katmanlarıdır. Geliştirici fonksiyonunu istediği zaman görüntüleyebilir, silebilir, yaratabilir veya güncelleyebilir. Bu CRUD işlemlerinin yapılabilmesi için bir API katmanı gerekmektedir. Bu API katmanı da aynı şekilde sunucunun üzerinde ya da dışarıda konumlandırılabilir. Eğer fonksiyonların tüm verileri, sunucusuz fonksiyonun çalışacağı sunucu üzerinde tutulacaksa içeride konumlandırılabilir. Tasarlanan çözümde, fonksiyonların yönetilmesi amaçlı kullanılacak API'lar sunucunun dışında başka bir çalışma ortamında konumlandırılmıştır. CRUD API'ları CouchDB üzerinde, Ubuntu sunucusunun dışında konumlandırılmıştır. Sunucusuz bir fonksiyonun hangi sunucuda çalışacağına da bu API katmanı karar verecek ve vekil sunucuda ilgili konfigürasyonu yaparak yönlendirme işlemini sağlayacaktır.

Şekil 13'de sunucusuz fonksiyonların yönetilmesi amaçlı kullanılacak olan, CRUD işlemlerini barındıran API katmanının, tasarlanan çözüme eklenmiş hali gösterilmektedir.



Şekil 13. Tasarlanan çözüme yönetimsel API katmanının eklenmesi. (Adding the administrative API layer to the designed solution.)

Ubuntu sanal sunucusunun içerisine, Docker konteyner altyapısı ve o altyapının orkestrasyonu için ise Kubernetes kurulmuştur. Burada Kubernetes ve Docker tercih edilmesinin sebebi tamamen elde bulunan kaynağın kısıtlı olmasındandır. Eğer birden çok sunucu olsaydı, sunucularda hiçbir konteyner bileşeni kullanmadan, farklı dizinlerde çalışan Node.js ve Python çalışma ortamı ile fonksiyonları rahatça ayağa kaldırılabilirdi. Fakat Kubernetes ve Docker, belirli formattaki konfigürasyonları ile kolayca bir çalışma ortamı ayağa kaldırabilme yeteneğine sahip olduğundan tercih edilen çözüm bu ikili olmuştur. Kubernetes içindeki iki konteynere iki tane farklı Node.js fonksiyonu koyulmuştur. Şekil 14'te bu görselleştirilmiştir.



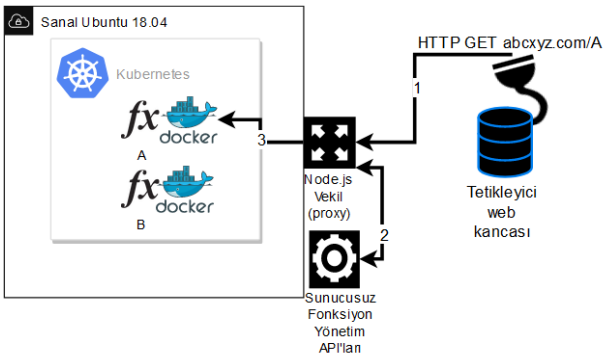
Şekil 14. Tasarlanan çözüme fonksiyonların eklenmesi. (Adding functions to the designed solution.)

İki farklı geliştiricinin A ve B fonksiyonlarını çalıştırabilmesi için, tasarlanan sunucusuz platforma ilettiği varsayıldığında, bu iki fonksiyonun da web üzerinden erişilebilmesi için fonksiyonların çalışacağı konteynerlerin her birinin bir porttan dinleme yapması ve porta gelen isteği bu fonksiyonlara iletebilmesi gerekmektedir. Tasarlanan çözümde basit şekilde, konteynerler içerisinde aşağıdaki Node.js kod bloğuyla iki farklı porttan dinleme yapılmış ve bu fonksiyonların çağrılmasını sağlanmıştır. Şekil 15'de A fonksiyonunun Node.js http sunucusu ile çağrılmasının örneği verilmiştir. İşaretli kısım A fonksiyonudur. B fonksiyonunu da farklı bir port ve farklı bir konteyner IP'si üzerinden açılmıştır. Buradaki işlem manuel değil, daha önce tasarlanan CRUD API'ları vasıtasıyla yapılmaktadır. Fonksiyon yaratma API'ı, ilgili port ve konteyneri Kubernetes üzerinden seçerek, 'MerhabaDunya' isimli A fonksiyonunun içinde olduğu Şekil 15'de gösterilen kod bloğunu ayağa kaldırır.

```
1  const http = require('http');
2
3  const hostname = 'docker_container1';
4  const port = 8080;
5
6  function MerhabaDunya() {
7   return "Merhaba Dunya";
8  }
9
10 const server = http.createServer((req, res) => {
11   res.statusCode = 200;
12   res.setHeader('Content-Type', 'text/plain');
13   res.end(MerhabaDunya());
14 });
15
16 server.listen(port, hostname, () => {
17
18 });
```

Şekil 15. A fonksiyonunun dışarıdan erişilmesi için yazılan kod. (The code written to access the A function from the outside.)

Şekil 16'da, tasarlanan çözümdeki sunucusuz fonksiyon yönetim API'ları sayesinde, Ubuntu sunucusu üzerine konuşlandırılmış, geliştirici tarafından etkinleştirilmiş bir A ve B fonksiyonu ve bu fonksiyonları tetikleyecek web kancası gösterilmiştir. Web kancası (webhook), daha önce tasarlanan altyapıda var olan bir veri tabanı üzerine kancası takılı bir biçimde belirli bir tipteki kaydın veri tabanında yaratılmasını beklemektedir. Veri tabanına kayıt geldiğinde Web kancası tetikleme işlemini başlatacaktır. Tetikleme sonrası web kancasının yarratığı HTTP isteği GET metodu ile 1 numaralı okta gösterilmiştir. İstek Node.js ile yazılmış vekil sunucusuna gelir (1). Hangi adresin hangi konteynere, hangi hosta yönlendirileceği API'ların bağlı olduğu veri tabanında tutulmaktadır. Proxy sunucusu API'lara isteğin nereye yönlendirileceğini sorar (2), API ise cevabında URI yönlendirmesine bakarak, URI dizinindeki /A yönlendirmesine istinaden A konteynerine yönlendirileceğini iletir. İstek tekrar Node.js vekil sunucusuna gelir ve vekil sunucu A fonksiyonunun gömülü olduğu A konteynerine, kancanın yaratmış olduğu isteği iletir (3).



Şekil 16. Tetikleyici vasıtasıyla A fonksiyonunun çağrılması. (Calling of function A via trigger.)

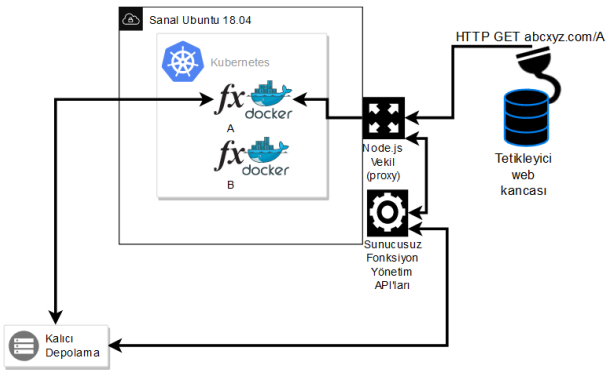
Böylelikle, fonksiyon A için yazılmış kod bloğu, geliştiricinin nerede çalıştığını hiç göz önünde bulundurmadan, yani sunucusuz mimari kavramına uygun bir şekilde çalışmış olur. Sunucusuz mimari hizmeti verecek bir kurumun, basit bir şekilde inşa edebileceği yapı bu şekilde, temel olarak tasarlanabilir. Burada Kubernetes gibi bir konteyner orkestrasyon aracı kullanmak, birçok çalışma ortamını da hızlıca entegre edebilmeyi sağlar. Yazılımcı, bu işlem sonunda, sunucusuz mimari için yazılmış olan CRUD API'ları ile, kendi fonksiyonunu, sadece ilgili fonksiyona ait kod bloğunu ileterek hızlıca altyapı içerisindeki LAN (Local Area Network) ortamına açacaktır. Buradaki çalışma ortamı platformları, kurumun yazılım geliştirme politikalarına göre genişletildiğinde, örneğin Python, Dotnet Core, ruby vb. platformlarla genişletildiğinde çok daha çeşitli mimarilerle dallanabilir.

4.1.2. Stabilite ve Ölçeklendirme Katmanı (Stability and Scaling Layer)

Şekil 16'daki gibi bir mimari elde edildiğinde, sunucusuz mimari içerisinde basit ve temel bir FaaS hizmeti verilebilmiş olunmaktadır. Bu mimari, kurumun geneline yayılmak istendiğinde ve üretim ortamına da açılmak istediğinde iki önemli başlık göz önünde bulundurulmalıdır.

Bunlardan birincisi stabilitedir. Eğer inşa edilecek sunucusuz platform sık sık hata alıp kapanırsa veya performans açısından sorunlar yaratarak, sunucusuz mimari üzerindeki işletim bileşenlerinin bellekleri şişerse bu tür bir işletim kurumun mevcut altyapıdan geriye gitmesine neden olacaktır.

Stabilitenin önemli başlıklarından bir diğeri de veri kaybının yaşanmamasıdır. Örneğin bir X fonksiyonu, web kancası aracılığıyla tetiklendiğinde, fonksiyonun kodlarının, varsa dışarıdan tüketilecek başka bir API'a ait anahtarların ve fonksiyona ait konfigürasyonların kalıcı olarak tutulması gereken bir depolama alanına ihtiyacı olacaktır. Aynı şekilde yönetim API'larının da kullanacağı ana veri kaynağı bu veri deposu olabilir. Sektörde Amazon'un Dynamo DB'si, MongoDB, CosmosDB gibi veritabanı sistemleri sıklıkla kullanılmaktadır. Şekil 17'de kalıcı depolama alanının mimariye entegrasyonu gösterilmiştir. Tasarlanan çözümde kalıcı depolama alanı ayrı bir sunucuda kurulu CouchDB ürünü olarak belirlenmiştir. Tasarım temel prensiplerle resmedildiğinden tasarım için gerekli ihtiyaçları karşılamaktadır.



Şekil 17. Tasarıma kalıcı bir depolama alanının eklenmesi. (Adding a permanent storage to the design.)

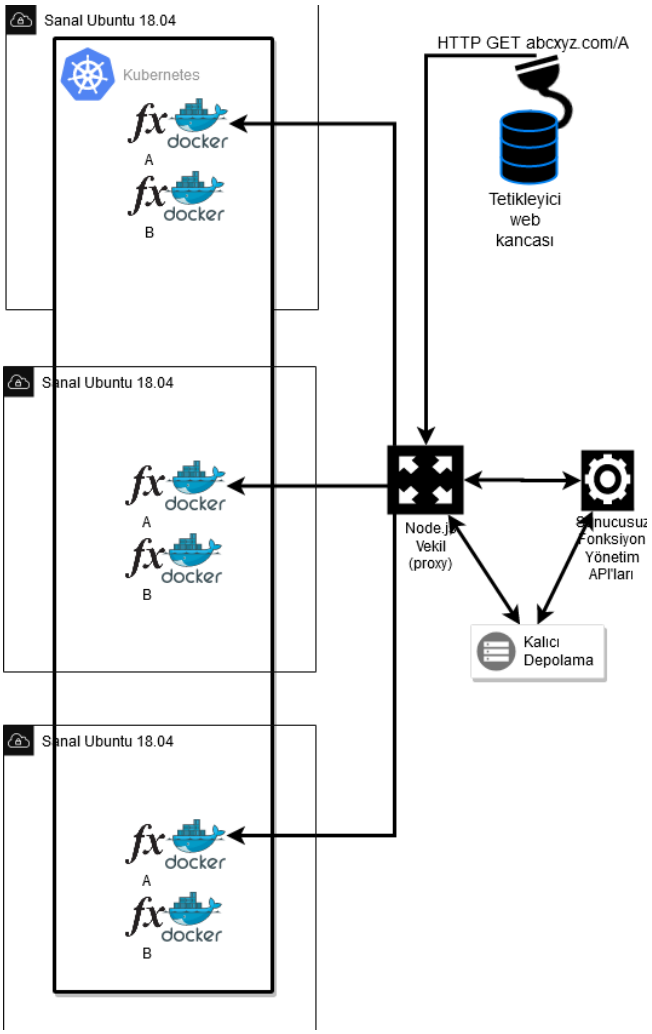
Geliştirici, kodu içerisinde bazı verileri veri tabanına atmak isteyebilir. Fakat geliştirici verisinin hangi veri tabanında tutulduğunu bilmez, çünkü o hizmetler sunucusuz mimari içinde mevcuttur. Yazılımcı fonksiyonu ilettiğinde, veri deposuna, sunucusuz mimari platformunun sağladığı kütüphanelerle erişecektir. Bu kütüphanelere ait kodlar da bu kalıcı depolama alanında muhafaza edilerek, konteynerin içerisinde ihtiyaç

duyduğu anlarda kullanılacaktır. Şekil 18'de, örnek olarak A fonksiyonu geliştiricisinin, kendi uygulaması için, dışarıdan kullanacağı bir adresi, kalıcı depolama alanı olan CouchDB'ye atma yöntemi gösterilmiştir. Görüldüğü üzere, kalıcı depolama alanına '../couchdb' şeklinde erişilmektedir. Yazılımcıya bu erişimi sağlatacak olan merkezi sunucusuz mimari kütüphaneleridir. Örneğin Amazon ve Microsoft bu erişimleri SDK haline getirmiştir ve bu şekilde paylaşmıştır. Tasarlanan çözümde, couchDB'nin varsayılan kütüphaneleri ile bu gereksinim sağlanmıştır.

```
var endpoints = require('../couchdb').use('endpointsForFunctionA');
exports.create = function create(endpoints) {
  endpoints.insert(endpointName, endpointAddress);
};
```

Şekil 18. A fonksiyonuna tanımlı endpoint verilerine yenisinin eklenmesi. (Adding new endpoint data defined to A function.)

İkinci önemli konu ise, sunucusuz mimarinin ihtiyaca göre yatayda ya da dikeyde ölçeklendirilebilmesidir. Ölçeklendirme temelde dikey ve yatay olarak iki yolla yapılabilir. Dikey ölçeklendirme, Kubernetes içerisindeki konteynerleri çoklayarak hızlıca yapılabilir. Tasarlanan çözümde bu büyüme için Kubernetes'in autoscale mekanizması kullanılmıştır. Büyük bulut sağlayıcıları bunu saniyeler içerisinde gerçekleştirebilmektedir. Ölçeklendirme artı yönde büyüme ya da eksi yönde daralma anlamlarını içerebilir. Sunucusuz mimari platformu içerisine koyulan fonksiyonun, bazı zaman aralıklarında çok fazla istek almasından kaynaklı ekstra bellek kaynağına, ekstra Node.js konteynerine ihtiyacı olabilir. Bu anlarda uygulamanın yatayda büyümesi gerekebilir. Yatayda büyümek, dikeyde büyümeye oranla daha fazla zaman ve finansal maliyet gerektirecektir. Bu nedenle yatayda büyümeden önce Kubernetes'in performansı izlenmeli ve bu verilere göre karar verilmelidir, Eğer ki ekstra bir sanal sunucuya ihtiyaç varsa, bunu sanallaştırma katmanı üzerindeki sanallaştırma katmanı API'ları ile, mevcut sunucuya ek sunucu eklenerek çözülebilir. Tasarlanan çözümde Vmware kullanıldığı için dikey büyüme ihtiyacı VMware API'ları aracılığıyla, Ubuntu host'unu çoklayarak yönetilmiştir. İlk sunucu Kubernetes master olarak kalırken, diğer sunucular Kubernetes node'ları olarak kalmıştır. Böylelikle Şekil 19'daki gibi bir mimari elde edilmiştir.



Şekil 19. Tasarlanan çözümün yatay büyüme ile ölçeklendirilmesi. (Scaling the designed solution with horizontal growth.)

Ölçeklendirme politikası, uygulamadan uygulamaya değişkenlik göstereceğinden, genel bir politika ile yönetilmesi, uygulamaların istedikleri ölçeklendirmeye erişememelerini ya da az örnekleme ile çalışacak uygulamanın, politika gereği çok fazla örnekleme (instance) elde ederek fazladan kaynak tüketmesine neden olabilir. Fonksiyonların yönetilecek olduğu API katmanında, yeni bir fonksiyon eklenmesi isteği sırasında mutlaka ölçeklendirme parametreleri de istemelidir. Bu parametrelere göre hem Kubernetes içindeki AutoScale yönetimi hem de sanallaştırma katmanı yönetimiyle, dikey ya da yatay şekilde büyümeyi gerçekleştirebilir.

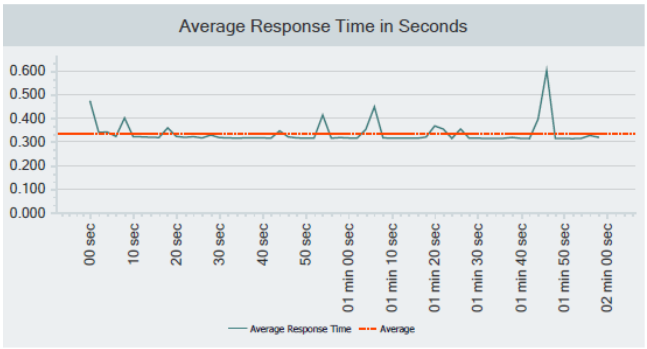
4.1.3. Geliştirilen Sunucusuz Mimari Çözümünün Performans Testi (Performance Test of Developed Serverless Architecture Solution)

Tasarlanan mimarinin performansını ölçmek amacıyla sunucusuz altyapı mimarisi üzerinde, 'sunucusuz' bir uygulama oluşturulmuştur. Uygulamanın içine de Şekil 20'de gösterilmiş olan kod bloğu yerleştirilmiştir. Bu kod Node.js 12.0.x ile yazılmış bir fonksiyondur. Bu fonksiyon için 128 MB bellek ayrılmış ve 2dk boyunca her saniye 100 sanal kullanıcının yük vermesi sağlanmıştır. Bu süre zarfında toplam 2447 oturum oluşmuştur.

```
1
2  function MerhabaDunya() {
3    return "Merhaba Dunya";
4  }
```

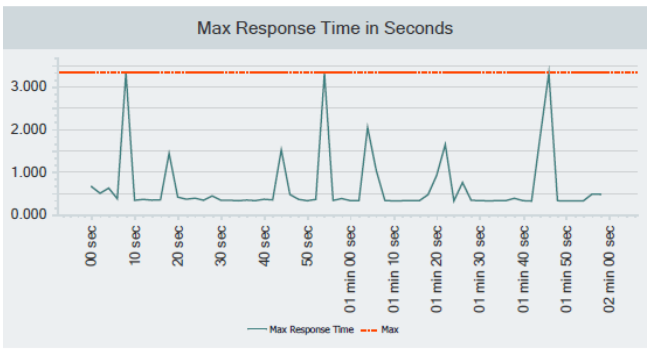
Şekil 20. Tasarlanan çözümün performansını ölçmek amaçlı kullanılan kod. (The code used to measure the performance of the designed solution.)

Şekil 21'de ise isteklere verilen ortalama cevap süresi saniye cinsinden yer almaktadır. İlk oturum, tasarlanan mimari üzerindeki konteynerin ayağa kalkmasının ardından başladığından 0,5 saniye civarındadır fakat sonrasında 2447 oturuma ortalama 0.336 saniyede cevap verebilmiştir. Yeşil çizgi Ortalama cevap süresini, kırmızı ise genel ortalamaya etkisini göstermektedir.



Şekil 21. Tasarıma ait performans testinin ortalama cevap süresi grafiği. (Average response time graph of the performance test of the design.)

Şekil 22'de maksimum cevap süreleri saniye cinsinden gösterilmektedir. Tasarlanan mimaride belirli periyotlarda cevap süreleri yükselmektedir. 2 dakikalık test boyunca maksimum cevap süresi 3.338 saniyedir. Yeşil çizgi maksimum cevap sürelerini kırmızı ise maksimum değeri işaretlemektedir.



Şekil 22. Tasarıma ait performans testinin maksimum cevap süresi grafiği. (Maximum response time graph of the performance test of the design.)

4.1.3. Geliştirilen Sunucusuz Bilişim Mimarisi Çözümünün Performans Testinin Sağlayıcılarla Kıyaslanması (Performance Test Comparison of Developed Serverless Architecture Solution and Popular Cloud Providers)

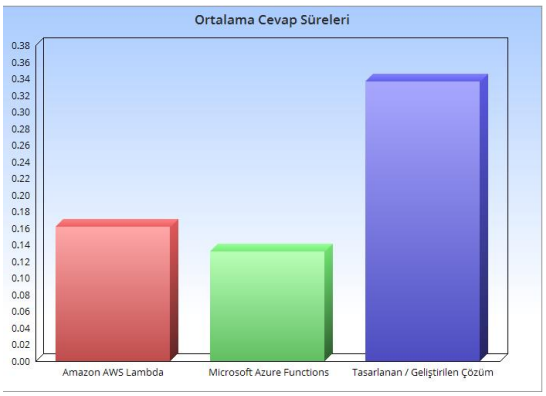
Çalışma kapsamında, iki popüler bulut sağlayıcısı ile tasarlanan çözümün performansları, aynı özelliklerle yapılan performans test kriterleriyle ölçülmüştür. Cevap yükleri hepsinde aynı şekilde ve aynı formatta planlanmıştır. Fonksiyonların kod içeriği de hemen hemen aynıdır. Burada temel fark, Amazon ve Azure platformlarının yazılım geliştiricilere sunduğu geliştirme platformun farkından meydana gelmektedir.

Testlerin koştuğu sunucuların fiziksel lokasyonlarının belirlenmesinde 3 test için de aynı yer seçilmiş ve testi yapan ajanların, bellek, işlemci, ağ kaynakları eşit tutulmuş, kullanılan ajanlar birbirlerinin klonları şeklinde görev almıştır.

Testlerin yapıldığı fiziksel lokasyonlar ve fonksiyonların bulunduğu yerler Şekil 23'de gösterilmiştir. Kırmızı işaret Amazon'daki test edilen fonksiyonun bulunduğu veri merkezi olan IOWA'yı, mavi Azure'daki fonksiyonun fiziksel lokasyonu olan North Virginia'yı, yeşil tasarlanan ve geliştirilen çözümün Ubuntu sunucusunun bulunduğu yer olan İstanbul'u, Sarı işaret de testi yapan sunucunun bulunduğu fiziksel lokasyon olan Minnesota'yı göstermektedir. Tasarlanan ve geliştirilen çözüme ait sunucusuz mimari, testin yapıldığı yere, diğer sağlayıcılara oranla daha uzaktır. Bu nedenle 'ağ gecikmesi' kavramı burada göz önünde bulundurulmalıdır.

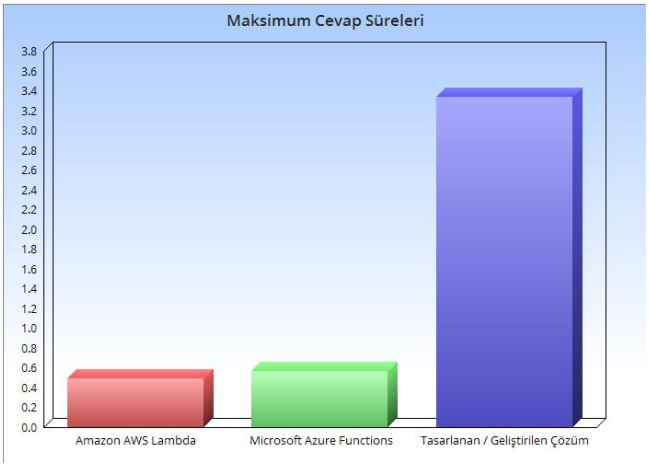
Şekil 23. Yapılan performans testlerinin fiziksel lokasyonları. (Physical locations of the performance tests performed.)

Şekil 24'de ortalama cevap süreleri grafikleştirilmiştir. Aynı test süresi içerisinde en kısa sürede cevabı Azure verirken, ona çok yakın bir değerle Amazon ikinci sırada yer almıştır. Tasarlanan çözümde ise bu süre hemen hemen Amazon'un iki katı, Azure'un üç katı civarındadır.



Şekil 24. Sağlayıcılar ve tasarlanan çözümün ortalama cevap süresi grafiği. (Average response time graph of the providers and the designed solution.)

Şekil 25'de maksimum cevap süreleri gösterilmiştir. Maksimum cevap süresinde Azure en düşük değerle, en hızlı şekilde cevap verebilen sunucusuz platform olmuştur. Amazon ikinci sırada, tasarlanan çözüm ise son sıradadır. Bu grafik açık bir şekilde tasarlanan çözümde performans iyileştirmeleri yapılması gerektiğini göstermektedir.



Şekil 25. Sağlayıcı ve tasarlanan çözümün maksimum cevap süresi grafiği. (Maximum response time graph of the providers and the designed solution.)

Yapılan performans testine ait diğer tüm veriler karşılaştırmalı olarak Şekil 26'da gösterilmiştir.

Çözüm Adı	Amazon AWS Lambda	Microsoft Azure Functions	Tasarlanan / Geliştirilen Çözüm
Hata alan istek sayısı	0	0	0
Standart sapma - Varyans	0.0249	0.0431	0.162
Ortalama cevap süresi (sn)	0.1618	0.1323	0.336
Maksimum Bekleme Süresi (sn)	0.49	0.568	3.338
Toplam Oturum Sayısı	2531	2548	2447
Hata Alan Oturum Sayısı	0	0	0
Başarılı Oturum Sayısı	2531	2548	2447
Bir saniyedeki maksimum kullanıcı sayısı	100	100	100
Test Süresi (saat:dakika:saniye)	0:02:11	0:02:11	0:02:11
İşletim Sistemi Teknolojisi	Linux Konteyner	Windows	Linux Konteyner
Fonksiyonun Fiziksel Lokasyonu	IOWA	Kuzey Virgina	İstanbul
Test Lokasyonu	Minnesota/Amerika	Minnesota/Amerika	Minnesota/Amerika
Test Tarihi	07.06.2020	07.06.2020	12.06.2020

Şekil 26. Sağlayıcı ve tasarlanan çözümün performans testinin karşılaştırmalı verileri. (Comparative performance test data of the providers and designed solution.)

5. Sonuç ve Tartışma (Result and Discussion)

Değerlendirme ve kıyaslamalar göstermektedir ki, tasarlanacak olan sunucusuz platformda en çok efor gerektirecek konulardan biri performanstır. Eğer tasarlanacak olan sistemde performans açısından mevcut sisteme oranla bir yavaşlık olursa, bu sistem yalnızca geliştirme zamanından kazandıracağı için çok faydalı olmayacaktır. Geliştirme zamanı önemli bir konudur fakat asıl önemli olan uygulamanın üretim ortamında göstereceği performans ve son kullanıcı memnuniyeti olduğundan, sunucusuz mimari platformu tasarımında performans önemli bir kalem olarak ele alınmalıdır.

Çalışmalardan edinilen bilgilere göre, sunucusuz mimari hizmeti veren sağlayıcılar, çok fazla programlama diliyle entegre olabilmektedir. Tasarlanan yapıda ise yalnızca Node.js dili ve ona ait tek bir platform (12.0.x) desteklenmektedir.

Tasarlanan mimaride herhangi bir kimlik ve erişim yönetimi sistemi IAM (Identity and Access Management) bulunmamaktadır. Bu nedenle de fonksiyonların erişim adresleri, eğer yanlış bir şekilde konfigure edilirse, servisi tüketmesi gereken uygulamaların yerine başka uygulamaların da aynı servisi tüketmesi sonuçları meydana gelebilir. Uygulama erişim kullanıcılarının ve erişim yetkilerinin merkezi bir sistemde tutulması ve mimari içerisinde ayağa kalkacak fonksiyonun bu yetkilere bakarak işlemleri kabul etmesi, güvenlik açısından önemli bir maddedir.

Tasarlanan mimaride tetikleyici olarak yalnızca web kancası kullanılmıştır. Tetikleyici türleri, bazı dış hizmetlerin tetikleyicilerini veya bir API yönetim aracını da tetikleyici olarak kullanarak genişletilebilir. Bu genişleme sunucusuz mimarinin kullanım senaryolarını da arttıracaktır.

Tasarlanan mimaride tek bir istekte çalışacak maksimum süre ile ilgili bir zaman aşımı değeri koyulmamıştır. AWS bu değeri 900 saniye olarak belirlerken Microsoft bu değeri 600 saniye olarak belirlemiştir. Bu değerin limitsiz olması bütün fonksiyonların bulundukları Pod'lar ya da konteynerler içerisinde şişmelerine neden olacağı için bu limit mutlaka sunucusuz mimari bileşenleri içerisinde koyulmalıdır.

Diğer koyulması gereken bir limit ise eş zamanlı çalıştırılma limitidir. Eğer bir fonksiyon eş zamanlı olarak sonsuz istek kabul etmek zorunda kalırsa bu da aynı şekilde bütün sistemin yorulmasına ve hizmet verememesi anlamına gelebilecektir. AWS bu değeri 1000 paralel çalıştırma limiti ile sınırlandırırken, Microsoft tetikleyici türüne göre kimi zaman limitsiz kimi zaman ise tetikleyicinin verdiği limitlere göre sınırlandırmıştır. Tasarlanan mimaride, tetikleyici olarak web kancası kullanıldığı için bu limitin detaylı performans testlerine göre belirlenerek atanması gerekmektedir.

Sunucusuz mimari içerisine fonksiyon geliştirecek yazılım ekiplerinin, geliştirdikleri kodu nasıl yaygınlaşacakları da önemli bir maddedir. AWS hem sıkıştırılmış format türü olan zip yüklemelerini hem de AWS önyüzü içerisindeki kod düzenleme sayfalarıyla, geliştirilen kodu yaygınlaşabilmektedir. Aynı şekilde AWS'nin eklentileri ile birçok yaygınlaşma ürününe entegrasyon kolaylıkla sağlanabilir. Microsoft ise direkt git kod reposu üzerinden, kendi geliştirme aracı olan Visual Studio üzerinden, yine kendi depolama aracı olan OneDrive üzerinden ve ZİP şeklinde tüm yaygınlaştırmaları kabul etmektedir. Aynı şekilde önyüzünde de bir kod düzenleme sayfası bulunmaktadır. Azure platformunun da birçok yaygınlaştırma aracıyla entegrasyon eklentileri bulunmaktadır. Tasarlanan yapıda kod geliştirmeleri aşamasında bir çözüm sunulmamış fakat yaygınlaşma için CRUD API'ları tasarımda yer almıştır. Bu çalışmada tasarlanan mimarinin, ve yapılan analizlerin yerinde bir veri merkezi, altyapısı olan kurumların kendi sunucusuz mimarisini yapması konusunda temel bir biçimde ışık tutması beklenmektedir.

Çıkar Çatışması (Conflict of Interest)

Yazarlar tarafından herhangi bir çıkar çatışması beyan edilmemiştir. No conflict of interest was declared by the authors.

Kaynaklar (References)

Bebortta S, Das K, Kandpal M, Barik K, Dubey H., 2020. Geospatial Serverless Computing: Architectures, Tools and Future Directions. ISPRS International Journal of Geo-Information, 9(5), 311.

Eismann S, Scheuner J, Van Eyk E, Schwinger M, Grohmann J, Herbst N, Abad C, Losup A., 2020. Serverless Applications: Why, When, and How?. IEEE Software, doi: 10.1109/MS.2020.3023302.

Gartner. 2019. Google Cloud gains in Gartner's 2019 cloud infrastructure Magic Quadrant. https://www.zdnet.com/article/google-cloud-gains-in-gartners-2019-cloud-infrastructure-magic-quadrant/ (Erişim Tarihi: 29.09.2020).

Ghaemi S, Khazaei H, Musilek P. ChainFaaS., 2020. An open blockchain-based serverless platform. IEEE Access, 8, 131760-131778.

Gimenez-Alventosa V, Molto G, Caballer M., 2019. A framework and a performance assessment for serverless MapReduce on AWS Lambda. Future Generation Computer Systems, 97, 259-274.

Gupta V, Kadhe S, Courtade T, Mahoney W, Ramchandran K., 2019. Oversketched newton: Fast convex optimization for serverless systems. arXiv preprint, 1903.08857.

Jain P, Munjal Y, Gera J, Gupta P., 2020. Performance Analysis of Various Server Hosting Techniques. Procedia Computer Science, 173, 70-77.

Li J, Kulkarni G, Ramakrishnan K, Li D., 2019. Understanding open source serverless platforms: Design considerations and performance. Proceedings of the 5th International Workshop on Serverless Computing, December 2019, 37-42.

Manner J, Kolb S, Wirtz G., 2019. Troubleshooting Serverless functions: a combined monitoring and debugging approach. SICS Software-Intensive Cyber-Physical Systems, 34(2-3), 99-104.

McGrath J., 2017. Serverless Computing: Applications, Implementation, and Performance. MSc Thesis, University of Notre Dame, Indiana, USA.

Mustafa Yıldırım, 2015. Bulut Bilişim. http://www.yildirimmustafa.com/2015/06/bulut-bilisim/ (Erişim Tarihi: 29.07.2020).

Nabeel A., 2019. Orchestration and Management of Application Functions over Virtualized Cloud Infrastructures. Ph.D. Thesis, Boston University, Boston, USA.

Perez A, Molto G, Caballer M, Calatrava A., 2018. Serverless computing for container-based architectures. Future Generation Computer Systems, 20(1), 50-59.

Sarkar S, Wankar R, Srirama N, Suryadevara K., 2019. Serverless management of sensing systems for fog computing framework. IEEE Sensors Journal, 20(3), 1564-1572.

Schleier-Smith J, Holz L, Pemberton N, Hellerstein M., 2020. A FaaS File System for Serverless Computing. arXiv preprint arXiv:2009.09845.

Singhvi A, Houck K, Balasubramanian A, Shaikh D, Venkataraman S, Akella., 2019. A. Archipelago: A Scalable Low-Latency Serverless Platform. arXiv preprint arXiv:1911.09849.

Soltani B, Ghenai A, Zeghiba N., 2018. Containerized Serverless Architecture in Multi Cloud Environment. Procedia Computer Science, 134, 121-128.

Sreekanti V, Subbaraj H, Wu C, Gonzalez E, Hellerstein M., 2020. Optimizing Prediction Serving on Low-Latency Serverless Dataflow. arXiv preprint arXiv:2007.05832.