KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

SANAL SUNUCU VE KONTEYNER TEKNOLOJİLERİNİN PERFORMANS KARŞILAŞTIRMASI VE YAZILIM GELİŞTİRME SÜREÇLERİNE ETKİSİ

FERHAT SANUÇ

KOCAELİ 2019

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

SANAL SUNUCU VE KONTEYNER TEKNOLOJİLERİNİN PERFORMANS KARŞILAŞTIRMASI VE YAZILIM GELİŞTİRME SÜREÇLERİNE ETKİSİ

FERHAT SANUÇ

Doç.Dr. Halil YİĞİT
Danışman, Kocaeli Üniversitesi .....

Doç.Dr. Hikmet Hakan GÜREL
Jüri Üyesi, Kocaeli Üniversitesi .....

Dr.Öğr.Üyesi Adem TUNCER
Jüri Üyesi, Yalova Üniversitesi .....

Tezin Savunulduğu Tarih: 03.07.2019

ÖNSÖZ VE TEŞEKKÜR

Günümüzde kurum ve kuruluşlar ile birlikte kullanıcıların da uygulama gereksinim ve kullanım hızları artış göstermektedir. Uygulamaların kullanımında artan ihtiyacı karşılayacak altyapı katmanlarında da gelişmeler yaşanmaktadır. Bulut bilişim teknolojileri ve sanallaştırma çözümleri uygulamalardaki kullanım ve ihtiyaç artışı ile birlikte popülerliğini artırmaktadır. Bu çalışmada, konteyner tabanlı sanallaştırma çözümleri ile hipervizör tabanlı sanallaştırma çözümlerinin performans ve operasyonel karşılaştırılması yapılmıştır. Ayrıca, konteyner tabanlı sanallaştırma çözümlerinin uygulama yaşam döngüsüne katkısı, konteyner tabanlı sanallaştırma çözümleri kullanılarak geliştirilen konteyner otomasyonu ve orkestrasyon çözümleri kullanılarak incelenmiştir.

Tez çalışmasının her aşamasında değerli yardımlarını esirgemeyen, yol gösteren ve her konuda destek olan değerli hocam Doç.Dr. Halil YİĞİT'e teşekkür ederim.

Çalışmanın çeşitli aşamalarında çalışmayı değerlendirip görüşlerini ileten Emre YAZICI ve S. Enes HACIBEKTAŞOĞLU'na da teşekkür ederim.

Her zaman bana güç veren ve en büyük destekçim olan sevgili eşim Yasemin SANUÇ'a ve aileme de sonsuz minnet duygularımı sunarım.

Haziran - 2019 Ferhat SANUÇ

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLALAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

gflops	: Saniyede bir milyar kayan noktalı sayı işlemi
GB	: Gigabayt
MHz	: Megahertz
RPM	: Birim zamandaki dönüş hızı
s	: Saniye
TB	: Terabayt
µs	: Mikro saniye

Kısaltmalar

API	: Application Programming Interface (Uygulama Programlama Arayüzü)
DNS	: Domain Name System (Alan Adı Sistemi)
IAAS	: Infrastructure as a Service (Altyapı Olarak Servis)
KVM	: Kernel-based Virtual Machine (Çekirden Tabanlı Sanal Makina)
LAN	: Local Area Network (Yerel Alan Ağları)
LXC	: Linux Container (Linux Konteyner)
MBPS	: Megabits per Second (Saniye Başına Megabit)
NAS	: Network Attached Storage (Ağa İliştirilmiş Depolama)
NIC	: Network Interface Card (Ağ Arabirim Kartı)
NIST	: National Institute of Standarts and Technology (Ulusal Standartlar ve Teknoloji Enstitüsü)
PAAS	: Platform as a Service (Platform Olarak Servis)
SAAS	: Software as a Service (Yazılım Olarak Servis)
SAN	: Storage Area Network (Depolama Alanı Ağı)
TCP	: Transmission Control Protocol (Geçiş Kontrol Protokolü)
UDP	: User Datagram Protocol (Kullanıcı Veri Birimi Protokolü)
VLAN	: Virtual Local Area Network (Sanal Yerel Alan Ağları)
VMM	: Virtual Machine Manager (Sanal Sunucu Yöneticisi)
ZFS	: Zettabyte File System (Zetabayt Dosya Sistemi)

SANAL SUNUCU VE KONTEYNER TEKNOLOJİLERİNİN PERFORMANS KARŞILAŞTIRMASI VE YAZILIM GELİŞTİRME SÜREÇLERİNE ETKİSİ

ÖZET

Günümüzde, teknolojinin gelişmesine bağlı olarak uygulamaların kullanım alanları ve uygulamaların sayısı artış göstermektedir. Uygulamalardaki ihtiyacın ve kullanımın artması ile birlikte altyapı çalışmalarındaki gelişmeler de artış göstermektedir. Bu nedenle, kaynakların verimli kullanılmasını sağlayan sanallaştırma çözümlerinin popülerliği artırmıştır. Geleneksel yöntemler, her uygulama için ayrı bir fiziksel sunucu kullanır ve fiziksel sunucu üzerine bir işletim sistemi kurulur. Hipervizor tabanlı sanallaştırma teknolojisi, birden çok sanal sunucunun fiziksel bir sunucuda çalışmasını sağlar; ancak, hiper yönetici tabanlı sanallaştırma teknolojilerinde, her sanal sunucu için fiziksel sunucuya ayrı bir işletim sistemi kurulur. Son zamanlarda daha yaygın hale gelen konteyner tabanlı sanallaştırma teknolojileri, hipervizor tabanlı sanallaştırma teknolojilerine kıyasla daha verimli kaynak kullanımı ve daha hızlı uygulama dağıtımı gibi performans ve operasyonel avantajlar sağlamaktadır. Bu çalışmada sanal sunucu ve konteyner ortamları işlemci, bellek, ağ kaynağı performans değerleri ve operasyonel faydalar açısından karşılaştırılmıştır. Performans değerleri ve operasyonel faydaların karşılaştırılması sonuçlarına göre, konteyner tabanlı sanallaştırma çözümlerinin sanal sunucular üzerinde avantajlı olduğu tespit edilmiştir. Konteyner tabanlı sanallaştırma teknolojilerini temel alan ve servis olarak bir platform çözümü sunan Openshift konteyner platformu incelenmiş ve konteyner tabanlı sanallaştırma çözümlerinin uygulama yaşam döngüsüne katkısı incelenmiştir. Bu tez çalışmasında, Openshift konteyner platformunda örnek bir uygulama yaşam döngüsü otomasyonu gerçekleştirilmiştir. Sonuç olarak, uygulama geliştirme aşamasından kullanıcılara hizmet aşamasına kadar geçen süre ve operasyonel iş yükünün azaldığı analiz edilmiştir.

Anahtar Kelimeler: Docker, Hipervizör, Konteyner, Sanal Sunucu, Sanallaştırma.

PERFORMANCE COMPARISON OF VIRTUAL MACHINE AND CONTAINER TECHNOLOGY AND ITS EFFECT OF SOFTWARE DEVELOPMENT PROCESS

ABSTRACT

Nowadays, depending on the development of technology, it is seen that the scope and the number of applications has increased. The developments in the infrastructure works continue with the increasing need at the applications. For this reason, virtualization solutions that provide efficient use of resources have increased popularity. In conventional methods, separate physical servers are used for each application and the operating system is installed on only one physical server for these applications. Hypervisor-based virtualization technology allows multiple virtual servers to run on a physical server. However, in hypervisor-based virtualization technologies, a separate operating system is installed on the physical server for each virtual server. Compared to hypervisor-based virtualization, the container-based virtualization method which has recently become widespread provides operational benefits such as more efficient resource utilization and faster application deployment. In this study, virtual servers and containers will be compared in terms of cpu, memory, network performance metrics and operational benefits. According to the comparison results, container-based virtualization has been found to be more advantageous than virtual servers. In addition to this, Openshift container platform, which is based on container-based virtualization technologies and serves as a platform solution as a service, is examined for the contribution of container-based virtualization solutions to the application life cycle. In the study, a sample application life cycle automation was realized on Openshift container platform and it was analyzed that the time and operational workload decreased from the development stage of the applications to the users service.

Keywords: Docker, Hypervisor, Container, Virtual Machine, Virtualization.

GİRİŞ

Bulut bilişim teknolojileri sahip olduğu temel özellikler sebebiyle kullanıcılara birçok fayda sağlamaktadır. Bulut bilişimin temel özellikleri olarak esneklik, erişilebilirlik, ölçeklenebilirlik ve servis olarak kaynak sağlamasını belirtebiliriz. Getirdiği bu kazanımlardan dolayı, bilişim teknolojileri sanayi, sağlık, işletme alanları gibi farklı disiplinlere sahip kurumlar tarafından benimsenmekte ve yaygın olarak kullanılmaya başlamaktadır. Bulut bilişim teknolojilerinin bu özellikleri sağlamasının temel nedenlerinden biri, altyapısında sanallaştırma çözümleri kullanması ve sahip olduğu büyük kaynak havuzunu paylaştırarak hizmet sunmasıdır [1]. Bulut bilişim teknolojilerinde, kullanıcılar hizmet aldığı altyapının gereksinimlerine veya nasıl barındırıldığına odaklanmak yerine, hizmeti nasıl alacaklarına veya gereksinimlerine odaklanırlar. Bulut bilişim hizmeti veren firmalar, sahip oldukları büyük veri merkezleri üzerinden kullanıcılarına altyapı olarak servis (IaaS), platform olarak servis (PaaS) ve yazılım olarak servis (SaaS) hizmetleri vermektedir. Bulut bilişim teknolojisinde sahip olduğu kaynakları kullanıcıları ile paylaşmak için sanallaştırma teknolojileri ve sanal makinalar kullanılmaktadır. Bulut bilişim hizmetlerinin yaygınlaşması ve altyapısında sanallaştırma teknolojileri kullanması sebebiyle, sanallaştırma çözümlerinin de performansı, verimliliği ve kolay işletilebilir olması önem kazanmıştır [2].

Hipervizör tabanlı çalışan sanal sunuculara ek olarak, konteyner tabanlı sanallaştırma, bulut bilişim ortamlarında çalışan sanal sunuculara farklı bir alternatif sunmaktadır. Linux çekirdeğinde kullanıcı uzayı oluşturarak izole bir şekilde işlemlerimizi çalıştırma ve bu alanın kaynaklarını sınırlama özellikleri gelmesi ile birlikte konteyner sanallaştırma çözümleri ortaya çıkmaya başlamıştır. Linux çekirdeğine gelen bu özelliklere standart oluşturması ile birlikte standart çalışma zamanı, imaj ve yapı standardı oluşturması imkanı sağlayan Docker ile konteyner tabanlı sanallaştırma çözümleri popüler hale gelmiştir [3]. Konteyner mimarisine ilginin artması ve hipervizör mimarisine göre artan faydalar, bulut bilişim sağlayıcıları tarafından platform olarak servis çözümlerinde konteyner tabanlı sanallaştırmanın kullanılmasını

artırdı. Sanallaştırma çözümlerinin çeşitliliği ve kullanım alanları arttıkça, bu alanda yapılan çalışmaların da sayısı artmıştır. Konteynerlerin kullanımının artış göstermesiyle birlikte konteyner ve sanal sunucularını işlemci, bellek, ağ ve depolama gibi performans kriterleri açısından karşılaştıran çalışmalar yapılmıştır [3]. Literatürde, sanal sunucu ve konteynerlerin performans değerlerinin karşılaştırılmasının yanı sıra veritabanı [4] ve büyük veri [5] ortamlarında sağladığı performans değerlerinin karşılaştırılmalarını içeren çalışmalar da yapılmıştır.

Bu çalışmada hipervizör tabanlı sanallaştırma ve konteyner tabanlı sanallaştırma çözümü olan sanal sunucular işlemci, bellek ve ağ performans değerleri açısından karşılaştırılmıştır. Performans ölçümlerine ek olarak, konteyner ortamlarının orkestrasyonu ve servis olarak platform hizmeti amacıyla kullanılmasının yazılım geliştirme süreçlerine olan etkisi incelenmiştir.

Tez çalışmasındaki amaç, birbirinden izole olarak çalışan sanal sunucu, Docker ve linux konteyner ortamlarının performans karşılaştırmasını yapmak, aynı yük altında monitör araçları ile izleyerek stres testinde cevap verdiği yükleri karşılaştırmaktır. Çalışmada, konteyner tabanlı ve hipervizör tabanlı sanallaştırma ürünlerinin performans değerlerinin karşılaştırmasına ek olarak, konteyner tabanlı sanallaştırma teknolojilerinin sağladığı altyapı avantajları ile uygulamaların geliştirilmeye başladığı zamandan kullanıcıların hizmetine alındığı ana kadar yer alan süreçlerin kısaldığı ve operasyonel iş yükünün azaldığı ortaya konulmuştur.

1. GENEL BİLGİLER

1.1. Tezin Amacı

Bu tez çalışmasında, günümüzde kullanılırlığı ve popülaritesi oldukça yükselmiş olan ve gün geçtikçe daha da önem kazanan bulut bilişim sistemleri, sanallaştırma teknolojileri ve bu teknolojilerin yazılım geliştirme süreçlerine sağladığı katkılar incelenerek, performans ve operasyonel avantajların ortaya konulması amaçlanmıştır. Aynı fiziksel sunucu üzerinde bulunan Docker, linux konteyner ortamları ve sanal sunucu ortamları işlemci, bellek ve ağ performans değerleri kullanılarak karşılaştırılmıştır. Performans karşılaştırılmasına ek olarak, konteyner tabanlı sanallaştırma ortamlarının yönetimi, operasyonel maliyeti ve uygulama yaşam döngüsüne olan katkıları incelenerek, yazılım geliştirme süreçlerine ve uygulama yaşam döngüsüne etkilerinin incelenmesi amaçlanmıştır.

1.2. Literatür Taraması

Bulut bilişim ve sanallaştırma çözümlerinin son yıllarda popülerliğinin artması ile birlikte bu alanda yapılan çalışmaların da sayısı artış göstermiştir. Bulut bilişimi kapsayan çalışmalarda genel olarak bulut bilişimin tarihsel gelişimi ve kullanım alanları incelenmiştir. Sanallaştırmanın genel başlık olarak ele alındığı çalışmalarda ise sanallaştırmanın tarihsel gelişimi, yöntemleri ve uygulama alanları incelenmiştir. Bulut bilişim ve sanallaştırma teknolojilerinin ele alındığı çalışmalardan sonra sanallaştırma teknolojileri kapsamında konteyner ve hipervizör tabanlı sanallaştırma çözümlerinin performans karşılaştırılmasını içeren çalışmalar da gerçekleştirilmiştir. Altyapıların performans karşılaştırmasına ek olarak, konteyner mimarisinde çalışan uygulamaların performans analizlerini gerçekleştiren çalışmalar da literatürde yer almaktadır.

Mirzaoğlu, gerçekleştirdiği çalışmada bulut bilişim teknolojilerinin teknik boyutunu, kurum ve kuruluşlarda bulut bilişim teknolojilerinin uygulama alanlarını ve Türkiye'de bulut bilişim teknolojilerinin durumunu incelemiştir [5].

Kaya, gerçekleştirdiği çalışmada bulut bilişim teknolojisinin temel kavramlarını, güvenlik ve hukuki boyutunu incelemiştir. Mevcut ortamların bulut bilişim teknolojilerine taşınmak istediğinde kullanılabilecek teknolojileri araştırmıştır [6].

Felter, çalışmalasında sanal makine ve konteyner ortamlarını işlemci, bellek, ağ ve disk performansları açısından karşılaştırmıştır. Konteyner ortamlarının performans değerlerinin sanal makine ortamlarının performans değerlerine göre daha başarılı olduğu sonucuna varmıştır. Çalışma gerçekleştirilirken performans verilerinin toplanması, analizi ve karşılaştırılması yöntemi kullanılmıştır [7]. Yılmaz vd. çalışmasında linux konteyner, Docker ve Roket konteyner ortamları ile sanal sunucu ortamlarının işlemci, bellek, ağ ve disk performansları açısından karşılaştırılmasını gerçekleştirmiştir. Çalışmada, konteyner ortamlarının performans değerlerinin daha başarılı sonuçlar verdiği tespit edilmiştir. Performans izleme araçları ile veri toplama ve karşılaştırma yöntemi izlenmiştir [8].

Piraghaj vd. çalışmasında servis olarak konteyner hizmeti veren bulut veri merkezlerinde sanal sunucu ortamlarının kaynak planlamasının belirlenmesi üzerine çalışmıştır. Çalışmada, sanal sunucuların gerçek kaynak kullanım verilerinin analiz edilmesi yöntemi kullanılmıştır [9].

Desai, çalışmasında Docker konteyner ve sanal sunucu ortamlarıın işlemci, bellek, ağ ve disk performans değerlerine ek olarak, uygulama performansı açısından kıyaslamasını gerçekleştirmiştir. Yapılan karşılaştırma yöntemlerinin sonuçlarına göre, konteyner ve sanal sunucu ortamlarının, uygulama ihtiyacına göre birbirlerine göre performans üstünlüğü yerine özel konfigürasyonlarla daha verimli sonuçlar aldıkları sonucuna varılmıştır [10].

Cuadrado vd., Docker konteyner ortamlarının ve sanal sunucu ortamlarının servis kalitesi ve enerji verimliliğinin incelenmesini ve karşılaştırılmasını gerçekleştirmiştir. Bu çalışmada, aynı fiziksel ortamda hizmet verilen konteyner ve sanal sunucu ortamlarının miktarı referans alınarak servis kalitesi ve enerji verimliliği analiz edilmiştir. Analiz ve karşılaştırma sonucunda, konteyner ortamlarının servis kalitesi ve enerji verimliliği açısından daha başarılı olduğu sonucuna varılmıştır [11].

Zhang vd., konteyner ve sanal sunucu ortamlarının büyük veri bileşenlerine hizmet

verdiğinde performans değerlerinin karşılaştırmasını gerçekleştirmiştir. Konteyner ve sanal sunucu ortamları büyük veri uygulamasına hizmet verdiğinde, ön yükleme süreleri, uygulama performansı, işlemci ve bellek performansları açısından analiz edilmiştir ve analiz sonuçları karşılaştırılmıştır. Çalışma sonucunda, konteyner ortamlarının büyük veri ortamlarının iş yüklerinin dağıtımında, ölçeklenmesinde ve performansında daha başarılı olduğu tespit edilmiştir [12].

Literatürdeki çalışmalarda, bulut bilişim ve sanallaştırma teknolojilerinin avantajları ve bu teknolojilerin temelini oluşturan konteyner ve sanal sunucu ortamlarının işlemci, bellek, ağ, disk ve uygulama performansı gibi performans metriklerinin karşılaştırılması incelenmiştir. Karşılaştırma yöntemleri kullanılarak, konteyner ortamlarının kaynak kullanımı, verimliliği ve performans değerleri açısından avantajları literatürdeki çalışmalarda gerçekleştirilen analizler sonucunda ortaya çıkartılmıştır.

Bu tez çalışmasında, literatürdeki benzer çalışmalardan farklı olarak, konteyner tabanlı sanallaştırma teknolojileri kullanılarak kaynakların daha verimli kullanıldığı ve uygulama yaşam döngüsünde uygulamanın geliştirilmeye başladığı andan kullanıcılara hizmet vermesine kadar olan süreçlerin otomatize, daha hızlı ve düşük operasyonel maliyet ile gerçekleştirildiği ortaya konulmuştur.

1.3. Tezin Yapısı

Bu tez dört ana başlıktan oluşmaktadır. Birinci bölümde, tezin amacı belirtilmiş ve literatürdeki benzer çalışmalardan bahsedilmiştir. Tezin yapısı kısaca özetlenmiştir. İkinci bölümde, bulut bilişim, sanallaştırma teknolojileri ve yazılım süreçleri hakkında ayrıntılı bilgi verilmektedir. Üçüncü bölümde, ilgili çalışmaların bulgularını desteklemek veya eksiklikleri gidermek amacıyla araştırmanın bulguları ayrıntılı olarak verilmektedir. Dördüncü bölümde, çalışmada elde edilen bulgular, ilgili literatür taraması bulguları ile birleştirilerek sonuç bölümü oluşturulmuştur. Ayrıca, gelecekteki çalışmalara ışık tutabilmek amacıyla öneriler sunulmuştur.

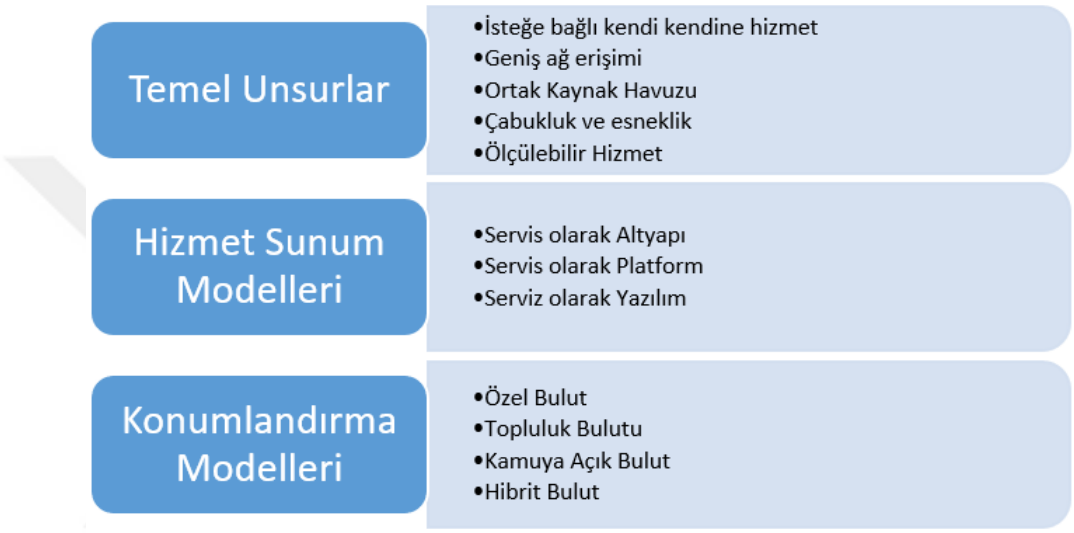
2. BULUT BİLİŞİM, SANALLAŞTIRMA TEKNOLOJİLERİ VE YAZILIM SÜREÇLERİ

2.1. Bulut Bilişim

Bulut bilişim tarihsel olarak 1960'lı yıllarda ortaya çıkmaya başlamıştır; ancak son yıllarda mevcut internet altyapılarının büyümesi ve daha yüksek kapasitede veri taşınabilir duruma gelmesi, işlemci ve depolama teknolojilerinin gelişimi, bilgi teknolojileri altyapılarının daha güçlü bir noktaya gelmesi ile birlikte bulut bilişim teknolojilerinin popülerliği ve kullanım alanları artış göstermektedir. Bulut bilişim birçok kaynakta farklı şekilde tanımlanmıştır. Bulut bilişim yaygın olarak, yapılandırılabilir bilgi işlem kaynaklarından oluşan paylaşılabilir bir havuza, isteğe bağlı ve uygun koşullar altında her zaman ve her ortamdan erişime izin veren bir model olarak tanımlanmıştır. Yönetilen bu kaynaklar (sunucu, bilgisayar ağları, uygulamalar, hizmetler vb.) minimum operasyonel iş yükü ile alıcı-hizmet sağlayıcı etkileşimi gerektirecek şekilde kolaylıkla tedarik edilebilir veya mevcut kaynakları kolaylıkla bırakılabilir [1]. Temel olarak tanımlamak gerekirse, bulut bilişim işlem gücü, depolama alanı, uygulama, veritabanı gibi bilgi işlem hizmetlerinin kullandıkça öde modeli kapsamında internet üzerinden sağlandığı, kullanıcıların basit bir arayüz ile bu kaynaklara eriştiği ve ihtiyaçlarına göre kaynaklarını yönetebildiği teknolojidir [2].

Bulut bilişim modelinde kullanıcılar hizmetlerin hangi yapılar üzerinde çalıştığını, nerede barındırıldığını veya nasıl teslim edildiklerini bilmeden hizmetlere gereksinimlerine göre erişmeye odaklanmaktadır [3]. Bulut bilişim teknolojisinde gerekli donanım sağlayıcı tarafından sunulmaktadır. Hizmeti sağlayan kaynaklar ihtiyaca göre sağlayıcı tarafından dinamik olarak optimize edilebilmektedir. Kullanıcılar hizmeti tek bir noktadan veya arayüzden alsada sağlayıcı tarafından hizmet birden çok lokasyonda veya donanım üzerinde sağlandığı için altyapıda gerekli optimizasyon yapılması gereken durumlarda trafiği yönlendirerek hizmetlerin devamlılığı sağlanmaktadır [1].

Bulut bilişim sağlayıcıları sadece bilgi işlem kaynaklarının oluşturduğu havuzu paylaşmanın dışında birçok farklı hizmet sağlamaktadır. Bulut bilişim hizmetlerinin, yeni uygulama ve hizmetlerin oluşturulabilmesi, veri depolama, yedekleme veya kurtarabilme, ana internet siteleri veya bloglar açabilme, ses ve video akışı sağlayabilme, ihtiyaca göre yazılım ihtiyaçlarının bulut ortamından sağlanabilmesi, veri analizi, verilerin değerli içgörüler sunularak akıllı makine zekası modelleri kullanılabilmesi gibi birçok farklı hizmet alanı bulunmaktadır [4].



Şekil 2.1. NIST tanımına göre bulut bilişim [5]

2.1.1. Bulut bilişimin tarihsel gelişimi

Bulut bilişimin bilinirliği son yıllarda artış göstermiştir; ancak bulut bilişimin çıkış noktası 1960'lı ve 1970'li yıllarda olmuştur. İlk olarak bilişim ihtiyaçlarını karşılamak için üniversite ve kurumsal şirketler fiziksel olarak oldukça büyük alan kaplayan ana bilgisayarlar (mainframe) kullanılmaktaydı. Kullanıcılar işlemlerini anabilgisayarlar ile aralarında arayüz görevi gören terminaller ile yapmaktaydı. Bu yapı ile bilişim altyapı hizmetine erişim, merkezi bir modele sahipti [5].

1980'li yıllarda teknolojinin gelişimi, üretim maliyetlerinin düşmesi ile kişisel bilgisayarlar yaygınlaşmıştır. Sadece ana bilgisayara erişim görevi gören terminaller yerine kullanıcılar kişisel bilgisayarlar temin ederek hizmetleri buradan karşıyabildiler. Bu teknolojik değişim ile bilişim hizmetlerine erişim merkezi yapı yerine dağınık bir modele geçmiştir. Kaynakların kontrolü bu yapı ile kullanıcıların

eline geçmiştir. Bu gelişim ile birlikte işletim sistemi ve kişisel uygulama hizmetleri sunan sektörler hızla büyümeye başladı [5].

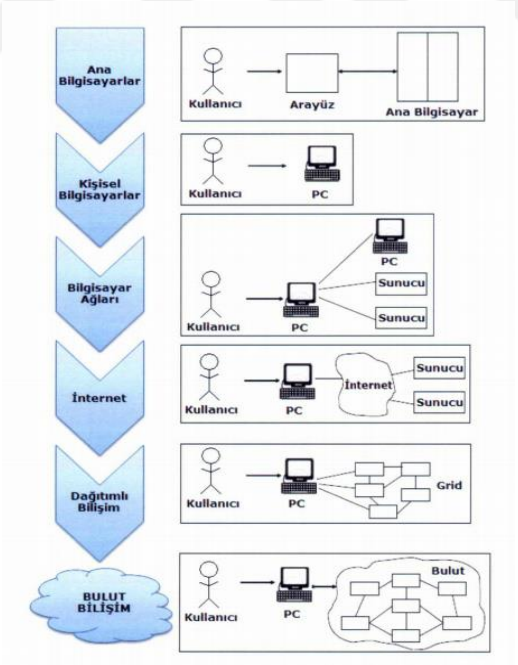
1990'lı yıllarda ise kişisel bilgisayarların kaynaklarının paylaşılarak performanslarını artırmak amacıyla yerel alan ağları (LAN) kullanılmaya başlamıştır. Bu gelişim ile birlikte kurumlar kendilerine ait sunucu bilgisayarlardan oluşan sistem odaları oluşturarak hizmetlerini kendi ortamları için merkezi şekilde sağlamaya başlamışlardır. Bu noktada donanım ve yazılım alanlarında sektörde gelişim hızlanmaya başlamıştır [1].

1990'ların sonunda ise yerel alan ağlarının birbirlerine bağlanarak uzaktaki kaynakların birbirleriyle paylaşılması amaçlanmıştır. Bu gelişim ile yerel ağlar interneti oluşturmuştur. İnternet ilk çıkış noktası olarak haberleşme amacı ile kullanılıyordu; ancak ağ teknolojilerindeki gelişim, bant genişliği ve bağlantı hızlarındaki artış, maliyetin düşmeye başlaması ile birlikte içerik paylaşımı amaçlı da kullanılmaya başlamıştır. Kaynak paylaşımının yanında içerik paylaşımınında yaygınlaşmasıyla birlikte bilişim altyapılarını kullanan kuruluşlar gerçekleştirdiği işlemler için kullandıkları sunuculara ek olarak anti-virüs, e-posta, uygulama hizmeti veren sunucularını da konumlandırmaya başlamışlardır. Bu yıllarda sadece kaynak paylaşımı yerine içerik paylaşımınında yaygınlaşması bilişim sistemlerine erişim modelinin dağıtık bir yapıya sahip olmasına sebep olmuştur [6].

2000'li yıllarda kuruluşların sahip olduğu sunucuların satın alma, enerji, bakım, soğutma, güvenlik gibi toplam sahip olma maliyetlerini düşürmek amacıyla bilişim hizmetlerini dışardan temin edebilme arayışı başlamıştır. Bunun sonucunda dağıtımlı bilişim (grid computing), kamu hizmeti bilişimi (utility computing) ve barındırma (hosting) gibi hizmetler ortaya çıkmıştır [9]. Dağıtımlı bilişim, dağınık haldeki sunucu, depolama ve ağ teknolojileri gibi bilişim teknolojileri kaynaklarının ortak bir havuzda toplanarak tek bir bilişim kaynağı olarak kullanıcılara sunulmasıdır. Bu yapının kullandığın kadar öde mantığıyla bir kuruluşa veya kamuya açık tüm kullanıcılara özel olarak sunulması ise kamu hizmeti bilişimi olarak adlandırılmaktadır [1]. Dağıtımlı bilişim sistemleri toplam sahip olma maliyeti olarak avantaj sağlasada, bir arayüz ile kendi kendine hizmet, ihtiyaca göre kapasite değişimi, dinamik olarak optimizasyon gibi hizmetleri sağlamada yeterli olamamıştır. Bu ihtiyaçları karşılama noktasında

bulut bilişim ortaya çıkmıştır. Bulut bilişimin ilk örneği 2002 yılında Amazon Web Servisleri olmuştur. Amazon firması veri merkezlerini taşıma için çalışma yaptıkları sırada mevcut kapasitelerinin %10'unu kullandıklarını diğer kaynaklarını sadece kampanya dönemlerinde kullandıklarını tespit etmiştir. Bu kullanılmayan bölümleri verimli kullanmak için ilk olarak kendi bünyelerindeki diğer ekiplere kaynakları sunmuşlardır. Daha sonraki süreçte bu kaynakları firmalara hizmet olarak konumlandırarak bulut bilişim kavramını sektörde ticari olarak yer bulmasına zemin hazırlanmıştır. Bu hizmeti sağlarken kullandıkça öde stratejisini benimsemiştir ve bu yenilik sektörde standart haline gelmiştir [14].

Amazon firmasından sonra 2010 yılında Microsoft Azure, 2011 yılında IBM Smart Cloud, 2012 yılında Oracle Cloud servisleri hizmete sunulmuştur. Ticari olarak ortaya çıkan servislerin yanında açık kaynak kodlu bulutlaştırma ürünleri de geliştirilmeye ve kullanılmaya başlamıştır. Bu ürünlere Openstack, Openshift, Opennebula, Apache Cloudstack gibi örnekler verilebilir.



Şekil 2.2. Bulut bilişim tarihsel gelişimi [13]

Anabilgisayarların kullanıldığı süreçten günümüzde bulut bilişim teknolojilerinin kullanımına kadar belirtilen süreçler Şekil 2.2.'de belirtilmiştir.

2.1.2. Bulut bilişim mimarisi

Bu bölümde, bulut bilişim hizmetlerinde etkileşimde olan bileşenler tanımlanmaktadır. Daha sonra, bulut bilişimde hizmet sunum modelleri ve konumlandırma modelleri incelenmiştir. Son kısımda, bulut bilişim mimarisinde yer alan katmanlar incelenmiştir.

2.1.2.1. Bulut bilişim bileşenleri

Bulut bilişim hizmet modelinde birbirleriyle etkileşim halinde bulunan beş ana aktör bulunmaktadır [15].

- Tüketici (Cloud Consumer); servis sağlayıcısının sunduğu yazılım, platform veya altyapı gibi hizmetleri kullanan son kullanıcı veya kurumlardır. Kullandıkça öde prensibine göre hizmetlerden yararlanırlar. Aldıklara hizmetlere arayüzden veya programlama arayüzlerinden (API) erişebilirler. Aldığı hizmetin içeriğine göre tüketici servis sağlayıcı ile çalışabilir ve problem yaşadığı durumlarda servis sağlayıcıdan destek alabilir.
- Servis Sağlayıcı (Cloud Providor); tüketicilere servis sağlayan ve desteğini veren kurum veya kuruluşlardır. Tüketiciler ile aralarındaki anlaşmalara göre servis ve destek vermektedirler. Hizmetin bütününün erişilebilirliği, planlaması ve bakımı servis sağlayıcı tarafından karşılanmaktadır.
- Servis Denetçisi (Cloud Auditor); bulut hizmetinin, operasyonunun, uygulamaların, performans ve güvenliğin değerlendirmesini bağımsız şekilde gerçekleştiren kişilerdir.
- Bulut Komisyoncusu (Cloud Broker); bulut servislerinin teslimatı, kullanımı ve performansı gibi konularda birden fazla servis sağlayıcısı ile tüketici arasındaki görüşmeleri yürüten kurum veya kuruluştur.
- Bulut Taşıyıcısı (Cloud Carrier); servis sağlayıcılar ile tüketici arasındaki bağlantıyı ve hizmetin erişimini internet, telekomünikasyon ve diğer araçlarla sağlayan kurum ve kuruluşlardır.

2.1.2.2. Bulut bilişim modelleri

Bulut bilişim hizmet sunumu üç farklı modele sahiptir. Bu modeller aşağıdaki gibi açıklanmaktadır.

Servis olarak altyapı (IaaS); tüketicilerin gereksinimi olan temel bilişim kaynaklarını konfigure edebildiği ve yapılandırabildiği, bu kaynakların üzerinde işletim sistemi ve uygulamalarını kurabildiği hizmet modelidir. Tüketicinin altyapı üzerinde yetkisi bulunmamaktadır; ancak ihtiyaç duyduğu ölçüde kaynakları aldığı hizmet kapsamında belirleyebilmektedir. İşletim sistemi seviyesinde ise istediği konfigurasyonları ve kurulumları yapabilmesi için tam erişimi vardır [1].

Servis olarak platform (PaaS); servis sağlayıcılar tarafından tüketicilere kendi yazılımlarını geliştirebileceği, test edebileceği ve yürütebileceği platformların sağlandığı hizmet çözümüdür. Tüketici hizmet olarak sağlanan platform içerisinde gerekli konfigurasyonlarını gerçekleştirebilir; ancak platformun sağlandığı altyapı bileşenlerine erişme ve yönetme yetkisi bulunmamaktadır [1].

Servis olarak yazılım (SaaS); servis sağlayıcılar tarafından birden fazla kullanıcıya sunulan, kullandıkça öde prensibine göre sunulabilen yazılım hizmeti çözümüdür. Uygulamanın çalıştığı altyapı, ara yazılımlar, uygulama yazılımları ve verilerinin tamamı servis sağlayıcı tarafında bulunmaktadır. Tüketici internet arayüzü veya programlama arayüzlerinden uygulama hizmetine erişebilmektedir. Bulut altyapısından hizmet aldığı uygulamanın konfigurasyonlarına erişme ve yönetme yektisi yoktur. Sadece alınan yazılım hizmeti içerisinde kendi kullanımını sağlayabilmektedir. Bu model ile kullanıcılar, uygulamanın gereksinimi olan tüm altyapıları ve uygulamaları satın almak ve kurmak, bakımını yapmak ve erişebilirliğini sağlamak gibi ek efor ve maliyet harcamak durumunda kalmamaktadır [4].

Geleneksel yapılarda kurumlar kendi veri merkezlerine sahiptir ve veri merkezi içerisindeki tüm bileşenleri kendileri yönetmektedir. Sadece internet servis sağlayıcılardan ağ hizmetini almaktadırlar. Geleneksel yaklaşımdan bulut bilişim modellerine doğru gidildikçe bilişim sistemleri bileşenlerinin kontrolü tüketiciden çok servis sağlayıcıya geçmektedir. Tablo 2.1.'de bileşenlerin kontrollerinin hizmeti alan tüketiciden servis sağlayıcıya geçtiği gözlemlenmektedir. Hizmeti sağlayan bileşenlerin kontrolü ve yönetimi servis sağlayıcıya geçse de tüketicinin aldığı servisin kalitesini izleme ve ölçme sorumluluğu vardır [5].

Tablo 2.1. Geleneksel yaklaşım ve bulut bilişim modellerinin bileşenleri [5]

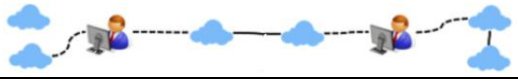
Geleneksel Yaklaşım	IAAS	PAAS	SAAS
Yazılım Uygulamaları***	Yazılım Uygulamaları***	Yazılım Uygulamaları***	Yazılım Uygulamaları*
İşletim Sistemleri***	İşletim Sistemleri***	İşletim Sistemleri***	İşletim Sistemleri*
Sanal Makine***	Sanal Makine*	Sanal Makine*	Sanal Makine*
Sunucular***	Sunucular*	Sunucular*	Sunucular*
Veri Tabanları***	Veri Tabanları*	Veri Tabanları*	Veri Tabanları*
Bilgisayar ağı bileşenleri**	Bilgisayar ağı bileşenleri*	Bilgisayar ağı bileşenleri*	Bilgisayar ağı bileşenleri*

*Hizmet sağlayıcı sorumluluğunda olan katman.
**İnternet servis sağlayıcı sorumluluğunda olan katman.
***Hizmet alıcı sorumluluğunda olan katman.

2.1.2.3. Bulut bilişim konumlandırma modelleri

NIST tarafından tanımlanan bulut bilişim konumlandırma modelleri dört başlık altında toplanmıştır. Bunlar; genel bulut (public cloud), özel bulut (private cloud), karma bulut (hybrit cloud) ve topluluk bulut (community cloud) modelleridir.

Tablo 2.2. Bulut bilişim modellerinin hizmet sağlayıcı ve hizmet alıcısı yapıları [6]

	Hizmet Sağlayıcı	Hizmet Alıcı
Özel Bulut		
Topluluk Bulutu		
Genel Bulut		
Hibrit (Karma) Bulut		

Bulut bilişim modellerinin hizmet sağlayıcı ve hizmet alıcısı arasındaki ilişkisi ve konumlandırma yöntemleri Tablo 2.2.'de belirtilmiştir.

Genel bulut modeli; son kullanıcıya veya kurum ve kuruluşlara hizmet veren konumlandırma modelidir. Kullanıcılar internet aracılığı ile bir arayüz üzerinden

hizmete erişebilirler. Kullandıkça öde prensibi ile ücretlendirilmektedir ve çoğunlukla son kullanıcılara hizmet veren bulut bilişim konumlandırma modelidir.

Özel bulut; kurum ve kuruluşların kendi veri merkezlerinde konumlandırdığı veya hizmet aldığı dışarıdaki bir veri merkezinde bulunan altyapılar üzerinde konumlandırdığı bulut bilişim ortamının modeli özel bulut olarak adlandırılır. Özel bulut modelinde kurum ve kuruluşlar veri üzerinde daha fazla kontrole sahiptir. Ayrıca özel bulut ortamlarının ağ gecikmesi, güvenlik ve mimari kaygıları değerlendirildiğinde diğer modellere göre tüketicinin ihtiyaçları noktasında daha idealdir.

Karma bulut modeli; birden fazla bulut modelinin bir araya gelmesi ile oluşturulan yapılardır. Modeli oluşturan bulut ortamları arasında uygulama ve veri transferi yapılmasına imkan sağlar. Karma bulut modellerinde daha az güvenliğe ihtiyaç duyulan hizmetler genel bulut ortamında, denetlenmesi ve güvenlik altında tutulması gereken hizmetler ise özel bulut ortamında tutulmaktadır. Performans gerektiren hizmetlerin genel bulut ortamında tutularak ihtiyaç durumunda kaynakların artırılması modelin avantajıdır.

Topluluk bulutu; özel bulut modeline ek olarak birden fazla tüketicinin aynı bulut ortamını paylaştığı modeldir. Kurum ve kuruluşların topluluk bulutu modelinde ortamların birden fazla tüketici tarafından paylaşılması sebebiyle güvenlik gereksinimleri, şirket politikaları ve uyumluluk konularında çekinceleri olabilir. Bu model ile tüketiciler altyapıyı kendi veri merkezlerinde sağlayabilir veya bulut hizmeti sağlayıcısından temin edebilir.

2.1.2.4. Bulut bilişim mimari katmanları

Bulut bilişim servis modelleri bulut bilişim mimari katmanları göz önünde bulundurularak oluşturulmuştur. Bu katmanlar dört başlıkta değerlendirilebilir. Donanım katmanı, altyapı katmanı, platform katmanı ve uygulama katmanı olarak bu bileşenler tanımlanabilir.

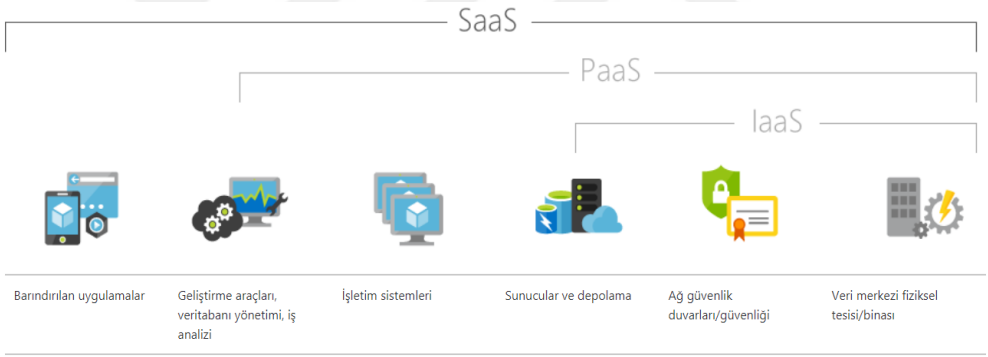
Donanım katmanı; bu katman bulut bilişim hizmetlerini sağlayan veri merkezlerinde konumlandırılmış fiziksel sunucular, depolama alanları, iklimlendirme üniteleri, ağ ekipmanları, enerji ekipmanları gibi fiziksel bileşenleri kapsamaktadır. Donanım

konfigurasyon değişikliği, bakım işleri, fiziksel kapasite artırma veya azaltma bu katmanda tanımlı olan işlerdir [14].

Altyapı katmanı; fiziksel kaynakların yönetiminin yapıldığı, otomatik kaynak atama, yük dağıtımı, kapasite artırımı gibi operasyonların gerçekleştirildiği katmandır. Bu katman sanallaştırma katmanı olarak da belirtilmektedir. Fiziksel kaynaklardan kaynak havuzu oluşturularak kaynakların paylaştırılması bu katmanda gerçekleştirilmektedir [14].

Platform katmanı; altyapı katmanının üzerinde konumlandırılan uygulamaların geliştirilmesi, test edilmesi ve çalıştırılmasına imkan sağlayan katmandır.

Uygulama katmanı; bilişim sistemleri hizmetlerinin en üst katmanıdır. Servis sağlayıcılar geliştirdikleri uygulamaları bu katmanda tüketicilere servis olarak sağlamaktadır.



Şekil 2.3. Bulut bilişim mimari katmanları [4]

2.1.2.5. Bulut bilişim hizmetinin avantaj ve dezavantajları

Bulut bilişim servisleri büyük ve küçük kurum ve kuruluşlara ve bireysel kullanıcılara çeşitli hizmetler sunmaktadır. Kullanıcılar ihtiyaçlarını karşılamak ve ek avantaj elde etmek için bulut bilişim hizmetlerine yönelmektedir. Bu bölümde bulut bilişim hizmetinin avantaj ve dezavantajları incelenmektedir.

Bulut bilişim hizmetlerinin kullanıcılara sağladığı avantajlar aşağıdaki şekilde açıklanabilir.

Maliyet; bulut bilişim, donanım ve yazılım satın alma, bakım ücretleri, şirket içi veri

merkezleri bulundurma ve yürütme, ekipmanların toplam sahip olma maliyetleri gibi masrafları ortadan kaldırır [4].

Ölçeklenebilir olma; kurum ve kuruluşlarda bilişim altyapılarına ihtiyaç ve kullanım zamana göre değişebilir. Ortamların belirli günlerde yüksek kullanım yapılırken diğer günlerde bu kaynaklara ihtiyaç olmayabilir. Bulut bilişim hizmetlerinde kullanıcıların ihtiyaca göre kaynakları artırması veya azaltması ve kullandıkça öde prensibi sayesinde kullanılmayan kaynakların yönetimi ek maliyet ve operasyonel maliyetlerinden kurtulmaktadır.

Kendi kendine hizmet; bulut bilişim hizmetlerinde kullanıcılar servis sağlayıcıya ihtiyaç duymadan kullanılan arayüz ile aldığı hizmetleri yönetebilir.

- Verimlilik; veri merkezlerinde fiziksel ekipmanların yönetimi ve bakımı için birçok operasyonel iş yükü gerekmektedir. Bulut bilişim hizmetlerinde bu iş yükleri servis sağlayıcı tarafından sağlanmaktadır.
- Performans; bulut bilişim hizmeti veren servis sağlayıcılar birçok tüketiciye hizmet verdiği için yüksek performanslı donanımları kullanarak bu kaynakları tüketicilere paylaştırmaktadır.
- Güvenlik; servis sağlayıcılar merkezi kaynakları tüketicilere paylaştırdığı için güvenlik ekipmanları ve yazılımları ile ortamlarının güvenliğini garanti altına almaktadırlar.
- Erişilebilirlik; servis sağlayıcılar hizmetlerin kesintiye uğramaması için hizmet olarak sağladığı kaynakları fiziksel ve coğrafi olarak yedekli yapıda sunmaktadır. Servis sağlayıcıların altyapılarını yedekli olarak sağlaması durmunda bir veri merkezinde problem yaşaması durumunda diğer veri merkezlerinden hizmet verilmeye devam etmektedir.
- Hizmetin ölçülebilir olması; kaynak kullanımının hizmet sağlayıcı ve tüketici tarafından belirli araçlar kullanılarak ölçülmesi hizmetlerin izlenebilir, denetlenebilir ve raporlanabilir olmasını sağlamaktadır. Bu sayede hizmet için kullanılan kaynaklar optimize edilebilmektedir [5].

Bulut bilişim hizmetleri ihtiyaca göre bazı kısıtlamalara veya dezavantajlara sahip olabilir. Aşağıda bulut bilişim hizmetlerinin dezavantajları özetlenmiştir.

- Hizmet sağlayıcıya bağımlılık; hizmet alınan bir bulut bilişim ortamından farklı bir servis sağlayıcısının ortamına hizmetimizi taşımak istediğimizde ek operasyonel yük ve maliyet ortaya çıkmaktadır [5].
- Hizmet kalitesinin öngörülemez olması; kullanıcılar hizmetin devamlılığının ve kalitesinin belirli bir düzeyin üstende sağlanmasının sürekliliğini öngöremeyebilirler. Bu hizmet servis sağlayıcılar tarafından garanti edilmektedir. Ayrıca hizmete erişimin kalitesi servis sağlayıcının ortamı ile tüketicinin arasındaki erişimin durumuna bağlıdır. Hizmetin içeriği ve şartları servis sağlayıcı ve tüketici arasındaki anlaşmalar ile sağlanmaktadır.
- Güvenlik açıkları; tüketici için kritik bilgi ve belgelerin bulut ortamında tutulması tüketici için endişeye sebep olabilir [6].
- Yasal güçlükler; kişisel verilerin korunması, tarafların sorumluluklarının korunması, kritik verilerin korunması gibi noktalarda yasal kısıtlamalar ortaya çıkabilir.

2.2. Sanallaştırma

Bu bölümde bulut bilişim teknolojilerinin gelişiminde önemli rol oynayan sanallaştırma teknolojileri, tarihsel gelişimi, sanallaştırma türleri ve karşılaştırmaları incelenecektir.

Bulut bilişim teknolojilerinin merkezini sanallaştırma teknolojileri oluşturmaktadır. Bulut bilişimin gelişiminde sanallaştırma teknolojilerinin getirdiği faydalardan yararlanılmıştır. Sanallaştırma teknolojisinin birçok tanımı yapılmıştır. Yaygın olarak kullanılan tanımına göre sanallaştırma, fiziksel bir kaynağı mantıksal yapılara ayırması ve her mantıksal birimi farklı fiziksel kaynak gibi göstererek hizmet olarak sağlamasıdır. Sanallaştırma, farklı metotlar kullanarak fiziksel kaynak ile işletim sisteminin ve kullanıcıların soyutlanmasıdır. Sanallaştırma fiziksel kaynakların daha verimli kullanılmasını sağlamakta, bir fiziksel kaynak üzerinden birçok farklı işletim sistemine izole şekilde hizmet vererek kaynağın tamamını verimli kullanmayı amaçlamaktadır.

Sanallaştırmada kullanıcı ile fiziksel kaynak arasındaki yönetim iki farklı metodla yapılmaktadır. Hipervizör veya sanal makina monitörü olarak adlandırılan yazılım ile fiziksel kaynakları mantıksal yapılara ayırarak, bir fiziksel sunucu üzerinden birden fazla işletim sistemine sahip sanal sunucular oluşturulmasına imkan sağlamaktadır.

Diğer yöntem ise, konteyner tabanlı sanallaştırmanın temelini oluşturan isim uzayı fonksiyonları gibi linux çekirdeğinin özelliklerini kullanarak gerçekleştiren sanallaştırma yöntemidir. Bu yöntem ile fiziksel kaynağın üzerinde bulunan işletim sisteminde bağımsız süreçler ve isim uzayları oluşturularak mevcut işletim sistemi üzerinde izole yapılar oluşturulmaktadır. Bu izole yapılar konteynerlerin temelini oluşturmaktadır.

2.2.1. Sanallaştırmanın tarihsel gelişimi

Sanallaştırmanın gelişimi 1960'lı yıllarda çoklu programlama ve zaman paylaşımı kavramlarının öne sürülmesi ile başlamıştır. Yapılan çalışmalar sonucunda işletim sistemleri üzerinde toplu işlem yapma ve zaman paylaşımı teknolojilerinin temelleri atılmıştır. Zaman paylaşımı teknolojisi ile birden fazla uygulama ve kullanıcının eş zamanlı olarak kaynakları kullanabilmesine imkan sağlanmıştır. Kullanıcıların hissedemeyeceği kısa sürelerde fiziksel kaynağın paylaştırılması ile kullanıcı ve uygulamalar fiziksel kaynağın kendilerine ait olduğunu düşünmektedir. Bu yapıda fiziksel kaynağın üzerindeki sistemler izole olmadığı için bir uygulamanın hata alması durumunda bütün sistem veya tüm kullanıcılar etkilenmektedir. Bu sebeple paylaşılan kaynakların yalıtılması ihtiyacı oluşmuştur [14]. Bu yıllarda yürütülen atlas projesiyle bir tür sanal makina monitörü veya hipervizör olarak adlandırabileceğimiz supervizor ve sanal bellek kavramları kullanılmaya başlamıştır. IBM firmasının yaptığı deneysel çalışmalarla sanal makina kavramı konuşulmaya başlamıştır ve yürütülen bu çalışmalar sonucunda 1966 yılında ilk kez tam sanallaştırma gerçekleştirilmiştir [15].

1980'li yıllarda fiziksel donanım maliyetlerinin düşmesi ile kişisel bilgisayar kullanımları yaygınlaşmıştır. Bunun sonucunda yapılar merkezi sistemlerden dağınık sistemlere dönüşmüş ve istemci/sunucu mimarisi yaygınlaşmıştır. Bu sebeple bu yıllarda sanallaştırma çözümlerinin geliştirilmesine ilgi azalmıştır [15].

1990'lı yıllarda bilgisayar ve sunucu kullanımının da artması sonucunda dağınık mimaride sistemleri yönetmenin ve kaynakları etkin kullanmanın maliyet ve operasyonel süreçler açısından ekonomik olmadığı anlaşıldı. Bunun sonucunda uygulama hizmetlerinin merkezi olarak konumlandırılması için çalışmalar yapıldı. İlk olarak her uygulama için farklı fiziksel sunucular konumlandırılarak uygulamalar birbirinden izole olarak hizmet verilmesi amaçlandı. Bu mimaride ise yapı büyüdükçe

altyapının maliyeti arttığı ve fiziksel ortamlarda kullanılmayan atıl kaynak miktarınında yükseldiği gözlemlendi [15]. Bunun sonucunda 1990'lı yılların sonlarında sanallaştırmanın önemi tekrar gündeme geldi ve bu alandaki çalışmalara odaklanıldı. Stanford Üniversitesinde araştırılan ve proje olarak geliştirilen sanal makine teknolojisi 1998 yılında sanallaştırma yazılımı merkezli kurulan Vmware şirketi ile birlikte geliştirilmeye başlanmıştır ve ilk ürün olarak Wmware iş istasyonu (workstation) pazara sürülmüştür [14].

Sanallaştırmanın tekrar popüler olması ile Vmware şirketine ek olarak Sun, IBM, Microsoft, Xensource gibi diğer teknoloji firmaları da sanallaştırma alanında geliştirmelerini hızlandırmış ve ürünlerini piyasaya sürmüştür. Yazılım şirketlerine ek olarak donanım üreticisi olan AMD ve Intel şirketleri de sanallaştırma desteği veren ürünlerini geliştirmiş ve piyasaya sürmüştür [14].

Tarihsel gelişiminde bahsettiğimiz bu sanallaştırma teknolojisi hipervizör tabanlı sanallaştırma teknolojisi olarak da bilinmektedir. Sanal makina teknolojilerinin gelişimi için de son dönemlerde popülerliğini artıran konteyner tabanlı sanallaştırma teknolojisi geliştirilmiştir.

1979 yılında Unix V7 ile bir işlemin kök dizinini değiştirmeye ve ana dosya sisteminden bağımsız ilgili işleme ait alt dosya sistemi oluşturmaya imkân sağlayan "chroot" özelliği tanıtılmıştır. Bu özellik ile her işlem için ayrı bir dosya sistemi oluşmasına imkân sağlanarak işlem izolasyonunun temeli atılmıştır. 1982 yılında ise "chroot" özelliği linux BSD sürümünde çekirdeğe özellik olarak eklenmiştir.

2000'li yıllarda FreeBSD'nin güvenliği artırılmış versiyonları geliştirilmiştir. 2000'li yıllarda her sistem ve konfigurasyon için farklı ip adresi atama imkânı sağlayan FreeBSD Jail tanıtılmıştır. 2001 yılında dosya sistemi, ağ adresleri, ana bellek gibi kaynakları ayrı olarak bölümlemeye imkan sağlayan Linux Vserver tanıtılmış ve 2006 yılına kadar desteği verilmiştir [16].

2004 yılında anlık kopya ve ZFS'den klon almayı kolaylaştıran sistem kaynak kontrollerini birleştiren ve bölge (zone) yapısı ile kaynakları ayırma imkanı sağlayan Solaris Konteyner ürünü piyasaya sürülmüştür. 2005 yılında sanallaştırma, izolasyon, kaynak yönetimi ve denetim noktası belirleme için linux çekirdeğine yama olarak

eklenen OpenVZ geliştirilmiştir. Bu ürün ile linux çekirdeğinin bir parçası olarak değil bir yama olarak geliştirilmiştir [16].

2006 yılında Google firması tarafından geliştirilen kontrol grupları (cgroups) ile bir işlemin kullandığı işlemci, bellek, disk ve ağ kaynaklarının yönetimi, sınırlandırması ve yalıtılması özellikleri linux çekirdeğine eklenmiştir. Bu özellik linux çekirdeğine 2.6.24 sürümü ile eklenmiştir [14]. 2008 yılında duyurulan LXC (linux konteyner) ile linux konteyner yönetimi tam olarak uygulanabilmiş ve işletim sistemi seviyesinde konteyner sanallaştırmasından yararlanılmıştır. Bu geliştirme ile kullanıcı uzayları ve kontrol grupları bir yama gerektirmeden linux çekirdeğine entegre olmuştur. Bu özellikler ile linux işletim sistemi üzerinde işlemci, bellek, disk ve ağ kaynakları her işlem için izole edilebilir ve yönetilebilir olmuştur.

2013 yılında dotCloud olarak bilinen daha sonra ismini Docker olarak düzenleyen firma tarafından Docker açık kaynak kodlu olarak geliştirmeye başlanmıştır. Docker ilk geliştirildiğinde linux konteyner teknolojisinin üzerine geliştirilmeye başlanmıştır. Daha sonra kendi kütüphanelerini ve konteyner yöneticisini kullanmıştır. Docker'ın geliştirilmesi ile birlikte konteyner tabanlı sanallaştırma mimarisi popülerliğini artırmıştır. Konteyner teknolojilerinde günümüz yaygın olarak Docker konteyner teknolojisi kullanılmaktadır.

2014 yılında CoreOS tarafından roket konteyner teknolojisi geliştirilmiştir. Roket konteyner teknolojisini docker teknolojisinin güvenliği artırılmış versiyonu olarak düşünebiliriz.

2.2.2. Sanallaştırma modellerinin sınıflandırılması

Sanallaştırma modellerini sınıflandırırken sanallaştırılan kaynağın türü ve hangi yöntem ile sanallaştırıldığı referans alınmaktadır. Sanallaştırma modelleri aşağıdaki başlıklarda sınıflandırılabilir.

Sunucu sanallaştırma; sanallaştırma modelleri arasında en yaygın olarak kullanılan model sunucu sallaştırma modelidir. Geleneksel yöntemlerde bir fiziksel kaynak bir işletim sistemi için kullanılmaktadır. Bu yapılar büyüdükçe toplam sahip olma maliyetleri yükselmekte, operasyonu zorlaşmakta ve fiziksel kaynağın tamamı her uygulama tarafından kullanılmadığı için atıl kaynakların miktarı artmaktadır.

Belirtilen olumsuzlukları aşmak için bir fiziksel kaynağın üzerine birden fazla işletim sistemi kurulmasına imkân sağlayan sunucu sanallaştırma teknolojileri geliştirilmiştir.

Masaüstü sanallaştırma; masaüstü sanallaştırma yöntemlerinde bir fiziksel kaynak üzerinden birden fazla son kullanıcıya hizmet verilerek her kullanıcıya kendi ortamı sağlanmaktadır. Masüstü sanallaştırma kullanıcılara kaynakları en verimli kullanarak ve işletim sistemi lisans maliyetlerinden tasarruf elde ederek hizmet vermeyi amaçlamaktadır.

Ağ Sanallaştırma; ağdaki fiziksel kaynakların ve bant genişliğinin birbirinden bağımsız istemcilere paylaştırılmasıdır. Sanal ağ katmanı ağ arayüz kartı (NIC), ağ anahtarı (switch), ağ taşıyıcıları (VLAN) ve ağ depolama araçları (NAS) gibi bileşenlerden oluşmaktadır.

Bellek Sanallaştırma; bellek sanallaştırma modelinde uygulamalar fiziksel bellek bilgisini bilmeden işletim sistemi üzerinden bellek alanlarına erişebilirler. Bu modelde veri merkezindeki sunucuların belleklerinin tek bir havuzda toplanarak belleğin ortamlar arasında paylaşımı amaçlanmaktadır.

Veri Depolama Sanallaştırma; veri depolama sanallaştırma modeli birbirinden bağımsız ortamların verilerini saklamak için merkezi depolama kaynaklarının kullanılmasıdır. Bu modeli referans alan en çok kullanılan sistemler depolama alanı ağı (SAN) ve ağa iliştirilmiş depolama (NAS) ortamlarıdır.

2.2.3. Sanallaştırma teknolojileri

Sanallaştırma mevcut fiziksel katmanın mantıksal birimlere ayrılarak istemcilere paylaştırılmasıdır. Sanallaştırma teknolojileri sanallaştırmanın gerçekleştirildiği katmanda kullanılan teknoloji ve yöntemlere göre sınıflandırılmaktadır.

- Hipervizör Tabanlı Sanallaştırma
  Hipervizör Tip 1
  Hipervizör Tip 2
- Konteyner Tabanlı Sanallaştırma

2.2.3.1. Hipervizör tabanlı sanallaştırma

Hipervizör tabanlı sanallaştırma mimarisinde hipervizör veya sanal sunucu yöneticisi (VMM) olarak adlandırılan yazılım ile ana sunucunun sahip olduğu donanımlar (işlemci, bellek, disk, ağ arayüzü gibi) üst katmanda bulunan sanal sunuculara paylaşılmaktadır. Bu sanallaştırma mimarisinde sanal sunucular kendi işletim sistemi ve kütüphanelerine sahiptir. Hipervizörün alttaki donanımı sanallaştırarak paylaştırması sayesinde ana sunucu ve kendi işletim sistemlerine sahip sanal sunucular birbirlerinden bağımsız olarak çalışmaktadır [17].

Hipervizör tabanlı sanallaştırma teknolojisinde çeşitli firmaların sağladığı teknolojiler bulunmaktadır. Bu hipervizörler altta bulunan donanımı yönetme ve birbirinden izole sanal ortamlara hizmet vermeyi amaçlamaktadır. Bu hipervizörler tip 1 ve tip 2 olarak iki kategoriye ayırılmaktadır.

Tip 1 hipervizörler; doğrudan donanım üzerinde çalışmaktadır ve üzerindeki sanal sunucuların işletim sistemlerini izleyebilmektedir. Bu tip 1 hipervizörlerin ana sunucu üzerindeki işletim sistemininde yerini almakta ve doğrudan donanıma erişmektedir. Bu hipervizörler çekirdek modunda çalışır ve kendi özel fiziksel işlemcilere sahiptir. Oracle VM Server for SPARC, Citrix XenServer, VMware ESX/ESXi ve Microsoft Hyper-V tip 1 hipervizörlere örnek verilebilir [18].

Tip 2 hipervizörler; ana sunucunun sahip olduğu işletim sistemi üzerinde çalışmaktadır. Bu tip hipervizörlerin işletim sisteminde çalışan yazılımlar olarak düşünebiliriz. Tip 2 hipervizörlere Oracle Virtual Box, Wmware Workstation ve Çekirdek tabanlı sanal sunucular (KVM) örnek verilebilir [18].

Çekirdek tabanlı sanal makina (KVM) teknolojisi Intel VT veya AMD-V sanallaştırma uzantılarını destekleyen x86 işlemci mimarisi için tasarlanmış bir tam sanallaştırma çözümüdür [19]. KVM ana sunucuda bulunan işletim sisteminin hipervizör olarak kullanılmasını sağlar ve tam sanallaştırma çözümü olarak işlemci, bellek ve disk alanlarını üzerindeki sanal sunuculara sanallaştırır [13]. Ana sunucuda bulunan ağ ve disk araçlarını ise QEMU ve virtio aracılığı ile sanal sunuculara paylaşmaktadır [6]. KVM, ana sunucu üzerindeki işletim sisteminin üzerinde bir hipervizör olarak çalıştığı için tip 2 hipervizör mimarisine sahiptir. Hipervizör tabanlı sanallaştırmada tip 1 ve

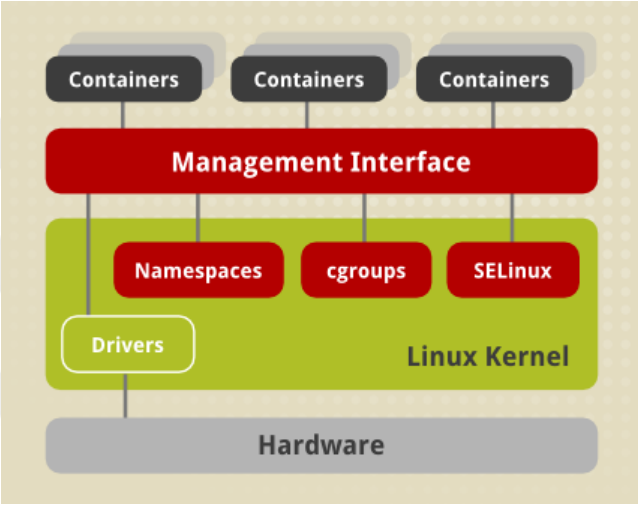
tip 2 hipervizörler arasındaki temel fark hipervizör katmanının konumudur. Tip 1 de hipervizör yazılımı fiziksel kaynaklara direk ulaşırken tip 2 hipervizörlerde ise hipervizör ile fiziksel kaynaklar arasında işletim sistemi bulunmaktadır.

2.2.3.2. Konteyner tabanlı sanallaştırma

Konteynerler, mevcut işletim sistemi üzerinde yeni bir işletim sistemine ihtiyaç duymadan mevcut çekirdek üzerinde bağımsız süreçlere ve isim uzayına sahip yapılardır. Konteyner tabanlı sanallaştırmada hipervizör tabanlı sanal sunuculardan farklı olarak konteynerler kendi işletim sistemine sahip değildir. Ana sunucudaki işletim sistemi üzerinde konteynerler yeni ve izole süreçlere sahiptir [8]. Konteyner mimarisinde, işletim sistemi çekirdeğinin isim uzayı oluşturma özelliğini kullanarak birbirinden izole birden fazla başlangıç süreci oluşturabilir. Kendi başlangıç sürecine sahip her isim uzayını konteyner olarak tanımlayabiliriz. Konteynerler kendi süreçlerine, adres alanına ve dosya sistemine sahiptir [8]. Konteynerler ana işletim sisteminde süreç olarak çalıştığı için ana sunucudaki tüm kaynakları kullanabilmektedir. Bu sebeple control grupları (cgroup) özelliğini kullanarak konteynerlerin kullanabileceği işlemci ve bellek kaynakları sınırlandırılabilmektedir. Konteynerler kendi adres alanına ve grup haklarına sahip olduğu için aynı çekirdek üzerindeki konteynerler dışarıdan erişime sahip değildir. İsim uzayındaki bu izole yapı sayesinde konteynerlerde güvenlik sağlanmaktadır [13].

Konteynerlerin yönetimi için birçok araç bulunmaktadır. LXC [20] ve Docker [21] yaygın olarak kullanılan yönetim araçlarındandır. LXC, linux çekirdeğinde gelen isim uzaylarının yönetimi için kullanılabilen yönetim aracıdır. LXC güçlü bir uygulama programlama arayüzü (API) ve basit araçlar sayesinde kullanıcıların linux sistem ve uygulama konteylerini oluşturmasını ve yönetmesini sağlamaktadır [21]. Docker ise LXC'nin sağladığı avantajlara ek olarak yazılımları kitaplıklar, çalışma zamanı, kodları ve yazılımların çalışması için gerekli olan bileşenleri standartlaştırmaktadır ve birimler halinde paket olarak sunmaktadır. Docker, konteynerlerin yönetimi için basit komutlar da sunmaktadır [22]. Konteyner tabanlı sanallaştırmanın popülerlik kazanmasıyla birçok çözüm geliştirilmeye başlamıştır. Bu bölümde konteyner tabanlı sanallaştırmada en çok bilinen linux konteyner sanallaştırma ve Docker sanallaştırma çözümleri incelenmektedir.

Linux konteyner sanallaştırma çözümü; işletim sistemi üzerinde izole edilmiş bir veya birden fazla işlemler olarak tanımlanabilir. İlk olarak imaj tabanlı devreye alım metodları ile birlikte izole olarak devreye alınabilen açık kaynak bir uygulama paketleme ve dağıtım teknolojisi olarak ortaya çıkmıştır. Linux konteyner işletim sistemi çekirdeğinin sağladığı temel teknolojileri kullanılır. Kaynakların yönetimi için kontrol grupları, işlem izolasyonu için isim uzayları, güvenlik için SELinux özelliklerini kullanmaktadır. Linux konteyner mimarisi Şekil 2.4.'da gösterilmektedir. Şekil 2.4.'da görüldüğü gibi linux konteynerlerin sorunsuz çalışması için tüm bileşenler linux çekirdeğinde sağlanmaktadır [23].



Şekil 2.4. Linux konteyner mimarisi [23]

Linux konteyner mimarisini oluşturan bileşenler aşağıdaki gibi açıklanmaktadır.

İsim uzayı (namespaces); linux çekirdeği her bir konteyner için ayrı bir isim uzayı oluşturulmasını ve işlem izolasyonu sağlanmasını destekler. İsim uzayı işletim sistemindeki ana işlemlerden ayrı ve izole olarak oluşmuş bağımsız işlemlerden oluşmaktadır. İşlemler bağımsız olduğu için aynı kaynakta bulunan ve farklı isim uzaylarında yer alan işlemler ile çakışma yaşamaz. İsim uzaylarının işlem ve kaynağa göre farklı tipleri bulunmaktadır. Konteynerlerin kendilerine ait ve izole dosya sistemi için isim ve alan adı, bellek kullanımı için işlemleri arası haberleşme, izole işlemlerin yönetimi ve izole ağ kontrollerinin yönetimi için ise isim uzayı tipleri bulunmaktadır.

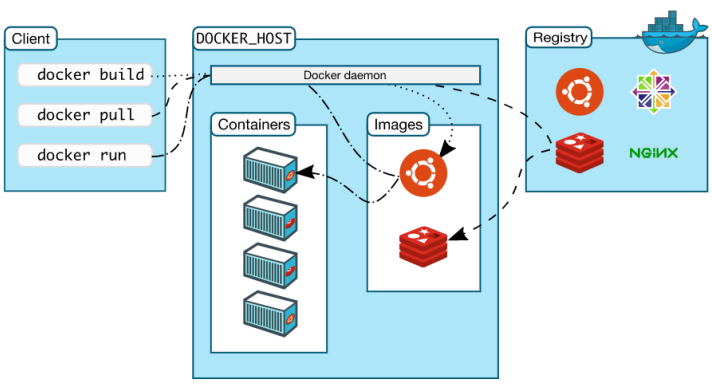
Kontrol grupları (cgroups); linux çekirdeği sistem kaynaklarını yönetmek ve işlemleri gruplandırmak için kontrol gruplarını kullanır. Kontrol grupları işlemci, bellek ve ağ

kaynaklarının yönetimi yapar. Linux konteynerlerin kaynaklarını yönetmek ve sınırlandırmak için linux çekirdeğinin özellik olarak sağladığı bu kontrol grupları kullanılmaktadır.

SElinux; güvenlik politikaları ve etiketlerini uygulayarak konteynerlerin kendileri ve ana işletim sistemi ile arasında etkileşimi güvenli hale getirir.

Yönetim arayüzü; konteynelerin yönetimine imkan sağlayan linux çekirdeğinin üstünde bulunan bir araçtır.

Docker sanallaştırma çözümü, işletim sistemi seviyesinde sanallaştırmayı sağlayan konteyner tabanlı bir sanallaştırma çözümüdür. Docker temelde linux konteyner teknolojisini kullanmaktadır ve linux konteynerlerin sağladığı avantajları kapsamaktadır. Docker uygulamaların geliştirildiği, taşındığı ve yürütüldüğü açık kaynak geliştirilmiş ortamlardır. Docker uygulamaları altyapıdan bağımsız olarak geliştirmeyi ve hızlı bir şekilde geliştirilen yazılımların yürütülmesini amaçlar. Docker'ın temel avantajı yazılım taşıma, geliştirme ve devreye alma süreçlerini hızlandırarak, bir yazılım ürününün geliştirilmeye başlamasından canlı ortamda çalışabilir duruma geldiği adıma kadar oluşan süreçleri hızlandırmaktır [21].



Şekil 2.5. Docker sanallaştırma mimarisi [21]

Şekil 2.5.'de görüldüğü gibi docker mimarisi Docker sunucu ve istemcisi, Docker imajları, Docker kayıt defteri ve Docker konteynerleri olarak dört ana bileşenden oluşmaktadır.

Docker sunucu ve istemci katmanı; Docker istemicisi ile Docker sunucusu programlama arayüzü (RESTFul API) aracılığıyla birbirleri ile bağlantı kurmaktadır.

Docker istemcisi ile Docker sunucusu aynı ortamda veya farklı ortamlarda olabilir. Docker istemcisini ilk olarak Docker servisi karşılar. Docker istemcisi, Docker servisi üzerinden komutlarını ileterek talebini gerçekleştirir [14].

Docker imajları; bir Docker konteyner oluşturmak için gerekli talimatları içeren salt okunur bir şablondur. Bir imaj üzerine konfigurasyon yapılarak yeni bir imaj oluşturulabilir. İmaj içeriğinde konteynerin veya konteyner üzerindeki uygulamanın gereksinim ve kütüphaneleri tutulmaktadır. Kullanıcılar kendi imajlarını oluşturabilir veya halka açık kayıt defterlerinden imajları indirebilirler.

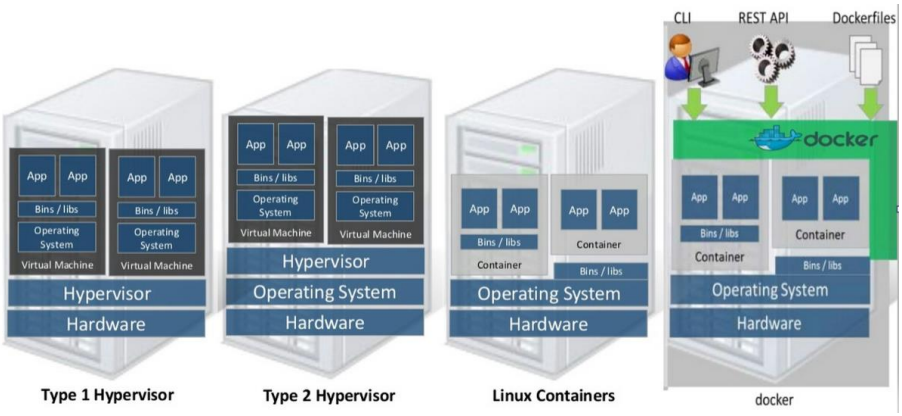
Docker kayıt defteri; docker imajların, imajlara ait versiyon, geliştirici gibi bilgilerin saklandığı halka açık veya özel depolama alanlarıdır. Docker Hub, Docker firmasının sunduğu halka açık Docker kayıt defteri alanıdır. Kullanıcılar ihtiyaçlarına göre kendi özel Docker kayıt defteri alanlarını oluşturabilir [14].

Docker konteynerler; Docker imajlarını kullanılarak oluşturulan konteyner yapılarıdır. Üzerinde bulunan uygulamanın tüm bağımlılıklarını ve kütüphanelerini bulundurmaktadır. Bir Docker konteynerin çalıştırılması için istemci tarafından üç temel süreç başlatılmaktadır. Docker yapılandırılır, kayıt defterinden docker sunucuya imaj indirilir ve Docker konteyner çalıştırılır.

2.2.4. Sanallaştırma çözümlerinin karşılaştırılması

Hipervizör tabanlı sanallaştırma, linux konteyner ve Docker mimarilerinde sanallaştırmanın uygulandığı katman, teknoloji ve erişim arayüzleri noktasında farklar bulunmaktadır. Konteyner sanallaştırma ile hipervizör sanallaştırma arasındaki en temel fark sanallaştırma katmanının üstünde hipervizör tabanlı sanallaştırmada bir işletim sistemi bulunurken konteyner tabanlı sanallaştırmada ana sunucudaki işletim sistemi kullanılmaktadır.

Şekil 2.6.'de görüldüğü gibi hipervizör, donanım katmanındaki kaynakları sanallaştırarak sanal sunuculara paylaştırmaktadır. Uygulama bağımlılıkları ve kütüphaneler için hipervizör katmanın üstünde ek işletim sistemi gerekmektedir. Konteyner tabanlı sanallaştırmada ise konteynerler sadece uygulamanın gereksinim duyduğu temel dosya ve kütüphaneleri içermektedir. Diğer gereksinimleri ana sunucudaki işletim sistemi çekirdeği üzerinden sağlanmaktadır.



Şekil 2.6. Hipervizör, Linux konteyner ve Docker mimarisi [24]

2.2.5. Sanallaştırma teknolojilerinin faydaları

Sanallaştırma teknolojileri kullanım kolaylığı, erişebilirlik ve maliyet gibi birçok etkende sağladığı faydalar sebebiyle popülerliğini artırmıştır ve kullanım oranında artış hızla devam etmektedir. Kurum ve kuruluşlar tarafından veri merkezlerinde ve kullanıcılar tarafından kişiler bilgisayarlarda sanallaştırma teknolojilerinin sağladığı faydalar aşağıdaki gibi sıralanabilir.

- Bir fiziksel sunucu üzerine birden fazla sanal sunucu kurulabilmektedir.
- Fiziksel sunucularda atıl kaynak kullanımının önüne geçilmesi ile aynı iş yükü daha az fiziksel ortam ile karşılanabilmektedir.
- Sanallaştırma teknolojileri veri merkezlerinde konsolidasyon sağlanarak fiziksel sunucuların operasyonel, bakım ve toplam sahip olma maliyetlerinin düşürülmesini sağlamaktadır.
- Fiziksel sunucular sanallaştırma yazılımları ile ortak bir havuzda toplanabilir. Havuzdaki fiziksel donanımların yedekliliği sağlanarak sanal sunucuların erişilebilirliği artırılmaktadır. Kaynak artırımı, yedekleme ve sanal sunucuların taşınma işlemleri sanallaştırma yazılımları ile hizmette kesinti olmadan dinamik olarak yapılabilmektedir.
- Uygulamaların gereksinimlerine göre sanal sunucuların işletim sistemi ve versiyonları ana sunucudan farklılık gösterebilmektedir.
- Sanallaştırma teknolojileri ile sistemin anlık görüntüsü alınarak hata ve problem durumunda sistem çalışılabilir noktasına geri getirilebilir.

2.2.6. Konteyner sanallaştırma teknolojilerinin yönetimi

Konteynerler, düşük kaynak kullanan ve kolay devreye alınabilen yapılardır. Bir fiziksel sunucu üzerinde yüzlerce konteyner çalışabilmektedir. Fiziksel sunucu sayımız düşük olsada konteyner sayısının yüksek olması konteynerlerin yönetiminde orkestrasyon araçları kullanılması ihtiyacını ortaya çıkarmıştır. Bu bölümde Docker Swarm, Kubernetes gibi konteyner orkestrasyonu sağlayan araçlar ve uygulama yaşam döngüsü yönetimi ve konteynerleri platform olarak bir arayüzle yönetmeye imkan sağlayan Redhat tarafından geliştirilen Openshift platformu incelenmektedir.

2.2.6.1. Konteyner orkestrasyonu

Konteyner tabanlı sanallaştırma bir fiziksel sunucu üzerinde çok sayıda konteyner çalıştırma imkanı sağlamaktadır; ancak yüzlerce farklı konteyneri aynı fiziksel ortam üzerinden çalıştırmak konteynerlerin üzerinde çalışan uygulamaların erişilebilirliklerinin risk altında olmasına sebep olmaktadır. Bu yapıda fiziksel sunucuda veya işletim sisteminde yaşanan problemden üzerindeki tüm konteynerler etkilenerek birçok uygulamada probleme sebep olacaktır.

Bu sebeple fiziksel kaynaklardan bir havuz oluşturarak konfigurasyon, taşıma, çalıştırma gibi konteyner orkestrasyonu gerçekleştiren araçlar geliştirilmiştir. Konteyner orkestrasyon araçlarından en yaygın olarak kullanılan Docker Swarm ve Kubernetes teknolojileridir.

Docker Swarm, Docker sanallaştırma ortamlarının orkestrasyonunu gerçekleştiren araçtır. Docker Swarm ile veritabanı ve uygulama gibi yapılarımızı çalıştırdığımız konteyner ortamlarını yönetme ve birden fazla fiziksel sunucudan oluşturduğu küme üzerinde konteynerleri taşıma, yük altında konteynerleri ölçeklendirme ve birden fazla örnek oluşturma gibi operasyonları gerçekleştirmektedir.

Güncel Docker versiyonlarında Docker Swarm, Docker motorunda yüklü bir özellik olarak gelmektedir. Docker Swarm teknolojisinin mimarisi Şekil 2.7.'de belirtilmiştir.

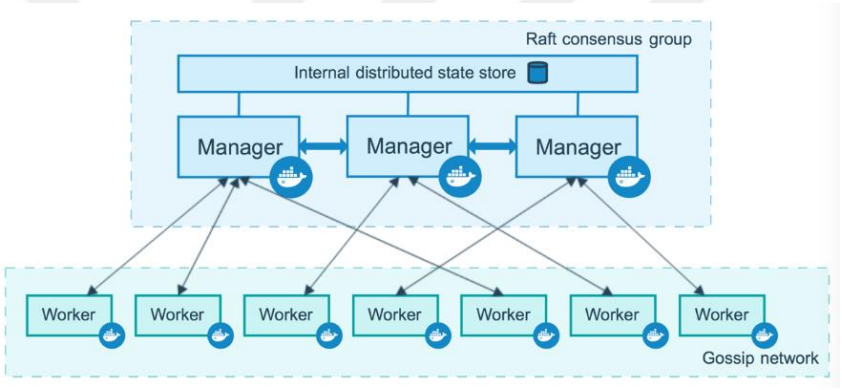
Docker Swarm mimarisi yönetici (manager) ve işçi (worker) bileşenlerinden oluşmaktadır.

Swarm; kubernetes mimarisindeki küme ile aynı yapıdadır. Swarm, bir veya birden fazla fiziksel veya sanal sunucuları yöneterek küme yapısı oluşturur [26].

Yönetici; kümenin erişilebilirliğini koruma, servisleri zamanlama, HTTP API (programlama arayüzü) uç noktalarının servis edilmesi gibi yönetim görevlerini gerçekleştirir.

İşçiler; yöneticilerin yönlendirdiği servisleri çalıştırmaktadır. İşçilerin ana görevi konteynerleri çalıştırmaktır.

Servis; hangi konteyner imajının kullanılması, hangi komutun hangi konteynerde çalışması gerektiğini gerçekleştiren bileşendir.



Şekil 2.7. Docker Swarm mimarisi [25]

Docker swarm aracının yetenek ve faydalarını aşağıdaki gibi belirtebiliriz [25].

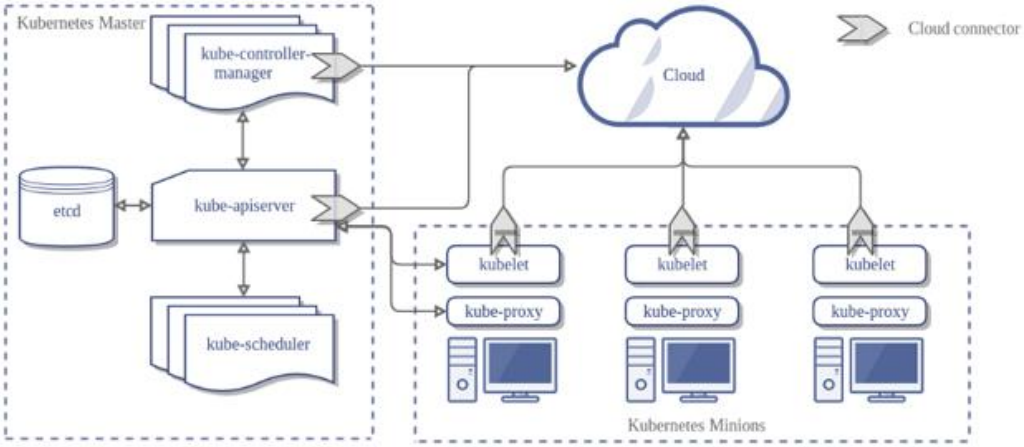
- Docker motoru ile küme yönetimi; ek bir konteyner orkestrasyon yazılımı öğrenmeden Docker komutları ile kümeler yönetilebilmektedir.
- Servis modeli bildirimi; birden fazla konteyneri gruplayarak bir servis olarak tanımlayabilir.
- Ölçekleme; bir uygulama servisine ait konteyner sayısını artırabilir veya azaltabilir.
- Beklenen durum karşılaştırması; konteynerlerin durumunu izleyerek servise ait konteynerlerin birinde problem olması durumunda yeni bir konteyner başlatır.
- Servis keşifi; her servis için bir alan adı sistemi (DNS) kaydı oluşturarak ip yerine servis ismi ile servise ulaşmayı sağlamaktadır.
- Yük dağıtımı; gelen istekleri servise ait konteynerler arasında dağıtmaktadır.
- Güvenlik; tanıma sertifika ekleyerek trafiğin güvenli olarak iletilmesini

sağlamaktadır.

- Güncellemeler; konteynerler üzerindeki uygulamaların güncellemelerini dinamik olarak gerçekleştirebilmektedir. Yeni sürüm uygulamaya sahip konteyneri eski konteyneri kapatarak sıralı olarak kaldırmaktadır.

Kubernetes teknolojisi, konteynerlerin konfigurasyon, taşıma, ölçeklendirme ve otomasyon gibi iş yükü ve servislerini yönetmek için geliştirilen açık kaynaklı bir platformdur. Kubernetes konteyner orkestrasyonu için kullanılan yaygın bir araçtır. Kubernetes projesi 2014 yılında Google firması tarafından açık kaynak olarak geliştirmeye başlanmıştır.

Şekil 2.8.'da görüldüğü gibi Kubernetes mimarisi ana bileşen, düğüm bileşeni ve eklentilerinden oluşmaktadır.



Şekil 2.8. Kubernetes mimarisi [27]

Ana bileşeni (Kubernetes Master); kümenin kontrol panelini sağlanmaktadır. Ana bileşen, kümenin bütününün bileşen ve hizmetlerinin konfigurasyonlarını tutar ve yönetimsel kararlarını verir. Bu hizmetleri gerçekleştirebilmek için "kube-controller-manager", "etcd", "kube-apiserver", "kube-scheduler" ve "cloud-control-manager" bileşenlerini kullanmaktadır.

"kube-controller-manager", düğümleri izleyerek problem durumlarında farketmekte ve konfigurasyonuna göre aksiyon almaktadır. Kümede tanımlı ölçeklendirme birimini denetleyerek konteynerlerin konfigurasyonda belirtilen adette çalışmasını sağlamaktadır. Servis ve pod gibi bitiş noktası nesnelerini yönetir. İsim uzayları için

temel hesapları ve programlama arayüzlerini (API) erişim jetonlarını (token) yaratmaktadır.

“etcd”, konfigurasyon veritabanı olarak belirtilmektedir. Kümenin konfigurasyonu ile ilgili verileri tutmaktadır.

“kube-apiserver”, Kubernetes programlama arayüzü (API) bileşenini master üzerinden dış ortama açan bileşendir. Kubernetes kontrol panelinin ön yüzü olarak çalışmaktadır.

“kube-scheduler”, konfigurasyona göre başlatılmak istenen konteynerler için düğümü seçen ve konteyneri başlatan yapıdır.

“cloud-control-manager”, temel bulut sağlayıcı ile etkileşimde bulunan bileşenlerin yönetimini sağlamaktadır. Kümeye ait düğümlerin hizmeti kesmesi durumunda düğümün durumunu denetlemek, kümenin dış ortama erişmesi için rota oluşturmak, küme içerisinde yüklerin dağıtımını yapan bileşenlerin yönetimini sağlamak ve konteyner ile hizmetlerin kullandığı saklama alanlarının yönetimini gerçekleştirmek bu bileşenin temel görevleridir.

Düğüm bileşeni (kubernetes node); düğüm bileşeni kümeye ait tüm düğümlerde çalışarak düğüm üzerinden hizmet veren konteyner ve çevre birimlerin erişilebilirliğini sağlamaktadır. “kubelet”, “kube-proxy” ve “container runtime” bileşenlerinden oluşmaktadır.

“kubelet” bileşeni, düğümler üzerinde bir ajan olarak çalışarak düğüm üzerindeki konteynerleri içeren pod’ların sağlığını ve çalışabilirliğini denetlemektedir. Bileşen sadece Kubernetes tarafından çalıştırılan konteynerlerin kontrolünü sağlamaktadır. “kube-proxy” bileşeni, düğümün ağ kurallarını uygulayarak bağlantıların yönlendirmesini gerçekleştirir. “container runtime” bileşeni, konteynerlerin çalışmasından sorumlu yazılımdır. Kubernetes Docker, containerd, cri-o ve rktlet gibi konteyner çalışma bileşenlerini destekler.

Eklentiler; kubernetes alan adı sistemi (DNS), arayüz paneli, konteyner kaynak izleme ve küme seviyesi loglama eklentilerine sahiptir [27].

Openshift konteyner platformu, Docker konteynerleri kullanan, uygulamaları devreye alıp yönetebilen, halka açık, özel veya karma bulut ortamlarında konumlandırılabilen servis olarak platform hizmeti sunan bulut bilişim yazılımıdır. Red Hat firması tarafından geliştirilmektedir. Açık kaynak kodlu ve Red Hat tarafından desteği verilen ücretli versiyonları mevcuttur. Altyapısında Docker ve Kubernetes kullandığı için Docker ve Kubernetes'in sağladığı tüm avantajları kapsamaktadır. Openshift konteyner platformu, son dönemlerde popüler olan devops, mikroservis mimarisi, konteyner ve bulut bilişim kavramlarını tek çatı altında toplayarak kullanıcılara servis olarak platform hizmeti vermektedir.

Openshift konteyner platformu mimarisi Şekil 2.11.'de gösterilmektedir. Openshift mimarisinin bileşenlerini; altyapı bileşenleri, temel bileşenler, ek bileşenler, ağ bileşenleri ve servis katalog bileşenleri olarak beş ana başlıkta incelenebilir [28].

Altyapı bileşenleri; Kubernetes altyapısı, imaj kayıt defteri ve internet arayüzü altyapı bileşenlerini oluşturmaktadır.

Openshift konteyner platformu, kubernetesin içerdiği ana bileşen, düğüm bileşeni ve eklentileri kapsamaktadır. Bu bileşenler bölüm 3.5.1.2'de belirtilmiştir.

Openshift konteyner platform üzerinde kendi kayıt defterini bulundurmaktadır. Buna ek olarak üçüncü parti veya halka açık kayıt defterleri de kullanılabilir.

Openshift konteyner platformunun kullanıcılara sağladığı avantajlardan en önemlilerinden biri de platform üzerinde kullanıcıların bütün operasyonlarını gerçekleştirebilecekleri internet arayüzü sağlamasıdır. Web arayüzü ile kullanıcılar kendi kullanıcı alanlarında uygulamalarını çalıştırmak ve hizmet vermek için tüm konfigurasyonlarını gerçekleştirebilirler. Openshift konteyner platformunu kullanan küme yöneticileride web arayüzünü kullanarak platformun bileşenlerini yönetebilir.

Temel bileşenler; bu katmandaki birçok obje Docker ve Kubernetes teknolojilerinin sağladığı objelerdir. Openshift konteyner platformu bu objelere eklentiler sağlayarak daha fazla özelliklere sahip bir geliştirme yaşam döngüsü sağlamaktadır. Temel bileşenleri konteyner ve imajlar, pod ve servisler, projeler ve kullanıcılar, yapı ve imaj akışları ve dağıtımlar olarak belirtebiliriz.

Konteyner ve imajlar, Openshift platformunun uygulamaların çalışması için devreye alınan en temel yapılarıdır. Konteyner ve imaj tanımlarına bölüm 3.2.2.2'de yer verilmiştir.

Pod, bir veya birden fazla konteyneri barındıran ağ ve depolama alanlarına erişebilen en küçük küme bileşenidir. Servis, küme içerisinde altında tanımlı olan podlara erişimleri dağıtan bir yük dağıtıcıdır. Pod'ları gruplandırarak küme içerisinde diğer podlarla veya dışarı erişim için yönlendirici ile bağlantısını sağlamaktadır.

Kullanıcılar platform üzerinde işlemleri gerçekleştirmek için kullanılırlar. Görevlerine göre normal kullanıcı, sistem kullanıcısı ve servis kullanıcısı olarak kullanıcılar sınıflandırılmaktadır. Openshift konteyner platformunda, Kubernetes teknolojisinde yer alan isim uzayı kavramına ek özellikler eklenerek projeler tanımlanmıştır. Projeler, platform üzerindeki kullanıcıların birbirinden izole olarak platformu kullanmalarını sağlamaktadır.

Yapı, kaynak kodu veya giriş parametlerini çalıştırılabilir bir imaja dönüştüren bileşenlerdir. İmaj akışları, imajların verilerini bulundurmaz ancak imajların görüntüsü tutarlar. İmaj stream, imaj etiketi ile imajları sınıflandırmaya yaramaktadır. İmaj akışı takip edilebilir bir yapı oluşturulabilir ve dağıtımı hazırlanabilir.

Dağıtımlar, konteynerlerin, Pod'ların ve servislerin konfigurasyonlarını, nasıl çalışacağı bilgileri gibi dağıtımın tüm yaşam döngüsünü tutmakta ve yönetmektedir [28].

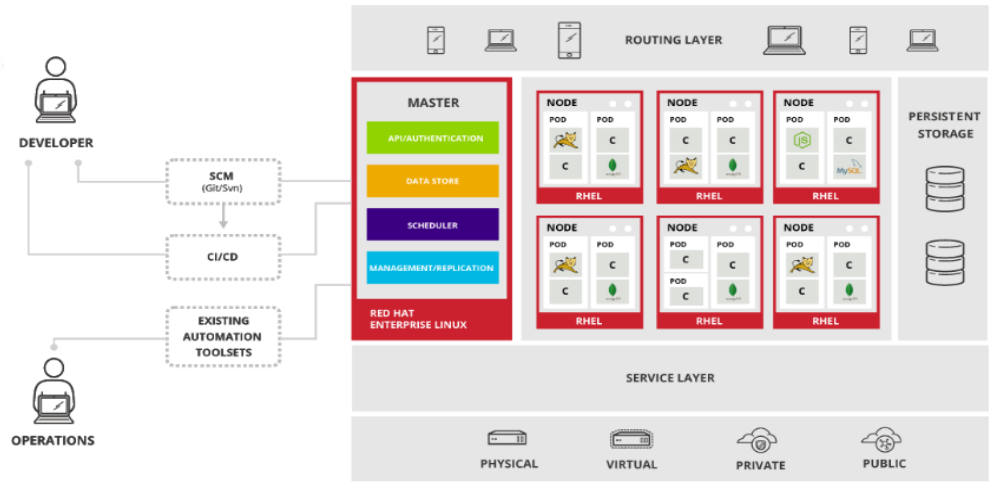
Ek bileşenler; Openshift konteyner platformunda ek bileşenlerde, platforma erişimin yöntemi, kullanıcıların yetkilendirmesi, kalıcı saklama alanlarının yönetimi, konteynerlerin kullandığı geçici alanların yönetimi, kaynak kod kontrol yönetimi ve uygulama programlama arayüzlerinin yönetimi yapılmaktadır.

Ağ bileşenleri; Openshift konteyner platformunda ağ bileşenleri küme içindeki ağın yönetimini ve kümenin dış dünya ile erişimini yönetmek için kullanılmaktadır. Openshift konteyner platformunda Pod'lara ip adresi atanarak dış ortamla direkt olarak Pod'ların haberleşmesi önerilmemektedir.

Bir Pod'u veya birkaç Pod'u kapsayan servisler tanımlanarak ve bir alan adı sistemi (DNS) adresi atanarak pod'ların dış ortamla haberleşmesi sağlanmaktadır. Ağ bileşenleri DNS sunucusu görevi görerek küme içerisinde DNS tanımlarını gerçekleştirmektedir.

Openshift konteyner platformu, küme içerisindeki bileşenler arasında haberleşmeyi sağlamak için yazılım tanımlı ağ kullanmaktadır. Küme içerisindeki ağ yönetimi yazılım tanımlı ağ bileşeni ile gerçekleştirilmektedir. Openshift konteyner platformu üzerindeki uygulamaların küme dışı ile haberleşmesi için yönlendiriciler kullanılmaktadır. Tanımlı olan servis ve uygulamaya dış dünyadan erişimi yönlendirici bileşenleri ile sağlanmaktadır [28].

Servis katalog bileşenleri; bulut temelli platformlarda mikroservis temelli uygulamalar geliştirirken farklı kaynaklar sağlamanın birçok yöntemi vardır. Servis katalog bileşeni ile kullanıcıların Openshift ortamında geliştirdiği uygulamasını çeşitli servis komisyoncularına (service broker) bağlamasını sağlamaktadır [28].



Şekil 2.9. Openshift konteyner platformu mimarisi [28]

Genel tanımı ile Openshift konteyner platformu geliştiricilerin imaj haline getirdikleri uygulamalarını bir arayüz ile dağıtma, test etme ve canlıda çalıştırma gibi uygulama yaşam döngüsünün tamamını yönetebilecekleri, tüm gereksinimlerini altyapı kurulumu gerektirmeden karşılayabilecekleri platform olarak servis hizmeti sunmaktadır.

2.3. Uygulama Geliştirme Süreçlerinin Sanallaştırma Teknolojileri ile İlişkisi

Bulut bilişim ve konteyner çözümlerinin popüler hala gelmesi uygulama geliştirme süreçleri ve yapılarında da değişikliğe gidilmesi ihtiyacını ortaya çıkarmıştır. Bulut bilişim ve konteyner teknolojilerinin popülerliğinin artması ile uygulama geliştirme süreçlerinde devops metodolojisi, uygulama mimarisinde ise mikroservis mimarisi yaygınlaşmaya başlamıştır.

DevOps modelinde uygulama geliştiren ekipler ile uygulamaları yöneten operasyon ekipleri birbirinden izole değildir. Bu modelde tüm ekipler tek bir ekip olarak düşünülebilir, geliştirme, test, dağıtım ve operasyon sürecine kadar uygulama yaşam döngüsünün tüm katmanlarında çalışabilir. Bu sayede ekipler kısıtlamaların önüne geçerek ürün ortaya çıkarma sürecini hızlandırır. Üretimde yer alan ekiplere ek olarak kalite ve güvenlik güvence ekipleride entegre olarak çalışırsa bu modele DevSecOps denir.

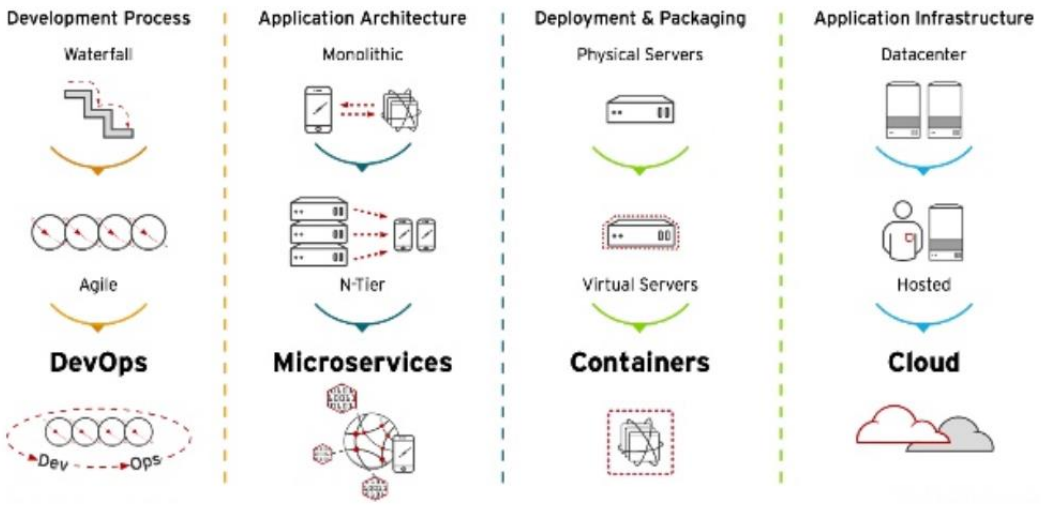
DevOps modelinde çalışan ekipler ürünün geliştirilmeye başladığı noktadan canlıda çalışacağı noktaya kadar gelen süreçte eski süreçlerde el ile ilerleyen adımları otomatik olarak gerçekleştirmeyi amaçlar. Bu gereksinim ile otomasyona imkan sağlayan birçok DevOps ürünü geliştirilmeye başlanmıştır.

DevOps yazılım geliştirme sürecinin uygulamaları aşağıdaki şekilde belirtilmektedir [30].

- Sürekli entegrasyon; geliştiricilerin merkezi bir depo alanında kod değişikliklerini sakladığı ve otomatik derleme ve test süreçlerini gerçekleştirdiği yazılım geliştirme uygulamalarıdır. Bu süreçte, uygulamalardaki hataları hızlı bir şekilde tespit edip gidermek, yazılımın kalitesini geliştirmek ve güncellenmelerin yayınlanması için geçen süreyi azaltmak hedeflenmektedir.
- Sürekli teslim; kod tarafında yapılan değişikliklerin otomatik olarak derlendiği, test edildiği ve canlı ortama yayınlamaya hazırlandığı bir yazılım geliştirme uygulamasıdır. Sürekli entegrasyon sürecinden farklı olarak kod değişikliklerinin derleme aşamasından sonra kodun test ortamına ve üretim ortamına dağıtılması sürekli teslim sürecinin kapsamındadır. Sürekli teslim süreci doğru olarak yapılandırıldığında

geliştiricilerin aynı standartlarda test süreçlerinden geçen ve dağıtıma hazır hale gelmiş yapıları olmaktadır.

- Mikroservisler; mikroservis mimarisi, tek bir uygulamanın büyük veya parçalı yapıya sahip olması yerine uygulamanın küçük servisler şeklinde oluşturulduğu uygulama mimarisi yaklaşımıdır. Her hizmet kendi yaşam döngüsüne sahip olur ve diğer hizmetler ile uygulama programlama arabirimi (API) ile haberleşir. Mikroservislerin gereksinimleri referans alınarak ayrılır ve her servis kendi kapsamında bir amaca sahip olmaktadır. Bir hizmet olarak gruplanmış olan mikro servislerin hepsinin aynı iskelet (framework) veya programlama dilini kullanması kısıtı bulunmamaktadır. Mikroservisler birbiri ile farklı yapıda olabilirler.
- Kod olarak altyapı; altyapının versiyon denetimi ve sürekli entegrasyon gibi yazılım geliştirme teknikleri kullanılarak temin edilen ve yönetilen uygulamalardır.
- İzleme; altyapı ve uygulamalarda hizmetin kalitesini ve performansını garanti altına almak için son kullanıcının aldığı hizmet izlenir ve ölçümlenir.
- İletişim ve iş birliği; DevOps süreci temel olarak ekiplerin kazanması gereken kültürü ve çalışma yöntemini tanımlar. DevOps araçlarının kullanılarak yazılım dağıtım sürecinin otomatik hale gelmesi, geliştirme ve operasyon ekiplerinin iş tanımlarını ve sorumluluklarını birleştirmektedir. Bunun sonucunda kurum ve kuruluşlarda ekipler arası iş birliği ve bilgi paylaşımının yaygınlaşması sağlanmaktadır.



Şekil 2.10. Uygulama geliştirme süreçlerinin değişimi [29]

Konteyner tabanlı yazılım geliştirme süreçlerinin avantajları aşağıda özetlenmektedir.

- Hız; mikroservisler ve sürekli teslim yöntemleri ile ekipler uygulama yaşam döngüsünde yer alan süreç ve sorumlulukları sahiplenmekte ve sürecin hızla ilerlemesine katkı sağlamaktadır.
- Hızlı teslim; sürekli entegrasyon ve sürekli teslim süreçleri ile bir kodun derlenmesinden dağıtımına kadar uygulama yayınlama süreçlerini otomatize ederek uygulamanın canlıya çıkma süreleri kısalmaktadır.
- Güvenilirlik; sürekli entegrasyon ve sürekli teslim ile uygulama yaşan döngüsü otomatikleştirilerek belirli bir standart sağlanmaktadır ve izleme uygulamalarıyla verilen hizmetin kalitesi garanti altına alınmaktadır.
- Ölçeklendirme; uygulama yaşam döngüsünde kullanılan altyapılar kolaylıkla konfigure edilebilir ve gereksinim duyulduğunda kaynaklar artırılabilir veya azaltılabilir.
- Güvenlik; konteyner seviyesinde güvenlik araçları ile kodların ve platformun güvenliği garanti altına alınabilir.
- İş birliği geliştirme; konteyner tabanlı geliştirilen yazılımların yönetilmesi için izlenen DevOps süreçleri ile uygulamanın yaşam döngüsündeki tüm ekiplere sorumluluk bölünmektedir ve ekipler arası iş birliği artmaktadır.

2.3.1. On iki faktör uygulama geliştirme

Son yıllarda uygulamalar servis olarak yazılım veya web uygulaması yapısında servis olarak hizmet verebilmektedir. Servis olarak verilen yazılımların yaşam döngülerinin standartlarını tanımlamak, süreci otomatikleştirerek kod geliştirmeden canlıya alınma sürecine kadar geçen adımları hızlandırmak amacıyla on iki faktör uygulama geliştirme metodu geliştirilmiştir. On iki faktör uygulamalarda, projeye eklenen yeni geliştiricilerin projeye dahil olması için harcanan zaman ve maliyeti en düşüğe indirmek için yazılımların kurulumları otomatikleştirilmektedir. Yazılımlar farklı uygulama çalıştırma ortamlarına taşınabilecek ve sorunsuz çalıştırılabilecek mimaride olmalıdır. On iki faktör geliştirilen uygulamalar, konteyner tabanlı yazılım geliştirme süreçlerini içererek uygulama yaşam döngüsünü standart ve otomatize yapıda geliştirmeye olanak sağlamakta ve canlıya alma süreçlerini hızlandırmaktadır [31].

On iki faktör uygulama geliştirme süreci adımlarını bu bölümde incelenmektedir.

2.3.1.1. Kod tabanı

On iki faktör uygulamalar Git, Mercurial veya Subversion gibi versiyon kontrol sistemleri ile takip edilmektedir. Her uygulamanın kendine ait depolama alanı olmalıdır. Kod tabanı uygulamanın kodlarının merkezi noktada tutulmasını amaçlamaktadır. Her uygulamanın bir adet kod tabanı olmalıdır; ancak uygulamaya ait bir adet kod tabanı kullanılarak uygulama geliştirme, test ve canlı gibi farklı ortamlar için çalıştırılabilir [31].

2.3.1.2. Bağımlıklar

Projelerde uygulamaların gereksinim duyduğu bağımlılıkları proje içerisinde direkt olarak yükleyerek değil konfigurasyonda belirtilerek bağımlılıkların paket yöneticisi ile projeye getirilmesi sağlanmalıdır. Nmp, Maven, Pip, Bundle paket yöneticilerine örnek verilebilir.

2.3.1.3. Yapılandırma

Yapılandırma, bir uygulamanın dağıtımları arasındaki konfigurasyonel farkları içerir. Bir uygulamanın konfigurasyonu, çevresel parametreleri ve erişim bilgileri gibi kodun bir üstündeki uygulamanın mimarisini tanımlamaktadır.

2.3.1.4. Arka plan servisleri

Arka plan servislerine uygulama üzerinden erişim sağlanırken kimlik bilgileri ve DNS kayıtları kullanılmalıdır. Bu çevresel değişkenler yapılandırma dosyasında belirtilmelidir. Arka plan servislerinde güncelleme olması durumunda kodda değişiklik yapılmadan yapılandırma dosyasında değişiklik yapılarak gerekli konfigurasyon güncellenebilir [31].

2.3.1.5. Yapılandır, yayınla, çalıştır

Uygulamanın koddan bilgisinin alınarak canlı ortama alındığı süreç üç adımda tanımlanmıştır. Kaynak koddan Docker imajı gibi bir paket oluşturulur. Oluşturulan paket yapılandırma dosyasındaki konfigurasyon ve çevresel değişkenler ile birleştirilir. Oluşturulan versiyon ile ortam ayağa kaldırılır. Bu süreçler otomasyon araçları ile otomatize olarak kullanılmalıdır [32].

2.3.1.6. Süreçler

Uygulamaların çalıştığı katmanda bir veya birden fazla süreç çalışabilmektedir. Süreçlerin verileri veri tabanları gibi arka plan servislerinde tutulmalıdır. Bir sonraki işlemde aynı sürecin devam edeceği kesin olmadığı için süreçler birbirlerine bağımlı olmamalıdır [32].

2.3.1.7. Port bağlama

On iki faktör uygulamalar işlemleri kendi içerisinde gerçekleştirmektedir. Uygulamaları dış ortam erişilebilir ve hizmet verebilir olmasını sağlamak ve istekleri almak için servisler port bağlama yöntemiyle dış ortama bağlanır.

2.3.1.8. Eş zamanlılık

Uygulamalardaki süreçler yük durumunda çoğaltılabilmek veya yeniden başlatılabilmek için eş zamanlı olarak tasarlanmalıdır. Bu sayede süreçler birbirini etkilemeden bağımsız olarak eş zamanlı çalışabilmekte ve uygulamalarda performans problemleri yaşanmasının önüne geçilmektedir [32].

2.3.1.9. Tek kullanımlık

On iki faktör uygulamalarda süreçler tek kullanımlık olmalıdır ve hızlı bir şekilde sonlandırılabilmeli veya yeniden başlatılabilmelidir. Bu sayede uygulamalar kolaylıkla ölçeklenebilir veya yeni versiyon dağıtımları uygulanabilir.

2.3.1.10. Geliştirme ve canlı ortam benzerliği

On iki faktör uygulamalar geliştirme ve canlı ortam arasındaki benzerliği maksimum düzeyde ve ortamlar arası boşluğu minimum olacak şekilde tasarlanmalıdır. Ortamlar arasındaki farklılaşma; zaman farkı, personel farkı ve farklı araçların kullanılması sebebiyle oluşmaktadır [32].

2.3.1.11. Günlükler

On iki faktör uygulamalarda, uygulamalar günlük verilerini saklamak için ek maliyet harcamamalıdır. Günlük veriler elasticsearch, hadoop gibi merkezi ortamlara aktıralarak bu ortamlarda görüntüleme, arama ve analiz yapılabilmesi gerekmektedir.

2.3.1.12. Yönetici İşlemleri

Tek seferde yapılabilen yönetici işlemleri otomatize edilerek gerçekleştirilmelidir. Süreçlerde yer alan adımlar otomatize edilerek işlerin aynı standart içerisinde gerçekleştirilmesi amaçlanmalıdır.

3. BULGULAR VE TARTIŞMA

3.1. Sanal Sunucu ve Konteyner Sanallaştırma Ortamlarının Performans Karşılaştırması

Bu bölümde, aynı işletim sistemi üzerinde bulunan hipervizör tabanlı sanal sunucu ile konteyner ortamlarının işlemci, bellek ve ağ bant genişliği performansları analiz edilmiş ve elde edilen sonuçlara göre ortamların karşılaştırılması yapılmıştır. Ek olarak, ortamlar için yük talebi oluşturularak, ortamların davranışları izleme araçlarıyla analiz edilmiştir. Konteynerlerin performans karakteristikleri incelendikten sonra, orkestrasyon ve otomasyonları sağlanarak konteynerlerin operasyonel davranışlarını analiz etmek için platform olarak servis çözümü sunan Openshift konteyner platformunun katkısı incelenmiştir. Daha sonra, on iki faktör uygulama geliştirme ilkesine uygun olarak, uygulama platform üzerinde çalıştırılmış ve uygulama yaşam döngüsü adımları otomatize edilmiştir. Bu yapı ile konteyner tabanlı sanallaştırma teknolojilerinin operasyonel katkıları analiz edilerek avantajları tespit edilmiştir.

Bu çalışmada, Docker, linux konteyner ve sanal sunucuların işlemci, bellek ve ağ performanslarını karşılaştırmak için nicel araştırma yöntemi kullanılmıştır. Openshift platformunda, uygulama yaşam döngüsünün incelenmesi ve platformun yetkinliklerinin analizinde ise nitel araştırma yönteminden yararlanılmıştır. Aynı değişken altında konteyner ve sanal sunucu ortamları incelenmiştir. Bu nedenle, çalışmada deneysel araştırma yöntemlerinden yararlanılmıştır.

Kaynakların analizini eşit koşullarda yapmak için, Docker ve LXC linux konteyner ortamlarının kaynakları sınırlanmadan alttaki donanım kaynaklarının hepsinin kullanımı sağlanmıştır. Sanal sunucuya ise atanan kaynak olarak donanımın sahip olduğu tüm kaynaklar verilmiştir. Docker, linux konteyner ve KVM sanal sunucu ortamlarının çalıştırılması için 8. nesil Intel i5-8300H model, 2,3–4 GHz, 1 soket 4 çekirdek 8 parçacık işlemci, DDR4 2666 MHz 8 GB bellek ve 7200 RPM 1 TB disk özelliklerine sahip fiziksel bilgisayar ortamı kullanılmıştır. Ana sunucu işletim sistemi

olarak Centos 7 işletim sistemi kullanılmıştır. Bu işletim sistemi üzerinde Docker, linux konteyner ve KVM sanal sunucu ortamları kurulmuştur.

İşlemci performans testlerini gerçekleştirmek için, 64 bit tam sayılar kullanılarak belirtilen değere kadar asal sayıları hesaplayarak işlemci yükü oluşturan ve açık kaynak olarak sunulan Sysbench aracı kurulmuştur [34]. Ek olarak, Intel'in sunduğu matematik çekirdek kitaplığını kullanarak vektörel işlemler ile işlemci yükü oluşturan Linpack aracı ortamlara kurulmuştur [35].

Bellek performans testleri için, Sysbench aracı ve temel vektörel işlemleri kullanarak bellek üzerinde yük oluşturan stream testi kullanılmıştır [8].

Konteyner ve sanal sunucu ortamlarında işlemci ve bellek yük talebi oluşturmak için C programlama dili ile yazılmış stres aracı kullanılmıştır [38]. Stres aracı ile oluşan yükü analiz etmek için "Cadvisor", "Node Exporter", "Prometheus" ve "Grafana" araçları kurulmuştur. Cadvisor ve Node Exporter araçları ile konteyner ve sanal sunucu ortamlarının kaynak kullanım metrikleri toplanarak, prometheus veri tabanına yazılmıştır. Prometheus veri tabanındaki verilerin görselleştirilmesi için ise grafana aracı kurulmuştur.

Ağ performansı ölçümleri Nuttcp aracı ile gerçekleştirilmiştir [37]. Ana işletim sistemi ile üzerindeki konteyner ve sanal sunucu ortamları arasında veri aktararak ağ bant genişliği ölçümleri yapılmıştır. Ağ gecikmesi ölçümü için ise Netperf aracı kullanılmıştır [38].

Konteyner tabanlı uygulamaların yaşam döngüsünün adımlarının kontrolü için ortama Minishift uygulaması kurularak Openshift platformu sağlanmıştır. Uygulama yaşam döngüsünün adımlarının otomasyonu için ise Openshift platformu üzerine Jenkins uygulaması kurulmuştur.

3.1.1. İşlemci performansı

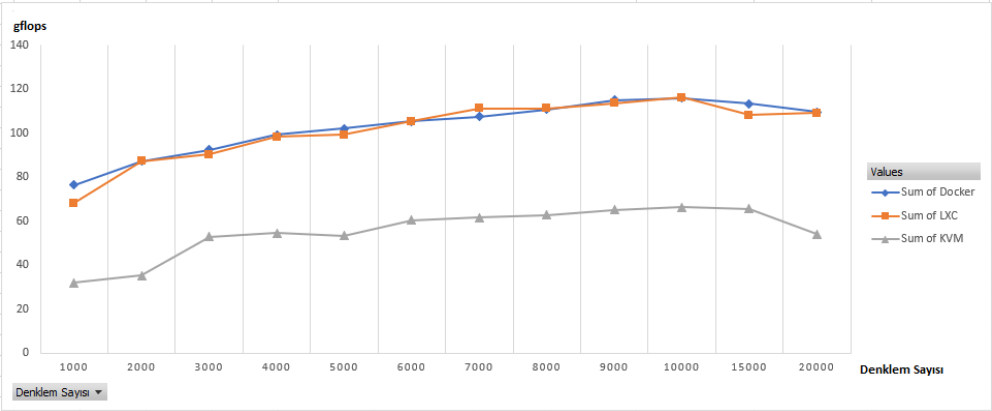
İşlemci performansını kıyaslamak için sysbench ve linpack araçları kullanılarak testler gerçekleştirilmiştir. Sysbench aracı açık kaynak olarak sunulan kıyaslama aracıdır. Sysbench aracı ile işlemci kıyaslama oldukça basit bir yapıya sahiptir. Bu testte 64 bit tam sayılar kullanılarak belirtilen değere kadar asal sayıların hesaplanması ile işlemci

üzerinde yük oluşturulmaktadır. Sysbench aracı ile konteyner ve sanal sunucu ortamlarında bir işlem ve buna paralel sekiz işlem halinde 20000'e kadar asal sayılar hesaplatılmıştır ve hesaplamaların süreleri analiz edilmiştir. Tablo 3.1.'de görüldüğü gibi, analizler sonucunda elde edilen değerler arasındaki fark çok büyük olmasa da konteyner ortamlarının performans sonuçlarının sanal sunucu ortamına kıyasla daha başarılı olduğu tespit edilmiştir.

İşlemci performansını karşılaştırmak için Intel'in sunduğu matematik çekirdek kitaplığı tabanlı Linpack aracı kullanılmıştır. Linpack aracı lineer denklem sistemlerini çözerek işlemci yükü oluşturmaktadır. Hesaplamada kayan noktalı sayılar bir vektör ile çarpılarak yeni bir vektöre eklenmektedir. Tablo 3.1.'de görüldüğü gibi, konteyner ortamlarının performans değerleri sanal sunucu ortamına göre yaklaşık %20 daha yüksektir. Şekil 3.1.'de görüldüğü gibi, Linpack aracı ile hesaplanan denklem sayısını 1000'den 20000'e kadar artırdığımızda konteyner tabanlı sanallaştırma çözümleri tüm aralıklarda %20 - %40 oranında daha başarılı sonuç vermektedir. Tüm ortamlarda ise denklem sayısı artarken performans sonuçlarında lineer bir artış gözlemlenmiştir. Ancak, denklem değeri ortamların limitlerindeki değerlere yaklaştıkça performans sonuçlarında düşme eğilimi gözlemlenmiştir. Linpack testi Sysbench testine göre daha yoğun yük oluşturarak, kıyaslama sonuçlarının daha belirgin olması sağlamıştır.

Tablo 3.1. İşlemci performansı kıyaslama ortalama sonuçları

	Docker	LXC	KVM
Sysbench (1 thread)(s)	10,0015	10,0018	10,0056
Sysbench (8 thread) (s)	10,0012	10,0012	10,0039
Linpack (gflops)	105,2944	103,2719	83,875

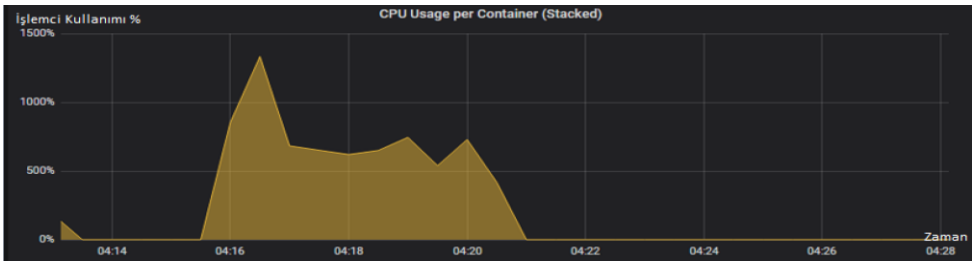


Şekil 3.1. İşlemci performans sonuçlarının denklem sayılarına göre değişimi

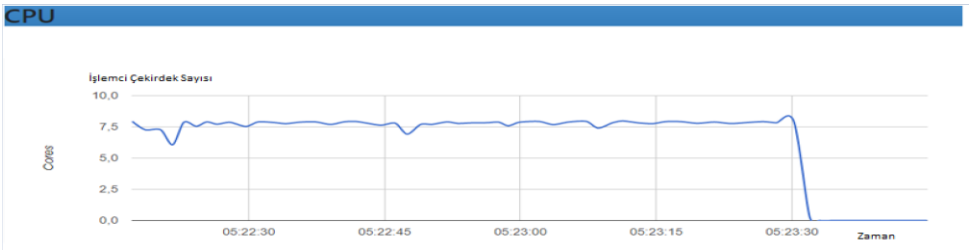
Karşılaştırmada kullanılan Sysbench ve Linpack araçlarının çıktılarına Ek–A bölümünde yer verilmiştir.

C programlama dili ile geliştirilmiş Stres aracı ile ortamlarımızda işlemci iş yükü oluşturulmuştur [36]. Ortamlarımıza beş dakika boyunca sekiz çekirdek kullanılacak şekilde yük verilmiştir. Konteyner ve sanal sunucu ortamları benzer şekilde yükü karşılamış ve ortamlarda herhangi bir problem yaşanmamıştır. Ancak sanal sunucu ortamları kendi işletim sistemine sahip olduğu için üzerindeki kaynağın bir kısmını işletim sistemi için harcamıştır. Konteyner ortamları ise ana sunucunun işletim sistemi üzerinde çalıştığı için eşit kaynaktaki sanal sunuculara kıyasla üzerindeki uygulamaya daha fazla kaynak sunabilmektedir.

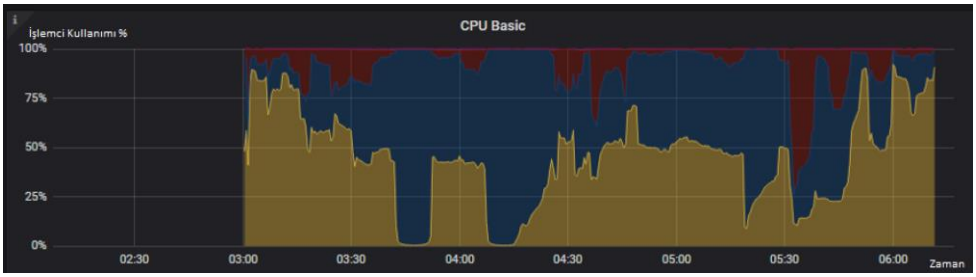
Stres aracı ile Docker, LXC ve sanal sunucu ortamlarına beş dakika boyunca sekiz çekirdeklik bir yük oluşturulmuştur. Ortamların bu test sırasında kaynak yönetimi izlenmiştir. Şekil 3.2., Şekil 3.3. ve Şekil 3.4.'de görüldüğü gibi benzer yük altında ortamlar talebe karşılık verebilmektedir. Sanal sunucu ortamlarında bir işletim sistemi de mevcut olduğu için gelen yük talebine mevcut işletim sistemi kaynağının kullandığı kaynakların dışında kalan kaynaklar kadar yanıt verebilmiştir. Sanal sunucuların işletim sisteminin sanal sunucuya atanan işlemci kaynağının bir kısmını kullanıyor olması, konteynerler ile sanal sunucuların temel farkını oluşturmaktadır.



Şekil 3.2. Benzer yük altında Docker CPU kullanımı (işlemci kullanımı yüzdesi/zaman)



Şekil 3.3. Benzer yük altında Linux konteyner CPU kullanımı (çekirdek sayısı / zaman)



Şekil 3.4. Benzer yük altında sanal sunucu CPU kullanımı (işlemci kullanımı/zaman)

3.1.2. Bellek performansı

İşlemci testinde olduğu gibi, bellek performansı testinde de sysbench aracı kullanılmıştır. Sysbench aracı bellek performansını test etmek için bellekte belirlediğimiz değerde alan ayırmakta ve bu alana okuma veya yazma işlemi yapmaktadır. Bu işlemi, belirlediğimiz değere kadar tekrarlamakta ve saniyede gerçekleştirdiği işlem sayısını çıkartmaktadır. Böylece, bu değere göre kıyaslama imkanı sağlanmaktadır. Tablo 3.2.'de 1Mb boyutundaki blok alanı ile test sonuçları görülmektedir. Performans sonuçları, sanal sunucu ve konteyner ortamları için yakın değerler vermiştir. Konteyner ortamlarında %1-%4 aralığında bellek performans sonuçlarının daha başarılı olduğu görünmektedir. Şekil 3.5.'de görüldüğü gibi, sysbench testinde okuma veya yazma işlemi yapılan bellek miktarı artırıldığında, konteyner çözümlerinin sanal sunucuya göre tüm değerlerde başarılı sonuç verdiği gözlemlenmektedir. Ancak, test edilen veri miktarı artış gösterdikçe sonuçlar birbirine yakın değerler vermektedir. Stream testinde ise bellek bant genişliği değerleri ölçülmüştür. Stream testinde, Tablo 3.3.'de belirtilen temel vektörel işlemlerden yararlanılmaktadır. Bellek bant genişliği değeri aktarılacak verinin büyüklüğü ile doğru orantılıdır. Tablo 3.2.'de belirtilen sonuçlara göre ortamlar için bellek bant genişliği performans testinin sonuçları yakın değerler vermektedir.

Stres aracı ile işlemci yük testine ek olarak, bellek yük testi de gerçekleştirilmiştir. Beş dakika boyunca sekiz gigabayt bellek yükü oluşturacak yük ortamlara sağlanmıştır ve ortamların herhangi bir problem yaşamadan verilen yüke cevap verdiği gözlemlenmiştir. Sanal sunucu ortamlarında konteyner ortamlarından fakrlı olarak işletim sistemi kullanımı olduğu için aynı kaynaktaki konteyner ortamına göre uygulama süreçlerine işletim sisteminin kullandığı kadar daha az kaynak ayırdığı

gözlemlenmiştir. Sysbench aracının çıktılarına ve stream testi sonuçlarına Ek-B bölümünde yer verilmiştir.

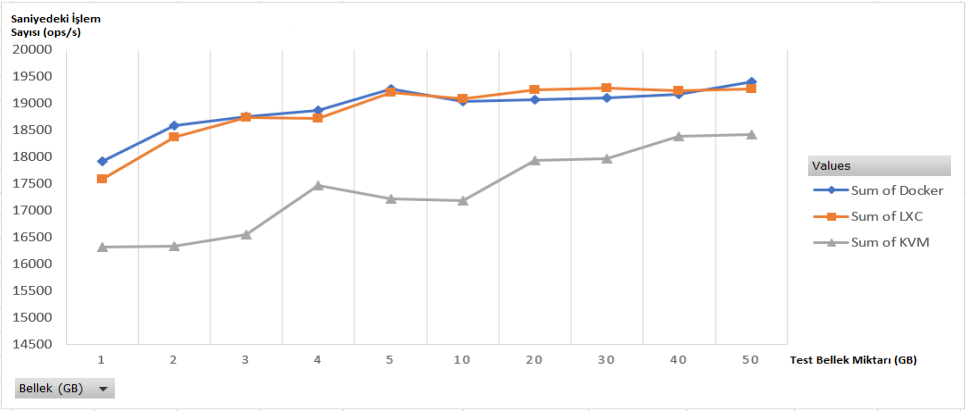
Tablo 3.2. Bellek performans testi ortalama sonuçları

		Docker	LXC	KVM
Sysbench (ops/s)		19038,34	18807,63	18390,66
Stream (Mb/s)	Copy	20801,7	19478,9	20661
	Scale	15140,1	14405,1	14522,9
	Sum	16065,6	15632,2	15633,1
	Triad	16813,4	15599,5	15558,2

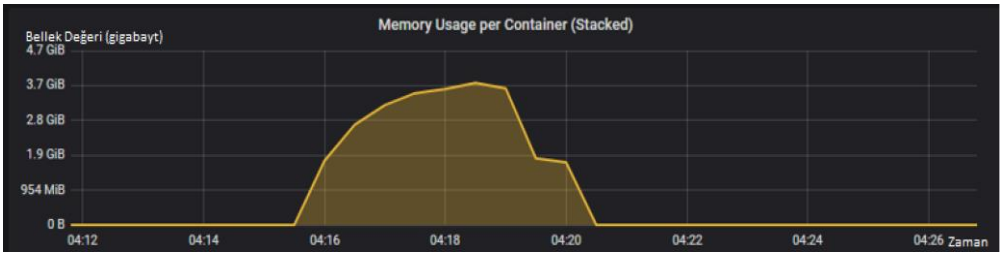
Tablo 3.3. Stream testi işlemleri

İşlem ismi	Çekirdek	Bayt/iterasyon	Flops/iterasyon
Copy	a(i) = b(i)	16	0
Scale	a(i) = q*b(i)	16	1
Sum	a(i) = b(i)+c(i)	24	1
Triad	a(i) = b(i)+q*c(i)	24	2

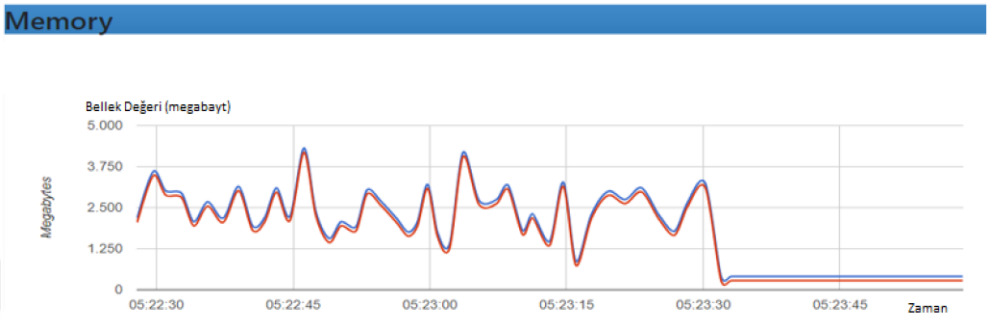
Stres aracı ile Docker, linux konteyner ve sanal sunucu ortamlarına dört gigabayt kaynak yükü sağlanmıştır. Tüm ortamlar bellek talebini karşılamıştır. Ortamların bellek talebine verdikleri cevaplar Şekil 3.6., Şekil 3.7. ve Şekil 3.8.'de gösterilmiştir. Sanal sunucu ortamlarında bulunan işletim sistemi, işlemcide olduğu gibi kendine atanan kaynakların bir kısmını kullanmaktadır. Konteyner tabanlı sanallaştırma çözümlerinin sınırlı kaynaklarda hipervizör tabanlı sanallaştırma çözümlerine göre, üzerlerinde bulunan uygulamaya daha fazla kaynak ayırdığı gözlemlenmiştir.



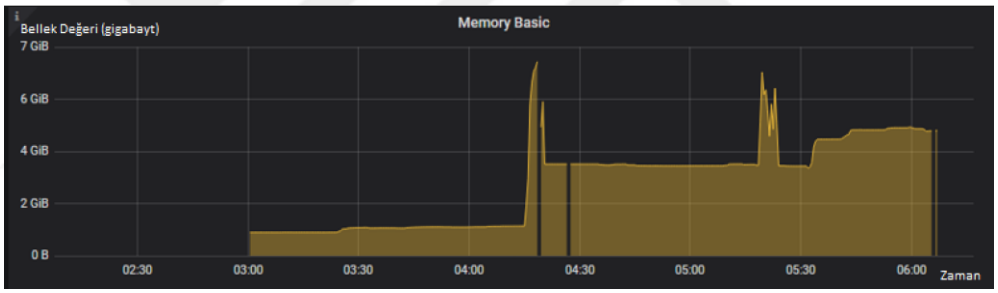
Şekil 3.5. Bellek performans sonuçlarının test bellek sayılarına göre değişimi



Şekil 3.6. Benzer yük altında Docker bellek kullanımı



Şekil 3.7. Benzer yük altında Linux konteyner bellek kullanımı



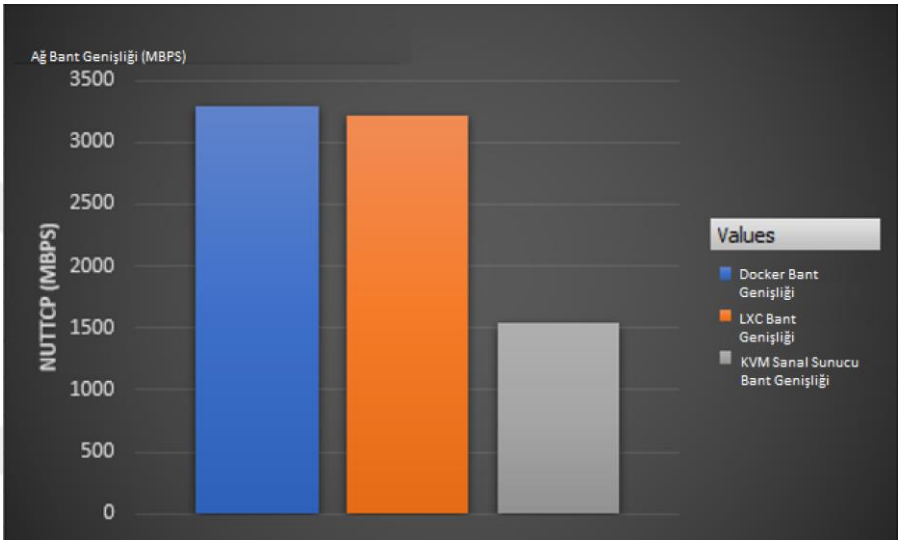
Şekil 3.8. Benzer yük altında sanal sunucu bellek kullanımı

3.1.3. Ağ performansı

Ağ performans ölçümleri için kullanılan nuttcp aracı ile ana sunucumuzdan üzerindeki sanal sunucu ve konteyner ortamlara UDP ve TCP protokelleri ile veri aktarılarak ağ bant genişliği ölçümleri yapılmıştır. Şekil 3.9.'da elde edilen sonuçlara göre, sanal sunucu ortamının ağ bant genişliği değeri konteyner ortamlarına göre daha düşük çıkmıştır. Konteyner ortamları ana sunucumuzun işletim sisteminin üzerinde çalışmaktadır. Sanal sunucu katmanında ise ara katman olarak hipervizör bulunmaktadır ve KVM ortamımızda ağ arabirimi için Virtio servisi kullanılmaktadır. Bu nedenle, beklediğimiz gibi sanal sunucu ortamı ile ana sunucu arasındaki ağ bant genişliği değeri konteynerlere oranla daha düşük olarak tespit edilmiştir.

Ağ gecikmesini ölçmek için Netperf aracı kullanılmıştır. Testler ana sunucu ile sanal

sunucu ve konteyner ortamları arasında gerçekleştirilmiştir. Netperf aracını kullanarak istek gönderdiğimiz ana sunucudan yüz bayt boyutunda istek gönderilmekte ve iki yüz bayt değerinde yanıt geri alınmaktadır. İstemciden yanıt dönene kadar diğer isteği yollamak için beklenmektedir. Bu süreç bir işlem döngüsünü belirtmektedir. Tablo 3.4.'de belirtilen sonuçlara göre ağ bant genişliği testinde olduğu gibi sanal sunucuların performans sonuçları konteyner ortamlarına göre düşük değerlerde çıkmıştır. Sanal sunucu ortamındaki ağ gecikmesinin konteyner ortamlara göre daha fazla olduğu gözlemlenmiştir.



Şekil 3.9. Ağ bant genişliği testi ortalama sonuçları

Tablo 3.4. Ağ gecikmesi testi ortalama sonuçları

Netperf	Docker	LXC	KVM
TCP_RR (µs)	59,063	57,125	180,314
UDP_RR (µs)	58,014	52,328	174,328

Ağ performansı ölçümleri için kullanılan Nuttcp ve Netperf araçlarının komutlarına ve sonuç değerlerine Ek-C bölümünde yer verilmiştir.

3.2. Konteyner Tabanlı Sanallaştırma Teknolojilerinin Operasyonel Faydaları

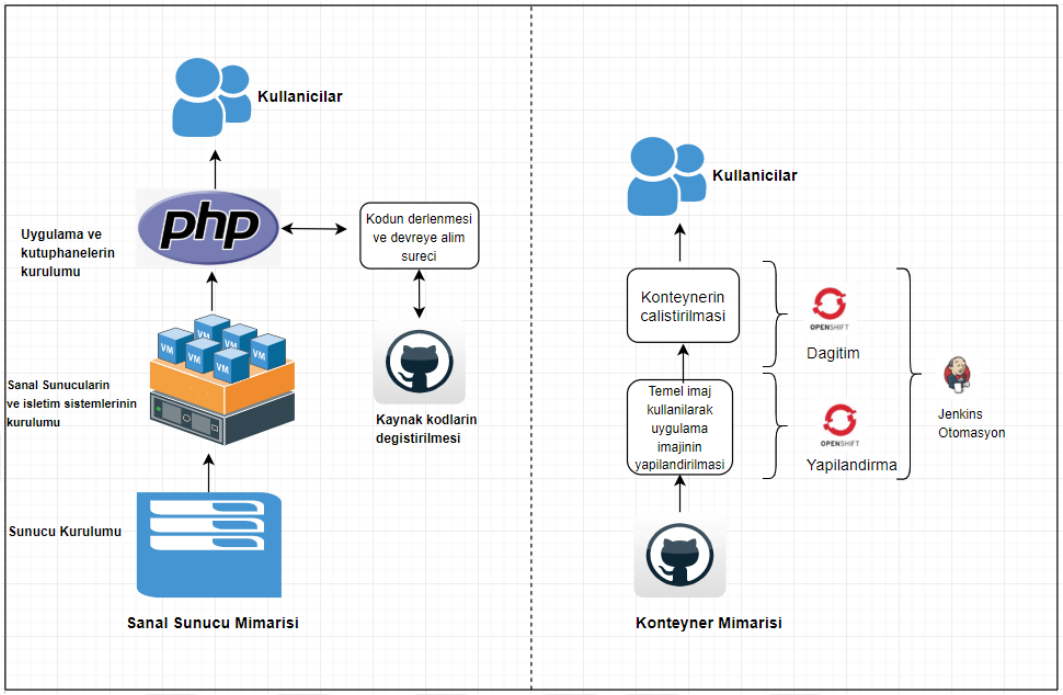
Bu bölümde, konteyner tabanlı sanallaştırma teknolojilerinin performans kazanımlarına ek olarak, uygulamanın geliştirilme aşamasından canlı ortama gelme aşamasına kadar yer alan süreçlerdeki katkısı incelenmiştir. On iki faktör uygulama geliştirme süreçleri takip edilerek, Openshift konteyner platformunda bir web uygulamasının geliştirme sürecinden kullanıcılara hizmet verme durumuna geçme

sürecine kadar yer alan adımlar otomasyon ve orkestrasyon kullanılarak gerçekleştirilmiştir. Fiziksel ve sanal sunucu mimarilerinde uygulama yaşam döngüsü sunucunun, işletim sisteminin, uygulamanın ve kütüphanelerinin kurulması ve her değişiklikte uygulamanın yeni versiyonunun yüklenmesi adımlarını içermektedir. Konteyner mimarisinde ise uygulama kodlarının yapılandırılarak, konteyner imajı oluşturulması, imajın dağıtımı ve konteynerin çalıştırılması adımlarını içermektedir. Konteyner teknolojilerinin altyapı kurulum ihtiyacının olmamasından dolayı operasyonel işlemlerden ve kaynak maliyeti yönünden daha avantajlı olduğu gözlemlenmiştir. Temel imaj kullanılarak merkezi depoda tutulan kodlarımız ile konteyner yapılandırılmakta ve bunların Openshift konteyner platformu üzerine dağıtımı saniyeler içerisinde yapılarak çalıştırılabilmektedir. Şekil 3.10.'da sanal sunucu ortamında ve konteyner ortamlarında örnek uygulama yaşam döngüsü adımları belirtilmiştir. Konteyner mimarisinde uygulamanın kullanıcılara hizmet vermesine kadar yer alan süreç otomasyon ile ilerlemekte ve konteyner ortamlarının ölçeklenebilir olması sebebiyle güncel versiyon yükleme işlemleri de aynı şekilde otomasyon ile kullanıcıları etkilemeden yapılabilmektedir.

Openshift konteyner platformu Gitlab, Github, Bitbucket gibi merkezi olarak yönetilebilen versiyon kontrol sistemleri ile entegre olabilmektedir. Her uygulamamız için Github üzerinde ayrı bir kod tabanı kullanılmıştır. Uygulamamızı Openshift konteyner platformuna yüklemek için redhat.io, docker.io gibi halka açık kayıt defteri ortamları kullanılabilir. Bu ortamların yanında nexus uygulaması gibi kendi ortamımıza konumlandırabileceğimiz kayıt defteri ortamları da kullanılabilir. Kod üzerinde değişiklik yapıldığında, bu değişikliğin platform üzerindeki uygulamada devreye alınması amaçlandığı için uygulamanın kodlar merkezi olarak Github kod tabanına yüklenmiştir.

Kod tabanında uygulamanın kodları kullanılarak, uygulama Openshift konteyner platformunda bağımlılıklarını, konfigurasyonunu ve çevresel parametlerini içerecek şekilde yapılandırılmıştır. Bunun için yapılandırma konfigurasyonları kullanılmaktadır. Bu konfigurasyon ile uygulama Openshift platformu üzerine yüklendiğinde pod'lar üzerinde uygulama ayağa kalkmaktadır. Uygulamanın platform içerisi ve dış ortam ile haberleşmesi için; arka plan servislerinin tanımları ve dış ortam

ile uygulamanın haberleşmesini sağlayan port bağlama tanımları otomatik olarak Openshift platformu tarafından temel konfigurasyon ile yapılabilmektedir.



Şekil 3.10. Sanal sunucu ve konteyner mimarisi uygulama yaşam döngüsü çalışması

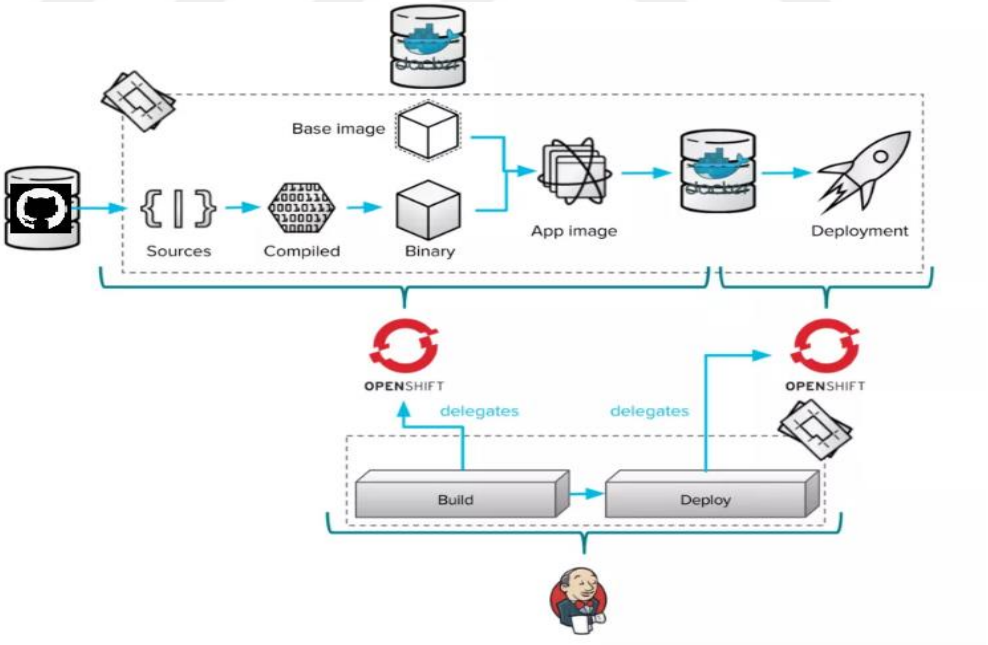
Sürekli entegrasyon, sürekli teslim süreçlerine uygun olarak kod üzerinde bir değişiklik yapıldığında, Openshift konteyner platformunda imajın yapılandırılmasını ve uygulamanın yeni versiyon ile çalıştırılmasını sağlayan süreçlerin otomasyonu, Jenkins uygulaması ile sağlanmaktadır. Yapılandırma ve devreye alma süreçlerinin adımları Jenkins konfigurasyonunda belirtilmektedir. Jenkins uygulaması ile Github kod tabanımızın entegrasyonu yapılmıştır. Github'da kod üzerinde değişiklik yapıldıktan ve onaylandıktan sonra Openshift konteyner platformunda Jenkins uygulaması tetiklenmekte ve Jenkins uygulamasında tanımlanan boru hattı (pipeline) süreci ile yapılandırma konfigurasyonları yeniden yapılandırılmakta, yayınlanmakta ve uygulamanın yeni versiyonu devreye alınarak çalıştırılmaktadır.

Openshift konteyner platformu üzerinde her uygulama kendi kod tabanını ve konfigurasyonunu kullanmaktadır. Her uygulamanın kendi süreci birbirine paralel olarak ilerleyebilmektedir.

Openshift konteyner platformunda pod'lar kalıcı veri içermez. Podların sayısı

artırılabilir veya azaltılabilir. Uygulamanın kaç adet pod üzerinde çalışacağı yapılandırma konfigurasyonunda belirtilmektedir. Jenkins konfigurasyonunda tanımlanarak pod sayısı otomatik olarak artırılmıştır. Openshift platformunda tüm konfigurasyonların ve uygulamaların günlükleri izlenebilmektedir.

Openshift konteyner platformunun sağladığı arayüz kullanılarak geliştirilen uygulamalar yapılandırılabilir, dağıtılabilir ve çalıştırılabilir. Otomasyon araçlarını platforma entegre ederek sürecin otomasyonu tanımlanabilir. Openshift platformu, Kubernetes ve Docker Swarm teknolojilerine ek olarak operasyonel kolaylık sağlayan ve konfigurasyoların otomatik olarak yapılandırılmasına imkan veren arayüz sunmaktadır. Openshift konteyner platformunu kullanarak geliştirdiğimiz kodun test etme, dağıtma ve çalıştırma adımlarını düşük zaman ve öğrenme maliyeti ile gerçekleştirebiliyoruz. Şekil 3.11.'de Openshift konteyner platformunda bir uygulamayı yapılandırmak ve dağıtmak için kullanılan örnek bir mimari gösterilmiştir. Ayrıca, mimaride yer alan katmanların konfigurasyonlarına EK-D bölümünde yer verilmiştir.



Şekil 3.11. Github Jenkins Openshift örnek uygulama mimarisi [41]

Konteyner otomasyonu ve orkestrasyonu ile uygulama yaşam döngüsünün yönetiminin sağladığı operasyonel faydalar aşağıdaki gibidir;

- Uygulama kodları ve kütüphaneleri tek paket halinde yapılandırılabilmektedir.

- İşletim sistemi ve uygulama kurulumu gerektirmeden kısa süre içerisinde uygulama çalıştırılabilmekte ve kullanıcılara hizmet verebilmektedir.
- Uygulamanın çalıştığı konteynerlerin sayısı artırılabilmekte veya azaltılabilmektedir. Bu sayede versiyon değişikliklerinden kullanıcılar etkilenmemektedir.
- Sanal sunucu veya işletim sistemi kurulumu gerektirmediği için uygulamanın altyapı gereksinimleri kısa süre içerisinde karşılanabilmektedir.
- Geliştiricilerin arayüz ile uygulamalarının yaşam döngüsündeki tüm süreçleri yönetebilmesi sağlanmaktadır.
- Düşük kaynak gereksinimi, konfigurasyon ve operasyonel iş yükü ile uygulamalar kod geliştirme sürecinden kullanıcılara hizmet verme adımına kadar yönetilebilmektedir.

4. SONUÇLAR VE ÖNERİLER

Bu bölümde, çalışma kapsamında yapılan performans karşılaştırmaları ve incelemeler sonucunda elde edilen veriler değerlendirilmiştir. Sonuç değerlendirmelerine ek olarak ileride yapılacak çalışmalar için öneride bulunulmuştur.

Günümüzde kurum ve kurumsal firmalar tarafından veri merkezlerinde iş yükü ve toplam sahip olma maliyetlerinde tasarruf hedeflenmektedir. Sağladığı avantajlar sebebiyle bulut bilişim ve sanallaştırma teknolojilerinin popülerliği artmaktadır. Çalışmada bulut bilişim teknolojilerinin bileşenleri, mimarileri ve katmanlarından bahsedilmiştir ve kullanıcıların ihtiyacına göre doğru bulut bilişim çözümlerine karar vermesi için bulut bilişim modelleri açıklanmıştır. Firmaların ihtiyaçlarına ve yasal gereksinimlere göre bulut bilişim modelleri arasında tercih yapılabilmektedir. Ancak, son kullanıcılar için genel bulut hizmetlerini kullanımının daha avantajlı olduğu düşünülmektedir.

Bu çalışmada, bulut bilişimin temelini oluşturan sanallaştırma teknolojileri incelenmiştir. Konteyner tabanlı sanallaştırma ve hipervizör tabanlı sanallaştırma ortamları iki adımda incelenmiştir. İlk olarak, monitor ve yük testi araçları ile işlemci, bellek ve ağ performanslarının karşılaştırılması yapılmıştır. İkinci adımda, Openshift konteyner platformunda örnek bir web uygulaması geliştirilmiştir ve bu uygulamanın kod geliştirmesinden kullanıcılara hizmet verme sürecine kadar olan adımları otomatize olarak gerçekleştirilmiştir. Bu platform kullanılarak on iki faktör uygulama geliştirme süreçlerini referans alan örnek bir web uygulamasının kod geliştirmesinden kullanıcılara hizmet verme sürecine kadar olan adımları gerçekleştirilmiş ve otomasyonu sağlanmıştır. Konteyner tabanlı sanallaştırma ortamında uygulama yaşam döngüsü adımları analiz edilerek avantajları tespit edilmiştir.

İşlemci performansları için yapılan analizler sonucunda konteyner çözümlerinin işlemci performansı bakımından hipervizör tabanlı sanal sunucuya göre %20-%30 daha başarılı performans sağladığı tespit edilmiştir.

Bellek performansı ve bellek bant genişliği değerleri karşılaştırıldığında, kontenyer çözümleri ve sanal sunucu arasında belirgin olarak performans farkı gözlemlenmemiştir. Ancak, konteyner çözümlerinin %1-%10 aralığında bellek performans sonuçlarının daha başarılı olduğu gözlemlenmiştir.

Stres aracı ile yapılan yük testi sonucunda, konteyner çözümleri ve sanal sunucu ortamlarında problem yaşanmadan talep edilen yüke cevap verildiği gözlemlenmiştir. Ancak, sanal sunucu ortamlarında ana sunucunun işletim sistemine ek olarak sanal sunucu için ayrı bir işletim sistemine ihtiyaç duyması sebebiyle, sanal sunucu kendisine atanan kaynakların bir kısmını işletim sistemine sağladığı için gelen kaynak talebinin bir kısmını karşılamamıştır. Ağ performansı ölçümünde TCP ve UDP paketleri ile veri aktararak ağ bant genişliği ölçümü yapılmıştır. Yapılan testlerde konteyner çözümlerinin ağ performansının sanal sunucuya göre daha başarılı olduğu görülmüştür.

Yapılan performans testleri sonucunda konteyner çözümleri olan Docker ve linux konteynerlerin işlemci, bellek ve ağ performansı testlerinde daha başarılı sonuç verdiği tespit edilmiştir. Başarılı performans değerlerine ek olarak konteynerler, daha düşük kaynak gereksinimine ihtiyaç duyması ve aynı fiziksel sunucu üzerine sanal sunucuya oran ile daha fazla konteyner çalışabilmesi sebebiyle maliyet kriteri açısından da avantajlı olduğu düşünülmektedir. Ayrıca, konteyner ortamları sanal sunucu ortamlarına göre daha kısa açılış süresine sahiptir. Sanal sunucular ile karşılaştırıldığında konteyner çözümleri kullanımının maliyet ve performans bakımından daha avantajlı olduğu görülmüştür.

Sanal sunuculara göre bir fiziksel ortamda daha fazla sayıda konteyner çalıştırılabiliyor olması kaynağın verimli kullanılması avantajını sağlarken, ortamda yönetilecek daha fazla bileşen oluşmasına da sebep olmaktadır. Bir fiziksel sunucu üzerinde yüzlerce konteyner çalıştırılabildiği göz önüne alınınca, bu ortamların yönetiminde orkestrasyon ve otomasyon kullanımı zorunlu hale gelmektedir. Kubernetes, Docker Swarm'a göre daha karmaşık bir mimariye sahiptir ve Kubernetes mimarisinde erişilebilirlik daha ön planda tutulmuştur. Bu sebeple, büyük yapılarda Kubernetes daha avantajlı olduğu görünmektedir.

Günümüzde bir ürünü ortaya çıkarırken temel amaç; uygulamanın kodunun

geliştirildiği süreçten kullanıcının hizmetine sunulduğu ana kadar yer alan süreçlerde uygulama yaşam döngüsünü standart, otomatize ve hızlı olarak gerçekleştirmek ve uygulamanın altyapı maliyetlerini minimumda tutmaktır. Konteyner tabanlı sanallaştırma teknolojileri altyapı maliyetlerini ve yönetimini hipervizör tabanlı sanallaştırma ve fiziksel sunucu ortamlarına göre düşürmektedir. Openshift konteyner platformu, konteyner tabanlı sanallaştırma teknolojilerinin sağladığı bu avantajların yanında uygulama yaşam döngüsü süreçlerinin yönetiminde avantaj sağlamaktadır. Platform incelenerek ve örnek süreçler platform üzerinde gerçekleştirilerek sağlanan bu avantajlar tespit edilmiştir. Openshift konteyner platformunun yetkinliklerini incelemek için uygulama geliştirme ve yönetim süreçlerini standartlaştırma ve iyileştirme için geliştirilen on iki faktör uygulama geliştirme süreçleri referans alınarak Openshift platformunda temel bir PHP koduna sahip web uygulaması kullanıcıların hizmet verebilir yapıya getirilmiştir. Openshift konteyner platformu Docker ve Kubernetes teknolojilerini altyapısında kullanarak bu ortamların tüm avantajlarından yararlanmaktadır. Buna ek olarak, mikroservis mimarisini temel alarak uygulama geliştirme, sürekli entegrasyon ve sürekli teslim süreçlerini destekleme avantajları sayesinde uygulama yaşam döngüsünün uçtan uca otomasyonunu desteklemektedir. Openshift konteyner platformunda uygulamaların Docker Swarm ve Kubernetes ortamlarına göre daha geliştirilebilir ve yönetilebilir olduğu gözlemlenmiştir. Openshift konteyner platformunun geliştiricilere sağladığı arayüz ortamı sayesinde düşük öğrenme maliyeti ve efor ile uygulamalar çalıştırılabilmektedir.

Bu çalışmada, konteyner tabanlı sanallaştırma çözümlerinin performans olarak daha başarılı olduğunu kanıtlanmış ve konteyner sanallaştırma çözümleri ile mevcut bir fiziksel kaynakta daha fazla uygulamaya hizmet vererek kaynakların daha verimli kullanıldığı ortaya koyulmuştur. Konteyner tabanlı sanallaştırma çözümlerinde bir fiziksel sunucuda daha fazla bileşenin yer alması ve karmaşıklığı dezavantajını ortadan kaldırmak için konteyner otomasyon ve orkestrasyon çözümlerinin gerekliliği belirtilmiştir. Bu karşılaştırma ve analizlere ek olarak, çalışmadaki temel katkı konteyner tabanlı sanallaştırma teknolojilerini içeren Openshift konteyner platformu kullanılarak uygulama yaşam döngüsünde geliştirme ve uygulamanın hizmet verme süreçlerinin otomatize edilebildiği ve uygulamaların kullanıcılara ulaşma sürecinin hızlandığını ortaya konulmasıdır. Bulut bilişim teknolojilerinin ve konteyner tabanlı

sanallaştırma teknolojilerinin erişilebilirlik ve büyük yapıların yönetim maliyetlerine olan katkısını belirgin olarak değerlendirmek için birden fazla fiziksel ortamın küme yapısında kullanılması gelecekte bu çalışmayı referans alarak yapılabilecek çalışmalar için önerilmektedir. Bu sayede, fiziksel kaynaklardan bir havuz oluşturularak kaynakların paylaşımı gözlemlenebilecek ve yedeklilik testleri yapılarak erişilebilirliğe olan katkısı incelenebilecektir.

KAYNAKLAR

[1] Bulut Bilişim, Bilgi Teknolojileri ve İletişim Kurumu, https://www.btk.gov.tr/uploads/pages/slug/bulut-bilisim.pdf (Ziyaret tarihi: 25 Mayıs 2019).

[2] Bulut Bilişim Nedir, Amazon Web Services Inc., https://aws.amazon.com/tr/what-is-cloud-computing/ (Ziyaret tarihi: 25 Mayıs 2019).

[3] Ranjan R., The Cloud Interoperability Challenge, *IEEE Cloud Computing*, DOI: 10.1109/MCC.2014.41.

[4] Bulut Bilişim Nedir, Microsoft, https://azure.microsoft.com/tr-tr/overview/what-is-cloud-computing/ (Ziyaret tarihi: 25 Mayıs 2019).

[5] Mirzaoğlu A., Bulut Bilişimin Teknik, Uygulama ve Düzenleme Boyutuyla Değerlendirilmesi, Dünya Örnekleri ve Ülkemize İlişkin Öneriler, Bilişim Uzmanlığı Tezi, Bilgi Teknolojileri ve İletişim Kurumu, Ankara, 2011, 0089.

[6] Kaya F., Bulut Bilişim Uygulamaları ve İller Bankası'na Uygulanabilirliği, Uzmanlık Tezi, İller Bankası, Ankara, 2017, 14678.

[7] Felter W., An Updated Performance Comparison of Virtual Machines and Linux Containers, http://course.ece.cmu.edu/~ece845/docs/containers.pdf (Ziyaret tarihi: 15 Mayıs 2019).

[8] Yılmaz E., Sanal Makineler ve Linux Konteynerlerin Performans Karşılaştırması, *XVIII.* , Aydın, Türkiye, 30 Ocak – 5 Şubat 2016.

[9] Piraghaj S., Dastjerdi A. V., Calheiros R., Buyya R., Efficient Virtual Machine Sizing For Hosting Containers as a Service, *2015 IEEE World Congress on Services*, DOI: 10.1109/SERVICES.2015.14.

[10] DESAI, P. (2016), A Survey of Performance Comparison between Virtual Machines and Containers, *International Journal of Computer Sciences and Engineering*, 4(7), 55-59.

[11] Ismael C., Anne O., Jean M., Comparative Experimental Analysis of the Quality-of-Service and Energy-Efficiency of VMs and Containers' Consolidation for Cloud Applications, *2017 25th International Conference on Software, Telecommunications and Computer Networks*, DOI: 10.23919/SOFTCOM.2017.8115516.

[12] Zhang Q., Ling L., Calton P., Qiwei D., Liren W., Wei Z., A Comparative Study of Containers and Virtual Machines in Big Data Environment, *2018 IEEE 11th International Conference on Cloud Computing*, DOI: 10.1109/CLOUD.2018.00030.

[13] Furth B., Escalante A., *Handsbook of Cloud Computing*, 2010th ed., Springer Science Business Media, London, 2010.

[14] Doğru A., Sunucu Sanallaştırma ve Uygulama Sanallaştırma Teknolojileri Performans Karşılaştırması, Yüksek Lisans Tezi, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul, 538721.

[15] Şener C., Ulu C., Ergin O., Sanallaştırma, , TBD/Kamu-BIB/2010-ÇG, 3-69, 2010.

[16] Osnat R., A Brief History of Containers, Aqua Security Software Inc., https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016, (Ziyaret tarihi: 25 Mayıs 2019).

[17] Silberschats A., Galvin P., Gagne G., *Operating System Concept*, 9th ed., John Wiley&Sons Inc, United States of America, 2013.

[18] Yadav A., Garg M., Mehra R., Docker Containers Versus Virtual Machine-Based Virtualization, *Emerging Technologies in Data Mining and Information Security*, DOI: 10.1007/978-981-13-1501-5_12.

[19] Kernel Virtual Machine, Red Hat Inc., https://www.linux-kvm.org/page/Main_Page, (Ziyaret tarihi: 20 Nisan 2019).

[20] What is LXC, Canonical Ltd., https://linuxcontainers.org/lxc/introduction/ (Ziyaret tarihi: 13 Nisan 2019).

[21] Docker Documentation, Docker Inc., https://docs.docker.com/ (Ziyaret tarihi: 25 Mayıs 2019).

[22] Docker Nedir, Amazon Web Servics Inc., https://aws.amazon.com/tr/docker/ (Ziyaret tarihi: 25 Mayıs 2019).

[23] Introduction to Linux Container, Red Hat Inc., https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers#linux_containers_architecture (Ziyaret tarihi: 25 Mayıs 2019).

[24] Morgan T., IBM Techies Pit Docker, KVM Against Linux Bare Metal, Enterpriese AI, https://www.enterpriseai.news/2014/08/18/ibm-techies-pit-docker-kvm-bare-metal/ (Ziyaret tarihi: 1 Haziran 2019).

[25] Swarm Mode Overview, Docker Inc., https://docs.docker.com/engine/swarm/ (Ziyaret tarihi: 25 Mayıs 2019).

[26] Eldridge I., What is Container Orchestration, New Relic, https://blog.newrelic.com/engineering/container-orchestration-explained/ (Ziyaret tarihi: 25 Mayıs 2019).

[27] Kubernetes Documentation, The Linux Foundation, https://kubernetes.io/docs/concepts/ (Ziyaret tarihi: 25 Mayıs 2019).

[28] Openshift Documentation, Red Hat Inc., https://docs.openshift.com/ (Ziyaret tarihi: 2 Haziran 2019).

[29] Michiels H., Openshift an Introduction, Ordina Belgium, https://ordina-jworks.github.io/paas/2017/06/29/Openshift-an-introduction.html (Ziyaret tarihi: 2 Haziran 2019).

[30] DevOps Nedir, Amazon Web Services Inc., https://aws.amazon.com/tr/devops/what-is-devops/ (Ziyaret tarihi: 2 Haziran 2019).

[31] Wiggins A., The Twelve Factor App, https://12factor.net/ (Ziyaret tarihi: 2 Haziran 2019).

[32] Çelik T., Uygulamada On İki Faktör Manifestosu, Bilişim İO, http://bilisim.io/2018/12/31/uygulamada-12-faktor-manifestosu/ (Ziyaret tarihi: 2 Haziran 2019).

[33] Yılmaz E. C., Oktaş R., Konteyner Tabanlı Sanallaştırma ve 12 Faktör Yöntembilimine Dayalı Web Uygulama Geliştirme Süreci Önerisi, XVIII. Akademik Bilişim Konferansı, Aydın, Türkiye, 30 Ocak – 5 Şubat 2016.

[34] Timme F., How to Benchmark Your System, HowtoForge, https://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench (Ziyaret tarihi: 13 Nisan 2019).

[35] Gennady F., Shaojuan Z., Intel MKL Benchmarks, Intel, https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite (Ziyaret tarihi: 14 Nisan 2019).

[36] Gite V., Stress Test CPU and Memory, nixCraft, https://www.cyberciti.biz/faq/stress-test-linux-unix-server-with-stress-ng/ (Ziyaret tarihi: 14 Nisan 2019).

[37] Nuttcp Welcome Page, Nuttcp Development Team, https://www.nuttcp.net/Welcome%20Page.html (Ziyaret tarihi: 13 Nisan 2019).

[38] Netperf, HewlettPackard, https://github.com/HewlettPackard/netperf (Ziyaret tarihi: 13 Nisan 2019).

[39] Create Your Own Pipelines with Openshift 3.3, Red Hat Inc., https://blog.openshift.com/create-build-pipelines-openshift-3-3/ (Ziyaret tarihi: 1 Haziran 2019).

[40] Muchandi V., Bgdemo, https://github.com/VeerMuchandi/bgdemo (Ziyaret tarihi: 1 Haziran 2019).

[41] Multiple Deployment Methods for Openshift, Red Hat Inc., https://blog.openshift.com/multiple-deployment-methods-openshift/ (Ziyaret tarihi: 1 Haziran 2019).

EKLER

Ek–A

Docker konteyner ortamı işlemci performans testleri sysbench ve linpack araçları komut çıktıları;

[root@3f5c034fff21 /]# sysbench --test=cpu --cpu-max-prime=20000 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 1

Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:

events per second: 505,36

General statistics:

total time: 10,0015s

total number of events: 5055

Latency (ms):

min: 1,88

avg: 1,98

max: 7,55

95th percentile: 2,07

sum: 9997,37

Threads fairness:

events (avg/stddev): 5055,0000/0.00

execution time (avg/stddev): 9,9974/0.00

[root@3f5c034fff21 /]# sysbench --test=cpu --cpu-max-prime=20000 --num-threads=8 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

WARNING: --num-threads is deprecated, use --threads instead

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 8

Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:

events per second: 3423,79

General statistics:

total time: 10,0012s

total number of events: 34250

Latency (ms):

min: 2,01

avg: 2,34

max: 35,37

95th percentile: 2,43

sum: 79978,34

Threads fairness:

events (avg/stddev): 4281,2500/41,12

execution time (avg/stddev): 9,9973/0

[root@3f5c034fff21 linpack]# ./xlinpack_xeon64

Input data or print help ? Type [data]/help :

Number of equations to solve (problem size): 18000

Leading dimension of array: 18000

Number of trials to run: 2

Data alignment value (in Kbytes): 4

Current date/time: Fri Apr 26 07:02:27 2019

CPU frequency: 3,839 GHz

Number of CPUs: 1

Number of cores: 4

Number of threads: 4

Parameters are set to:

Number of tests: 1

Number of equations to solve (problem size) : 18000

Leading dimension of array : 18000

Number of trials to run : 2

Data alignment value (in Kbytes) : 4

Maximum memory requested that can be used=2592364096, at the size=18000

========== Timing linear equation system solver ==========

Size LDA Align. Time(s) GFlops Residual Residual(norm) Check

18000 18000 4 35,869 108,4117 3,012383e-10 3,298929e-02 pass

18000 18000 4 38,058 102,1771 3,012383e-10 3,298929e-02 pass

Performance Summary (GFlops)

Size LDA Align. Average Maximal

18000 18000 4 105,2944 108,4117

Residual checks PASSED

End of tests

Linux konteyner ortamı işlemci performans testleri sysbench ve linpack araçları komut çıktıları;

[root@centos_lxc ~]# sysbench --test=cpu --cpu-max-prime=20000 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 1

Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:

events per second: 509,44

General statistics:

total time: 10,0018s

total number of events: 5096

Latency (ms):

min: 1,93

avg: 1,96

max: 5,11

95th percentile: 2,00

sum: 9996,72

Threads fairness:

events (avg/stddev): 5096,0000/0

execution time (avg/stddev): 9,9967/0

[root@centos_lxc ~]# sysbench --test=cpu --cpu-max-prime=20000 --num-threads=8 run un

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

WARNING: --num-threads is deprecated, use --threads instead

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 8

Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:

events per second: 3255,64

General statistics:

total time: 10,0012s

total number of events: 32565

Latency (ms):

min: 2,17

avg: 2,46

max: 33,16

95th percentile: 2,97

sum: 79964,82

Threads fairness:

events (avg/stddev): 4070,6250/37,08

execution time (avg/stddev): 9,9956/0

[root@centos_lxc linpack]# ./xlinpack_xeon64

Input data or print help ? Type [data]/help :

Number of equations to solve (problem size): 18000

Leading dimension of array: 18000

Number of trials to run: 2

Data alignment value (in Kbytes): 4

Current date/time: Fri Apr 26 07:07:18 2019

CPU frequency: 3,808 GHz

Number of CPUs: 1

Number of cores: 4

Number of threads: 4

Parameters are set to:

Number of tests: 1

Number of equations to solve (problem size) : 18000

Leading dimension of array : 18000

Number of trials to run : 2

Data alignment value (in Kbytes) : 4

Maximum memory requested that can be used=2592364096, at the size=18000

========== Timing linear equation system solver ==========

Size LDA Align. Time(s) GFlops Residual Residual(norm) Check

18000 18000 4 47,308 82,1987 3,012383e-10 3,298929e-02 pass

18000 18000 4 37,654 103,2719 3,012383e-10 3,298929e-02 pass

Performance Summary (GFlops)

Size LDA Align. Average Maximal

18000 18000 4 92,7353 103,2719

Residual checks PASSED

End of tests

KVM sanal sunucu ortamı işlemci performans testleri sysbench ve linpack araçları komut çıktıları;

[root@centos7-kvm ~]# sysbench --test=cpu --cpu-max-prime=20000 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 1

Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:

events per second: 494,26

General statistics:

total time: 10,0056s

total number of events: 4946

Latency (ms):

min: 1,90

avg: 2,02

max: 6,39

95th percentile: 2,18

sum: 9996,87

Threads fairness:

events (avg/stddev): 4946,0000/0

execution time (avg/stddev): 9,9969/0

[root@centos7-kvm ~]# sysbench --test=cpu --cpu-max-prime=20000 --num-threads=8 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

```
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.17 (using system LuaJIT 2.0.4)
Running the test with following options:
Number of threads: 8
Initializing random number generator from current time
Prime numbers limit: 20000
Initializing worker threads...
Threads started!
CPU speed:
    events per second:  3275,29
General statistics:
    total time:                          10,0039s
    total number of events:              32770
Latency (ms):
         min:                                    2,06
         avg:                                    2,44
         max:                                   26,33
         95th percentile:                        2,61
         sum:                                79954,31
Threads fairness:
    events (avg/stddev):           4096,2500/32,98
    execution time (avg/stddev):   9,9943/0
sysbench --test=cpu --cpu-max-prime=20000 run
```

Ek–B

Docker konteyner ortamı bellek performans testleri sysbench ve linpack araçları komut çıktıları;

[root@3f5c034fff21 /]# sysbench --test=memory --memory-block-size=1M --memory-total-size=10G run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 1

Initializing random number generator from current time

Running memory speed test with the following options:

block size: 1024KiB

total size: 10240MiB

operation: write

scope: global

Initializing worker threads...

Threads started!

Total operations: 10240 (19038,34 per second)

10240,00 MiB transferred (19038,34 MiB/sec)

General statistics:

total time: 0,5366s

total number of events: 10240

Latency (ms):

min: 0,05

avg: 0,05

max: 2,35

95th percentile: 0,07

```
    sum:                        533,09
Threads fairness:
   events (avg/stddev):         10240,0000/0
   execution time (avg/stddev):  0,5331/0
[root@3f5c034fff21 /]# ./stream-me
libgomp: Invalid value for enviroment variable GOMP_CPU_AFFINITY
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76,3 MiB (= 0,1 GiB).
Total memory required = 22,9 MiB (= 0,2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 4
Number of Threads counted = 4
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 7169 microseconds.
   (= 7169 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
```

WARNING -- The above is only a rough guideline.

For best results, please be sure you know the

precision of your system timer.

Function Best Rate MB/s Avg time Min time Max time

Copy: 23801,7 0,006998 0,006722 0,007336

Scale: 15140,1 0,010016 0,009913 0,010158

Add: 16065,6 0,013687 0,013285 0,014404

Triad: 16813,4 0,013925 0,013473 0,014533

Solution Validates: avg error less than 1,000000e-13 on all three arrays

---

Linux konteyner ortamı bellek performans testleri sysbench ve linpack araçları komut çıktıları;

[root@3f5c034fff21 /]# sysbench --test=memory --memory-block-size=1M --memory-total-size=10G run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:

Number of threads: 1

Initializing random number generator from current time

Running memory speed test with the following options:

block size: 1024KiB

total size: 102400MiB

operation: write

scope: global

Initializing worker threads...

```
Threads started!
Total operations: 102400 (18807,63 per second)
102400 MiB transferred (18807,63 MiB/sec)
General statistics:
    total time:                          5,4433s
    total number of events:              102400
Latency (ms):
         min:                                    0,05
         avg:                                    0,05
         max:                                    3,24
         95th percentile:                        0,07
         sum:                                 5406,36
Threads fairness:
    events (avg/stddev):           102400,0000/0
    execution time (avg/stddev):   5,4064/0,00
[root@centos_lxc ~]# ./stream-me
-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76,3 MiB (= 0,1 GiB).
Total memory required = 228,9 MiB (= 0,2 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
```

Number of Threads requested = 8

Number of Threads counted = 8

Your clock granularity/precision appears to be 1 microseconds.

Each test below will take on the order of 9953 microseconds.

(= 9953 clock ticks)

Increase the size of the arrays if this shows that

you are not getting at least 20 clock ticks per test.

WARNING -- The above is only a rough guideline.

For best results, please be sure you know the

precision of your system timer.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	19478,9	0,009828	0,008214	0,012994
Scale:	14405,1	0,012978	0,011107	0,018441
Add:	15632,2	0,017216	0,015353	0,024361
Triad:	15599,5	0,016542	0,015385	0,021014

Solution Validates: avg error less than 1,000000e-13 on all three arrays

---

KVM sanal sunucu ortamı bellek performans testleri sysbench ve linpack araçları komut çıktıları;

[root@centos7-kvm ~]# sysbench --test=memory --memory-block-size=1M --memory-total-size=100G --num-threads=1 run

WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.

```
sysbench 1.0.17 (using system LuaJIT 2.0.4)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time

Running memory speed test with the following options:
  block size: 1024KiB
  total size: 102400MiB
  operation: write
  scope: global
Initializing worker threads...

Threads started!

Total operations: 102400 (18390,66 per second)

102400 MiB transferred (18390,66 MiB/sec)

General statistics:
    total time:                          5,5661s
    total number of events:              102400

Latency (ms):
         min:                                    0,05
         avg:                                    0,05
         max:                                    3,34
         95th percentile:                        0,08
         sum:                                 5531,72

Threads fairness:
    events (avg/stddev):           102400,0000/0
    execution time (avg/stddev):   5,5317/0

[root@centos7-kvm ~]# ./stream-me
libgomp: Invalid value for enviroment variable GOMP_CPU_AFFINITY
```

STREAM version $Revision: 5.10 $

This system uses 8 bytes per array element.

Array size = 10000000 (elements), Offset = 0 (elements)

Memory per array = 76,3 MiB (= 0,1 GiB).

Total memory required = 228,9 MiB (= 0,2 GiB).

Each kernel will be executed 10 times.

The *best* time for each kernel (excluding the first iteration)

will be used to compute the reported bandwidth.

Number of Threads requested = 8

Number of Threads counted = 8

Your clock granularity/precision appears to be 1 microseconds.

Each test below will take on the order of 11550 microseconds.

(= 11550 clock ticks)

Increase the size of the arrays if this shows that

you are not getting at least 20 clock ticks per test.

WARNING -- The above is only a rough guideline.

For best results, please be sure you know the

precision of your system timer.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	20661,0	0,008174	0,007744	0,009212

Scale:	14522,9	0,011827	0,011017	0,015259
Add:	15633,1	0,016113	0,015352	0,018417
Triad:	15558,2	0,015865	0,015426	0,016783

Solution Validates: avg error less than 1,000000e-13 on all three arrays

---

Ek-C

Docker konteyner ortamı ağ performans testleri sysbench ve linpack araçları komut çıktıları.

[root@vmcentos ~]# netperf -n 16 -H 172.17.0.2 -c -C -t TCP_RR

MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 172.17.0.2 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

16384 87380 1 1 10,00 12713,41 9,39 9,39 59,063 59,63

16384 87380

[root@vmcentos ~]# netperf -n 16 -H 172.17.0.2 -c -C -t UDP_RR

MIGRATED UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 172.17.0.2 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

212992 212992 1 1 10,00 12717,79 9,22 9,22 58,014 58,014

212992 212992

[root@vmcentos benchmarks_2018]# nuttcp -i1 172.17.0.2

4064,8750 MB / 1,00 sec = 34093,7334 Mbps 0 retrans

4012,5000 MB / 1,00 sec = 33648,7575 Mbps 0 retrans

3947,2500 MB / 1,00 sec = 33112,5621 Mbps 0 retrans

3905,7500 MB / 1,00 sec = 32762,0693 Mbps 0 retrans

3876,1250 MB / 1,00 sec = 32517,9922 Mbps 0 retrans

3759,4375 MB / 1,00 sec = 31533,6725 Mbps 0 retrans

3962,2500 MB / 1,00 sec = 33238,9254 Mbps 0 retrans

3921,3750 MB / 1,00 sec = 32896,4238 Mbps 0 retrans

3910,7500 MB / 1,00 sec = 32807,2251 Mbps 0 retrans

3861,8750 MB / 1,00 sec = 32400,1295 Mbps 0 retrans

39222,4375 MB / 10,00 sec = 32895,6421 Mbps 99 %TX 97 %RX 0 retrans 0,12 msRTT

Linux konteyner ortamı ağ performans testleri sysbench ve linpack araçları komut çıktıları;

[root@vmcentos ~]# netperf -n 16 -H 192.168.122.135 -c -C -t TCP_RR

MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.122.135 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

16384 87380 1 1 10,00 13493,92 9,64 9,63 57,125 57,068

16384 87380

[root@vmcentos ~]# netperf -n 16 -H 192.168.122.135 -c -C -t UDP_RR

MIGRATED UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.122.135 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

212992 212992 1 1 10,00 14106,97 9,23 9,23 52,328 52,328

[root@vmcentos ~]# nuttcp -i1 192.168.122.135

4475,5625 MB / 1,00 sec = 37513,2786 Mbps 0 retrans

4375,4375 MB / 1,00 sec = 36719,7664 Mbps 0 retrans

3101,5625 MB / 1,00 sec = 26026,6411 Mbps 0 retrans

4138,5000 MB / 1,00 sec = 34694,1540 Mbps 0 retrans

4306,6875 MB / 1,00 sec = 36146,1984 Mbps 0 retrans

4065,4375 MB / 1,00 sec = 34085,6711 Mbps 0 retrans

4368,0000 MB / 1,00 sec = 36646,8635 Mbps 0 retrans

4103,5625 MB / 1,00 sec = 34431,1308 Mbps 0 retrans

1473,5000 MB / 1,00 sec = 12363,7048 Mbps 0 retrans

3867,8750 MB / 1,00 sec = 32432,1089 Mbps 0 retrans

38276,1875 MB / 10,00 sec = 32106,7815 Mbps 99 %TX 84 %RX 0 retrans 0,10 msRTT

KVM sanal sunucu ortamı ağ performans testleri sysbench ve linpack araçları komut çıktıları;

[root@vmcentos ~]# netperf -n 16 -H 192.168.150.131 -c -C -t TCP_RR

MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.150.131 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

16384 87380 1 1 10,00 7741,21 17,45 2,77 180,314 28,601

16384 87380

[root@vmcentos ~]# netperf -n 16 -H 192.168.150.131 -c -C -t UDP_RR

MIGRATED UDP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.150.131 () port 0 AF_INET : first burst 0

Local /Remote

Socket Size Request Resp. Elapsed Trans. CPU CPU S.dem S.dem

Send Recv Size Size Time Rate local remote local remote

bytes bytes bytes bytes secs. per sec % S % S us/Tr us/Tr

212992 212992 1 1 10,00 7954,01 17,33 2,90 174,328 29,07

212992 212992

[root@vmcentos ~]# nuttcp -i1 192.168.150.131

1148,8125 MB / 1,00 sec = 9635,7044 Mbps 0 retrans

1970,1875 MB / 1,00 sec = 16527,0315 Mbps 0 retrans

1982,6250 MB / 1,00 sec = 16628,8698 Mbps 0 retrans

1893,0000 MB / 1,00 sec = 15883,1928 Mbps 0 retrans

1950,8750 MB / 1,00 sec = 16365,8621 Mbps 0 retrans

1869,6875 MB / 1,00 sec = 15683,6050 Mbps 0 retrans

1929,6250 MB / 1,00 sec = 16186,7706 Mbps 0 retrans

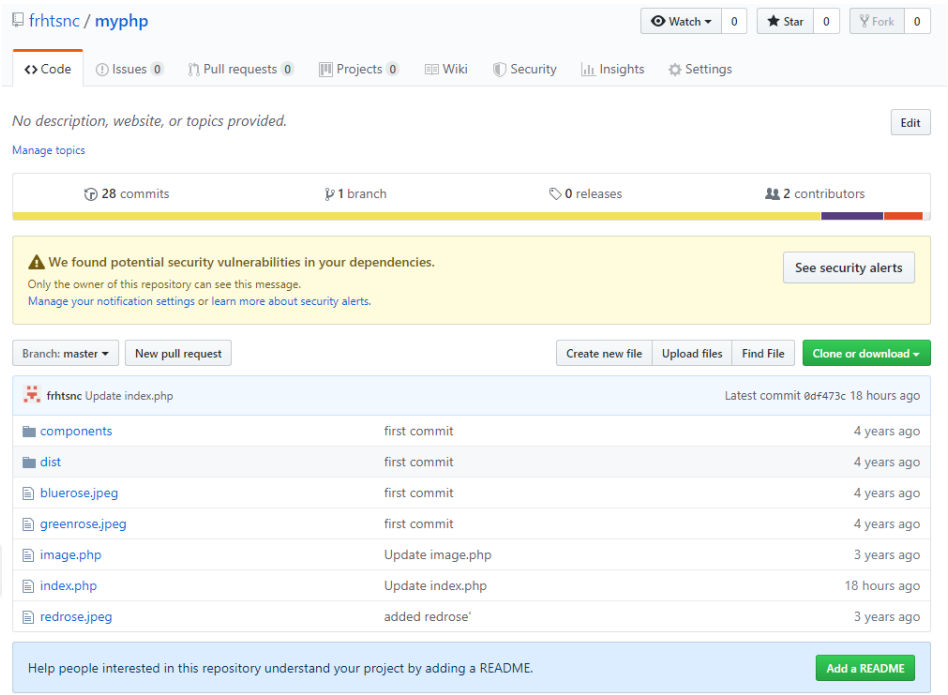
1823,8750 MB / 1,00 sec = 15298,4720 Mbps 0 retrans

1891,3750 MB / 1,00 sec = 15866,8761 Mbps 0 retrans

1932,9375 MB / 1,00 sec = 16214,1199 Mbps 0 retrans

18398,5000 MB / 10,00 sec = 15431,9348 Mbps 54 %TX 95 %RX 0 retrans 0,50 msRTT

Ek-D



Şekil D.1. PHP Uygulaması Github Kod Tabanı

Openshift boru hattı (pipeline) konfigürasyonu;

```
node() {
stage 'build'
openshiftBuild(buildConfig: 'myphp', showBuildLogs: 'true')
stage 'deploy'
openshiftDeploy(deploymentConfig: 'myphp')
openshiftScale(deploymentConfig: 'myphp',replicaCount: '2')
}
```

Openshift platformu php web uygulaması yapılandırma konfigürasyonu;

```
apiVersion: v1
kind: ReplicationController
metadata:
  annotations:
    openshift.io/deployer-pod.completed-at: '2019-06-08 18:29:03 +0000 UTC'
```

openshift.io/deployer-pod.created-at: '2019-06-08 18:28:28 +0000 UTC'

openshift.io/deployer-pod.name: myphp-6-deploy

openshift.io/deployment-config.latest-version: '6'

openshift.io/deployment-config.name: myphp

openshift.io/deployment.phase: Complete

openshift.io/deployment.replicas: "

openshift.io/deployment.status-reason: manual change

openshift.io/encoded-deployment-config: >

{"kind":"DeploymentConfig","apiVersion":"apps.openshift.io/v1","metadata":{"nam
e":"myphp","namespace":"php-
app","selfLink":"/apis/apps.openshift.io/v1/namespaces/php-
app/deploymentconfigs/myphp","uid":"5df2ce36-8a03-11e9-9e1a-
0800278e099f","resourceVersion":"438156","generation":8,"creationTimestamp":"2
019-06-
08T15:37:47Z","labels":{"app":"myphp"},"annotations":{"openshift.io/generated-
by":"OpenShiftWebConsole"}},"spec":{"strategy":{"type":"Rolling","rollingParams
":{"updatePeriodSeconds":1,"intervalSeconds":1,"timeoutSeconds":600,"maxUnavai
lable":"25%","maxSurge":"25%"},"resources":{},"activeDeadlineSeconds":21600},"
triggers":[{"type":"ImageChange","imageChangeParams":{"containerNames":["myp
hp"],"from":{"kind":"ImageStreamTag","namespace":"php-
app","name":"myphp:latest"},"lastTriggeredImage":"172.30.1.1:5000/php-
app/myphp@sha256:96843e195e61662e0d383ae1583954887738aaefa9449436b03a
9d98f36cc1e6"}}],"replicas":2,"test":false,"selector":{"deploymentconfig":"myphp"
},"template":{"metadata":{"creationTimestamp":null,"labels":{"app":"myphp","depl
oymentconfig":"myphp"}},"spec":{"containers":[{"name":"myphp","image":"172.30
.1.1:5000/php-
app/myphp@sha256:96843e195e61662e0d383ae1583954887738aaefa9449436b03a
9d98f36cc1e6","ports":[{"containerPort":8080,"protocol":"TCP"},{"containerPort":8
443,"protocol":"TCP"}],"resources":{},"terminationMessagePath":"/dev/termination
-
log","terminationMessagePolicy":"File","imagePullPolicy":"Always"}],"restartPolic
y":"Always","terminationGracePeriodSeconds":30,"dnsPolicy":"ClusterFirst","securi
tyContext":{},"schedulerName":"default-
scheduler"}}},"status":{"latestVersion":6,"observedGeneration":7,"replicas":2,"upda
tedReplicas":2,"availableReplicas":2,"unavailableReplicas":0,"details":{"message":"
manual

change","causes":[{"type":"Manual"}]},"conditions":[{"type":"Available","status":"
True","lastUpdateTime":"2019-06-08T15:40:10Z","lastTransitionTime":"2019-06-
08T15:40:10Z","message":"Deployment

config has minimum

availability."},{"type":"Progressing","status":"True","lastUpdateTime":"2019-06-08T16:50:56Z","lastTransitionTime":"2019-06-08T16:50:54Z","reason":"NewReplicationControllerAvailable","message":"replication

controller \"myphp-5\" successfully rolled out"}],"readyReplicas":2}}

```
    openshift.io/jenkins-build-uri: >-
      https://jenkins-php-app.192.168.99.101.nip.io/job/php-app/php-app-myfirstpipeline/6/
  creationTimestamp: '2019-06-08T18:28:27Z'
  generation: 3
  labels:
    app: myphp
    openshift.io/deployment-config.name: myphp
  name: myphp-6
  namespace: php-app
  ownerReferences:
    - apiVersion: apps.openshift.io/v1
      blockOwnerDeletion: true
      controller: true
      kind: DeploymentConfig
      name: myphp
      uid: 5df2ce36-8a03-11e9-9e1a-0800278e099f
  resourceVersion: '682424'
  selfLink: /api/v1/namespaces/php-app/replicationcontrollers/myphp-6
  uid: 359aca4a-8a1b-11e9-a67e-0800278e099f
spec:
  replicas: 2
  selector:
```

```
    deployment: myphp-6
    deploymentconfig: myphp
  template:
    metadata:
      annotations:
        openshift.io/deployment-config.latest-version: '6'
        openshift.io/deployment-config.name: myphp
        openshift.io/deployment.name: myphp-6
      creationTimestamp: null
      labels:
        app: myphp
        deployment: myphp-6
        deploymentconfig: myphp
    spec:
      containers:
        - image: >-
            172.30.1.1:5000/php-
app/myphp@sha256:96843e195e61662e0d383ae1583954887738aaefa9449436b03a
9d98f36cc1e6
          imagePullPolicy: Always
          name: myphp
          ports:
            - containerPort: 8080
              protocol: TCP
            - containerPort: 8443
              protocol: TCP
          resources: {}
```

```
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 2
  fullyLabeledReplicas: 2
  observedGeneration: 3
  readyReplicas: 2
  replicas: 2
```

Openshift platformu php web uygulaması servis konfigürasyonu;

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftWebConsole
  creationTimestamp: '2019-06-08T15:37:47Z'
  labels:
    app: myphp
  name: myphp
  namespace: php-app
  resourceVersion: '396423'
  selfLink: /api/v1/namespaces/php-app/services/myphp
  uid: 5df2b556-8a03-11e9-a67e-0800278e099f
spec:
  clusterIP: 172.30.243.175
  ports:
    - name: 8080-tcp
```

```
    port: 8080
    protocol: TCP
    targetPort: 8080
  - name: 8443-tcp
    port: 8443
    protocol: TCP
    targetPort: 8443
 selector:
  deploymentconfig: myphp
 sessionAffinity: None
 type: ClusterIP
status:
 loadBalancer: {}
```

Openshift platformu php web uygulaması port bağlama konfigürasyonu;

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
 annotations:
  openshift.io/generated-by: OpenShiftWebConsole
  openshift.io/host.generated: 'true'
 creationTimestamp: '2019-06-08T15:37:47Z'
 labels:
  app: myphp
 name: myphp
 namespace: php-app
 resourceVersion: '396430'
 selfLink: /apis/route.openshift.io/v1/namespaces/php-app/routes/myphp
spec:
 host: myphp-php-app.192.168.99.101.nip.io
 port:
```

```
    targetPort: 8080-tcp
  to:
    kind: Service
    name: myphp
    weight: 100
  wildcardPolicy: None
status:
  ingress:
    - conditions:
        - lastTransitionTime: '2019-06-08T15:37:47Z'
          status: 'True'
          type: Admitted
      host: myphp-php-app.192.168.99.101.nip.io
      routerName: router
      wildcardPolicy: None
```

KİŞİSEL YAYIN VE ESERLER

[1] **Sanuç F.**, Yiğit H., Sanal Sunucu ve Konteyner Teknolojilerinin Performans Karşılaştırması ve Yazılım Geliştirme Süreçlerine Etkisi, *Marmara Fen ve Sosyal Bilimler Kongresi,* Kocaeli, Türkiye, 26-28 Nisan 2019.

ÖZGEÇMİŞ

Ferhat Sanuç, 1990 yılında Karabük'de doğdu. İlköğretim eğitimini Karabük'de, lise eğitimini İstanbul'da tamamladı. 2009 yılında girdiği Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliği Bölümü'nden 2013 yılında mezun oldu. 2016/2017 eğitim öğretim yılından itibaren Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Bilişim Sistemleri Mühendisliği Bölümü'nde eğitimine devam etmektedir. 2013 ile 2017 yılları arasında Türk Telekom firmasında Unix Sistem Yöneticisi olarak görev yaptı. 2017 ile 2019 yılları arasında Akbankfirmasında Unix Sistem Yönetici olarak görev yaptı. 2019 yılı itibari ile Vodafone Firmasında Bulut Sistemleri Operasyon Eksperi olarak görev yapmaktadır.