T.C.

KIRIKKALE ÜNİVERSİTESİ

FEN BİLİMLER ENSTİTÜSÜ

UYGULAMA YAZILIMLARI İÇİN KONTEYNER TABANLI UYGULAMA HAVUZLARININ GELİŞTİRİLMESİ

AHMET ÖZCAN

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DOKTORA TEZİ

DANIŞMAN

Doç. Dr. Atilla ERGÜZEN

KIRIKKALE-2022

ÖZET

UYGULAMA YAZILIMLARI İÇİN KONTEYNER TABANLI UYGULAMA HAVUZLARININ GELİŞTİRİLMESİ

Kırıkkale Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı, Doktora Tezi
Danışman: Doç. Dr. Atilla ERGÜZEN
Ocak 2022, 114 sayfa

Günümüzde internet herkes tarafından ulaşılabilir ve yaygın hale gelmiştir. Hızlı internetin yaygınlaşması ile eğitim kurumları da dönüşüm sürecine girmiş ve bazı eğitim içeriklerini çevrimiçi sunmaya başlamıştır. Önceleri alternatif bir öğretim modeli olarak internet üzerinden sunulan uzaktan öğretim sistemi, Covid19 salgınının yaşandığı günümüzde bir zorunluluk haline gelmiştir. Salgın döneminde tüm eğitim kurumları sınıf içi eğitimlerini uzaktan eğitim modeline dönüştürmüştür. Teknolojik eğitimlerde olduğu gibi mühendislik eğitimi müfredatında teorik ve uygulamalı eğitimler bulunmaktadır. Dersliklerde verilen teorik eğitimler laboratuvarlarda yapılan uygulamalarla pekiştirilmekte ve öğrencilerin deneyim kazanması sağlanmaktadır. Pandemi döneminde uzaktan eğitime katılmak zorunda kalan öğrenciler uygulamalı eğitimlerden uzak kalmaktadır.

Uzaktan öğretim modelinde öğrenciler teorik dersleri sanal sınıflarda, uygulamalı laboratuvar eğitimlerini de sanal laboratuvarlarda yapılabilmektedir. Yazılım mühendisliği için oluşturulan sanallaştırma tabanlı laboratuvarlar uzun yıllardır kullanılmaktadır. Sanallaştırma teknolojisi, fiziksel bir sunucuya ait kaynakların mantıksal olarak paylaştırılması esasına dayanır. Bu teknolojide her sanal makine için bir sunucu örneği kopyalama zorunluluğu bulunmaktadır. Sunucu örnekleri, gerçek bir makine gibi eksiksiz bir işletim sistemi ve kütüphanelerle birlikte mantıksal olarak kendilerine ayrılan donanım kaynaklarını tüketmektedir. Sanal makine çalışmakta olduğu sürece kendisine ayrılan kaynaklar başka bir sanal makine tarafından kullanılamaz. Bu durum sanallaştırma yapısını kullanan tüm kurumlar için önemli bir

problemdir. Başta bulut hizmet sağlayıcıları olmak üzere tüm teknoloji şirketleri bu soruna çözüm için yenilikçi ve ekonomik yöntemler aramaktadır.

Son yıllarda geliştirilen konteyner teknolojisi bu sorunun çözümü olarak görülmektedir. Konteyner teknolojisi, aynı işletim sistemi çekirdeğini kullanan birbirinden yalıtılmış uygulamalar olarak ifade edilebilir. Bir yazılım işletim sistemi tarafından çalıştırıldığında sistem kaynaklarının bir bölümünü kullanmaktadır, bu yazılım işletim sisteminin kullandığı bölümü ile birlikte bir paket haline getirildiğinde konteyner olarak tanımlanmaktadır. Bu sayede her yazılım ihtiyaç duyduğu kadar bir büyüklük ile paketlenecektir. Bu pakete konteyner adı verilmektedir. Bu teknolojide her uygulamanın kendine özel bağımlılıkları, kütüphaneleri ve dosyaları mevcuttur. Aynı fiziksel makinede birbirinden izole birçok uygulamayı, hatta aynı uygulamanın farklı sürümlerini çok düşük sistem kaynaklarıyla çalıştırabilmek mümkündür. Özetle; Bir yazılımın çalıştırılabilşmesi için, içerisinde yüklü bir işletim sistemi olan bir bilgisayara ihtiyaç vardır. Bu işlem aslında oldukça maliyetlidir, konteyner teknolojisi sayesinde özellikle işletimin sisteminin kendisine yetecek kadar olan özellikleri ve sınırlı sistem kaynakları kullanılarak maliyet çok düşürülmektedir.

Bu çalışmada, uzaktan öğretim sistemi için konteyner tabanlı sanallaştırma alternatifi bir platform oluşturulmuştur. Yazılım geliştirme eğitimlerinde kullanılmak üzere konteyner haline getirilmiş yazılım geliştirme ortamlarının bu platform üzerinden uzaktan eğitim öğrencilerinin kullanımına sunulması amaçlanmıştır. .Net Core 5.0 altyapısıyla çoklu platform uyumluluğu sağlanarak, MGD katmanlı modeliyle geliştirilen platform Docker konteynerlerini kullanmaktadır. Çalışmada test amaçlı yaygın kullanılan yazılım dillerini destekleyen ve eklentileriyle yetenekleri artırılabilen Visual Studio Code yazılım geliştirme editörü konteyneri kullanılmıştır. Dünyada büyük bir pandemi yaşanmaktadır. Pandemi döneminde öğrenciler okullardan uzak kalarak uzaktan eğitim faaliyetlerine katılmaktadır. Çalışma Uzaktan eğitim gören öğrenciler ve yazılım geliştirme laboratuvarına ihtiyaç duyan öğrenciler için önemlidir. Oluşturulacak benzeri laboratuvarlar pandemi döneminde öğrencilere büyük kolaylık sağlayacaktır.

Anahtar kelimeler: Sanallaştırma, Konteyner Teknolojisi, Robot İşletim Sistemi, Mikro Servisler, Docker.

ABSTRACT

DEVELOPMENT OF CONTAINER-BASED APPLICATION POOLS FOR APPLICATION PROGRAMS

Kırıkkale University
Graduate School of Natural and Applied Sciences
Department of Computer Engineeering, Ph. D. Thesis
Supervizor: Assoc. Prof. Dr. Atilla ERGUZEN
January 2022, 114 pages

Today, the Internet is accessible and widely used by everyone. With the spread of high-speed Internet, educational institutions have also made the change and started to offer some educational content online. Distance education, which used to be offered over the Internet as an alternative model of instruction, has now become a necessity in light of the Covid19 epidemic. During the epidemic, all educational institutions converted their classroom training to a distance learning model. As in technical education, there is theoretical and applied training in engineering education. Theoretical training in classrooms is reinforced by practice in laboratories, and students gain experience. Students who are required to attend distance learning during the pandemic period stay away from applied training.

In the distance learning model, students can take theoretical classes in virtual classrooms and hands-on lab exercises in virtual labs. Virtualization-based labs created for software engineering have been used for many years. Virtualization technology is based on logical allocation of resources belonging to a physical server. In this technology, there is an obligation to copy one server instance for each virtual machine. Like a real computer, the server instances consume the hardware resources logically allocated to them, as well as a complete operating system and libraries. As long as the virtual machine is running, the resources allocated to it cannot be used by another virtual machine. This is an important problem for all institutions that use a virtualization structure. All technology companies, especially cloud service providers, are looking for innovative and economical methods to solve this problem.

Container technology, developed in recent years, is seen as a solution to this problem. Container technology can be expressed as applications that are isolated from each other and use the same operating system kernel. When a software is executed by the operating system, it uses some of the system resources. This software is defined as a container if it is packaged with the part of the operating system that it uses. In this way, any software is packaged with as much size as it needs. This package is called a container. In this technology, each application has its own dependencies, libraries and files. It is possible to run many isolated applications, even different versions of the same application, on the same physical machine with very low system resources. To summarize: In order for software to run, a computer with an operating system installed on it is required. This process is actually quite expensive. However, thanks to container technology, the costs are significantly reduced, especially by using the functions that are sufficient for the operating system itself and the limited system resources.

In this study, an alternative virtualization platform based on containers was created for the distance learning system. Containerized software development environments for use in software development training are to be made available to distance learning students through this platform. The platform, developed with the MVC layered model, uses Docker containers to provide cross-platform compatibility with the .Net Core 5.0 infrastructure. The study used the Visual Studio Code software development editor container, which supports common software languages for testing purposes and whose capabilities can be extended through plug-ins. There is a major pandemic in the world. During the pandemic period, students stay away from schools and participate in distance learning activities. The work is important for distance learning students and students who need a software development lab. Similar labs to be established will be of great benefit to students during the pandemic period.

Key Words: Virtualization, Container Technology, Robot Operating System, Microservices, Docker

TEŞEKKÜR

Tez çalışmamın her aşamasında yardımlarını esirgemeyen ve bilimsel konularda büyük destek veren tez danışman Hocam, Sayın Doç. Dr. Atilla ERGÜZEN ve araştırma konusunun değerlendirilmesinde ve testlerinde desteğini esirgemeyen Hocam, Sayın Dr. Öğr. Üyesi Erdal ERDAL'a teşekkür ederim. Tez izleme sürecinde çalışmamızı geri bildirimleriyle destekleyen, yol gösteren Hocam, Prof. Dr. Tamer EREN'e teşekkür ederim. Testlerin gerçekleştirilmesinde desteğini esirgemeyen Kırıkkale Üniversitesi Bilgi İşlem Daire Başkanlığı personeli Sayın Süleyman Bekir DURSUN'a teşekkür ederim. Son olarak her konuda olduğu gibi eğitimim ve tezimin hazırlanması sırasında desteklerini ve yardımlarını esirgemeyen eşim Sevgi ÖZCAN'a ve çocuklarıma teşekkür ederim.

İÇİNDEKİLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

Şekil Sayfa

KISALTMALAR DİZİNİ

LK	Linux Konteyner
SM	Sanal Makine
HOK	Hizmet Olarak Konteyner
KÖA	Kablosuz Örgü Ağı
BT	Bilişim Teknolojileri
SY	Sanallaştırma Yöneticisi
YTA	Yazılım Tanımlı Ağ
KÇAD	Kitlesel Çevrimiçi Açık Ders
MİB	Merkezi İşlemci Birimi
REB	Rastgele Erişimli Bellek
HOA	Hizmet Olarak Altyapı
HOP	Hizmet Olarak Platform
HOY	Hizmet Olarak Yazılım
YGUE	Yazılım Geliştirme ve Uygulama Ekibi
MHM	Mikro Hizmetler Mimarisi
HOM	Hizmet Odaklı Mimari
ÜMAP	Üst Metin Aktarım Protokolü
TDA	Temsili Durum Aktarımı
ÜMİD	Üst Metin İşaretleme Dili
JSNG	JavaScript Nesnesi Gösterimi
GİD	Genişletilebilir İşaretleme Dili
UPA	Uygulama Programlama Ara yüzü
BDEP	Basit Dizin Erişim Protokolü
LWAS	Linux için Windows Alt Sistemi
İSAT	İkinci Seviye Adres Tercümesi
UVD	Uzun Vadeli Destek
MEKA	Medya Erişim Kontrol Adresi
SYAA	Sanal Yerel Alan Ağı

DGİ	Docker Güvenli İçerik
HR	Hizmet Reddi
KK	Kullanıcı Kimliği
GKK	Grup Kimliği
MGD	Model Görünüm Denetleyicisi
VÇ	Varlık Çerçevesi
EGO	Entegre Geliştirme Ortamı
ÖYS	Öğrenme Yönetim Sistemi
GPM	Güvenilir Platform Modülleri
LAPP	Linux Arka Plan Programı
RKD	Rocket Konteyner Dosyası
KÇZA	Konteyner Çalışma Zamanı Ara yüzü
AAK	Ağ Ara yüz Kartı
VAD	Virgülle Ayrılmış Değer
SGÇİ	Saniyede Giriş/Çıkış İşlemi
Kb	Kilobit
KB	KiloByte
Mb/s	Megabit/saniye
MB	MegaByte
RİS	Robot İşletim Sistemi
GRİS	Güvenli Robot İşletim Sistemi
UE	Uzaktan Eğitim
UMP	Uzak Masa üstü Protokolü
Nİ	Nesnelerin İnterneti
PMD	Programlanabilir Mantıksal Denetleyici

1. GİRİŞ

Günümüzde teknolojinin hızla gelişmesi hayatın büyük bir bölümünü çevrimiçi yaşanır duruma getirmiştir. Mobil teknolojilerin gelişmesi ve internetin yaygınlaşmasıyla kullanıcılar yüksek bağlantı hızlarına sahip akıllı telefon ve tabletleriyle her an sosyal medyaya erişir durumdadır. Teknolojik değişimin hızlı yaşandığı bir dönemde tüm kuruluşlar gibi eğitim kurumları da etkilenerek birçok eğitimlerini çevrimiçi olarak sunmaya başlamıştır [1]. Önceden sadece uzaktan eğitim sistemine kayıtlı öğrenciler için sunulan içerikler ve daha fazlası artık tüm öğrenciler için erişilebilir duruma gelmiştir. Ayrıca bulut ortamında yayınlanan ücretsiz eğitim içerikleri bu çabaya katkıda bulunmuştur. Bulut teknolojileri sanallaştırma üzerine kurulmuştur [2]. Sanallaştırma, fiziki bir kaynağı ihtiyaca göre mantıksal parçalara ayırma ve sunucu verimliliğini en iyi duruma getirme olarak ifade edilebilir. Sanallaştırma, bilgisayardaki işlemci, bellek, disk ve ağ kartı donanımlarının yazılımsal formudur [3]. Sanallaştırma ile sunucu donanımları verimli ve güvenli biçimde yönetilebilir.

Sanallaştırma teknolojisi sunucular üzerinde farklı alanlara uygulanabilmektedir. En yaygın kullanım alanları; Sunucu, masaüstü, ağ, depolama ve uygulama sanallaştırmadır [4]. Veri merkezlerinde sunucu sanallaştırma, paylaşım ve kullanımı artırırken bakım masraflarını azaltarak donanım maliyetlerini düşürür. Kişisel bilgisayarların yanında akıllı telefon, tabletlerde uygulanan masaüstü sanallaştırma, verinin depolanması ve yedeklenmesi işlemlerini kolaylaştırır ve iş sürekliliğini artırır. Çoklu servis kullanan sunucularda ağ sanallaştırması, birbirinden izole sanal ağlar meydana getirir. Sunucularda bulunan fiziksel depolama alanları, depolama sanallaştırma ile mantıksal olarak farklı boyutlarda kullanıcılara sunulabilir. Kullanıcıların uygulamaları kendi bilgisayarına kurmadan merkezi sunucu ortamında kullanmalarına imkân sağlayan uygulama sanallaştırma teknolojisidir. Uygulama sanallaştırma günümüzde her sektörde yaygın olarak kullanılmaktadır [5]. Bu

teknoloji ile çok sayıda kullanıcının aynı uygulamayı bir sunucudan paylaşabilmesi mümkündür. Böylece son kullanıcılar çalıştırdıkları uygulamanın kurulması, güncellenmesi ve sorunların giderilmesi gibi olaylardan kurtulmuş olur. Bu teknolojiyle uygulamalar tek merkezden kurulur, lisanslar kolay yönetilir, uygulamalar gruplara göre kolayca tanımlanır ve internet bağlantısı olan tüm cihazlardan erişim mümkün hale gelir.

Uygulama sanallaştırma teknolojisinde, uygulamaların mantıksal olarak tanımlanan fiziksel kaynakları kullanabilmesi için arada bir katman oluşturulmakta ve kaynak yönetimi bu katman üzerinden yapılmaktadır. Bu durum sanallaştırma yazılımlarının fiziksel makineye göre performansını düşürmektedir [5]. Uygulama sanallaştırma ortamlarının performansının artırılması için pek çok çalışma yapılmaktadır [6]. Son yıllarda uygulama sanallaştırma ortamlarında Docker yaygın olarak kullanılmaktadır. Günümüzde küresel firmaların çoğu sanallaştırma ortamlarında barındırılan hizmetlerinde Docker teknolojisini kullanmaktadır. Docker, uygulamaların ihtiyaç duyduğu bileşenleriyle birlikte paketlenmesini, taşınmasını ve çalıştırılmasını sağlayan bir teknolojidir. Bu teknoloji konteyner imajlarını kullanmaktadır [7].

Konteyner imajları, çalışabilmesi için gerekli olan kod, araç ve kaynaklara sahip bağımsız yazılım parçalarıdır. Konteyner uygulamalar için güvenli, izole bir çalışma ortamı sağlar. Docker, konteynerlerin oluşturulması ve yönetimi konusunda başarılı çözümler sunmaktadır. Docker konteynerlerinde ayrı bir işletim sistemi bulunmaz. Mevcut işletim sistemine ait kaynakları (bellek, işlemci, ağ ve disk) kullanır. Docker konteyner geleneksel sanal makinelerin aksine Linux çekirdeği ve LK kullanır. Bu sebeple işlemci ve belleğin aşırı kullanımı en aza indirgenmiştir [8]. Internet üzerinde bulut hizmeti veren firmalar konteyner hizmeti de (HOK) vermektedir [9]. HOK sanal makinelere göre çok avantajlıdır. Konteyner çok hızlı başlatılır. Sanal makinelere (SM) göre daha az sistem kaynağı kullanır. Aynı anda çok fazla uygulama konteyneri çalıştırılabilir.

1.1. Tezin Amacı ve Önemi

Bu çalışmada, uzaktan eğitim sistemi eğitimlerinde kullanılmak üzere geleneksel sanallaştırma yöntemlerine alternatif olarak geliştirilmiş esnek ve ölçeklenebilir konteyner tabanlı yazılım geliştirme platformu önerilmektedir.

Sanallaştırma tabanlı laboratuvarlar [10] ile uzaktan öğrenim [11] yıllardır yapılmaktadır. Son yıllarda bu yaklaşım bulut platformlarında uygulanmaktadır. Ancak sanallaştırmanın yapısı gereği her sanal makine (SM) için bir sunucu örneği kopyalanmaktadır.

Sanallaştırılmış bir sunucuda her sanal makineye ait sürücüler, ikili dosyalar ile kitaplıklar ve aynı uygulamaya ait eksiksiz bir işletim sistemi bulunur [12]. Bu durum fiziksel kaynakların ortak kullanımı yanında birden fazla işletim sisteminin aynı anda çalıştırılmasına neden olmaktadır.

Günümüzde yaygınlaşmaya başlayan konteyner teknolojisi ise sanallaştırmanın aksine mevcut bir işletim sistemi çekirdeğini (kernel) birbirinden izole çalışan pek çok uygulamanın ortak kullanımına izin vermektedir. Bu durum mevcut sistem kaynaklarının daha etkin kullanımını sağlamaktadır.

Konteyner teknolojisi yazılım geliştirme alanında yaygın kullanım alanına sahiptir. Aynı zamanda konteynerler yazılım geliştirme eğitimlerinde de kullanılabilmektedir. Günümüzde uygulama geliştirmek için sadece kod yazmak yeterli görülmemektedir. Bunların yanında yazılımın yaşam döngüsü için gerekli olan tüm bileşenlerin kontrolünü de gerektirmektedir. Docker konteyner teknolojisi yazılımcılara çalıştıkları her proje için seçtikleri araçları, uygulama yığınlarını ve dağıtım ortamlarını kontrol etmelerine imkân tanımaktadır [13].

Çalışma ile kampüs ağlarında oluşturulan konteyner tabanlı yazılım geliştirme laboratuvarlarında geleneksel sanallaştırma servislerine alternatif bir platform oluşturulmuştur. Bu platform üzerine yüklenebilen konteyner tabanlı yazılım geliştirme uygulamalarının uzaktan bağlanan öğrencilere servis edilmesi sağlanmaktadır. Çalışma farklı platformlarda (Linux, Windows) çalışabilmesi için.

Net Core ortamında geliştirilmiştir. Bu yöntem sistem kaynaklarının yetersizliği, bant genişliği, yavaş başlatma ve lisans problemlerini önleyebilecektir.

Bu çalışmada kampüs ağlarında mevcut bulunan sistem kaynakları ile platform bağımsız çalışabilecek, güvenli, Docker konteyneri tabanlı sanal yazılım geliştirme donanımı mimarisi üzerinde çalışılmıştır. Önerilen mimarinin ilk örnek uygulama detayları tartışılmıştır. Bu çalışma ile amacımız bulut kaynaklarını en iyi şekilde kullanmak ve aynı anda daha fazla kullanıcıya hizmet verebilmektir.

1.2. Tezin Organizasyonu

Tezin sonraki kısımlarında sırasıyla: Bölüm 2'de eğitim alanında konteyner teknolojisi ve sanallaştırma ile ilgili araştırmaları içeren literatür taraması sunulurken, son yıllarda kullanılan yöntemlerdeki genel yönelim özetlenmiştir.

Bölüm 3'te tez çalışması kapsamında ele alınan materyaller ve önerilen modele ilişkin yöntemler açıklanmıştır. Bölüm 4'te tez çalışmasında elde edilen bulgular ve bulguları elde ederken izlenen süreçler raporlanmış, önerilen model ile literatürdeki çalışmalar karşılaştırılmıştır. Bölüm 5'te ise, araştırma sonuçları ve gelecek çalışmalara yönelik öneriler sunulmaktadır.

2. LİTERATÜR

Konteyner teknolojisi son yıllarda sanallaştırma teknolojisine alternatif olarak görülse de sanallaştırma teknolojisini desteklemek amacıyla pek çok alanda kullanılmaktadır. Bu teknolojinin eğitim alanında kullanıldığına dair pek çok çalışma yapılmaktadır. Eğitimde sanal laboratuvarlar ilgili yapılan pek çok çalışma bulunmaktadır. Sanallaştırma ve konteyner teknolojisinin eğitim alanında kullanımıyla ilgili çalışmalar bu bölümde ele alınmıştır.

Önceki yıllarda yapılan çalışmalar incelendiğinde ve arkadaşları Kablosuz Örgü Ağları (KÖA) yönlendirme protokolleri ve yazılım ortam uygulamalarını analiz etmiştir. Ağ tasarım kararları için yazılım sistemi tasarlanmış ve uygulanmıştır [14]. Uygulama ile grafiksel kullanıcı ara yüzü oluşturulmuş (Sanal Örgü Ağları) ve kullanıcılara kendi ağ topolojilerini oluşturma, yapılandırma imkânı sağlamıştır. Peredo ve arkadaşları [15], etkileşimli ses sistemi ve Java programlama için sanal laboratuvar oluşturdu. Sistem karmaşıklığını azaltan akıllı web eğitim sistemi, etkileşimli çoklu ortam ara yüze sahip, ihtiyaçlara göre uyarlanabilen web öğrenme ortamına sahiptir. Cabrera ve diğerlerinin [16] yapmış olduğu çalışmada (GILABVIR-Grup d'Interès en Laboratoris Virtuals I Remots) ise yükseköğrenim uyum sürecini değerlendirmek ve farklı alanlarda öğrenim gören mühendislik öğrencileri için özel kurulan 8 uzaktan erişim laboratuvarı incelenmiştir.

Günümüzde bilişim teknolojileri müfredatları sanallaştırma yöneticisi (hipervizör) tabanlı sanallaştırma ortamlarında yaygın olarak kullanılmaktadır. Bu yöntem için amaca uygun donanım, kaynak ve yazılım gereksinimi bulunmaktadır. Bu sorunların çözülebilmesi için, Jiang ve arkadaşı sınıfta ve laboratuvarda kullanılmak üzere konteyner tabanlı sanallaştırmanın ön araştırmasını yapmıştır. Yeni teknolojinin avantajları yanında sınırlamalara sahip olduğunu gözlemlemiştir [17]. Zhou ve arkadaşları, Docker konteyner teknolojisi kullanarak Phyton programlama kursunun öğretim deney platformunu oluşturmak için yeni bir yöntem önermişlerdir [18]. Bu

yöntem ile öğrencilerin ihtiyacına göre çalışan ortam görüntülerinin dağıtımı ve yönetimi gerçekleştirilebilmekte ve kaynak kullanımı, eş zamanlı kullanım, yanıt süresi bakımından avantajları yanında sanallaştırma yöneticisi tabanlı sanallaştırmaya göre düşük kurulum maliyetlerine sahiptir. Tobarra ve arkadaşlarının yaptığı çalışmada [19] BT eğitimlerinde konteyner tabanlı sanallaştırmanın öğrenciler tarafından benimsenmesi araştırılmıştır. Araştırma için kurulan deneysel ortamda sanallaştırma yöneticisi (SY) tabanlı sanallaştırma ortamları ile konteyner tabanlı sanallaştırma ortamları karşılaştırılmıştır. Çalışma sonunda öğrencilerin BT eğitim ortamı olarak konteyner tabanlı platformları; algılanan fayda, tahmini çaba, sosyal etki, tutum, erişim kolaylığı ve kullanım amacı bakımından yüksek oranda benimsedikleri gözlemlenmiştir.

Wang ve arkadaşları mühendislik eğitimi için ihtiyaç duyulan karmaşık programlama becerileri için eğitim platformu önermişlerdir [20]. Educoder adı verilen çözüm önerisinde klasik kod yazımının yanında Docker, Spring, Hadoop Machine Learning gibi karmaşık teknolojilerin eğitimi desteklenmektedir. Çalışmada karmaşık eğitim teknolojilerini kullanmak için Docker Web Terminal ara yüzü geliştirilmiştir. Fletscher ve arkadaşları özel bir yazılım yüklemeden bilgisayar ağ uygulamalarını çalıştıran bir platform geliştirdiler. Çalışmada konteyner tabanlı YTA alanını hedefleyen bir KÇAD önerilmiştir [21]. Önerilen platform birkaç kullanıcıyı aynı anda destekleyebilme ve teknolojik olarak ölçeklenebilir özelliktedir. Kozhirbayev ve arkadaşı ise bulut ortamında çalışan sanallaştırma platformları ile konteyner tabanlı sanallaştırma platformlarının performanslarını karşılaştırmıştır [22].

Literatür incelendiğinde SY tabanlı sanallaştırma ortamlarının uzaktan eğitimde halen yaygın olarak kullanılmakta olduğu görülmektedir. Bunun yanında konteyner tabanlı sanallaştırma çözümlerinin sınırlı alanda kullanımları görülmektedir. Konteyner tabanlı sanallaştırmanın yeni bir yöntem olması ve uygulamada karşılaşılan sorunlar sebebiyle yaygınlaşmasının süre alacağı görülmektedir.

2.1. Sanallaştırma Teknolojilerinin Gelişimi

Sanallaştırma teknolojisi ile mevcut fiziksel donanıma birden fazla işletim sistemi kurularak, bir makine üzerinde birden fazla işletim sisteminin çalıştırılması

sağlanmaktadır. Böylece tek fiziksel sisteme ait kaynaklar üzerinde kurulu bulunan sistemler tarafından paylaşılmakta ve gerekli görülen durumlarda kaynakların ihtiyaç duyulan sanal sistemlere aktarılması sağlanabilmektedir. Yoğun işlemci ve bellek ihtiyacı duyan sanal işletim sistemine, o an boşta olan sanal işletim sisteminin bellek ve işlemcileri atanabilmektedir. Böylece mevcut fiziksel kaynaklar efektif olarak kullanılabilmektedir.

Sanallaştırma teknolojileri kurumların yazılım maliyetlerini düşürmektedir. Tek lisansla kullanılan bir yazılım birden fazla sanal makine üzerine kurulabilmekte ve çalıştırılabilmektedir. Bir fiziksel sunucu üzerine kurulan birden fazla sanal sunucu ile sahiplenme, bakım ve operasyon maliyetleri düşürülmektedir. Ayrıca daha az fiziksel sunucunun bulundurulmasıyla sistem odalarında bulunan elektrik, soğutma ve raf maliyetleri düşürülmektedir [23]. Fiziksel sunucunun kurulumu yapılarak hizmete sunulması saatler hatta günler alabilmektedir. Sanallaştırma teknolojisiyle önceden hazırlanmış bir şablon ile bu işlem dakikalar içinde gerçekleştirilebilmektedir [24].

Ancak sanallaştırma teknolojisinin de yetersiz kaldığı durumlar ortaya çıkmıştır. Uygulama ve verilerin sadece bir veri merkezine bağımlı kalması, uygulama ve verilerin veri merkezleri arasında paylaşılması durumunda klasik sanallaştırma yöntemleri yetersiz kalabilmektedir. Bu amaçla veri ve uygulamaların farklı veri merkezleri arasında taşınabilmesi, yönetilmesi ve izlenebilmesi yanında daha fazla esnek yapıya sahip olması sebebiyle bulut teknolojisi günümüzde yaygın olarak kullanılmaktadır. Bununla birlikte Spiceworks firmasınca yapılan araştırmada, Kuzey Amerika ve Avrupa'daki kurumsal sanallaştırmanın önümüzdeki yıllarda daha büyüyeceği, sanallaştırma ortamlarının da buluta taşınacağı öngörülmektedir [25]. Araştırmaya göre, sunucu sanallaştırma işletmelerin %92'si tarafından kullanılmakta ve gelecek birkaç yıl içerisinde masa üstü, veri, ağ sanallaştırma kullanımında çift haneli büyüme beklenmektedir.

Bulut teknolojisi ile verilerin internet üzerinde depolanması, uygulama ve verilere her yerden erişilebilirlik sağlanmıştır. Bulut teknolojisini sanallaştırmadan ayıran en büyük özellik budur. Bulut teknolojisi için internet erişimi zorunludur. Büyük veri merkezleri tüm kaynaklarını internet üzerinden kişi ve kuruluşların erişimine açarak bulut platformunu oluşturmaktadır. Bulut altyapısında sanallaştırma teknolojileri kullanılmaktadır. Bulut platformunda ücreti karşılığında kullanıcılara istenilen

donanıma sahip sanal makineler tahsis edilmektedir. Ayrıca bulut hizmet sağlayıcıları isteğe göre altyapı (HOA), platform (HOP), yazılım (HOY) hizmetleri de sunabilmektedir.

Sanallaştırma için mevcut fiziksel donanımın üzerine birden fazla sanal işletim sisteminin kurulması ve sanal işletim sistemlerinin mevcut kaynakları (işlemci, bellek, disk) paylaşması esastır. Donanımın sanallaştırılmasını benimseyen bu yöntem yaygın olarak kullanılmaktadır. Donanım sanallaştırma ana makineden ayrı sanal bir fiziksel ortam yaratmaktadır. Bu sebeple çok tercih edilmektedir. Son yıllarda bu yönteme alternatif olarak soyut sanal makine kullanımı [26] yaygınlaşmaktadır. Bu bölümde iki sanallaştırma yaklaşımı açıklanacaktır.

2.1.1. Hipervizör Tabanlı Sanallaştırma

Donanım tabanlı sanallaştırma uygulamaları için sistem kaynaklarını oluşturulan sanal makinelere tahsis eden uygulamalardır. Bu uygulamaları iki grupta incelemek mümkündür. Birincisi; donanım üzerine doğrudan kurulan (Bare-Metal) uygulamalardır. Oracle VM, Citrix Xenserver, Microsoft Hyper-V, Vmware ESXi/ESX yazılımları donanım üzerine kurulan sanallaştırma yöneticilerindendir.

İkincisi; işletim sistemi üzerine kurularak sistem kaynaklarının paylaştırılmasını sağlayan sanallaştırma yönetimi (SY) uygulamalarıdır. Hyper-V for Windows, Vmware Workstation, Vmware Player, Virtualbox, Qemu uygulamaları bir işletim sistemi üzerine kurularak çalışan sanallaştırma yöneticileridir.

2.1.2. Konteyner Tabanlı Sanallaştırma

SY tabanlı sanallaştırmanın aksine konteyner sadece konumlandığı işletim sisteminin çekirdeğine ortak olmaktadır. Böylece daha az kaynak kullanarak hızlı hizmet sunmaktadır. Konteyner yapısı yazılımcılar için büyük bir problemi çözmektedir. Mevcut bir yazılımın bir bilgisayar ortamından diğerine taşındığında doğru çalıştığından emin olunmasını sağlar. Bu işlemi yapmak için konteyner imajları kullanır. Konteyner teknolojisi 1980'li yıllardan bu yana bilinmektedir. Unix kullanıcıları "chroot" komutu ile günümüze benzer yalıtılmış işlemleri başlatabilmekteydi. Konteyner teknolojisi son yıllarda sunucular üzerinde çalıştırılan

yazılımların mikro-servis yapısıyla minimize edilerek hafif ve esnek yapıya kavuşturulmasıyla oldukça popüler olmuştur.

2.2. Sanallaştırma Teknolojilerinin Eğitim Ortamlarında Kullanımı

Web tabanlı uzaktan eğitim günümüzde daha fazla tercih edilen bir eğitim yöntemi olmuştur. Özellikle 2019 yılında ortaya çıkan Covid-19 pandemisi sebebiyle örgün eğitim ortamlarının kapatılması uzaktan eğitimi zorunlu kılmıştır [27]. Uzaktan eğitimde bilişim altyapısının kullanımı bu sebeple daha fazla önem kazanmıştır.

Öğrencilere internet üzerinden erişilebilen, zaman ve mekân sınırı olmaksızın, teknolojik alt yapı kullanılarak eğitim ortamlarının sunulması mümkündür. Günümüzde ders içeriklerinin ve uygulamaların sanallaştırılarak kullanıcılara internet aracılığıyla sunulduğu sanal sınıflar ve sanal laboratuvarlar kullanılmaktadır [28]. Bu ortamlarda belirli bir ders içeriği sunulabildiği gibi, öğrencide mevcut olmayan bir uygulamanın da internet üzerinden sunumu gerçekleştirilmektedir.

Sanallaştırma teknolojisi kullanılarak, yazılım geliştirme eğitimleri için platformdan bağımsız olarak öğrencilerin yazılım geliştirme ortamlarına erişimi mümkün olmaktadır. Böylece öğrenciler düşük sistem özelliği olan bilgisayarlarına yazılım kurmadan, karmaşık ayarlamalarla uğraşmadan sanal laboratuvara uzaktan erişerek yazılım geliştirme platformlarını kullanabilmektedir [29].

Son yıllarda dünyada ve ülkemizdeki yükseköğretim kurumlarında platform sanallaştırma, masaüstü sanallaştırma, uygulama sanallaştırma yöntemleri kullanılarak ofis yazılımları, yazılım geliştirme ortamları ve özel uygulamaların uzaktan erişilerek kullanıldığı görülmektedir [30].

2.3. Konteyner Teknolojisinin Eğitim Ortamlarında Kullanımı

Eğitim kurumları için başarılı laboratuvarların geliştirilmesi önemli olduğu kadar maliyetlidir. Değişim ve yenilik konusunda dinamik yapıya sahip olan mühendislik eğitimlerinde pratik öğrenme senaryoları yeterlilik kazanımı ve uygulamalı beceri elde etme çok daha önemlidir [19]. Tüm mühendislik ilgi alanları gereksinimlerini karşılayacak tümleşik bir laboratuvar platformunun oluşturulması ise oldukça

pahalıdır. Laboratuvarlar gerekli alt yapının sağlanması, heterojen çalışma ortamlarının oluşturulması, bireysel çalışmaların özendirilmesi gibi zorluklarla başa çıkabilmelidir [31]. Bu amaçla günümüzde sanallaştırma teknolojilerinin yanında konteyner tabanlı ve açık kaynak destekli laboratuvarlar tercih edilmektedir. Son yıllarda farklı eğitim alanlarında ve özel laboratuvar gereksinimi duyulan sanal ortamlara bu ilgi artmaktadır. Mühendislik eğitimlerinde yüz yüze eğitimin yanında uzaktan eğitim için; Programlama dillerinin eğitimi [32] ve geliştirme ortamlarının konteyner tabanlı sanallaştırılmasında [32,20,33,34] kullanılmaktadır. Özellikle yazılım geliştirme alanında popüler bir yaklaşım olan yazılım geliştirme ve bakım uygulamalarının yükseköğretimde uygulanması [36], akıllı robotik eğitimleri için RİS tabanlı açık web ortamlarının oluşturulmasında [37] konteyner tabanlı sanallaştırma çözümleri kullanılabilmektedir.

Hatta bazı üniversiteler web tabanlı çevrim içi eğitim ortamlarında uygulamalı deneysel öğrenme deneyimi oluşturma hedefli, konteynerlerin doğrudan web sayfasından çalıştırılmasına imkân veren konteyner servisleri geliştirmiştir. Bu sayede öğrenciler tarafından konteynerlerin kullanılması, öğrenci katılımının artırılarak sonuçların iyileştirilmesi hedeflenmiştir [38].

Ayrıca bazı üniversitelerde siber güvenlik eğitimlerinde kullanılmak üzere konteyner tabanlı sanal laboratuvarlar geliştirilmiştir. Bu laboratuvarlarda ihtiyaç duyulan ve kurulum ve ayarlamaları zahmetli olan NETLAB+, NS-3, SEED, GENI gibi araçların platformdan bağımsız olarak öğrencilerin kullanımına sunulmuştur [39]. Çalışma sonunda uygun maliyetli, heterojen platformlu, ölçeklenebilir, donanım ve yazılımdan bağımsız bir öğretim ortamı oluşturulmuştur.

Yazılım mühendisliği eğitimlerinde yazılım geliştirme ve bakım konusu önemli bir yer tutmaktadır. YGUE, yazılım geliştirme ve operasyonların birleşiminden oluşmaktadır. Yazılım geliştirme ve bakım ile daha iyi ve güvenilir yazılımlar üretmek, koordinasyonu sağlamak, müşteri gereksinimlerine cevap verebilmek mümkün hale gelir. Bu amaçla üniversitelerdeki yazılım geliştirme ve bakım eğitimleri için çalışmalarda yazılım geliştiricilerin ve profesyonellerin yaşam boyu öğrenme açısından konteyner araçlarına çok fazla ihtiyaç duydukları görülmüştür [31,32].

Üniversiteler tarafından düzenlenen kurslarda farklı disiplinler için temel haline gelen Python programlama dili konteyner tabanlı olarak öğrenciler tarafından kullanılmıştır. Bu yöntemle ortam görüntüsü dinamik olarak dağıtılmış, kaynak kullanımı optimize edilmiş ve yanıt süresi azaltılmıştır [18].

2.4. Konteyner Teknolojisinin Endüstriyel Ortamlarda Kullanımı

Linux konteynerlerindeki gelişmeler sadece belirli uygulamaların sanallaştırılması ile sınırlı kalmamıştır. Endüstride önemli bir yeri olan robot teknolojisi de konteyner teknolojisine ilgi duymaktadır ve Robot İşletim Sistemi (RİS) ile Docker arasında sürekli büyüyen bir kesişim alanı oluşmaktadır [41]. Robotik planlama projesinde (MoveIt!) Docker ve Robot İşletim Sistemi kullanmaktadır. Ayrıca Güvenli Robot İşletim Sistemi (GRİS) gibi projeler de konteyner teknolojisini kullanmaktadır. Robotikte sürekli bütünleşme ve doğrulama için uygun bir iş akışı bulunmadığından test etme ve yeniden kullanım için çalıştırılabilen yazılımlara ihtiyaç bulunmaktadır. Açık kaynaklı RİS kodların test edilebilmesi için eksiksiz ve çalıştırılabilir ortama ihtiyaç vardır [42]. Karmaşıklığın fazla olması sebebiyle robotik araştırmalarda deneylerin tekrarlanabilmesi oldukça güçtür [43].

Bu amaçla robotik sistemler için robot işletim sistemi uyumlu doğrulama ortamının oluşturulması, düzenlenmesi ve dağıtımı için çalışmalar devam etmektedir. Doğrulama ortamında Docker kullanarak sistemin taşınabilirliğini sağlamak ve kaynakları zamanında işleyebilmek mümkündür [43,44]. Linux konteynerlerindeki gelişmeler sayesinde tekrarlanabilir, yeniden üretilebilir ağlar oluşturma ile birlikte taşınabilir RİS uygulamalarını çalıştırılabilmektedir [41]. Bulut bilişim ve konteyner teknolojisi gibi sunduğu hizmetler sayesinde robotlara yüksek performanslı bilgi işlem kaynakları sunulabilir. Bu hizmet robotların özerkliğini ve işbirliği kapasitesini artıracaktır [46]. Araştırmacılar robotik sistem algoritmalarının kolayca ekleneceği veya düzenleneceği bulut uygulamaları üzerinde çalışmaktadır. Bu çalışmada destek alınan teknolojiler ise, sanallaştırma, Docker, Gazebo ve robot işletim sistemidir [45,44].

Konteyner teknolojisi Endüstri 4.0 çalışmalarında da kullanılmaktadır. Soto ve arkadaşları tarafından yapılan çalışmada, Endüstri 4.0'la ilgili olarak ürün hatalarının

erken tespiti için çevrimiçi bir makine öğrenmesi modeli oluşturulmuştur. Elektronik cihaz üretim hattı için özgün ve gerçekçi bir benzetim yapılmıştır [48]. Barrenechea ve arkadaşlarının yaptığı çalışmada, Endüstri 4.0 için kullanılabilecek teknolojik mimarilerin analizi yapılarak, takım tezgâhı sektörünün servis edilebilmesi için ölçeklenebilir bir mimari tanımlanmaktadır. Farklı Nesnelerin İnterneti (Nİ) cihazlarının birbirleriyle haberleşebilmesi için Docker konteyner teknolojisinin kullanımı gerçekleştirilmiştir [49].

Mellado ve arkadaşlarının yaptıkları çalışmada, Endüstri 4.0 bağlamında Nesnelerin İnterneti ve Programlanabilir Mantıksal Denetleyici (Nİ-PMD) cihazlarının tasarımında konteyner tabanlı programlanabilir mantıksal denetleyici kullanılmıştır. Endüstri 4.0 için önerilen yapı (Nİ-PMD), her bir işlevsellik için ayrı bir konteyner önermektedir. Önerilen yapı, kontrol yeteneğine, filtreleme, sis hesaplama işlevlerine ve bağımsız kablosuz arabirime sahiptir [50]. Teoh ve arkadaşlarının yaptığı çalışmada, Endüstri 4.0'da makine öğrenimi kullanarak varlık yönetimi için Nesnelerin İnterneti (Nİ) ve sis hesaplama modeliyle öngörücü bakım hizmeti önerilmiştir. Çalışmada üretim cihazlarının arızasını önceden tahmin edebilmek için, nesnelerin interneti cihazlarında kullanılan varlık tanıma etiketleriyle tüm üretim hattını etkileyecek arıza öncesinde ilgili parçayı yenilemek veya onarmak için karar vermektedir [51].

3. MATERYAL VE METOT

Bu bölümde çalışmamızın temelini oluşturan Mikro Servis Mimarisi (MSM), Docker teknolojisi mimarisi ve konteyner teknolojileri açıklanacaktır. Ayrıca çalışmamızda kullandığımız teknolojiler ve kullanım nedenleri ve önerilen araştırma modelinin gelişim aşamaları belirtilecektir.

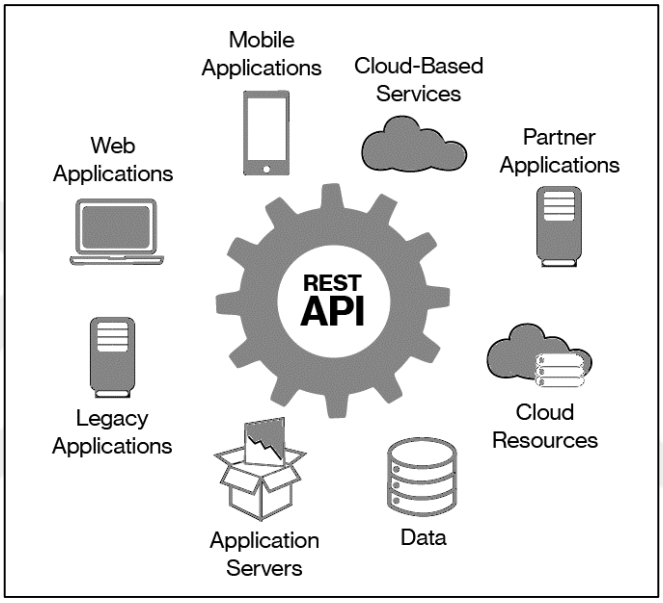
3.1. Mikro Servis Mimarisi

Bulut hizmetlerinde kullanılan yerel bir mimari olan Mikro Hizmetler Mimarisi (MHM), Hizmet Odaklı Mimari'den (HOM) ilham alınmıştır. Geleneksel olarak mikro hizmetler geliştirilebilen, test edilen ve dağıtılan küçük hizmet paketlerinden oluşur [52].

Mikro servisler, birlikte çalışan bağımsız küçük kodlardan oluşan programlardır. Geleneksel yazılım geliştirme yöntemlerinde taban kodlarının üzerine yeni kodlar eklenmek suretiyle yazılıma yeni özellikler eklenmektedir. Böylece kod tabanı genişler. Ancak kod tabanının büyümesiyle yapılacak değişikliklerde veya hata durumlarında sorunun çözümü güçleşir [2].

Mikro hizmetler, belirli yazılım görevlerini gerçekleştiren ve web ara yüzleri üzerinden iletişim kuran ayrık olarak dağıtılabilir hizmetlerdir. Mikro hizmet mimarisi tek parça (monolitik) mimariye göre pek çok avantaj sağlamaktadır. Bir hizmetin yürütülmesi esnasında ortaya çıkacak kilitlenme problemi programın diğer bölümlerinin çalışmasını engelleyebilir. Ayrıca bir programın çalışması sırasında geliştiricilerin yazılım üzerinde güncelleme veya değişiklik yapması gerekebilir. Kilitlenme problemlerinin sadece ilgili bölümle kısıtlı kalması ve geliştiricilerin yazılımın diğer bölümleri etkilenmeden yeni hizmetler ekleyebilmesi mikro hizmetler ile mümkün olmaktadır.

Mikro hizmet mimarisi diğer hizmetlerle iletişim kurmak için Temsili Durum Aktarımı (TDA) protokollerini kullanır. Temsili Durum Aktarımı uygulamaları haberleşirken internet sayfalarının önemli parçası olan Üst Metin Aktarım Protokolü (ÜMAP) kullanır. Üst Metin Aktarım Protokolü'nün temelde kullanılıyor olması Temsili Durum Transferi uygulamalarının (TDA) herhangi bir programlama dilinden bağımsız olarak çalışmasına olanak sağlar. Temsili Durum Transferi ile genelde internet aktarım verilerinin (ÜMİD, JSNG, GİD) yanında farklı formattaki verilerde Şekil 3.1'de görüldüğü üzere aktarılabilmektedir.



Şekil 3. 1. TDA Uygulamaları

Temsili Durum Transferi mimarisinin tüm sınırlamalarına sahip olan yazılımlar Tam Temsili Durum Transferi (Tam TDA) yazılımlar olarak ifade edilir. Bir yazılımın Tam Temsili Durum Transferi olabilmesi için aşağıda sıralanan tüm özellikleri taşıması gerekir.

3.1.1. Durumsuzluk

Sunucu üzerinde istemciye ait oturum ve sorgu bilgileri barındırılmaz. Böylece sistem kaynaklarından tasarruf sağlanır. Bu özelliğin avantajları yanında dezavantajlı yönleri de bulunmaktadır. İstemciler tarafından sunucu yönüne yapılacak her istekte bazı bilgilerin tekrar iletilmesi gereklidir. Bu durum ağ trafiğini artırır. Ayrıca sunucunun her seferinde uygulama tutarlığını kontrol etmesi gereklidir [53].

3.1.2. Basit Ara yüz

Değişmeyen ve basit bir internet sayfası ara yüzünün (Tek düzen arayüz) kullanımı mimariyi sadeleştirmektedir. Basit ortak ara yüz ile her parça birbirinden bağımsız olarak geliştirilebilir ve iletişim yöntemi basitleşir.

3.1.3. Talep Üzerine Kod

Sunucular bazı durumlarda fonksiyonelliği artırmak için çalıştırılabilir kod parçaları göndermeyi seçebilmektedir.

3.1.4. Sunucu-Kullanıcı Mimarisi

Bu mimarinin en önemli bileşenlerinden biridir. Kullanıcının sunucu tarafında bulunan veri kaynağı hakkında bilgisinin olmaması ve sunucuya doğru istekler gönderildiğinde doğru yanıtların elde edilmesi sağlanmaktadır. Böylece platformdan bağımsız çalışma sağlanmakta ve ölçeklenebilirlik artırılmaktadır. Ayrıca sunucu ve istemcinin ortak bir ara yüzden kullanılması bağımsız olarak geliştirilmelerine imkân tanımaktadır [54].

3.1.5. Katmanlı Mimari

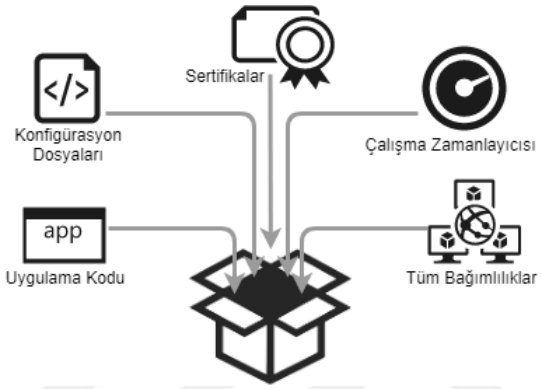
Sunucu mimarisinin katmanlı yapıda olması, istemcinin son sunucuya veya arada bulunan başka bir sunucuya bağlandığı bilgisinin bulunmamasına neden olmaktadır. Böylece arada bulunan sunucular ile yük dengelemesi sağlanabildiği gibi kullanıcılar belirli güvenlik adımlarının uygulanmasına zorlanabilir. Ancak güvenlik için alınacak önlemler istemci-sunucu arasındaki bağlantılarda gecikmelere neden olabilir.

3.1.6. Önbellekte Tutulabilme

İstemci, gönderilen bilgilere göre bir ön bellek mekanizması oluşturabilir. Böylece istemciler uygun olmayan veya bayat veri kullanımının önüne geçer. Bu sayede performans ve ölçeklenebilirlik artırılmış olur [55].

3.2. Docker Teknolojisi

Docker, konteyner teknolojisini kullanarak uygulama geliştirme ve çalıştırma işlemlerini kolaylaştıran açık kaynak kodlu platformdur. Docker platformu, uygulamaları hızlıca derleyip test edilmesine ve dağıtılmasına olanak sağlayan bir çözümdür. Şekil 3.2'de gösterildiği gibi üzerinde çalıştığı işletim sisteminden bağımsız olarak konteynerleri paketleyen ve çalıştıran bir yapıya sahiptir. Özellikle yazılım geliştirme ve dağıtım süreçlerinin basitleştirilmesini sağlayan bir teknolojiye



Şekil 3. 2. Konteyner bileşenleri
sahiptir.

Docker, uygulamaların ihtiyaç duyabileceği tüm kütüphaneler, yapılandırma dosyaları, bağımlılıklar ve uygulamanın çalışması için gerekli tüm bileşenleri paketleyerek konteyner haline getirir. Konteynerler, alt katmanda çalışan işletim sisteminde bulunan servisleri paylaşmaktadır. Docker teknolojisi geleneksel Linux Konteyner (LK) yapısını temel almaktadır. Ancak Linux Konteyner teknolojisi ile önemli farklılıkları bulunmaktadır.

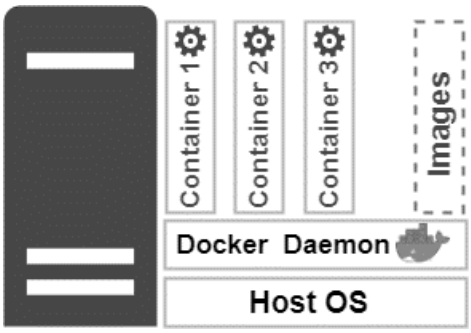
Docker teknolojisi ile geliştirilen konteynerler farklı platformlarda çalışabilmektedir. Bunun yanında Linux Konteynerler sınırlı yeteneklere sahipken Docker teknolojisi ile konteynerler küme yönetimi gibi önemli özellikler kazanmıştır. Bu sayede yüksek kapasiteli işlemleri barındıran uygulamaların konteyner yapılması, yönetilmesi ve ölçeklenebilmesi mümkün hale gelmiştir.

3.3. Docker Mimarisi

Docker mimarisi, istemci-sunucu yapısına dayanmaktadır. Bu mimaride istemci Docker istemci rolünün yüklü bulunduğu ve hizmet alan taraf olarak belirlenmiştir. Sunucu ise üzerinde yüklü bulunan Docker Motoru aracılığıyla Docker istemci tarafından gelen isteklere cevap verilen, konteyner işlemlerinin yapıldığı, takip edildiği taraftır. Sunucu literatürde Docker Sunucu (Docker Host) olarak ifade edilmektedir. Docker Sunucu üzerinde Docker yöneticisi (Docker Daemon) isimli yazılım aracılığıyla konteynerler ile ilgili tüm işlemler yapılabilmektedir. Docker mimarisi dört ana bileşenden oluşmaktadır;

- Docker Sunucu (Host)
- Docker İstemci (Client)
- Docker Kaydı (Registry)
- Docker Nesnesi (Objects)

Docker mimarisinde istemci ve sunucular Şekil 3.3'te görüldüğü gibi ayrı bilgisayarlarda çalışabildiği gibi aynı bilgisayar üzerinde çalışabilmektedir. Ayrı bilgisayarlar üzerinde çalışan Docker mimarisinde Docker istemci üzerinde verilen komutlar Docker sunucuda bulunan Docker yöneticisine iletilir. Docker yöneticisi istemci tarafından istenen uygulama isteklerini imaj, konteyner, depolama (volume) ve ağ işlemlerini yönetmektedir.



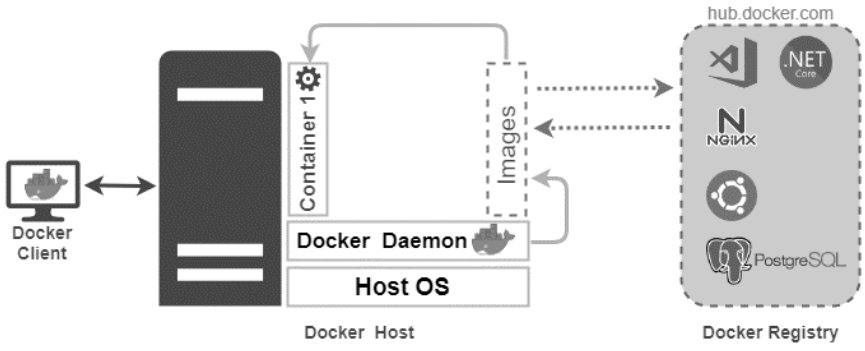
Şekil 3. 3. Docker Mimarisi

Docker Kaydı (Registry), docker içerisinde çalışacak imajların saklandığı bir kayıt defteridir. Docker yüklenmek istenen imaj dosyalarını farklı bir kayıt defteri belirtilmediği sürece varsayılan (https://hub.docker.com) internet adresinde arayacaktır. Bu alan ücretsiz üye olunan herkese açık kayıt defteridir. Aynı zamanda

belirli ücret karşılığında firma düzeyinde kayıt olunarak daha fazla özelliğe sahip olmak mümkündür. Bu adresin dışında farklı firmaların (Google, Amazon, Digital Ocean v.b.) internet üzerinde docker konteyner imajlarını (Docker Image) yayınladıkları kayıt defterleri (repository) bulunmaktadır.

Docker imajı (Image), çalışacak konteynerler için hazırlanmış şablonlardır. Bu şablonlar sunucuya indirilerek bir kopyası oluşturulur ve bu kopya çalıştırılır. Docker imajı, içerisinde konteyner ile ilgili yazılımların yanında gereksinim duyulan üst veriyi de (metadata) bulundurmaktadır. Günümüzde pek çok yazılım firması uygulamalarının imajlarını Docker ve benzeri siteler üzerinden yayınlamaktadır. Kullanıcılar resmi olarak yayınlanan bu imajlara erişebilmekte, kendi sunucularına indirip kullanabilmektedir. Hatta yazılım geliştiriciler Docker, Google, Amazon gibi firmaların internet üzerinde bulunan yazılım depolarında (repository) yazılımlarını yayınlamaktadır. Yazılım geliştiriciler bu depolara internetin olduğu her yerden erişebilmekte ve ilgili konteyneri istedikleri bilgisayarda saniyeler içinde ayağa kaldırmaktadır.

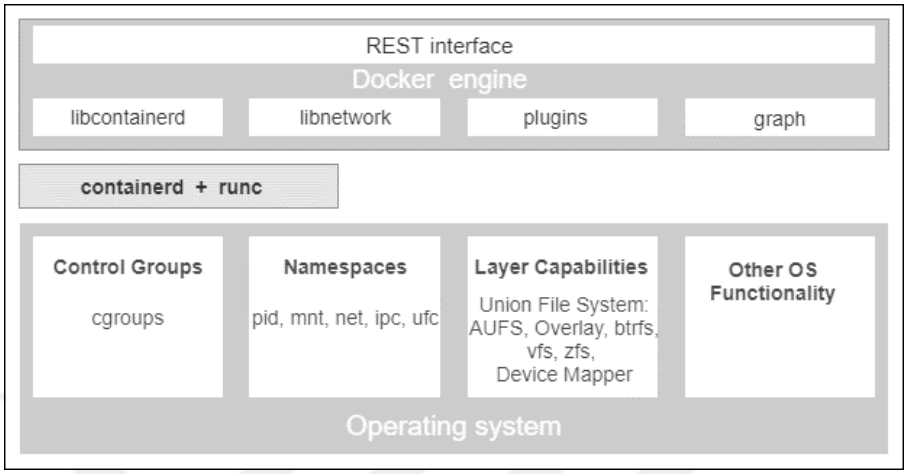
Docker Konteyner (Container), Docker imajlarından oluşturulan çalışan uygulamalardır. Şekil 3.4'te görüldüğü gibi aynı imajın birden fazla kopyası oluşturularak çalıştırılmaktadır.



Şekil 3. 4. Docker kaydı (registry)

Docker İstemci (Client) üzerinden Docker Sunucuya (Host) bir konteynerin çalıştırılması veya çekilmesi isteği geldiğinde sunucu üzerinde çalışmakta olan Docker yöneticisi öncelikle istenen imajın kendi üzerinde yüklü olup olmadığını kontrol eder. Eğer imaj kendisinde yüklü ise bulunan imajın bir kopyasını oluşturarak konteyneri çalıştırır. Sunucu üzerinde istenen imaj yok ise Docker Kaydı üzerinde arama işlemi

gerçekleştirilir. Bulunan imaj önce sunucuya indirilir sonra bir kopyası oluşturularak konteyner ayağa kaldırılır. Docker Sunucu üzerinde konteyner ile ilgili tüm işlemi gerçekleştiren yazılım Docker Motorudur.

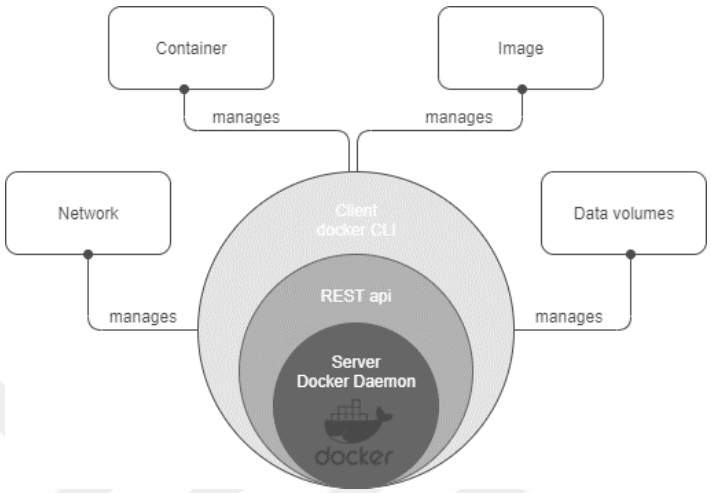


Şekil 3. 5. Docker motoru (engine)

Docker Motoru (Engine) üç bölümden oluşmaktadır. Birinci bölüm işletim sistemi üzerinde yer almaktadır. Şekil 3.5'te görüldüğü gibi bu bölümde bulunan kontrol grupları (Control Groups) ve isim uzayı (Namespaces) Linux işletim sistemi çekirdeğinde bulunmaktadır. Bu bileşenler Unix işletim sisteminin mirası olarak Linux işletim sistemi çekirdeğinde bulunmaktadır. Bu bileşenler Docker yazılımından çok önce Linux işletim sisteminde bulunmaktaydı. Ancak Docker yazılımı sayesinde bilinirliği artmıştır. Docker bu bileşenleri çok iyi yönetmesi, bunların üzerinde çalışan konteynerleri birbirinden bağımsız çalıştırması ve yalıtması sayesinde en bilinen konteyner yönetim aracı olmuştur.

İkinci bölümde ise konteyner çalışma zamanı (Container Runtime) bulunmaktadır. Bu bölüm "containerd" ve "runc" bileşenlerinden oluşmaktadır. Containerd, sistem üzerinde oluşturulan tüm konteynerlerin yaşam döngüsünden sorumlu servistir. Runc ise, konteynerlerin oluşturulması, başlatılması, durdurulması ve silinmesi gibi işlemlerden sorumlu servistir. Üçüncü bölüm ise Şekil 3.5'te en üstte bulunan Docker motoru (Engine) olup, tüm Docker konteyner işlemleri bu bölümden yönetilmektedir. Docker motoru çoklu-platform bir uygulamadır. Bu uygulama Windows, Linux ve Mac OS işletim sistemlerinde çalışabilmektedir.

Docker motoru içeriğinde Docker yöneticisi ve Temsili Durum Aktarımı'nı bulundurmaktadır. Docker yöneticisi tüm konteyner süreçlerini yönetirken, Temsili durum aktarımı uygulaması ise konteynerlerin dış dünya ile bağlantısını Şekil 3.6'da görüldüğü gibi yönetmektedir.



Şekil 3. 6. Temsili durum aktarımı uygulaması tarafından yönetilen süreçler

Temsili durum aktarımı, Docker yöneticisi ile Docker istemci arasındaki süreçleri yönetmektedir. Konteyner verilerinin veri yığını (volume) üzerinde tutulması, imaj yönetimi, ağ bağlantısı yönetimi ve konteynerler arası iletişimden sorumlu servistir. Docker motorunu yönetmek için hazırlanan tüm yazılımlar Temsili Durum Aktarımı servisi aracılığıyla iletişime geçmektedir.

3.4. Docker Kurulum Modelleri

Docker konteyner yönetim yazılımının iki sürümü bulunmaktadır. Docker Topluluk Sürümü (Docker CE) ve Docker Şirket Sürümü (Docker EE) dır. Docker Topluluk Sürümü, ücretsiz bir sürümdür. Docker konteyner kullanımına yeni başlayanlar, konteyner tabanlı uygulama geliştirmeyi denemek isteyenler ve küçük işletmeler için yeterlidir. Bu sürüm açık kaynak kodlu bir platformdur. Docker Şirket Sürümü ise ücretli sürümdür.

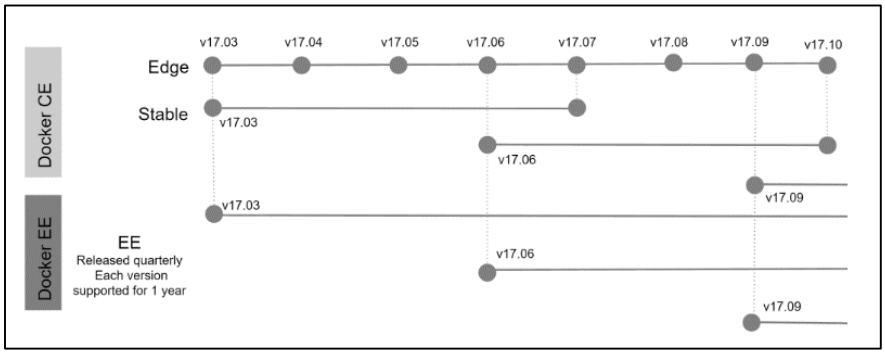
3.4.1. Docker Topluluk Sürümü (Docker CE)

Docker topluluk sürümü, ücretsiz bir sürüm olup, topluluk tarafından desteklenmektedir. Piyasaya kenar (Edge) ve kararlı (Stable) sürümleri olarak sunulmaktadır. Kenar sürümü her ay güncellenirken, kararlı sürümü üç ayda bir güncellenmektedir. Kararlı sürümü, güvenlik güncelleştirmelerini ve hata düzeltmelerini içeren kararlı çalışan sürümlerdir. Docker topluluk sürümü güncelleme döngüsü ile kullanıcılar kaynak kodu düzeyinde çok fazla farklılık olmadan benzer yazılımları kullanmaktadır. Kenar sürümünü kullanan kullanıcılar aylık güncelleştirme alırken, kararlı sürümünü kullananlar üç ay sonra güncelleştirme alabilmektedir [56].

3.4.2. Docker Şirket Sürümü (Docker EE)

Kritik iş uygulamalarını geliştirenler, şirketler ve bilgi işlem bölümleri için uygun sürümür. Bu sürümde güvenli kayıt defteri, ileri düzey kontrol paneli ve ilave araçlar bulunmaktadır. Bu sürümde kurumsal destek sağlanmaktadır. Docker şirket sürümü, üç farklı sürümle kullanıcılara ulaşmaktadır. Bu sürümler, temel (basic), standart (standard) ve gelişmiş (advanced)'tir. Docker şirket sürümü üç ayda bir güncellenmektedir. Ubuntu, Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Windows Server 2016, Oracle Linux ile bütünleşebilen ve bu işletim sistemi firmalarınca desteklenen sürümdür.

Docker şirket sürümü, Docker destekli sertifikalı altyapı sunar. Docker platformu ve Docker mağazasından sertifikalı konteyner ve eklentileri indirilebilir. Docker şirket standart sürümü, Microsoft aktif dizin (AD/BDEP) kullanıcı bütünleşmesinin yanında, gelişmiş konteyner ve imaj yönetimi, rol tabanlı erişim denetimi (docker datacenter) içermektedir. Docker şirket gelişmiş sürümü, Şekil 3.7'de görüldüğü gibi standart sürüm özelliklerinin yanında güvenlik taramaları ve güvenlik açıklarının sürekli izlenmesine yönelik bileşenler içermektedir [57].



Şekil 3. 7. Docker Sürümleri

Docker şirket sürümü ve Docker topluluk sürümü arasındaki sürüm farklılıkları ve sürüm aralıkları Şekil 3.7'de görülmektedir. 2015 yılı sonrasında piyasaya sürülen sürümler standarda bağlanmış ve sürüm numarasının ilk bölümü sürüm yılını, ikinci bölüm sürüm ayını ve son bölüm yama numarasını ifade etmektedir.

3.4.3. İşletim Sistemine Göre Docker Kurulumu

Konteynerler çalışabilmek için Linux çekirdeğine (kernel) ihtiyaç duymaktadır. Bu sebeple Docker, Şekil 3.8'de görüldüğü gibi farklı işletim sistemleri için farklı kurulumlar önermektedir. Varsayılan kurulumlarda, üzerinde Linux çekirdeği barındırmayan Windows işletim sistemi için geliştirilen ve desteklenen Linux için Windows alt sistemi (LWAS) özelliği sayesinde Docker bu platformda çalışabilmektedir. Windows için Docker (Docker for Windows) yazılımı üzerinde Windows uyumlu konteynerlerin yanında Linux konteynerleri de çalıştırılabilmektedir. Doğal olarak Linux çekirdeğine sahip CentOS, Ubuntu, Debian gibi Linux işletim sistemlerinde çalışabilmek için çekirdek (kernel) sürümünün en az 3.1 seviyesinde olması gerekmektedir [58].

Platform	x86_64 / amd64	arm64 (Apple Silicon)
Docker Desktop for Mac (macOS)	✓	✓
Docker Desktop for Windows	✓	

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	s390x
CentOS	✓	✓		
Debian	✓	✓	✓	
Fedora	✓	✓		
Raspbian			✓	
RHEL				✓
SLES				✓
Ubuntu	✓	✓	✓	✓
Binaries	✓	✓	✓	

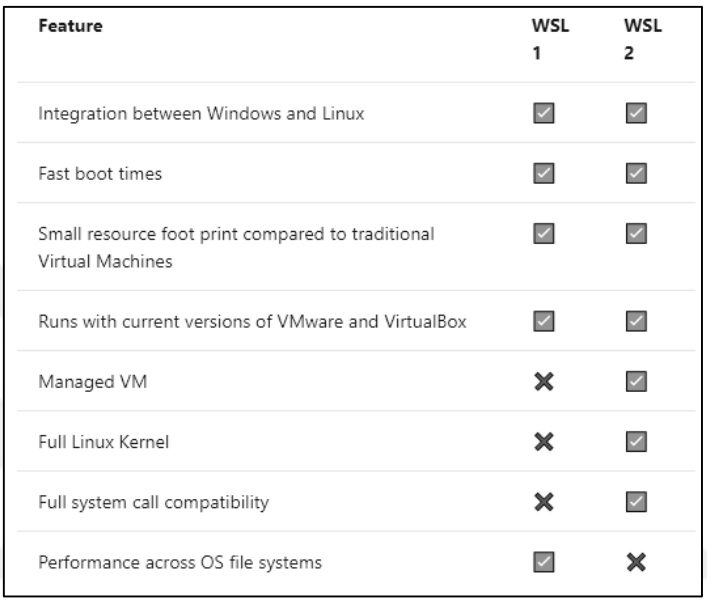
Şekil 3. 8. İşletim sistemine göre Docker sürümleri

3.4.3.1. Windows İşletim Sistemine Kurulum

Docker uygulaması sürümü en az Windows 10 64 bit Home/Pro 2004 (derleme 19041) veya üzeri, şirket veya eğitim sürümü (derleme 18363) veya üzeri olan bilgisayarlara yüklenebilmektedir. Eğer sunucu sürümü yüklenecek ise işletim sisteminin Windows Server 2016 ve üzeri olması gereklidir. Yükleme işlemi için docker dağıtım (hub) internet sayfası üzerinden kurulum dosyasının (Docker Desktop Installer.exe) indirilerek kayıt edilmesi gereklidir. İndirilen dosya üzerine çift tıklanarak kurulum işlemi başlatılır.

Yükleme işlemleri öncesinde donanım üzerindeki Temel Giriş ve Çıkış Sistemi (BIOS) ayarlarında sanallaştırma özelliklerinin aktif edilmesi, sonrasında ise Windows Özellikleri Aç/Kapat bölümünden Kapsayıcılar, Sanal Makine Platformu, Linux için Windows Alt Sistemi (LWAS) özellikleri etkinleştirilmelidir. Linux için Windows Alt Sistemi (LWAS) yükleme işlemi sonrasında Linux için Windows Alt Sistemi ikinci

(LWAS2) sürümüne güncellenmesi dosya sisteminin performansını artırmak ve tam Linux uyumluluğu açısından önemlidir [59]. Linux için Windows Alt Sistemi (LWAS1) ve Linux için Windows Alt Sistemi ikinci sürümü (LWAS2) hakkında detaylı karşılaştırma Şekil 3.9'da görülmektedir. Linux için Windows Alt Sistemi ikinci sürümünün (LWAS2) başarıyla çalıştırılabilmesi için İkinci Seviye Adres Tercümesi (İSAT) ile 64 bit işlemci, 4 GB sistem belleği gerekli olmaktadır.

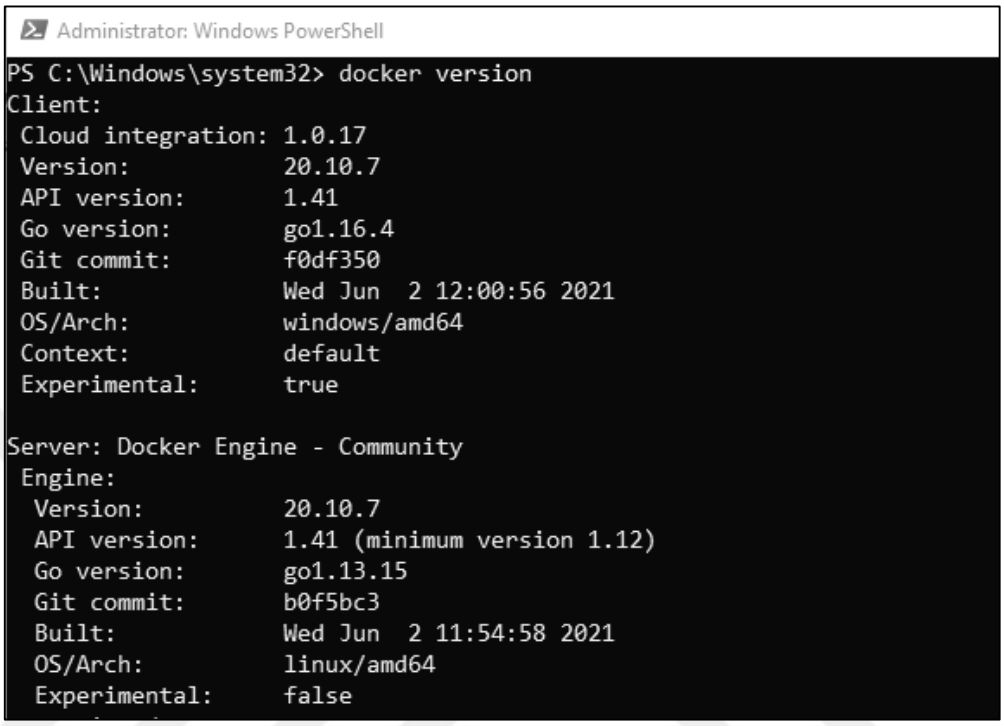


Feature	WSL 1	WSL 2
Integration between Windows and Linux	✓	✓
Fast boot times	✓	✓
Small resource foot print compared to traditional Virtual Machines	✓	✓
Runs with current versions of VMware and VirtualBox	✓	✓
Managed VM	✗	✓
Full Linux Kernel	✗	✓
Full system call compatibility	✗	✓
Performance across OS file systems	✓	✗

Şekil 3. 9. Linux için Windows Alt Sistemlerinin (LWAS1-LWAS2) karşılaştırılması

Kurulum işlemi sonrasında, bilgisayarda bulunan yönetici hesabı kullanıcı hesabından farklı ise Docker kullanıcı (docker-users) grubuna eklenmelidir. Kurulum sonrasında Docker yeniden başlatılacak ve durum çubuğunda balina simgesi aktif olacaktır [60].

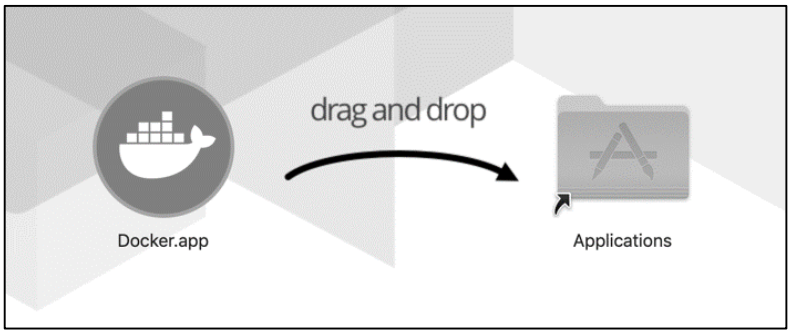
Docker kurulumu tamamlandıktan sonra Windows güçlü kabuk (Powershell) komut ekranı yönetici haklarıyla çalıştırılarak sürüm bilgisi kontrolü Şekil 3.10'da görülen biçimde yapılır. Bu komut Docker uygulamasının başarıyla yüklenip çalıştığını test etmemizi sağlayacaktır.



Şekil 3. 10. Docker kurulum sonrası kontrol ekranı

3.4.3.2. MAC İşletim Sistemine Kurulum

Windows sistemlerindeki kurulum adımlarına benzer olarak docker dağıtım adresi üzerinden MAC işletim sistemi masa üstü için Docker (Docker Desktop for MAC) seçilir. Kullanılan MAC bilgisayarının işlemci modeline göre (Intel veya Apple) dosya indirme işlemi gerçekleştirilir. Intel işlemcili bilgisayarlarda en az işletim sistemi sürümü MacOS 10.14 veya daha yeni bir sürüm (Mojave, Catalina, Big Sur) olmalıdır. Apple işlemcili bilgisayarlarda ise bazı kurulum dosyaları hala önceki sürüm (Darwin/AMD64) olduğundan öncelikle Rosetta2 yazılımı kurulmalıdır [61]. Kurulum işlemlerinin başarılı olabilmesi için 4 GB belleğe ihtiyaç duyulmaktadır. Ayrıca sanallaştırma için 4.3.30 sürümü sonrası sanallaştırma yöneticisi (Oracle VirtualBox) yazılımı gereklidir. Tüm gereksinimler tamamlandıktan sonra indirilen MAC işletim sistemi masa üstü için Docker (Docker Desktop for MAC - Docker.dmg) yazılımı çalıştırılır.

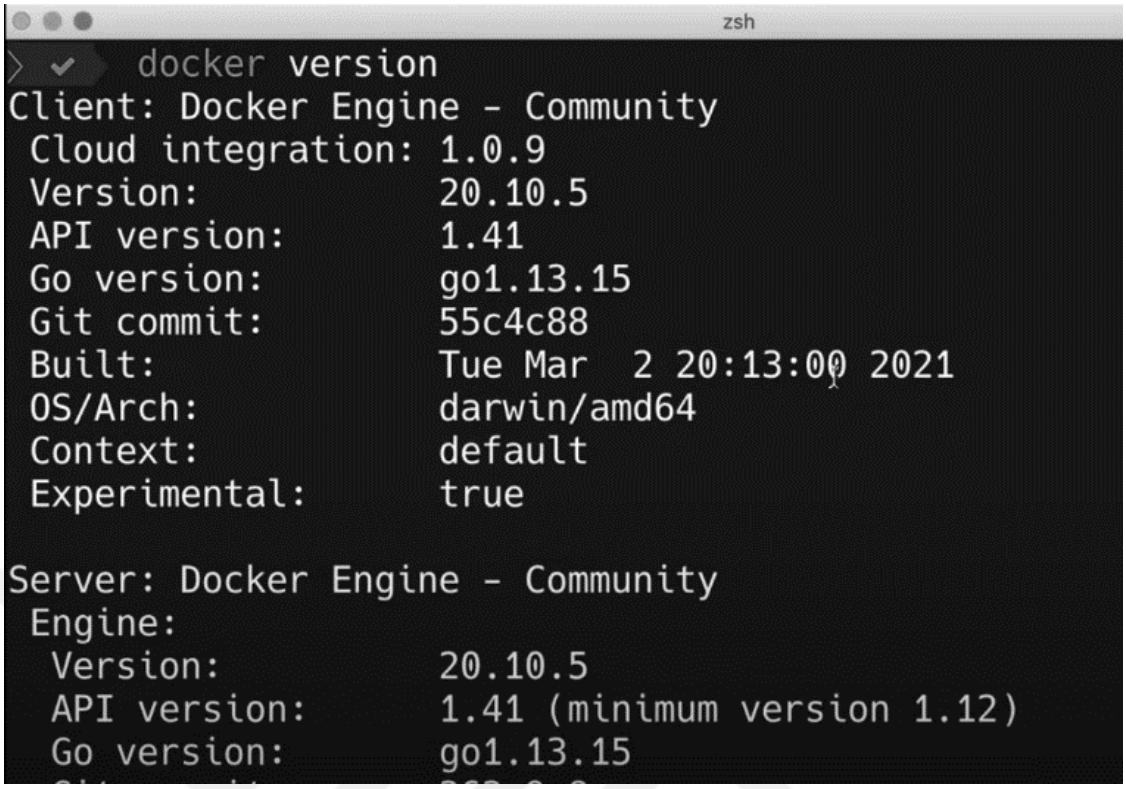


Şekil 3. 11. Mac Os işletim sistemine uygulamanın (Docker.app) yüklenmesi

Docker uygulama simgesi (Docker.app), uygulamalar klasörüne Şekil 3.11'de görülen şekilde sürüklenir. Docker'ı başlatmak için uygulamalar klasörü açılır. Şekil 3.12'de uygulamalar klasörü ızgara modunda görüntülenmektedir.

Şekil 3. 12. MacOs işletim sistemine Docker uygulamasının kurulması

Üst durum çubuğunda Docker'ı simgeleyen balina sembolü Docker masa üstünün (Docker Desktop) çalıştığını ve terminalden erişilebilir olduğunu Şekil 3.13'te göstermektedir.



Şekil 3. 13. MacOs işletim sistemi kurulum sonrası kontrol ekranı

3.4.3.3. Linux İşletim Sistemine (Ubuntu) Kurulum

Ubuntu Linux işletim sistemine Docker kurulumu için 64 bitlik Ubuntu Hirsute 21.04, Ubuntu Groovy 20.10, Ubuntu Focal 20.04 (UVD), Ubuntu Bionic 18.04 (UVD) sürümlerinden herhangi biri yüklenmiş olmalıdır. Docker Engine x86_64 (veya Amd64) Armhf, Arm64, ve s390x mimarilerinde desteklenmektedir. Kurulum işlemi birkaç farklı yöntemle yapılabilmektedir. Kullanıcıların çoğunluğu kurulum ve yükseltme kolaylığı sebebiyle Docker depolarını kullanmaktadır. Bu yöntem Docker tarafından da tavsiye edilmektedir. Test ve geliştirme ortamlarında ise otomatikleştirilmiş komut dosyalarını kullanmak hızlı bir kurulum sağlamaktadır [62]. Hızlı ve kolay kurulum için Ubuntu terminali açılarak sadece iki komutla Şekil 3.14'te görüldüğü gibi kurulum gerçekleştirilebilir.

Şekil 3. 14. Ubuntu işletim sisteminde hızlı Docker kurulumu

Linux işletim sistemine kurulum otomatik olarak gerçekleşecektir. Kurulum sonrasında test amacıyla sürüm kontrolü Şekil 3.15'de görüldüğü gibi yapılabilir.

Şekil 3. 15. Ubuntu işletim sistemi kurulum sonrası kontroller

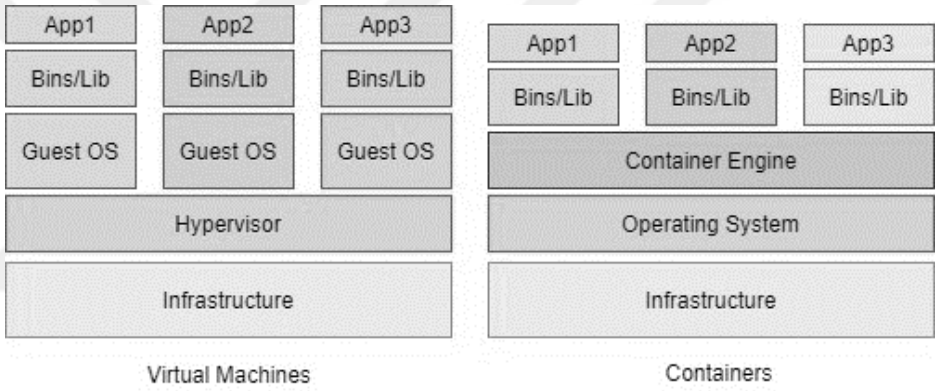
3.5. Konteyner Teknolojisi

Temelleri 1979 yılına kadar uzanan ve Unix işletim sistemi çekirdeğinde yazılan bir özelliğin günümüze kadar ulaşan süreçleridir. 2000'li yılların başında FreeBSD Jail ile düzenli hale gelmiş, VServer projesi ile Linux işletim sistemine girmiştir. Bu süreçte FreeBSD Jails, Linux VServer, Oracle Solaris Containers, Open VZ, Process Containers (Google), LK, Warden ve Lmctfy gibi gelişim seviyelerinden geçerek günümüze ulaşmıştır. 2013 yılında yapılan toplantıda bir konuşmacının sanallaştırma sistemlerinde yaşanan problemlere çözüm olabileceğini önermesiyle tekrar önem kazanmıştır [63].

3.6. Konteyner Mimarisi

İşletim sistemi seviyesinde sanallaştırma sağlayan bir yazılım olan Docker, 2013 yılında yayınlanmıştır. Docker konteyner adı verilen yazılım paketlerini çalıştırır. Konteyner bir uygulamanın çalıştırılabilmesi için gerekli olan tüm dosyaları, kütüphaneleri, eklentileri ve yapılandırma dosyalarını içerir. Çalışabilmesi için gerekli her şey bir arada olduğu için işletim sistemi sürüm farklılıkları gibi yazılımın çalışmasına engel olabilecek parametreler ortadan kaldırılmış olur.

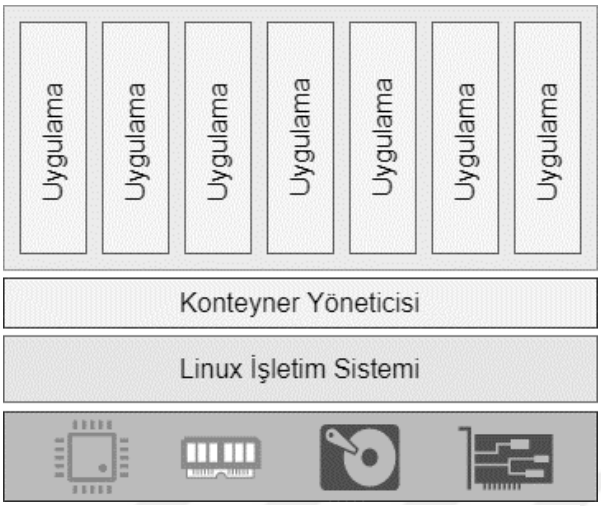
Ayrıca sanal makinelerin aksine üzerinde gömülü işletim sistemi barındırmadıkları ve üzerinde bulundukları sunucunun işletim sistemi çekirdeğini kullandıkları için küçük boyutlara sahiptir. Böylece bir konteyner birkaç megabayt büyüklüğünde iken aynı özelliklere sahip bir sanal makine birkaç gigabayt büyüklüğünde olabilir.



Şekil 3. 16. Sanallaştırma mimarisi ve konteyner mimarisi

Konteyner mimarisi, Şekil 3.16'da görüldüğü gibi çalıştırılan uygulamaları birbirinden izole etmek için sanallaştırma teknolojisi kullanmaz. Linux işletim sistemi çekirdeğinde bulunan "Namespaces" ve "Control Groups" bileşenlerini kullanır. İşletim sistemi çekirdeğinde bulunan bu bileşenler sayesinde Konteyner teknolojisi ortaya çıkmıştır. Bu iki bileşen sayesinde mevcut işletim sistemi üzerinde farklı yazılımlar birbirlerine izole biçimde çalışabilmektedir. Docker gibi mevcut bulunan

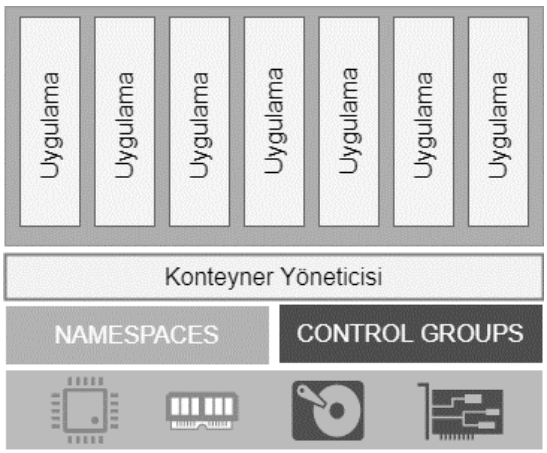
konteyner yöneticileri Şekil 3.17'de görüldüğü gibi bu teknolojiyi kullanarak konteyner haline getirilen yazılımları birbirlerinden izole çalıştırabilmektedir.



Şekil 3. 17. Konteyner yöneticisi ve uygulamalar

"Namespaces" teknolojisi, konteynerlerin sadece izin verilen alanları görmesini sağlamaktadır. Örneğin yeni oluşturulan ve çalıştırılan bir konteyner içerisine girilerek çalışan servislere bakıldığında sadece konteyner içinde çalışan servisler görülebilecektir.

Üzerinde çalıştığı işletim sistemi servisleri görüntülenemeyecektir. Ayrıca konteyner içindeki disk alanları kontrol edildiğinde, sadece konteyner için atanmış disk alanı görüntülenecektir. Konteynerler mevcut sistem kaynaklarına ulaşamayacağı gibi başka konteynerlere ait sistem kaynaklarını Şekil 3.18'de olduğu gibi göremeyecektir.



Şekil 3. 18. Konteyner uygulama izolasyonu

Günümüzde kullanılan Linux çekirdeklerinde aşağıdaki isim uzayları (namespace) bulunmaktadır;

- **Pid** : Çalışan prosesleri birbirinden izole hale getirir.
- **Network** : Network interface ve routing tablolarını birbirinden izole eder.
- **Mount** : Dosya sistemindeki mount pointleri birbirinden izole eder.
- **Ipc** : İşlemleri farklı Inter-process communication ile izole eder.
- **Uts** : Host ve Domain name olarak işlemleri birbirinden izole eder.
- **User** : Kullanıcı ID leri işlemler arasında birbirinden izole eder.

Örneğin, docker ile bir nginx web sunucusu konteynerini çalıştıralım. Çalışan konteynerde aktif olan süreçler (process) listelendiğinde konteynerin sadece kendine ait işlemlerin listelendiği Şekil 3.19'da görülmektedir.

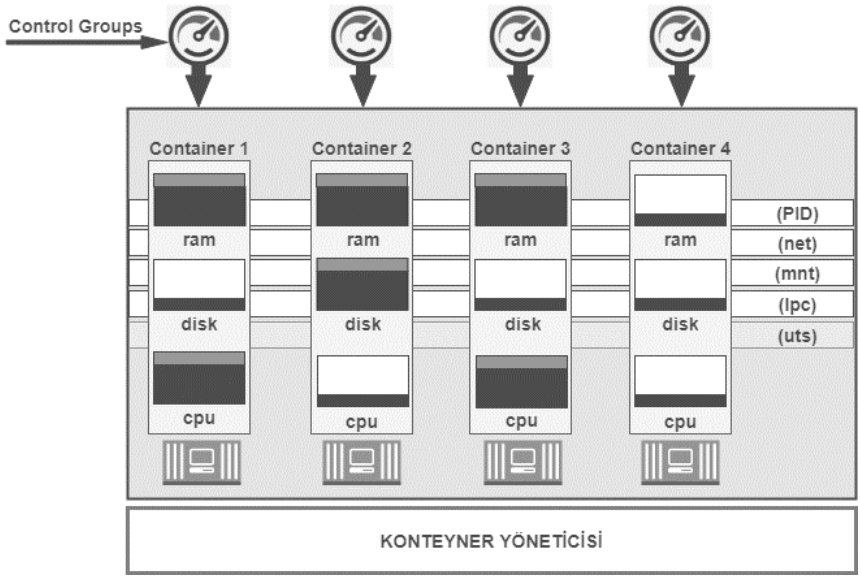
```
root@ozcan:~# ps -f --forest -C nginx
UID          PID    PPID  C STIME TTY          TIME CMD
root        4740    4720  0 22:44 ?        00:00:00 nginx: master process ngin
systemd+    4802    4740  0 22:44 ?        00:00:00  \_ nginx: worker process
systemd+    4803    4740  0 22:44 ?        00:00:00  \_ nginx: worker process
root@ozcan:~#
```

Şekil 3. 19. Örnek konteynerde çalışan işlemler

Çalışan süreçler incelendiğinde uygulamayı çalıştıran sürecin kimliği (Process ID) değerinin diğerlerinden daha küçük olduğu görülecektir. Bu durum bu sürecin ebeveyn (Parent) süreç olduğunu göstermektedir.

Konteynerlerde çalışan süreçler ağaç yapısına sahiptir. Bu sayede aynı konteyner içerisinde çalışan farklı süreçler bile birbirlerinden izole durumdadır. "Control Groups" Teknolojisi, oluşturulan konteynerlerin kullanacağı donanım kaynaklarını sınırlar. Klasik bilgisayar uygulamaları boşta bulunan sistem kaynaklarını (işlemci, bellek, disk, ağ) istedikleri gibi kullanabilmektedir. Bu ise sistemde yeni bir uygulamanın başlatılmasıyla sorunlar oluşturabilmektedir.

Linux çekirdeğinde bulunan “Control Groups” teknolojisiyle konteynerlerin kullanacağı sistem kaynakları Şekil 3.20’de görüldüğü gibi sınırlandırılabilmektedir. Bu teknoloji ile sistem kaynaklarının tek bir konteyner tarafından kullanılmasının önüne geçilerek tüm konteynerlere ihtiyacı olan sistem kaynakları tahsis



Şekil 3. 20. Konteynerlerde kontrol grupları

edilebilmektedir.

Linux çekirdeğinde bulunan “Control Groups”, “Namespaces” teknolojileri yanında geliştirilen mikro yapıdaki uygulamalar (images), günümüzde yaygın olarak kullanılan Docker konteynerlerini ortaya çıkarmıştır.

3.7. Docker ve Konteyner İlişkisi

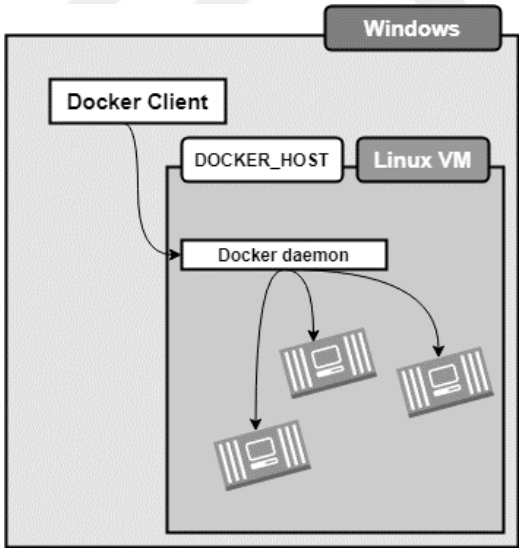
Docker, mevcut bulunan konteyner teknolojisi kullanarak uygulama oluşturma, dağıtma ve çalıştırma işlemlerini kolaylaştıran bir araçtır. Konteyner, geliştiricilerin ortaya çıkardığı uygulamayı, kütüphaneler ve diğer bağımlılıklarıyla birlikte ihtiyaç duyduğu tüm bileşenlerle birleştirerek, tek bir paket olarak oluşturulmasını sağlar. Böylece uygulama içerisinde kod yazımı ve test için kullanılan makineden farklı ortamda (başka bir Linux makinesinde) konteynerin çalışmasını sağlar [64]. Docker, fiziksel bilgisayar üzerinde çalışan bir sanal makine gibidir, ancak sanal bir makineden farklı olarak, sanal işletim sistemi kurulumu yerine, uygulamaların üzerinde çalıştığı işletim sisteminin (Linux) çekirdeğini kullanmaktadır. Bu durum önemli bir performans artışının yanında uygulamanın boyutunu azaltılmasına olanak verir.

3.8. Docker Bileşenleri

Docker platformu günümüzde geliştirilen kodların hızlı şekilde sunucuya gönderilerek test edilmesini sağlayan en önemli araçlardan biri haline gelmiştir. Bu platformu oluşturan en önemli bileşenler bu bölümde açıklanacaktır.

3.8.1. Docker Engine

"Docker Engine", geliştirilen uygulamaların fiziksel ortamdan bağımsız olarak izole biçimde çalışmasını sağlayacak imajların oluşturulmasını, oluşturulan imajlardan konteynerlerin meydana getirilerek bunların çalıştırılmasını sağlayan bileşendir.
Konteynerler önceden hazırlanan uygulama (imaj) şablonlarından türetilen çalışır kopyalardır. Docker Engine, istemci ve sunucu yeteneklerine sahip bir bileşendir. Yerel bir makinede hem istemci hemde sunucu rolü üstlenebilir. Uzak bir makinede sadece sunucu hizmeti verebilir. Docker yöneticisi (Daemon), Şekil 3.21'de görüldüğü gibi ifade edilebilmektedir. Docker platformunda sanallaştırmayı sağlayan sanallaştırma yöneticisi (hipervizör) olarak çalışmaktadır.



Şekil 3. 21. Docker yöneticisi

Linux çekirdeğindeki LK gibi çalışarak, konteynerlerin birbirlerinden izole çalışmasını, yaşam döngüsü takibini, dosya sistemini, işlemci ve bellek kısıtlamalarını kontrol etmektedir [65].

3.8.2. Docker Client

Fiziksel ortamdaki kullanıcı ile Docker yöneticisi (Daemon) arasında iletişimi sağlayan komut ara yüzüdür. Gerekli durumlarda Docker kaydından (Registry) yeni imajların indirilmesi, mevcut imajlardan konteynerlerin ayağa kaldırılması, konteynerlerin başlatılması, durdurulması, yeniden başlatılması, silinmesi, konteynerlere işlemci ve ram atamaları gibi yapılabilecek tüm işlemlerin gerçekleştirildiği komut ortamıdır.

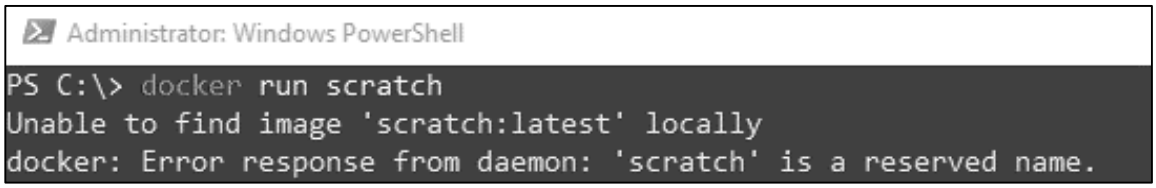
3.8.3. Docker Image

"Docker Image" kavramı, bir yazılımın çalışabilmesi için gerekli tüm kodları, kütüphaneleri, bağımlılıkları, ayar dosyalarını ve hesapları içerisinde barındıran bir pakettir. Docker Image paketleri Docker dağıtım sitesi (hub) üzerinden indirilmektedir. Ayrıca bazı bulut bilişim platformlarında farklı imaj depoları (repository) da bulunmaktadır. Google Cloud, Amazon ECR, Microsoft Azure, Huawei Cloud, IBM Cloud v.b. firmaların bulut üzerinde konteyner servisleri bulunmaktadır. Bu servislerden firmalara özgü resmi yazılım imajlarına erişilebilmektedir. Yapılan çalışmalarda kullanıcılara sunulan imajlar olduğu gibi kullanılmayıp, gerekli durumlarda indirilen imajlara programlar, kütüphaneler, dosyalar eklenebilmektedir. Böylece indirilen ham (orijinal) imajın üzerine yazılım projesini dâhil ederek yeni bir imajın oluşturulması ve dağıtılması mümkün olmaktadır.

Docker imajı katmanlardan oluşmaktadır. Bu katmanlar başka bir imajdan devralınabildiği gibi başka bir oluşturma işlemindeki talimatlarla imajla birlikte oluşturulabilir. Bir imajın kökeni, oluşturulmuş imaja temel olan imajı göstermektedir. Tüm Docker imajlarının temeli baz (scratch) imajdır. Aslında boş olan bu imajda dosya ve klasör bulunmamaktadır. Başka baz imaj görüntüleri oluşturmak için bu imaj Şekil 3.22'de görüldüğü gibi kullanılır.

Yeni bir konteyner imajı oluşturmak için Dockerfile dosyası kullanılır. Bu dosya içindeki ilk satır "FROM" parametresi başlar. Docker ile ilgili Temel ve Ebeveyn (Base, Parent) imaj kavramlarıyla sıkça karşılaşılmaktadır. Temel imajlar, bir Dockerfile içerisinde referans gösterilebilmesi dışında docker komut satırından

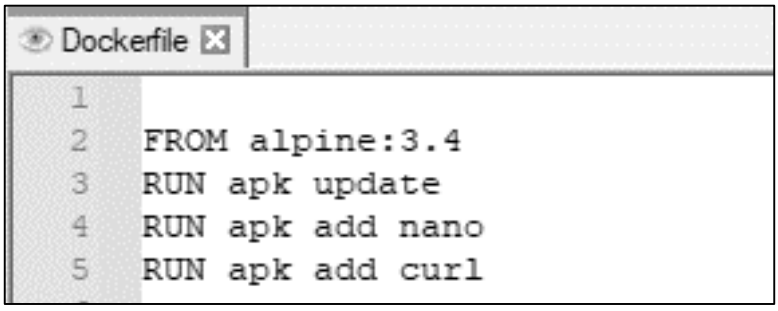
doğrudan indirilerek çalıştırılmayan imajlardır. Ebeveyn imajlar ise Dockerfile üzerinde hem referans gösterilebilmekte hem de docker komut satırından çalıştırılabilmektedir [66].



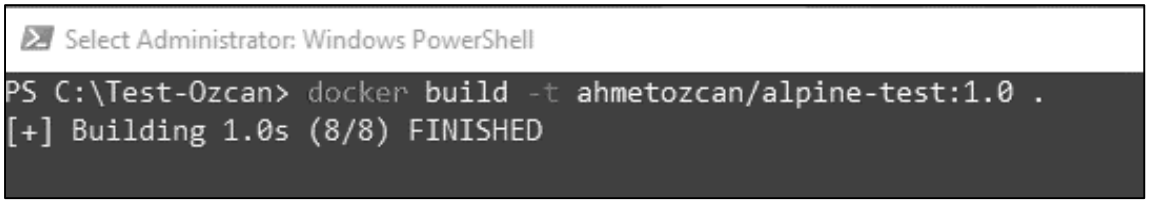
Şekil 3. 22. Scratch base imajı

3.8.4. Dockerfile

Dockerfile, Docker imajlarını oluşturmak için faydalanılan bir dosyadır. Bu dosya içinde yazılan her satırın başında Şekil 3.23'de görüldüğü gibi özel kelimeler bulunmaktadır. Bu kelimeler direktif (instruction) olarak ifade edilir. Docker imajları üzerinde her katman direktif olarak belirtilen katmanlardan oluşur. Oluşturulacak Dockerfile dosyası içerisine yazılacak direktif sayısı oluşacak imajın katman sayısını belirlemiş olur. Yazılan her direktif kendisinden önce yazılan direktifin üzerinde çalışacaktır.

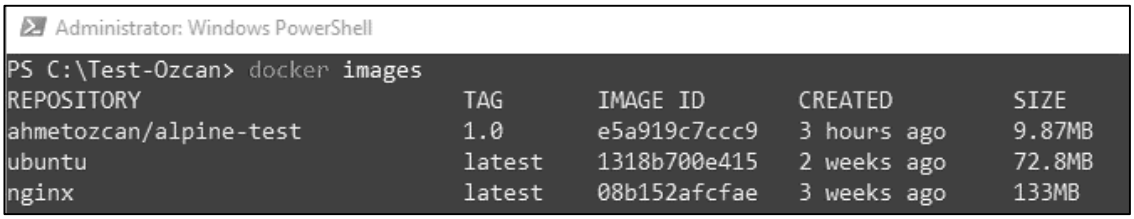


Şekil 3. 23. Örnek Dockerfile dosyası



Şekil 3. 24. Docker imajının yeniden derlenmesi

Hazırlanan Dockerfile dosyasının uzantısı yoktur. Dockerfile dosyası “build” komutu ile Şekil 3.24’te görüldüğü gibi çalıştırılarak imaj oluşturulur. Oluşturulan imaj dosyası Şekil 3.25’te görüldüğü gibi “docker images” komutu ile görüntülenebilir.

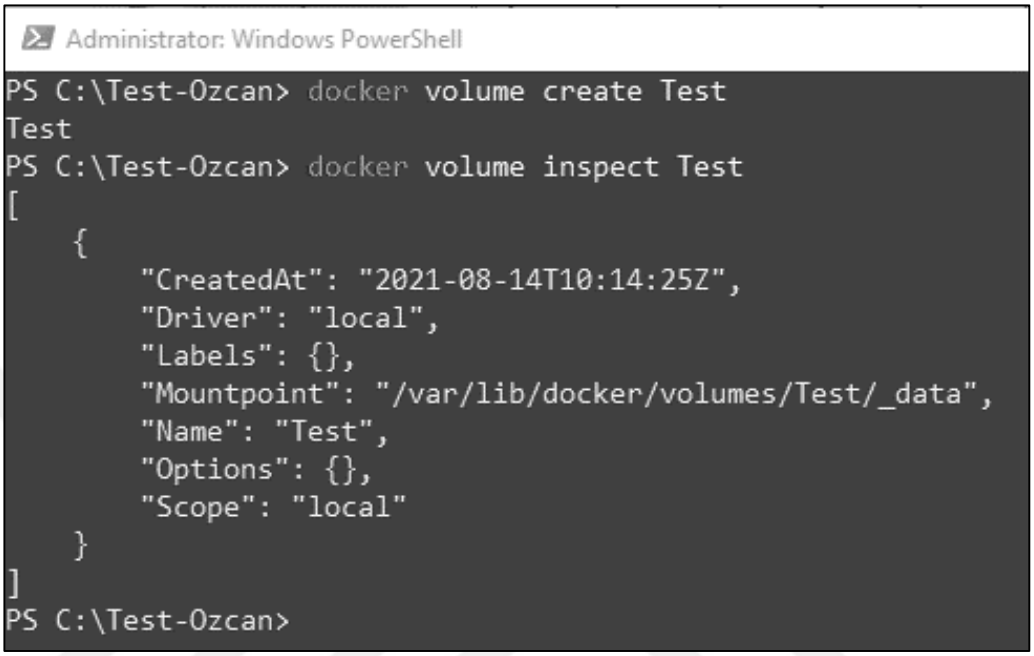


Şekil 3. 25. Docker imajlarının listelenmesi

Docker imajları oluştururken katmanlar kullanılır. Bu işlem öncesinde oluşturulacak imajlarda hangi konteynerlerin hangi sırada yükleneceği, hangi klasörlerde tutulacağı, hangi parametrelere sahip olacağının önceden belirlenmesi gereklidir. Tüm katmanların yükleme sıralamasını dikkate edilerek Dockerfile oluşturulmalıdır. Dockerfile oluşturulurken kullanılabilecek direktifler mümkün olduğunca minimum sayıda tutulmalıdır. Bu bölümde Dockerfile oluşturmak için gerekli temel direktifler “instructions” açıklanacaktır.

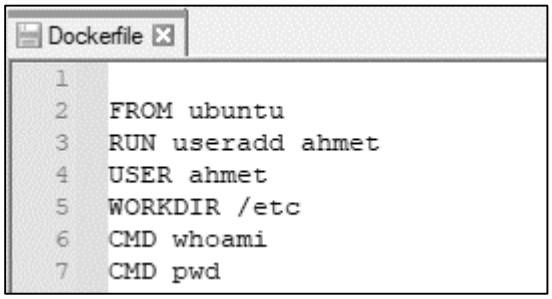
- **FROM** : Her Dockerfile bu direktifle başlar. Oluşturulacak konteyner imajının hangi imajı baz alacağını bu direktif belirler. Dockerfile içerisinde birden fazla “FROM” direktifi bulunabilir.
- **RUN** : Bu direktif ile önceden “FROM” direktifi ile referans edilen imaj üzerinde belirli komutlar çalıştırılabilir.
- **CMD** : Bu direktif önceden oluşturulan katman üzerinde komut çalıştırmak için kullanılır.
- **COPY** ve **ADD**: Bu iki direktif konteyner içerisine dosya kopyalamak için kullanılır. ADD direktifi Docker’a ilk eklenen direktiftir. Sonrasında işlemleri kolaylaştırmak için COPY direktifi eklenmiştir. Her iki direktif temelde aynı işleve sahip olmasına rağmen COPY direktifi diğerine göre daha fazla özelliğe sahiptir.
- **ENV** : Ortam değişkenlerini yapılandırmak için kullanılır. Ortam değişkeni olarak konteynerin çalışacağı port numaraları, işlemci sayısı, bellek büyüklüğü, konteynerin çalışacağı yol gibi parametreler kullanılmaktadır.
- **VOLUME** : Kapalı bir kap içerisinde çalışan konteynerlerin oluşturacağı çıktıların Şekil 3.26’te görüldüğü gibi kalıcı olarak saklanacağı disk ortamlarının

belirtilmesinde kullanılmaktadır. Bu direktif ile oluşturulan alanda; veri dosyaları, veri tabanları, uygulama tarafından sürekli ihtiyaç duyulan dosyalar ev sahibi dosya sisteminde saklanabilmektedir. Böylece konteyner işini bitirip yok edildiğinde veriler hala ulaşılabilir olacaktır.



Şekil 3. 26. Docker volume kullanımı

- **WORKDIR** : Konteynerin oluşturulması esnasında çalışacağı dizini belirtir. Kendisinden sonra gelen RUN, CMD, ENTRYPOINT, ADD ve COPY direktifleri bu klasör üzerinde çalışacaktır.
- **USER** : Çalışan konteynerin Namespace içerisinde geçerli olan kullanıcı ID (KK) ve grup ID (GKK) değerlerini Şekil 3.27'de görüldüğü gibi değiştirmeyi sağlar.

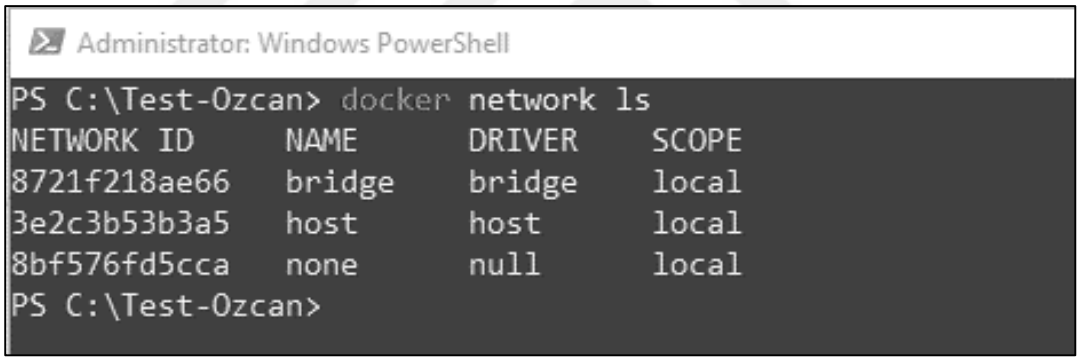


Şekil 3. 27. Dockerfile içinde kullanıcı tanımlama

- **SHELL**: CMD direktifini kullanırken komut sonuna bu komutun hangi kabukta çalışacağının belirtilmesi (/bin/bash -c) gerekir. Bu direktifi başlangıçta kullanmak suretiyle her CMD satırının sonuna kabuk ifadesini yazmaya ihtiyaç kalmaz..
- **ENTRYPOINT**: Bu direktif konteyner başlatıldığında neyin çalışacağının belirlenmesi için kullanılır.
- **ARG**: Docker imajının oluşturulması esnasında (docker build) parametre olarak verilmesi gereken değerleri belirtir.

3.8.5. Docker Network

Docker kapsayıcılarının birbirine bağlanabilmesi ya da docker olmayan uygulamalara bağlanabilmesi mümkündür. Docker ana bilgisayarında çalışan işletim sisteminin Windows veya Linux olması bağlantıyı engellemez. Şekil 3.28'de görüldüğü gibi Docker Network sisteminde birkaç ağ modeli bulunmaktadır.



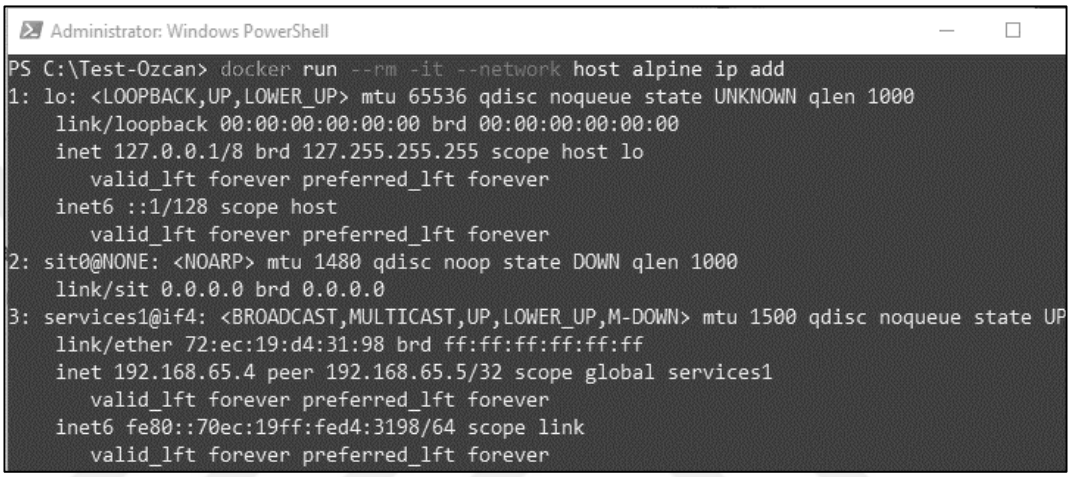
Şekil 3. 28. Docker network listeleme

- **Bridge**: Varsayılan ağ sürücüsüdür. Herhangi bir ağ belirtilmezse bu seçenek aktif olmaktadır. Bu ağ modeli sadece iletişim kurması istenen konteynerler arasında çalışmaktadır. Şekil 3.29'de görüldüğü gibi köprü ağına bağlanan konteynerler birbirleri arasında iletişim kurabilir. Farklı ağlardaki konteynerler bu ağdan izole durumdadır. Bu ağ modeli istenirse kullanıcı tanımlı olarak da oluşturulabilir [67].

Şekil 3. 29. Bir uygulamanın tanımlanan ağa bağlanması

Bir konteyneri kullanıcı tanımlı networke bağlamak için kullanılan yöntem Şekil 3.29'da görülmektedir.

- **Host**: Ana bilgisayar ve üzerinde çalışan konteynerlerin Şekil 3.30'da görüldüğü gibi birbirleriyle haberleşebildiği ağ modelidir. Bu ağ modeli sadece Linux işletim sistemi üzerinde çalışmaktadır. Windows ve Mac işletim sistemlerinde desteklenmemektedir.



Şekil 3. 30. Bir konteynerin Host ağına bağlanması

- **Overlay**: Eğer bir grup bilgisayar üzerinde Docker koşturulmak isteniyorsa (docker swarm), konteynerler oluşturulmadan önce bu ağ oluşturulmalıdır. Büyük işletmelerde birden fazla Docker ana bilgisayarı bulunabilir. Aynı anda birden fazla makine aynı konteyneri çalıştırıyor ve gelen isteklere cevap veriyor ise bu durumda Docker Swarm, Kubernetes, Open Shift gibi çözümler gerekecektir.

  Küme olarak konteynerlerin işletilebilmesi için bir "overlay" network gerekecektir. Bu işlem için ilk düğümde (node) yazılan komut sayesinde üretilen belirteç (token) ağa katılacak bilgisayarlarda Şekil 3.31'de görüldüğü gibi çalıştırılır. Böylece her yeni oluşturulan düğüm (node) ağa katılmış olur.

```
$docker swarm init --advertise-addr
<IP address of node 1>

$docker swarm join --token <TOKEN>
\
--advertise-addr <IP address of
node n> \
<IP address of node 1>:2377
```

Şekil 3. 31. Docker sürüsünün oluşturulması

- **Macvlan**: Bu ağ modeli, Şekil 3.32'de görüldüğü gibi konteynerin ağda fiziksel bir aygıt olarak görüntülenmesi için kullanılır. Bir sanal makine (SM) kurulumundan geçiş yapılırken veya konteynere bir fiziksel adres (MEKA) atanmak suretiyle Docker uygulaması ağ trafiğini MEKA adreslerine göre konteynerlere yönlendirebilir.

```
Administrator: Windows PowerShell
PS C:\Test-Ozcan> docker network create -d macvlan --subnet 172.16.86.0/24 --gateway
 172.16.86.1 -o parent=eth0 my-macvlan-net
7ddfc041672d7cbf9b792a68e3dfa9636e13abf62b7d1f03d726f5c76859cd09
PS C:\Test-Ozcan>
```

Şekil 3. 32. Docker ortamı için macvlan ağı oluşturma

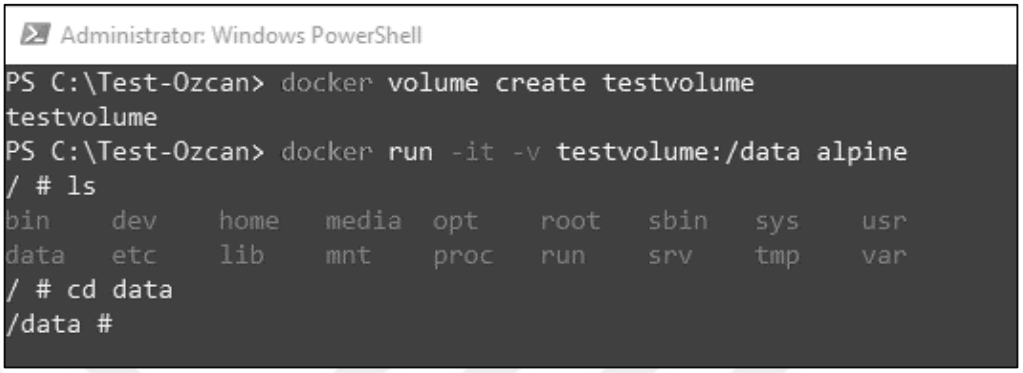
- **None**: Bu özellik tüm ağları Şekil 3.33'de görüldüğü gibi devre dışı bırakmaktadır. Özel bir ağ sürücüsü ile kullanılabilir. Sürü hizmetlerinde kullanılamaz.

```
Administrator: Windows PowerShell
PS C:\Test-Ozcan> docker run -it --rm --network none alpine
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
```

Şekil 3. 33. Çalışan konteynerde ağın devre dışı bırakılması

3.8.6. Docker Volume

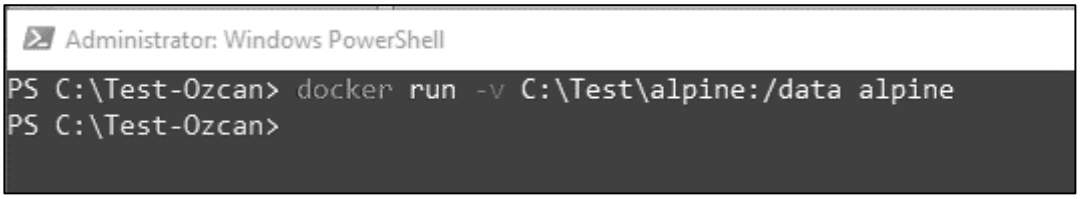
Konteyner teknolojisi günümüzde pek çok alanda kullanıldığı gibi yazılım geliştirme ve dağıtma süreçlerinde aktif olarak kullanılmaktadır. Günümüzde geliştirilen yazılımların tümüne yakın bölümü veri tabanı kullanmaktadır. Konteyner tabanlı



Şekil 3. 34. Disk üzerinde Docker volume oluşturma

yazılımların veri tabanı kullanması durumunda bilgilerin konteyner dışında tutulması gereklidir. Çünkü konteyner kapandığı veya çöktüğü zaman konteyner içine kaydedilen tüm bilgiler yok olacaktır. Bu amaçla verinin konteyner dışında tutulması ve konteyner her başlatıldığında dışardaki veri tabanına bağlanması mantıklı olacaktır. Bu amaçla ana bilgisayarda depolama birimleri oluşturularak konteynerleri bu birimlere birkaç yöntemle bağlamak mümkündür.

- **Volume Yöntemi** : Ana bilgisayar içinde belirlenen (/var/lib/docker/volumes) dizinde bir dizin oluşturularak istenilen konteynerin Şekil 3.34'de görüldüğü gibi bağlanmasıdır. Konteyner eğer bir Windows ana makinesinde çalışıyorsa bağlama işlemi Şekil 3.35'te yazılan komutlarla yapılabilir. Bu örnekte ana bilgisayarımızda bulunan Test dizini içerisinde oluşturulan "alpine" dizinine yazılan her dosya konteyner tarafından okunabilmektedir. Konteyner tarafından yazılan her dosya ise ana bilgisayar tarafından erişilebilmektedir.

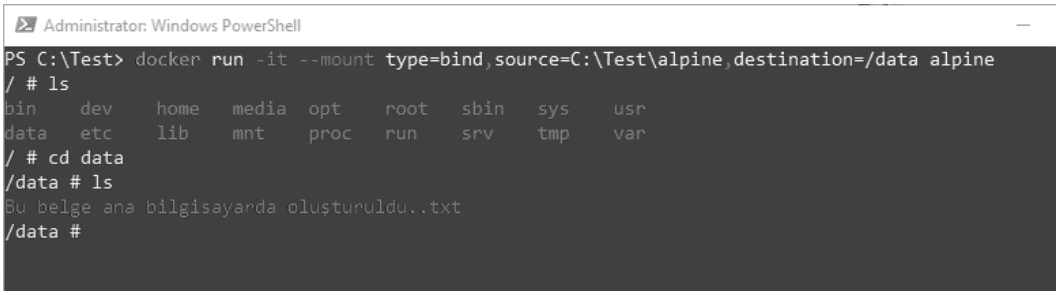


Şekil 3. 35. Disk üzerinde oluşturulmuş volumün aktif edilmesi

- **tmpfs Yöntemi**: Bazı durumlarda konteynerlere ait bilgilerin diske yazılması istenmeyebilir. Şekil 3.36'de görüldüğü gibi bellekte (RAM) tutulan bilginin buradan konteynere bağlanması yöntemidir. Özellikle hız gerektiren uygulamalarda ve bilginin diske yazılmasına ihtiyaç yoksa bu yöntem kullanılabilir.

Şekil 3. 36. Volüm yerine tmpfs yönteminin kullanılması

- **Bind Yöntemi**: Yazılım geliştirme sürecinde bilgisayarın bir dizinini konteynere bağlamak için kullanılan bir yöntemdir. Şekil 3.37'de görüldüğü gibi paylaşılacak klasör yolu tam olarak belirtilmelidir. Yazılım geliştiricilerin ürünlerini her seferinde temiz bir ortamda test edebilmeleri, bağımlılıkları ve uyumluluklarını görebilmeleri için uygun ortam sağlar.



Şekil 3. 37. Bind yöntemi ile bir dizinin konteynere bağlanması

3.8.7. Docker Registry

Docker imajlarını depolamak ve dağıtmak için kullanılan bir yöntemdir. Geliştiriciler Docker imajlarını oluşturmanın yanında oluşturulan çok sayıdaki imajın dağıtımının zor olduğunu keşfettiler. Docker kaydı (Registry) bu sorunu çözmek için geliştirildi. Docker Registry, ortamında depolanan imajların erişim kontrolünü sağlar, dağıtımını yönetir, güvenliğini geliştirir. Geliştiriciler kendilerine ait Docker kaydı (Registry)

kurabildikleri gibi bulut ortamında bulunan Oracle Container Registry (https://container-registry.oracle.com), Azure Container Registry (https://azure.microsoft.com/en-us/services/container-registry) gibi Docker hizmeti veren kuruluşları kullanabilir [68]. Ücretsiz erişim hizmeti veren Docker Hub bir kayıt defteridir. Bu alanda 100.000'den fazla resmi ve paylaşıma açık kullanıcı konteyner imajı bulunmaktadır.

3.8.8. Docker Repository

"Docker Repository" kavramı ile Docker kaydı (Registry) kavramı birbirine benzemekle birlikte, Docker kaydı imajları barındırmak ve dağıtmaktan sorumludur. Docker depo (Repository) ise birbirleriyle ilgili imajlardan oluşan bir koleksiyon niteliğinde olup, aynı hizmet veya uygulamanın farklı sürümlerini barındırır. Docker depo (Repository) etiketli "tag" bir imaj kümesidir ve içerisinde yazılımın sürümlerini barındırır.

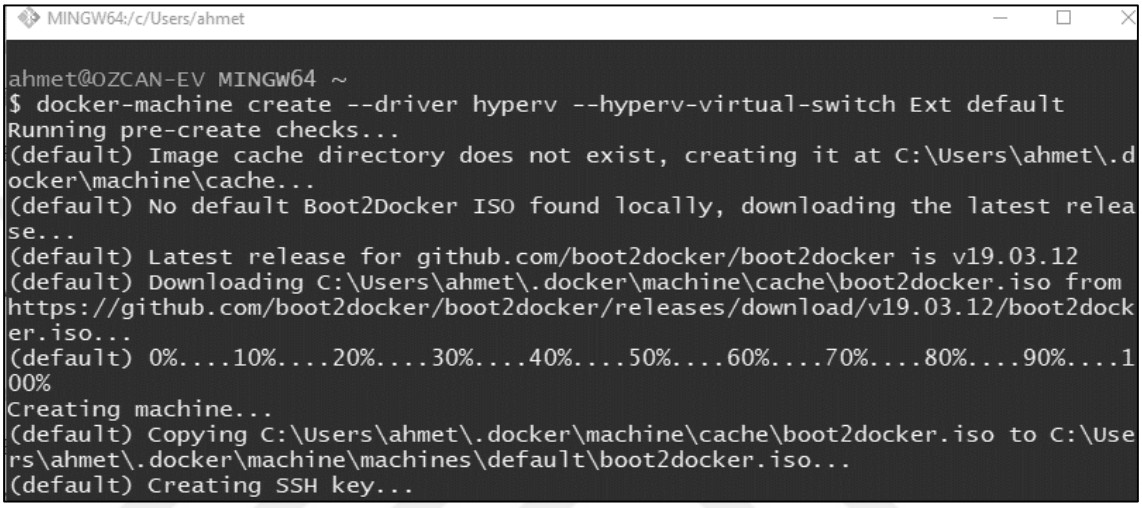
3.8.9. Docker Machine

Konteyner tabanlı bir uygulama sadece yerel bir işletim sistemine sahip Linux, Mac veya Windows ana makinesinde çalıştırılmayacaktır. Bu uygulama istenildiği zaman bulut ortamında sanal bir makine üzerinde de çalışabilecektir. Docker makinesi (Machine) uzakta bir sanal makine (SM) oluşturabilme ve üzerindeki konteynerleri yönetebilme yeteneğine sahiptir.
Uygulama geliştiriciler yerel bilgisayarında mevcut olmayan işlemci, bellek ve disk gibi sistem kaynaklarına bulut ortamlarında ulaşabilirler.

Saatlik ödeme kabul eden ve istedikleri özelliklere sahip bir makineyi ayağa kaldırarak tüm test süreçlerini gerçekleştirebilirler. İstenirse aynı anda farklı bulut sağlayıcılarında farklı makineleri ayağa kaldırarak test sürecine dâhil edebilirler.

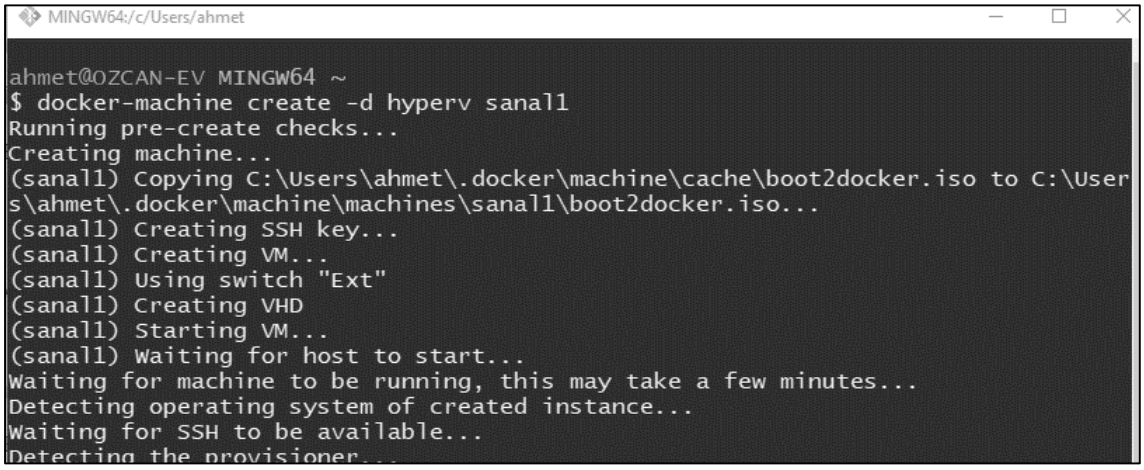
Docker makine yazılımı Docker Desktop yazılımından ayrı, işletim sistemine göre kurulmaktadır. Kurulum işlemlerini başlatabilmek için Docker makine için hazırlanan (https://docs.docker.com/machine/install-machine) internet sayfasında farklı işletim sistemleri için kurulum yönergeleri bulunmaktadır.

Windows işletim sistemi çalıştıran makinelere Docker makine kurulabilmesi için önce Git Bash konsolunun yüklenmesi ve Vmware, Virtualbox veya Hyper-V gibi sanallaştırma yazılımlarının yüklenmesi gereklidir. Git-Bash konsolundan yapılan yükleme işlemleri sonrasında kullanılan sanallaştırma platformu Virtualbox yazılımından farklı ise mutlaka gerekli eklentilerin yüklenmesi ve sanallaştırma yazılımının Docker Machine yüklenmesi gereklidir. Eklentilerin Git-Bash komut satırından yüklenmesi için kullanılacak komut Şekil 3.38'te görülmektedir.



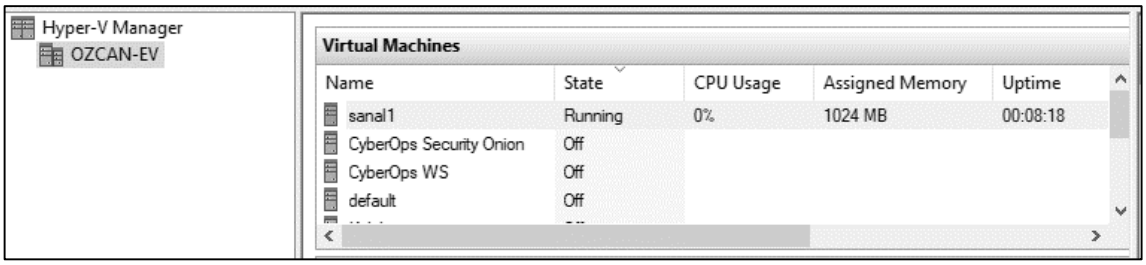
Şekil 3. 38. Docker-machine yükleme işlemleri

Bu işlem sonrasında aynı komut satırından yeni sanal makine Şekil 3.39'da görüldüğü gibi oluşturulabilecektir.



Şekil 3. 39. Docker-machine sanal makine oluşturma

Oluşturulan sanal makineye Şekil 3.40'da görüldüğü gibi Git-Bash komut satırından veya Hyper-V üzerinden erişim mümkün olacaktır.



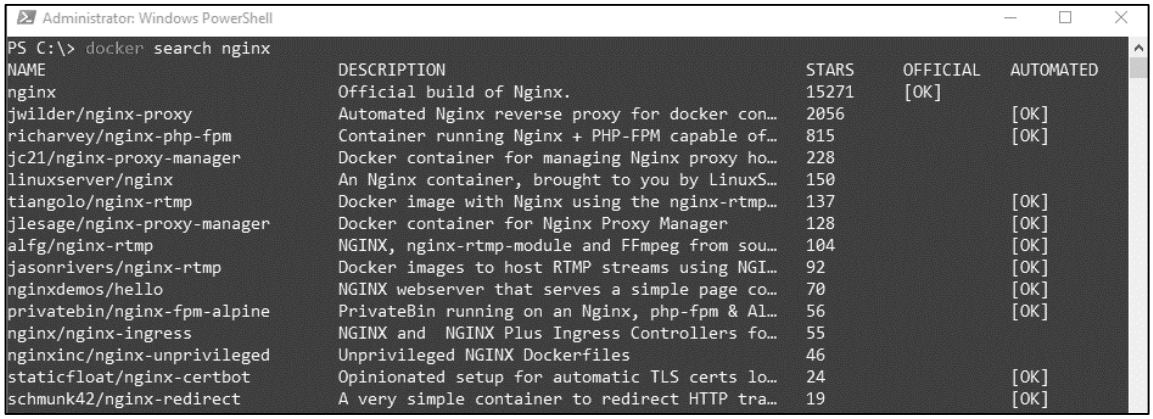
Şekil 3. 40. Docker makine' de oluşturulan sanal makinenin izlenmesi

3.9. Docker Konteyner Güvenliği

Siber güvenlik günümüzde veri merkezleri ve bulut servis sağlayıcıları yanında internet üzerinden yayın yapan pek çok medya için önemli bir konudur. Sanallaştırma yaklaşımının konteyner teknolojisine evrimleşmesi veya desteklenmesi bağlamında konteyner güvenliği oldukça önem taşımaktadır. Güvenlik konusunda oluşabilecek en küçük zayıflık çok büyük hasarlara yol açabilmektedir. Bilişim güvenliği uzmanları uzun zamandır sanallaştırma teknolojisinin zayıflıklarını tartışmaktadır. Ancak konteyner teknolojisi yeni bir yaklaşım olduğu için konu hakkındaki çalışmalar güncelliğini korumaktadır.

3.9.1. Zararlı İmajlardan Korunma

Docker imajları açık depolardan (repository) indirebilmektedir. Bu durum, konteynerleri oluşturan kişi veya kurumlara güvenilmesi anlamına gelmektedir. Bu sebeple indirilecek imajların Şekil 3.41'de görüldüğü gibi resmi (official) kurumların veya Docker dağıtım ücretli imaj servisini kullanmak bir çözüm olabilir. Alternatif olarak indirilen imajların içeriğinin ne olduğu ve hangi servislerin çalıştığı incelenebilir.



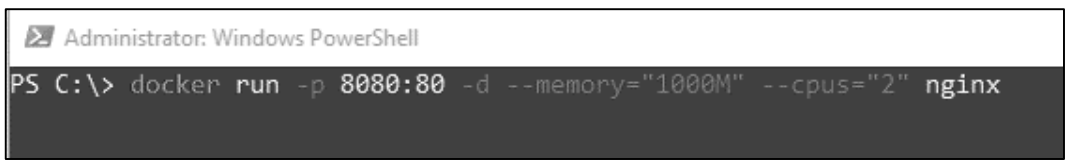
Şekil 3. 41. Resmi konteyner imajlarının listelenmesi

Docker, son yıllarda "Docker Güvenli İçerik" (DGİ) özelliğini geliştirmiştir. Bu özellik sayesinde indirilecek imajların orjinalliği, bütünlüğü ve yayınlanma tarihi doğrulanabilmektedir. Bu özellik varsayılanda aktif olmayıp sonradan aktifleştirilebilmektedir. Etkinleştirildiğinde ise imzalanmamış imajlar indirilememektedir.

3.9.2. Hizmet Reddi (Denial Of Service) Saldırılarından Korunma

Konteynerlerin ve üzerinde çalıştıkları sistemlerin HR saldırılarından korunmak için uygulanabilecek birkaç yöntem bulunmaktadır. Bu yöntemler arasında kaynakların sınırlanması, bellek ve işlemci kullanımının kısıtlanması ve işlemci döngüsünün sınırlandırılması sayılabilir.

Docker konteynerlerinin çalıştığı bir sistemde varsayılan olarak herhangi bir kaynak kısıtlaması bulunmamaktadır. Çalışan bir konteynerde sorun oluştuğunda ve çalışılan sistemin tüm kaynakları tüketilmeye başlandığında sorunlar büyüyecektir. Bu sebeple bir konteyner çalıştırılırken Şekil 3.42'de görüldüğü gibi kaynak limitlerinin belirtilmesi gereklidir. Bu işlem çalışacak konteynerin kullanabileceği maksimum sistem kaynağını sınırlayacaktır.

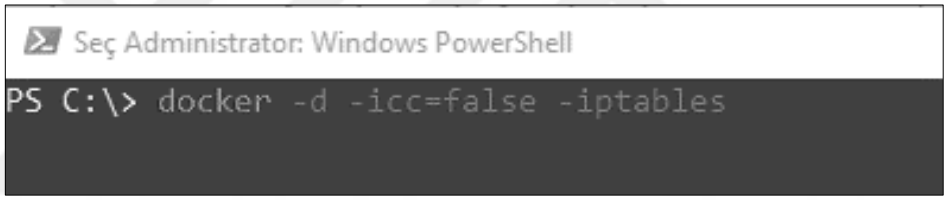


Şekil 3. 42. Çalıştırılacak konteynerde sistem kaynaklarının sınırlandırılması

3.9.3. Yetki Yükseltme ve Sömürmeden Korunma

Konteyner dünyasında sıklıkla karşılaşılabilecek sorunların başında yetkisiz erişimler ve erişim seviyelerinin yükseltilmesi görülmektedir. Bu problemin çözümü için uygulanabilecek birkaç yöntem bulunmaktadır. Konteynerler arası gereksiz iletişimin engellenmesi, kök (Root) kullanıcısının kullanılmaması ve bu kullanıcı izinlerinin başka kullanıcıya devredilmesi, Linux yeteneklerinin sınırlandırılması kullanılabilecek yöntemlerden birkaçıdır.

Uygulamaların çalıştırıldığı konteynerlerin fiziksel ortamla bağlantısı bulunmuyorsa veya diğer konteynerlerden izole çalışıyorsa, konteyner ortamının nispeten güvenli olduğu düşünülebilir. Böyle bir ortamdaki konteyner saldırıya uğramış olsa bile diğer konteyner ile iletişimde olmadığı için büyük sorunlara neden olmayacaktır [69]. Şekil 3.43'te yazılan komut ile konteynerler arası iletişim engellenebilmektedir.



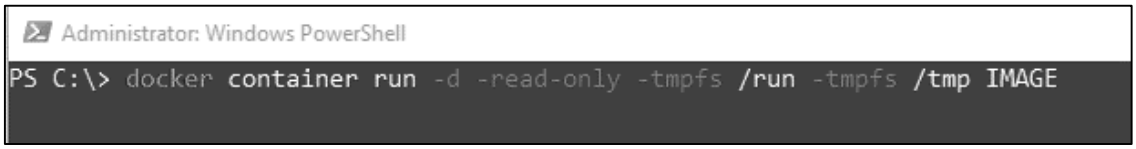
Şekil 3. 43. Konteynerler arası iletişimin engellenmesi

Konteynerler Linux işletim sistemi çekirdeğini kullandığı için, konteynerde çalışan işlemin bir Linux makinesinde çalışan işlemden farkı yoktur. Fiziksel bir Linux sisteminde Root kullanıcısıyla işlem yapılması tavsiye edilmediği gibi Linux çekirdeği üzerinde çalışan konteynerlerde de aynı durum geçerli kabul edilmektedir. Root kullanıcı yerine bilinen KK (Kullanıcı Kimliği) ve GKK (Grup Kullanıcı Kimliği) kimliğiyle bir kullanıcı oluşturulması önerilmektedir. Bu kullanıcı ile oluşturulan imajlara ve kaynaklara erişim sınırlandırılarak güvenli bir çalışma ortamı oluşturulabilir.

```
FROM ubuntu:latest
RUN groupadd -g 999 testgrubu && \
useradd -r -u 999 -g testgrubu testuser
USER testuser
CMD ["cat", "/tmp/gizli_dosya.txt"]
```

Şekil 3. 44. Linux'ta kullanıcılara yetki verilmesi

Linux işletim sistemlerinde, kullanıcılar "sistem" kullanıcıları veya "normal" kullanıcılar olarak ayrılmaktadır. Sistem yetkisine sahip kullanıcılar genellikle 1000'in altında bir kullanıcı kimliğine (KK) sahiptir ve programları çalıştırmak için Şekil 3.44'de görüldüğü gibi yetkilendirilir. Oturum açamazlar. Bu kullanıcılar web sunucularını çalıştırmak gibi yetkisine sahiptir. Diğer kullanıcı olan "normal" kullanıcıların tipik olarak 1000 veya daha yüksek KK'leri vardır ve sunucularda oturum açmalarına izin verilmektedir. Linux üzerinde yapılabilecek diğer sınırlandırma görevi ise Linux kabiliyetlerinin azaltılmasıdır. Kabiliyetler (capabilities) üzerinde çalışılan Linux "man" sayfasına göre bağımsız olarak etkinleştirilebilen veya devre dışı bırakılabilen farklı ayrıcalık durumlarıdır. İhtiyaç olmayan yeteneklerin kaldırılması yerine ihtiyaç duyulan yeteneklere izin verilmesi yaklaşımı bu sınırlandırma sürecinde daha başarılı olmaktadır. Bu işlem işletim sistemi seviyesinde uzmanlık gerektirmektedir. Konteynerlerin çalıştırılması esnasında arka planda yürütülen süreçler (process) izlenerek, gerekli olmayan işlemler belirlenmelidir. Şekil 3.45'te görüldüğü gibi Linux dosya sisteminin değiştirilmeye karşı salt-okunur hale getirilmesi ise yapılabilecek saldırıları önleyebilecek yöntemlerden biridir.



Şekil 3. 45. Linux dosya sisteminin salt-okunur hale getirilmesi

Çalıştırılan konteynerdeki dosyaların değiştirilmesi gerekmiyorsa salt-okunur hale getirilmesi yararlı olacaktır. Sisteme sızan saldırganın öncelikli yapmak isteyeceği işlem zararlı kodun uygulamaya yazılması olacaktır. Böylece uygulama her başlatılmasında zararlı kodu da beraberinde yükleyecektir. Eğer çalıştırılan konteyner imajı salt-okunur ise saldırgan her başlangıçta aynı işlemleri yapmak zorunda kalacaktır. Ancak uygulamaların salt-okunur yapılması pek çok uygulamanın başarısız olmasına neden olmaktadır. Bu sebeple uygulamaları "tmpfs" parametresi kullanarak geçici dosya sistemleri ile kullanmak hem kolay hemde güvenli olmaktadır.

4. GELİŞTİRİLEN ÇALIŞMA

Çalışmamız iki aşamadan oluşmaktadır. Çalışmamızın ilk aşamasında; uzaktan eğitim (UE) öğrencileri için sanallaştırılmış yazılım geliştirme ortamı oluşturulmuştur. Geliştirilen yazılım web tabanlı, güvenilir ve platformdan bağımsız yapıdadır [29]. Bu çalışma ile UE kurumlarına rehberlik etmek ve literatüre olumlu katkı sağlamak amaçlanmıştır.

Çalışmanın ikinci aşamasında; sanallaştırılmış laboratuvar uygulamalarında görülen olumsuzlukların giderilmesi amacıyla yenilikçi ve düşük maliyetli çözümler araştırılmıştır. Son yıllarda bulut teknolojilerinde yaygın kullanılan konteyner teknolojilerinin UE için laboratuvar altyapılarında kullanımı incelenmiştir. Konteyner teknolojisinin sanallaştırma çözümlerine alternatif olarak eğitim ortamlarında daha başarılı ve verimli olarak kullanılabileceği anlaşılmış ve bu çerçevede geliştirilen sistem ile bu görüş teyit edilmiştir. Bu bölümde sırasıyla yapılan çalışmalar açıklanacaktır.

4.1 Sanallaştırma Tabanlı Uzaktan Eğitim Yazılım Laboratuvarı

Bu çalışmada, uzaktan eğitimde karşılaşılan en önemli eksiklik olan laboratuvar eksikliği sorununun çözümü için bilgisayarlar, akıllı telefonlar ve tabletler için platformdan bağımsız, teknoloji tabanlı yeni bir laboratuvar sistem mimarisi geliştirilmiştir. Geliştirilen laboratuvar sistemi sayesinde öğrenciler lisans ücreti, kurulum, hız gibi maliyetler olmadan uzaktan, yer ve zaman sınırı olmaksızın bu laboratuvara bağlanabilmektedir. Böylece kurumların büyük maliyetlerle fiziksel laboratuvar kurmasına ihtiyaç yoktur.

Öğrenciler geliştirilen laboratuvara kampüs içinden ve dışından kolay erişebilmektedir. Kampüs içerisindeki öğrenciler doğrudan uzak masaüstü ile laboratuvara ulaşabilirken, kampüs dışından laboratuvara erişmek isteyen öğrenciler ağ geçidi sunucusu üzerinden sisteme erişebilmektedir. Aksi takdirde kampüs

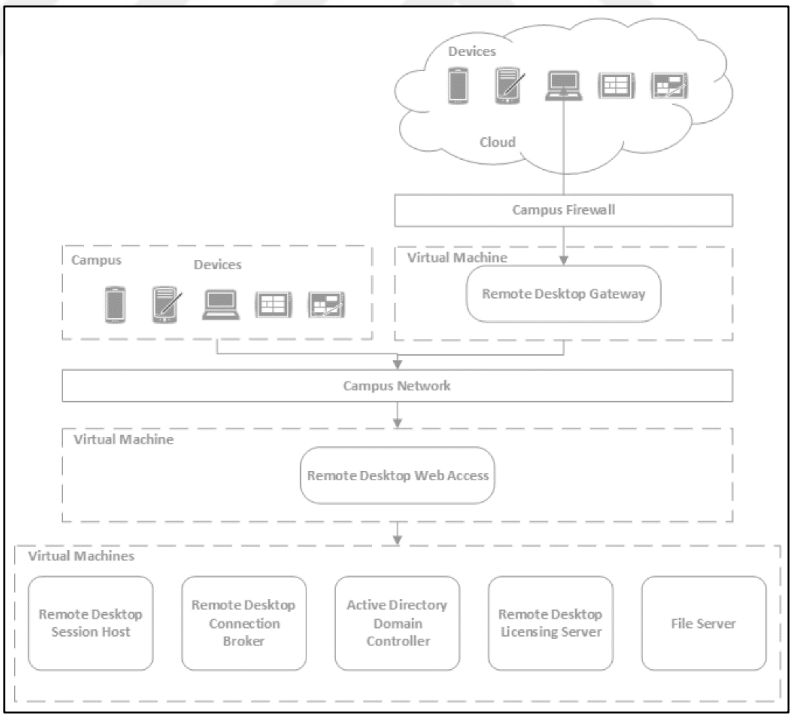
güvenlik duvarı sistem girişlerine izin vermemektedir. Sistemi kullanmak isteyen öğrenciler kendileri için tanımlanmış kullanıcı adı ve şifre bilgileri ile giriş yapabilmektedir. Geliştirilen platformdan bağımsız yazılım geliştirme laboratuvarına öğrenciler tablet ve akıllı telefonları ile de erişerek sistemi kullanmıştır.

Geliştirilen teknolojik tasarım veya uygulamanın olumlu veya olumsuz yönleri tarafsız olarak ele alınmalıdır. Tüketicilerin niyetlerini açıklayan Teknoloji Kabul Modeli (TKM), geliştirilen teknolojilerin değerlendirilmesinde kullanılmaktadır. TKM bilgi sistemleri modellerinde yaygın kullanılan modelleme yaklaşımı olup, kolay uygulanabilir ve anlaşılır özelliğe sahiptir. TKM modeli Nİ ve Endüstri 4.0 çalışmalarında sıkça kullanılmasına rağmen sanal veya uzak laboratuvar çalışmalarında literatürde örneği görülememiştir. Yapılan bu çalışma bu yönüyle literatüre katkı sağlamıştır. Çalışmada 43 öğrenciden oluşan rastgele iki grup kullanılmıştır. Birinci gruba geleneksel uzaktan eğitim atamaları yapılmıştır. İkinci gruba ise geleneksel uzaktan eğitim atamalarının yanında gelişmiş laboratuvar platformu açılmıştır. Her iki gruba da 4 hafta boyunca aynı dersler ve aynı ödevler verilmiştir. Sürenin sonunda her iki gruba da birer final ödevi verilmiş ve başarı oranları karşılaştırılmıştır. Ayrıca geliştirilen laboratuvar sistemini kullanan öğrencilerin davranışlarını ölçmek için TKM kullanılmıştır. Çalışmada geliştirilen laboratuvarı kullanan öğrencilerin laboratuvar değerlendirmelerinde daha başarılı (%12,89) not aldıkları ve eğitim sürecinden memnun oldukları gözlemlenmiştir. Çalışma sonrasında Güçlü Yönler, Zayıf Yönler, Fırsatlar ve Tehditler (GZFT) analizi yapılarak, geliştirilen sistemde kusurlu ve eksik yönler tespit edilmiş, çözüm önerileri sunulmuştur. Böylece TKM yaklaşımıyla literatüre iyi ve etkili bir katkı sağlanarak, literatürdeki boşluğu dolduracak bir çalışma yapılmıştır.

Önerilen laboratuvar yaklaşımında, öğrenciler laboratuvar ortamında aynı anda öğretmenin anlatımını dinleyebilmekte ve anında çalışabilmektedir. Öğretmenler, öğrencilere sanal ders araçları aracılığıyla rehberlik edebilmektedir. Öğrenciler sanal laboratuvarda hangi uygulamaları yapacağını öğrenerek, nasıl yapılacağını anlayabilir ve örnek kodları bulabilirler. Böylece geliştirilen laboratuvar ortamında öğrenci ve öğretmen arasında maksimum etkileşim sağlanmaktadır.

Geliştirilen laboratuvar sistemi, uygulamalı yazılım ihtiyacı olan bilim dallarının kullanımına uygundur. Geliştirilen sistemde ihtiyaca göre yeni yazılımlar eklenebilmektedir. Yapılan çalışma sonrasında, sistemi kullanan ve kullanmayan öğrencilerin durumları karşılaştırılmıştır. Ayrıca sistemi kullanan öğrencilere sistem hakkındaki görüşleri bir anket vasıtasıyla sorulmuştur. Geliştirilen laboratuvar sistem mimarisi Şekil 4.1'de görülmektedir.

Sanallaştırma mimarisinin ana bileşeni olarak uzak masaüstü protokolü (Remote Desktop Protocol) kullanılmıştır. Uzak masaüstü; kullanıcının uzaktaki bir sunucuyu kendi ekranından yönetebilmesine imkân sağlayan yazılımdır. Uzak masaüstü protokolünün kullanılabilmesi için geliştirme ortamında birden fazla sunucuya ihtiyaç vardır. Bazı sunucu rolleri aynı sunucuya atanabilirken, bazı roller sadece bir sunucuda bulunmak zorundadır. Geliştirilen sistemde zorunlu bulunması gereken sunucu rolleri 3 ayrı sunucuya atanmıştır.



Şekil 4. 1. Sanallaştırma yazılım geliştirme laboratuvarı mimarisi

Çizelge 4. 1. Fiziksel sunucu, Sanallaştırılmış sunucu ve konteyner teknolojileri

Parametreler	Fiziksel Sunucu Teknolojisi	Sanallaştırılmış Sunucu Teknolojisi (SM)	Konteyner Teknolojisi
Konuk İşletim Sistemi	Fiziksel donanım üzerine kendi işletim sistemini yükler ve kullanır. İşletim sistemi, işlemci ve belleği paylaşmaz.	Her sanal sunucu kendi bellek bölgesini ve işletim sistemini kendi donanım aygıtlarında çalıştırır.	Tüm konuklar aynı işletim sistemi çekirdeğini fiziksel belleğe yükleyip kullanır.
İletişim	Fiziksel ağ aygıtlarını kullanır.	Ağ aygıtlarından kendine izin verileni kullanır.	Standart IPC mekanizması kullanır. Sinyal, soket vb.
Güvenlik	Yüklenebilecek güvenlik yazılımları ile işletim sistemi güvenliğine bağlıdır.	Sanallaştırma yöneticisine bağımlıdır.	Zorunlu erişim kontrolü vardır.
Performans	Sistem kaynaklarının tümünü kullanır. Donanım değişikliği ile performans artırılabilir.	Sunucular kendilerine ayrılan sistem kaynakları kadar performans gösterirler.	Üzerinde çalıştıkları yerel sunucunun performansına yakındır
Yalıtım	Ağ üzerinden başka bilgisayarlar ile dosya ve klasör paylaşımı yapabilir.	Sanal sunucular ağda dosya ve kütüphaneleri paylaşabilirler. Doğrudan fiziksel ile bağlantı kuramazlar.	Alt klasörler şeffaf bağlanabilir, paylaşılabilir.
Başlatma Süresi	Onlarca dakikayı bulabilir. Üzerinde çalışan servislein durumuna bağlıdır.	Sanal makineler birkaç dakikada başlatılabilir.	Konteynerler birkaç saniyede başlatılabilir.
Depolama	Sistem kaynağı olarak tüm disk alanını kullanabilir.	Sanal makineler konuk oldukları işletim sistemi üzerinde Gigabyte seviyesinde depolama alanı kullanır.	İşletim sistemi üzerinden az miktarda depolama alanı kullanır.

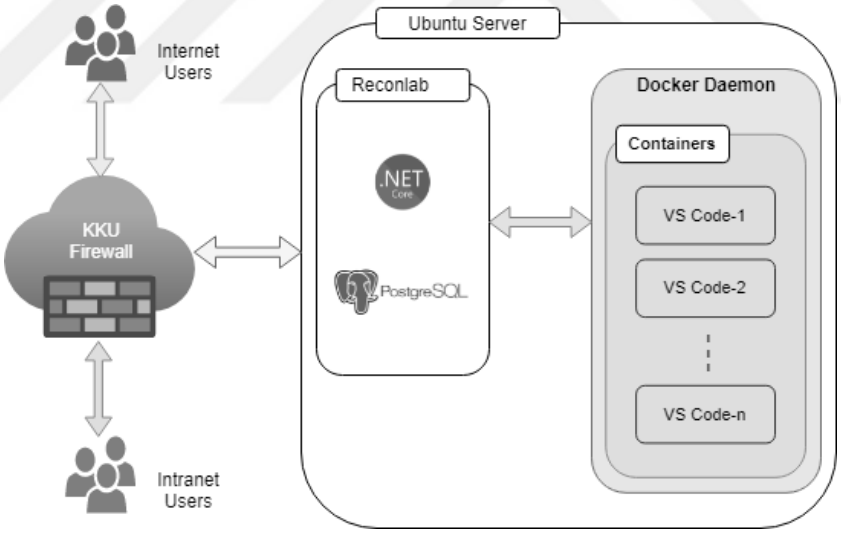
Ancak, sanallaştırma platformlarının olumsuz yönleri de bulunmaktadır. Uzak masaüstü bağlantı lisans ücretleri, uzak masaüstü bağlantı portlarının bilinen güvenlik problemleri ve sanallaştırma teknolojisinin yüksek bellek tüketimi bu sorunların başında gelmektedir. Çalışma esnasında karşılaşılan bu problemler, çalışmada düşük maliyetli, güvenli ve düşük bellek tüketimine sahip teknolojilerin kullanılmasını gerektirmiştir. Bulut platformlarında yaygınlaşmaya başlayan düşük maliyet, verimlilik ve güvenlik konusunda olumlu yönleri bulunan konteyner teknolojisinin çalışmanın ikinci aşamasında kullanılmasına karar verilmiştir.

4.2. Konteyner Tabanlı Uzaktan Eğitim Yazılım Laboratuvarı

Çalışmanın ikinci aşamasında konteyner teknolojileri incelenerek, geliştirilecek sistem için uygun konteyner teknolojisi seçimi yapılmıştır. Bu aşamada sürekli güncellenen,

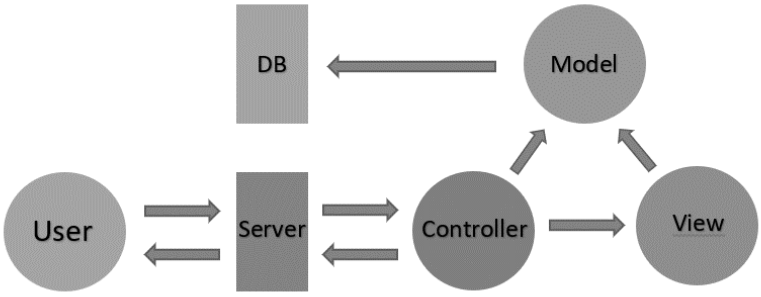
yaygın kullanıma sahip, ücretsiz ve çoklu platform desteği bulunan konteyner teknolojisi olan Docker tercih edilmiştir. Docker konteyner teknolojisi, günümüzde konteyner uygulamaları için yaygın kullanılan, ücretsiz, sürekli güncellenen ve farklı işletim sistemleri üzerinde çalışabilen yapıya sahiptir.

Docker konteyner teknolojisi, Windows ve Linux platformlarında problemsiz çalışmaktadır. Geliştirilecek sistemin her iki platform üzerinde çalışabilecek yapıya sahip olması planlanmıştır. Konteyner teknolojisi Linux işletim sistemi çekirdeğini (kernel) kullandığı için bu amaca uygun uyumlu Windows işletim sistemi sürümleri (Linux için Windows Alt Sistemi – LWAS desteği olan) kullanılmıştır. Geliştirilen sistem Windows10, Windows Server 2016 ve sonrası işletim sistemlerinin yanında Ubuntu, CentOs v.b. Linux işletim sistemi sürümlerinde de çalışmaktadır. Ayrıca bir Unix türevi olan MacOS işletim sisteminde de çalışmaktadır. Bu özelliğiyle geliştirilen sistem platformdan bağımsız bir yazılım özelliği taşımaktadır. Şekil 4.2'de görüldüğü gibi çalışma .Net Core 5.0 framework mimarisiyle geliştirilmiştir.



Şekil 4. 2. Önerilen proje mimarisi

Microsoft firması tarafından geliştirilen. Net Core 5.0 yazılım çerçevesi, farklı işletim sistemleri üzerinde çalışabilmekte ve bu işletim sistemleri için yazılım geliştirilmesine olanak vermektedir. Proje çalışmamızda kullanılan bir başka önemli özellik ise, MGD (Model-Görünüm-Denetim) mimari deseninin Şekil 4.3'te görüldüğü gibi kullanılmasıdır.



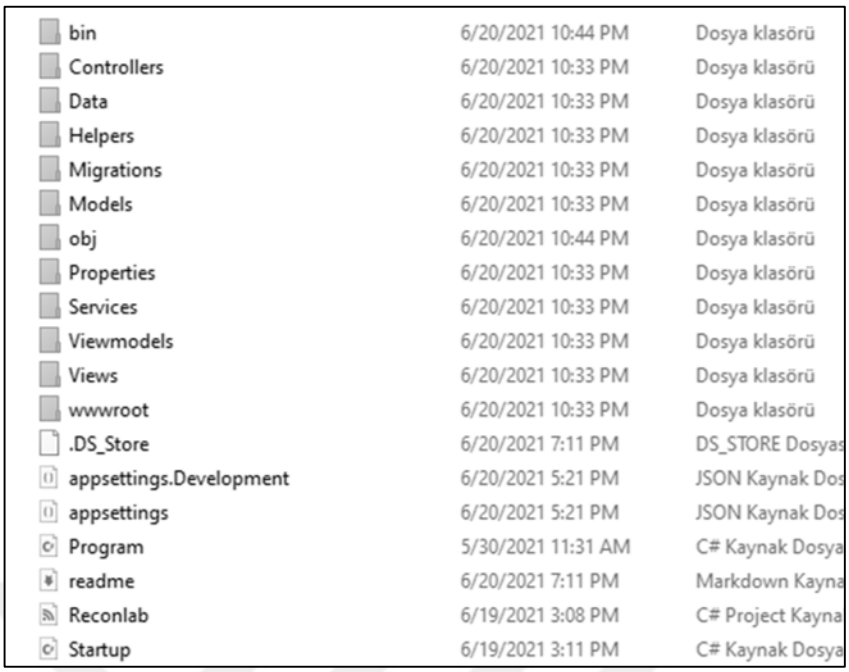
Şekil 4. 3. MGD Katmanlı modeli

MGD ifadesinde kullanılan her kelime ayrı bir katmanı ifade etmektedir. Bu sebeple hazırlanan proje yazılımı katmanlı mimaride tasarlanmıştır. Model katmanı, uygulama verisinin veya durumunun saklandığı katmandır. Genellikle veritabanı veya GİD/JSNG dosyasıdır. Bu katman veri katmanını uygulamadan izole ederek, diğer katmanların veritabanının nerede olduğunu bilmesine gerek kalmaz. Model katmanı genellikle Entity Framework, Nhibernate v.b. araçlar kullanılarak oluşturulur. Çalışmamızda Entity Framework aracı kullanılmıştır.

Görünüm katmanı, istemcilerin görecekleri arayüzü içeren katmandır. View katmanı Model katmanındaki verilerle oluşturulmaktadır. Görünüm katmanının Model ve Denetim katmanından ayrı olması sebebiyle ara yüz değişiklikleri diğer katmanları değiştirmeden kolayca uygulanabilmektedir.

Denetim katmanı, istemciden gelen isteklerin işleyerek Model ve Görünüm katmanları arasında köprü görevini yerine getirmektedir. Denetim içinde çok sayıda hareket bulunabilir.

Çalışmamızda konteyner imajlarının yüklenmesi, yüklenmiş imajların istemci istekleri doğrultusunda çalıştırılması bu katmanda gerçekleşmektedir. Ayrıca bu katmanda farklı işletim sistemlerine ait kabuk programlar (Shell Script, bash script) çalıştırılabilmektedir.



Şekil 4. 4. Proje klasör görünümü

.Net Framework ve MGD kullanarak hızlı çalışan, tekrar kullanılabilir ve test edilebilir web ara yüzü elde edilmiştir. Şekil 4.4'de proje klasörleri ve proje dosyaları görülmektedir. Çalışma Visual Studio EGO ortamı kullanılarak C# dilinde yazılmıştır. Bu yönüyle halen Kırıkkale Üniversitesi Uzaktan Eğitim ÖYS ortamının yazıldığı programlama dili ile uyumludur.

Çalışmada, Postgresql veritabanı uygulaması kullanılmıştır. Bu uygulama ücretsiz, açık kaynaklı, sağlam ve performans açısından yeterli olmasının yanında 30 yılı aşkın aktif gelişme sürecindedir. Ayrıca Windows, MacOS, Solaris, SuSE, Ubuntu Linux ailesi ve RedHat Linux ailesinde de desteklenmektedir. PostgreSQL aktarım, altsorgı, tetikleyici, görünüm, yabancı anahtar kaynak bütünlüğü ve geleneksel kilit gibi (user-defined types), kurallar, miras ve kilit cakışmalarını düşürmek için çoklu sürüm uyumluluk özellikleri rakipleri karşısında güçlü olduğu özellikleridir.

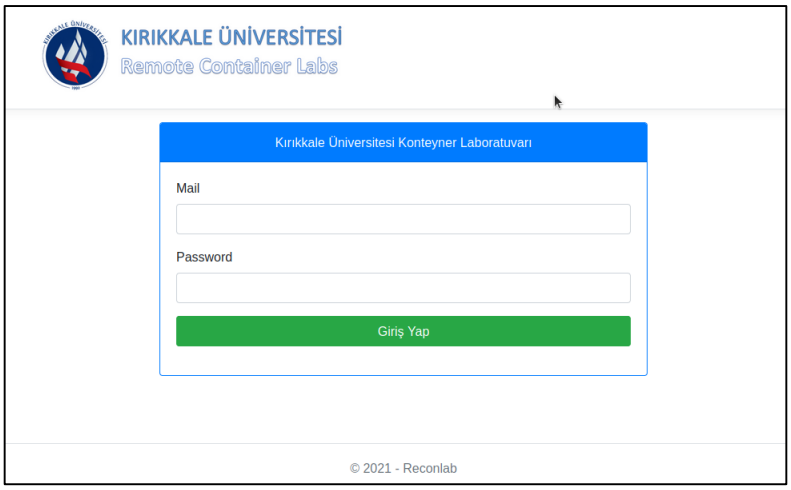
Konteyner dünyasında aynı işlevlere sahip farklı konteyner türleri bulunmaktadır. LK (Linux Container), Rocket, Mesos, OpenVZ, Docker gibi konteynerler yaygın kullanımdadır. Konteyner türlerinin farklı üstün yönleri bulunabilmektedir. Örneğin Rocket (rkt) özellikleri arasında Güvenilir Platform Modülleri (GPM) desteği bulunmaktadır. OpenVZ diğer konteyner çalışma zamanlarına kıyasla daha düşük bir

belleğe gereksinim duymaktadır. Linux Container'lerde Kubernetes desteği henüz yoktur.
Docker konteyner teknolojisi günümüzde en yaygın kullanılan konteyner teknolojisi olup, kullanım kolaylığı, sürekli güncellenmesi sebebiyle geliştiriciler tarafından çok tercih edilmektedir.

Ayrıca Docker firmasının kullanıcıları için ücretsiz sunduğu "http://hub.docker.com" sitesinde kullanıma hazır çok sayıda konteyner imajı bulunmaktadır. Geliştirdiğimiz konteyner tabanlı yazılım geliştirme ortamına RECONLAB (Remote Container Labs) ismi verilmiştir. Projemiz .Net Core 5.0 Framework altyapısı kullanılarak, MGD deseniyle tasarlanmıştır. Veritabanı uygulaması olarak Postgresql seçilmiştir. RECONLAB projesinde konteyner türü olarak Docker konteyner mimarisi benimsenmiştir. Proje çalışmasında örnek bir docker konteyneri (Visual Studio Code) seçilmiştir. Seçilen konteyner kullanıcılara yazılım EGO (Entegre Geliştirme Ortamı) sağlamaktadır. Seçilen EGO C, C#, Java, Python, PHP, Go, Javascript, NodeJS desteği sağlamaktadır. Söz dizimi vurgulama, kod algılama ve diğer gelişmiş özellikleri ile Visual Studio'nun bazı özelliklerini barındırmaktadır. Hatta istenilen programlama dilinin eklentisi indirilerek yazılıma yeni özellikler eklenebilmektedir. Visual Studio yazılımı sadece Windows işletim sistemlerinde kullanılabilirken, Visual Studio Code yazılımı platformdan bağımsız çalışmaktadır.

4.2.1. Proje Erişim Sayfası

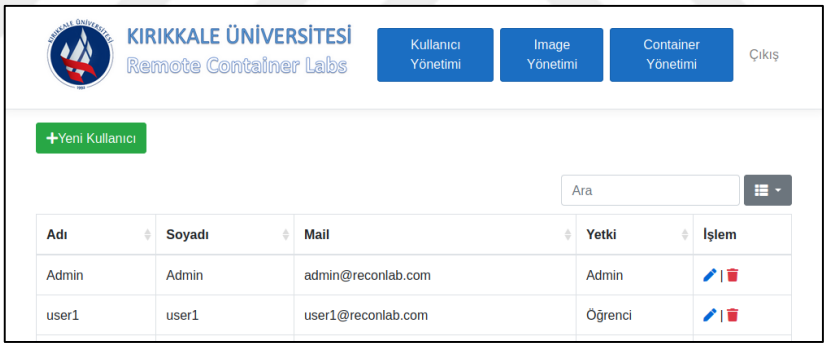
Proje çalışmamız sunucu üzerinde derlenerek çalıştırıldığında internet tarayıcı üzerinde http://sunucuadresi:5000 adresi ile yazılımın web ara yüzüne ulaşılabilmektedir. Kullanıcı e-posta adresi ve parola siteme giriş yapılabilmektedir. Şekil 4.5'de görüldüğü gibi e-posta adresi dışında parola ifadesi gizlenmiştir.



Şekil 4. 5. Proje erişim sayfası

4.2.2. Kullanıcı Yönetimi

Proje yazılımının Kullanıcı Yönetimi linki tıklandığında, Şekil 4.6'da görülen ekranda mevcut kullanıcılar görüntülenmektedir. Yeni Kullanıcı linki tıklanarak iki farklı kullanıcı türü tanımlamak mümkündür.

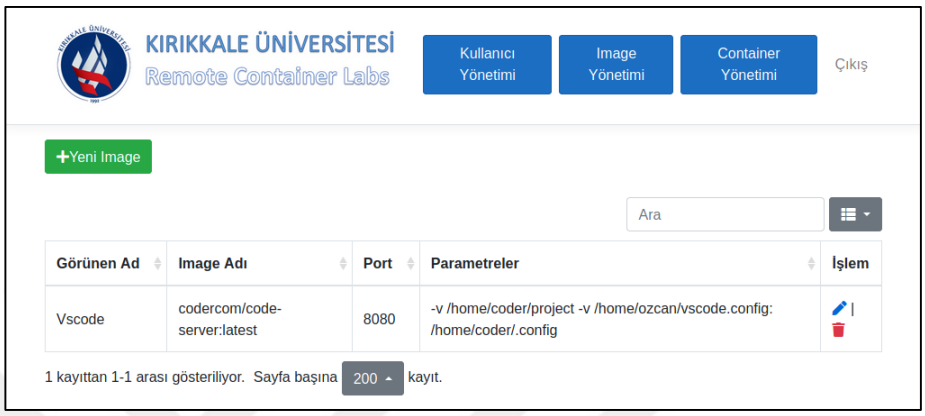


Şekil 4. 6. Kullanıcı yönetimi ekranı

4.2.3. İmaj yönetimi

Image Yönetimi bölümünde ise kullanıcıların hizmetine sunulacak konteyner imajlarının sisteme yüklenmesi Şekil 4.7'de görüldüğü gibi gerçekleştirilmektedir. Ayrıca bu ekranda daha önceden yüklenmiş bulunan konteyner imajlarının port ayarı ve parametre ayarları yapılmaktadır. Çalışma kapsamında varsayılan olarak Visual Studio Code imajından oluşturulan konteyner bilgisi yüklenmiştir. Farklı uygulamalara ait konteynerlerin platforma yüklenerek kullanıcılara sunulması mümkündür. Yazılımcılar tarafından önceden oluşturulan konteynerlerin çalışabilmesi

için farklı ayarlara, farklı çalışma klasörlerine sahip olabilmektedir. Özellikle farklı konteyner uygulamalarının Linux veya Windows işletim sisteminde bulundukları klasörler ve bu klasörlerin tanımlanmış yolu (path) değişebilmektedir. Bu değişiklikler bu ekranda bulunan Parametreler bölümünde düzenlenmektedir.



Şekil 4. 7. İmaj yönetimi ekranı

4.2.4. Konteyner yönetimi

Konteyner yönetimi ekranında ise çalışmakta olan konteyner bilgisi yanında konteynerin durdurulması veya durmuş bir konteynerin silinerek sistem yükünün azaltılması Şekil 4.8'de görüldüğü gibi mümkündür. Çalıştırılan bir konteyner durdurulana kadar veya ilgili servis öldürülene kadar çalışmaya devam edecektir. Ancak bu durum yoğun kullanılacak sistemde performansın düşmesine ve belleğin gereksiz yere tüketilmesine neden olacaktır. Bu sebeple sistemde oturumunu kapatmış kullanıcılara ait konteynerlerin durdurulması ve bellekte yüklü konteynerlerin silinmesi gereklidir. Kullanıcı sayısının az olduğu durumlarda bu ekrandan durdurma ve silme işlemleri yapılabilmektedir.

Kullanıcı sayısının fazla olduğu ve kontrolün güç olduğu durumlar için ayrıca bir servis yazılmış (ContainerCleanupService) ve bu eksiklik giderilmiştir. Bu servis başlatılan konteynerlerin 120 dakika sonra otomatik olarak silinmesini sağlamaktadır. Belirlenen silinme süresi ihtiyaca göre değiştirilebilmektedir. Bu ekranda ayrıca konteyner başlatma esnasında oluşabilecek hatalar görüntülenerek olası hata durumunda problem çözümü sağlanmaktadır.

Şekil 4. 8. Konteyner yönetimi ekranı

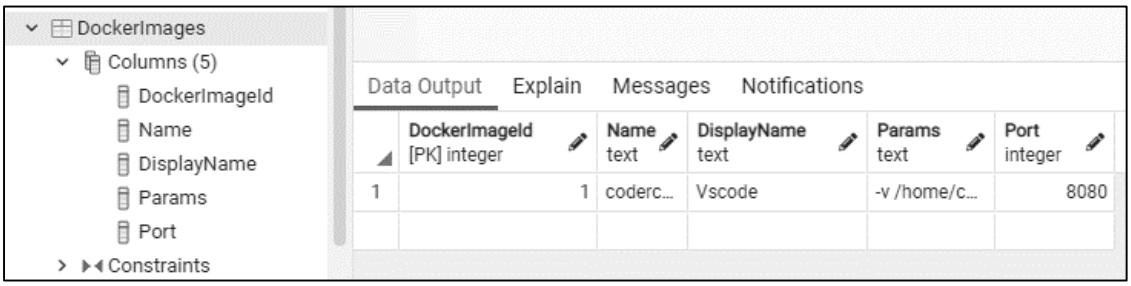
Kullanıcı ana sayfasında sisteme yüklenmiş ve başlatma ayarları yapılmış çalışabilir konteynerlerin başlatılabilmesi için linkler bulunmaktadır. Çalışmamızda Visual Studio Code uygulamasına ait konteyner Şekil 4.9'da yapılandırılmış ve çalışmaya hazır hale getirilmiştir.

Şekil 4. 9. Kullanıcı ana sayfa ekranı

4.2.5. Konteyner Başlatma Süreçleri

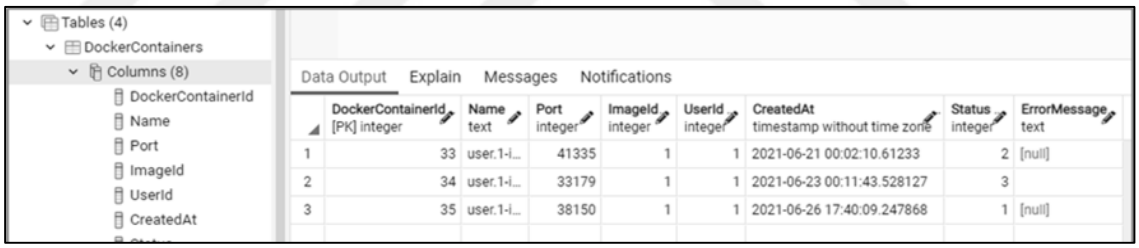
Proje ana sayfasında çalıştırılmak istenen konteynerin karşısında bulunan buton tıklandığında aşağıda listelenen işlem basamakları gerçekleşmektedir;

- Şekil 4.10'da görüldüğü gibi çalıştırılmak istenen bir imaj sisteme yüklendiğinde veri tabanında bir kayıt oluşturulur. Bu kayıtta imaj için DockerImages tablosunda bir ID tanımlanır (DockerImageId).



Şekil 4. 10. Docker imajları tablosu

- İlgili "ImageId" "Docker_Container_Controller / Run" a parametre olarak girilir.
- "ImageId" varlığı veritabanında kontrol edilir. Image bulunamazsa kullanıcı ana sayfaya yönlendirilir ve hata mesajı verilir.
- "UserId_Session" bilgisi alınır.
- Şekil 4.11'de görüldüğü gibi veri tabanında "DockerContainers" tablosunda "UserId" ve "ImageId" bilgisiyle konteyner oluşturulur.

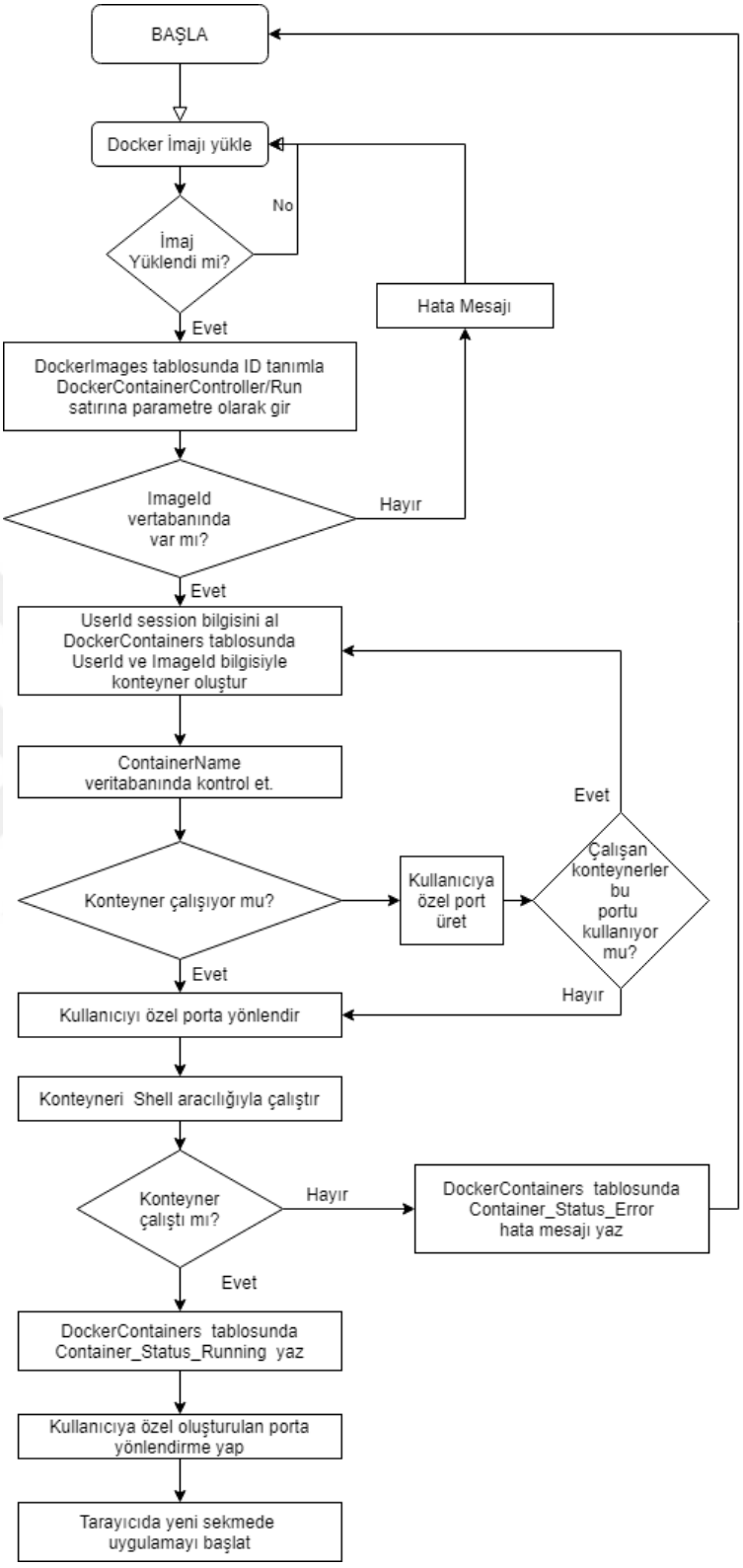


Şekil 4. 11. Docker konteyner tablosu

- "ContainerName" veri tabanından kontrol edilir. Eğer konteyner çalışıyorsa, kullanıcıya özel porta yönlendirme yapılır.
- "ContainerName" veri tabanından kontrol edildiğinde eğer konteyner çalışmıyorsa, kullanıcıya özel bir port üretilir. Veri tabanında çalışan herhangi bir konteyner bu portu kullanıyor mu? Kontrol edilir. Kullanıyorsa işlem tekrar edilir.
- Yeni konteyner objesi oluşturulur. Veri tabanında "DockerContainers" tablosuna kaydedilir.

- Konteyner “Shell” aracılığıyla çalıştırılır. Çalıştırma esnasında hata oluşursa, veri tabanında “DockerContainers tablosuna Container_Status_Error” olarak hata mesajı yazılır. Kullanıcı ana sayfaya yönlendirilir.
- Çalıştırma hatasız gerçekleşirse, “Container_Status_Running” olarak veri tabanına kaydedilir. Kullanıcıya özel oluşturulan porta yönlendirme yapılır. Yeni sekmede uygulama başlatılır.

Tez çalışması kapsamında geliştirilen Uzak Konteyner Laboratuvarı (RECONLAB) yazılımına ait konteyner başlatma süreçlerine ilişkin akış diyagramı Şekil 4.12’de görülmektedir.



Şekil 4. 12. RECONLAB akış diyagramı

4.2.6. Çalışma Test Süreçleri

Yazılımda test süreci hazırlanan ürünün istenilen seviyede olup olmadığını belirlemek amacıyla yapılmaktadır. Ürün istenilen seviyede değilse hataların, eksiklerin tespiti ve düzeltme işlemleri de bu sürecin bir parçasıdır. Test süreci yazılım proje döngüsünün önemli bir parçasıdır [70].

Günümüzde bulut bilişim ve bulut hizmet sağlayıcıları yazılım geliştirme ekiplerinin önemli bir parçası haline gelmiştir. Bulut bilişimin gelişiminde önemli rolü bulunan sanallaştırma teknolojisi ve konteyner teknolojisi yazılım geliştirme ve operasyon (YGUE) yaklaşımının benimsenmesine önemli katkıları olmuştur.

Sanallaştırma teknolojisi yöneticileri (donanım destekli/yerel ve barındırılan) ve konteynerlerin (Docker, LK, LAPP, RKD, KÇZA v.b.) seçilebilecek çok alternatifi bulunmaktadır. Seçilen sanallaştırma yöneticisi veya konteynerlerin performans düzeyinde karşılaştırmalarının yapılabilmesi için doğru metriklerin seçilmesi ve konuşlandırılacak uygulamanın kıyaslamaya uygun şekilde yapılandırılması gereklidir [71].

Bu sorunların çözümü, geliştiricilerin farklı sanallaştırma ortamlarına uygun yazılım hazırlamasıdır. Bu durum ise geliştiricilerin her yeni teknoloji hakkında bilgi edinmek için zaman harcamasına neden olacaktır. Ayrıca kurulum ve en uygun şekle sokma için harcanacak zamana ihtiyaç vardır.

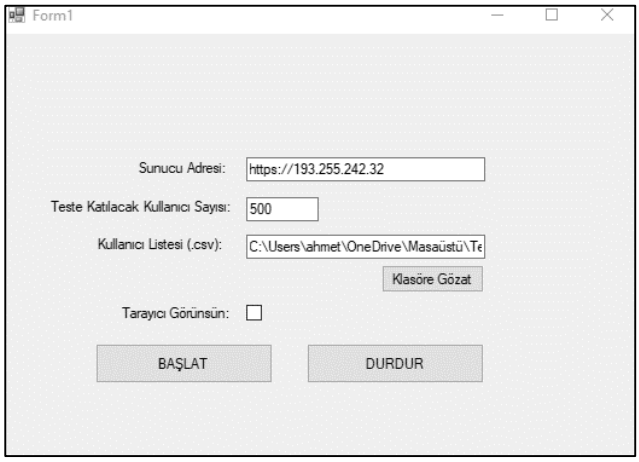
Genel çözüme yönelik olarak bu çalışmada aynı sanallaştırma ortamını kullanan ve aynı kodlama mimarisine (.Net Core 5.0) sahip fakat farklı işletim sistemlerinde çalışan aynı yazılım geliştirme (EGO) konteynerlerinin performans karşılaştırılması yapılarak güvenilir bir referans oluşturması amaçlanmıştır.

Farklı sürümlere sahip farklı işletim sistemi tabanlı konteyner içinde bulunan uygulamalar farklı performanslar gösterebilmektedir [72]. Bu sebeple her iki işletim sistemi platformu içinde aynı konteyner uygulaması seçilmiştir.
Test aşaması, yapılan çalışmanın uygunluğunu ve doğruluğunu denetleme sürecidir. Bu aşamada beklenen gereksinimleri karşılayıp karşılamadığı performans testi ile

belirlenecektir [73]. Performans testleri yazılım uygulamalarının belirlenen yükler altında hangi davranışı gösterdiğini ortaya koyar [74]. Performans testinin Yük (Load), Stres (Stress), Dayanıklılık (Endurance), En Tepe (Spike), Sel (Flood), Ölçeklenebilirlik (Scalability) gibi çeşitleri bulunmaktadır.

Test çalışmasında Kırıkkale Üniversitesi Bilgi İşlem Dairesi Başkanlığı kontrolünde bulunan veri merkezi üzerinde oluşturulan sanal sunucular kullanılmıştır. Sunuculara atanabilecek bellek, işlemci sayısı, disk boyutu veri merkezi kaynaklarıyla sınırlıdır. Bu sebeple veri merkezince ayrılabilecek en fazla işlemci sayısı dört adet, ayrılabilecek bellek miktarı 256 GB, disk boyutu ise 240 GB olarak belirlenmiştir.

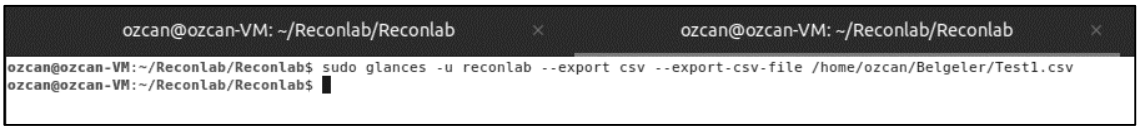
Test çalışması öncesi test parametreleri olarak kullanıcı sayısı, cpu sayısı ve bellek miktarı belirlenmiştir. Test ortamı olarak Kırıkkale Üniversitesi veri merkezi bünyesinde bulunan Vmware sanallaştırma (vSphere Client version 6.7.0.20000) ortamında bulunan aynı özelliklere (Cpu, Ram, Disk, Nic) Intel işlemcili (Intel Xeon 14-core e5-2680 v4) bilgisayarlar kullanılmıştır. Kullanıcıların yoğun saatlerde davranışlarını simüle edebilen bir test paketi olan Reconlab-Test programı C# programlama diliyle yazılmıştır. Şekil 4.13'de görüldüğü gibi test uygulamamız Reconlab Konteyner Laboratuvarı ortamına istenilen sayıda oturum açma isteği gönderen bir yazılımdır.



Şekil 4. 13. Reconlab test programı ekranı

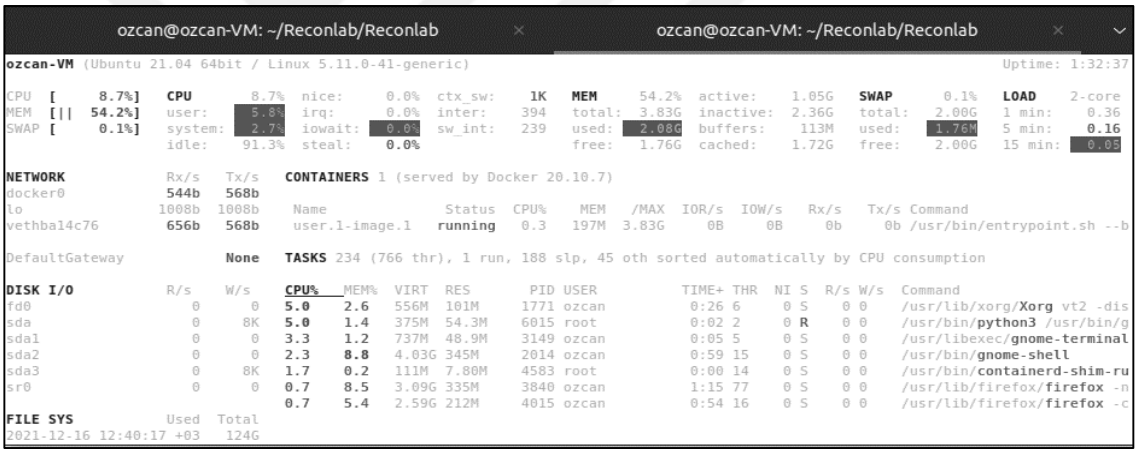
Glances izleme yazılımı ise, Windows ve Linux işletim sistemlerinde çalışabilen bir yazılımdır. Yazılım Linux işletim sistemine kurulduktan sonra Şekil 4.14'de

görüldüğü gibi terminal üzerinden komutlarla yönetilebilmektedir. İzlenen verilerin VAD, JSNG ve GİD formatlarında çıktılarını verebilmektedir.



Şekil 4. 14. Ubuntu sunucuda test çıktı parametrelerinin kullanılması

Şekil 4.15'de görüldüğü gibi glances etkileşimli bir araç olmayıp, sadece izleme için kullanılmaktadır [75]. Linux ortamlarında sistem kaynaklarının izlenmesi için çeşitli araçlar bulunmaktadır. Glances izleme yazılımı benzeri olan "Top", "Htop" gibi araçlardan farkı Linux yanında Windows platformunda çalışabilmesinin yanında daha düzenli ve anlaşılır ekran görüntüsü sunmasıdır.



Şekil 4. 15. Ubuntu sunucuda test izleme ekranı

Windows ve Linux kurulumları test etmek için, gerçekçi bir etkileşim gerçekleştiren 100, 200, 300, 400 ve 500 eşzamanlı sentetik kullanıcı simüle edilmiştir. Her kullanıcı bir ÜMAP isteği oluşturarak sisteme önceden kaydedilmiş bulunan farklı kullanıcı adı ve parola ile sisteme giriş yaparak sistemde bulunan örnek konteyner uygulamasını çalıştırmıştır.

Sistemi kullanan her kullanıcı için bir Docker imajı oluşturulmuştur. Docker hizmeti başlangıçta tek Cpu (Intel Xeon 14-core e5-2680 v4) ve 32 GB DDR3 RAM içeren bir sanal makine üzerinde oluşturulmuş ve çalıştırılmıştır. Sonrasında ise 32 GB, 64 GB, 96 GB, 128 GB ve 256 GB arasında değişen bellek ve 4 CPU sayısına sahip sanal

makineler üzerinde çalıştırılmıştır. Test için kullanılacak takas alanı, bellek büyüklüğünün 4 katı olarak belirlenmiştir. Test edilecek çalışma disk üzerinde ağırlıklı olarak okuma işlemi yapacağı, yazma işleminin okumaya göre çok düşük değerler alacağı öngörüldüğü için yazma performansı dikkate alınmamıştır.

Test işlemlerinde sunucuya web erişimi için yaygın kullanıma sahip tarayıcılar (Google Chrome, Microsoft Edge, Mozilla Firefox) kullanılmıştır. Simüle edilen her kullanıcı, simüle edilen kullanıcı sayısı artışına bağlı olarak normal dağılıma benzeyen bir yük modeli üretmektedir.

5. ARAŞTIRMA BULGULARI

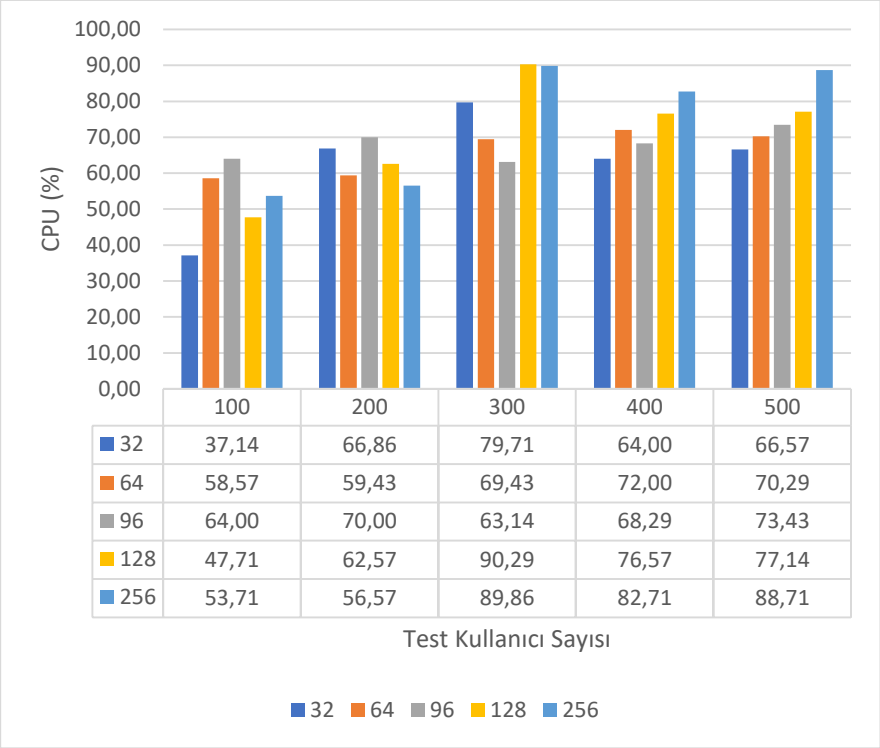
Bugüne kadar yapılan tüm çalışmalar konteynerlerin kaynaklar üzerinde düşük etkisi olduğunu göstermektedir [76]. Çalışmalara göre sunucularda birkaç konteyner uygulamasının çalıştırılması CPU ve bellek performansını neredeyse etkilememektedir. Ancak konteyner çalıştıran sistemlerin yükü arttıkça farklı performans değerleri ortaya çıkmaktadır. Çalışmanın bu bölümünde oluşturulan (RECONLAB) Remote Konteyner Laboratuvarı uygulamasının değişken sistem kaynaklarına sahip makineler üzerinde, değişken kullanıcı yükleriyle sistem kaynaklarına etkisi incelenmiş ve yorumlanmıştır.

5.1. İşlemci (CPU)

Çalışmada değişken kullanıcı sayıları ve değişik bellek miktarı ile Windows sunucuda elde edilen işlemci yükü yüzdeleri Şekil 5.1'de, Ubuntu Linux sunucuda elde edilen işlemci yük yüzdeleri Şekil 5.2'de gösterilmiştir.

	100	200	300	400	500
32	32,14	68,26	76,11	59,40	69,37
64	62,87	67,03	81,43	75,80	73,09
96	70,70	87,20	80,94	84,69	97,03
128	51,61	82,17	98,99	92,57	97,14
256	65,51	74,97	97,36	91,01	97,41

Şekil 5. 1. Kullanıcı sayılarına göre Windows sunucu işlemci yükü



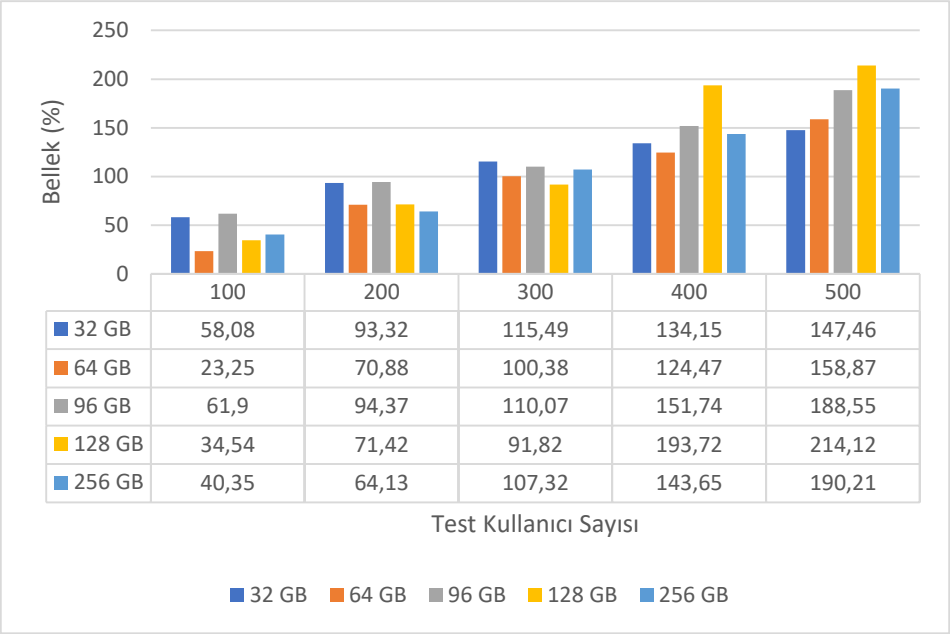
	100	200	300	400	500
32	37,14	66,86	79,71	64,00	66,57
64	58,57	59,43	69,43	72,00	70,29
96	64,00	70,00	63,14	68,29	73,43
128	47,71	62,57	90,29	76,57	77,14
256	53,71	56,57	89,86	82,71	88,71

Şekil 5. 2. Kullanıcı sayılarına göre Linux sunucu işlemci yükü

Şekil 5.1 ve Şekil 5.2'de belirlenen kullanıcı sayılarına göre sunucuların işlemci yük yüzdeleri karşılaştırıldığında, Windows sunucu 100 kullanıcı için %32,14 işlemci yükü oluştururken, Linux sunucu aynı sayıda kullanıcı için %37,14 işlemci yükü oluşturmaktadır. Şekil 5.1 ve Şekil 5.2 karşılaştırıldığında; Şekil 5.1'de işlemci kullanma oranları görülen Windows sunucunun, Şekil 5.2'deki Linux sunucuya göre işlemci kullanma oranının yüksek olduğu görülmektedir. İki farklı işletim sisteminde çalışan aynı programın farklı işlemci kullanım oranlarına sahip olması işletim sistemi mimarisinden kaynaklanmaktadır. Şekil 5.2'e göre geliştirilen sistem Linux sunucuda daha az işlemci yükü ile çalışmaktadır.

5.2. Rastgele Erişimli Bellek (RAM)

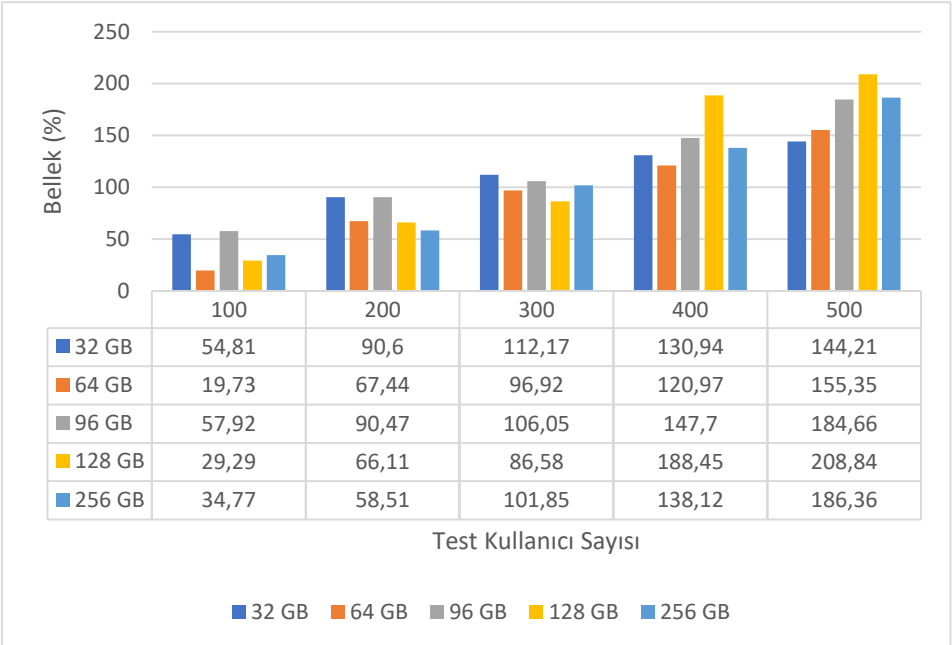
Çalışmada değişken kullanıcı sayıları ve atanan bellek miktarı ile Windows sunucuda elde edilen bellek yükü miktarı Çizelge 5.4'de, Ubuntu Linux sunucuda elde edilen bellek yükü miktarı Çizelge 5.5'de gösterilmiştir.



	100	200	300	400	500
32 GB	58,08	93,32	115,49	134,15	147,46
64 GB	23,25	70,88	100,38	124,47	158,87
96 GB	61,9	94,37	110,07	151,74	188,55
128 GB	34,54	71,42	91,82	193,72	214,12
256 GB	40,35	64,13	107,32	143,65	190,21

Şekil 5. 3. Kullanıcı sayısına göre Windows sunucu bellek yükleri

Bellek yükleri bakımından sunucular karşılaştırıldığında tüm bellek atamalarında Windows sunucunun değişken kullanıcı sayılarına göre daha fazla bellek tükettiği, Linux sunucunun daha az bellek tüketimine sahip olduğu söylenebilir. Şekil 5.3'te 32 GB atanmış bellek ve 100 kullanıcı için testte kullanılan bellek miktarı 58,08 Gb'tır. Bu değerin 32 GB'lık bölümünün bellek olduğu bilinmektedir. Değerin bellekten geriye kalan yaklaşık 22,8 GB'lık bölümü takas alanı olarak diskten kullanılmıştır.



	100	200	300	400	500
32 GB	54,81	90,6	112,17	130,94	144,21
64 GB	19,73	67,44	96,92	120,97	155,35
96 GB	57,92	90,47	106,05	147,7	184,66
128 GB	29,29	66,11	86,58	188,45	208,84
256 GB	34,77	58,51	101,85	138,12	186,36

Şekil 5. 4. Kullanıcı sayısına göre Linux sunucu bellek yükleri

Şekil 5.3 ve Şekil 5.4 incelendiğinde en az takas alanı kullanan test parametrelerine (Kullanıcı sayısı / Atanan Bellek Miktarı) göre uygun seçim gerçekleştirilebilir. Buna göre Şekil 5.4'teki 200 kullanıcı için 64 GB bellek ataması uygun seçim olacaktır.

5.3. Ağ Bant Genişliği

Bant genişliği, ağ bağlantısı bulunan bir cihazın belirli bir zaman diliminde alabileceği veri miktarıdır. Veri aktarım hızı (throughput) ise alınan ve gönderilen verilerin toplamı olarak ifade edilebilir. Çalışmanın bu bölümünde test esnasında elde edilen veriler doğrultusunda değişken yükler altında gösterdiği ağ başarımı değerlendirilecektir.

Çizelge 5. 1. Kullanıcı sayısına göre Windows sunucu ağ yük testi

Atanan Bellek Miktarı	Test Kullanıcı Sayısı				
	100	200	300	400	500
32 GB	175.23 Mb/s	209.94 Mb/s	214.44 Mb/s	188.60 Mb/s	215.40 Mb/s
64 GB	179.09 Mb/s	214.49 Mb/s	221.00 Mb/s	233.54 Mb/s	226.41 Mb/s
96 GB	208.85 Mb/s	214.54 Mb/s	230.16 Mb/s	218.78 Mb/s	196.33 Mb/s
128 GB	200.11 Mb/s	216.80 Mb/s	222.99 Mb/s	223.84 Mb/s	247.98 Mb/s
256 GB	199.69 Mb/s	204.35 Mb/s	223.56 Mb/s	253.67 Mb/s	280.36 Mb/s

Çizelge 5.1 incelendiğinde ağ trafiğinin artan kullanıcı sayısına paralel arttığı görülmektedir. Ancak 32 GB atanan bellek miktarı ile 100 test kullanıcısındaki ağ trafiğinin düşük olduğu görülmüştür. Windows sunucuda yapılan tüm testlerin ortalaması 216,80 Mb/s olduğu görülmüştür. Çizelge 5.2 incelendiğinde ağ trafiği kullanıcı sayısına göre artış göstermektedir. Linux sunucuda yapılan tüm testlerin ortalaması 216,34 Mb/s olarak hesaplanmıştır. Windows ve Linux sunucuların ağ trafiğinin birbirine çok yakın olduğu ifade edilebilir. Ayrıca bu ortalama değerlere göre geliştirilen sistem de kullanılacak tek sunucu için 1 Gigabit/s hızında tek ağ kartının yeterli olacağı söylenebilir.

Çizelge 5. 2. Kullanıcı sayısına göre Linux sunucu ağ yük testi

Atanan Bellek Miktarı	Test Kullanıcı Sayısı				
	100	200	300	400	500
32 GB	161.95 Mb/s	214.65 Mb/s	219.30 Mb/s	193.55 Mb/s	220.35 Mb/s
64 GB	177.25 Mb/s	212.60 Mb/s	219.40 Mb/s	232.00 Mb/s	224.95 Mb/s
96 GB	209.60 Mb/s	215.45 Mb/s	231.20 Mb/s	219.90 Mb/s	197.50 Mb/s
128 GB	199.85 Mb/s	216.70 Mb/s	222.95 Mb/s	223.65 Mb/s	247.65 Mb/s
256 GB	196.85 Mb/s	201.70 Mb/s	221.05 Mb/s	251.10 Mb/s	277.35 Mb/s

5.4. Blok I/O İşlemleri

Çalışılan sistemin I/O sınırlarının bilinmesi yüksek hızlı veri aktarımı için önemlidir. Test ortamı için kullanılan bilgisayarlarda 1 adet 240 GB SSD disk bulunmaktadır. Oluşturulan I/O Blok büyüklüğü 512 KB olup, disk işlemlerinin %90 okuma ve %10 yazma olduğu varsayılmıştır.

Çizelge 5. 3. Kullanıcı sayısına göre Windows sunucu Blok I/O yük testi

Atanan Bellek Miktarı	Test Kullanıcı Sayısı				
	100	200	300	400	500
32 GB	106.43 MB/s	212.81 MB/s	317.96 MB/s	432.21 MB/s	541.25 MB/s
64 GB	105.28 MB/s	211.13 MB/s	315.28 MB/s	427.56 MB/s	538.12 MB/s
96 GB	105.12 MB/s	212.10 MB/s	316.19 MB/s	430.57 MB/s	539.87 MB/s
128 GB	104.75 MB/s	210.63 MB/s	315.48 MB/s	429.43 MB/s	536.41 MB/s
256 GB	104.14 MB/s	210.41 MB/s	315.12 MB/s	427.64 MB/s	533.73 MB/s

Çizelge 5.3'de görülen Windows sunucuya ait disk üzerinden okuma işleminin artan kullanıcıya paralel olarak arttığı görülmüştür. Test işlemleri sırasında atanan bellek boyutunun disk okuma performansı üzerinde etkisinin olmadığı ancak, artan test kullanıcı sayısının etkili olduğu görülmüştür.

Çizelge 5. 4. Kullanıcı sayısına göre Linux sunucu Blok I/O yük testi

Atanan Bellek Miktarı	Test Kullanıcı Sayısı				
	100	200	300	400	500
32 GB	104.95 MB/s	210.21 MB/s	315.45 MB/s	420.63 MB/s	526.11 MB/s
64 GB	100.95 MB/s	207.60 MB/s	311.73 MB/s	415.82 MB/s	520.14 MB/s
96 GB	104.80 MB/s	209.87 MB/s	315.87 MB/s	420.46 MB/s	526.25 MB/s
128 GB	104.87 MB/s	209.73 MB/s	315.32 MB/s	421.47 MB/s	526.23 MB/s
256 GB	103.25 MB/s	209.24 MB/s	313.88 MB/s	419.23 MB/s	524.25 MB/s

Çizelge 5.4'te görülen Linux sunucuya ait disk işlemleri Windows sunucu ile benzerlik göstermektedir. Artan test kullanıcı sayısına paralel olarak okuma işlemi artmaktadır. Ancak tüm testlerde atanan bellek miktarının disk üzerindeki okuma işlemlerinde etkisi düşüktür.

Çalışmanın uzaktan eğitim kurumları ve literatür açısından referans olabilmesi amacıyla Çizelge 5.5'te örnek Windows sunucu için kullanıcı sayılarına göre sahip olunması gereken donanım bileşenleri gösterilmiştir.

Çizelge 5. 5. Geliştirilen sistem için Windows sunucu örneği

Donanım	100 Kullanıcı	200 Kullanıcı	300 Kullanıcı	400 Kullanıcı	500 Kullanıcı
İşlemci	1 cpu (14 core)	2 cpu (14 core)	2 cpu (14 core)	3 cpu (14 core)	4 cpu (14 core)
Bellek	64 GB	96 GB	128 GB	256 GB	256 GB
Disk hızı	106,43 MB/s	212,10 MB/s	315,48 MB/s	427,64 MB/s	533,73 MB/s
Ağ kartı hızı	175,23 Mb/s	214,49 Mb/s	222,99 Mb/s	253,67 Mb/s	280,36 Mb/s

Geliştirilen sistem aynı anda 200 kişilik bir öğrenci grubuyla kullanılmak isteniyorsa, bu çalışma için 14 çekirdekli 2 işlemci, 96 GB bellek, en az 220 MB/s okuma hızına ve yeterli alana sahip disk ve 100 Mb/s hızında 3 adet ağ kartı gereklidir.

6. SONUÇLAR VE TARTIŞMA

Fiziksel laboratuvarlarda uygulamaların bilgisayarlara kurulması, lisans işlemlerinin takibi, güncelleme ve bakımlarının yapılması oldukça zordur. Yeterli sayıda teknik bilgiye sahip personele ihtiyaç vardır. Aynı zamanda programların öğrenciler tarafından kurulması, lisans bedellerinin ödenmesi ve programı çalıştırmak için asgari donanım seviyesine sahip olma zorunluluğu gibi sebeplerde dikkate alındığında yeni teknolojik bir laboratuvara ihtiyaç olduğu aşikârdır. Bu problemlerin çözümü için sanallaştırma teknolojilerinin kullanımı uygun olacaktır. Böylece sanallaştırma sayesinde ortaya çıkabilecek sorunların tek merkezden çözümünü sağlandığı gibi ve izlenmesi de kolaylaşır.

Sanallaştırma teknolojisi günümüzde bulut ortamlarında sunucu sanallaştırma için %92 oranında kullanılmakta olup, önümüzdeki birkaç yılda %5 büyümesi öngörülmektedir. Diğer sanallaştırma yöntemlerinden depolama sanallaştırma %40 ve uygulama sanallaştırma %37 oranında kullanılmakta olup, depolama sanallaştırmanın %12 ve uygulama sanallaştırmanın %17 büyüyeceği tahmin edilmektedir. Bu durum yakın gelecekte sanallaştırma teknolojilerinin aktif olarak artan oranlarda kullanılacağını göstermektedir.

Konteyner tabanlı sanallaştırmanın ise günümüzde kurumlar için küçük ölçekte ve deneysel ortamlarda kullanımının yaygınlaştığı görülmektedir. Bununla birlikte konteyner tabanlı sanallaştırmanın küçük ölçekte ve deneysel amaçlarla kullanıldığı görülmektedir. Yapılan çalışmalara göre konteyner tabanlı uygulamalar geleneksel sanallaştırma uygulamalarına göre iş yükünü azaltıp, performans artışı sağlamakta ve verimliliği artırmaktadır.

Konteyner tabanlı sanallaştırma uygulamaları klasik sanallaştırmaya göre henüz yeni bir teknoloji olduğundan beraberinde güvenlik endişeleri yaratmaktadır. Ancak konteyner teknolojisi sürekli gelişen yapıda olduğu için güvenlik sorunlarına yönelik

çözümler üretilmektedir. Bulut sağlayıcıları kullanıcılara mevcut sanallaştırma çözümlerinin yanında konteyner tabanlı çözümleri de sunmaktadır. Bu durum ise konteyner teknolojisinin yakın geleceğin teknolojisi olacağını göstermektedir. Sanallaştırmaya göre konteyner teknolojisi cpu, bellek ve disk ortamlarını çok daha verimli kullanmaktadır.

Bu teknoloji geleneksel sanallaştırmanın rakibi değildir. Aslında birbirini tamamlayan teknolojilerdir. Günümüzde sanallaştırma teknolojisi çözümleri bulut gibi büyük yapılara hizmet ederken, konteyner teknolojisi daha özel çözümlere odaklanmaktadır. Bilişim alanında ihtiyaç duyulan özel yazılımların geliştirilmesi ve mikro ölçekte hizmet sunulması amacıyla konteyner teknolojisi yaygın olarak kullanılmaktadır. Konteyner teknolojisiyle kolay yönetilebilir ortamlar oluşturmak mümkündür. Konteynerlerin sürüm kontrollerinin kolay olması, yeni sürümlere adaptasyonu kolaylaştırır.

Yapılan çalışmada aşağıdaki sonuçlara ulaşılmıştır:
Sanallaştırma tabanlı uzaktan eğitim sistemi için yazılım laboratuvarı oluşturulmuştur. Sanallaştırma tabanlı yazılım geliştirme laboratuvarı uzaktan eğitim sistemi öğrencileri tarafından kullanılmıştır. Sistemi kullanan öğrencilerin laboratuvar değerlendirmelerinde daha başarılı (%12,89) not aldıkları ve eğitim sürecinden memnun oldukları gözlemlenmiştir. Sanallaştırılmış laboratuvar alt yapısında uzak masaüstü protokolleri kullanılmıştır. Uzak masaüstü protokolünün kullanılabilmesi için geliştirme ortamında birden fazla sunucuya ihtiyaç vardır. Bazı sunucu rolleri aynı sunucuya atanabilirken, bazı roller sadece bir sunucuda bulunmak zorundadır. Sanallaştırma platformlarının sanal laboratuvar olarak kullanılmasının olumsuz yönleri de bulunmaktadır. Uzak masaüstü bağlantı lisans ücretleri, uzak masaüstü bağlantı portlarının bilinen güvenlik problemleri ve sanallaştırma teknolojisinin yüksek bellek tüketimi bu sorunların başında gelmektedir.

Bu sebeple konteyner tabanlı uzaktan eğitim yazılım laboratuvarı oluşturulmuştur. Yazılım laboratuvarında yaygın kullanılan, sürekli güncellenen ve ücretsiz Docker konteynerleri kullanılmıştır. Laboratuvar yazılımı. Net Core 5.0 çerçevesiyle Windows ve Linux işletim sistemlerinde çalışabilecek şekilde programlanmıştır. Konteyner tabanlı laboratuvar sistemine kullanıcılar uzaktan web tarayıcı ile

ulaşmaktadır. Yazılım anlaşılır ve kolay kullanılabilir ara yüze sahiptir. Aynı anda yüzlerce kullanıcı konteyner tabanlı yazılım geliştirme uygulamalarını kullanabilmektedir. Laboratuvar yazılımı Windows ve Linux sunuculara benzer yöntemlerle kurulmaktadır. Linux ortamına kurulum için Linux komut satırı becerileri gerektirmektedir. Windows ortamında kurulum için buna gerek yoktur.

Sürdürülebilirlik açısından, Linux tabanlı sistemler için teknik açıdan yetkin personele daha fazla ihtiyaç duyulmaktadır. Windows tabanlı sistemler için konteynerler daha az bilgi birikimiyle yönetilebilmektedir. Yetkin personel ihtiyacı azalmaktadır. Ulaşılabilirlik açısından Windows tabanlı sistemlerde hatalar için teknik destek almak daha kolaydır. Linux tabanlı sistemler bu durumda yetersiz kalmaktadır. Ancak günümüzde kurulacak sistemden daha önemli olan sistemin sürdürülebilirliği için yetkin personel ihtiyacıdır. Bu bağlamda konteyner teknolojisi ile ilgili işlerin yürütülmesinde yetkin personele ihtiyaç olacaktır.

Yapılan çalışma ile:

- Öğrencilerin kendi bilgisayarlarına kurulum gerektirmemektedir
- Öğrenci bilgisayarının donanım özelliklerinden bağımsız çalışmaktadır
- Bilgisayar, akıllı telefon ve tabletlerle uzaktan erişilebilir
- Kurulum problemi olmayan
- Lisans sorunu olmayan
- Her yerden ve her zaman erişime açık
- Daha az işlemci ve bellek tüketen
- Geliştirilmeye açık
- Kaynak optimizasyonu açısından literatüre katkı sağlayan
- Tüm bilişim ve mühendislik alanlarına hizmet verebilir
- Kullanıcı sayısına göre ölçeklenebilir
- Sürdürülebilir ve ulaşılabilir özellikte

yazılım geliştirme laboratuvarı oluşturulmuştur.

Ayrıca çalışma gerçekleştirilirken dikkat edilmesi gereken bazı durumlar ortaya çıkmıştır. Bu durumlar şöyle ifade edilebilir; Kurumsal veri merkezlerinde kurulum süreçlerinde yavaşlama problemleri yaşanabilmektedir. Bunun yanında geliştirilen

yazılımda ortaya çıkabilecek güvenlik açığı tüm kurumun verilerini tehlikeye atabilmektedir. Bu tarz çalışmaların bulut ortamında yapılması güvenli ancak maliyetlidir. Yapılan çalışmada kurulum ve test aşamaları internete kapalı ortamda gerçekleştirildiği için güvenlik problemi olmamıştır. Sadece test aşamasının belirli bölümlerinde kısa süreliğine sunucuların internet erişimine açılması sağlanmıştır. Yazılım geliştirme aşamasında temiz kod yazımı sağlamak ve programcıya bağımlılıktan kurtulmak için, kod ifadelerinin belirli bölgelerine yazılımla ilgili hatırlatma satırları yazılmalıdır. Bu durum geliştirilen yazılımın sonradan değiştirilebilirliği ve sürdürülebilirliği açısından önemlidir.

Gelecekteki çalışmalarda mevcut konteyner tabanlı uzaktan eğitim yazılım laboratuvarına yazılımlar eklenerek, öğrencilerin haftalık, günlük çalışmaları değerlendirilebilir. Öğrencilere verilen ödev kontrolleri yazılım üzerinden yapılabilir. Pandemi sürecinde uzaktan eğitim gören öğrenci sayısı tüm dünyada olduğu gibi ülkemizde de artmıştır. Bu sebeple uzaktan eğitim altyapısına uygun laboratuvar ortamlarının önemi artmıştır. Benzeri yazılımların tüm alanlarda üretilmesi, kullanılması pandemiyle oluşacak kayıpların önüne geçebilecektir. Geliştirilen sistem üzerinden eğitim alan öğrencilere yönelik yapılacak eğitim faaliyetleri eğitim bilimleri uzmanları ile disiplinler arası bir yaklaşım ile değerlendirilmelidir. Gerçekleştirilen eğitim kurumlarının dışında, şirketler ve kuruluşlar için yazılım geliştirme platformu olarak kullanılabilir. Özellikle düşük bütçeli, yeni kurulan firmalar tarafından kullanımı yanında duruma göre özelleştirilerek kullanılabilir.

KAYNAKLAR

[1] H. GÜLER, Y. ŞAHİNKAYASI, and H. ŞAHİNKAYASI, "İnternet ve mobil teknolojilerin yaygınlaşması: fırsatlar ve sınırlılıklar," *Sos. Bilim. Derg.*, vol. 7, no. 14, pp. 186–207, 2017.

[2] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.," 2015.

[3] İsnet, "Sanallaştırma Teknolojileri Nelerdir?," *isnet.net.tr*, 2020. https://www.isnet.net.tr/BlogIcerik/sanallastirma-teknolojileri-isnet-blog (accessed Jan. 01, 2022).

[4] M. Parlakyiğit, "Sanallaştırma Teknolojileri," 2013. https://www.parlakyigit.net/sanallastirma-teknolojileri/ (accessed Jan. 01, 2022).

[5] A. Doğru, "Sunucu Sanallaştırma ve Uygulama Sanallaştırma Teknolojileri Performans Karşılaştırılması." Maltepe Üniversitesi Fen Bilimleri Enstitüsü, 2019.

[6] G. Işık, U. Gürel, and A. G. Yavuz, "Bulut ortamlarında hipervizör ve konteyner tipi sanallaştırmanın farklı özellikte iş yüklerinin performansına etkisinin değerlendirilmesi," *Uludağ Univ. J. Fac. Eng.*, vol. 25, no. 2, pp. 981–1002, 2020, [Online]. Available: http://acikerisim.uludag.edu.tr/bitstream/11452/12772/1/25_2_23.pdf.

[7] Openstack, "OpenStack Installation Guide for Ubuntu," *openstack.com*, 2016. https://docs.openstack.org/liberty/install-guide-ubuntu/ (accessed Oct. 18, 2021).

[8] X. Wan, X. Guan, T. Wang, G. Bai, and B.-Y. Choi, "Application deployment using Microservice and Docker containers: Framework and optimization," *J. Netw. Comput. Appl.*, vol. 119, pp. 97–109, 2018.

[9] M. K. Hussein, M. H. Mousa, and M. A. Alqarni, "A placement architecture for a container as a service (CaaS) in a cloud environment," *J. Cloud Comput.*, vol. 8, no. 1, p. 7, 2019.

[10] D. L. Lunsford, “Virtualization technologies in information systems education,” *J. Inf. Syst. Educ.*, vol. 20, no. 3, p. 339, 2009.

[11] W. D. Armitage, A. Gaspar, and M. Rideout, “A UML and MLN based approach to implementing a networking laboratory on a scalable Linux cluster,” *J. Comput. Sci. Coll.*, vol. 23, no. 2, pp. 112–119, 2007.

[12] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[13] S. Daya *et al.*, *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2016.

[14] N. Jovanović, A. Zakić, and M. Veinović, “VirtualMeshLab: Virtual laboratory for teaching Wireless Mesh Network,” *Comput. Appl. Eng. Educ.*, vol. 24, no. 4, pp. 567–576, Jul. 2016, doi: https://doi.org/10.1002/cae.21732.

[15] R. Peredo, A. Canales, A. Menchaca, and I. Peredo, “Intelligent Web-based education system for adaptive learning,” *Expert Syst. Appl.*, vol. 38, no. 12, pp. 14690–14702, 2011, doi: https://doi.org/10.1016/j.eswa.2011.05.013.

[16] M. Cabrera *et al.*, “GILABVIR: Virtual Laboratories and Remote Laboratories in engineering: A teaching innovation group of interest,” in *IEEE EDUCON 2010 Conference*, 2010, pp. 1403–1408, doi: 10.1109/EDUCON.2010.5492362.

[17] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, p. 2, 2014.

[18] S. ZHOU, L. LIN, Z. HE, and Y. SHU, “Application of Programming Experiment Platform Based on Docker Container,” *DEStech Trans. Environ. Energy Earth Sci.*, no. peems, pp. 1–6, 2020, doi: 10.12783/dteees/peems2019/33998.

[19] L. Tobarra, A. Robles-Gómez, R. Pastor, R. Hernández, A. Duque, and J. Cano, “Students’ Acceptance and Tracking of a New Container-Based Virtual Laboratory,” *Appl. Sci.*, vol. 10, no. 3, p. 1091, 2020.

[20] W. Wang, T. Wang, and G. Yin, “Container-Based Complex Programming Skills Training Platform,” in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 2019, pp. 1–5, doi: 10.1109/ICSESS47205.2019.9040819.

[21] L. Fletscher, J. F. Botero, N. Gaviria, É. F. Aza, and J. Vergara, “RECoNE: A Remote Environment for Computer Networks Education,” in *2020 IEEE Global Engineering Education Conference (EDUCON)*, 2020, pp. 787–791, doi: 10.1109/EDUCON45650.2020.9125383.

[22] Z. Kozhirbayev and R. O. Sinnott, “A performance comparison of container-based technologies for the Cloud,” *Futur. Gener. Comput. Syst.*, vol. 68, pp. 175–182, 2017, doi: https://doi.org/10.1016/j.future.2016.08.025.

[23] S. Defteri, “Hyper-V Sanallaştırma Teknolojisi,” *ITU BIDB*, 2013. https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/07/hyper-v-sanallaştırma-teknolojisi (accessed May 26, 2020).

[24] CISN, “Sanallaştırma,” *Computing & Information Services Newsletter*, 2011. http://cisn.metu.edu.tr/2011-19/sanal.php (accessed May 26, 2020).

[25] Spiceworks, “The 2020 State of Virtualization Technology,” *spiceworks.com*, 2020. http://www.spiceworks.com/marketing/reports/state-of-virtualization/ (accessed Dec. 15, 2021).

[26] T.-H. Kim, K. Jiang, and M. V. S. Rajput, “Adoption of container-based virtualization in it education,” 2016.

[27] J. Kim, “Learning and Teaching Online During Covid-19: Experiences of Student Teachers in an Early Childhood Education Practicum,” *Int. J. Early Child.*, vol. 52, no. 2, pp. 145–158, 2020, doi: 10.1007/s13158-020-00272-6.

[28] M. Öztürk, “Web Tabanlı Uzaktan Eğitimde Teknolojiye İlişkin Yeni Eğilimler,” *Abant İzzet Baysal Üniversitesi Eğitim Fakültesi Derg.*, vol. 14, no. 1, Jun. 2014, doi: 10.17240/aibuefd.2014.14.1-5000091512.

[29] A. Ergüzen, E. Erdal, M. Ünver, and A. Özcan, “Improving technological infrastructure of distance education through trustworthy platform-independent virtual software application pools,” *Appl. Sci.*, vol. 11, no. 3, pp. 1–17, 2021, doi: 10.3390/app11031214.

[30] S. Buyukgoze, “Kirklareli University Kayali Campus desktop virtualızation,” *Pressacademia*, vol. 3, no. 4, pp. 306–306, Dec. 2016, doi: 10.17261/Pressacademia.2016.354.

[31] J. L. F. Aleman, “Automated Assessment in a Programming Tools Course,” *IEEE Trans. Educ.*, vol. 54, no. 4, pp. 576–581, 2011, doi: 10.1109/TE.2010.2098442.

[32] D. Alexandru, A. Iftene, and D. Gîfu, “Using New Technologies to Learn

Programming Languages," 2019.

[33] S. Zhou, X. Liu, and L. Lin, "Design and implementation of Python teaching platform based on container and jupyter," in *Proceedings of the 2020 International Conference on Computers, Information Processing and Advanced Education*, 2020, pp. 446–450.

[34] Z. Toth, "Virtualized Software Development Infrastructure for Classrooms," in *2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, 2020, pp. 103–108.

[35] Y. Sugino, M. Niimura, K. Okano, and S. Ogata, "Implementation of Programming Environment based on Cloud IDE with Eclipse Che and Docker," *IEICE Tech. Report; IEICE Tech. Rep.*, vol. 119, no. 467, pp. 67–72, 2020.

[36] J. Kousa, "Teaching Container-based DevOps Practices in Higher Education Context," 2020.

[37] B. Curto and V. Moreno, "Robotics in education," *J. Intell. Robot. Syst.*, vol. 81, no. 1, p. 3, 2016.

[38] S. Mehringer and B. Barker, "Using Containers to Create More Interactive Online Training and Education Materials," in *Practice and Experience in Advanced Research Computing*, Jul. 2020, pp. 246–251, doi: 10.1145/3311790.3396641.

[39] S. S. Nair and V. K. J. Jeeven, "A brief overview of metadata formats," *DESIDOC J. Libr. Inf. Technol.*, vol. 24, no. 4, 2004.

[40] J. Kousa, P. Ihantola, A. Hellas, and M. Luukkainen, "Teaching Container-Based DevOps Practices," 2020, pp. 494–502.

[41] R. White and H. Christensen, "ROS and Docker," in *Studies in Computational Intelligence*, vol. 707, Springer, 2017, pp. 285–307.

[42] S. Cousins, "Welcome to ros topics [ros topics]," *IEEE Robot. Autom. Mag.*, vol. 17, no. 1, pp. 13–14, 2010.

[43] E. Guglielmelli, "Research reproducibility and performance evaluation for dependable robots," *IEEE Robot. Autom. Mag.*, vol. 22, no. 3, p. 4, 2015.

[44] C. Hu, W. Dong, Y. Yang, H. Shi, and G. Zhou, "Runtime verification on hierarchical properties of ROS-based robot swarms," *IEEE Trans. Reliab.*, vol. 69, no. 2, pp. 674–689, 2019.

[45] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A Cloud Robotics Platform," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp.

481–493, 2015, doi: 10.1109/TASE.2014.2329556.

[46] P. Ji, H. Chu, J. Chi, and J. Jiang, “A Resource Elastic Scheduling Algorithm of Service Platform for Cloud Robotics,” in *Proceedings of the 32nd Chinese Control and Decision Conference, CCDC 2020*, 2020, pp. 4340–4344, doi: 10.1109/CCDC49329.2020.9164155.

[47] V. Rashitov and M. Ivanou, “Continuous Integration and Continuous Delivery in the Process of Developing Robotic Systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11771 LNCS, pp. 342–348, doi: 10.1007/978-3-030-29852-4_29.

[48] J. A. Carvajal Soto, F. Tavakolizadeh, and D. Gyulai, “An online machine learning framework for early detection of product failures in an Industry 4.0 context,” *Int. J. Comput. Integr. Manuf.*, vol. 32, no. 4–5, pp. 452–465, 2019.

[49] F. LARRINAGA BARRENECHEA *et al.*, “Analysis of Technological Architectures for the New Paradigm of the Industry 4.0,” *Dyna*, vol. 94, no. 1, pp. 267–271, 2019, doi: 10.6036/8837.

[50] J. Mellado and F. Núñez, “Design of an IoT-PLC: A containerized programmable logical controller for the industry 4.0,” *J. Ind. Inf. Integr.*, p. 100250, 2021.

[51] Y. K. Teoh, S. S. Gill, and A. K. Parlikad, “IoT and Fog Computing based Predictive Maintenance Model for Effective Asset Management in Industry 4.0 using Machine Learning,” *IEEE Internet Things J.*, 2021.

[52] M. Waseem, P. Liang, and M. Shahin, “A Systematic Mapping Study on Microservices Architecture in DevOps,” *J. Syst. Softw.*, vol. 170, p. 110798, 2020, doi: https://doi.org/10.1016/j.jss.2020.110798.

[53] S. Lee, J.-Y. Jo, and Y. Kim, “Authentication system for stateless RESTful Web service,” *J. Comput. Methods Sci. Eng.*, vol. 17, no. S1, pp. S21–S34, 2017.

[54] F. Doglio, *Pro REST API Development with Node. js*. Apress, 2015.

[55] M. Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. “ O’Reilly Media, Inc.,” 2011.

[56] İ. S. Arı, “Klasik Yapılardan Konteyner Yapılarına Geçiş,” İstanbul, 2021. [Online]. Available: http://openaccess.maltepe.edu.tr/xmlui/bitstream/handle/20.500.12415/7912/İBRAHİM-SAMED-ARI-BASKI-KOPYA.pdf?sequence=1&isAllowed=y.

[57] M. Friis, “Announcing Docker Enterprise Edition,” *Docker.com*, 2017. https://www.docker.com/blog/docker-enterprise-edition/ (accessed Aug. 07, 2021).

[58] Docker, “Install Docker Engine,” *Docker.com*, 2021. https://docs.docker.com/engine/install/ (accessed Aug. 07, 2021).

[59] Microsoft, “Comparing WSL1 and WSL2,” *microsoft.com*, 2020. https://docs.microsoft.com/en-us/windows/wsl/compare-versions (accessed Aug. 08, 2021).

[60] Docker, “Install Docker Desktop on Windows,” *Docker.com*, 2021. https://docs.docker.com/docker-for-windows/install/ (accessed Aug. 14, 2021).

[61] Docker, “Install Docker Desktop on MAC,” *docs.docker.com*, 2021. https://docs.docker.com/docker-for-mac/install/ (accessed Aug. 08, 2021).

[62] Docker, “Install Docker Engine on Ubuntu,” *docs.docker.com*, 2021. https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script (accessed Aug. 15, 2021).

[63] Globaltechmagazine, “Konteyner teknolojisi nedir?,” *globaltechmagazine.com*, 2021. https://www.globaltechmagazine.com/2021/04/13/konteyner-container-teknolojisi-nedir/ (accessed Jul. 08, 2021).

[64] B. Security, “Docker (Konteyner) Nedir?,” *BGA Security*, 2018. https://www.bgasecurity.com/2018/04/docker-konteyner-nedir-docker-guvenligi-nasil-saglanir/ (accessed Jun. 08, 2021).

[65] G. Şengün, “Docker nedir? Nasıl çalışır? Nerede kullanılır?,” *gokhansengun.com*, 2016. https://gokhansengun.com/docker-nedir-nasil-calisir-nerede-kullanilir/ (accessed Aug. 07, 2021).

[66] M. Takács, “Dockerfile tutorial by example - basics and best practices,” 2018. https://takacsmark.com/dockerfile-tutorial-by-example-dockerfile-best-practices-2018/ (accessed Aug. 18, 2021).

[67] Docker, “Use bridge networks,” *Docker.com*, 2021. https://docs.docker.com/network/bridge/ (accessed Aug. 14, 2021).

[68] Oracle, “Docker Container’lar ve Container Cloud Services,” *Oracle.com*, 2021. https://www.oracle.com/tr/cloud-native/container-registry/what-is-docker/ (accessed Aug. 15, 2021).

[69] Docker, “Container Networking,” *docs.docker.com*, 2021. https://docs.docker.com/config/containers/container-networking/ (accessed

Jun. 08, 2021).

[70] A. GÜRBÜZ, A. Kurt, and M. Özbek, "Yazılım Test Aracı Seçiminde Tuzaklar," *III. Ulus. Yazılım Mühendisliği Sempozyumu, Ankara*, vol. 1, 2007.

[71] S. Giallorenzo, J. Mauro, M. G. Poulsen, and F. Siroky, "Virtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal," *SN Comput. Sci.*, vol. 2, no. 5, pp. 1–20, 2021.

[72] M. Dewar, *Getting Started with D3: Creating Data-Driven Documents*. " O'Reilly Media, Inc.," 2012.

[73] G. Alpaslan and O. Kalıpsız, "Bulut bilişim teknolojisinin yazılım performans testlerinde kullanımı." 2017.

[74] H. T. İnel, "Yazılım Sistemlerinde Yük & Performans Testi -PART 1," *medium.com*, 2019. https://medium.com/trendyol-tech/yazılım-sistemlerinde-yük-performans-testi-part-1-af825ccf7d69 (accessed Aug. 20, 2021).

[75] D. Both, *Using and Administering Linux: Volume 1*. Berkeley, CA: Apress, 2020.

[76] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393, doi: 10.1109/IC2E.2015.74.

EKLER

EK-1. Startup.cs dosyası

```
using System;
using AutoMapper;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Reconlab.Data;
using Reconlab.Data.Abstract;
using Reconlab.Data.Concrate;
using Reconlab.Data.Concrete;
using Reconlab.Helpers;
using Reconlab.Models;
using Reconlab.Services;
using Reconlab.Viewmodels;

namespace Reconlab
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to a
dd services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
```

```
            services.AddHostedService<ContainerCleanupService>();
            services.AddTransient<IUserRepository, UserRepository>();
            services.AddTransient<IDockerImageRepository, DockerImageRe
pository>();
            services.AddTransient<IDockerContainerRepository, DockerCon
tainerRepository>();

            services.AddAutoMapper(typeof(MappingProfiles));
            services.AddDbContext<ReconlabContext>(options => { options
.UseNpgsql(Configuration["ConnectionStrings:DefaultConnection"]); });
            services.AddAuthentication(CookieAuthenticationDefaults.Aut
henticationScheme)
                .AddCookie(cfg =>
                {
                    cfg.SlidingExpiration = true;
                    cfg.Cookie = new CookieBuilder()
                    {
                        Name = "auth",
                        HttpOnly = false,
                        SameSite = SameSiteMode.Strict,
                        SecurePolicy = CookieSecurePolicy.Always
                    };
                    cfg.LoginPath = "/login";
                    cfg.AccessDeniedPath = "/";
                });
            services.AddControllersWithViews().AddRazorRuntimeCompilati
on();
        }

        // This method gets called by the runtime. Use this method to c
onfigure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironm
ent env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to c
hange this for production scenarios, see https://aka.ms/aspnetcore-
hsts.
                app.UseHsts();
            }
```

```
            var cookiePolicyOptions = new CookiePolicyOptions
            {
                MinimumSameSitePolicy = SameSiteMode.Strict,
            };
            app.UseCookiePolicy(cookiePolicyOptions);
            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseRouting();
            app.UseAuthentication();
            app.UseAuthorization();
            UpdateDatabase(app, env);
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }

        public static void UpdateDatabase(IApplicationBuilder app, IWeb
HostEnvironment env)
        {
            using var serviceScope = app.ApplicationServices.GetRequire
dService<IServiceScopeFactory>().CreateScope();
            using var context = serviceScope.ServiceProvider.GetService
<ReconlabContext>();
            if (!context.Users.Any())
            {
                context.Users.Add(new User()
                {
                    FirstName = "Admin",
                    LastName = "Admin",
                    Mail = "admin@reconlab.com",
                    Password = "admin123".Hash(),
                    Role = Role.Admin
                });
                context.SaveChanges();
            }

            if (!context.DockerImages.Any())
            {
                context.DockerImages.Add(new DockerImage()
                {
```

```
                    Name = "codercom/code-server:latest",
                    DisplayName = "Vscode",
                    Params = "-v /home/coder/project -
v $HOME/vscode.config:/home/coder/.config",
                    Port = 8080
                });
                context.SaveChanges();
            }
        }
    }
}
```

EK-2. Program.cs dosyası

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace Reconlab
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder => { webBuilder.Us
eStartup<Startup>(); });
    }
}
```

EK-3. Appsettings.json dosyası

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Host=127.0.0.1; Database=reconlab; Username=r
econlab; Password=reconlab;"
  },
  "ContainerlarDakikaSonraOtomatikSilinsin": 120
}
```

EK-4. Controllers\AuthController.cs dosyası

```
using System;
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
using Microsoft.VisualBasic;
using Reconlab.Data.Abstract;
using Reconlab.Helpers;
using Reconlab.Viewmodels;

namespace Reconlab.Controllers
{
    public class AuthController : BaseController
    {
        private readonly IUserRepository _userRepository;

        public AuthController(IUserRepository userRepository, IMapper m
apper) : base(mapper)
        {
            _userRepository = userRepository;
        }

        // login viewi get metodu
        [HttpGet("login")]
        public IActionResult Login(string ReturnUrl) => View(new LoginV
M(){ReturnUrl = ReturnUrl});

        // login viewi post edildiğinde
        [HttpPost("login")]
        public async Task<IActionResult> Login(LoginVM model)
        {
            // model valid değilse geriye döndür
            if (!ModelState.IsValid)
                return View(model);

            // veri tabanına küçük ve boşluklar silinmiş kayıt ettik. o
 yüzden düzenliyoruz
            model.Mail = model.Mail.Trim().ToLower();
            // parolayı veritabanına şifreli olarak kayıt ettik
            var password = model.Password.Hash();

            // veritabanından kullanıcıyı mail ve şifre ile sorguladık
            var user = await _userRepository.Get(model.Mail, password);
```

```
            // kullanıcı yoksa tempdata içerisinde uyarı ile viewe geri
 döndük
            if (user == null)
            {
                Error("Kullanıcı bulunamadı!");
                return View(model);
            }

            // claimsleri oluşturduk
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Email, user.Mail),
                new Claim(ClaimTypes.Role, user.Role.ToString()),
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()
)
            };

            //12 saat expire süresi ekledik
            var expire = DateTime.UtcNow.AddHours(12);

            // claimsler ile birlikte cookie authentication yapacağımız
ı belirttik
            var claimsIdentity = new ClaimsIdentity(
                claims, CookieAuthenticationDefaults.AuthenticationSche
me);
            var authProperties = new AuthenticationProperties
            {
                ExpiresUtc = expire,
                IssuedUtc = DateTime.UtcNow,
                IsPersistent = true,

            };
            // oturum açtırdık
            await HttpContext.SignInAsync(CookieAuthenticationDefaults.
AuthenticationScheme,
                new ClaimsPrincipal(claimsIdentity),
                authProperties);

            // return url yoksa index home'ye yönlendir
            if (string.IsNullOrEmpty(model.ReturnUrl))
                return RedirectToAction("Index", "Home");
            //return url varsa return urle yönlendir
            return Redirect(model.ReturnUrl);
        }

        [HttpGet]
        public async Task<IActionResult> LogOut()
        {
            // oturumu kapattırdık, logine yönlendirdik
```

```
            await HttpContext.SignOutAsync(
                CookieAuthenticationDefaults.AuthenticationScheme);
            return RedirectToAction("Login");
        }
    }
}
```

EK-5. Controllers\BaseController.cs dosyası

```
using AutoMapper;
using Microsoft.AspNetCore.Mvc;

namespace Reconlab.Controllers
{
    public class BaseController : Controller
    {
        protected readonly IMapper Mapper;

        public BaseController(IMapper mapper)
        {
            Mapper = mapper;
        }

        public void Error(string message) => TempData["error"] = messag
e;
        public void Success(string message) => TempData["success"] = me
ssage;
    }
}
```

EK-6. Controllers\DockerContainerController.cs dosyası

```
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.VisualBasic;
using Reconlab.Data.Abstract;
using Reconlab.Helpers;
using Reconlab.Models;
using Reconlab.Viewmodels;

namespace Reconlab.Controllers
{
    [Authorize(Roles = "Admin,Student")]
    public class DockerContainerController : BaseController
    {
        private readonly IDockerImageRepository _dockerImageRepository;
        private readonly IDockerContainerRepository _dockerContainerRep
ository;

        public DockerContainerController(IDockerImageRepository dockerI
mageRepository, IMapper mapper,
            IDockerContainerRepository dockerContainerRepository) : bas
e(mapper)
        {
            _dockerImageRepository = dockerImageRepository;
            _dockerContainerRepository = dockerContainerRepository;
        }

        [HttpGet]
        public IActionResult List() => View();

        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var containers = await _dockerContainerRepository.GetContai
nersAsync();
            return Json(containers);
        }

        // id varsa image,  id yoksa bos ekleme viewini döner
        [HttpGet]
        public async Task<IActionResult> CreateOrUpdate(int id)
        {
            // id yoksa viewe bos kismi model don
```

```
            if (id == 0)
                return View(new DockerImageCreateUpdateVM());
            // id bos degilse veritabanindan image sorgula
            // image varsa viewmodele automapper yardimi ile aktar
            // viewe yolla
            var image = await _dockerImageRepository.GetImageAsync(id);
            if (image != null)
            {
                var imageVm = Mapper.Map<DockerImageCreateUpdateVM>(ima
ge);
                return View(imageVm);
            }

            Error("Image Bulunamadı!");
            return RedirectToAction("List");
        }

        [HttpPost]
        public async Task<IActionResult> CreateOrUpdate(DockerImageCrea
teUpdateVM image)
        {
            // model valid degilse modeli viewe geri dondur
            if (!ModelState.IsValid)
                return View(image);

            // eger id 0 ise image eklemedir
            if (image.DockerImageId == 0)
            {
                await _dockerImageRepository.AddImageAsync(Mapper.Map<D
ockerImage>(image));
                Success("Image başarı ile eklendi!");
                return RedirectToAction("List");
            }

            // eger id 0'dan buyukse guncellenecek container bul
            var updateImage = await _dockerImageRepository.GetImageAsyn
c(image.DockerImageId);
            if (updateImage == null)
            {
                Error("Güncellenecek image bulunamadı!");
                return View(image);
            }

            Mapper.Map(image, updateImage);
            await _dockerImageRepository.UpdateImageAsync(updateImage);
            Success("Image başarıyla güncellendi!");
            return RedirectToAction("List");
        }
```

```
        [HttpGet]
        public async Task<IActionResult> Stop(int id)
        {
            var container = await _dockerContainerRepository.GetContain
erAsync(id);
            if (container == null)
            {
                Error("Container bulunamadı!");
                return RedirectToAction("List");
            }

            var error = await DockerHelper.StopContainer(container.Name
);
            if (string.IsNullOrWhiteSpace(error) == false)
            {
                container.ErrorMessage += "\n" + error;
                await _dockerContainerRepository.UpdateContainer(contai
ner);
                Error("Container durdurulurken hata meydana geldi!");
                return RedirectToAction("List");
            }

            container.Status = ContainerStatus.Terminated;
            await _dockerContainerRepository.UpdateContainer(container)
;
            Success("Container başarıyla durduruldu!");
            return RedirectToAction("List");
        }

        [HttpGet]
        public async Task<IActionResult> Delete(int id)
        {
            if (id == 0)
            {
                Error("Container Seçiniz");
                return RedirectToAction("List");
            }

            var container = await _dockerContainerRepository.GetContain
erAsync(id);
            if (container == null)
            {
                Error("Container Bulunamadı");
                return RedirectToAction("List");
            }

            await DockerHelper.StopContainer(container.Name);
            await _dockerContainerRepository.DeleteContainer(container)
;
```

```
                Success("Container başarı ile silindi");
                return RedirectToAction("List");
            }
        }
    }
```

EK-7. Controllers\DockerImageController.cs dosyası

```
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Reconlab.Data.Abstract;
using Reconlab.Models;
using Reconlab.Viewmodels;

namespace Reconlab.Controllers
{
    [Authorize(Roles = "Admin")]
    public class DockerImageController : BaseController
    {
        private readonly IDockerImageRepository _dockerImageRepository;
        private readonly IDockerContainerRepository _dockerContainerRep
ository;

        public DockerImageController(IDockerImageRepository dockerImage
Repository, IMapper mapper,
            IDockerContainerRepository dockerContainerRepository) : bas
e(mapper)
        {
            _dockerImageRepository = dockerImageRepository;
            _dockerContainerRepository = dockerContainerRepository;
        }

        [HttpGet]
        public IActionResult List() => View();

        //image json olarak dönen get metodu
        // bootstrap tablo için
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            // veritabanında bulan tüm image aldık
            var images = await _dockerImageRepository.GetImagesAsync();
            // json olarak döndük
            return Json(images);
        }

        // id varsa image,  id yoksa bos ekleme viewini döner
        [HttpGet]
        public async Task<IActionResult> CreateOrUpdate(int id)
        {
            // id yoksa viewe bos kismi model don
            if (id == 0)
                return View(new DockerImageCreateUpdateVM());
```

```
            // id bos degilse veritabanindan image sorgula
            // image varsa viewmodele automapper yardimi ile aktar
            // viewe yolla
            var image = await _dockerImageRepository.GetImageAsync(id);
            if (image != null)
            {
                var imageVm = Mapper.Map<DockerImageCreateUpdateVM>(ima
ge);
                return View(imageVm);
            }

            Error("Image Bulunamadı!");
            return RedirectToAction("List");
        }

        [HttpPost]
        public async Task<IActionResult> CreateOrUpdate(DockerImageCrea
teUpdateVM image)
        {
            // model valid degilse modeli viewe geri dondur
            if (!ModelState.IsValid)
                return View(image);

            // eger id 0 ise image eklemedir
            if (image.DockerImageId == 0)
            {
                await _dockerImageRepository.AddImageAsync(Mapper.Map<D
ockerImage>(image));
                Success("Image başarı ile eklendi!");
                return RedirectToAction("List");
            }

            // eger id 0'dan buyukse guncellenecek image bul
            var updateImage = await _dockerImageRepository.GetImageAsyn
c(image.DockerImageId);
            if (updateImage == null)
            {
                Error("Güncellenecek image bulunamadı!");
                return View(image);
            }

            Mapper.Map(image, updateImage);
            await _dockerImageRepository.UpdateImageAsync(updateImage);
            Success("Image başarıyla güncellendi!");
            return RedirectToAction("List");
        }

        [HttpGet]
        public async Task<IActionResult> Delete(int id)
```

```
        {
            if (id == 0)
            {
                Error("Image Seçiniz");
                return RedirectToAction("List");
            }

            var image = await _dockerImageRepository.GetImageAsync(id);
            if (image == null)
            {
                Error("Image Bulunamadı");
                return RedirectToAction("List");
            }

            var containers = await _dockerContainerRepository.GetImageC
ontainersAsync(image.DockerImageId);
            foreach (var container in containers)
            {
                if (container.Status == ContainerStatus.Running)
                {
                    Error("Bu imajı kullanan, çalışan bir container var
!");
                    return RedirectToAction("List");
                }

                await _dockerContainerRepository.DeleteContainer(contai
ner);
            }

            await _dockerImageRepository.DeleteImage(image);
            Success("Image başarı ile silindi");
            return RedirectToAction("List");
        }
    }
}
```

EK-8. Controllers\HomeController.cs dosyası

```
using System;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Reconlab.Data.Abstract;
using Reconlab.Helpers;
using Reconlab.Models;

namespace Reconlab.Controllers
{
    [Authorize]
    public class HomeController : BaseController
    {
        private readonly IDockerImageRepository _dockerImageRepository;
        private readonly IDockerContainerRepository _dockerContainerRep
ository;

        public HomeController(ILogger<HomeController> logger, IDockerIm
ageRepository dockerImageRepository, IMapper mapper, IDockerContainerRe
pository dockerContainerRepository) : base(mapper)
        {
            _dockerImageRepository = dockerImageRepository;
            _dockerContainerRepository = dockerContainerRepository;
        }

        public async Task<IActionResult> Index()
        {
            var images = await _dockerImageRepository.GetImagesAsync();

            return View(images);
        }

             public async Task<IActionResult> RunContainer(int imageId)
        {
            // id'ye göre image'ı bul
            var image = await _dockerImageRepository.GetImageAsync(imag
eId);
            if (image == null)
            {
                Error("Image bulunamadı");
                return RedirectToAction("Index", "Home");
            }
```

```
            // oluşturulacak containerın ismini oluştur
            var userId = User.Claims.FirstOrDefault(x => x.Type == Clai
mTypes.NameIdentifier)?.Value;
            var containerName = $"user.{userId}-
image.{image.DockerImageId}";

            // container hali hazırda çalışıyorsa
            var container = await _dockerContainerRepository.GetRunning
ContainerAsync(containerName);
            if (container != null)
            {
                // TODO: burada direk return etmek yerine önce containe
r gerçekten çalışıyormu diye bash komutu kontrolü koyulup, çalışmıyorsa
 çalıştırıp ondan sonra return edilmesi daha güzel olabilir
                TempData["containerUrl"] = $"http://{Request.Host.Host}
:{container.Port}";
                // TODO: porta yönlendirme
                return Redirect($"http://{Request.Host.Host}:{container
.Port}"); //View("MyContainer");
            }

            // boş port bul
            var userPort = await _dockerContainerRepository.GetEmptyCon
tainerPortAsync();
            if (userPort == 0)
            {
                Error("Container için uygun port bulunamadı");
                return RedirectToAction("Index", "Home");
            }

            container = new DockerContainer()
            {
                ImageId = image.DockerImageId,
                Name = containerName,
                UserId = Convert.ToInt32(userId),
                Port = userPort,
            };

            await _dockerContainerRepository.AddContainer(container);
            var error = await DockerHelper.RunContainer(image, containe
r);
            // hata varsa
            if (string.IsNullOrWhiteSpace(error) == false)
            {
                Error("Hata: " + error);
                container.Status = ContainerStatus.Error;
                container.ErrorMessage += "\n" + error;
                await _dockerContainerRepository.UpdateContainer(contai
ner);
```

```
                return RedirectToAction("Index", "Home");
            }

            container.Status = ContainerStatus.Running;
            await _dockerContainerRepository.UpdateContainer(container)
;
            TempData["containerUrl"] = $"http://{Request.Host.Host}:{co
ntainer.Port}";

            // TODO: porta yönlendirme
            return Redirect($"http://{Request.Host.Host}:{container.Por
t}"); //View("MyContainer");
        }

    }
}
```

EK-9. Controllers\UserController.cs dosyası

```
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Reconlab.Data.Abstract;
using Reconlab.Helpers;
using Reconlab.Models;
using Reconlab.Viewmodels;

namespace Reconlab.Controllers
{
    [Authorize(Roles = "Admin")]
    public class UserController : BaseController
    {
        private readonly IUserRepository _userRepository;
        private readonly IDockerContainerRepository _dockerContainerRep
ository;

        public UserController(IUserRepository userRepository, IMapper m
apper,
            IDockerContainerRepository dockerContainerRepository) : bas
e(mapper)
        {
            _userRepository = userRepository;
            _dockerContainerRepository = dockerContainerRepository;
        }

        // kullanicilar get viewi
        [HttpGet]
        public IActionResult List() => View();

        //kullanicilari json olarak dönen get metodu
        // bootstrap tablo için
        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            // veritabanında bulan tüm kullanıcıları aldık
            var users = await _userRepository.GetAll();
            // json olarak döndük
            return Json(users);
        }

        // id varsa kullaniciyi,  id yoksa bos ekleme viewini döner
        [HttpGet]
        public async Task<IActionResult> CreateOrUpdate(int id)
        {
            // id yoksa viewe bos kismi model don
```

```
            if (id == 0)
                return View(new UserCreateUpdateVM());
            // id bos degilse veritabanindan kullaniciyi sorgula
            // kullanici varsa viewmodele automapper yardimi ile aktar
            // viewe yolla
            var user = await _userRepository.Get(id);
            if (user != null)
            {
                var userVm = Mapper.Map<UserCreateUpdateVM>(user);
                return View(userVm);
            }
            // kullanici yoksa kullanici bulunamadı hatası dön
            // liste sayfasina yonlendir
            Error("Kullanıcı Bulunamadı!");
            return RedirectToAction("List");
        }


        // kullanici ekleme ve guncelleme post metodu
        [HttpPost]
        public async Task<IActionResult> CreateOrUpdate(UserCreateUpdat
eVM user)
        {
            // model valid degilse modeli viewe geri dondur
            if (!ModelState.IsValid)
                return View(user);

            // veri tabanina emaili kucuk ve bosluklar silinmis ekliyor
uz.
            // bu islemleri uyguluyoruz
            user.Mail = user.Mail.Trim().ToLower();

            // ayni kullanicinin olup olmadigini kontrol ediyoruz
            // eger kullanici id varsa email ile sorgu yapilirken id fa
rkli olanlar ile sorgu yapiliyor
            var checkUser = await _userRepository.Exist(user.Mail, user
.Id);
            if (checkUser)
            {
                // kullanici varsa geriye bu mail adresi ile kullanici
zaten var hatasi donuyor
                Error("Bu Mail Adresi ile Bir Kullanıcı zaten var!");
                return View(user);
            }

            // eger id 0 ise kullanici eklemedir
            if (user.Id == 0)
            {
                // kullanicinin parolasini sifreliyoruz
```

```
                user.Password = user.Password.Hash();
                // veritabanina kullaniciyi ekliyoruz
                await _userRepository.AddUser(Mapper.Map<User>(user));
                // kullanici ekleme basarili uyarisi ekle ve liste view
ine yolla
                Success("Kullanıcı başarı ile eklendi!");
                return RedirectToAction("List");
            }

            // eger id 0'dan buyukse guncellenecek kullaniciyi bul
            var updateUser = await _userRepository.Get(user.Id);
            if (updateUser == null)
            {
                // id ile kullanici yoksa hata mesajını tempdataya ekle
                // viewe modeli geri döndür
                Error("Güncellenecek kullanıcı bulunamadı!");
                return View(user);
            }

            // eğer modelden gelen sifre bos degilse yeni sifreyi sifre
leyerek kayit et,
            // degilse eski sifreyi aynen kullan
            // guncellemede validasyonlar viewmodel icerisinde yapilmak
tadir.
            // sifre bosmu kontrolunden once uzunluk gibi validasyon ku
ralları modelin icersinde kontrol edilmektedir
            user.Password = string.IsNullOrEmpty(user.Password) ? updat
eUser.Password : user.Password.Hash();
            // guncel kullaniciyi, guncellenecek kullaniciya aktariyoru
z
            Mapper.Map(user, updateUser);
            // kullaniciyi kayit ediyoruz
            await _userRepository.SaveChanges();
            // uyari mesaji ekleyip liste viewine gonderiyoruz
            Success("Kullanıcı başarıyla güncellendi!");
            return RedirectToAction("List");
        }

        //kullanici silme
        [HttpGet]
        public async Task<IActionResult> Delete(int id)
        {
            if (id == 0)
            {
                Error("Kullanıcı Seçiniz");
                return RedirectToAction("List");
            }

            var user = await _userRepository.Get(id);
```

```
            if (user == null)
            {
                Error("Kullanıcı Bulunamadı");
                return RedirectToAction("List");
            }

            var containers = await _dockerContainerRepository.GetUserCo
ntainersAsync(user.Id);

            if (containers != null)
            {
                foreach (var container in containers)
                {
                    if (container.Status != ContainerStatus.Running)
                    {
                        await _dockerContainerRepository.DeleteContaine
r(container);
                        continue;
                    }
                    Error("Kullanıcının çalışan bir containeri var!");
                    return RedirectToAction("List");
                }
            }

            await _userRepository.DeleteUser(user);
            Success("Kullanıcı başarı ile silindi");
            return RedirectToAction("List");
        }
    }
}
```

EK-10. Helpers\DockerHelper.cs dosyası

```
using System.Threading.Tasks;
using Reconlab.Models;

namespace Reconlab.Helpers
{
    public static class DockerHelper
    {
        public static async Task<string> RunContainer(DockerImage image
, DockerContainer container)
        {
            var runCommand = $"docker run -d " +
                             $"--restart unless-
stopped " + //sunucu yeniden başlarsa containerları otomatik başlatsın.
                             $"-
p {container.Port}:{image.Port.ToString()} " + // kullanıcıya özel port
 image portuyla mapleniyor
                             $"--
name {container.Name} " + // consoledan müdahale kolay olsun diye anlam
lı isim veriliyor.
                             $"{image.Params} " + // varsa extra parame
treler
                             $"{image.Name}"; // image name
            var error = await runCommand.Bash();
            if (string.IsNullOrWhiteSpace(error))
                // TODO: docker başarılı sonuç dönse bile bazen contain
erın açılması zaman alabiliyor. bu sebeple buraya biraz delay konuldu.
deneme yanılmayla optimum süre hesaplanabilir..
                await Task.Delay(3000);

            return error;
        }

        public static Task<string> StartContainer(string containername)
        {
            var runCommand = $"docker start {containername}";
            var error = runCommand.Bash();
            return error;
        }

        public static Task<string> StopContainer(string containername)
        {
            var runCommand = $"docker rm -f {containername}";
            var error = runCommand.Bash();
            return error;   }   }  }
```

EK-11. Helpers\ShellHelper.cs dosyası

```
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading.Tasks;

namespace Reconlab.Helpers
{
    /*
     * https://loune.net/2017/06/running-shell-bash-commands-in-net-
core/
     */
    public static class ShellHelper
    {
        public static async Task<string> Bash(this string cmd)
        {
            var escapedArgs = cmd.Replace("\"", "\\\"");

            var shell = "";
            var arguments = "";
            if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux) || Ru
ntimeInformation.IsOSPlatform(OSPlatform.OSX))
            {
                shell = "/bin/bash";
                arguments = $"-c \"{escapedArgs}\"";
            }
            else if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows
))
            {
                shell = @"C:\Windows\System32\WindowsPowerShell\v1.0\po
wershell.exe";
                arguments = $"\"{escapedArgs}\"";
            }

            var process = new Process
            {
                StartInfo = new ProcessStartInfo
                {
                    FileName = shell,
                    Arguments = arguments,
                    RedirectStandardError = true,
                    UseShellExecute = false,
                    CreateNoWindow = true,
                }
            };

            process.Start();
            var error = await process.StandardError.ReadToEndAsync();
```

```
            await process.WaitForExitAsync();

            return error;
        }
    }
}
```

EK-12. Helpers\StringHelper.cs dosyası

```
using System.Linq;
using System.Security.Cryptography;
using System.Text;

namespace Reconlab.Helpers
{
    public static class StringHelpers
    {
        public static string Hash(this string metin)
        {
            var textHash = Sha1(metin);
            var saltStart = metin.Length;
            var saltHash = Sha1(Sha1(metin) + Sha1(metin));
            saltHash += Sha1(saltHash);
            var outHash = "";
            if (saltStart > 0 && saltStart < saltHash.Length)
            {
                var textHashStart = textHash.Substring(0, saltStart);
                var textHashEnd = textHash.Substring(saltStart);
                outHash = Sha1(textHashEnd + saltHash + textHashStart);
            }
            else if (saltStart > saltHash.Length - 1)
            {
                outHash = Sha1(textHash + saltHash);
            }
            else
            {
                outHash = Sha1(saltHash + textHash);
            }

            return saltHash + outHash;
        }

        private static string Sha1(string input)
        {
            var hash = (new SHA1Managed()).ComputeHash(Encoding.UTF8.Ge
tBytes(input));
            return string.Join("", hash.Select(b => b.ToString("x2")).T
oArray());
        }

    }
}
```

ÖZGEÇMİŞ

Adı Soyadı : Ahmet ÖZCAN
Yabancı Dil : İngilizce

Eğitim Durumu : (Kurum ve Yıl)
Lisans : Gazi Üniversitesi, 1993
Yüksek Lisans : Kırıkkale Üniversitesi, 2016

Çalıştığı Kurum/Kurumlar ve Yıl/Yıllar : Milli Eğitim Bakanlığı, 1993 - Devam

Yayınları (SCIE) :

1. A. Ergüzen, E. Erdal, M. Ünver, and A. Özcan, "Improving Technological Infrastructure of Distance Education through Trustworthy Platform-Independent Virtual Software Application Pools," Appl. Sci., vol. 11, no. 3, p. 1214, 2021.

Yayınları (Diğer) :

1. Atilla Erguzen, Ahmet Ozcan, Erdal Erdal, "Ecosystem of Virtualization Technologies", International Journal of Trend in Scientific Research and Development (IJTSRD), Volume 5 Issue 1, November-December 2020 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470
2. Atilla Erguzen, Ahmet Ozcan "Container Ecosystem And Docker Technology", International Journal of Trend in Scientific Research and Development (IJTSRD), Volume 6 Issue 1, December 2021, Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470
3. Atilla Ergüzen | Ahmet Özcan "An Overview of Emergency Call Systems Used in Highway Vehicles" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-6 Issue-2, February 2022, pp.75-78, URL: https://www.ijtsrd.com/papers/ijtsrd49155.pdf

4. Ahmet Özcan | Mahmut Ünver | Atilla Ergüzen "A Survey of Convolutional Neural Network Architectures for Deep Learning via Health Images" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-6 | Issue-2, February 2022, pp.79-82, URL: https://www.ijtsrd.com/papers/ijtsrd49156.pdf
5. Ahmet Özcan | Mahmut Ünver | Atilla Ergüzen "Deep Learning Applications and Image Processing" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-6 Issue-2, February 2022, pp.1-5, URL: https://www.ijtsrd.com/papers/ ijtsrd49142.pdf

Konferans Bildirileri :

1. A.ÖZCAN, A.ERGÜZEN, and E.ERDAL, "Düşük Modelli Araçlara Uygun Acil Çağrı Sisteminin Geliştirilmesi" presented at the 1 st ISIDIMT19' International Symposium On Implementations Of Digital Industry 2019, 2019.
2. A. ÖZCAN, E. ERDAL, and A. ERGÜZEN, "Uzaktan Eğitim ve Laboratuvar," presented at the II. Uluslararası Bilimsel ve Mesleki Çalışmalar Kongresi – Mühendislik ve Doğa Bilimleri (BILMES MÜHENDİSLİK 2018) , 2018.
3. A. ÖZCAN, E. ERDAL, and A. ERGÜZEN, "Uzaktan Eğitimde Yeni Yaklaşım Bulut," presented at the I. Uluslararası İleri Araştırmalar Ve Mühendislik Kongresi, 2017.

Araştırma Alanları : Sanallaştırma, Konteyner Teknolojileri, Robot İşletim Sistemi, Bilgisayar Ağları, Uzaktan Eğitim, Yazılım Geliştirme