



## BTÜ-İMEP SONUÇ RAPORU

### ÖĞRENCİNİN

Adı Soyadı	: Fatih ATEŞ
Numarası	: 19360859074
Bölümü	: Bilgisayar Mühendisliği
E-posta adresi	: fatiimates@gmail.com
BTÜ-İMEP akademik danışmanı	: Dr. Öğr. Üyesi Hayri Volkan AGUN
BTÜ-İMEP sektör danışmanı	: Kemal DEMİR
Çalışma dönemi	: 04.10.2021-21.01.2022

### BTÜ-İMEP Kurumunun:

Adı	: Bursa Büyükşehir Belediyesi
Adresi	: Zafer, Ankara Yolu Cd. No:1, 16270 Osmangazi/Bursa
İnternet sitesi	: <a href="https://www.bursa.bel.tr/">https://www.bursa.bel.tr/</a>
Görev alınan birim	: Yazılım Şube



## İÇİNDEKİLER

İÇİNDEKİLER.....	2
ŞEKİLLER DİZİNİ.....	4
ÇİZELGELER DİZİNİ.....	6
TANIMLAR VE KISALTMALAR.....	7
1. GİRİŞ.....	8
2. KURUM TANITIMI.....	10
1.1. Birim tanıtımı.....	11
3. ÇALIŞMALAR.....	12
3.1. Sunucu kurulumu.....	12
3.1.1. Çalışmanın Hedefi.....	12
3.1.2. Yöntem.....	13
3.1.3. Sonuçlar.....	20
3.2. Versiyon Kontrol Sistemi Kurulumu.....	21
3.2.1. Çalışmanın Hedefi.....	21
3.2.2. Yöntem.....	22
3.2.3. Sonuçlar.....	34
3.3. Portainer Kurulumu.....	35
3.3.1. Çalışmanın Hedefi.....	35
3.3.2. Yöntem.....	36
3.3.3. Sonuçlar.....	37
3.4. Moodle.....	38
3.4.1. Çalışmanın Hedefi.....	38
3.4.2. Yöntem.....	39
3.4.3. Sonuçlar.....	56



3.5.	.NET Core Web Servisi .....	57
3.5.1.	Çalışmanın Hedefi.....	57
3.5.2.	Yöntem.....	58
3.5.3.	Sonuçlar .....	61
3.6.	Buildpacks PR.....	62
3.6.1.	Çalışmanın Hedefi.....	62
3.6.2.	Yöntem.....	63
3.6.3.	Sonuçlar .....	67
3.7.	ZSH .....	68
3.7.1.	Çalışmanın Hedefi.....	68
3.7.2.	Yöntem.....	69
3.7.3.	Sonuçlar .....	71
4.	ÇALIŞMA DÖNEMİNİN DEĞERLENDİRMESİ .....	72
5.	KAYNAKLAR.....	74



## ŞEKİLLER DİZİNİ

Şekil 3.1. Proxmox ağ arayüzü.....	15
Şekil 3.2. Proxmox üzerinde bridge oluşturma .....	15
Şekil 3.3. Debian volume kullanım türü.....	17
Şekil 3.4. Debian takaslama alanı seçimi .....	17
Şekil 3.5. Debian Takaslama bölümünü tamamlama .....	18
Şekil 3.6. Debian disk bölümlemesini tamamlama.....	19
Şekil 3.7. GitLab web giriş arayüzü .....	27
Şekil 3.8. Docker imaj listesi.....	28
Şekil 3.9. Web servis uç noktası isteği .....	29
Şekil 3.10. Runner listesi .....	29
Şekil 3.11. Registry web servis uç nokta isteği .....	30
Şekil 3.12. Docker yapılandırma dosyası.....	31
Şekil 3.13. GitLab proje iş hatları.....	33
Şekil 3.14. GitLab proje iş hattı sonucu.....	33
Şekil 3.15. GitLab ile yayınlanan servis uç noktası isteği .....	33
Şekil 3.16. Portainer web arayüzü.....	36
Şekil 3.17. Moodle gerekli paketlerin kurulumu .....	39
Şekil 3.18. Apache2 servisini yeniden başlatma.....	39
Şekil 3.19. Git aracının kurulması .....	39
Şekil 3.20. Veri tabanı oluşturma ve yetkilendirme .....	40
Şekil 3.21. Moodle’ın git üzerinden indirilmesi.....	40
Şekil 3.22. Git branch değiştirme .....	40
Şekil 3.23. Moodle dosyalarını taşıma işlemleri .....	40
Şekil 3.24. Moodle verilerinin saklanacağı dizinlerin oluşturulması.....	41
Şekil 3.25. Moodle’ın dosya izinlerinin değiştirilmesi.....	41
Şekil 3.26. Moodle kurulum arayüzü.....	41
Şekil 3.27. PHP varsayılan yapılandırma dosyasını bulma .....	42
Şekil 3.28. Apache2 servisini yeniden başlatma.....	42
Şekil 3.29. Moodle’ın dosya izinlerinin değiştirilmesi.....	43
Şekil 3.30. Moodle admin arayüzü .....	43



Şekil 3.31. Moodle ayağa kaldırma.....	45
Şekil 3.32. Moodle 85 portu üzerinden erişim.....	46
Şekil 3.33. Moodle ölçeklendirme ve hata.....	46
Şekil 3.34. Moodle port yönlendirmeleri olmadan ayağa kaldırma ve ölçeklendirme.....	47
Şekil 3.35. Moodle admin arayüzü, yönlendirme hatası .....	48
Şekil 3.36. Moodle network hataları .....	49
Şekil 3.37. Bitnami Issue .....	50
Şekil 3.38. Bitnami PR.....	52
Şekil 3.39. Moodle ölçeklendirme .....	53
Şekil 3.40. Moodle load balancer arkasındayken admin arayüzü .....	54
Şekil 3.41. .NET Core Docker imajı oluşturma.....	58
Şekil 3.42. MBS servisinin ayağa kaldırılması .....	58
Şekil 3.43. Servisin çalışabilirliğinin test edilmesi.....	59
Şekil 3.44. Buildpacks ile önerilen builderların listelenmesi .....	59
Şekil 3.45. Buildpacks ile imaj oluşturmak.....	60
Şekil 3.46. Buildpack ile oluşan imajın çalışabilirliğinin test edilmesi.....	60
Şekil 3.47. İmajların listesi.....	61
Şekil 3.48. Buildpacks Issue.....	62
Şekil 3.49. Buildpacks PR.....	67
Şekil 3.50. ZSH görünümü .....	71



## ÇİZELGELER DİZİNİ

Çizelge 3.1. GitLab volume çizelgesi .....	24
Çizelge 3.2. GitLab community yayınlamak için YAML dosyası .....	26
Çizelge 3.3. GitLab community yayınlamak için Ruby yapılandırma dosyası .....	26
Çizelge 3.4. .NET Core Web servisini imajlamak için Dockerfile dosyası .....	28
Çizelge 3.5. Yerel bir registry servisi yayınlamak için YAML dosyası .....	30
Çizelge 3.6. Bir .NET Core servisini 8080 portu üzerinden yayınlamak için YAML dosyası .....	31
Çizelge 3.7. GitLab CI aşamalarında yapılacak olan otomatizasyon için YAML dosyası.....	32
Çizelge 3.8. Portainer'ı 8000 portu üzerinden yayınlamak için kullanılan YAML dosyası.....	36
Çizelge 3.9. Moodle'ı mikroservis mantığıyla yayınlayabilmek için YAML dosyası.....	45
Çizelge 3.10. Load balancer hizmeti sunabilmek için NGINX yapılandırma dosyası.....	47
Çizelge 3.11. Load balancer hizmeti sunabilmek için YAML dosyası .....	48
Çizelge 3.12. Yenilenmiş moodle_configure_wwwroot fonksiyonu .....	51
Çizelge 3.13. Load balancer hizmetini farklı uç noktalardan sunmak için yapılandırma dosyası.....	54
Çizelge 3.14. YAML dosyası içerisinde konteynerlerin kaynaklarını sınırlamak için anahtarlar .....	55
Çizelge 3.15. .NET Core Web servisini imajlamak için Dockerfile dosyası .....	58
Çizelge 3.16. Yeni eklenen createTarball fonksiyonu .....	65
Çizelge 3.17. /tmp dizini altında whiteout katmanı için geçici dizin oluşturma.....	66
Çizelge 3.18. Yenilenen else bloğunun içeriği .....	66



## TANIMLAR VE KISALTMALAR

BTÜ	: Bursa Teknik Üniversitesi
İMEP	: İşletmede Mesleki Eğitim Programı
BBB	: Bursa Büyükşehir Belediyesi
BİDB	: Bilgi İşlem Daire Başkanlığı
YŞM	: Yazılım Şube Müdürlüğü
BM	: Bilgisayar Mühendisliği
YM	: Yazılım Mühendisliği
MBS	: Muhtarlık Bilgi Sistemi



## 1. GİRİŞ

BTÜ-İMEP kapsamında, tam olarak istediğim alanda bir dönem boyunca çalışma fırsatı yakaladım. Okulda öğrendiğimiz birçok teknik bilginin iş hayatına girdiğimizde de karşımıza çıkması çok önemli bir pozitif etken. Genel anlamda Linux ve Docker üzerine yoğunlaştığım 4 ay boyunca hem mesleki gelişim hem de kişisel gelişim anlamında büyük yol kat ettiğime ve çalışma döneminden önceki halim ile şu an aramda büyük farklılıklar olduğunu hissedebiliyorum.

Çalışma dönemim boyunca açık kaynaklı projeler(örn: Moodle, Git) ile ilgilendim. Moodle gibi eğitim platformlarıyla daha önceden de ilgilendiğim için yabancı değildim ve ilgilenmek istediğimi eğitim teknolojileri alanında burada da çalışabilecek bir alan bulabilmenin benim kariyer planlarımla tam anlamıyla kesişen bir yer olduğunu söyleyebilirim.

Git projesinde GitLab tarafından sağlanan bir imaj kullandık ve bu imaj bir monolitik uygulama gibi paketlenmiş bir imaj. İçerisinde PostgreSQL'inden, Redis'ine her şeyi mevcut. Tamamen Docker ile yönetilen bir sistem ancak mikroservis mimarisine aykırı bir sistem. Bu imajı güzel bir arayüze sahip olduğu için ve yazılımcılarımızın kullanımını kolaylaştırması için kullanıyoruz.

Moodle projesi için bakıldığında hem monolitik olarak hem de Dockerize edilmiş bir imaj ile performanslarını, yönetim kolaylığını ve kullanılabilirlik arasından karşılaştırdık ve yine Docker imajı üzerinden çalıştırmaya karar verdik. Başlangıçta direkt olarak Bitnami'nin Dockerize ettiği imajı kullandık ancak ihtiyaçlarımızı karşılamayınca kaynak kodlarında ufak değişiklikler gerçekleştirerek yeni bir imaj ile çalıştırdık ve bir *“load balancer”* arkasına alarak ölçeklenebilir bir sistem elde ettik. En son adım olarak da kaynak sınırlamaları gerçekleştirdik ve şu an hazır halde test edilmeyi bekliyor.

Linux ve Docker eğitimi alanında, kendi odamızda bulunan bir personele çalışma dönemi boyunca eğitim verdim. Bunun yanı sıra odamızda Windows kullanan üç kişinin ikisini Linux tabanlı farklı işletim sistemlerine geçirmeyi başardım. Ek olarak diğer yazılımcıyı da macOS işletim sistemine geçirmeyi başardım. Şu an tüm Dockerize sistemleri Linux üzerinden yönetiliyor ve çok daha verimli çalışabildiklerini ifade ediyorlar. MacOS ve Ubuntu kullanan yazılımcılarımız ise Windows'tan sonra daha rahat geliştirme yapabildiklerini en azından çıkan işletim sistemi bazlı hataların çok daha azaldığını çıksa da giderebildiklerini dile getiriyorlar.





Rapor içerisinde ağırlıklı olarak Linux, Docker ve açık kaynak kodlu projelerin nasıl ayağa kaldırılacağı, nasıl daha verimli kullanılabileceği, neden ölçeklenebilir olması gerektiği gibi sorulara cevaplar bulabilir, karşılaşılan problemlerin nasıl çözüldüğünü öğrenebilir ve açık kaynak bir projede sorunu çözdükten sonra PR atarak topluluklara katkı sağlanması gerektiğini öğrenebilirsiniz.



## 2. KURUM TANITIMI

BBB'nin temel faaliyet alanı birden fazladır. Öncelikle imar, su ve kanalizasyon, ulaşım gibi kentsel alt yapı; çevre ve çevre sağlığı, temizlik ve katı atık; zabıta, itfaiye, acil yardım, kurtarma ve ambulans; şehir içi trafik; defin ve mezarlıklar; ağaçlandırma, park ve yeşil alanlar; konut; kültür ve sanat, turizm ve tanıtım, gençlik ve spor; sosyal hizmet ve yardım, evlendirme, meslek ve beceri kazandırma; ekonomi ve ticaretin geliştirilmesi hizmetlerini yapar veya yaptırır.

Tarihi olarak bakıldığında kökleri Osmanlı Devleti'ne dayanır. Kendi sitesinde de anlatıldığı gibi *“Türkiye’de asıl belediye örgütü 28 Aralık 1857 tarihinde nizamname ile İstanbul’da kurulmuştur. 1867 yılında ise belediye yasası çıkarıldı. Daha sonra Bursa’nın da içinde bulunduğu 3 belediye kuruldu.”* BBB 100 yılı aşkın senedir Bursa ahalisine hizmet vermektedir ve tüm illerimiz ile dayanışma içinde birçok çalışma gerçekleştirmektedir.

Aktif olarak kendi bünyesinde on bini aşkın çalışanı bulunan ve bunun yanı sıra Belediye şirketlerinde de binlerce çalışanı bulunan BBB hizmetlerine azimle devam etmektedir. Kaynaklar kısmından BBB'nin 4 yıllık stratejik planına<sup>[1]</sup> ve idari yapısına<sup>[2]</sup> ulaşabilirsiniz.



### 1.1. Birim tanıtımı

İMEP süresi boyunca YŞM altında görevlerimi gerçekleştirdim. YŞM altında birden fazla proje desteklenmektedir. Bursa’da halkın kullandığı, içeride kendilerinin kullandığı veya yöneticilerin kullandığı onlarca projeyi kullanıcılara sağlamaktadırlar. Birim yeni kurulmuş bir birimdir ve BİDB altında bulunmaktadır. Organizasyon şemasından bahsedilecek olursa YŞM Kemal DEMİR tarafından yönetilmektedir ve YŞM BİDB altında yer almaktadır. Ek olarak BİDB Kemal ALİOĞLU tarafından yönetilmektedir. YŞM’nin amacı Belediyenin ve Belediye Organizasyonlarının temel anlamda yazılım ihtiyacını karşılamaktır.

İş bölümü, çalışanların ilgi alanlarına ve projenin büyüklüğüne göre oluşturulan ekipler aracılığıyla gerçekleştirilmektedir. Ek olarak iş uygulayış biçimlerinde de kesinlikle güncel metodolojiler kullanılmaktadır. Scrum, Kaizen gibi modern teknikler ile optimum şekilde sorunları çözerek iki tarafın(müşteri-üretici) da çalışmalardan mutlu olması sağlanmaktadır. Çalışanları BM mezunu veya YM mezunu ağırlıklı personellerden oluşmaktadır. Birimde onlarca proje destekleniyor ancak bu projeler hakkında dışarıya bahsedilmesi maalesef söz konusu olmamaktadır. Yalnızca benim denk geldiğim MBS projesi hakkında şunları aktarma imkanım bulunmaktadır. Proje, iki adet yazılımcı ile .NET Core ve Flutter kullanılarak geliştirilmektedir. Bunun yanı sıra tüm süreç Scrum ile işlediği için gerçekten sorunsuz ve iki tarafı da mutlu edecek bir sistem ortaya çıkarılmaktadır. Bu konuda yalnızca YŞM’nin fikir ve davranışlarıyla ilerlenmeyip tüm ilgili birimler ile ortak çalışma gerçekleştirerek yazılımın geliştirilmesi sağlanmaktadır.

Ek olarak içeride güncel teknolojilere büyük bir farkındalık var özellikle açık kaynak ve yeni teknolojilere örneğin şu an gözde bir araç olarak Docker’a büyük bir farkındalık var ve bu farkındalık da yaptıkları yeni projelerde kullandıkları teknolojiler ile işlerine de yansıtılmaktadır.



### 3. ÇALIŞMALAR

#### 3.1. Sunucu kurulumu

##### 3.1.1. Çalışmanın Hedefi

BBB'nin temin ettiği sunucu içerisine bir sanallaştırma yazılımı kurulması planlanmaktadır. Bir test sunucusu olacağından dolayı sürekli olarak işletim sistemi kurup silmek yerine sanal makineler oluşturarak onları kaldırıp kurmak daha kolay ve zaman kazancı sağlamaktadır. Ek olarak snapshot alıp direkt olarak taşıyabilmek gibi birçok özelliği de beraberinde getirmektedir. Bunun yanı sıra açık kaynak kodlu bir sanallaştırma aracı da test edilmesi istenmiştir. Bu yazılımın adı Proxmox'tur. Bu görev sonucunda Proxmox kurulumunun gerçekleştirilmesi ve içerisine bir Debian işletim sisteminin kurulması hedeflenmektedir.



### 3.1.2. Yöntem

Tüm işlemler yüksek yetki gerektirdiğinde root kullanıcısı ile gerçekleştirilecektir. Aşağıdaki komut ile root kullanıcısına geçilmesi beklenmektedir.

su -

/etc/hosts dosyasının düzenlenmesi gerekmektedir. Localhost'un aynı şekilde loopback adresini çözmesi sorun oluşturmamaktadır. Ancak makinenin kendi IP adresini yalnızca bir alan adının çözmesi gerekmektedir. Aşağıdakine benzer şekilde olmalıdır.

```
127.0.0.1    localhost
172.16.22.16 proxmox.bbb.com proxmox

::1          localhost ip6-localhost ip6-loopback
ff02::1      ip6-allnodes
ff02::02     ip6-allrouters
```

Yapılandırma düzgün çalışıyorsa aşağıdaki komut girildiğinde geriye makinenin IP adresinin dönmesi beklenmektedir.

```
hostname --ip-adress
```

Eğer ki geriye bilgisayarın IP adresi yerine farklı bir adres dönerse aşağıdaki komut ile host adı değiştirilebilmektedir.

```
hostnamectl set-hostname proxmox.bbb.com
```

Bu ufak yapılandırmadan sonra Proxmox deposunu makinenin depoları arasına eklemek gerekmektedir.

```
echo "deb [arch=amd64] http://download.proxmox.com/debian/pve bullseye pve-no-  
subscription" > /etc/apt/sources.list.d/pve-install-repo.list
```



Proxmox deposunun GPG anahtarının güvenilen anahtarlar arasına eklenmesi beklenmektedir.

```
wget https://enterprise.proxmox.com/debian/proxmox-release-bullseye.gpg -O  
/etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
```

Anahtarı doğrulamak için aşağıdaki komutlar ile sağlaması yapılmalıdır.

```
sha512sum /etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
```

Çıktı kesinlikle aşağıdaki gibi olmalıdır. Aşağıdakinden farklı bir çıktı alınırsa programın Proxmox tarafından şifrelenmediği anlaşılır ve kesinlikle kurulmaması gerekmektedir.

```
7fb03ec8a1675723d2853b84aa4fdb49a46a3bb72b9951361488bfd19b29aab0a789a4f8c740  
6e71a69aabb727c936d3549731c4659ffa1a08f44db8fdcebfa /etc/apt/trusted.gpg.d/proxmox-  
release-bullseye.gpg
```

Sonrasında sistem güncellemeleri alınmalıdır ve güncellemesi olan araçlar yükseltilmelidir. Bu işlem aynı zamanda Proxmox'un son versiyonunu da indirilebilir hale getirecektir.

```
apt update && apt full-upgrade
```

İşlem tamamlandıktan sonra Proxmox aracının kurulumuna başlanması gerekmektedir.

```
apt install proxmox-ve postfix open-iscsi -y
```

Kurulum esnasında bir mail servisi bağlanmayacaksa postfix yapılandırması için gelen ekranda local only seçeneği seçilmeli ve alan adı aynı şekilde bırakılmalıdır. Kurulum sonucunda sistemi yeniden başlatmanız gerekmektedir.

```
reboot
```

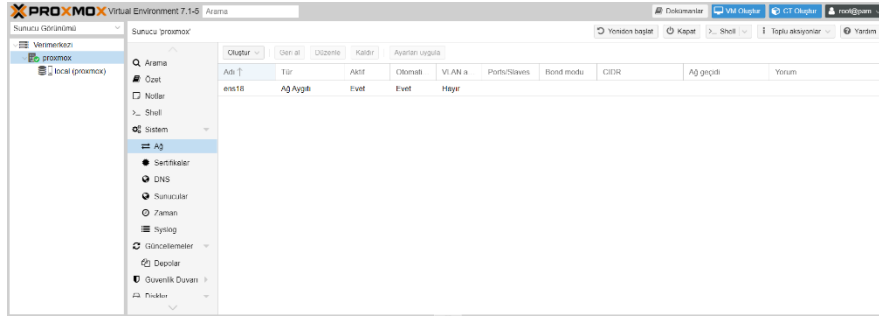
Makine sorunsuz şekilde yeniden başlatıldıktan sonra Proxmox'u dual boot olarak kurmadıysanız sonra aşağıdaki komut yardımı ile os-prober paketinin sistemden silinmesi güvenlik açığına sebep olmamak için tavsiye edilmektedir.

```
apt remove os-prober
```

Daha sonrasında [https://<ip\\_address>:8006](https://<ip_address>:8006) adresinden Proxmox'un web arayüzüne girilmelidir. Varsayılan olarak root kullanıcısının adı ve şifresiyle bir kullanıcı tanımlamaktadır.

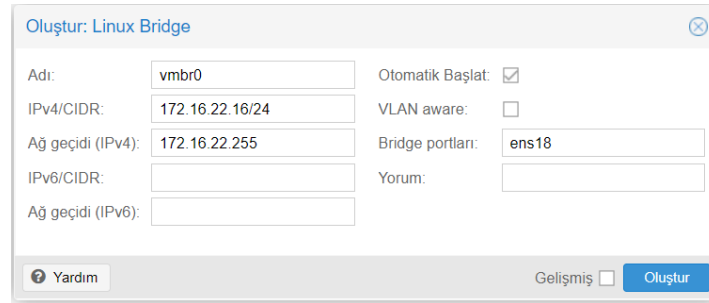
Giriş gerçekleştirdikten sonra ilk iş olarak bir köprü tanımlanmalıdır. Köprü tanımlanmaması sonucunda oluşturulan bilgisayarlar IP adresi alamazlar.

Köprü tanımlayabilmek için Proxmox > Ağ menüsüne girilmesi gerekmektedir.



Şekil 3.1. Proxmox ağ arayüzü

Yukarıdaki yönetim menüsü içerisinde oluştur butonuna tıklanarak aşağıdaki gibi metin kutularının doldurularak bir köprü oluşturulmalıdır. Bridge portları kutusuna varsayılan ethernet kartının adını yazmak gerekmektedir.



Şekil 3.2. Proxmox üzerinde bridge oluşturma

Bridge oluşturduktan sonra makine yeniden başlatılır ve artık makineye yeni atanan IP adresi üzerinden erişilmesi gerekmektedir.



Proxmox içerisinde sağ üst menüden VM oluştur butonuna tıklayarak bir sanal makine oluşturulmalıdır. Oluşturulan sanal makine donanımları kesinlikle kısıtlandırılmış halde olmalıdır. Kurulacak olan Debian işletim sistemi üzerinde GitLab, Registry ve Portainer çalıştırılması planlanmaktadır. Sistem üzerinde ayrılacak donanım için aşağıdaki durumlar göz önünde bulundurulmuştur;

- GitLab kendi dökümantasyonunda 4-6 GB aralığında bir RAM ihtiyacı olacağını belirtmiştir.
- GitLab konteynerinin 2 adet yazılımcı olmak üzere toplamda 5 adet kullanıcısı bulunmakta ve yerel ağda bulunacağı için dışarıdan da istek kabul etmeyecektir. Dolayısıyla aynı anda sunucuya GitLab için bağlanacak kullanıcı sayısı 2 olarak kabul edilebilir.
- Registry sistemi GitLab üzerinden kullanılmaktadır dolayısıyla yine aynı zamanda en fazla 2 kullanıcı kullanacağı düşünülebilir.
- Portainer sistemi yalnızca konteynerleri yönetmek için kullanılmaktadır. Burada da 2 adet görevli olduğu düşünülürse yine aynı anda en fazla 2 kullanıcının istek attığı bir durum ortaya çıkabilir.

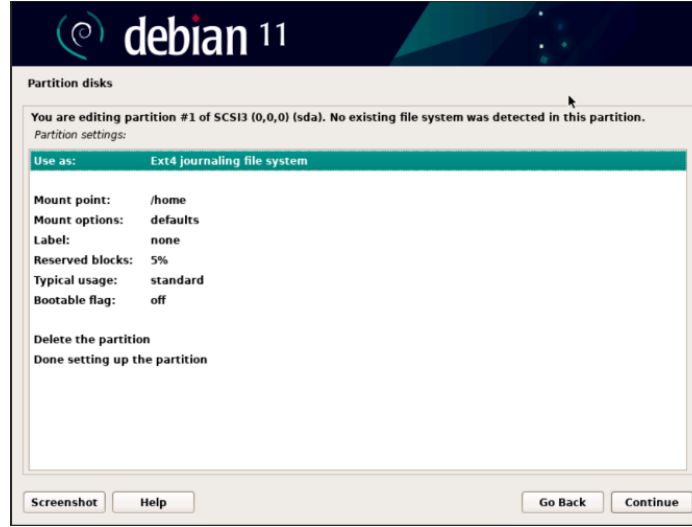
Sistemde program hataları, kilitlemeler, konteyner problemleri gibi durumlar da göz önünde bulundurulduğunda; 8GB RAM, 40GB+100GB bir disk volume alanı ve 4 çekirdekli bir işlemci ayırmak yeterli gelecektir ve hatta sistemin çoğunlukla 5-6 GB RAM ve %10 CPU dolaylarında seyretnesi beklenmektedir.

Debian kurulumu tipik bir işletim sistemi kurulumuyla eşdeğerdir. Linux'a özel olarak bazı izin yapılandırılmaları gerçekleştirilmelidir. Bu içerik kapsamında yalnızca özel olarak izinler işlenmiştir.

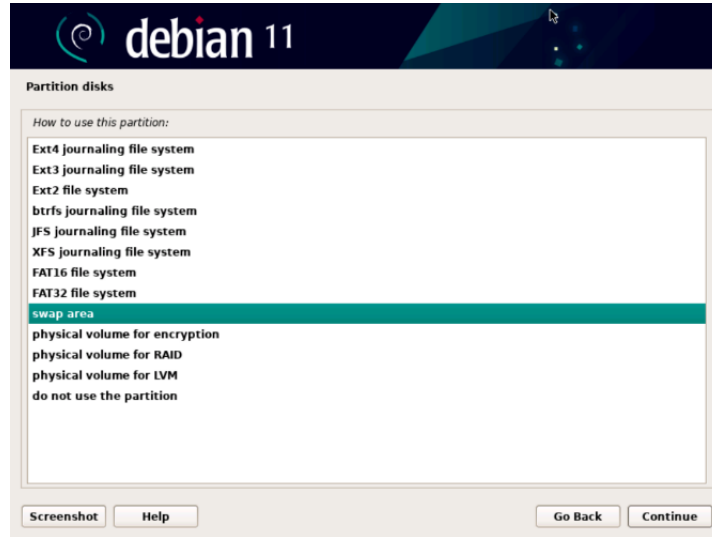
Birçok işletim sisteminde olduğu gibi Linux işletim sistemlerinde de takaslama alanları ayarlanabilmektedir. Swap alanları genellikle ram boyutu veya ram boyutunun yarısı kadar ayrılmaktadır. Bu alanı işletim sisteminin kendisi ayarlamasını istenirse ayarlamadan geçilebilmektedir. Ancak tavsiye edilen takaslama alanını da ayrı bir disk bölümü olarak atamak ve ana disk bölümünden bağımsız bir takaslama alanı oluşturmaktır. Bu size tamamen adanmış bir takaslama alanı oluşturmanıza olanak sağlamaktadır. Dolayısıyla diskiniz taşmış olsa dahi takaslama alanı ile ilgili bir sorun yaşamamanızı önlemektedir.



Bir takaslama alanı oluşturmak için öncelikle boş disk alanı seçilmeli ve 1 GB “Primary” türünde yer ayrılmalıdır. Bölümün başlangıç noktası olarak boş alanın başlangıcı veya sonu seçilebilir. Daha sonrasında bölüm düzenleme kısmında “Use as:” niteliği “swap area” olarak değiştirilmelidir.

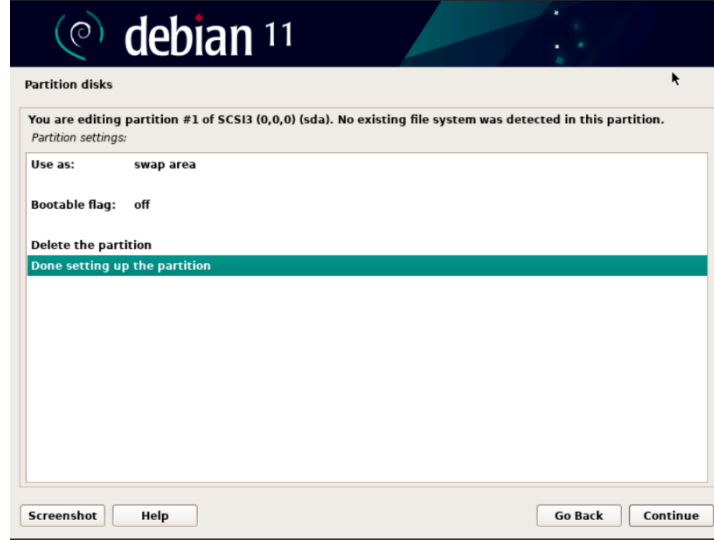


Şekil 3.3. Debian volume kullanım türü



Şekil 3.4. Debian takaslama alanı seçimi

İleri butonuyla devam edilir ve “Done setting up the partition” seçeneği seçilerek takaslama alanı oluşturma işlemi tamamlanmaktadır.



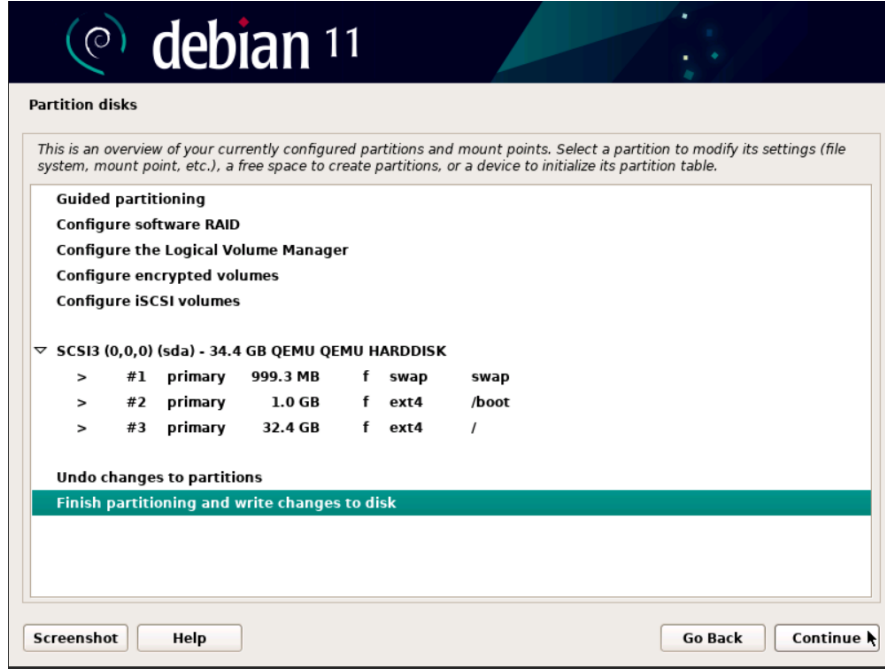
Şekil 3.5. Debian Takaslama bölümünü tamamlama

Linux tabanlı işletim sistemlerinde /boot dizini altında sistemin önyükleyici dosyaları bulunmaktadır. Boot dizininin ayrı bir disk bölümü olarak mount etmek kullanıcıya esneklik kazandırmaktadır. En temel kazandıracağı özellik ise ana disk bölümüne bir hasar geldiğinde boot disk bölümü ayrı bir bölümde duracağı için cihazın nasıl başlayacağını yönlendirilebilmektedir.

Boot disk bölümünü ayrı bir bölüm olarak oluşturmak için öncelikle boş disk alanı seçilmeli ve 1 GB “Primary” türünde yer ayrılması gerekmektedir. Bölümün başlangıç noktası olarak boş alanın başlangıcı veya sonu seçilebilmektedir. Daha sonrasında bölüm düzenleme kısmında “Use as:” niteliği “Ext4” olarak bırakılmalıdır.

Bölüm oluşturma işlemine “mount point” niteliğini “/boot” dizini olarak ayarlayarak devam edilir. Son olarak ise “Done setting up the partition” seçeneği seçilerek takaslama alanı oluşturma işlemi tamamlanmaktadır.

Takaslama ve Boot bölümleri oluşturulduktan sonra diskiniz yaklaşık olarak aşağıdaki gibi bölümlenmiş olmalıdır.



Şekil 3.6. Debian disk bölümlemesini tamamlama

En kritik yer olan disk bölümlemesinden sonraki işlemler tipik bir işletim sistemi kurumuyla aynıdır ve grafik arayüzünden devam etmektedir.



### 3.1.3. Sonuçlar

Proxmox İMEP dönemi boyunca sorunsuz bir şekilde test sunucumuz üzerinde çalışmaktadır. Yedekleme disk silme genişletme gibi birçok işlemde başarılı olmuştur. Kendi adıma açık kaynak bir program olarak Docker'dan sonra en beğendiğim araç olduğunu söyleyebilirim. Yüksek kullanılabilirlik açısından bakılacak olursa Docker'dan daha iyi çalıştığını rahatlıkla söyleyebilirim. Çünkü Docker tarafında imajlar son kullanıcıyla sürekli bir şekilde etkileşimlerde, ancak burada işletim sistemleri zaten sabitken ve aracı kullanan kişi de aracı tanıyorken “99.9% high availability” düşüncesini elektrik kesintileri dahil edilmezse Proxmox ile gerçekleştirebildik. Ve tüm bunları gerçekleştirirken bir taşması veya diskte gereksiz dosyaların birikmesi veya arada bir yeniden başlatılması gibi diğer türevi araçlarda karşılaşılan sorunlar ile karşılaşmadık.



## 3.2. Versiyon Kontrol Sistemi Kurulumu

### 3.2.1. Çalışmanın Hedefi

BBB bünyesinde daha önce geliştirilen uygulamalar için bir sürüm kontrol sistemi kullanılmamaktadır. Uzak bir sunucuda kodları barındırmak kurum açısından büyük bir güvenlik açığı oluşturmaktadır. Yerel bir sunucuda bir sürüm kontrol sistemi yapılandırmak ise hem sunucu maliyeti hem de çalışan maliyeti getirmektedir. Bunun yanı sıra tüm kaynakların bir sunucuda depolanması yine bir güvenlik açığı ortaya çıkarmaktadır. Kaynakların tutulacağı bir sunucuda güvenlik önlemleri hat safhada olması gerekmektedir. Dünya üzerinde sürüm kontrol sistemlerinin bu kadar yaygın olması rağmen kurumlar içerisinde kullanılmamasına sıklıkla rastlanmaktadır.

Git aracı monolitik olarak kurmak istenilirse de kendi dökümantasyonu<sup>[3]</sup> kurmayı anlatacaktır. Ancak konteyner imajlarıyla bu işlemler daha kolay hale getiriliyor, verilerinizi bir araya toplama imkânı sunuyor ve tek bir dosya ile sanal ortamınızı yönetebilmenize olanak sağlıyor. Bu görevin temel hedefi bir Git sunucusu ayağa kaldırmaktır. Başarılı sağlanırsa bir web arayüzü ile yazılımcıların kullanabileceği bir açık kaynak proje ile hedef genişletilmektedir.



### 3.2.2. Yöntem

Git'in monolitik olarak kurulması gayet basit ve anlaşılır bir şekilde dökümantasyonunda anlatılmıştır. Öncelikle git adında bir kullanıcı oluşturulmalıdır. Ve SSH keylerini yönetebilmesi için kendi içerisinde .ssh dizini ve içerisinde authorized\_keys dosyası oluşturulmalıdır.

```
sudo adduser git  
su git  
cd ~  
mkdir .ssh && chmod 700 .ssh  
touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

Daha önceden oluşturulmuş geliştiricilerin SSH keyleri aşağıdaki komut aracılığıyla servera eklenebilmektedir.

```
cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
```

Daha sonra server tarafında boş bir repository oluşturmak için aşağıdaki komutlar kullanılabilir.

```
cd /srv/git  
mkdir project.git  
cd project.git  
git init --bare
```

Client/Geliştirici tarafında yerel olarak yukarıda oluşturulan uzak noktayı baz alan bir depo aşağıdaki komutlar aracılığıyla oluşturulabilir.

```
cd myproject  
git init  
git add .  
git commit -m 'Initial commit'  
git remote add origin git@gitserver:/srv/git/project.git  
git push origin master
```



Veya direkt olarak proje aşağıdaki komutlar aracılığıyla çekilebilir ve sunucu ile eşitleme gerçekleştirilebilmektedir.

```
git clone git@gitserver:/srv/git/project.git
$ cd project
$ vim README
$ git commit -am 'Fix for README file'
$ git push origin master
```

Görüldüğü üzere kurulum oldukça kolay, ancak şu ana kadar anlatılan kısım yalnızca SSH üzerinden bağlantı kurmanızı sağlamaktadır. HTTP üzerinden bağlantı kurulmak istenirse dökümantasyon<sup>[4]</sup> üzerinden protokoller için çeşitli yapılandırmalar mevcuttur.

Ancak bu şekilde bir web arayüzüne sahip değiliz ve yönetim biraz zor anlaşılabilir. Dolayısıyla GitLab CE imajı kullanılarak bir web arayüzü de elde edilebilir.

Git tek başına kurulduğunda kabiliyetlerine göre çok hafif bir sistemdir. Projenin yönetiminin kolaylaştırılması açısından bir web arayüzü olan bir sistem tercih edilmiştir. Bunun için çeşitli alternatifler mevcuttur ancak GitLab hem Git tabanını kullanarak tabanında kararlı bir sistem oluşturmuştur, hem açık kaynak ve ücretsiz olmasıyla maliyetsizdir ve daha öncesinde de kullanıp test ettiğim stabil çalışan sistemlere sahiptir. Son olarak da Dünya üzerinde ülkeler çapında kullanıcı sahipliği ve Nasdaq üzerinde listelenmesiyle birlikte çok daha güvenilir bir hale gelmiştir.

Her açık kaynak yazılımın olduğu gibi GitLab CE projesi de kapsamlı bir dokümantasyona sahiptir. Yapılandırma işlemlerinin tümünü dökümantasyon<sup>[5]</sup> üzerinden okuma yaparak gerçekleştirmemiz gerekmektedir.

Öncelikle Docker'ın ayakta olup olmadığını kontrol etmek gerekmektedir.

```
docker version
```

Docker'ın ayakta olduğuna kanaat getirdikten sonra daha önce oluşturduğumuz Debian 11 sunucumuza SSH aracılığıyla giriş yapmamız gerekmektedir. Root kullanıcısıyla işlemler yapmak geri dönülemez sonuçlara yol açabilmektedir, dolayısıyla düşük yetkili bir kullanıcıyla aşağıdaki işlemleri gerçekleştirilmesi tavsiye edilmektedir!

```
ssh -p 2222 fatih@172.16.22.28
```

Giriş başarıyla yapıldıktan sonra oluşturulacak dosyalarının düzenli bir halde tutulabilmesi için home dizininizde assets/gitlab adında bir klasör oluşturulması gerekmektedir.

```
mkdir gitlab
```

Daha sonra volume patikalarını tanımlayabilmek için aşağıdaki şekilde ortam değişkenini tanımlamamız gerekmektedir.

```
export GITLAB_HOME=/srv/gitlab
```

GITLAB\_HOME dizinini konteynerlere volume olarak verdiğinizde aşağıdaki gibi dizin ağacı oluşturur.

\$GITLAB_HOME/data	/var/opt/gitlab	Tüm uygulama verileri burada depolanır. Git sisteminde tutulan kaynaklar dahil.
\$GITLAB_HOME/logs	/var/log/gitlab	Tüm uygulama günlükleri burada depolanır.
\$GITLAB_HOME/config	/etc/gitlab	GitLab'i yapılandırırken ayarladığınız tüm ayar dosyalarınız burada bulunur.

Çizelge 3.1. GitLab volume çizelgesi





GitLab CE için varsayılan olarak bir root hesabı oluşturulmaktadır. Bu root hesabının şifresi dışarıdan verilmediği takdirde kendi içerisinde rastgele bir şifre oluşturarak /etc/gitlab/initial\_root\_password dosyasına kaydetmektedir. Bunu yapılandırmak için öncelikle bir Docker Secret oluşturulması gerekmektedir.

```
echo "123" | docker secret create gitlab_root_password -
```

Secret başarıyla oluşturulduysa aşağıdaki komut ile Docker üzerinden şifrelenmiş halini görerek onaylanmalıdır.

```
docker secret ls
```

Artık bir YAML dosyası oluşturulmalıdır. YAML dosyası içerisinde düşünebileceğiniz tüm Swarm yapılandırmalarının bulunması gerekmektedir.

YAML dosyasına sürüm tanımı ile başlamak gerekmektedir. Şu an en son sürüm olarak bulunan 3.6 sürümü tavsiye edilmektedir. Ve daha sonrasında servis tanımları ile devam edilmektedir. BBB bünyesinde test için iki adet servise ihtiyacımız bulunmaktadır. Birincisi GitLab CE, ikincisi ise GitLab Runner servisleridir. Bu servislerden GitLab CE bize bir Git ortamı ve web arayüzü sunmakla görevlenecektir. GitLab Runner ise GitLab üzerinde çeşitli CI/CD işlemlerini gerçekleştirmenize olanak tanımaktadır.

GitLab CE web arayüzüne yerel ağdaki herkesin erişebilmesi için external url değeri yapılandırılmalıdır. Eğer ki bu değere bir domain adı girilirse yerel ağdaki bilgisayarlar bu domaini çözebilmek için bir çeviriciye ihtiyaç duymak zorunda kalmaktadırlar. Bu çevirici bir nameserver, DNS sunucu veya /etc/hosts dosyası olabilmektedir. Ancak bu tarzda bir aracıyla vakit kaybetmemek için buraya makinenin IP adresinin girilmesi gerekmektedir.

Yukarıda oluşturulan gitlab\_root\_password secret nesnesi gitlab.rb dosyası içerisinde yapılandırılan initial\_root\_password değişkenine dosya üzerinden okutularak atanmalıdır. Docker servisi oluşturulduğunda verilen secret nesneleri /run/secrets/<secret\_adı> dizinine mount edilir ve kullanılmak istenirse oradan okunmalıdır.



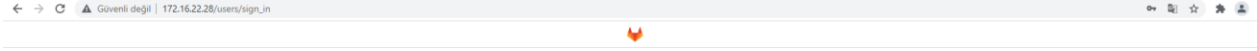
```
version: "3.6"
services:
  gitlab:
    image: gitlab/gitlab-ce:latest
    ports:
      - "22:22"
      - "80:80"
      - "443:443"
    volumes:
      - $GITLAB_HOME/data:/var/opt/gitlab
      - $GITLAB_HOME/logs:/var/log/gitlab
      - $GITLAB_HOME/config:/etc/gitlab
    environment:
      GITLAB_OMNIBUS_CONFIG: "from_file('/omnibus_config.rb')"
    configs:
      - source: gitlab
        target: /omnibus_config.rb
    secrets:
      - gitlab_root_password
  gitlab-runner:
    image: gitlab/gitlab-runner:alpine
  configs:
    gitlab:
      file: ./gitlab.rb
  secrets:
    gitlab_root_password
```

Çizelge 3.2. GitLab community yayınlamak için YAML dosyası

```
external_url 'http://172.16.22.28'
gitlab_rails['initial_root_password'] = File.read('/run/secrets/git-
lab_root_password')
```

Çizelge 3.3. GitLab community yayınlamak için Ruby yapılandırma dosyası

Artık YAML dosyasını Docker Swarm'a vermemiz gerekmektedir. Docker Swarm içerisindeki yapılandırmaları uygulayarak sonuçta iki adet örnek üretecek ve bunları koşturacaktır. Yapılandırmalar sorunsuz çalışıyorsa <http://172.16.22.28> adresinde GitLab web arayüzü bulunmaktadır.



Şekil 3.7. GitLab web giriş arayüzü

Depo oluşturma işlemleri klasikleşmiş Git servisi sağlayıcıları ile aynı adımlara sahiptir. GitLab CI özelliklerini kullanabilmek için bir YAML dosyası ve yeni commit gönderildiğinde tetiklenecek bir Runner yapılandırılmalıdır.

.NET Core ortamında geliştirilecek olan RESTful API servisini yayınlayabilmek için öncelikle bir Docker imajı oluşturulmalıdır. Docker imajı oluşturma yolu da Dockerfile oluşturmaktan geçmektedir.

Servisimizin tamamen boş bir Ubuntu üzerinde de çalıştırılabilir. Ancak Microsoft kendi SDK paketini içeren bir imaj yayınlamış ve güvenlik açığı gibi sorunları da kapatmıştır. Microsoft'un yayınladığı imajı temel olarak almak doğru olan yaklaşımdır.

Bir .NET Core servisi yayınlamak için temel gereksinimler şunlardır;

- Kullanılacak portların dışarı açılması(örn: 80,443)
- .NET Core SDK ile derlenmiş projenin bileşenleri

Bir imaj oluşturulurken temel olarak iki yol izlenmektedir. Birincisi uygulamayı kendi makinenizde derleyerek konteyner içerisine kopyalayarak paketlemektir. İkincisi ise imaj oluşturma esnasında bir konteyner içerisinde derleyerek o konteynerden yeni bir imaj oluşturmaktır. BBB bünyesinde ikinci tip imaj oluşturma şekli best-practice olarak kullanılmaktadır.

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["FlutterAPI/FlutterAPI.csproj", "FlutterAPI/"]
RUN dotnet restore "FlutterAPI/FlutterAPI.csproj"
COPY . .
WORKDIR "/src/FlutterAPI"
RUN dotnet build "FlutterAPI.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "FlutterAPI.csproj" -c Release -o /app/publish

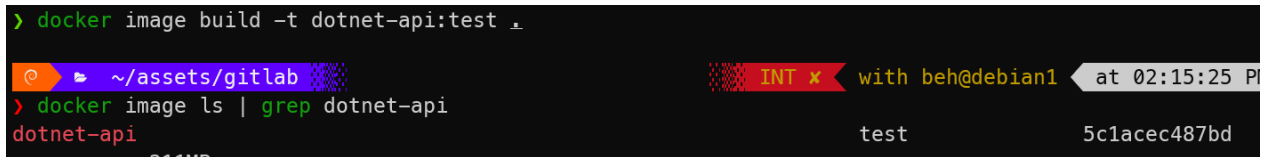
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "FlutterAPI.dll"]
```

Çizelge 3.4. .NET Core Web servisini imajlamak için Dockerfile dosyası

Dockerfile yapılandırması tamamlandıktan sonra artık imaj oluşturulmalı ve imajın güvenilirliği test edilmelidir. Kaynak kodlarının aynı bağlam içerisinde olduğu bir ortamda alttaki imaj alttaki komut yardımıyla oluşturulmalıdır.

```
docker image build -t dotnet-api:test .
```

İmaj oluşturulduktan sonra *docker image ls* komutuyla listelenerek sağlanması gerçekleştirilmelidir.

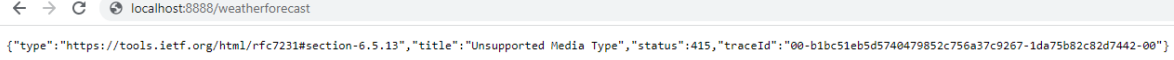


```
> docker image build -t dotnet-api:test .
@ ~/assets/gitlab
> docker image ls | grep dotnet-api
dotnet-api          test                5c1acec487bd
```

Şekil 3.8. Docker imaj listesi

İmajdan aşağıdaki komut yardımıyla bir konteyner oluşturarak konteynerin güvenilirliği test edilmelidir.

```
docker run -p 8888:80 -d dotnet-api:test
```



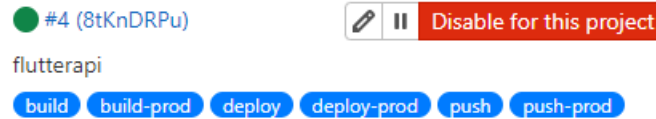
Şekil 3.9. Web servis uç noktası isteği

Yukarıdaki uç noktadan benzer bir çıktıyı elde ettikten sonra GitLab üzerinde bir *specific runner* oluşturulmalıdır. Runner oluşturmak için depo içerisinde Ayarlar > CI/CD > Runner sekmesine gelinir ve aşağıdaki gibi ekran ile karşılaşılmalıdır. Buradan sonra gerekli bilgileri alıntılanarak *gitlab-runner* konteyneri içerisine girerek bir Runner oluşturulmalıdır.

```
docker exec -it <konteyner_adi> gitlab-runner register
```

Yukarıda komut sonucunda istenen bilgiler doldurulur ve neticede bir Runner oluşturulur ve artık Ayarlar > CI/CD > Runner sekmesi altında listelenir ve arayüzden kontrol edilebilmektedir.

### Available specific runners



Şekil 3.10. Runner listesi

Runner ayarlamaları tamamlanmıştır. Artık yeni bir commit atıldığında veya elle tetiklendiğinde otomatik olarak YAML dosyası içeriğinde işlemleri otomatize bir şekilde gerçekleştirecektir.

Bir uygulamayı otomatize etmenin üç aşaması vardır;

- Build aşaması
- Test aşaması
- Deploy aşaması

Şu an için burada bir test aşaması bulunmayacaktır. Yani otomatizasyon işlemlerimiz iki aşamalıdır.

## Yerel Registry Servisi

Otomatize edilmiş bir sistemde imajların registry ortamında saklanması ve dağıtım esnasında registry üzerinden kullanılmalıdır. Bunun için online ortamlarda herkese açık olarak imajlarınızı tutan registry hizmetleri mevcuttur. Ancak açık kaynak bir proje geliştirilmiyorsa ve sistemin gizliliği esas ise imajların yerel registry hizmetlerinde tutulması gerekmektedir.

Yerel bir registry servisi oluşturmak için yaygın olarak Docker'ın yayınladığı registry imajı kullanılmaktadır. Registry oluşturmak için mount noktaları belirlenmelidir. BBB içerisinde test ortamlarında tüm servisler /srv dizini altındadır. /srv/registry dizini ise yerel registry hizmetinin servisi olarak kullanılacaktır. Bir test servisi olarak yayınlanacağı için docker-compose ile yayınlamak yeterlidir. Yine bir test ortamında bulunacağı için hatalar alınırsa dahi sürekli açık kalması kullanılabilirlik açısından geliştiricilere kolaylık sağlamaktadır.

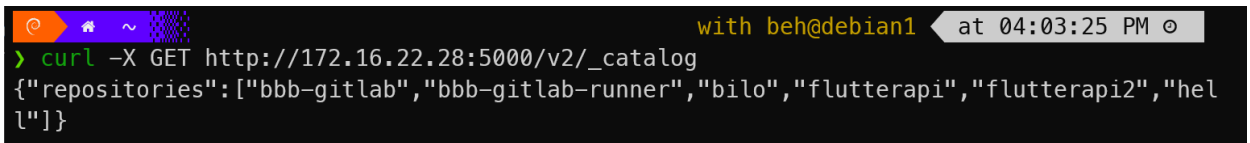
```
version: '3.6'

services:
  web:
    image: 'registry'
    restart: always
    ports:
      - '5000:5000'
    volumes:
      - /registry:/var/lib/registry
```

Çizelge 3.5. Yerel bir registry servisi yayınlamak için YAML dosyası

## Sürekli Entegrasyonun sağlanması

Registry hizmeti başarıyla yayınlandıktan sonra aşağıdaki komut ile uç noktaya istek atarak bulunan imajlara erişim sağlanabilmelidir.



```
with beh@debian1 at 04:03:25 PM
> curl -X GET http://172.16.22.28:5000/v2/_catalog
{"repositories":["bbb-gitlab","bbb-gitlab-runner","bilo","flutterapi","flutterapi2","hel
l"]}
```

Şekil 3.11. Registry web servis uç nokta isteği

Registry kurulumu tamamlandıktan sonra kullanacak olan istemciler sertifika problemleriyle karşılaşacaktır. Yerel ve güvenilir bir ağdaysanız SSL kullanılması bir gereklilik değildir. Test ortamında SSL sertifikası oluşturarak dağıtmak yerine HTTP protokolüyle servise bağlanabilmek için */etc/docker/daemon.json* dosyasına *insecure-registries* alanı açılmıştır.

```
with beh@debian1 at 04:09:00 PM
> cat /etc/docker/daemon.json
{
  "insecure-registries": ["localhost:5000", "172.16.22.28:5000"]
}
```

Şekil 3.12. Docker yapılandırma dosyası

Artık GitLab CI üzerinde projelerimizi otomatize etmek için yapılması gereken işlemler sadece proje spesifik işlemlerdir. .NET Core RESTful API projesini otomatize etmek için projeye iki adet dosya dahil edilmelidir. Birinci dosya *.gitlab-ci.yml*, ikinci dosya *docker-compose.yml* dosyasıdır. *.gitlab-ci.yml* dosyası Runner'ın aşamalarını belirleyecek ve bu aşamalarda hangi komutları çalıştırması gerektiğini anlatacak olan yapılandırma dosyasıdır. *docker-compose.yml* dosyası ise docker-compose ortamında yürütülmesi gereken yapılandırma komutlarını içeren dosyadır.

```
version: "3.8"

services:
  dotnetapi:
    build:
      context: .
    image: 172.16.22.28:5000/dotnet-api:test
    ports:
      - '8888:80'
```

Çizelge 3.6. Bir .NET Core servisini 8080 portu üzerinden yayınlamak için YAML dosyası



```
image: docker/compose
stages:
  - build
  - push
  - deploy
build:
  stage: build
  script:
    - docker-compose build
  only:
    - main
push:
  stage: push
  script:
    - docker-compose push
  only:
    - main
deploy:
  stage: deploy
  script:
    - docker-compose up -d
  only:
    - main
```

Çizelge 3.7. GitLab CI aşamalarında yapılacak olan otomatizasyon için YAML dosyası

Yukarıda görüldüğü gibi üç aşamalı bir CI süreci bulunmaktadır;

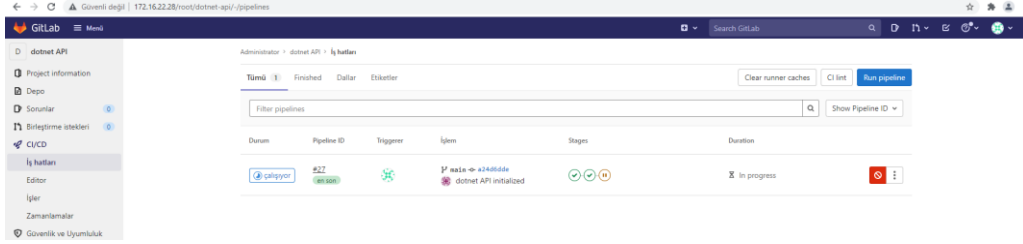
Build aşamasında docker-compose build komutu yürütülür. Bu komut "*docker-compose.yml*" dosyasında bulunan yapılandırmalara bakarak aynı bağlam içerisinde 172.16.22.28:5000/dotnet-api:test etiketine sahip imaj oluşturmaktadır.

Push aşamasında docker-compose push betiği koşturulur. Bu komut ise yeni üretilen 172.16.22.28:5000/dotnet-api:test etiketine sahip imajı başındaki registry etiketine bakarak servisine göndermektedir.

Deploy aşamasında docker-compose up -d komutu çalıştırılır. Bu komut ise oluşturulan yeni imajı [http://<ip\\_adresi>:8888](http://<ip_adresi>:8888) adresi üzerinden yayınlamaktadır.



Boş bir projeye yeni bir commit atıldığında tüm aşamalar gerçekleştirilir ve hatasız bir CI süreci gerçekleştirilirse yeni proje yayına geçmektedir. Eğer ki hata alınırsa eski sürümdeki proje 8888 üzerinde yayın yapmada devam etmektedir.



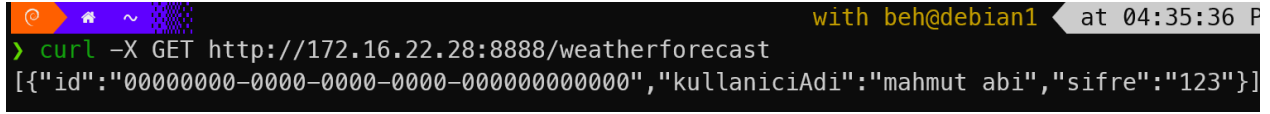
Şekil 3.13. GitLab proje iş hatları

İşlemler başarıyla tamamlandığında Runner çalışmayı durduracaktır ve servis 8888 portu üzerinden yayınlanacaktır.



Şekil 3.14. GitLab proje iş hattı sonucu

Web Servis:



Şekil 3.15. GitLab ile yayınlanan servis uç noktası isteği



### 3.2.3. Sonuçlar

Git sistemi başlangıçta monolitik olarak kuruldu, dökümanda da görüldüğü gibi zaten tek bir servis olarak çalışıyor. Ancak bu bir yazılımcı için yeterli değildi ve biraz araştırma sonucunda GitLab'ın ücretsiz olarak bir imaj sağladığını gördük. Bu imajı Docker ile gerekli konfigürasyonlarını gerçekleştirerek test ortamında ayağa kaldırdık. Hedeflendiği gibi bir web arayüzü olan sistem üzerinden kontrol edilebilmektedir. Yazılımcıların işletim sistemleri üzerinden üretilen SSH keyleri de GitLab sistemine dahi edildi ve ufak seanslar halinde kullanımı için anlatımlar gerçekleştirildi.

Sonuç olarak bakıldığında imaj tam olarak isteneni vermektedir. Kodlar yerel sunucumuzda, yazılımcılarımız aktif bir şekilde Git kullanıyor ve çeşitli problemlerden yazılımcıları kurtarıyor. Örneğin APK üretilmesi gereken durumlarda USB aracılığıyla kod alıp kod birleştirme gibi sorunlar ortadan kalkmış bulunmaktadır.



### 3.3. Portainer Kurulumu

#### 3.3.1. Çalışmanın Hedefi

Portainer, Docker soketine iletebildiğiniz her komutun bir web arayüzünden gerçekleştirilmesine olanak sağlayan açık kaynak kodlu bir araçtır. K8s ya da Docker Swarm gibi bir araç değildir. Yalnızca çalışan servislerinizi, yığınlarınızı veya tekil haldeki konteynerlerinizi izleyip yönlendirebilmenize olanak sağlamaktadır. Portainer, Docker Swarm ve K8s ile de uyumlu çalışmaktadır. Şu an için BBB bünyesinde yalnızca Docker Swarm ve Docker Compose kullanılmaktadır dolayısıyla Portainer kurulumunda da Docker Swarm olan imaj tercih edilmiştir. Bu görevin amacı bir konteyner yönetim aracını sunucu üzerinde barındırabilmektir.

### 3.3.2. Yöntem

Portainer yayınlayabilmek için yalnızca bir YAML dosyası oluşturulması yeterlidir. Ancak bu YAML dosyasında kontrol etmek istenilen Docker soketinin yolu volume olarak verilmelidir, verilmediği bir senaryoda teorik ve teknik olarak yönetim imkansızdır.

```
version: "2"
```

```
services:
```

```
  portainer:
```

```
    image: portainer/portainer-ce:latest
```

```
    ports:
```

- '8000:9000'
- '9443:9443'

```
volumes:
```

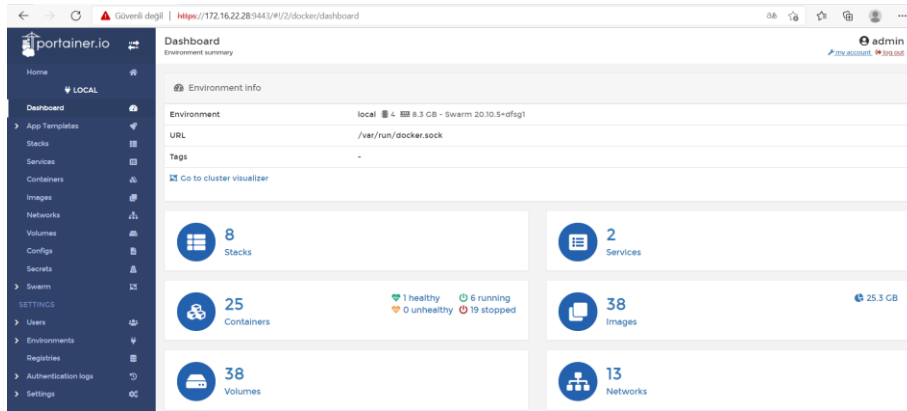
- '/var/run/docker.sock:/var/run/docker.sock'

Çizelge 3.8. Portainer'ı 8000 portu üzerinden yayınlamak için kullanılan YAML dosyası

Aşağıdaki komut ile test ortamında ayağa kaldırdıktan sonra [https://<ip\\_adresi>:9443](https://<ip_adresi>:9443) adresinden web arayüzüne erişilerek konteynerlere erişim sağlanabilmektedir.

```
docker-compose up -d
```

Portainer, oldukça sağlıklı çalışan bir web arayüzü olmakla birlikte Docker Swarm ve K8s desteği sağlaması sayesinde ile diğer yönetim araçlarından bir adım öne taşımaktadır. Bütün konteynerizasyon kaynakları tek bir arayüz üzerinden yönetilebilmektedir. Aşağıdaki resim üzerinden de arayüzün nasıl olduğu görülmektedir.



Şekil 3.16. Portainer web arayüzü

Bu form BTÜ-İMEP Koordinatörlüğü tarafından hazırlanmıştır.

İzinsiz kopyalanması, çoğaltılması kullanılması ve değiştirilmesi yasaktır. Sayfa 36



### 3.3.3. Sonuçlar

Portainer aracı gerçekten çok güçlü bir araç ve üzerinden neredeyse tüm işlemlerimizi gerçekleştirebiliyoruz. Hedefe başarıyla ulaşıldı ancak bekleneni vermediği söylenebilir. Pratik bir şekilde işlem gerçekleştirmek için bir web arayüzüne gidip fare ile işlemler gerçekleştirmektense CLI üzerinden gerçekleştirmenin çok daha ergonomik olduğunu söyleyebilirim. Portainer şu anda sistem üzerinde sorunsuz çalışmakta ancak kullanıma sıklığı epey bir az olan bir araç olarak yerini almaktadır.



### 3.4. Moodle

#### 3.4.1. Çalışmanın Hedefi

BBB, kendi içerisinde sürekli bir eğitim döngüsü içeren bir devlet kurumudur. 2019 yılına kadar uzaktan eğitim modeline ihtiyaç duyulmamıştır ancak COVID-19 ile eğitimler sekteye uğramaya başladığından kurum içerisindeki farklı şubelerin uzaktan eğitim gerçekleştirebilmek için YŞM’e talepleri olmaya başlamıştır. YŞM’de gelen talepleri vizyoner bir tutumla karşılayarak bir uzaktan eğitim sistemi arayışına girmiştir.

Günümüzde uzaktan eğitim alanında SaaS servislerinin artması ve rekabet ortamının artmasıyla beraber her ne kadar maliyetlerin düşmüş olsa da Belediye gibi bir kurumda veri güvenliğinin hat safhada olması beklenmektedir. Moodle uzaktan eğitim sistemi Dünya üzerinde kabul görmüş ve kullanıcı sayısı ile çok büyük bir topluluğa sahip açık kaynak kodlu, ücretsiz ve GNU lisansına sahip bir yazılımdır. Bir yazılımı kullanmaya başlamadan önce kesinlikle kılavuzunun okunması gerektiği gibi Moodle sistemi için de dökümantasyonu okuyarak işe başlanması gerekmektedir. Dökümantasyonunun başında Moodle’in özelliklerinden bahsediyor, her ne kadar daha önce üniversitemiz aracılığıyla kullanıcı olarak da test ettiğim bir sistem olsa da BBB açısından birkaç çarpıcı özelliğini vurgulamak gerekmektedir;

- ✓ Online olarak ortam üzerinden ders, sınıf ve topluluk yönetimi yapılabilir,
- ✓ Online olarak ortam üzerinden sınav yapılabilir, sınav puanlaması gerçekleştirilebilir,
- ✓ Türkçe dil desteği başta olmak üzere birçok dil desteği mevcut,
- ✓ Eklenti sistemi sayesinde esnek bir yapı sunuyor, BBB olarak gerekli farklı özellikler uygulanabilir,

Moodle platformu geniş bir dokümantasyona<sup>[6]</sup> sahiptir. Bu dokümantasyon içerisinde nasıl kurulacağından, nasıl güncelleneceğine kadar detaylı argümanlar ile açıklanmıştır. Ancak tam bu noktada YŞM farklı bir vizyon ile hareket ederek konteynerizasyon teknolojilerinin dünya çapında güvenilirliğinin arttığını düşünerek, Moodle platformunu mikroservisler aracılığıyla son kullanıcıya sunarak, gerekli anlarda gerekli ölçeklendirmeleri ve güncellemeleri daha rahat ve güvenilir bir şekilde gerçekleştirebilmeyi, yayınlanan güvenlik yamalarını sorunsuz şekilde geçebilmeyi ve en önemlisi yüksek kullanılabilirlik ile hizmet verebilmeyi hedeflemektedir.

### 3.4.2. Yöntem

Moodle platformu başlangıç olarak bir Linux işletim sistemi üzerinde monolitik bir uygulama gibi ayağa kaldırarak çalışabilirliğini incelenmelidir.

Öncelikle aşağıdaki paketlerin makine üzerinde kurulu olması gerekmektedir.

```
with beh@debian at 04:13:47 PM
> sudo apt install apache2 mysql-client mysql-server php7.4 libapache2-mod-php7.4 php7.4-curl php7.4-zip php7.4-mysql php7.4-xml php7.4-mbstring php7.4-gd php7.4-intl php7.4-xmllrpc php7.4-soap
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Şekil 3.17. Moodle gerekli paketlerin kurulumu

İşlem başarıyla gerçekleştirildikten sonra aşağıdaki komut ile Apache2 tekrar başlatılmalı ve yapılandırmaların gerçekleştirilmiş olması beklenmelidir.

```
with beh@debian at 04:18:34 PM
> sudo systemctl restart apache2.service
```

Şekil 3.18. Apache2 servisini yeniden başlatma

Moodle uygulamasını güncel tutabilmek adına Git kullanılmaktadır. Dolayısıyla servise git kurulmalıdır.

```
with beh@debian at 04:20:37 PM
> sudo apt install git
Reading package lists... Done
Building dependency tree... Done
```

Şekil 3.19. Git aracının kurulması

Artık MySQL8 yapılandırmaları gerçekleştirilmelidir. Yapılması gerekenler;

- ‘moodle’ adında bir veri tabanı oluşturulmalıdır.
- ‘moodle’ veri tabanı üzerinde yetkilendirilecek yeni bir kullanıcı oluşturulmalıdır.
- Oluşturulan kullanıcıya gerekli yetkiler aktarılmalıdır.

```
mysql> CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
Query OK, 1 row affected (0.00 sec)

mysql> create user 'bbbm'@'localhost' IDENTIFIED BY '123';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE TEMPORARY TABLES,DROP,INDEX,ALTER ON
moodle.* TO 'bbbm'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> exit
Bye
```

Şekil 3.20. Veri tabanı oluşturma ve yetkilendirme

MySQL8 yapılandırmaları tamamlandıktan sonra artık Moodle içi yapılandırmalara geçilmelidir. Moodle uygulaması mevcut GitHub üzerinden /opt dizinine indirilmelidir.

```
@ /opt with beh@debian at 04:22:56 PM
> sudo git clone https://git.in.moodle.com/moodle/moodle.git
Cloning into 'moodle'...
remote: Enumerating objects: 1260693, done.
```

Şekil 3.21. Moodle'ın git üzerinden indirilmesi

İndirme tamamlandıktan sonra ./moodle klasörünün içerisine girip branchler kontrol edilmelidir. Ve listelenen branchler arasından istenen branch kullanılabilir. Şu an en kararlı sürüm MOODLE\_311\_STABLE olduğu için bu sürüm ile devam edilecektir.

```
@ /opt/moodle with beh@debian at 04:26:09 PM
> sudo git branch --track MOODLE_311_STABLE origin/MOODLE_39_STABLE
Branch 'MOODLE_311_STABLE' set up to track remote branch 'MOODLE_39_STABLE' from 'origin'.
```

Şekil 3.22. Git branch değiştirme

Apache2 /var/www/html dizini altındaki dosyaları yayınlamaktadır. Dolayısıyla indirilen Moodle dosyaları bu dizine taşınmalıdır. Ek olarak güvenlik açısından dosya yetkilendirmeleri de değiştirilmelidir.

```
@ /opt with beh@debian at 04:28:48 PM
> sudo cp -R /opt/moodle /var/www/html/ && \
cmdand> sudo chmod -R 0755 /var/www/html/moodle

@ /opt with beh@debian at 04:29:04 PM
> ls /var/www/html
index.html moodle
```

Şekil 3.23. Moodle dosyalarını taşıma işlemleri



Dosyalar taşındıktan sonra /srv dizini altında moodledata adında bir klasör oluşturulmalıdır ve yetkilendirilmeleri yapılandırma tamamlanana kadar aşağıdaki gibi bırakılmalıdır. Tüm Moodle yapılandırmaları bu klasör içerisinde saklanacaktır. Veri tabanında saklanacak veriler hariçtir.

```
with beh@debian at 04:32:03 PM
> sudo mkdir /srv/moodledata && \
cmdand> sudo chown -R www-data /srv/moodledata && \
cmdand cmdand> sudo chmod -R 777 /srv/moodledata

with beh@debian at 04:33:31 PM
> ls /srv
moodledata
```

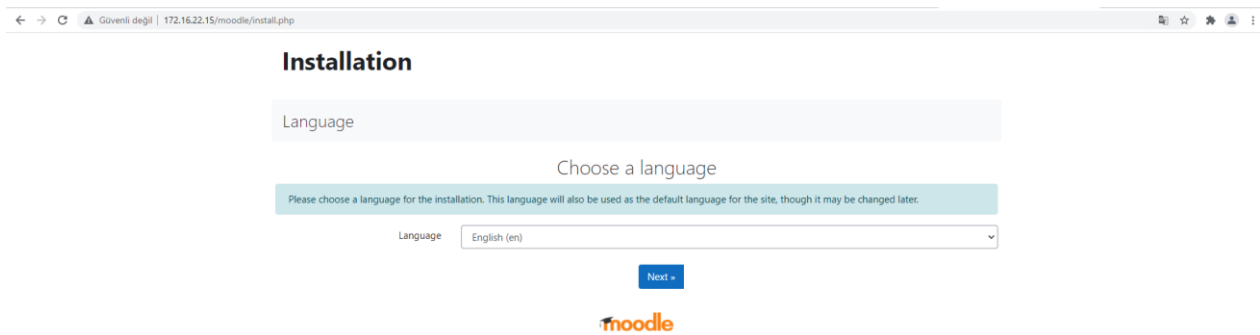
Şekil 3.24. Moodle verilerinin saklanacağı dizinlerin oluşturulması

Yapılandırmaya başlamadan önce klasörler için tüm kullanıcıların yetkili olacağı şekilde ayarlanmalıdır.

```
with beh@debian at 04:44:07 PM
> sudo chmod -R 777 /var/www/html/moodle
```

Şekil 3.25. Moodle’ın dosya izinlerinin değiştirilmesi

Ardından <http://172.16.22.15/moodle> adresinden Moodle arayüzüne bağlanılır. Ve aşağıdaki adımlar sırasıyla uygulanmalıdır. Eğer ki bir adımda hata veya farklı bir durum meydana gelirse dökümantasyon aracılığıyla hatalar giderilmelidir. Kurulum esnasında karşılaşılan hata ve çözümleri de rapor içerisinde belirtilmiştir.



Şekil 3.26. Moodle kurulum arayüzü

- Dil Türkçe olarak seçilmelidir
- Ortam kontrolü başarıyla tamamlandıysa sıradaki adıma geçilmelidir
- Veri tabanı dizini /srv/moodldata olarak değiştirilir
- Veri tabanı sürücüsünü MySQL olarak seçilir
- İlerleyen sekmede sizden veri tabanı bilgileri istenir ve eksiksiz bir şekilde doldurulur
- Telif hakları ile ilgili istenen kutucuk onaylanır
- Onaydan sonra PHP için gerekli kütüphanelerin durumlarını geliştiriciye gösterilir ve tüm paketler yeşil ise devam edilebilir. Aksi takdirde o paket(ler) kurulmalı ve daha sonraki adıma geçilmelidir

Genelde php.ini içerisindeki `max_input_vars` değişkeni varsayılan olarak 1000 değerinde gelir. Ancak Moodle bunun 5000 olarak ayarlanmasını tavsiye etmektedir. Bu yapılandırmaları gerçekleştirmek yapmak için aşağıdaki komutlar uygulanmaktadır.

```
@ /opt with beh@debian at 04:50:26 PM o
> php -i | grep php.ini
Configuration File (php.ini) Path => /etc/php/7.4/cli
Loaded Configuration File => /etc/php/7.4/cli/php.ini
```

Şekil 3.27. PHP varsayılan yapılandırma dosyasını bulma

Buradan *loaded configuration file* dosyasının yolu kopyalanır ve *nano* ile düzenleme modunda açılır ve ardından `max_input_vars` değerinin önünde varsa “;” ifadesini kaldırılmalıdır ve “5000” olarak değiştirilmelidir. Ardından yapılandırmaların aktifleştirilmesi için *apache2* servisi yeniden başlatılmalıdır.

```
> sudo systemctl restart apache2.service
@ with beh@debian at 10:22:31 AM o
```

Şekil 3.28. Apache2 servisini yeniden başlatma

Burada durumu “Tamam” olmayan tüm paket varsa hepsini sunucuya indirilmeli ve kurulmalıdır.

```
sudo apt install <library_name>
```

Tüm paketlerin durumu “Tamam” ise sonraki adıma geçilmeli ve yapılandırmalar tamamlanana kadar beklenmelidir. Bir sonraki formda istenen admin bilgilerini eksiksiz doldurarak devam edilir. Site için istenen bilgiler de eksiksiz olarak doldurulmalıdır.

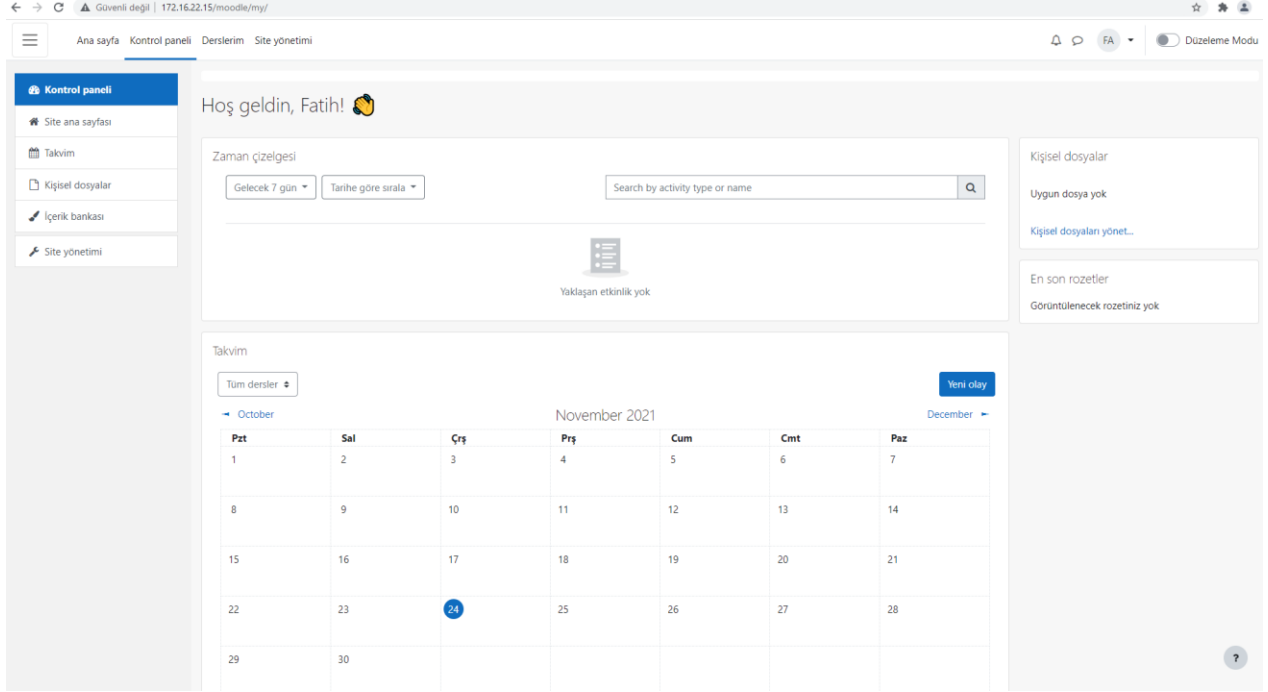
Son olarak da dosya izinleri eski haline getirilmelidir.

```
> sudo chmod -R 755 /var/www/html/moodle
```

with beh@debian at 11:14:00 AM

Şekil 3.29. Moodle’ın dosya izinlerinin değiştirilmesi

Ve artık kurulum tamamlanmıştır.



Şekil 3.30. Moodle admin arayüzü

## Troubleshooting

### Apache2 (exit-status=139)

Yeni Linux dağıtımlarında php8.0 otomatik olarak gelmektedir. Ancak moodle php7.4 ile çalışır.

- Apache2 üzerinde php8.0 deaktif edilir ve ardından php7.4 aktifleştirilir.

```
sudo a2dismod php8.0 && sudo a2enmod php7.4
```

- Ardından apache2 servisini yeniden başlatılır.

```
sudo systemctl restart apache2
```



### **libc6 is not installable**

Kurulan işletim sistemine göre bu paket yüklenmemiş olabilir. Ve bağlı olarak diğer paketlerin yüklenmesini engelleyebilir.

- Aşağıdaki adresten paketi indirilir.

<https://www.ubuntuupdates.org/package/core/focal/main/base/libc6>

- Paketle aynı dizine gelinir ve aşağıdaki komut çalıştırılır

`dpkg -i <package_name>`

### **xmlrpc-epi0 is not installable**

Kurulan işletim sistemine göre bu paket yüklenmemiş olabilir. Ve bağlı olarak diğer paketlerin yüklenmesini engelleyebilir.

- Aşağıdaki adresten paketi indirilir.

<https://www.ubuntuupdates.org/package/core/focal/main/base/libxmlrpc-epi0>

- Paketle aynı dizine gelinir ve aşağıdaki komut çalıştırılır

`dpkg -i <package_name>`

### **Dockerize edilmiş bir imaj ile Moodle Ayağa kaldırmak**

Bunun için VMware'in bir alt şirketi olan Bitnami'nin sağladığı ücretsiz Docker imajları kullanılabilir. Bitnami'nin GitHub üzerinden yayınladığı [Readme<sup>\[7\]</sup>](#) dosyasında “*docker-compose.yml*” dosyası üzerinde mariadb yerine mysql kullanarak bir Moodle platformu ayağa kaldırılabilir.

```
version: '2'
services:
  mysql:
    image: mysql:latest
    container_name: mysql
    environment:
      MYSQL_USER: bbb
      MYSQL_PASSWORD: 123
      MYSQL_ROOT_PASSWORD: 123
      MYSQL_DATABASE: moodle
      TZ: Europe/Istanbul
  moodle:
    image: docker.io/bitnami/moodle:3
    ports:
      - '85:8080'
      - '443:8443'
    environment:
      - MOODLE_DATABASE_HOST=mysql
      - MOODLE_DATABASE_TYPE=mysqli
      - MOODLE_DATABASE_PORT_NUMBER=3306
      - MOODLE_DATABASE_USER=bbbmoodle
      - MOODLE_DATABASE_NAME=moodle
      - MOODLE_DATABASE_PASSWORD=123
      - MOODLE_ROOT_PASSWORD=123
      - BITNAMI_DEBUG=1
      - MOODLE_SITE_NAME="BBB MOODLE"
      - MOODLE_USERNAME=bbbmoodle
      - MOODLE_PASSWORD=123
      - PHP_UPLOAD_MAX_FILESIZE=10M
  depends_on:
    - mysql
```

Çizelge 3.9. Moodle'ı mikroservis mantığıyla yayınlayabilmek için YAML dosyası

Docker-compose yardımıyla ayağa kaldırdıktan sonra 85 portu üzerinden HTTP isteklerine karşılık verecektir.

```
> docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

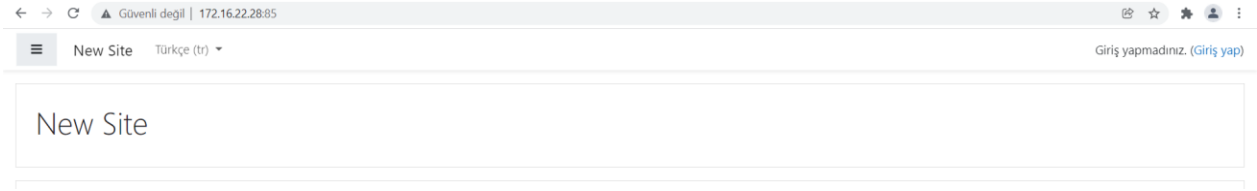
Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Starting moodle_moodle_1 ... done
```

Şekil 3.31. Moodle ayağa kaldırma

Sunucunun 85 portundan geriye dönen yanıt aşağıdaki gibidir ve başarılıdır.



Şekil 3.32. Moodle 85 portu üzerinden erişim

Ancak compose dosyasında da görüldüğü üzere halen mikroservis mimarisiyle çalışan bir uygulama ortaya koyulamamıştır. Uygulama aşağıda komutla beraber ölçeklenmek istenirse dışarıya açılan portların zaten daha önceden açık olduğuna dair bir hata ile karşılaşılır.

```
> docker-compose scale moodle=2
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
WARNING: The "moodle" service specifies a port on the host. If multiple containers for this service are created on a single host, the port will clash.
Starting moodle_moodle_1 ... done
Starting moodle_moodle_2 ...

ERROR: for moodle_moodle_2 a bytes-like object is required, not 'str'
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/docker/api/client.py", line 261, in _raise_for_status
    response.raise_for_status()
  File "/usr/lib/python3/dist-packages/requests/models.py", line 943, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 500 Server Error: Internal Server Error for url: http+docker://localhost/v1.22/containers/01fc9fd1e81c51a1350597085332b8a0d697300b20d1b9ec44f4290d4f14ad8f/start
```

Şekil 3.33. Moodle ölçeklendirme ve hata

Alınan hata beklenen bir durumdur, içeride birden fazla çalışan konteynerin tek bir port üzerinden dışarıya sunulmak istenmesi sonucunda ortaya çıkmaktadır. Bu durumda konteynerlerin portlarını dışarıya açık hale getirmek yerine önlerine bir servis(örn: load balancer) eklenerek konteynerlerin bu servis üzerinden işlem görmesi sağlanır.

Compose dosyası içerisinde port yönlendirmelerini sildikten sonra uygulamayı yeniden ayağa kaldırarak ölçeklendirme işlemi aşağıdaki gibi gerçekleştirilebilir.

```
> docker ps | grep moodle
> docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating moodle_moodle_1 ... done
> docker-compose scale moodle=2
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
Starting moodle_moodle_1 ... done
Creating moodle_moodle_2 ... done
> docker ps | grep moodle
5408da967cab 172.16.22.28:5000/moodle:latest      "/opt/bitnami/script..." 3 seconds ago Up 2 seconds 8080/tcp, 8443/tcp
moodle_moodle_2
7a800b97daea 172.16.22.28:5000/moodle:latest      "/opt/bitnami/script..." 10 seconds ago Up 9 seconds 8080/tcp, 8443/tcp
moodle_moodle_1
```

Şekil 3.34. Moodle port yönlendirmeleri olmadan ayağa kaldırma ve ölçeklendirme

Görüldüğü üzere ölçeklendirme başarıyla gerçekleştirildi. Ancak şu anda platforma istemciler erişmiyor. Öncelikle aşağıdaki compose dosyası yardımıyla bir load balancer oluşturulmalıdır.

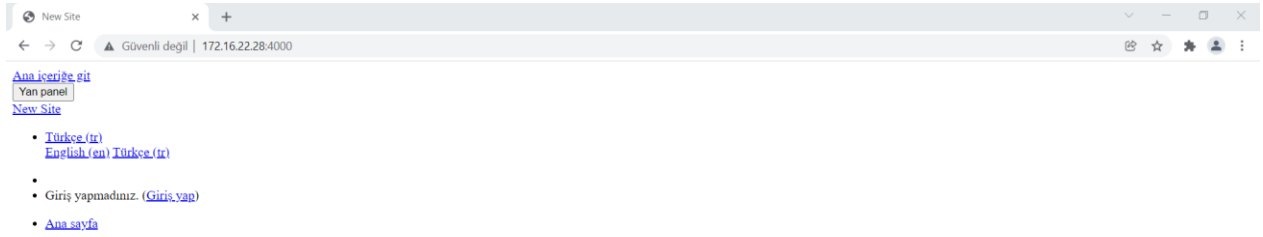
```
user nginx;
events {
    worker_connections 1000;
}
http {
    server {
        listen 4000;
        location / {
            proxy_pass http://moodle:8080;
        }
    }
}
```

Çizelge 3.10. Load balancer hizmeti sunabilmek için NGINX yapılandırma dosyası

```
version: '2'
services:
  nginx:
    image: nginx:latest
    volumes:
      - /home/beh/assets/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "4000:4000"
    networks:
      - moodle_moodle_network
networks:
  moodle_moodle_network:
    external: true
```

Çizelge 3.11. Load balancer hizmeti sunabilmek için YAML dosyası

Artık 4000 portu üzerinden Moodle uygulamasına erişilebilir ve Moodle compose dosyası istenildiği gibi ölçeklendirilebilir.

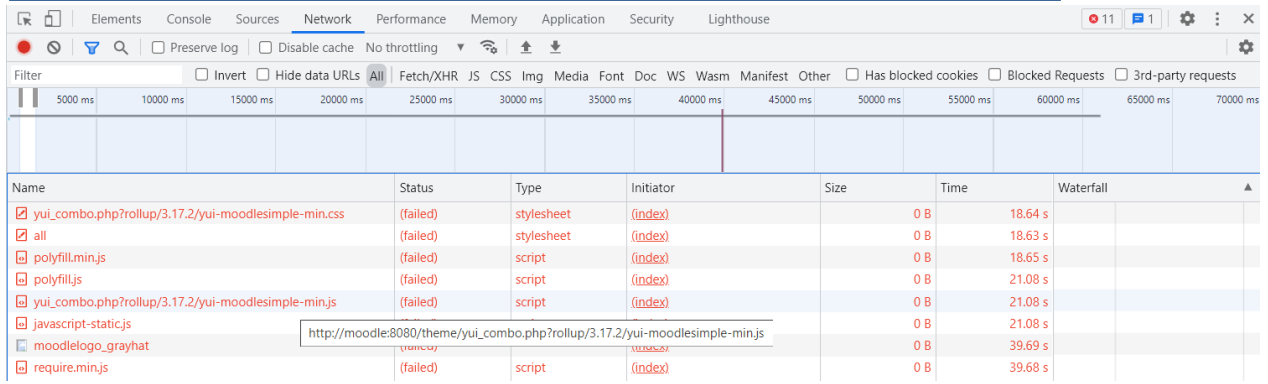


New Site

Şekil 3.35. Moodle admin arayüzü, yönlendirme hatası

Ekran görüntüsünde görüldüğü gibi beklenmedik bir şekilde CSS ve JS dosyalarının bir şekilde çalışmamaktadır. Chrome tarayıcı üzerinde Network bölümü incelendiğinde <http://moodle:8080> adresinden CSS ve JS dosyaları getirmeye çalışmaktadır. Fakat bu adrese yalnızca nginx konteyneri içerisinde erişilebilmektedir.





Şekil 3.36. Moodle network hataları

İmaj içerisinde Moodle için host ayarlayabileceğiniz bir konfigürasyon bulunmalıdır. Ama ne yazık ki Bitnami dökümantasyonunda bununla ilgili bir kaynak bulunmuyor. Moodle host bilgilerini *config.php* dosyası içerisindeki *wwwroot* özelliğinden almaktadır. Bitnami imajı *config.php* dosyasını dışarıdan volume ile bağlanmasına izin veriyor ancak tek istenen şey host adı değiştirmek olduğu için bu tam anlamıyla isteğimizi karşılamıyor.

GitHub deposu üzerinde issuelar incelendiğinde başkalarının da bu sorunu yaşadığını rahatlıkla görebiliyorsunuz. En son olarak ise bot otomatik olarak cevap yazılmadığı için sorunu kapatmış bulunuyor.

### wwwroot is not configurable #146

 samueldmq opened this issue on Oct 20, 2020 · 9 comments



samueldmq commented on Oct 20, 2020

BUG REPORT INFORMATION

#### Description

wwwroot is an important config within moodle. There is no way to set it in the image, which configures 127.0.0.1. One of the effects is that users get an email with 127.0.0.1 as the moodle address when trying to reset their passwords.

#### Steps to reproduce the issue:

1. Install the instance and configure SMTP
2. As a user, ask for a reset

#### Describe the results you received:

As wwwroot is not properly configured, the email has a wrong address.

#### Describe the results you expected:

There should be a var to configure wwwroot.

#### Version

Latest.

#### Additional environment details (AWS, VirtualBox, Docker for MAC, physical, etc.):

Installed via Bitnami Moodle Chart

 1

Şekil 3.37. Bitnami Issue

Kullanacağım imajın tamamen yalın YAML dosyaları ile birlikte bir volume’a ihtiyaç duymadan çalışmasını gerektiğine inanıyorum dolayısıyla kaynak kodlar içerisinde değişiklik gerçekleştirerek bir MOODLE\_HOST değişkeni oluşturmalıydım. Kaynak kodlar incelendiğinde “moodle-env.sh” dosyası altında bütün Moodle ile ilgili ortam değişkenlerinin tanımlandığı görülüyor. Diğer tanımlamalardan yardım alarak bu script içerisinde aşağıdaki üç satır eklenmiştir.

moodle-env\_vars değişkeni içerisine;

```
MOODLE_HOST
```

değişkeni eklenmiştir.

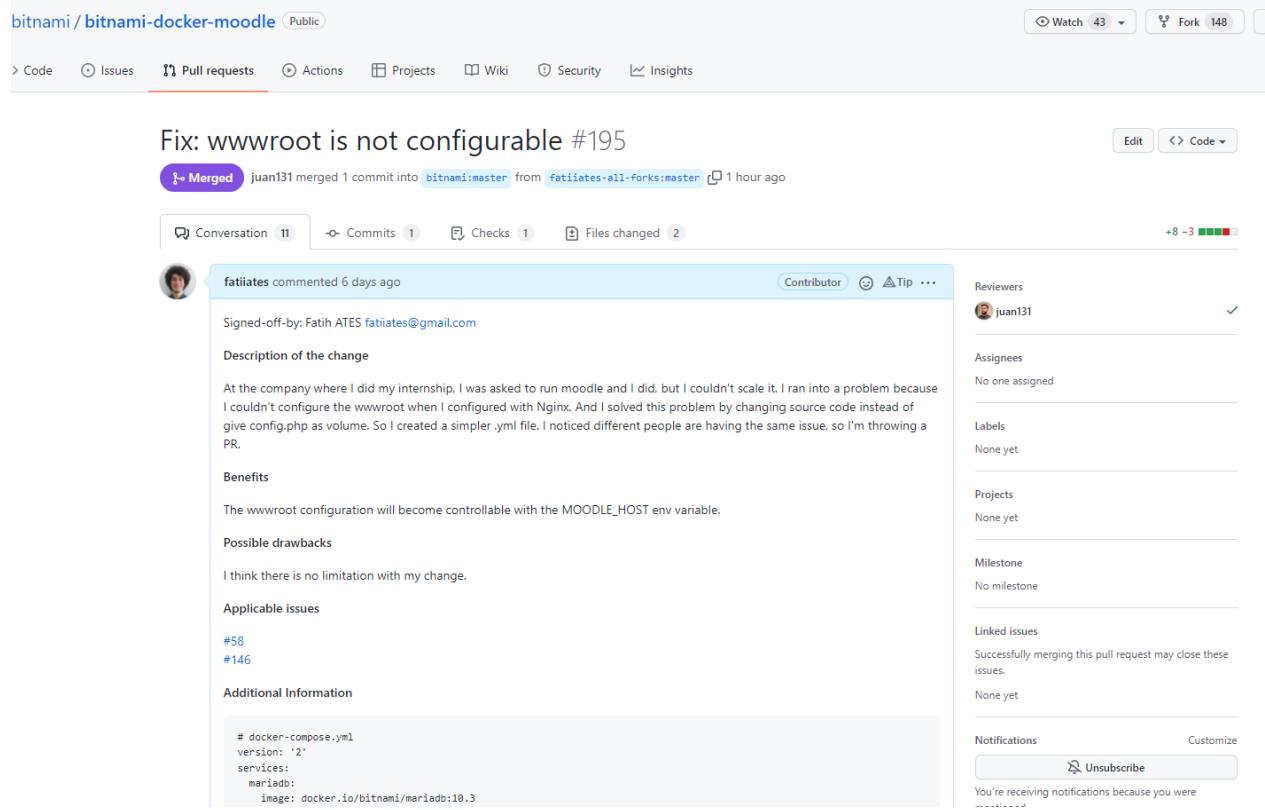
“# Moodle configuration” yorumunun altında konfigürasyonların verildiği yerde ise ortam değişkeni tanımlanmıştır.

```
MOODLE_HOST="${MOODLE_HOST:-}"
```

```
export MOODLE_HOST="${MOODLE_HOST:-}" # only used during the first initialization
```



Kod yenilendikten sonra sorunun çözülmesi için bir PR<sup>[8]</sup> gönderilmiştir. Aşağıdaki ekran görüntüsünde PR'ın içeriği görülebilir.



Şekil 3.38. Bitnami PR

Kaynak kod değiştirildikten sonra Bitnami tarafından dökümanda bahsedilen EXTRA\_LOCALES değişkeni içerisinde Türkçenin aşağıdaki şekilde eklenmesi gereklidir. Bu değişken yalnızca imaj build esnasında kullanılacaktır dolayısıyla derleme esnasında da verilebilir ben Dockerfile içerisinde default değer olarak atamayı tercih ettim. Aşağıdaki şekilde değer güncellenerek Türkçe desteği eklenebilmektedir.

```
ARG EXTRA_LOCALES='tr_TR.UTF-8 UTF-8'
```

Buradaki test sunucusunda daha önceden kurduğum registry servisine uygun halde tag vererek sunucu üzerinde de kullanılması sağlanmalıdır. Artık aşağıdaki komut yardımıyla imaj build edilmelidir ve registry servisine pushlanmalıdır.

```
docker image build -t 172.17.22.28:5000/moodle .
```

docker push

İmaj artık hazır olduğuna göre yukarıda verdiğim *docker-compose.yml* dosyası içerisinde Moodle servisine aşağıdaki ortam değişkeni dahil edilerek nginx load balancer üzerinden Moodle servisine erişilebilir.

- MOODLE\_HOST=172.16.22.28:4000

Nginx üzerinden artık erişilebildiğine göre artık ölçeklendirme gerçekleştirilebilir halde olmalıdır. Aşağıdaki komut girildiğinde oluşturduğumuz imajdan üç adet konteyner ayakta olmalıdır.

docker-compose scale moodle=3

Ancak bu şekilde konteynerler ayakta olmalarına rağmen herhangi bir trafik karşılayamazlar. Bunun için *nginx.conf* dosyasında değişiklikler yapılmalıdır. Scale edilmiş konteynerler sıralı şekilde scale edilir. Aşağıdaki örnekte *docker ps* çıktısı sonucunda verilen isimlendirmeler görülmektedir.

```
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
33554b697ee4   172.16.22.28:5000/moodle            "/opt/bitnami/script..." 2 minutes ago  Up 2 minutes  8080/tcp, 8443/tcp                 moodle_moodle_1
5880c446ca99   bitnami/mariadb:10.3                "/opt/bitnami/script..." 2 minutes ago  Up 2 minutes  3306/tcp                           moodle_mariadb_1
$ docker-compose scale moodle=3
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
Starting moodle_moodle_1 ... done
Creating moodle_moodle_2 ... done
Creating moodle_moodle_3 ... done
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
f7edf17f95d9   172.16.22.28:5000/moodle            "/opt/bitnami/script..." 6 seconds ago  Up 6 seconds  8080/tcp, 8443/tcp                 moodle_moodle_2
2796d3157765   172.16.22.28:5000/moodle            "/opt/bitnami/script..." 6 seconds ago  Up 5 seconds  8080/tcp, 8443/tcp                 moodle_moodle_3
33554b697ee4   172.16.22.28:5000/moodle            "/opt/bitnami/script..." 2 minutes ago  Up 2 minutes  8080/tcp, 8443/tcp                 moodle_moodle_1
5880c446ca99   bitnami/mariadb:10.3                "/opt/bitnami/script..." 2 minutes ago  Up 2 minutes  3306/tcp                           moodle_mariadb_1
```

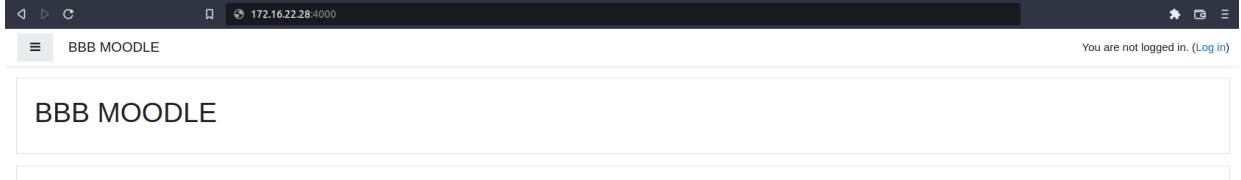
Şekil 3.39. Moodle ölçeklendirme

Nginx konteyneri içerisinde konteynerlere bu isimler aracılığıyla erişilmektedir. Bu isimleri bir *upstream* altında toplayarak Proxy'e gelen istekleri bu *upstream* üzerinden dağıtmak gerekmektedir. *nginx.conf* dosyası aşağıdaki gibi düzenlenmelidir.

```
user nginx;
events {
    worker_connections 1000;
}
http {
    upstream moodle_containers {
        server moodle_moodle_1:8080
        server moodle_moodle_2:8080
        server moodle_moodle_3:8080
    }
    server {
        listen 4000;
        location / {
            proxy_pass http://moodle_containers
        }
    }
}
```

Çizelge 3.13. Load balancer hizmetini farklı uç noktalardan sunmak için yapılandırma dosyası

Artık aşağıdaki ekran görüntüsünde görüldüğü üzere Moodle arayüzüne ölçeklendirilmiş sistem üzerinden erişilebilmektedir.



Şekil 3.40. Moodle load balancer arkasındayken admin arayüzü

**Not:** Moodle kendi içerisinde bir reverseproxy özelliği bulundurmaktadır. Bu özelliği açık hale getirmeden load balancer arkasından servis sağlanamamaktadır. Özelliği açmak için *config.php* içerisine aşağıdaki satır eklenmelidir.

```
$CFG->reverseproxy = true;
```

Ölçeklemenin asıl amacı servis az kullanılırken servisin kaynak kapasitesini sınırlamaktır. Yüksek trafik beklendiği zamanlarda kolay bir şekilde ölçekleyerek servis dışı kalmadan sorunu pratik bir şekilde



çözmektir. Dolayısıyla ölçeklemenin işe yaraması için cpu ve ram sınırlamaları da aşağıdaki şekilde moodle servisi için ayarlanmalıdır.

```
deploy:
  resources:
    limits:
      cpus: '0.25'
      memory: 4G
    reservations:
      cpus: '0.1'
      memory: 500M
```

Çizelge 3.14. YAML dosyası içerisinde konteynerlerin kaynaklarını sınırlamak için anahtarlar



### 3.4.3. Sonuçlar

Ölçeklendirme anlamında kolaylık olması ve yeni teknolojilere uyum sağlama sürecinde test ortamında Docker ile Moodle ayağa kaldırma hedefi başarılı. Monolitik teknolojiler ile ayağa kaldırma süreci de tecrübe edilmiştir ancak onlarca komut yazdıktan sonra bir Moodle servisi kaldırmak yerine açık kaynak kodlu bir imaj bularak onun üzerinden bir YAML dosyası oluşturmak ve uygulamayı buradan dağıtmak çok daha anlamlı ve kullanılabilir sonuçlar vermiştir. Ek olarak saniyeler içerisinde ölçeklendirme gerçekleştirilebilmesi gerçekten büyük fayda ve fark sağlamaktadır.





### 3.5. .NET Core Web Servisi

#### 3.5.1. Çalışmanın Hedefi

BBB’de geliştirilen yeni yazılımlarda artık daha yeni teknolojiler kullanılıyor. Örneğin; Flutter, .NET Core gibi... Burada kurduğum test ortamında da hali hazırda bir .NET Core web servisi çalıştırılmaktadır. Ek olarak GitLab CI yardımıyla otomatize de edilmiştir. Ancak teknoloji ne kadar gelişmiş ve yazılımcı odaklı hale gelmiş olursa yayınlarken fazlasıyla konfigürasyon gerçekleştirme ihtiyacı duyulmaktadır.

Docker ile .NET Core imajı oluşturmak oldukça basittir, kaldı ki zaten Dockerfile dosyasını da otomatik oluşturmaktadır. Ancak otomatik oluşturulmuş bir Dockerfile dosyasının çeşitli sorunları olabiliyor. Base imajlarda açıkların ortaya çıkması, ara servislerle imajın uyumlu çalışmaması gibi çeşitli problemleri de çözerek ilerlemek gerekmektedir. Bunun için 2010 yılında Heroku ile başlayan bir imaj paketleme sürecinden söz edilmektedir. Heroku’nun ortaya attığı buildpacks kavramı ile zaten daha önceden mikroservis bazlı veya çeşitli ortamlar için hazırlanmış imajları base olarak alındığı için birçok problemin üstesinden gelebiliyor ve ana amaç olaraksa Dockerfile’den bağımsız olarak pack CLI aracıyla imaj üretilmesine olanak sağlamaktadır.

Hedef .NET Core Web servisini öncelikle Dockerfile ile imaj haline getirerek daha sonrasında Buildpacks yardımıyla Dockerfile olmadan imaj haline getirerek yayınlanabilir bir hale getirmek. Ürün ortamında yayınlanması için Dockerfile dosyasının yine el ile her katmanı kendi kontrol ekseniniz içerisinde tutularak gerçekleştirilmelidir. Ancak ürün ortamında yayınlanacağı zaman ben kurumda bulunmayacağım için ana hedefimiz gelecek kişinin kolay ve güvenilir bir şekilde imaj alabilmesini sağlamak ve bunu en az tecrübeli kişiye göre hareket ederek gerçekleştirmek.

### 3.5.2. Yöntem

.NET Core için bir Web API uygulaması oluşturulduğunda Docker seçeneği seçilirse otomatik olarak aşağıdaki gibi bir *Dockerfile* dosyası oluşturmaktadır.

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
COPY ["MuhtarlikServis/MuhtarlikServis.csproj", "MuhtarlikServis/"]
RUN dotnet restore "MuhtarlikServis/MuhtarlikServis.csproj"
COPY . .
WORKDIR "/src/MuhtarlikServis"
RUN dotnet build "MuhtarlikServis.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "MuhtarlikServis.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "MuhtarlikServis.dll"]
```

Çizelge 3.15. .NET Core Web servisini imajlamak için Dockerfile dosyası

Docker ile imaj aşağıdaki komutlar aracılığıyla oluşturulup yayınlanabilir.

```
> docker ps | grep muhtarlik-servis:test
> ls
docker-compose.yml Dockerfile MuhtarlikServis MuhtarlikServis.sln README.md
> docker build -t muhtarlik-servis:test .
Sending build context to Docker daemon 71.68kB
Step 1/17 : FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
--> 6c1368a6a36c
```

Şekil 3.41. .NET Core Docker imajı oluşturma

```
> docker run -d -p 9090:80 muhtarlik-servis:test
4347cf9ce3facaee82564af97ed728d47dee15faa107fbc3e6973c1023d2c612
```

Şekil 3.42. MBS servisinin ayağa kaldırılması

```
> curl -v 172.16.22.28:9090
* Trying 172.16.22.28:9090...
* Connected to 172.16.22.28 (172.16.22.28) port 9090 (#0)
> GET / HTTP/1.1
> Host: 172.16.22.28:9090
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< Date: Tue, 11 Jan 2022 10:31:50 GMT
< Server: Kestrel
< Content-Length: 0
<
* Connection #0 to host 172.16.22.28 left intact
```

Şekil 3.43. Servisin çalışabilirliğinin test edilmesi

Görüldüğü üzere geriye http kodu olarak 404 döndürüyor yani karşıda yanıtlarımıza cevap veren bir servis bulunuyor. Dockerfile ile sorunsuz bir şekilde imajımızı oluşturduk ve çalıştırdık. Ancak ürün ortamında Dockerfile dosyaları genelde yüzü aşkın satır içerdiği için ortaya karmaşık yapılar çıkmaktadır.

Dolayısıyla aynı kaynak kodu ile ancak bu sefer Dockerfile olmadan Buildpacks<sup>[9]</sup> aracı kullanarak bir imaj oluşturulacaktır. Buildpacks ile imaj oluşturabilmek için öncelikle .NET Core için bir builder imajı bulunmalıdır. Bu konuda Buildpacks topluluğuna büyük katkılarda bulunan *Paketo*<sup>[10]</sup> imajları kullanılacaktır. Bu imaj Paketo'nun kendi dökümantasyonundan rahatlıkla bulunabilir. Ek olarak “--builder” parametresini vermeden pack aracını çalıştırılırsa önerilen builder imajlarını ekrana yazdıracaktır.

```
> pack build myapp --env BP_DOTNET_FRAMEWORK_VERSION=5.0.12
Please select a default builder with:

    pack config default-builder <builder-image>

Suggested builders:
  Google:          gcr.io/buildpacks/builder:v1      Ubuntu 18 base image with buildpacks for .NET, Go, Java
, Node.js, and Python
  Heroku:          heroku/buildpacks:18             Base builder for Heroku-18 stack, based on ubuntu:18.04
base image
  Heroku:          heroku/buildpacks:20             Base builder for Heroku-20 stack, based on ubuntu:20.04
base image
  Paketo Buildpacks: paketobuildpacks/builder:base  Ubuntu bionic base image with buildpacks for Java, .NET
Core, NodeJS, Go, Python, Ruby, NGINX and Procfile
  Paketo Buildpacks: paketobuildpacks/builder:full  Ubuntu bionic base image with buildpacks for Java, .NET
Core, NodeJS, Go, Python, PHP, Ruby, Apache HTTPD, NGINX and Procfile
  Paketo Buildpacks: paketobuildpacks/builder:tiny  Tiny base image (bionic build image, distroless-like ru
n image) with buildpacks for Java, Java Native Image and Go

Tip: Learn more about a specific builder with:
    pack builder inspect <builder-image>
```

Şekil 3.44. Buildpacks ile önerilen builderların listelenmesi

Buildpacks ile imaj oluşturmak için aşağıdaki komut pack cli ile kullanılabilir. Ek olarak Buildpacks Tekton, kpack gibi araçlarla birden fazla platformda imaj oluşturulmasına olanak sağlamaktadır.

```
> pack build myapp --builder paketobuildpacks/builder:base --env BP_DOTNET_FRAMEWORK_VERSION=5.0.12
base: Pulling from paketobuildpacks/builder
Digest: sha256:64edeb7c15cd4c2c067480a99cb96d3162739e7c050b242614f5b726d71510fd
Status: Image is up to date for paketobuildpacks/builder:base
base-cnb: Pulling from paketobuildpacks/run
Digest: sha256:59aa1da9db6d979e21721e306b9ce99a7c4e3d1663c4c20f74f9b3876cce5192
Status: Image is up to date for paketobuildpacks/run:base-cnb
==> ANALYZING
```

Şekil 3.45. Buildpacks ile imaj oluşturmak

```
> docker run -d -p 9091:8080 myapp:latest
006b9daebdb4879c1aa6bd1b7c90946ef0937e599175fa5d23a273ef59c90175
> curl -v 172.16.22.28:9091
* Trying 172.16.22.28:9091...
* Connected to 172.16.22.28 (172.16.22.28) port 9091 (#0)
> GET / HTTP/1.1
> Host: 172.16.22.28:9091
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< Date: Tue, 11 Jan 2022 10:57:15 GMT
< Server: Kestrel
< Content-Length: 0
<
* Connection #0 to host 172.16.22.28 left intact
```

Şekil 3.46. Buildpack ile oluşan imajın çalışabilirliğinin test edilmesi

cURL çıktısında görüldüğü üzere 9001 portu üzerinde servis hazır bir şekilde bekliyor yani imaj pack cli aracılığıyla otomatik üretilmiştir ve Dockerfile ihtiyacı ortadan kaldırılmıştır.



### 3.5.3. Sonuçlar

Görüldüğü üzere aynı servisi Buildpacks ile imaj olarak çalıştırılmasını başarılmış, ek olarak Buildpacks ile oluşturulduğunda Dockerfile sorunlarını da ortadan kaldırmıştır. Son olarak iki imajın boyutları karşılaştırılırsa;

```
> docker image ls | grep "myapp\|muhtarlik-servis"
```

muhtarlik-servis	test	bdb005c326f6	35 minutes ago	227MB
myapp	latest	467c8cca4f86	42 years ago	244MB

Şekil 3.47. İmajların listesi

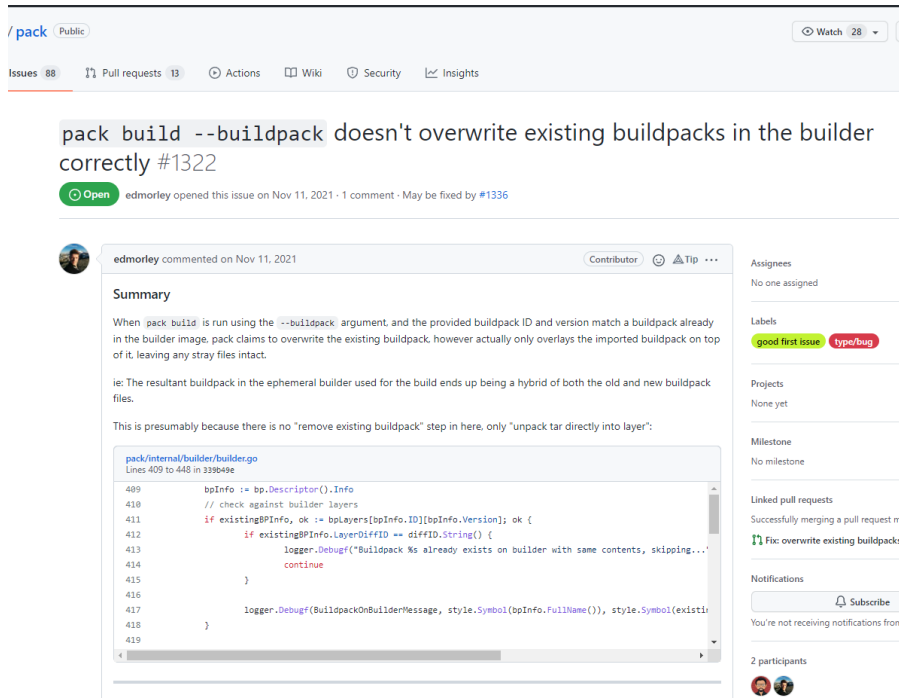
Görüldüğü üzere arada 17 MB bir fark mevcut, güvenli ve güvenilir bir imaj üzerinde Dockerfile dosyası oluşturulmadan imaj elde etmenin bir maliyeti olarak bence fazlasıyla kabul edilebilir bir fark. Ek olarak ileriye dönük düşünülürse ileride kendi base imajları ile imaj boyutlarını daha da küçültebilmenin de önü açılmaktadır.

### 3.6. Buildpacks PR

#### 3.6.1. Çalışmanın Hedefi

.NET Core Web Servisini Buildpacks ile Dockerfile olmadan bir şekilde imaj haline getirirken çeşitli buildpackler de denenmiştir. Daha önceden imaja dahil edilmiş buildpackleri --buildpack parametresi ile yeniden yazmaya zorlandığında tam anlamıyla yeniden yazmadığının ve ortaya eski ve yeni ile melez bir karışım çıkarttığı fark edilmiştir.

Buildpacks aracının kurum içerisinde ileride kullanacağı düşüncesiyle kritik bir sorun olduğu düşünülerek Buildpacks topluluğu ile etkileşime geçilmiştir. Aşağıdaki ekran görüntüsünden de görüleceği gibi daha önceden böyle bir sorunun oluşturulduğunu ancak iş kuyruğunda olduğu öğrenilmiştir.



Şekil 3.48. Buildpacks Issue

Geçtiğimiz yaz döneminde topluluk ile etkileşimde olduğumdan ve aracın kurum içerisinde ileride kritik bir yere sahip olacağı düşünüldüğü için topluluğa katkı yapmaya karar verilmiştir. #1322 numaralı sorunun kapatılmasıyla beraber buildpacklerin melez oluşturma durumunun engellenmesi beklenmektedir.



### 3.6.2. Yöntem

Docker, k8s gibi araçlar Go dili ile yazılmıştır. Bunların çevresinde gelişen Buildpacks gibi araçlarda doğal olarak Go dilinde geliştirilmişlerdir. Aslına bakılırsa CNCF altındaki birçok proje Go dili benimserek yazılmaktadır. Dolayısıyla Buildpacks projesinde bir sorunu kapatmak için Go diline hakim olmak gerekmektedir.

Hali hazırda zaten Go dili üzerinde algoritma soruları çözerek alıştırmalar gerçekleştirmiş olduğum için dil ile aramdaki bağ bir projeye katkı verebilecek seviyededir. Sorunu kapatmak için yaz döneminde bana mentörlük gerçekleştiren Javier ile iletişime geçerek ondan fikirler alınmıştır ve sorun için alternatif iki çözüm yöntemi için fikrini sunmuştur;

1. *Create a new image with the old buildpack layer excluded from the layers list.*

Yeni bir imaj oluşturup bu imajdan eski buildpack'in bulunduğu katmanı çıkartmak.

2. *Append a “whiteout” layer before appending the new buildpack layer.*

*Hazır üretilmiş imaja yeni bir beyazlık katmanı eklemek.*

İki fikir incelendiğinde birinci fikrin gerçekleştirilmesi daha kolay ve daha anlaşılır bir çözüm olduğu ortaya çıkmaktadır. Birinci çözümü gerçekleştirmek için var olan bir imajın açılarak katmanlarına ulaşılması ve bu katmanlar kullanılarak tamamen yeni bir imaj oluşturulması planlanmaktadır. Bu da tamamen gereksiz bir maliyet oluşturur.

İkinci fikir Linux üzerinde de bulunan Whiteout<sup>[11]</sup> yöntemini referans almaktadır. Whiteout anahtar kelimesi OCI tanımları içerisine de eklenmiş ve İngilizce olarak yer verilmiştir. Whiteout yöntemi şu şekildedir; İmaja yeni buildpack parametre olarak geçildiği zaman Buildpack API'leri ile daha önce eklenip eklenmediği kontrol edilir ve eğer ki eklendiyse eski katmanın bulunduğu dizine “.wh.” ayırıcısı ile bir whiteout katmanı eklenir ve bu katmandan sonra bu dizin sanki orada yokmuş gibi davranılır. Bu katman önceki katmanlarda ise yine eskisi gibi erişilebilir. Ancak imajlar halihazırda etiketlenmiş katmanlar oldukları için son katmandan erişilemeyecektir. Eklenen beyazlık katmanında sonra yeni buildpack katmanı eklenir ve işlemlere devam edilir.



Whiteout katmanının bir imaja eklenmesi hakkında Ahmet Alp BALKAN'ın blog<sup>[12]</sup> yazısında Go ile nasıl eklenebileceği hakkında bir blog yazısı yazmıştır. Bu blog yazısından yola çıkarak whiteout katmanlarının nasıl ekleneceği hakkında bir altyapı oluşturmak anlamlı bir yoldur. Buradaki yöntemi direkt olarak Buildpacks içerisinde kullanmak hem Buildpacks'in yapısının değişmesini sağlar hem de yeni bağımlılıklar içerdiği için PR'ın kabul edilme olasılığını yüksek oranda düşürür. Dolayısıyla bu yazıdan mantığı anlayarak Buildpacks içerisine kullandıkları yöntem ile dahil etmek gerekmektedir.

Buildpacks'de *pack* CLI için bu işlemleri gerçekleştirmek istenildiğinde *pack* deposunun *internal/builder/builder.go* dosyasında tüm buildpack dahil etme işlemleri yapıldığı için bu dosya içerisinde değişiklik yapılması gerekmektedir. Biraz inceleme sonrasında *builder.go* içerisinde *addBuildpacks* fonksiyonunun buildpackleri katmanlar haline getirmek için kullanıldığı farkedilmektedir. Burada beklenildiği gibi daha önceden API yardımıyla buildpack'in daha önceden dahil edilip edilmediği anlamak için bir karar yapısı eklenmiştir. Yapılması gereken bu karar yapısının içeriğini değiştirmektir.

Öncelikle her katman bir tar dosyası halinde sıkıştırıldığı ve bu formatta saklandığı için kod ile tar dosyası oluşturacak bir işlev gerekmektedir. Kodlar incelendiğinde tar dosyaları oluşturan spesifik işlevler mevcut olduğunu ancak genel anlamda bir dizin olarak geriye tar dosya yolunu döndüren bir işlev ile karşılaşmamaktadır. Tar dosyası yaratan *createTarball* işlevi aşağıdaki gibidir;





```
func createTarball(tarPath string, data map[string][]byte) error {
    tarFile, err := os.Create(tarPath)
    if err != nil {
        return err
    }

    defer tarFile.Close()

    tw := tar.NewWriter(tarFile)
    defer tw.Close()

    for name, content := range data {
        hdr := &tar.Header{
            Name: name,
            Mode: 0600,
            Size: int64(len(content)),
        }
        if err := tw.WriteHeader(hdr); err != nil {
            return err
        }
        if _, err := tw.Write(content); err != nil {
            return err
        }
    }
    return nil
}
```

Çizelge 3.16. Yeni eklenen createTarball fonksiyonu

createTarball işlevi tanımlandıktan sonra artık mevcut olan buildpackler için bir whiteout yolu içeren tarball oluşturarak oluşturulan tarball yeni bir katman olarak eklenmelidir. Daha sonrasında yeni buildpack sanki daha önce eski versiyonu eklenmemiş gibi işlem görebilmesi için buildpacksToAdd dizisine eklenmelidir. Yazılan kod parçaları kesinlikle eski yapıya uyumlu olmalıdır. Aşağıdaki kod parçası yardımıyla whiteout için /tmp dizini altında buildpackin API bilgilerinden yola çıkarak onun bulunduğu yolun bir whiteout kopyasını oluşturacaktır.

```
bpWhiteoutsTmpDir := filepath.Join(tmpDir, strconv.Itoa(i)+"_whiteouts")
    if err := os.MkdirAll(bpWhiteoutsTmpDir, os.ModePerm); err != nil {
        return errors.Wrap(err, "creating buildpack whiteouts temp dir")
    }
}
```

Çizelge 3.17. /tmp dizini altında whiteout katmanı için geçici dizin oluşturma

Aşağıdaki kod parçası yardımıyla silinecek yollar bir MAP içerisinde tutulur ve whiteout dizinlerden oluşturulacak olan tarball dosyasının yolu belirlenir. Ardından tarball oluşturulur ve whiteout katmanı olarak imaja dahil edilir.

```
deletedMaps := map[string][]byte{
    filepath.Join(
        buildpacksDir,
        strings.ReplaceAll(bpInfo.ID, "/", "_"),
        ".wh."+bpInfo.Version,
    ): {},
}
whiteoutsTarFile := filepath.Join(bpWhiteoutsTmpDir, "whiteouts.tar")

if err := createTarball(whiteoutsTarFile, deletedMaps); err != nil {
    return errors.Wrapf(err,
        "creating whiteout layers' tarfile for buildpack %s",
        style.Symbol(bp.Descriptor().Info.FullName()),
    )
}

if err := image.AddLayer(whiteoutsTarFile); err != nil {
    return errors.Wrap(err, "adding whiteout layer tar")
}
```

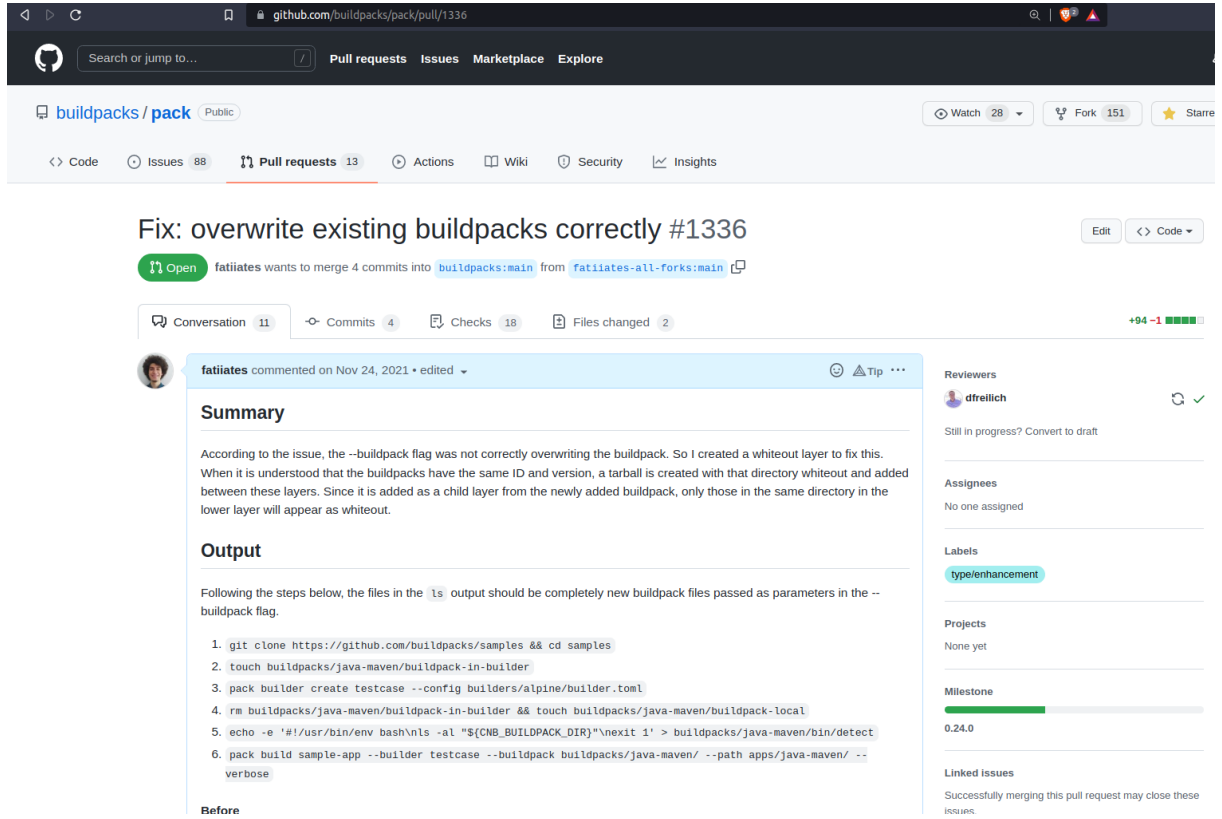
Çizelge 3.18. Yenilenen else bloğunun içeriği

Kodlar gösterildiği gibi yenilendikten PR kurallarına uygun olarak *pack* deposuna gönderilmiştir.

### 3.6.3. Sonuçlar

Hedeflendiği gibi melez oluşturulması durumu en az maliyet ve en uygun şekilde çözümlenmiştir. Artık bir buildpack yenilenmek istenirse bir melez oluşturmak yerine tam anlamıyla yeni buildpack kullanılabilir.

Projeye destek amaçlı olarak PR<sup>[13]</sup> gönderilmiştir. Kabul süreçleri konusunda commit imzalama gibi farklı istekler de giderilerek inceleyici ile Slack üzerinden ve PR üzerinden iletişime geçilmiş ve inceleyicinin onayını alma işlemi başarıyla gerçekleştirilmiştir. Şu an PR pack deposuna dahil edilebilir ve hazır bir halde ofis saatlerinde tüm ekip ile code review süreci gerçekleştirildikten sonra dahil edilmesi planlanmaktadır. Aşağıdaki ekran görüntüsünde inceleyicinin kabul ettiği görülebilir.



Şekil 3.49. Buildpacks PR



### 3.7. ZSH

#### 3.7.1. Çalışmanın Hedefi

Sürekli olarak bir işletim sistemi kurulmaktadır veya bir konteyner kaldırılmaktadır ve bu işletim sistemleri kesinlikle arayüz kurmadan kurulmaktadır. Dolayısıyla aktif bir şekilde Shell kullanılmaktadır. Bu kadar Shell kullandığımız için sürekli bir şekilde BASH üzerinden bir şeyler yazmak gerçekten anlamsız ve yorucu bir hal almaktadır. Bunun için geliştirilen bir Shell mevcut, ZSH. ZSH'nin eklenti sistemi ile yazdığım karakter sayısı büyük oranda azalma sağlayarak büyük oranda olumlu katkı gerçekleştirmiştir. Ek olarak neredeyse her türlü aracın ZSH eklentisi mevcut ve bu eklentiler ile kullanılması gerçekten işi çok kolaylaştırıyor. Örnek verilecek olursa; göreceli bir şekilde bir dizinin içerisine girdiğinizde onu tekrar yazmak yerine ayırıcı karakterleri yazarak ok tuşlarıyla çok kolay bir şekilde dizinin içerisine girilmesi verilebilir.

Bu projenin hedefi ZSH kullanıcılarını yormadan yalın bir ZSH kurulumu gerçekleştirmek ve yalnızca belirli eklentileri kullanacağımız ergonomik bir kurulum scripti geliştirmektir.



### 3.7.2. Yöntem

Genelde Debian tabanlı işletim sistemleri tercih ettiğimiz için şu an için yalnızca Debian desteği olan ufak bir script yazılacaktır. Ancak bu script macOS sistemlerinde de çalışabilir olacaktır. Script'in gerçekleştirmesi gereken şeyler şunlardır;

- 1- APT repolarından ZSH paketini kurması
- 2- Eklenen dosyaları \$HOME dizini içerisine kopyalaması
- 3- Varsayılan Shell'i değiştirmesi

Bu işlemler gerçekleştirildiğinde ZSH kurulmuş ve kullanılabilir halde olacaktır.

Bunun için varsayılan Shell olarak SH kullanacağım. Çünkü Debian bazlı sistemlerde zaten varsayılan olarak gelen bir Shell olarak yer almaktadır. Ek olarak macOS sistemlerde de bulunduğu için sorun olmayacaktır. Bunun için oluşturulan dosyanın başına aşağıdaki satırın eklenerek varsayılan olarak SH ile çalıştırılması gerektiği söylenmelidir.

```
#!/bin/sh
```

Daha sonrasında sistem güncellenmesi gerçekleştirilmelidir ve ardından ZSH paketi kurulmalıdır.

```
sudo apt-get update
```

```
sudo apt-get install zsh -y
```

Bu script ve yanındaki dosyalar bir tarball dosyası olarak dağıtılacağı için tar dosyasından çıkartıldığında kullanıcı yetkilerinin geçerli kullanıcıya atanması gerekmektedir. Çünkü tarballlar oluşturulurken yetkiler dahil tam olarak bir snapshot alınmış şekilde sıkıştırma gerçekleştirilir.

```
sudo chown -R $USER:$USER ./.*
```

Ubuntu için konuşulacak olursa yeni bir font eklemek için /usr/local/share/fonts/ dizini altına atıldığında font sisteme eklenmiş olur. Ancak bir sunucuya kuruyorsanız bu işlem gereksizdir. Çünkü sunucuya SSH ile bağlandığınızda oradan fonttan bağımsız bir şekilde yalnızca metinler gönderilir. Aşağıdaki komut yardımıyla Ubuntu için font ekleme işlemi gerçekleştirilir.

```
sudo cp ./font/'MesloLGS NF Regular.ttf' /usr/local/share/fonts/
```



ZSH paketi eklendiğine ve font dahil edildiğine artık sistemin varsayılan Shell'i ZSH olarak aşağıdaki komut yardımıyla atanabilir.

```
chsh --shell /bin/zsh $USER
```

Ve artık ZSH kurulumu tamamlanmıştır. Artık tar dosyası oluşturarak sürekli olarak erişilebilecek bir domain altında tutulacaktır. Bu domain için şu anlık kişisel bir domaini tercih ettik. Aşağıdaki komut yardımıyla tarball elde edilebilir.

```
tar -czf zsh.tar.gz ./zsh
```

Tar dosyası bir domain altına yerleştirildikten sonra aşağıdaki üç komut yardımıyla rahat bir şekilde ZSH kurulabilir.

```
wget http://domain.com/zsh.tar.gz  
Tar xf zsh.tar.gz && cd zsh  
./zsh.sh
```

Ancak görüldüğü üzere kullanıcı bazlı işlemler gerçekleştirdik. Bu kullanıcı bazlı işlemler dolayısıyla farklı bir kullanıcıya geçildiğinde görülecektir ki yine BASH veya herhangi varsayılan farklı Shell üzerinden devam edilecektir. Bu script özelinde yeniden ZSH kurulmak istenirse o kullanıcı için script tekrar çalıştırılmalıdır.

### 3.7.3. Sonuçlar

Script başarılı bir şekilde oluşturuldu ve her bilgisayarda sürekli olarak ZSH ve eklentilerini kurma zahmetinden kurtularak büyük bir vakit kazanılmıştır. Ek olarak oluşturduğumuz kodları bir tarball haline getirerek herkese açık bir domaine ekledik ve depolama araçları ile taşıma sorununu da giderdik. Çünkü SSH ile bağlandığımız yerlerde USB bağlamak veya bir dizini mount etmek epey yorucu olabiliyor. Hatta mount etmektense tekrardan eklentileri kurmak daha fazla zaman kazandırabilmektedir. Hedef başarıyla gerçekleştirilmiştir, gerçekleştirilmesinin yanı sıra oda içerisindeki personellere de ZSH kurularak onların da işleri kolaylaştırılmıştır.

ZSH ekran görüntüsü aşağıdaki şekildedir.



Şekil 3.50. ZSH görünümü



#### 4. ÇALIŞMA DÖNEMİNİN DEĞERLENDİRMESİ

Korona nedeniyle sürekli bir şekilde kongre, sempozyum gibi etkinlikler gerçekleştirilmiyor. Ancak eğitim anlamında iki farklı eğitim süreci geçirdim;

- 1- Docker Eğitimi:** Bu eğitim için UDEMY üzerinden bir eğitim satın alması gerçekleştirildi. Burada hem öğrenci hem öğretmen olarak görev aldım. Daha önceden Docker ile ilgilendiğim için bir öğrenci olarak kendi bilgileri tazeledim. Öğretmen görevi olarak ise yeni istihdam edilen bir personele tam anlamıyla Docker eğitimini vererek gerçekleştirdim.
- 2- Kubernetes Eğitimi:** Bu eğitim tamamen AR-GE çalışması olması için gerçekleştirildi. Belediye içerisinde Docker Swarm kullanılacağına zaten çok önceden karar verilmişti. Ancak hem kendimi geliştirmek hem de farkındalık yaratabilmek adına Kubernetes eğitimi alındı ve Kubernetes çalışmaları gerçekleştirildi. Bu eğitim tam anlamıyla tamamlanması uzun sürecek bir eğitim olduğu için ve İMEP döneminde %70 oranında tamamlanmıştır.

BTÜ-İMEP programıyla beraber tam anlamıyla açık kaynak alanında geliştiğimi gözle görülür bir şekilde hissetmeye başladım. Linux alanında ciddi bir farkındalık oluştuğunu veya Bursa ekosisteminin bu alanda ne kadar az bulunduğunu fark ettim. Bundan sonra da Linux ve açık kaynak alanında ilerlemeyle ilgili kafamdaki soru işaretlerini de gidererek tam anlamıyla bu yoldan ilerlemeyi tercih edeceğime de karar vermemde büyük bir katkısı olmuştur. İMEP programı boyunca hem iş hayatını hem akademik hayatı harmanlayarak elde ettiğimiz kazanımlarımız bizi diğer öğrencilerden öne çıkartıyor.

Birim içerisinde zaten halihazırda yenilikçi birçok çalışma gerçekleştiriliyor. Birim için verilebilecek en iyi tavsiye benim açımdan bakılacak olduğunda Windows ekosisteminden tam anlamıyla kopmaları olabilir. Söylemesi kolay olsa da bu alandan kopmak maalesef kolay gerçekleştirilebilecek bir aksiyon değildir. Bu yönde birçok adım zaten ben gelmeden atılmış daha da atılmaya devam ediyor. İleride bir gün zaten gerçekleşeceğine inanıyorum. İkinci bir tavsiye ise dışarıya açık kaynak projeler çıkarabilirler. Zaten Belediye olarak amaçları halka fayda sağlayacak işler yapmak olduğu için bu yönde de halkın yazılımcı kesimi ile birbirlerine karşılıklı –desteklenecek projeler ve buna karşın ücretsiz destek sağlayacak yazılımcılar ve öğrenciler- fayda sağlayabilirler.





Gelecek dönem firma ile çalışmak isterdim ancak çok plansız bir şekilde tercih gerçekleştirmek zorunda kaldım dolayısıyla şu an için maalesef devam edemeyeceğim. Geçtiğimiz dönem yoğun bir yaz dönemi geçirdim ve dolayısıyla İMEP için zaten okulun bize şirket bulacağını bildiğimiz için anlaşmalı bir şirket aramak yerine önümüzdeki dönemi planlamaya ağırlık verdim. Önümüzdeki dönem için kendim daha önceden bir planlama gerçekleştirmiştım ve bu planlamaya sadık kalmam gerektiğime inanıyorum.



## 5. KAYNAKLAR

- [1] **BBB Stratejik Planı.** Erişim: 23 Ocak 2022, [https://www.bursa.bel.tr/dosyalar/yayinlar/191011104504\\_0.0.0.BBB-2020-2024-Stratejik-Plani.pdf](https://www.bursa.bel.tr/dosyalar/yayinlar/191011104504_0.0.0.BBB-2020-2024-Stratejik-Plani.pdf)
- [2] **BBB İdari Yapı.** Erişim: 23 Ocak 2022, <https://www.bursa.bel.tr/idari>
- [3] **Setting Up The Git Server.** Erişim: 27 Ekim 2021, <https://git-scm.com/book/en/v2/Git-on-the-Server-Setting-Up-the-Server>
- [4] **The Protocols for Git Server.** Erişim: 27 Ekim 2021, <https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols>
- [5] **GitLab Dökümantasyonu.** Erişim: 29 Ekim 2021, <https://docs.gitlab.com/ee/install/docker.html>
- [6] **Moodle Dökümantasyonu.** Erişim: 02 Ocak 2022, [https://docs.moodle.org/311/en/Main\\_page](https://docs.moodle.org/311/en/Main_page)
- [7] **Bitnami Dökümantasyonu.** Erişim: 06 Ocak 2022, <https://github.com/bitnami/bitnami-docker-moodle/blob/master/README.md>
- [8] **Bitnami PR.** Erişim: 23 Ocak 2022, <https://github.com/bitnami/bitnami-docker-moodle/pull/195>
- [9] **Buildpacks Resmi Sitesi.** Erişim: 23 Ocak 2022, <https://buildpacks.io/>
- [10] **Paketo Resmi Sitesi.** Erişim: 05 Kasım Ocak 2021, <https://paketo.io/>
- [11] **Whiteout Definition in OCI.** Erişim: 10 Kasım 2021, <https://github.com/opencontainers/image-spec/blob/main/layer.md#whiteouts>
- [12] **Ahmet Alp BALKAN Blog Yazısı.** Erişim: 13 Kasım 2021, <https://ahmet.im/blog/building-container-images-in-go/>
- [13] **Buildpacks PR.** Erişim: 23 Ocak 2022, <https://github.com/buildpacks/pack/pull/1336>