Linux ağ yönetimi ve kurulumu

Linux network management and installation

Anisa Emin, Hasibullah Mohmand

Bilişim sistemleri Mühendisliği, Teknoloji Fakültesi,

Kocaeli Üniversitesi

[anisaaae04@gmail.com](mailto:anisaaae04@gmail.com) , [hasibullah.mohmand13@gmail.com](file:///C:\Users\anisa\AppData\Local\Microsoft\Windows\INetCache\IE\SMJ375PE\hasibullah.mohmand13@gmail.com)

Özet

*Linux ağ yönetimi, modern bilgi teknolojilerin altyapılarında kritik bir rol oynamaktadır. Bu çalışmada, Linux'ta ağ yönetimi ve kurulumu konusunu ele almakla birlikte, Docker (Linux tabanlı platform) ve onun fonksionlarının kullanmasiyle (containerisation) ağ oluşturma ve ağ güvenliği kavramlarını bir araya getirmektedir. Konusu* Linux ağ yönetimi ve kurulumu *olan bu projede, Linux ve Docker ağ teknolojilerini birleştirerek nasıl yalıtılmış, güvenli ve iyi yönetilen bir uygulama oluşturabileceğının gösterilmesi amaçlanmiştir. Bu bağlamda, iptables ve Docker Compose gibi teknolojiler kullanılarak farklı ağ senaryoları uygulanmıştır. Sonuç olarak, bu rapor hem teorik hem de pratik bir bakış açısıyla Linux ağ yönetiminin önemini vurgulamaktadır.*

Abstract

*Linux network management plays a critical role in modern IT infrastructures. This study addresses the issue of network management and installation in Linux and brings together the concepts of containerization and network security through the use of Docker (Linux based platform) and its networking functions. The Linux network management and installation project aims to show how to create an isolated, secure and well-managed application while combining Linux and Docker networking technologies. In this context, different network scenarios have been implemented using technologies such as iptables and Docker Compose. In conclusion, this report highlights the importance of Linux network management from both a theoretical and practical perspective.*

# Giriş

Linux, modern bilişim altyapılarında ağ yönetimi için vazgeçilmez bir araçtır. Gelişmiş güvenlik, esneklik ve açık kaynak topluluğundan sağladığı güçlü destek sayesinde, Linux hem bireysel hem de kurumsal ağ yöneticileri için en çok tercih edilen işletim sistemi haline gelmiştir. Docker ise, tek tek kod parçalarını izole edebilen ve bunların her ortamda çalışmasını sağlayabilen, Linux tabanlı güçlü bir teknoloji olduğu kanıtlanmıştır. Bu projede, Linux'un ağ yönetimini ele alarak, pratik örneklerle bu araçların nasıl kullanılabileceğini göstermek hedeflenmiştir. Çalışma kapsamında Docker teknolojisi kullanılarak ağ simülasyonları gerçekleştirilmiştir. Docker Compose, çok kapsamlı uygulamaların kolayca oluşturulmasına olanak sağladığından, Linux'ta ağ yönetimini göstermek için Docker Compose'u kullanmayı tercih ettik. Farklı hizmetler (konteynerler) arasında ağ oluşturma gibi karmaşık ağ yapılandırmalarını simüle edebilir, farklı bileşenlerin birbirleriyle nasıl iletişim kurduğunu test etmek için ağlar oluşturup yönetebiliriz. Bu Linux ağ yönetiminin temelidir.

Raporun ilk bölümünde Linux ağ yönetiminin temel prensipleri, iptables gibi araçlarla nasıl yönetildiğini ve ağ iletişiminin nasıl güvenli hale getirilebileceği açıklanmaktadır. İlerleyen bölümlerde ise Docker niye aslında bir Lınux ortamdır ve Docker Compose Projemiz ile birden fazla Linux konteynerinin nasıl bir ağ üzerinde çalışabileceği ve iptables kullanılarak trafik kontrolünün nasıl sağlanabileceği detaylandırılacaktır. Projemizin amacı ürün yönetimidir. Frontend kısmından yalnızca mevcut olan tüm ürünleri göstermektedir. Ön uç, ürünleri mongo veritabanından getirmek için arka uca istekler gönderir. Sipariş ekleme ve silme görevlerini yalnızca yöneticileri izin vermekte. Başka yerlerde kullanıcılar veritabanında hangi ürünlerin bulunduğunu görebilir. Bu süreçlerin gerçekte nasıl gerçekleştiği dikkate alınmıştır.

# Linux ağ yönetiminin temel prensipleri

Ağ yönetimlerini araştırmak için ağ arayüzleri, alt ağları, IP adresleri, iptables, TCP/IP protokolleri ve köprü ağların (bridge networks) ne demek olduğunu anlamamız lazım. Ağ arayüzü bir donanım ya da sanal bir bileşendir. Cıhazlar onu kullanarak bir ağa bağlanmak ve ağla iletişim kurmak yeteneğini sahip oluyorlar. Bu, cihaz yerel (LAN) ile küresel (İnternet) ağ arasındaki "temas noktasıdır".

Bizim projede docker konteynerleri bağlamak için docker tarafından oluşturulan sanal ağ arayüzleri kullanılmaktadır.

Ağ, veri alışverişi yapan bir grup cihazdır. Verilerin doğru yere gönderilebilmesi veya doğru yerden alınabilmesi için her cihazın veya bilgisayarın bir adı ve adresi (bir IP adresi) vardır. Yani bir cihazın nerede olduğunu ve diğer cihazların onu nasıl bulabileceğini gösterir. Veri alışverişi sırasında karışıklığı ve hataları önlemek için verileri, TCP/IP protokollerinde belirlenen belirli kurallar aracıyla gönderilir.

Bahsettiğimiz gibi her arayüzün genellikle kendisini ağda tanımlayan benzersiz bir IP adresi vardır. Veri aktarımında veriler paketler şeklinde gönderip alınmaktadır. Bu paketler kaynak, gönderci, alıcı ve verinin kendisi (örneğin bir web sayfası) hakkında bilgiler içerir. Arayüzler, bir cihazın aynı ağdaki ya da İnternet üzerinden diğer cihazlarla iletişim kurmasına yardımcı olur. Projemiz ağ trafiğini iptables üzerinden yönetiliyor. Iptables, Linux çekirdeği (kernal) tarafından sağlanan Netfilter çerçevesini temel alan bir araçtır. Yaptığı işlevler, gelen, giden ve iletilen paketlerin kabul edilip edilmeyeceğini belirleyerek paket filtrelemedir. Pojede aşağıdaki IP adreslerine uygulanan kurallardan biri 1.Şekilde gibidir:



*1.Şekil*

-A GİRİŞ: Gelen bağlantılar için bir kural ekler.

-s: Trafiğin kaynağını belirtir (makinenin IP adresi).

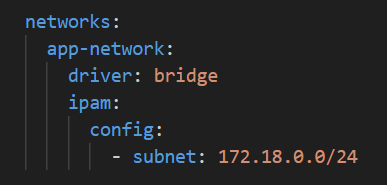
-d 0/0: Herhangi bir hedefe giden trafiğe izin veriyor.

-p tcp: TCP protokolüne uygulanır.

-m conntrack --ctstate YENİ, KURULU: Yalnızca yeni ve kurulu bağlantılara izin verileceğini belirtir.

-j KABUL: Gerçekleştirilecek eylem trafiğini kabul etmektedir.

Linux ağ yönetiminde diğer iki önemli kavram alt ağlar (subnetworks) ve köprü ağlardır (bridge networks). Alt ağlar ağ alanını daha küçük ve yalıtılmış bölümlere ayırır. Bu, güvenliği artırıp ağ trafiğini optimize eder. Alt ağlar bir alt ağ maskesiyle tanımlanır. Köprü ağları ise birden fazla arayüzün bağlanmasına olanak tanır, böylece cihazlar birbirleriyle sanki aynı fiziksel ağdaymış gibi iletişim kurabilir.



*2.Şekil*

2. Şekil Docker Compose proje dosyasında oluşturulan ağı gösterilmektedir. Linux'taki köprü ağına benzer bir köprü ağı (app-network) oluşturulur. Ayrıca Docker'ın kullanacağı IPv4 adres aralığını tanımlayan 172.18.0.0/24 ile bir alt ağ (subnet) oluşturulur. Bu, ağ adresi (ağ tanımlayıcısı) 172.18.0.0 ağda olduğu göstermektedir. Bu adres cihazlar (konteyneler) için kullanılmaz, ancak ağın kendisini ifade eder. "/24" bize ilk 24 bitin ağ kısmı (172.18.0) ve son 8 bitin ana bilgisayarlar (konteynerler) için olduğunu söyler. Bu nedenle ağ adres aralığı 172.18.0.1 ile 172.18.0.254 arasındadır. 172.18.0.0/24 ağındaki 172.18.0.255 adresi bir yayın adresidir (broadcast adress). Bu, ağdaki tüm cihazlara (konteynerlere) mesaj göndermek için kullanıldığı ifade ediyor. Dolasıyla, toplam 256 adres anlamına gelir, ancak iki tanesi yayın ve ağı temsil etmek için ayrılmıştır, bu nedenle bu ağda 254 aktif host (konteyner) bulunur.

# Docker Linux teknolojisi olarak

Docker Linux içinde “Linux” olarak görüntülünebilir. Docker Linux çekirdek özelliklerini kullanan bir konteynerleştirme platformudur. İşlemleri izole etmek ve konteynerleri oluşturmak için aşadaki Linux özellikleri kullanılmaktadır:

* Namespaces: İşletim sisteminin süreçleri, ağlari, dosya sistemlerini ve daha fazlası gibi farklı yönlerini izole ederler. Her konteynerin süreçler, ağlar vb. için kendi alanı vardır ve bu da onun ayrı bir makine gibi görünmesini sağlar.
* Cgroups (Control groups): Kapların kullanabileceği kaynakları (CPU, RAM, disk alanı) sınırlandırırlar.
* Union File Systems (OverlayFS): Docker, katmanları oluşturmak ve kapsayıcıları hafif ve verimli hale getirmek için bu tür dosya sistemini kullanır.

Her Docker konteyneri ayrı bir Linux makinesi gibi görünür ve davranır, ancak aslında ana bilgisayarın işletim sisteminin çekirdeğini paylaşır. Bu, tüm işletim sistemini taklit eden sanal makinelerden çok daha hafif bir çözümdür.

## "Bireysel Linux makineleri" olarak konteynerler

Docker'daki konteynerler birbirlerinden ve ana sistemden yalıtılmıştır. Ancak aynı ağa bağlı oldukları durumunda birlikte çalışabilirler. Bu izolasyon ve paylaşılan bir ağ üzerinde çalışabilme yeteneği, Docker'ın ağ altyapısını simüle etmek için ideal kılar.

* Her konteynerin kendi kök alanı, kendi süreçleri ve ağ arayüzü vardır. Bu onun küçük bir Linux makinesi gibi görünmesini sağlar.
* Paylaşılan ağ erişimi: Docker'da bir ağ oluşturduğunuzda, konteynerler sanal IP adreslerini kullanarak birbirleriyle iletişim kurabilir.

## Konteynerler arasında ağ simülasyonu

Docker, konteynerleri aynı sanal ağa bağlayarak aralarındaki ağı simüle etmenize olanak tanır. Bu, ağ ayarlarını, güvenliğini ve uygulamalarını test etmek için kullanışlıdır. Docker, konteynerlerin birbirleriyle ve dış dünyayla nasıl iletişim kurduğunu kontrol eden çeşitli ağ modları sağlar. Bu modları anlamamız, konteynerli uygulamalarda ağ iletişimini yapılandırmak için önemlidir; özellikle de proje iptables, güvenlik duvarı vb. gibi Linux ağ teknolojileriyle çalışmayı içerdiği için.

# Docker Ağ Modları ve Linux Ağ Teknolojileri

## Bridge network

### Bu mod Docker'da varsayılan modudur. Köprü ağı kullanan konteynerler, bir ana sistem içerisinde birbirleriyle dahili bir ağ (köprü ağı) aracılığıyla iletişim kurabilirler. Konteynerlere ulaşmak için harici trafiğin Ağ Adresi Çevirisi'nden (NAT) geçmesi gerekir.

### Konteynerlerin birbirleriyle iletişim kurmak için kullanılabilecek dahili IP adresleri vardır. Konteynerler dış dünyaya bağlanmak için ana bilgisayarın ağ arayüzünü kullanırlar ve Docker, bağlantı noktalarını ana bilgisayardan konteynere otomatik olarak iletir.

Ana sistemde iptables veya güvenlik duvarı kullanıldığında, NAT (Network Address Translation) kullanılarak konteynerlere giren veya konteynerlerden çıkan trafiği kontrol etmek için ağ kuralları uygulanabilir. Bu, harici kaynaklardan kapsayıcılara erişimi kısıtlamak için kullanışlıdır.

## Host network

## Ana bilgisayar ağını kullanırken konteyner, ana sistemin ağ arayüzünü kullanır. Bu, konteynerin kendi sanal ağına sahip olmadığı ve ana sistemle aynı IP adreslerini ve ağ arayüzlerini kullanacağı anlamına gelir.

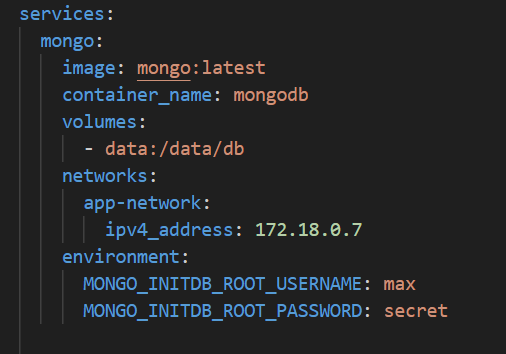
## Konteyner iletişim kurmak için NAT veya dahili bir ağ kullanmayacak, ancak ana bilgisayarın ağ altyapısıyla doğrudan iletişim kuracaktır.

Linux ağ teknolojiler ile nasıl çalışır?

Ana bilgisayar ağını kullandığımızda, konteyner ağdan yalıtılmış değildir, dolayısıyla iptables gibi Linux ağ araçları, ana bilgisayardaki diğer işlemler gibi konteynerle çalışabilecektir. Bu, ana bilgisayarın ağ altyapısına tam erişim sağlar.

Bu, kapsayıcıların ağ izolasyonuna ihtiyaç duymadan ana bilgisayar ağına doğrudan erişime sahip olması istediğinizde yararlı olabilir.

Projemizde bridge network kullanmaktadır. Dockerde bir network oluşturmak için **docker network create app-network** komutu kullanılır. Default olarak bridge network oluşturmakta. Oluşturulan bu ağa bağlı olan tüm konteynerler birbirleriyle iletişim kurabilir. Bir konteynerin hangi ağa bağlanması gerektiğini belirtilerek, bu konteynerin bu ağa hangi IP ile kaydedilmesi gerektiği de belirtilebilir. Projede mongo kontainer oluştururken uygulanan komutlar 3. Şekilde gösterilmektedir.



*3.Şekil*

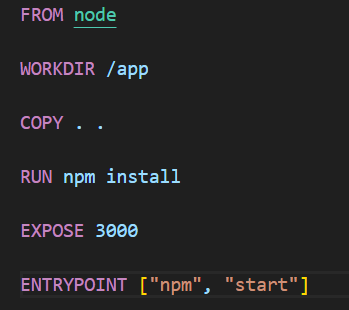
## None network

## Bir konteyner hiçbir ağı kullanmadığında ağ bağlantısı yoktur. Bu mod, diğer konteynerlerle veya dış dünyayla iletişim kurması gerekmeyen konteynerler için kullanışlıdır.

## Bu konteynerler tamamen izole edilebilir ve trafik gönderip alamazlar. O yüzden bizim projemizde bu mod kullanılmamaktadır.

# Docker ve Konteynerleştirme Temelleri

Konteynerleştirme, bir uygulama veya hizmetin bir konteyner olarak paketlendiği bir yaklaşımdır. Her konteyner görüntüden (image) oluşturulur. Görüntü (Image), bir konteyner oluşturmak için bir dizi talimat içeren salt okunur bir şablondur. Bağımlılıkları ve yapılandırmasıyla paketlenmiş yazılım içerir ve sanal bir ortamda çalışmak üzere tasarlanmıştır. Konteyner, bir görüntünün çalıştırılabilir bir örneğidir. Özel bir görüntü oluşturmak için dockerfile oluşturuyoruz. Dockerfile derleme talimatları içerir. Bu talimatlar bir ara görüntü oluşturur. Gelecekteki derlemelerin süresini azaltmak için önbelleğe alınabilir. docker build komutuyla kullanılır. Bir kaynak kodunu derlemek gibidir. Projemizde frontend ve backend containers için Docker file oluşturmakta. Aşağdaki örnekte frontendin Dockerfıle gösterilmiştir:



Frontend Dockerfilede kullanılan temel talimatlar şunlardır :

**FROM** – başka bir görüntüden bir görüntü oluşturur (çoklu aşamalandırmayı destekler). Her FROM yeni bir aşama başlatır. Oluşturacağımız görüntü, düğüm görüntüsüne dayanır.

**WORKDIR** – dosyalarımızın bulunduğu görüntünün çalışma dizinidir. Örnekte Docker'a kapsayıcının içindeki geçerli dizini /app olarak değiştirmesini söyler. WORKDIR /app'i izleyen tüm komutlar (örneğin, RUN, ENTRYPOINT, COPY, vb.) /app'e göre yürütülür.

**RUN** – komutu yürütür.

**COPY** – görüntüdeki farklı dosyaları kopyalar. ". ." komutuyla ilk Nokta (.) ana makinemizdeki kaynak dizini (Dockerfile'ımızın bulunduğu dizin, aynı zamanda derleme bağlamı olarak da adlandırılır) temsil eder.

İkinci Nokta (.) kapsayıcının içindeki hedef dizini temsil eder. Bizim durumumuzda WORKDIR'i bu dizine göreli olacak şekilde ayarladık; aksi takdirde /'e (kapsayıcının köküne) görelidir.

**ENTRYPOINT** – kapsayıcıyı hangi komutun başlattığını tanımlır.

# Docker compose dosyası oluşturma

Bazı uygulamalar birden fazla bileşeni birleştirir. Projemiz ön uç (Frontend), arka uç (Backend), veritabanı ve kullanıcıları birleştirir. Her bileşen ayrı bir kapsayıcıda çalışabilir. Docker Compose, birden fazla bağlı kapsayıcıyı çalıştırmak için kullanılır. Docker Compose uygulamasını çalıştırmak için docker compose.yaml dosyasını yazmamız gerekir. Docker-compose.yml dosyası başlatılacak konteynerleri, kullanılacak servisleri ve ağ kurallarını açıklar. Ayrıca servislerimizi oluşturur ve başlatır ve servislerinizi yönetir. Ek b'de ekli, projenin docker compose oluşturma dosyasının bir parçasıdır.

# Proje mimarisi

Proje, kullanıcı rolü, erişim kısıtlaması ve ağ güvenliğini sağlamak için Linux araçlarının kullanılması gibi kavramları gösteren bir görev yönetim sistemidir. Mimari, belirli rolleri oynayan ayrı kaplar oluşturmak için Docker Compose'u kullanır. Mimari modüler olup, yeni kullanıcı rollerinin veya işlevlerin kolayca eklenmesine olanak tanır. Konteynerler şu şekilde tanımlamaktadır:

1. Mongo DB – veri tabanlı kontayner. Image: mongo:4.4. Veritabanına erişim, ortam değişkenleri aracılığıyla belirlenen kullanıcı adı ve şifre ile korunmaktadır. Uygulama ağına 172.18.0.7 sabit IP adresiyle bağlanıldı.
2. Backend- server logic. Image: Uygulamayı oluşturan özel bir Dockerfile kullanır. İş mantığını ve ön uçtan veya kullanıcılardan gelen istekleri yönetir. 80 numaralı bağlantı noktasında çalışır ve uygulama ağına bağlıdır. Belirli ağ güvenliği kuralları iptables kullanılarak ayarlanır: Yalnızca 172.18.0.4 (ön uç) ve 172.18.0.6 (yönetici) sabit IP adreslerine sahip kullanıcılardan gelen isteklere izin verir. 80 numaralı bağlantı noktasından gelen diğer TCP isteklerini engeller.
3. Frontend - Kullanıcı etkileşimi için bir arayüz sağlar. Özel bir Dockerfile kullanır. Uygulamaya 3000 numaralı bağlantı noktasından erişilir. 172.18.0.4 sabit IP adresiyle uygulama ağına bağlanır.
4. User containers (admin, user1) - Sistemdeki kullanıcıları simüle edin. İsteklerin arka uca gönderilmesine izin veren uygun/kıvrık görüntüyü kullanırlar. Her kapsayıcı, uygulama ağına sabit IP adresleriyle bağlanır: kullanıcı1 (sıradan kullanıcı) - 172.18.0.5. yönetici (yönetici) - 172.18.0.6. Bu konteynerlerin konfigürasyonunu kopyalayarak yeni kullanıcıları kolaylıkla eklemek mümkündür.

# Projeyi kurma ve projeyle çalışma adımları

Kurulum işlemini kolaylaştırmak için ek A'ya gerekli tüm kurulum ve başlatma adımlarını otomatikleştiren özel bir install.sh betiği eklenmiştir. install.sh kullanma şekli aşadaki adımlar göstermektedir:

1. Install.sh dosyasını çalışacağı dizinde oluşturur ve bu dosya executable dönüştürür.

Sonra GitHub deposu URL'sini sağlayarak betiği çalıştırır:

Örneğin: gedit install.sh

chmod +x install.sh

1. ./install.sh https://github.com/AnisaaE/Add-Task---Docker.git
2. Bir görevi gönderme veya silme isteği, arka uç konteynerinin terminalinde oturum açılarak gerçekleşir.

POST request - curl -X POST http://backend:80/products -H "Content-Type: application/json" -d '{"text": "New product"}'

DELETE request - curl -X DELETE [http://backend:80/products/{productId}](http://backend:80/products/%7bproductId%7d)

GET all request - curl <http://backend:80/products>

Docker compose çalıştırdıktan en çok kullanılan ve gereklı komutlar ağadaki olanlardır:

1. **sudo systemctl start docker** – bu komut docker deamon yani docker engine başlatıyor. Her zaman sistemi açtığımızda docker engine de açmamız gerekir. Bunu her zaman yazmak istemiyorsak **sudo systemctl enable docker** yazdığımızda docker engine automatik olarak çalıştırır.
2. **docker ps** – hangi kontaynerlerin çalıştığnı gösteriyor ve onlari bilgi vermekte.
3. **docker compose up** – bütün docker composdaki kontaynlerleri çalıştırıyor.
4. **docker start \*kontaynerin adı\*** - bir tane kontayneri çalıştırıyor.
5. **docker logs \*kontaynerin adı\*** - kontaynerın çıkışları göstermektedir.
6. **docker network ls** – var olan ağlari gösterir.
7. **docker netwrok create \*network\*** – yeni ağ oluşturur.
8. **docker network inspect** \*ağının adı\* – ağa bağlan bütün konteynerleri kontrol ediyor.

# Proje geliştirme sırasındaki zorluklar

1. Arka uç (Backend) veya ön uç (Frontend) kodu yerel olarak çalışıyordu ancak Docker'da çalıştırıldığında hatalar gösteriyordu.

**Çözüm:** Docker dosyalarındaki yolları ve bağımlılıkları kontrol ettik ve hata ayıklama için docker günlüklerini kullandık.

1. Docker, Node.js, MongoDB veya kullanılan diğer araçların sürümleri arasındaki uyumsuzluk çıkıyordu.

**Çözüm:** Dockerfile veya docker-compose.yml'de belirli sürümleri belirterek sıralı sürümleri kullandık.

1. İptables doğrudan Docker comse dosyasında ayarlanamadı.

**Çözüm:** Docker'ı başlattıktan sonra, kuralları arka uç konteynerinin terminaline manuel olarak yazmamız gerekir.

1. Konteynerlerin her başlatılmasından ve durdurulmasından sonra, girilen veriler her seferinde kayboluyordu.

**Çözüm:** Her açılışta hiçbir verinin kaybolmaması için Volumes'u kullandık.

# Sonuç

"Linux Ağ Yönetimi ve kurulumu" projesi, ağ ortamlarını

simüle etmek ve yönetmek için Linux sistem yönetimi ve Docker kullanımına ilişkin bilgileri uygulama sağladı. Çalışma sırasında ağların oluşturulması, yönetilmesi ve korunmasına yönelik temel kavramlar ve araçlar dikkate alındı ​​ve uygulandı; bu, bunların pratik yönlerinin derinlemesine anlaşılmasına katkıda bulundu.

Docker ve Docker Compose kullanılarak, özel olarak yapılandırılmış bir ağ üzerinden birbirleriyle iletişim kuran çeşitli hizmetlerden oluşan eksiksiz bir ortam oluşturuldu. Aşağıdaki gibi temel unsurlar ele alındı:

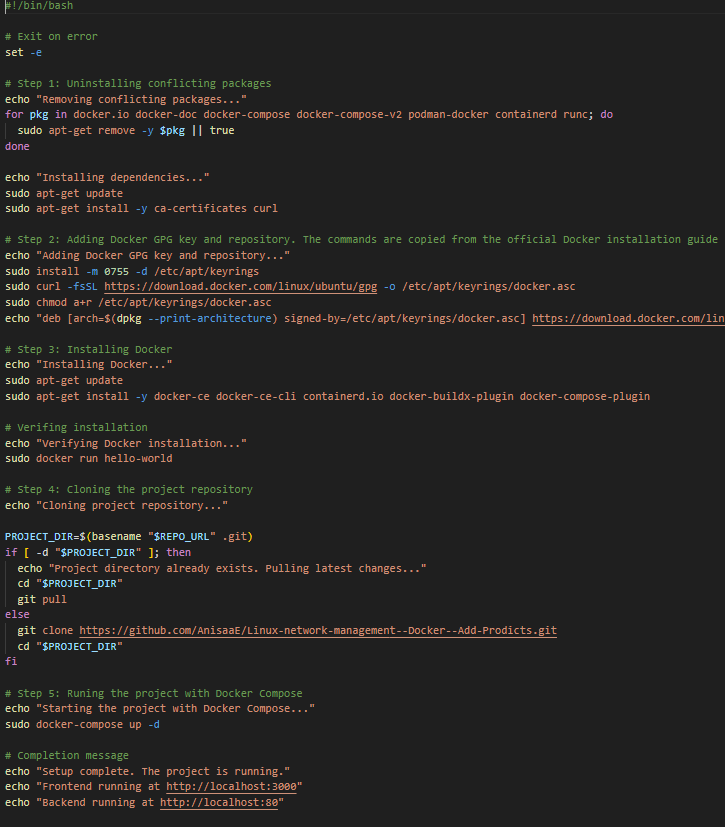
* Docker Compose ve köprü ağlarını kullanarak ağları yapılandırma.
* Kapsayıcılar kullanarak farklı rollere ve IP adreslerine sahip kullanıcıları simüle etmek.
* Hizmetlere erişimi kısıtlayan iptables kuralları ayarlayarak ağ güvenliği.

Bu proje, Linux'un ağ yönetimine ilişkin teknik yeteneklerini göstermektedir.

# Kaynaklar

1. **Docker Documentation**. (n.d.) : <https://docs.docker.com/>
2. Command line tool in Network Manangment in Linux - <https://www.geeksforgeeks.org/command-line-tools-and-utilities-for-network-management-in-linux/>
3. **Docker Hub**. (n.d.). Official Docker Images: <https://hub.docker.com/>
4. **Bash Manual**. (n.d.): <https://www.gnu.org/software/bash/manual/>
5. Docker Ubuntu - https://docs.docker.com/engine/install/ubuntu/#uninstall-docker-engine

**Ek A**

****

**Ek B**

