

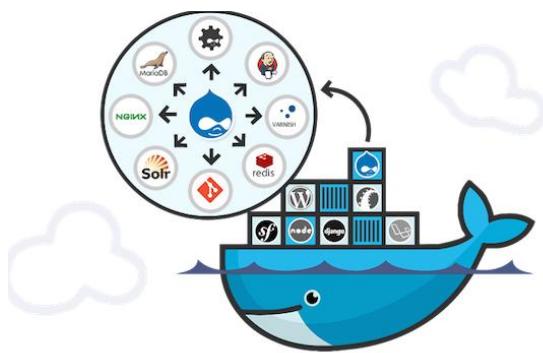


## İçerik

<b>1.</b>	<b>Docker Nedir.....</b>
<b>2.</b>	<b>Kontainer Çalışır .....</b>
<b>3.</b>	<b>Kontainer ve Sanallaştırma.....</b>
<b>4.</b>	<b>İmaj Nedir.....</b>
<b>5.</b>	<b>Docker Kurulumu .....</b>
<b>6.</b>	<b>Docker Images.....</b>
6.1	Docker images komutları.....
<b>7.</b>	<b>Docker Network.....</b>
7.1	Docker Network Komutları.....
<b>8.</b>	<b>Docker Volume.....</b>
<b>9.</b>	<b>Docker Temel Komutları.....</b>
9.1	Docker Run.....
9.1.1	Docker Deadmon Nedir ..
9.1.2	Docker Run - Tag .....
9.2	Docker ps.....
9.3	Docker rename.....
9.4	Docker Container Silme.....
9.4.1	Docker rm.....
9.4.2	Docker container prune .....
9.4.3	Docker rmi.....
9.5	Docker version.....
9.6	Docker info.....
9.7	Docker pull.....
9.8	Docker exec.....
9.9	Docker sleep.....
9.10	Docker stop & start & restart .....
9.11	Docker pause & unpause.....
9.12	Docker inspect.....
9.13	Docker logs.....
9.14	Docker top.....
9.15	Docker stats.....
9.16	Docker commit.....
<b>10.</b>	<b>Grafik Arayüz ile Docker Yönetim Tolları.....</b>
10.1	Kitematic.....

10.2 Portainer.....
<b>11. Docker Registry.....</b>
<b>12. Docker Trusted Registry.....</b>
<b>13. Container üzerine data aktarma.....</b>
13.1 Copy.....
13.2 Mount Volume.....
13.3 Internet.....
<b>14. Jenkins Container Örneği.....</b>
<b>15. Docker File.....</b>
15.1 Docker File Komutları.....
15.1.1 SHELL & EXEC.....
15.1.2 FROM.....
15.1.3 RUN.....
15.1.4 ADD.....
15.1.5 COPY.....
15.1.6 WORKDIR.....
15.1.7 VOLUME.....
15.1.8 EXPOSE.....
15.1.9 LABEL.....
15.1.10 USER.....
15.1.11 END.....
15.1.12 CMD.....
15.1.13 ENTRYPOINT.....
15.2 Docker File Örnekleri.....
15.3 VisualStudio Code ile Docker File.....
<b>17. Docker Compose.....</b>
17.1 Docker Compose Kurulumu.....
17.2 Docker Compose Komutları.....
17.3 Docker Compose Örnekleri.....
<b>18. Docker Swarm.....</b>
18.1 Docker Swarm Komutları.....
18.2 Docker Swarm Kurulumu.....
18.3 Docker Swarm Üzerinde Servis Oluşturma.....
18.4 Docker Swarm Monitor Tool.....
18.5 Docker Swarm Overlay Network.....
18.6 Docker Swarm Rolling Update.....
<b>19. Docker Stack.....</b>
19.1 Docker Stack Komutları.....
<b>20. Docker Secret.....</b>
20.1 Docker Secret Komutları.....
20.2 Docker Secret Örnekleri.....

## Docker Nedir ?



Docker üzerinde bulundurduğu her bir container'ını, basitçe uygulamaları(pyhton,nginx,wordpress,mysql,linux,mongoDB,Jenkins ...) üzerinde yalıtılmış bir şekilde çalıştırabileceğimiz bir container yönetimidir.

Docker'dan günümüzde ihtiyacımızı nasıl karşıladığımıza bir örnekle anlatmaya çalışalım.

Bir web sunucusu ayağa kaldırılmaya çalıştığımızda

Nodejs kullanan bir web sunucusu ve mongoDB veritabanı, redis ile mesajlaşması sistemi gibi altyapıyı sağlayacak birden fazla sunucu kurulum ihtiyacı söz konusudur.

Docker burada devreye girerek, Docker sunucu üzerinde her bir uygulamayı docker kütüphanesinden elde ederek bu uygulamaları container olarak çalışmamıza saniyer içinde ayağa kaldırırmamıza imkan sağlar.

## Kontainer Nedir



Konteynerler tamamen yalıtılmış ortamlardır.

Aynı işletim sistemi çekirdeğini paylaşmaları dışında, sanal makineler gibi kendi süreçlerine veya hizmetlerine, kendi ağ arayuzlerine ve kendi bağlantılarına sahip olabilirler.

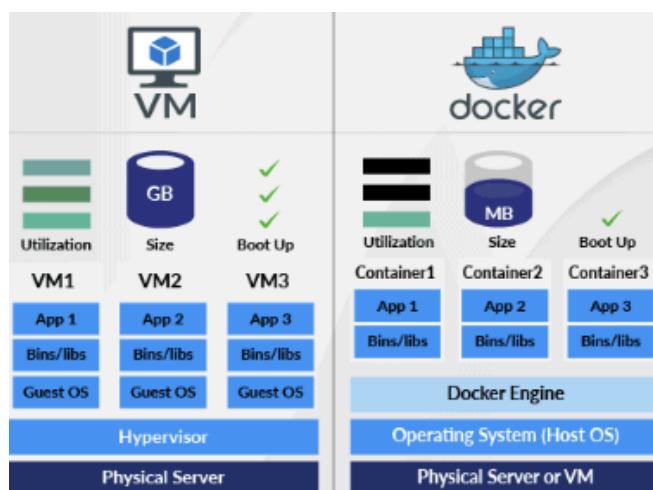
Temel işletim sistemi Ubuntu ise Docker, Debian fedora, suse veya centos gibi başka bir dağıtıma dayalı bir konteyner çalıştırılabilir.

Her docker konteyneri, bu işletim sistemlerini farklı

kılan ek yazılımlara sahiptir ve docker, yukarıdaki tüm işletim sistemleriyle çalışan docker ana bilgisayarının temel çekirdeğini kullanır.

Örneğin IIS, Windows DNS gibi işletim sistemi kernel windows olan bir container yaratmak istersek, Windows tabanlı bir konteyneri Linux yüklü bir Docker ana bilgisayarında çalıştırıramazsınız, bunun için windows sunucusu üzerinde bir Docker'a ihtiyacınız olacaktır. Docker'in temel amacı, uygulamaları paketlemek ve konteyner haline getirmek, bunları göndermek ve istediğiniz yerde istediğiniz kadar çalıştırılmaktır, bu da bizi sanal makineler ve konteynerler arasındaki farklara getirir.

## Kontainer vs Sanallaştırma



Resim üzerinden inceleyeceğiz olursak,

Baremetal üzerinde kurullu bir hipervizör ve onun üzerinde yönettiğimiz sanal makineler, her sanal makinenin içinde kendi işletim sistemi vardır.

Daha sonra bağımlılıklar ve ardından uygulama ek yükü, birden fazla sanal işletim sistemi ve sanal makineleri çalıştıran çekirdek olduğu için temel kaynakların daha yüksek kullanımına neden olur, ayrıca her VM ağır olduğundan

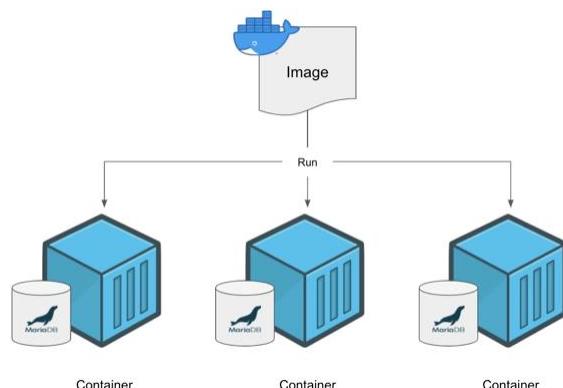
ve genellikle gigabayt boyutunda olduğundan bu alanı daha fazla tüketir,

oysa docker konteynerleri hafiftir ve genellikle megabayt boyutundadır.

Docker'in çekirdek gibi konteynerler arasında daha fazla kaynak paylaşıldığından daha az izolasyona sahip olduğunu, VMS'nin ise alta yatan işletim sistemi veya çekirdeğe dayanmadığından birbirinden tamamen izole olduğunu bilmeliyiz.

Bir proje mimarisinde, ihtiyacımız olan DB Sunucusu, Web sunucusu, Yönetim paneli gibi bir çok uygulamaya sahip mimariyi ayrı işletim sistemleri üzerinde ayağa kaldırırmak saatlerimizi belki günlerimizi alırken, Containerlar üzerinde çok hızlı bir şekilde tümleşik yapıyı ayağa kaldırırmak mümkündür.

## İmaj Nedir



Üzerinde çalışmış olduğumuz her container bir imajdan oluşur. Docker hostumuz üzerinde her container ayağa kaldırma aşamasında istediğimiz imajı docker hostumuzda arar eğer yoksa docker.hub üzerinden ilgili imajı docker hostumuz üzerine indirir. Bu imajlar üzerinde çalıştığımız uygulamalarımızın bir şablon hali olduğunu söyleyebiliriz.

Bu bir container oluşturduğumuzda bu şablon üzerinden uygulamayı çalıştırır ve ayağa kaldırır. Uygulama üzerinde düzenleme ve konfigürasyonlar yaptıktan sonra bu uygulamayı bir şablona dönüştürmek mümkün. bunu Docker Hub Deposuna göndererek herkese açık hale getirebilir istedigimiz zaman istediğimiz docker host üzerinde containerimizi ayağa kaldırabiliriz.

## Docker Kurulumu

Docker kurulumunu Mac/Windows/Linux işletim sistemlerince desteklemektedir. Product ortamlarda linux üzerine docker sunucuları kullanılmaktadır. Windows işlem sistemlerince docker kernel'lı kullanmaktadır, Biz windows docker üzerinde bir containerları çalıştırıldığımızda arka planda windows sanal linux sunucu çalıştmakta ve servislerini kullanmaktadır. Windows üzerine kurmadan önce hyper-v aktif olduğundan emin olun. Ve windows üzerine kurumlarda sanallaştırmın aktif olması gerekmektedir.

Biz ubuntu üzerine bir kurulum çalışması yapacağız.

1. Script ile kurulum için <https://get.docker.com> gidip adreste yer alan script ile kurulum yapabilirsiniz.

```
# $ curl -fsSL https://get.docker.com -o get-docker.sh
# $ sh get-docker.sh
```

```
>sudo usermod -aG docker ubuntu
>sudo chown ubuntu /var/run/docker.sock
>reboot
```

```
root@docker2:/home/ubuntu# curl -fsSL https://get.docker.com -o get-docker.sh
root@docker2:/home/ubuntu# ls
get-docker.sh
root@docker2:/home/ubuntu# sh get-docker.sh
# Executing docker install script, commit: 4f282167c425347a931ccfd95cc91fab041d414f
```

2.Manuel Kurulum için [Install Docker Desktop on Linux | Docker Documentation](#) adresine gelip işletim sistemi versionunuza göre ilerleyip, aşağıdaki gibi yönlendirmeler dahilinde kurulumu gerçekleştirebilirsiniz.

Ubuntu üzerine docker kuracağız

```
sudo apt-get update
```

```
sudo apt-get install \
ca-certificates \
curl \
gnupg \
lsb-release
```

```
sudo mkdir -p /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
```

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

```
>sudo usermod -aG docker ubuntu
```

```
>sudo chown ubuntu /var/run/docker.sock
```

```
>reboot
```

Artık docker kullanmaya başlayabiliriz.

3. Olarak eğer docker kurabileceğiniz bir ortamınız yok ise, docker bunuda düşünmuş bir lab platformu yaynlamış. İlerleyen bölümlerde ihtiyac ortamına göre bu alanında kullanıyor olacağım. Docker hesap bilgilerimiz ile giriş yaptıktan sonra kullanabiliriz.

<https://labs.play-with-docker.com/>

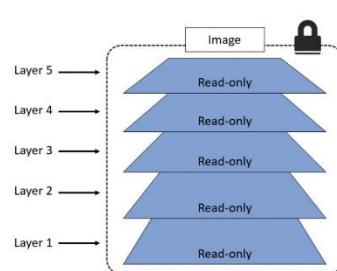
## docker images

Docker imaj container oluşturmak için salt okunur uygulama, işletim sistemleri şablonlarıdır.

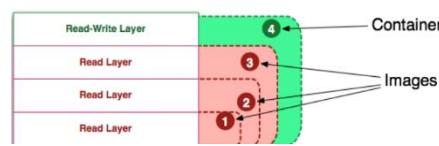
Containerlar imajlara bağlı olarak çalışır, bir imajı silmek için önce bağlı olduğu containeri silmelisiniz.

Dockerfile kullanarak imaj oluşturulmaktadır.

Tek bir imaj dosyasıyla birçok container oluşturabiliriz, Containerların oluşturulması yada silinmesi imaj dosyasını etkilememektedir Docker hostunuzda bir imaj indirdiğinizde UNIC bir ID tanımlanmaktadır



İmajlar katmanlardan oluşmaktadır. Bir python container çalıştırduğumda bir işletim systemin çalışacağı için 1. Katmanı os, 2. Katmanı python uygulaması olarak nitelendirilir. Bunların bütündede imaj deriz.



Salt okunur katmanlarından sonra, siz imaj kullanarak bir container oluşturduğunuzda üzerine okunabilir-yazılabilir bir katman ekler. Container üzerinde yaptığınız işlemler bu katman üzerinde işlem yapmaktadır.

Eğer sil bir container silerseniz bu katmanlardaki kendi oluşturulan okunabilir-yazılabilir katmanı silinecektir.

İmaj oluşturulken her katman SHA256 ile şifrelenmektedir

docker images komutu ile docker hub kütüphanesinden docker sunucumuza indirdiğimiz imajları listelemek için kullanırız.

### Docker images komutları

>`docker image ls` (localdeki imajları listeler)

>`docker image rm "imaj"` (localdeki imajları siler)

>`docker image prune` (local indirilen tüm imajları siler)

>`docker image inspect "imaj"` (imaj detaylarına listeler)

Writer: @HalilGÖKSEL

**>docker image pull** (dockerhub üzerinden imaj indirir docker pull ile aynı işlev sahiptir)

**>docker image login / logout** (docker hostumuzda dockerhub hesabımıza oturum açmak için kullanılır, oturum sonrasında imajlarımızı push edebiliriz.) Çıkış yapmak için docker logut komutunu kullanabiliriz.

**>docker image push** dockerhub hesabımıza imaj gönderir

**>docker image commit** (containerilerimizi imaj haline dönüştürmek için kullanılır kullanımı “docker image commit “container adı” “repostiyory adı/imajı adı” )

**>docker image tag** (tag verilmemiş containerlarımıza tag belirlemek yada tag değiştirmek için kullanılır kullanımı “docker image tag “image ID” “repostroy adı/imaj adı” )

Docker image history (imajların katman detaylarının çıktısını verir)

Docker search (docher.hub üzerine imaj veya kullanıcı sorgulaması için kullanılır.)

Ör:

**>docker login** (dockerhub user / password)

**>docker image push halilgoksel/myweb** “container id”

Dockerhub ortamımıza imaj yüklemek için bir örnek yapalım, yaptığımız işlemler sırayla aşağıdaki gibi,

- 1.docker.hub web adresine gidip halilgoksell/myjen repo'su oluştur.
2. docker host üzerine jen adında volume map'li ve port maplı Jenkins container oluştur.
- 3.containerı imaja dönüştür.
4. docker.hub'a login ol
5. Oluşturduğun docker imajını push et
6. Docker.hub namespace olarak domainin search et altındaki imajları görüntüle

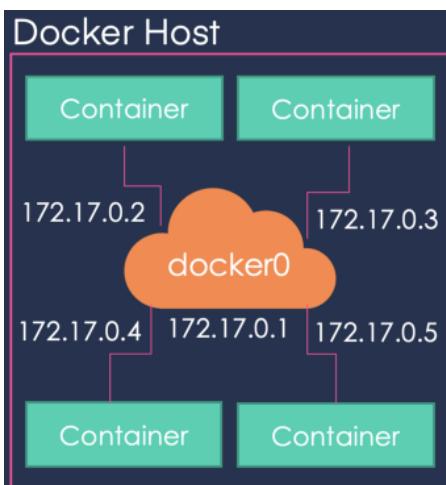
```

root@node1225-ubuntu:/# docker container run --name jen -d -p 3333:8080 -v jdata:/var/log jenkins/jenkins
2062e8538bd29dfcd4cec1a18777331a889bc908f9777f46f8f0693e2341fa4
root@node1225-ubuntu:/# docker commit jen halilgoksell/myjen
sha256:084583c3391489f14bd071ada56151a98500db8542158e8d9af16c28cea31f6a
root@node1225-ubuntu:/# docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@node1225-ubuntu:/# docker image push halilgoksell/myjen
Using default tag: latest
The push refers to repository [docker.io/halilgoksell/myjen]
b3b483849e5b: Pushed
3dfec50bffa8: Mounted from jenkins/jenkins
0b346799714e: Mounted from jenkins/jenkins
42eaf17a1ef0: Mounted from jenkins/jenkins
9a3c617bcf67: Mounted from jenkins/jenkins
f37c6713e3e0: Mounted from jenkins/jenkins
2b0b5e9af282: Mounted from jenkins/jenkins
0b0dbd4f7f9e: Mounted from jenkins/jenkins
da02099aada1: Mounted from jenkins/jenkins
6e3d6cd2abf3: Mounted from jenkins/jenkins
3d2cb0fdc3f2: Mounted from jenkins/jenkins
be035b6e530e: Mounted from jenkins/jenkins
36baf16b8069: Mounted from jenkins/jenkins
461689f80e7e: Mounted from jenkins/jenkins
3e079fee2186: Mounted from library/python
latest: digest: sha256:f9f8eac2192dd498030fa59c6b12024d6e392a635cd77a00487fc9c87e4bf196 size: 3458
root@node1225-ubuntu:/# docker search halilgoksell
NAME          DESCRIPTION      STARS      OFFICIAL      AUTOMATED
halilgoksell/myweb        0
halilgoksell/myjen        0
root@node1225-ubuntu:/# 

```

## Docker Network



Docker hostumuzda default olarak docker0 adında var sayılan bridge network tanımı gelmektedir. Oluşturulan her container default olarak docker0 networkünü kullanır.

Tüm container birbirileriyle ve internet ile erişimi bulmaktadır.

Default olarak docker ile gelen 3 network driver modeli bulunmaktadır.

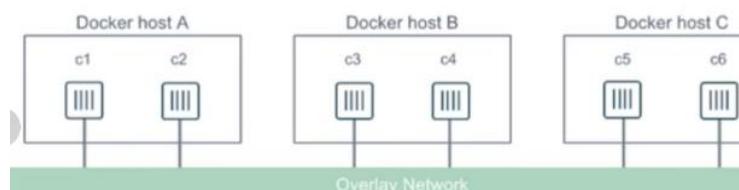
- Host:** Docker hostunuzun NIC'in kulanır, container kendisi üzerinde ağ sanallaştırması yapmaz. IP'ini yapılandırmaz. Örnekteki gibi bir centos ayağa kaldırıldım ve docker hostumun networkünü kullandığını görüyorum

```
root@node1225-ubuntu:~# docker container run --rm -it --network host centos
[root@node1225-ubuntu /]# cat etc/centos-release
CentOS Linux release 8.4.2105
[root@node1225-ubuntu /]# ip a
: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
: venet0: <NOARP,BROADCAST,POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
```

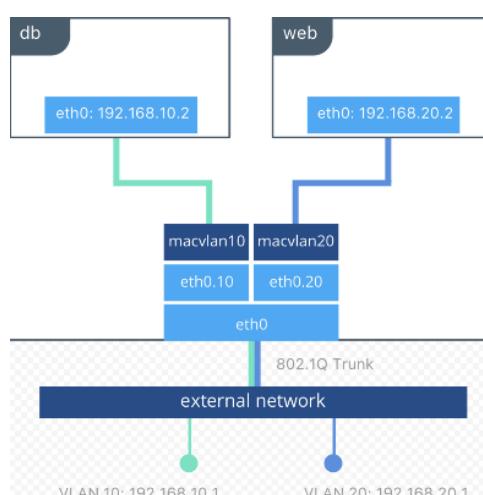
- Null:** Container üzerinden network tamamen izole edilmiş kapalı şekilde, IP adresi almadan çalışır

```
root@node1225-ubuntu:~# docker container run --rm -it --network none centos
[root@b8addf91c7c6 /]# ping 8.8.8.8
connect: Network is unreachable
[root@b8addf91c7c6 /]#
```

- Bridge:** Bir çok sanallaştırma üzerinde gördüğümüz gibi bridge modu, Container üzerindeki sanal network ile default(docker0:172.17.0.0/24) ile docker hostunuz arasında köprü oluşturarak iletişimini kullanır.



**Overlay:** Docker network overlay modeli ise farklı docker hostlar arasında container iletişimini sağlayan modeldir. Docker swarm üzerinde kullanılmaktadır.



**5.MACVlan:** Bu modelde container üzerinde mac adresi ataması yapılır, Bu cihaz network üzerinde cihaz olarak görüntülenmesi sağlanmaktadır. Doğrudan fiziksel ağa bağlı donanım olarak görünür, doğrudan kaynaklarla iletişime geçerler.

Containerların birbirleri ve internet ile haberleşmesi için 2 model bulunmaktadır.

CNM (container network modem) ve CNI (Container network interface)'dır

Docker container sistemi CNM desteklemektedir. Bu modellerinden temel işlevleri

- Containerlar ve desteklenen driverler arasında trafiğinin bölümlenmesi
- Ağ bağlantısı için sandbox içinde endpoint kullanmasını

3. Bir container'da birden fazla network dahil edilmesini
4. Aynı ağdaki containerların haberleşmesi

CNM Modeli 3 ana bileşen üzerine kurulmuştur.

**1.Sandbox** (Containerin arabirimlerinin yönetimi, yönlendirme tablosu ve DNS ayarları dahil olmak üzere bir kapsayıcının ağ yiğininin yapılandırmasını içerir, Dış dünya ve containerlar arasında izolasyon sağlar.)

**2.Endpoint** (Sandbox bileşenini dış dünyaya bağlamakla yükümlüdür. Bir endpoint sadece bir network'e bağlanabilir)

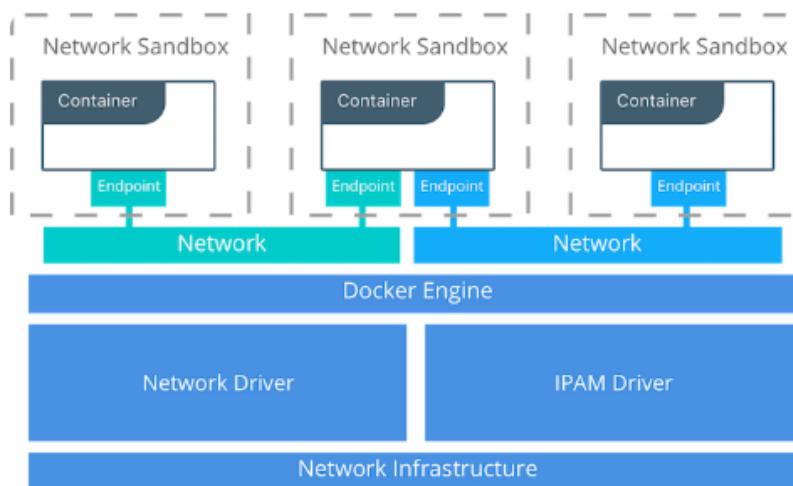
**3.Network** (Birbirleri ile iletişim kurabilen endpoint bileşenini yönetir, Farklı container arasında iletişimini belirler.)

CNM modelinde 2 farklı network driver desteklemektedir.

Network Driver; farklı kullanım durumlarını desteklemek için farklı sürücülerin kolayca kullanılabilmesi ve değiştirilebilmesi için takılabilir. Belirli bir Docker Engine veya Cluster'da aynı anda birden çok ağ sürücüsü kullanılabilir, ancak her Docker ağının örneği yalnızca tek bir ağ sürücüsü aracılığıyla oluşturulur. İki geniş CNM ağ sürücüsü türü vardır:

- **Yerel Ağ Sürücülerı;** Docker Engine'in bir parçasıdır ve Docker tarafından sağlanır
- **Uzak Ağ Sürücülerı;** topluluk ve diğer satıcılar tarafından oluşturulur. Bu sürücüler, yerleşik yazılım ve donanımla tümleştirme sağlamak için kullanılabilir.

IPAM Driver:Docker, ağlar ve uç noktalar için varsayılan alt ağlar veya IP adresleri sağlayan hem yerel hem de uzak IP Adresi Yönetimi Sürücüsü destegine sahiptir.



## Docker Network Komutları

**Docker network ls** docker networkleri listelemek için kullanız

**Docker network create:** Yeni bir network oluşturmak için kullanız subnet, driver modeli gibi conf eklemek için –help komutundan yararlanabilirsiniz. Ör (>`docker network create --driver bridge "network adı"`)

**Docker network inspect:** inspect ile image container gibi detayları aldığımız gibi network modellerinde detay bilgi almamızı sağlar inspect “network adı” ile oluşturulmuş network modelini inceleyebilir, subnet, gateway, driver model, bağlı olduğu container bilgilerini alabiliriz.

**Docker network connect :** Connect komutu ile var olan containerlarımızı var olan networklere dahil etmek için kullanırz. Kullanıcıımı (>`docker network connect "network adı" "container adı"`)

**Docker network disconnect:** Komutu ile container üzerine connect edilmiş network driver çıkarabiliriz. Ör (>`docker network disconnect "network adı" "container adı"`)

Ör:

Myswitch adında bridge driver ile bir network oluşturduğum, ls komutu ile listeleyip inspect ile detaylarına baktığımda 172.18.0.0/16 blogunda auto verdiği gördüm.

```
root@node1225-ubuntu: ~
b04c9b7d06b4    none      null      local
root@node1225-ubuntu:~# docker network create --driver bridge myswitch
3be2e2382f639f7b2ee26808dec2ca0c04ed9e51e8b9c07736d3aaeac3cad042
root@node1225-ubuntu:~# docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
552228b01912   bridge    bridge      local
92a706307155   host      host       local
3be2e2382f63   myswitch  bridge      local
b04c9b7d06b4   none      null      local
root@node1225-ubuntu:~# docker network inspect myswitch
[
  {
    "Name": "myswitch",
    "Id": "3be2e2382f639f7b2ee26808dec2ca0c04ed9e51e8b9c07736d3aaeac3cad042",
    "Created": "2022-10-29T11:24:22.124037709Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    }
  }
]
```

Daha sonra –network parametresini kullanarak oluşturduğum myswitch network ekledim sonrasında bir centos container ayağa kaldırıldım. Sonrasında bash ekranına bağlantıkten sonra ip a komutu ile containerimin ip'sini kontrol ettiğimde oluşturduğum network blogundan aldığıni farkettim.

```
root@node1225-ubuntu:~# docker container run -it --network myswitch centos
[root@6d856403f98e /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
129: eth0@if130: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
```

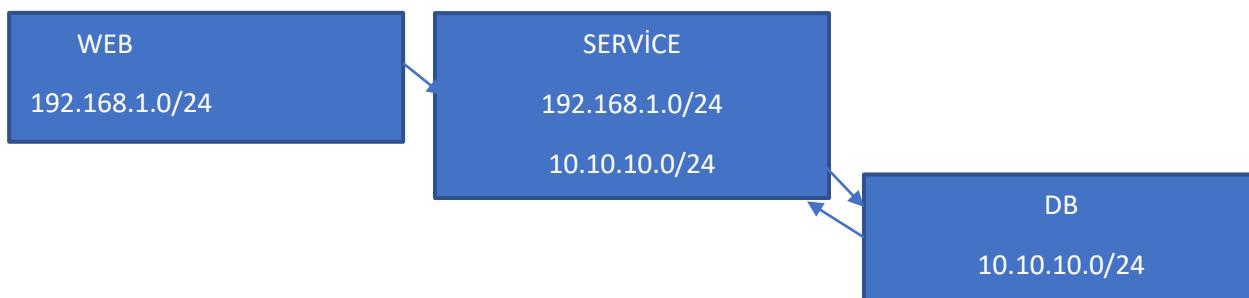
Son olarak'da docker hostum üzerinde oluşturduğum network'ü, ayağa kaldırıldığım centos'dan ayırmak için disconnect komutunu kullandım. Arından oluşturduğum network sildim. Bir network container üzerinde kullanımda ise silme işlemi yapamazsınız

```
docker network disconnect myswitch agitated_chatelet
```

```
root@node1225-ubuntu:~# docker network rm myswitch
```

Aşağıdaki gibi 3 container 2 network örneği üzerinde basit bir mimari uygulacak olursak, service containerına 2 network atamalı, web ve DB network kendi subnetlerine ait network atamalıyız.

Bu örneği gerçekleştiricek olursak



```
>docker network create --driver bridge --subnet "192.168.1.0/24" --gateway "192.168.1.1"
webetc
```

```
>docker network create --driver bridge --subnet "10.10.10.0/24" --gateway "10.10.10.1" dbetc
```

```
root@node1225-ubuntu:~# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
552228b01912    bridge    bridge      local
9279467235b7    dbetc    bridge      local
92a706307155    host      host      local
b04c9b7d06b4    none      null      local
5112a2505856    webetc    bridge      local
```

```
>docker container run --name myweb01 -d --network webetc centos sleep 60m
```

```
>docker container run --name DB -d -e MYSQL_ROOT_PASSWORD=test1231 --network dbetc
mysql
```

```
>docker container run --name service -d --network dbetc centos sleep 60m
```

```
>docker network connect webetc service
```

```
"ConfigOnly": false,  
"Containers": {  
    "5b8ad9e5cde6f52c3ed8f39c36634985300876991326f92043b28b81dc832a8": {  
        "Name": "service",  
        "EndpointID": "5aa7ba1f271b394f7b0619e65861dcc925d5cc7e2d21c2bc9a2a6956d1cb6e7a",  
        "MacAddress": "02:42:c0:a8:01:03",  
        "IPv4Address": "192.168.1.3/24",  
        "IPv6Address": ""  
    },  
    "699525400f53e8b0e7709780d156714a153788e8222a24c21088e07069c27fe5": {  
        "Name": "myweb01",  
        "EndpointID": "af0ff1294976eb748118cdb390e9950ce4ea5d3710292655bd42037bba6c7302a",  
        "MacAddress": "02:42:c0:a8:01:02",  
        "IPv4Address": "192.168.1.2/24",  
        "IPv6Address": ""  
    }  
},  
"NetworkSettings": {  
    "Bridge": "dbetc",  
    "ContainerID": "699525400f53e8b0e7709780d156714a153788e8222a24c21088e07069c27fe5",  
    "GlobalIPv4": "192.168.1.1",  
    "GlobalIPv6": "",  
    "IPAM": {  
        "Driver": "bridge",  
        "Options": {}  
    },  
    "IPV4CIDR": "192.168.1.0/24",  
    "IPV6CIDR": "",  
    "LinkLocalIPv4": "192.168.1.1",  
    "LinkLocalIPv6": "",  
    "MacAddress": "02:42:c0:a8:01:02",  
    "NetworkMode": "bridge",  
    "Ports": {},  
    "SecondaryIPv4": null,  
    "SecondaryIPv6": null  
}
```

Db containerima baktığında 10.10.10.2 IP'sini aldığıni görüyorum.

Exec komutu yardımı ile myweb01 'den DB containerimin IP'sine ping atmaya çalıştığmda erişemediğini görüyorum.

```
root@node1225-ubuntu:~# docker container exec myweb01 ping 8.8.8.8  
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=59 time=15.8 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=59 time=15.6 ms  
^C  
root@node1225-ubuntu:~# docker container exec myweb01 ping 10.10.10.2
```

## Docker Volume

Docker bize 2 farklı depolama imkanı sunmaktadır. Bunlar kalıcı olan ve kalıcı olmayan depolama birimleridir.

Bir container oluşturduğunuzda volume ile ilgili bir tool kullanmadığınızda bu container kalıcı olmayan depolama alanında tutulacaktır, container üzerine yazılan tüm veriler container silinmesiyle beraber geri dönüşü olmayacak şekilde silinecektir. Geçici saklanılan path yolu docker host üzerinde linux “var/lib/Docker/..” Windows “C:\Programdata\ Docker\..” yolundadır.

Kalıcı datalar üzerinde işlem yapmak için volume tanımı yapılmalıdır

Bunun için volumeler oluşturmalı, containerlara bağlamalız, Oluşturulan volume'ler birden çok containera baglama imkanı sunar. Data volume içerisindekidatalara doğrudan erişebiliriz, Taşınması ve yedekleme imkanı sunmaktadır.

Docker üzerine 2 farklı volume türü bulunmakta

**Data Volume:** Docker host üzerinde volume oluşturuluktan sonra containera baglama işlemidir, Docker hostlar üzerinde oluşturulmaktadır. Oluşturulan data volumeler /var/lib/docker/volumes üzerinde saklanmaktadır

Ör.

Docker volume create data (create parametresi ile volume oluşturuyoruz, Volume oluşturmadan container run ederken -v parametresi ile volume belirtirsek, kendisi arka planda volume oluşturacaktır.)

Docker volume inspect data (komutu ile oluşturduğumuz volume'nin lokasyonunu doğruluyoruz.)

```
root@node1225-ubuntu:~# docker volume inspect data
[{"Created": "2022-10-31T07:24:28Z", "Driver": "local", "Labels": {}, "Mountpoint": "/var/lib/docker/volumes/data/_data", "Name": "data", "Options": {}, "Scope": "local"}]
```

cd /var/lib/docker/volumes/data/\_data (ilgili lokasyona gidip bir test klasörü oluşturduk)

```
root@node1225-ubuntu:~# cd /var/lib/docker/volumes/data/_data
root@node1225-ubuntu:/var/lib/docker/volumes/data/_data# mkdir test datax
root@node1225-ubuntu:/var/lib/docker/volumes/data/_data# ls
datax  test
root@node1225-ubuntu:/var/lib/docker/volumes/data/_data#
```

docker container run -it -v data:/data ubuntu (ardından oluşturduğumuz volume'ı bir container bağlayıp container başlatıyoruz. Container içerisine girdiğimde oluşturduğum datanın geldiği görmekteyim.)

```
root@node1225-ubuntu:/var/lib/docker/volumes/data/_data# docker container run -it -v data:/data ubuntu
root@6ada80fb7682:/# ls
bin  boot  data  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@6ada80fb7682:/# cd data
root@6ada80fb7682:/data# ls
datax  test
root@6ada80fb7682:/data#
```

★ -v "volume":":container path":ro (ro "readonly parametresi eklersem sadece okunabilir olur")

**Bind Mounting:** Host üzerinde oluşturulan dosya yada stora üzerinde path bilgisi ile container'a baglama imkanı sunar bunu windows işletim sistemlerinde Dosya sunucusu üzerinden maplenmiş driver gibi düşünebiliriz.

★ Bind mounting yöntemi dockerfile üzerinde kullanılmamakta

```
>mkdir ortakpaylasim
```

```
>mkdir ortakpaylasim/ortakdosyalar
```

```
root@node1225-ubuntu:/ortakpaylasim# ls
ortakdosyalar
root@node1225-ubuntu:/ortakpaylasim#
```

```
>docker container run -d --name myweb1 --volume /ortakpaylasim:/paylasim nginx
```

```
docker container run -d --name myweb1 --volume /ortakpaylasim:/paylasim nginx -
```

```
>docker container exec -it myweb1 bash
```

```
exit  
root@node1225-ubuntu:/ortakpaylasim# docker container exec -it myweb1 bash
```

```
root@d331c35327d6:/paylasim# root@d331c35327d6:/paylasim# ls  
ortakdosyalar  
root@d331c35327d6:/paylasim#
```

Gördüğümüz gibi oluşturduğumuz dosyayı container içerisine aynı volume gibi göstererek map işlemi yapabiliyoruz.

Aynı şekilde readonly olarak bağlamak istersen /paslasim:ro “:ro” parametresini eklemeliyiz.

--volume komutu yerine –monut komutunuda kullanarak aynı işlemi yapabiliriz.

### Docker Temel Komutları

Docker üzerindeki kullanılan bir çok komuturu terminal ekranında >docker komutu ile aşağıda yer verildiği üzere listeleye biliriz.

>docker Komut “ör:exec” –help “ile parametre detaylarına ulaşabiliriz.

```
attach      Attach local standard input, output, and error streams to a running container
build       Build an image from a Dockerfile
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local filesystem
create      Create a new container
diff        Inspect changes to files or directories on a container's filesystem
events      Get real time events from the server
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
history    Show the history of an image
images     List images
import     Import the contents from a tarball to create a filesystem image
info       Display system-wide information
inspect    Return low-level information on Docker objects
kill        Kill one or more running containers
load       Load an image from a tar archive or STDIN
login      Log in to a Docker registry
logout     Log out from a Docker registry
```

logs Fetch the logs of a container  
pause Pause all processes within one or more containers  
port List port mappings or a specific mapping for the container  
ps List containers  
pull Pull an image or a repository from a registry  
push Push an image or a repository to a registry  
rename Rename a container  
restart Restart one or more containers  
rm Remove one or more containers  
rmi Remove one or more images  
run Run a command in a new container  
save Save one or more images to a tar archive (streamed to STDOUT by default)  
search Search the Docker Hub for images  
start Start one or more stopped containers  
stats Display a live stream of container(s) resource usage statistics  
stop Stop one or more running containers  
tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE  
top Display the running processes of a container  
unpause Unpause all processes within one or more containers  
update Update configuration of one or more containers  
version Show the Docker version information

Ek olarak management komutları eski ve yeni versiyon'da kullanılmaktadır. Eski ve yeni arasında fark yoktur kullanıcı deneyimi için 1.13 sürümünde getirilmiştir.

Management Commands:	
builder	Manage builds
config	Manage Docker configs
container	Manage containers
context	Manage contexts
engine	Manage the docker engine
image	Manage images
network	Manage networks
node	Manage Swarm nodes
plugin	Manage plugins
secret	Manage Docker secrets
service	Manage services
stack	Manage Docker stacks
swarm	Manage Swarm
system	Manage Docker
trust	Manage trust on Docker images
volume	Manage volumes

### ESKİ KULLANIM

#docker (komut)  
  
#docker run  
  
#docker (yönetim komutu) komut  
  
#docker container run

### YENİ KULLANIM

## Docker run

Docker run komutu, docker üzerinde kontainerlerimizi çalıştırırmak için kullandığımız komuttur. Eğer docker sunucumuzda run dediğimiz container bulunmuyorsa docker repository (dockerhub) üzerinden imaj çekip, daha sonra çalıştıracaktır.

Oluşturduğumuz container'lar üzerinde aktif bir servis çalışmaz ise pasif durumda olacaktır, Yani containerı aktif tutmak için üzerinde birşeyler çalışıyor durumda olması gerekmektedir.

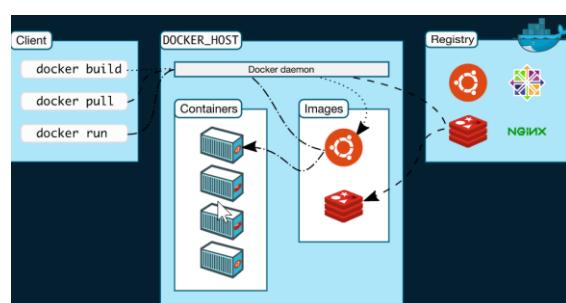
Docker run komutunu kullandığında docker arka planda yapılan adımlar kısa şöyledir.

1. Docker deamon, run ettiğimiz container docker sunucumuzda var mı yok mu diye control eder, Sistem imajın üzerinde olmadığıını doğrular
2. Docker deamon, Docker.hub'a giderek repository üzerinde containerı arar bulduktan sonra imajı indirmeyi başlar
3. Docker deamon indirilen imajdan container oluşturur
4. Son aşamada docker deamon container çıktısını ekranan verir.

```
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(amd64)  
3. The Docker daemon created a new container from that image which runs the  
executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
to your terminal.
```

## Docker Deamon Nedir

Docker API isteklerini dinler ve görüntüler, kapsayıcılar, ağlar ve birimler gibi Docker nesnelerini yönetir



Ör;

```
>docker run nginx
```

```
>docker run mysql
```

```
>docker run -it ubuntu
```

“-it” intreractive parametresi

```
root@node1225-ubuntu:~# docker run -it ubuntu /bin/bash  
root@2641489eed0b:/#
```

>docker run -it (interactive) ubuntu /bin/bash (ubuntu containerimize çalıştırıp bash üzerinde çalışmak için “-it” parametresini kullandık )

Ek olarak image parametresinden sonra kullandığımız komut, container üzerinde çalışacaktır.

Ör:

```
>Docker run -it ubuntu cat /etc/*release*
```

```
root@node1225-ubuntu:~# docker run -it ubuntu cat /etc/*release*  
DISTRIB_ID=Ubuntu  
DISTRIB_RELEASE=22.04  
DISTRIB_CODENAME=jammy  
DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"  
PRETTY_NAME="Ubuntu 22.04.1 LTS"  
NAME="Ubuntu"  
VERSION_ID="22.04"  
VERSION="22.04.1 LTS (Jammy Jellyfish)"  
VERSION_CODENAME=jammy  
ID=ubuntu  
ID_LIKE=debian  
HOME_URL="https://www.ubuntu.com/"  
SUPPORT_URL="https://help.ubuntu.com/".
```

```
>Docker run -it centos yum update -y && yum install nginx -y
```

Run-tag

“-d”detach

-d parametresi ile bir container başlattığımızda, container ekranında sıkışmamak için -d parametresini kullanırız. Böylece containiri başlatıldığından arka planda ayırma (detach mode)modunda çalışacaktır.

Docker -d “container id”

Containera geri bağlanmak istersek bu seferde “attach” komutu kullanmalıyız (örneklerde yer verildiği gibi exec’de kullanabiliriz.)

```
>Docker attach “container ID”
```

ör

```
>docker run -d ubuntu sleep 100
```

“:(version)

Docker sunucusuna indirdiğimiz paketler son version docker hub repository’den dowland edilmektedir.

Peki ihtiyacımız olan bir containirin eski sürümünü indirmek istersek ?, Buna run-tag eklemeyiz. Destelenen tüm sürümler dockerhub'da ilgili containerin description bilgilerinde yer olacaktır.

örnek ile

Docker run redis:**4.0**

Docker run wordpress:**4.8**

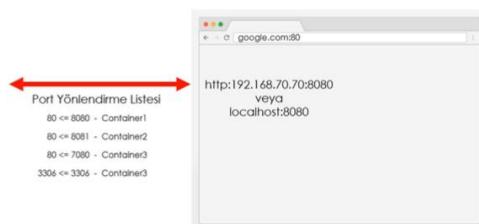
```
[root@node1225-ubuntu: ~] - □ ×
root@node1225-ubuntu:~# docker run wordpress:4.8
Unable to find image 'wordpress:4.8' locally
4.8: Pulling from library/wordpress
25b1f47fb949: Downloading 31.11MB/52.6MB
```

### **"-p" (publish)**

Çalışan containerlarımıza 2 tür erişim imkanı bulunmaktadır

1. Docker sunucumuzun networkünden, veya iç network'den “container ip:port” olarak erişebiliriz.
2. Erişim de dışarıdan erişmek istersek docker sunucumuza port maplemek için -p parametresini kullanmalıyız. Bu şekilde docker sunucumuza eriştiğimizde port mapping sayesinde ilgili containerimize erişeceğiz.

**8080 : 80**  
**(host) (container)**



>**Docker container run -p 8080:80 nginx /** komutu ile bir nginx containerı başlattım ve 8080 için mapledim. 2. Bir nginx container başlattığımızda 8080 tekrar kullanamam bir diğer farklı bir port bilgisi girebilirim.

Anlaşılması için 8080(buradaki docker sunucumun hangi port ile servisi dinleyeceği) (80 ise default olarak containerların kullandığı dinlenen servis portları olarak özetleyebiliriz.)

Network tabirinde binevi PAT (port address translation) işlemi yapmakta.

Diğer parametreler açıklamalarıyla aşağıdaki gibidir.

Flag value	Description
-p 8080:80	Map TCP port 80 in the container to port 8080 on the Docker host.
-p 192.168.1.100:8080:80	Map TCP port 80 in the container to port 8080 on the Docker host for connections to host IP 192.168.1.100.
-p 8080:80/udp	Map UDP port 80 in the container to port 8080 on the Docker host.
-p 8080:80/tcp -p 8080:80/udp	Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host.

★ -P (büyük P parametresi random olarak host port tanımı yapacaktır.)

Ör:

```
>Docker run -P -d --name halilgoksel.com nginx
```

```
root@node1225-ubuntu:~# docker run -P -d --name halilgoksel.com nginx
c8ace3fcff5fb370e780f3d1b92d3bb12c50e40a256af0dae958d646486d6851
root@node1225-ubuntu:~# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c8ace3fcff5f nginx "/docker-entrypoint...." 3 seconds ago Up 2 seconds 0.0.0.0:49154->80/tcp, :::49154->80/tcp halilgoksel.com
560da9de0d2 nginx "/docker-entrypoint...." 38 seconds ago Up 37 seconds 0.0.0.0:49153->80/tcp, :::49153->80/tcp angry_papier
```

### **"-v" (Volume mapping)**

Biz container oluşturduğunuzda içerisindeki veriler /var/lib/"mysql" dosyasının içeriğine yazmaktadır. Sil containiri sildiğinizde bu datalar'da uçacaktır. Bu dataları kalıcı olarak bir birime yasmak istiyorsak -v parametresini kullanmalıyız.

-v (Yazılacakhostpath:Containerpath) sırasıyla

```
>Docker run -v /opt/datadir:/var/lib/mysql mysql
```

```
ubuntu@docker:/opt$ docker run -v /opt/datadir:/var/lib/mysql mysql
[...]
```

>Docker rm "container" (mysql sildiğimizde datalarımızın opt/datadir dizininde var olduğunu göreceğiz.)

### **"-i"-Interactive terminal**

Interactive ile container içerisinde bağlanmamızı izin veren parametredir kısaltması "-i" geçer.

-tty container terminaline erişim sağlar kısaltması "-t" dir.

İkisinin kullanım şekli -it şeklindedir. Container'den sonra işletim sistemi kabuğunu belirtmeliyiz ör: "bash"

Ör;

```
Docker container run -it centos bash
```

```
@5277a173eed1:/
root@node1225-ubuntu:~# docker container run -it centos
root@5277a173eed1:~#
```

Ör

```
Docker container run -it python bash
```

```
Docker container run -it Microsoft/iis powershell
```

### **"-e" –env (environment)**

Bu parametre ile container içerisindeki uygulamaya ortam değişkeni atamak için kullanabiliriz.

Ortam değişkenleri için docker.hub üzerinden yardım alabililiriz, Her uygulama farklılık göstecektir.

Ör: Bir DB containerı oluştururken bu parametre ile database şifresini belirtebiliriz. Sonrasında management toolbar ile (SSMS, dbForge gibi) yada docker host üzerinden intancelarımızın yönetebilriiz.

Kontainer oluştururken DB bağlanmak için -e kullanmamız gereklidir, Örnek kullanım aşağıdaki gibidir.

Maria Container

```
>Docker container run -p 3306:3306 -e MYSQL_ROOT_PASSWORD='Password1' mariadb
```

SQL Server Container

```
>Docker container run -p 1433:1433 -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=Password1'
```

```
mcr.microsoft.com/mssql/server
```

İnstancelara bağlanmak için ise tool'ar ile port, user ,passwd bilgileriyle bağlanır yada, docker host üzerinden yönetebiliriz. Docker host üzerinden bağlanmak için örnek komut aşağıdaki gibidir

Maria DB container

Docker container exec -it "container id" /opt/mssql-tools/bin/sqlcmd -S "-S bağlanacağımız instance" -U "-U user belirtiyoruz" SA -P "-P şifre belirtiyoruz"

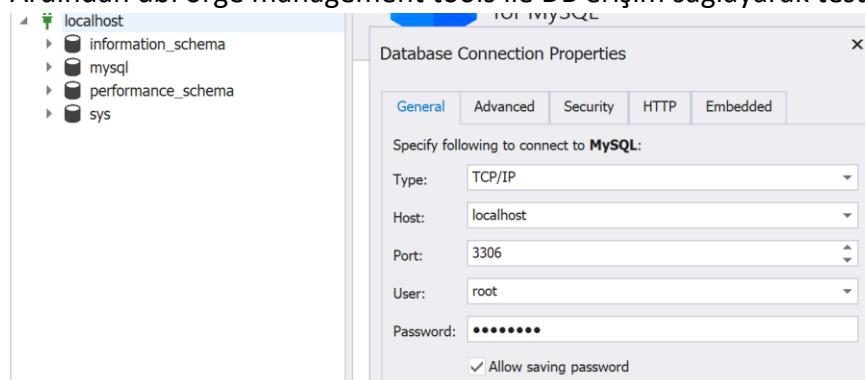
Ör

İlk önce docker hostumuzda bir mariadb containerı oluşturuyoruz.

```
Docker container run --name maria -p 3606:3606 -e MYSQL_ROOT_PASSWORD='test123!'  
maradb
```

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
PS C:\Users\hllgo> docker container run --name maria -p 3606:3606 -e MYSQL_ROOT_PASSWORD='test123!' mariadb
```

Ardından dbForge management tools ile DB erişim sağlayarak test ediyoruz.



## docker ps

Docker ps komutu docker üzerindeki tüm **çalışan** kontainerlerimizi ve bu kontainerler hakkında bize bazı bilgi veren komuttur.

```
>docker ps
```

Yeni version komut: docker container (list, ls, ps)'dir.

Her container rastgele bir ID alır.

(-a) parametresi tüm **çalışan ve çalışmayan** kontainerları listeler

```
>docker ps -a & docker container ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7fac3b4aacf	centos	"/bin/bash"	37 hours ago	Created		xenodochial_varahamihira
bdfc5df5173e	ubuntu	"bash"	42 hours ago	Exited (130) 41 hours ago		hungry_shirley
0bc8ea005df	ubuntu	"bash"	42 hours ago	Exited (129) 42 hours ago		elated_agnesi
b29035edb10a	ubuntu	"/bin/bash"	43 hours ago	Exited (0) 43 hours ago		admiring_borg
fd5c1fff2e39	ubuntu	"bash"	43 hours ago	Created		myubuntu
cde7ec49cb61	nginx	"/docker-entrypoint...."	4 days ago	Exited (0) 4 days ago		keen_franklin
77e224005974	jenkins/jenkins	"/usr/bin/tini -- /u..."	4 days ago	Exited (129) 4 days ago		zen_colden
f26e8c74459c	jenkins/jenkins:lts	"/usr/bin/tini -- /u..."	5 days ago	Exited (129) 5 days ago		goofy_margulis
6827d2f6dd27	ubuntu:17.10	"sleep 400"	5 days ago	Exited (137) 5 days ago		ecstatic_brown
7b4f578291dc	mysql	"docker-entrypoint.s...."	5 days ago	Exited (1) 5 days ago		infallible_yonath
d9871d98303d	wordpress	"docker-entrypoint.s...."	5 days ago	Exited (0) 5 days ago		clever_babbage
ab4287887915	wordpress:4.8	"docker-entrypoint.s...."	5 days ago	Exited (0) 5 days ago		vigilant_feynman
e47e03a5d1da	redis	"docker-entrypoint.s...."	5 days ago	Exited (0) 5 days ago		great_bohr
root@node1225:~#						

docker ps -a çıktısını incelediğimizde sırayla

**CONTAINER ID:** Her oluşturduğumuz container için bir unic ID tanımlanmaktadır.

**IMAGE:** Üzerinde çalıştığımız kontainerimi ve sürümünü listeler

**COMMAND:** Containerimizi hangi komut ile ayağa kaldırdığımızı, hangi komutu kullandığımızın bilgisini verir

**CREATE:** Container oluşturul zamanı hakkın bilgi verir

**STATUS:** Containerınız çalışıp çalışmadığı, status bilginini yansıtır.

**PORTS:** Containerlarımızın için port mapping bilgisini verir.

**NAMES:** Container id gibi benzersiz bir docker adı tanımlar.

## --quiet parametresi

--quiet “-q” parametresi sadece container id listelemek için kullanırız,

```
Docker ps -a -q
```

```
root@node1225-ubuntu:~# docker ps -a --quiet
6d208221e570
7fac3b47aacf
bdfe5df5173e
0bc8ea00e5df
b29035edb10a
fd5c1fff2e39
cde7ec49cb61
77e224005974
f26e8c74459c
6827d2f6dd27
7b4f578291dc
d9871d98303d
ab4287887915
e47e03a5d1da
root@node1225-ubuntu:~#
```

## Docker rename

komutu ile containerlarımız adlandırabiliriz

```
>Docker container rename "elated_agnesi" "ubuuntu"
```

```
CONTAINER ID  IMAGE      COMMAND      CREATED     STATUS      PORTS      NAMES
0bc8ea00e5df  ubuntu      "bash"       42 hours ago  Up 6 minutes          elated_agnesi
cde7ec49cb61  nginx      "/docker-entrypoint...."  4 days ago   Up 4 minutes  80/tcp     keen_franklin
root@node1225-ubuntu:~# docker container rename elated_agnesi ubuuntu
root@node1225-ubuntu:~# docker ps
CONTAINER ID  IMAGE      COMMAND      CREATED     STATUS      PORTS      NAMES
0bc8ea00e5df  ubuntu      "bash"       43 hours ago  Up 6 minutes          ubuuntu
cde7ec49cb61  nginx      "/docker-entrypoint...."  4 days ago   Up 5 minutes  80/tcp     keen_franklin
root@node1225-ubuntu:~#
```

## docker container silme

### Docker rm

Docker üzerindeki kontainerlerimizi kaldırmak ve tamamen silmek için docker rm komutunu kullanılır

```
>docker rm "container ID"
```

Çalışan bir containeri silmek için ise -f (force) parametresini eklemeliyiz.

Birden çok container tek satır'da kullanabiliriz, her bir ID'nin ilk iki satırını (unic olmalı) yazarak tek satırda silmek mümkün

```
>docker rm b7 36
```

```
root@node1225-ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND     CREATED      STATUS
b7cf5a420356   ubuntu   "bash"      33 seconds ago   Exited (0) 33 seconds ago
363906e71fcc   ubuntu   "/bin/bash"  22 hours ago   Exited (127) 21 hours ago
root@node1225-ubuntu:~# docker rm b7 36
b7
36
root@node1225-ubuntu:~#
```

Başka bir örnekte ise container oluştururken -rm parametresini kullanırsak container'dan çıkış yaptığımızda ilgili containeri silecektir.

Ör: >docker run -it -rm -name myubuntu ubuntu

Kontainer'dan çıkış yaptığımızda stop durumda olduğunu görmeliydi fakat silinmiş durumda

```
root@node1225-ubuntu:~# docker run -it --rm --name myubuntu ubuntu
root@5a9130ed6f2a:/# exit
exit
root@node1225-ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND     CREATED      STATUS          PORTS      NAMES
6c8a9c96cd58   ubuntu   "bash"      42 seconds ago   Exited (130) 36 seconds ago
0a821be28157   ubuntu   "bash"      56 seconds ago   Exited (0) 54 seconds ago
```

★ Docker üzerinde komut içinde komut çalıştırmak istersek \$() kullanırız.

Yani docker komut \$(docker komut 2)

Bir örnek yaparsak

>Docker rm -f \$(docker ps -a -q)

```
root@node1225-ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND     CREATED      STATUS      PORTS      NAMES
0bc8ea00e5df   ubuntu   "bash"      43 hours ago  Up 35 minutes
cde7ec49cb61   nginx   "/docker-entrypoint...."  4 days ago   Up 34 minutes  80/tcp    keen_franklin
root@node1225-ubuntu:~# docker rm -f $(docker ps -a -q)
0bc8ea00e5df
cde7ec49cb61
root@node1225-ubuntu:~# docker ps -a
CONTAINER ID   IMAGE    COMMAND     CREATED      STATUS      PORTS      NAMES
root@node1225-ubuntu:~#
```

Docker rm -f ile force sil parametresini kullanıyoruz, 2. Kısmında docker ps -a -q ile tüm container id'leri listeletiyoruz. İşlemin tamamlandığını görüyoruz.

Başka bir diğer örnek container oluştururken -rm parametresini kullanırsak kontainerden çıkış yaptığımızda kendisini silecektir.

Yine tüm çalışan imajları silmek için komut içinde komut kullanabiliriz

>Docker image rm \$(docker image ls -q)

### Docker container prune

Docker prune komutu ile tüm çalışmayan containerlerimizi toplu halde silmek için kullanırız.

>Docker container prune



Writer: @HalilGÖKSEL

```
root@node1225-ubuntu:~# root@node1225-ubuntu:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND          CREATED        STATUS      PORTS     NAMES
38224e73ad4a   ubuntu    "bash"           29 seconds ago  Exited (0) 28 seconds ago  mystifying_pike
3896a7ce5083   ubuntu    "bash"           31 seconds ago  Exited (0) 30 seconds ago  funny_pare
0e702f2864f8   ubuntu    "bash"           34 seconds ago  Exited (0) 33 seconds ago  competent_hamilton
9ce5bb5df2e6   nginx    "/docker-entrypoint..."  59 seconds ago  Exited (0) 48 seconds ago  focused_germain
root@node1225-ubuntu:~# docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
38224e73ad4a2707c5d4f69442f588ea6dff0bb251c74f34cf690102b8fb4a46
3896a7ce508399999b540f26a0760928494750a3b18c9b74a1ed990b0659ab1d
0e702f2864f8dfce1e2f6b723f8030ffda5c0c89f99518c3a6f23a662b19c4d7
9ce5bb5df2e6fcad6fc1fb759b96147a5d20f5bc6f4f8056805763e0fd92e66b
```

## Docker rmi

Docker imajlarını silmek ise, rmi komutunu kullanırız. Docker imajlarını silmeden önce, çalıştırığınız kontainerların bu imaj üzerinden çalışmadığından emin olmanız gerekmektedir. Aksi takdirde silemez hata alırsınız.

```
>docker rmi nginx
```

```
root@node1225-ubuntu:~# docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:2f770d2fe27bc85f68fd7fe6a63900ef7076bc703022fe81b980377fe3d27b70
Deleted: sha256:51086ed63d8cba3a6a3d94ecd103e9638b4cb8533bb896caf2cda04fb79b862f
Deleted: sha256:c22f011a5c63a718e3155ef21b930f5583102384c8e333299913ed660baa230c
Deleted: sha256:1235ee8acd48a34c389280d8192ae79ef241d546eeeaa2f3416b64608d68d8538
Deleted: sha256:80ab7667b1007f2ed4b5387e7585e18d3ca1899c76449240e2890373a8e77285
Deleted: sha256:4833b18722fc3d06fea0f61726b1b11baa1daa0ea455e6e2ab66a7c8db283
Deleted: sha256:98b8d2ed046082a8f6c2fb2f34430f5142fea7a7078326d980b323d71640d8ff
Deleted: sha256:fe7b1e9bf7922fbc22281bcc6b4f5ac8f1a7b4278929880940978c42fc9d0229
root@node1225-ubuntu:~# █
```

## Docker version

Docker version detaylarına bakmak için kullandığımız komuttur ust kısıkda docker client alt kısıkda docker engine ciktisini verir.

```
>docker version
```

```
Client: Docker Engine - Community
Version:           20.10.19
API version:       1.41
Go version:        go1.18.7
Git commit:        d85ef84
Built:             Thu Oct 13 16:46:17 2022
OS/Arch:           linux/amd64
Context:           default
Experimental:      true

Server: Docker Engine - Community
Engine:
  Version:          20.10.19
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.7
  Git commit:       c0964641
  Built:            Thu Oct 13 16:44:09 2022
  OS/Arch:          linux/amd64
  Experimental:    false
  containerd:
    Version:         1.6.8
    GitCommit:       9cd3357b7fd7218e4aec3eae239db1f68a5a6ec6
  runc:
    Version:         1.1.4
    GitCommit:       v1.1.4-0-g5fd4c4d
  docker-init:
    Version:         0.19.0
    GitCommit:       de49bad0

root@arker:/home/ubuntu#
```



## Docker Info

Docker sunucumuz hakkında detaylı bilgiler (imajlar, start-stop durumda olan containerlar, kaynak bilgileri, vs gibi) yer verir

```
>docker info
```

## docker pull

Docker sunucusuna imaj indirmek için kullandığımız komuttur. (Docker run komutundada aynı şekilde var olanmayan imajları kütüphaneden indirir fakat run komutu ile çalıştırır, bu komut sadece imajları depolamak içindir)

```
>docker pull nginx
```

```
>doxker pull mysql
```

```
root@node1225-ubuntu:~# docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
295ca2342728: Extracting [=====]
79af4312a7e0: Download complete
48d3d73d1704: Download complete
521b8724b397: Download complete
b2af260b4a14: Download complete
] 8.946MB/40.59MB
```

## Docker exec

Bu komut çalışan bir kontainerlerimizin üzerinde bir komut çalıştırmak ve işlem yapmak için kullandığımız parametredir.

Ör

```
>Docker exec "container ID" cat /etc/*release*
```

```
>Docker exec "container id " sudo apt update
```

```
root@node1225-ubuntu:~# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
68d60320189f ubuntu "sleep 100" About a minute ago Up About a minute dreamy_lederberg
root@node1225-ubuntu:~#
root@node1225-ubuntu:~# docker exec 68d60320189f cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"
PRETTY_NAME="Ubuntu 22.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.1 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
root@node1225-ubuntu:~#
```

Başka bir örnek yine çalışan centos bir containerimda bash'de bağlanmadan nginx paket kurulumu yapabiliriz.

```
PS C:\Windows\System32> docker container exec ng yum install -y nginx
CentOS Linux 8 - AppStream 84 B/s | 38 B 00:00
```

Writer: @HalilGÖKSEL

Aktif durumda olan container içeresine girmek içinde exec parametresini kullanabiliriz. Aşağıdaki örnekteki gibi centos containerimiz -d (detach) kullanarak bash ekranında ayrılmak için kullandık daha sonra aktif olan containerimize exec ile bağlandık

**>docker exec -it "container name" bash**

```
root@node1225-ubuntu:~# root@node1225-ubuntu:~# docker run -d --name mycentos centos sleep 30m
62a433a414e9a909a8097687cec236d2bdfb59f91af6cd8911aafdf8f6d5481d7
root@node1225-ubuntu:~# docker exec -it mycentos bash
[root@62a433a414e9 /]#
```

### Docker sleep

Kontainerlarımızı belli bir süre zarfında çalıştırılmak için kullandığımız komuttur, Sleep komutu ile saniye cinsinden zaman aralığı verebiliriz.

**>Docker run -d ubuntu sleep 20**

```
root@node1225-ubuntu:~# docker run -d ubuntu sleep 20
ec387814d21a230f59fdde09e97b9fe5e0bd9134d547e0bcd10c1adf903068c
root@node1225-ubuntu:~#
```

**>Docker ps** yaptığımızda çalıştığını görebiliriz

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e6b5e1298c69	ubuntu	"sleep 20"	3 seconds ago	Up 1 second		confident_blackwell

```
root@node1225-ubuntu:~#
```

Eğer -d parametresini koymasaydık terminale sıkışacaktır ctrl+c gibi işlemler ile geri gelemeyeceğiz.

Bundan çıkmak için sleep süresince bekleyebiliriz, yada 2. Bir terminal ile docker bağlantı sağlayıp containirimizi stop etmeliyiz

### Docker stop & start & restart

Çalışan container'ları durdurmak ,başlatmak ve yeniden başlatmak için kullandığımız komut

**>docker container stop "container id"****>docker container start "container id"****>docker container restart "container id"**

### Docker pause & unpause

Çalışan containerleri duraklatmak ve devam ettirmek için kullandığımız komut türüdür.

**>Docker container pause "container id"****>Docker container unpause "container id"**

### Docker inspect

Inspect komutu containerlar, Volume'ler, Network, Swarm servis durum bağları, verileri, ağ ayarları vb. Json biçimde içeren bilgileri ekranımıza verir.

### >Docker container inspect “container ID”

Bir Jenkins container ayağa kaldırındı var sayalım local’den arayüzüne erişmek için container IP ve port bilgilerine ihtiyaç var docker ps komutu ile çalışan containerı görebiliriz yada -p ile port mapleyebiliriz. IP bilgisi için inspect’den yararlanacağız, aşağıdaki gibi bizi detayları yer verecek

```
"Gateway": "172.17.0.1",
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"MacAddress": "02:42:ac:11:00:02",
"DriverOpts": null
```

Ayrıca grep parametresi ile çıktıyı filreleyebiliriz.ör; Sadece IP bilgisi öğrenmek için,

Docker inspect “container id” | grep IPAddress

```
root@node1225-ubuntu:~# docker inspect myweb | grep IPAddress
    "SecondaryIPAddresses": null,
    "IPAddress": "172.17.0.7",
    "IPAddress": "172.17.0.7",
```

### Docker logs

Containerlerimizin arka planda ürettiği logları görüntülemek için logs komutunu kullanmalıyız.

### >docker container logs “container id”

### Docker top

komutu ile container içerisinde çalışan processleri listeleyebiliriz.

### >docker container top “container id”

```
root@node1225-ubuntu:~# docker top d9d0b51c5429
UID          PID  PPID      C      STIME     TTY      TIME     CMD
root        41199  41179      0   14:18      ?  00:00:00  nginx: master pr
ocess nginx -g daemon off;
systemd+    41258  41199      0   14:18      ?  00:00:00  nginx: worker pr
ocess
systemd+    41259  41199      0   14:18      ?  00:00:00  nginx: worker pr
```

### Docker stats

Docker stats komutu ile containerimizin kaynak(cpu, memory) kullanımını görebiliriz

### docker stats “container id”

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
d9d0b51c5429	laughing_gates	0.00%	6.559MiB / 4GiB	0.16%	656B / 0B	791kB / 8.19kB	7

-a parametresini kullanarak tüm çalışan kontainerlerin kaynak kullanımılığını görebiliriz.

### Docker commit

Docker üzerindeki containerimizin imajı oluşturmak için commit kullanırız.

```
>docker container commit "container ID"
```

Docker images ile oluşturduğunu görebiliriz.

## Grafik Arayüz ile Docker Yönetim Toolları

Grafik araçlarını kullanarak arka planda docker engine yönetmi bazı toolar ile mümkün bunları incelersek

### 1.KITEMATIC

Opensource bir ürünüdür, Windows ve linux OSX gibi işletim sistemlerinde kullanım desteklemektedir. Gui olarak container yönetimi (network, port, volume gb) oluşturma silme işlemleri vs yapabiliriz.

### 2.PORTAINER

Aynı şekilde linux, windows ve OSX işletim sistemlerinde Docker GUI yönetim aracıdır. Kurulum şekli kullandığınız docker host üzerinden. Standalone yapıda çalışmaktadır. Ücretlidir free trail-30

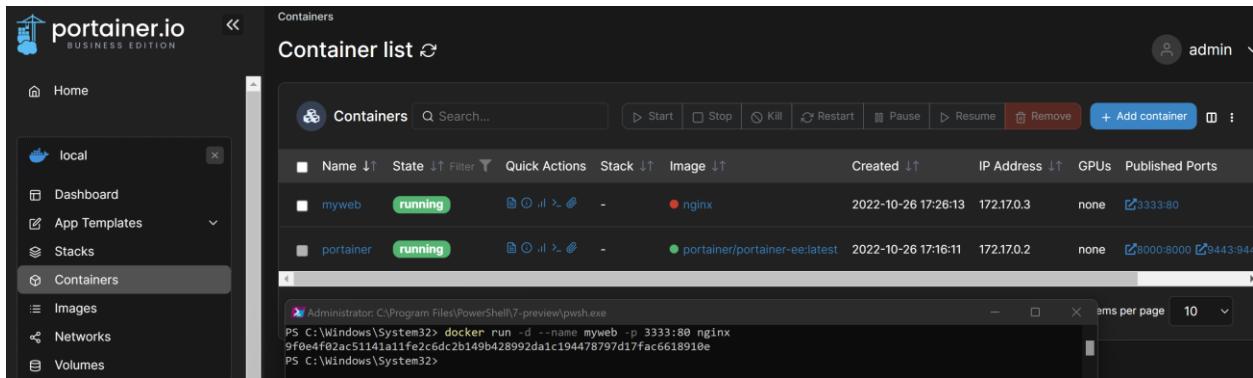
(Kurulum için adresinden bakabilirsiniz[Docker Standalone \(portainer.io\)](https://portainer.io/))

Windows Docker host üzerine kurulum için

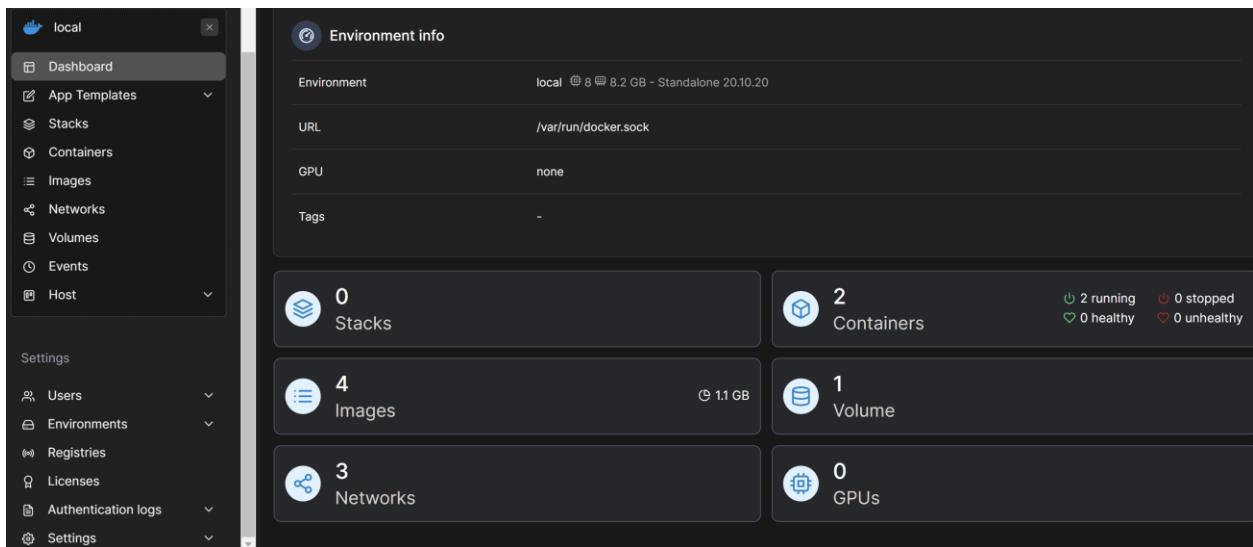
1. docker volume create portainer\_data
2. docker run -d -p 8000:8000 -p 9443:9443 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer\_data:/data portainer/portainer:ee:latest
3. Login : <https://localhost:9443>
4. Giriş yaptıktan sonra admin hesabı belirlememizi isteyecek ardından lisans aktivasyonu girmemizi isteyecektir, kayıt olduktan sonra free treail için e-posta adresinie lisans key iletilecektir
5. Şifre test123123.!

Giriş yaptıktan sonra kurmuş olduğum docker host üzerinde tüm containerları kendisi syn edecektir. İster docker komutları ile isterseniz portal ile yönettiğiniz docker engine kullanabilirsiniz.

Gördüğünüz gibi docker komutları ile oluşturduğum nginx imajını portainer'de görüntüleyebiliyorum.



Aşağıda portainer anasayfa arayüzünde göründüğü gibi, Containerlerimizi , Stacks'larımı (coklu container yapısı) image, volume, network gibi alanlarımida yönetebiliyorum.



## Docker Registry

Docker hub üzerinde imaj indirebildiğimiz gibi docker.hub üzerinde bir kullanıcı hesabı oluşturarak kendi imajlarımızı yükleyebilir daha sonra bu imajlarımızı docker hostlara cekebiliriz.

Docker hostunuza bir imaj cekmek istediğinizde docker deamon ilk önce local imajlarda pull ettiğiniz imajı aramaya başlayacak eğer yoksa register üzerinden çekecektir. Default docker register docke.hub alanıdır. Registerden sonra repo yani namespace alınını ekler yani halilgoksel / mynginx gibi.

En sonra tags olarak aratır, tags'lar version içermektedir :latest olarak kullanırsan son versiyonunu çekmektedir.

**REGISTRY > REPO > TAGS**

Docker.io > halilgoksel > myweb :latest

## DTR – (Docker Trusted Registry)

Docker imajlarını on-prime ortamlarda güvenli bir şekilde saklayabilmemiz için DTR hizmeti sunmuştur. Böylece dağıtım yönetimini ve saklamasını kendimiz yapabiliriz.

## Container üzerine data transferi

Containere'larımıza data transfer etmenin 3 yolu bulunmakta bunlar;

**1-docker container cp (copy)**

**2-Mount Volume**

**3-internet (get, sftp)**

Sırayla incelersek,

CP

“cp” parametresi ile host’dan containere data transfer

1- Cp parametresi ile docker hostumuzdaki dataları container’larımıza aktarabiliriz.

### Kullanım şekli (Host’dan Container’e)

Docker container cp “hostpath” “container:containerpath”

```
PS C:\> dir
Directory: C:\

Mode                LastWriteTime       Length Name
----                -----           -----    Name
d----            3/6/2022 11:15 PM          desktop
d----            11/23/2021 11:10 PM        DRIVERS
d----            10/20/2022 9:30 PM        Intel
d----            6/5/2021 3:10 PM        PerfLogs
d-r--            10/24/2022 1:38 PM        Program Files
d-r--            10/18/2022 9:07 AM        Program Files (x86)
d-r--            9/24/2022 11:35 AM        Users
d----            10/13/2022 6:40 PM        Windows
-a--            3/20/2022 7:08 PM        12288 DumpStack.log
-a--            10/24/2022 4:07 PM        0 index.html
-a--            12/1/2006 10:37 PM        904704 msdia80.dll
```

İlk olarak docker hostumuzun C: dizininde index.html dosyasını oluşturdum.

```
PS C:\> docker container cp C:\index.html myweb:/home
PS C:\> docker exec -it myweb bash
root@8fe944e37458:/# cd /home
root@8fe944e37458:/home# ls
index.html
root@8fe944e37458:/home#
```

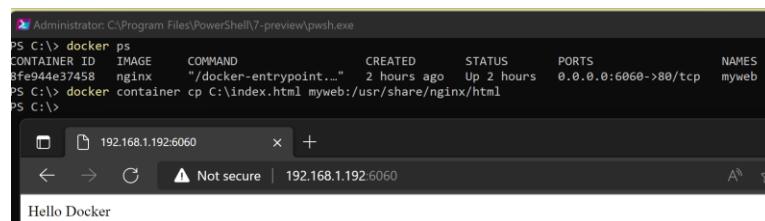
Daha sonra cp parametresini kullanarak ilk önce kaynak sonrada hedef path’imi belirttim.

Kopyalama işleminden sonra control ettiğimde index.html dosyamın container içerisinde aktarıldığını gördüm.

Farklı bir örnek olarak index.html çalışmasını bir nginx container upload edip çalıştmak istersek aynı şekilde

```
>docker container run -d -p 6060:80 --name myweb nginx (6060 port maplediğim myweb adında bir nginx containeri çalıştırıyorum)
```

```
>docker container cp C:\index.html myweb:/usr/share/nginx/html (ardından index html dosyasını, nginx'in yayın için kullandığı html path'ine transfer ettim, Kontrol ettiğimde html dosyamı yayınladığını görüyorum.)
```



A screenshot of a Windows PowerShell window titled "Administrator: C:\Program Files\PowerShell\7-preview\pwsh.exe". The window shows the command "PS C:\> docker ps" which lists a single container named "myweb" with ID "8fe94e37458", running the "nginx" image and mapping port 6060 to 80. Below it, the command "PS C:\> docker container cp C:\index.html myweb:/usr/share/nginx/html" is shown. At the bottom, a browser window is open at "192.168.1.192:6060", displaying the text "Hello Docker".

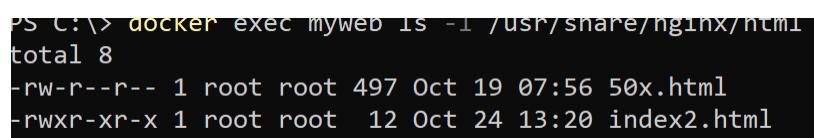
★ Windows kernele sahip container'da (windows server gibi) kopyalama işlemi container stop durumda iken yapılmamaktır.

### "cp" parametresi ile container'den host'a data transfer

Kullanım şekli (Container'den Host'a)

Docker container cp "container:path" "hostpath"

İlk önce containerımız verileri almak istediğim dizinde exec ile içerisine girmeden komut çalıştırarak listeliyorum. Index2.html dosyamı host'uma alacağım.



A screenshot of a Windows PowerShell window titled "Administrator: C:\Program Files\PowerShell\7-preview\pwsh.exe". The command "PS C:\> docker exec myweb ls -l /usr/share/nginx/html" is run, showing the contents of the directory: "total 8", "-rw-r--r-- 1 root root 497 Oct 19 07:56 50x.html", and "-rwxr-xr-x 1 root root 12 Oct 24 13:20 index2.html".

Ardından cp komutunu kullanarak tam tersi şekilde

docker container cp myweb:/usr/share/nginx/html/index2.html C:\ ("container id:container path" boşluk host path yazıyorum ve çalışıyorum, göründüğü gibi C: altında index2.html dosyası gelmiş görünüyor.)



Writer: @HalilGÖKSEL

Mode	LastWriteTime	Length	Name
d---	3/6/2022 11:15 PM		desktop
d---	11/23/2021 11:10 PM		DRIVERS
d---	10/24/2022 4:31 PM		html
d---	10/20/2022 9:30 PM		Intel
d---	6/5/2021 3:10 PM		PerfLogs
d-r--	10/24/2022 1:38 PM		Program Files
d-r--	10/10/2022 9:07 AM		Program Files (x86)
d-r--	9/24/2022 11:35 AM		Users
d---	10/13/2022 6:40 PM		Windows
-a--	3/20/2022 7:08 PM	12288	DumpStack.log
-a--	10/24/2022 4:20 PM	12	index.html
-a--	10/24/2022 4:20 PM	12	index2.html
-a--	12/1/2006 10:37 PM	904704	msdia80.dll

## Monut Volume (-v)

-v parametresi ile docker containerimize volume mount etmek için kullanırız. Bu volume'ler ister docker host büzerinde oluşturulabilir, ister bir NTFS dosya sistemini kullanabiliriz. (Windows üzerinde dosya mapleme gibi düşünebiliriz.) ör;

Docker container run –volume /data1

--workdir “-w”

Bununla beraber diğer bir kullanacağımız parametre –workdir parametresidir. Mount ettiğimiz volume üzerinde map işlemi için kullanız. Ör;

```
Docker container run --workdir /"volumehos" --volume "volume host:container" "container"
```

## Docker Host überinden Mount

## Linux kernelde göre bi örnek

Docker hostumun C: dizininde myvolume olarak bir klasör oluşturuyorum. Bunu nginx container bağlamak istiyorum. Bunun için

File	LastWriteTime	Length	Name
---	3/6/2022 11:15 PM		desktop
---	11/23/2021 11:10 PM		DRIVERS
---	10/24/2022 4:31 PM		html
---	10/20/2022 9:30 PM		Intel
---	10/24/2022 5:26 PM		myvolume
---	6/5/2021 3:10 PM		PerfLogs
-r-	10/24/2022 1:38 PM		Program Files
-r-	10/16/2022 9:07 AM		Program Files (x86)
-r-	9/24/2022 11:35 AM		Users
---	10/13/2022 6:40 PM		Windows
a--	3/20/2022 7:08 PM	12288	DumpStack.log
a---	10/24/2022 4:20 PM	12	index.html
a---	10/24/2022 4:20 PM	12	index2.html
a---	12/1/2006 10:37 PM	904704	msdiabe.dll

```
container run -d -v C:\myvolume:/myvolume --workdir /myvolume nginx
```

Komutunu çalışıyorum. (--workdir “-w” parametresi maplemek için kullanıyorum)

```
PS C:\> docker container run -d -v C:\myvolume:/myvolume --workdir /myvolume nginx  
34b39df8c076887686b32ece096cba77d7ec81ecd5795bdacc8e900a80d8a14d
```



Containerimizn içerisinde girdiğimde mount edildiğini gördüm, arından içerisinde bir testvolume adında dosya oluştururdum.

```
root@34b39df8c076:/# dir
bin dev docker-entrypoint.sh home lib64 mnt opt root sbin sys usr
boot docker-entrypoint.d etc lib media myvolume proc run srv tmp var
root@34b39df8c076:/# df -Th
Filesystem      Type  Size  Used  Avail Use% Mounted on
overlay        overlay 251G  1.8G  237G  1% /
tmpfs          tmpfs   64M    0    64M  0% /dev
tmpfs          tmpfs   3.9G   0    3.9G  0% /sys/fs/cgroup
shm             tmpfs   64M    0    64M  0% /dev/shm
drvfs          gp     954G  373G  581G  40% /myvolume
/dev/sdd        ext4   251G  1.8G  237G  1% /etc/hosts
tmpfs          tmpfs   3.9G   0    3.9G  0% /proc/acpi
tmpfs          tmpfs   3.9G   0    3.9G  0% /sys/firmware
root@34b39df8c076:/#
```

Docker hostumda control ettiğimde klasörünü docker host'a yazdığını teyit etmiş oldum.

```
root@34b39df8c076:/myvolume# mkdir testvolume
root@34b39df8c076:/myvolume# ls
testvolume
root@34b39df8c076:/myvolume# exit
exit
PS C:\> cd /myvolume
PS C:\myvolume> ls

Directory: C:\myvolume

Mode                LastWriteTime         Length Name
----                <-----           <-----  --
d---       10/24/2022  5:31 PM            1K    testvolume

PS C:\myvolume>
```

### Volume Create ile Mount

Başa bir örnekte ise docker hostum üzerinde bir volume oluşturuyorum. Ardından ngnixx adlı containerimi bu volume bağlayarak oluşturuyorum. Kontrol ettiğimde bağlanmış şekilde görüyorum.

docker volume create vm2

docker container run --name ngnixx -d -v vm2:/vm2 --workdir /wm2 nginx

docker container exec -it ngnixx bash

df -Th

```
PS C:\Users\hllgo> docker container run --name ngnixx -d -v vm2:/vm2 --workdir /wm2 nginx
f7e7cb76a0e8:7f4b6f68b747dd2ce21d2697eef3a5d70e0124b1beefab6d03c1
PS C:\Users\hllgo> docker container exec -it ngnixx
"docker container exec" requires at least 2 arguments.
See "docker container exec --help".
Usage: docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]

Run a command in a running container
PS C:\Users\hllgo> docker container exec -it ngnixx bash
root@f7e7cb76a0e8:/wm2# df -T
Filesystem      Type  1K-blocks  Used Available Use% Mounted on
overlay        overlay 263174212 1787464 247948592  1% /
tmpfs          tmpfs   65536    0    65536  0% /dev
tmpfs          tmpfs   3987988   0    3987988 0% /sys/fs/cgroup
shm             tmpfs   65536    0    65536  0% /dev/shm
/dev/sdd        ext4   263174212 1787464 247948592  1% /vm2
tmpfs          tmpfs   3987988   0    3987988 0% /proc/acpi
tmpfs          tmpfs   3987988   0    3987988 0% /sys/firmware
root@f7e7cb76a0e8:/wm2#
```

Windows kernele göre örnü aynı şekildedir.

Docker container run -v C:\data:C\app Microsoft/iis

Ör:

Docker host'uma gelip

Docker create volume "volume name"

Daha sonra oluşturduğum volume'I bir container üzerine mount etmek için

Docker container run -d -v "dockervolumename:/tmp" (volum parametresini kullanarak mount etmiş oluyorum.)

### Internet ile dosya yükleme

Container üzerine 3. Bir dosya yükleme yolu container içerisinde wget paketini kurarak yada FTP yolu ile dosya yükleyebiliriz.

### Jenkins Container örneği

Docker pull Jenkins/Jenkins (Jenkins imajını docker hub reposistory'den docker sunucumuza indiriyoruz)

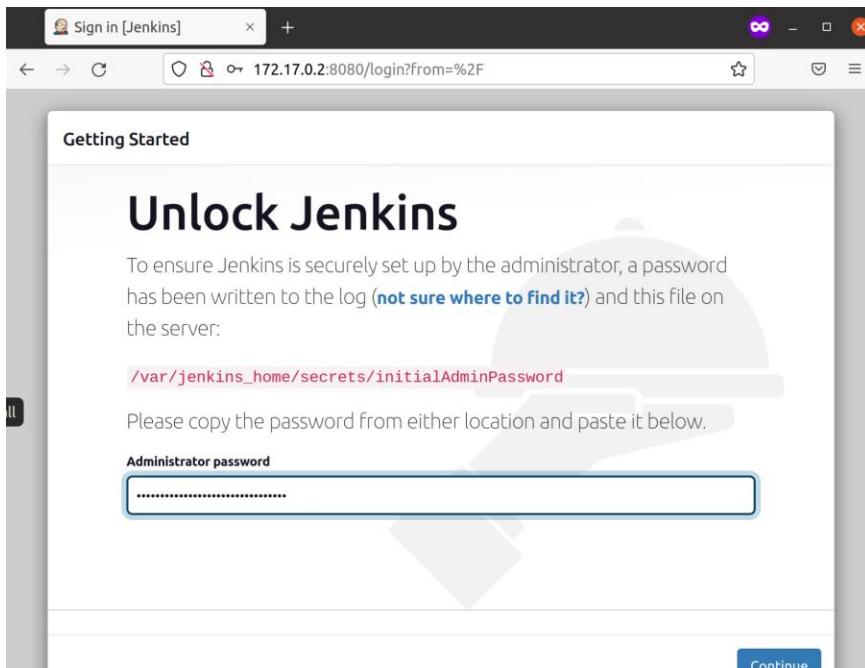
➤ Docker run Jenkins/Jenkins (ile kurulumu başlatıyoruz)

```
| Software Install |sktop$ docker ps
CONTAINER ID   IMAGE      COMMAND           CREATED          STATUS          PORTS     NAMES
506cbcfb2b3a   jenkins/jenkins "/usr/bin/tini -- /u..."  12 minutes ago   Up 12 minutes   8080/tcp, 50000/tcp   upbeat_mendel
```

```
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
```

```
a7df3ba97e9b442a952a9ae08a6dc7a5
```

```
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
```

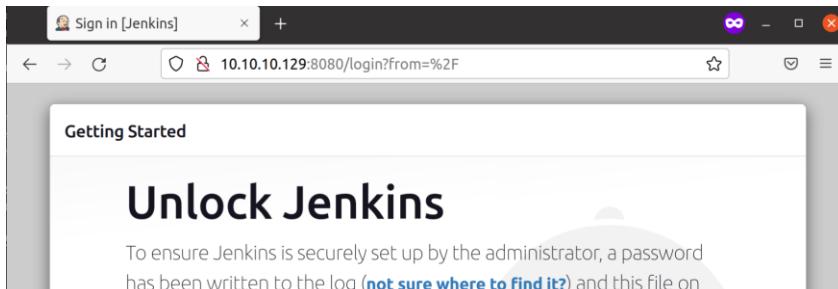


Containerimizi sonlandırip -p parametresi ile tekrar ayağa kaldırıyorum

Docker run -p 8080:80 Jenkins/Jenkins

Docker networkümüzden erişim için port map yaptık ve dışarıdan docker sunucumuzun IP ile erişim yaptığımızı görüyoruz. >docker run -p 8080:8080

172.17.0.2 container network / 10.10.10.129 docker hosta ait network



Burada bu şekilde tecrübe etmiş olduk şimdide containirimizi stop edelim

>docker stop "container ID"

Ben tekrar Jenkins containerini run ettiğimde yaptığım konfigasyon kaybolmuş olacak, Containerları doğar yaşar ve ölürlər deməştik, Ölmeden öncəse amel defterini bir yere yazalım ki sonradan ihtiyacımız olabilir.

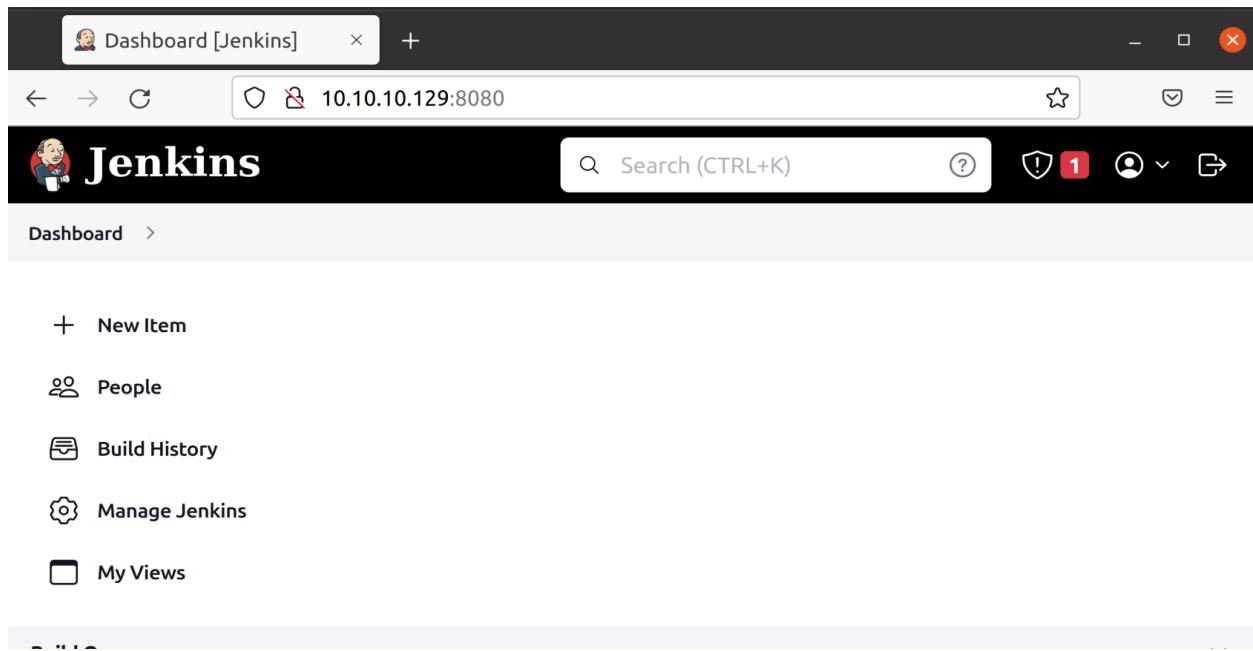
Bunun için ubuntu hostumda root dizininde "J" adında bir folder oluşturuyorum ve Jenkins containerimi run ederken volume mapping kullanıyorum

Writer: @HalilGÖKSEL

```
docker run -p 8080:8080 -v /root/J:/var/jenkins_home -u root jenkins/jenkins
```

```
root@ubuntu:~# docker run -p 8080:8080 -v /root/J:/var/jenkins_home -u root jenkins/jenkins
Running from: /usr/share/jenkins/jenkins.war
```

Kısa sürede oluşturdu, hemen bir installation yapıyorum, Jenkins üzerinde bir servis oluşturup bir kullanıcı oluşturuyorum.



Ardından docker inspect "docker id" ile volume mapping kontrolünü ile görebiliriz

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/root/J",
        "Destination": "/var/jenkins_home",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
]
```

/root/J dizinine geldiğimde Jenkins sistem dosyalarının olduğunu gördük

```
root@ubuntu:~/J# ls
config.xml
copy_reference_file.log
hudson.model.UpdateCenter.xml
jenkins.install.InstallUtil.lastExecVersion
jenkins.install.UpgradeWizard.state
jenkins.model.JenkinsLocationConfiguration.xml
jenkins.telemetry.Correlator.xml
jobs
nodeMonitors.xml
root@ubuntu:~/J# 
```

<code>nodes</code> <code>plugins</code> <code>secret.key</code> <code>secret.key.not-so-secret</code> <code>secrets</code> <code>updates</code> <code>userContent</code> <code>users</code> <code>war</code>	
--	--

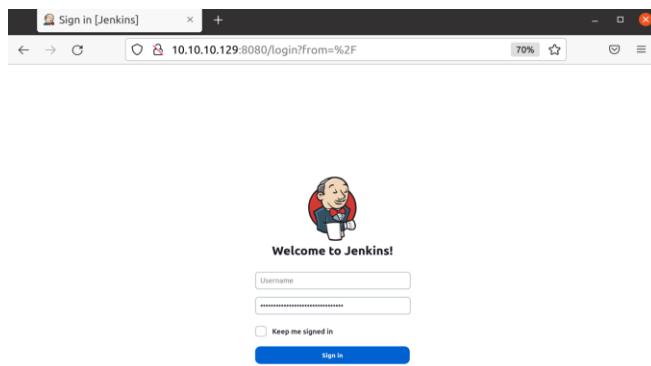
Writer: @HalilGÖKSEL

Daha sonra kontainirimi kill ediyorum, -V parametresi ile volume map yaptığım için datalarım kaybolmaması gereklidir aynı şekilde J klasörümü göstererek, containirimiz run ediyorum.

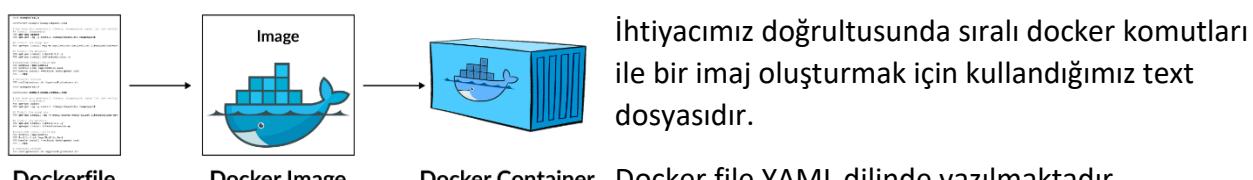
```
root@ubuntu:~# docker run -p 8080:8080 -v /root/J:/var/jenkins_home -u root jenkins/jenkins
```

Running from: /usr/share/jenkins/jenkins.war

Calistirdıktan sonra yeni bir kurulum olarak ekran karşılamıyor, oluşturduğum kullanıcı ile oturum açtığını ve oluşturduğum projeyi görüyorum.



## Docker File

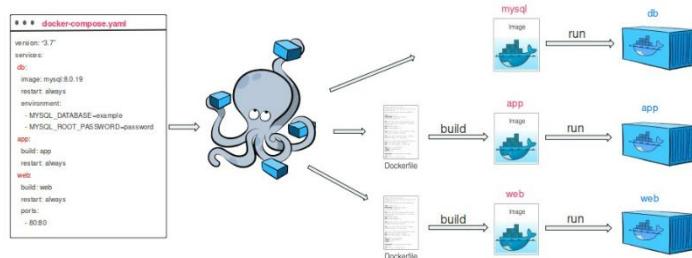


Docker file YAML dilinde yazılmaktadır.

Satırlardan oluşur her satırda bir işlem kaydetmektedir.

Örneğin bir nginx containerı ayağa kaldırırmak isteniyor, tek tek containerda işlem yapmak docker file üzerinde tek seferlik çalışıp bunu imaja dönüştürdüğümüzde işlerimiz daha kolaylaşacaktır.

Writer: @HalilGÖKSEL



kaldırmak mümkündür. İleriki konularda compose detaylı yazacağım.

## Docker File Komutları

### SHELL & EXEC

Docker file üzerinde 2 farklı yazım metodu kullanılmaktadır. Bunlar shell ve exec metodlarıdır

**SHELL;** Container üzerinde çalıştırılması istenen komutlar docker file üzerinde doğrudan yazılmaktadır.

**RUN apt-get update**

**RUN apt-get install openssh-client**

**RUN systemctl stop ssh**

**EXEC;** JSON yazım formatındaki fark her bir komut köşeli parantez ve ile yazılmakta, tırnak ile değerler belirtilmekte virgül ile ayrılmaktadır

**RUN ["apt-get", "updaate"]**

**RUN ["apt-get", "install","openssh-client"]**

**RUN ["systemctl","stop","ssh"]**

### FROM

Kullanacağımız imaj dosyasını belirtmek için kullanırız. Sistemimizde imaj yoksa dockerhost üzerine pull edecktir. (FROM "image":tag) ör

**FROM python:2.7**

**FROM ubuntu:latest**

**FROM Microsoft/iis:nanoserver**

### RUN

Container üzerinde komut çalıştırırmak için kullanılır. (RUN komut)

**FROM nginx:latest**

**RUN apt-get update && apt-get install -y nginx**

## ADD

Container içerisinde dışarıdan yani dockerhost üzerinden yada internet üzerinde data aktarmak için kullanılır (ADD kaynak dosya container hedef dosya)

```
ADD web1 /data/web1
```

```
ADD http://www.program/programlar/xx.exe /var/xx.exe
```

## COPY

ADD komutu ile aynı işlev sahip olan komut tek farkı internet ortamından dosya aktarılması yapılamamaktadır. (COPY “kaynak” “hedef”)

```
COPY html /var/www/html
```

```
COPY konf .py C:\programfiles
```

```
COPY etc/system* /etc/system (source belirtirken * kullanırsak system içeren tüm dosyaları alacaktır)
```

## WORKDIR

Container içerisinde çalışma dizini oluşturmak için kullanılmaktadır, Yani workdir ile oluşturduğunuz bir dizinde tüm işlemler workdir dizininde sağlanacaktır (WORKDIR “path”)

```
WORKDIR /web1
```

```
WORKDIR C:\APP1
```

## VOLUME

Daha önceki bildiğimiz gibi container içerisinde depolama alanı sağlamak için kullanılır (VOLUME “mountpoint”)

```
VOLUME /data
```

```
VOLUME C:\data
```

## EXPOSE

Container üzerinde port açma işlevi yapar, --Publish komutu gibi düşünebiliriz, Containerların belirtilen port üzerinden talepleri dinlemeyi sağlar. Default olarak TCP protokolünü kullanır, port/UDP eklerse UDP olarak listening durumda olur. (EXPOSE “PORT”/”PROTOKOL”)

```
EXPOSE 8080/UDP
```

```
EXPOSE 443/TCP
```

## LABEL

Metadata eklemek için kullanılır, açıklama gibi düşünebiliriz. (LABEL value)

```
LABEL version=1.0
```

**LABEL HalilGOKSEL****USER**

Container üzerinde işlem yapacak kullanıcıyı belirtir. (USER “user”[：“usergroup”])

**USER halil[:admin]****USER Göksel****ENV**

Daha öncede gördüğümüz gibi oluşturulan container için ortam değişkeni kullanılmak için kullanılır (ENV “key” “value”)

**ENV MYSQL\_ROOT\_PASSWORD='Password1'****CMD**

Container oluşturulduğunda çalıştırılması istenen komut olarak kullanılır. Run komutu ile benzer olan komutun run’dan farklı Container ilk ayağa kalktığında CMD komutunu çalıştırmaktadır, RUN ise çalışan container üzerinde yürütülür. Docker file bir kere CMD komutu kullanılmaktadır, birden fazla kullanırsanız sadece son yazdığınız komutu çalıştıracaktır.(CMD “komut”)

**CMD script.py****CMD [“python”, “script.py”]****ENTRYPOINT**

CMD komutu ile aynı işlev sahip olup ENTRYPOINT komutu daha esnek kullanılmaktadır. CMD komutu ile kullanılabilirliktedir. (ENTRYPOINT “value”)

**ENTRYPOINT echo merhaba****ENTRYPOINT[“ping”, “8.8.8.8”, “-t”]****ENTRYPOINT[“ping”]****CMD[“8.8.8.8”, “-t”]**

Tüm dockerfile komutlarına bakmak için dökümana bakabilirsın.

[Dockerfile reference | Docker Documentation](#)

# Docker File Örnekleri

## Docker File Örnekleri yapalım.

## 1. Örnek

Alpine üzerine figlet yükleyerek ekranıma yazı yazdırması için bir dockerfile yazıyorum.

```
FROM alpine:latest  
RUN apk update & apk add figlet  
ENTRYPOINT [ "usr/bin/figlet", "Halil GÖKSEL" ]
```

(ENTRYPOINT yerinde CMD komutunuda kullanabilirdik.)

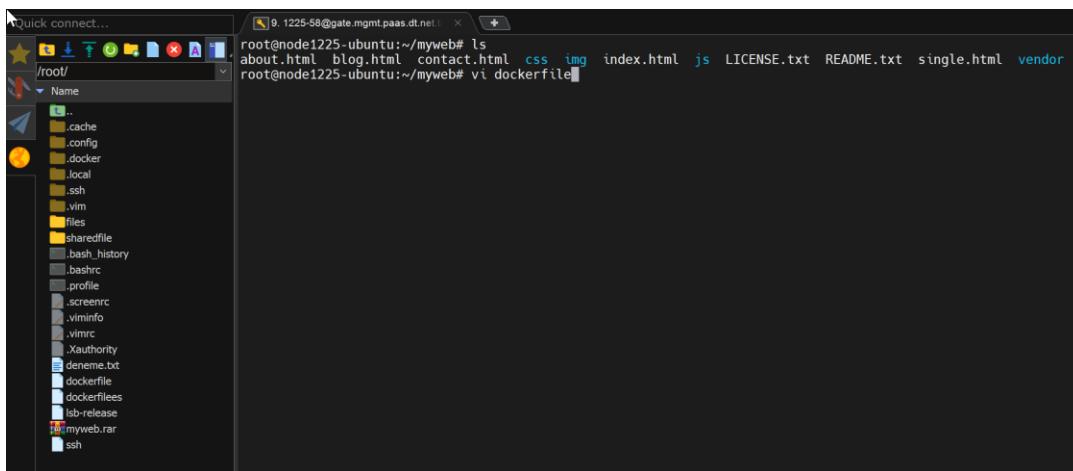
```
ubuntu@ubuntu:~$ docker image build --tag hg . (Dockerfile'ımı build ediyorum.)
```

Ardından oluşturulan imajı çalışıyorum. Görevini tamamlıyor ve container sonlanıyor.

## 2. Örnek

(Bir html web dosyasını, Ubuntu containeri üzerinde nginx kurup ayağa kaldıracağız)

1. İlk önce webprojemizi dockerhostumuzun içerisinde atıyorum.
  2. Ardından unrar ettikten sonra projemin içerisinde vim ile dockerfile dosyası olusturuyorum.



3. Docker file dosyasımı yazıyorum.

```

1 FROM ubuntu:latest
2 RUN apt-get update
3 RUN apt-get install -y nginx
4 WORKDIR /var/www/html
5 COPY .
6 EXPOSE 80/tcp
7 CMD ["nginx","-g daemon off;"]

```

4. Docker file tamamladıktan sonra build etmem gerekiyor yani dockerfile'dam bir imaj oluşturmalıyım bunun için docker image build komutunu kullanıyorum build ederken kullandığımız “.” Nokta dockerfile dosyamızın bulunduğuumuz dizinde olduğunu ifade etmektedir. Farklı biz path üzerinde ise yolunu yazmaliyiz.

```
root@node1225-ubuntu:~/myweb# docker image build --tag websitem .
```

5. Sorunsuz bir şekilde tamamlandığını görüyorum imajlarımı listelediğimde build ettiğim projem imaj olarak duruyor.
6. Bir sonraki adımda imajını container run diyerek canlıya alıyorum, 8080 portu üzerinden control ettiğimde websitemin yayınlandığını gorebiliyorum.

```

root@node1225-ubuntu:~/myweb# docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
websitem            latest   d0cead6bcb2c  2 minutes ago  143MB
root@node1225-ubuntu:~/myweb# docker container run -dp 8080:80 websitem

```

3. Windows kernele göre bir örnek yapacak olursaka

```

FROM micorosoft/iis
RUN powershell -NoProfile -Command Remove-Item -Recurse C:\inetpub\wwwroot\*
COPY . C:\inetpub\wwwroot
EXPOSE 80
~_

```

1. Katmanda iis imajı ayağa kaldırıyoruz
2. Katmanda iis içerisindeki dosyaları siliyoruz
3. Localde bulunan dosyaları, wwwroot dizinine atıyoruz
4. Uygulamamızın Listenin portunu belirliyoruz

### Docker File Yazarken Dikkat Edilmesi Gerekenler

1. İmajlar katmanlar halinde oluşur demşikin, Kullandığınız her bir RUN komutu bir katman oluşturur buda imajınızın gereksiz olarak büyümeye yol açmaktadır, Kullanacağınız komutlar arasında && kullanarak tek satırda oluşturabilirsiniz.
2. COPY komutunu kullanırken sadece ihtiyaç duyduğumuz dosyaları imaj içinde bulundurmamalıyız.
3. Docker File yazarken işlemler sıralı gitmesi gerektiği gibi yazmalıyız, aksi takdirde hata alırız
4. Bir imaj kullanırken veya oluştururken imajın boyutunu minimum seviyede tutmak daha verimli olacaktır.

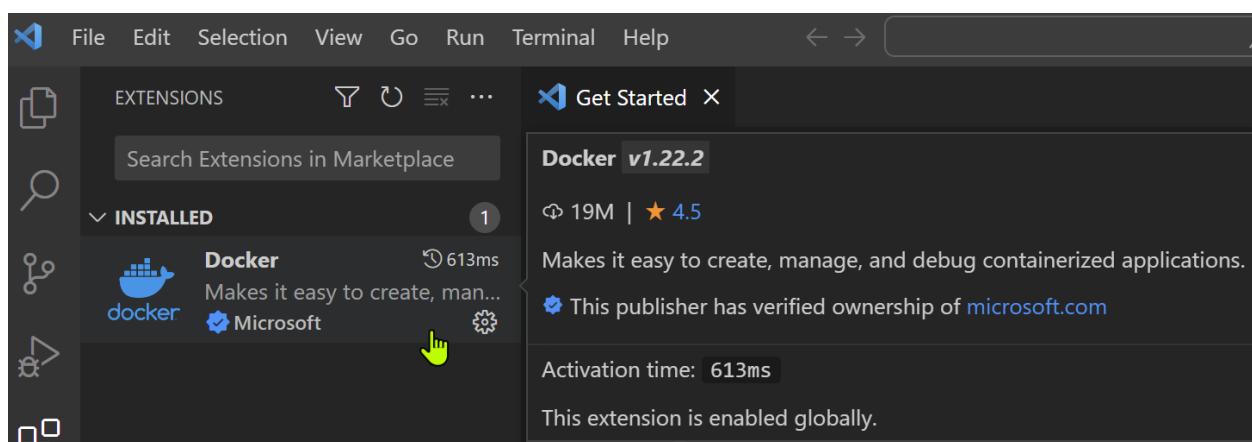
★ Son olarak docker file üzerinde multi-stage kavramı vardır. Eğer bir docker file üzerinde birden fazla from kullanılıyorsa docker-file multistage yapısındadır. Hem performans hemde boyut açısından çok katmanlı dockerfile yazımı tercih edilmektedir. Detaylar google da.

### VisualStudio Code ile Docker File

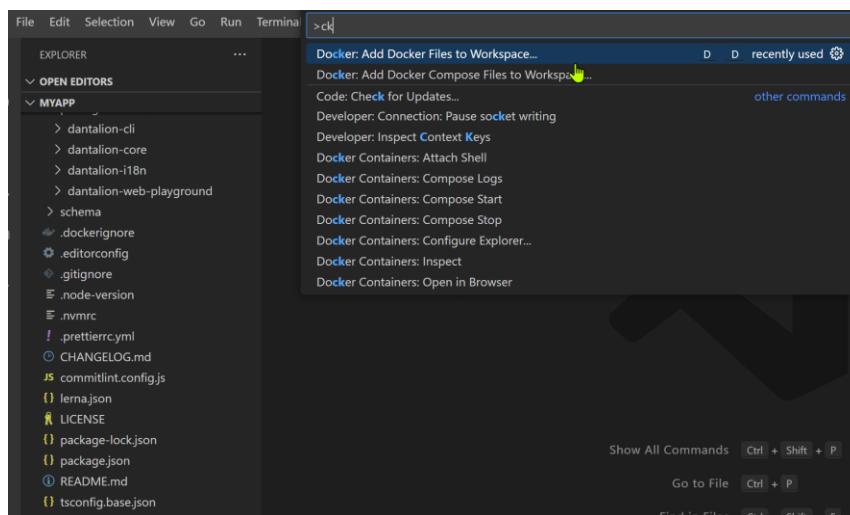
En Kolay Docker File yazma araçlarından biri VisualStudio Code dur.

[Download Visual Studio Code - Mac, Linux, Windows](#) linkinden indirelim

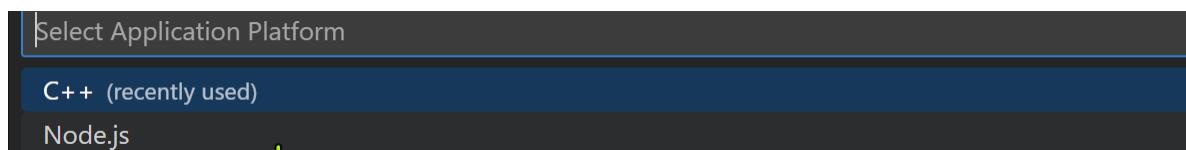
Crtl + Shift + X ile Docker paketini üzerine yükleyelim



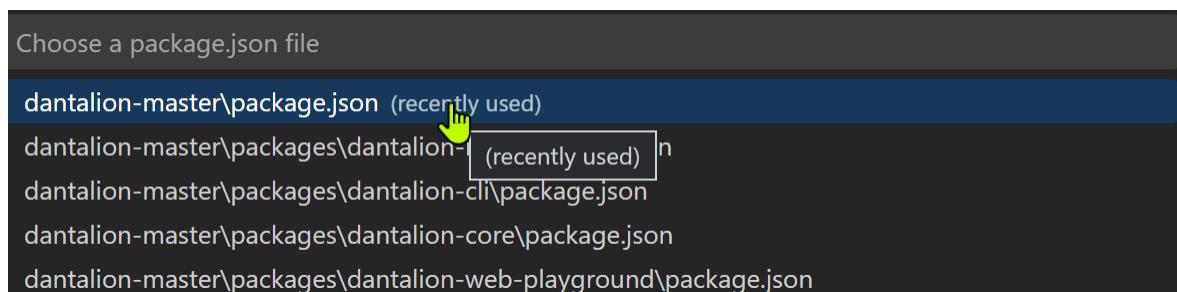
Projemizin açtıktan sonra ardından Crtl + Shift + P kısayolu ile kısa yoldan "Docker Add Docker Files to Workspace " diyerek dockerfile yazmaya başlayalım.



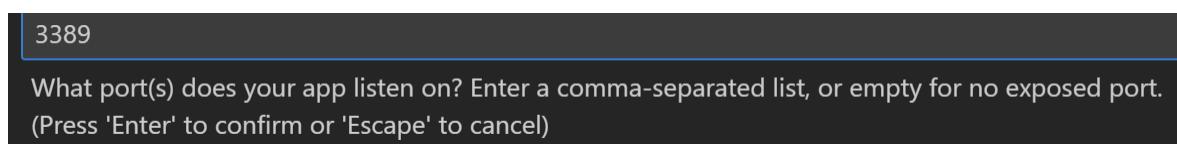
Projemizin dilini seçiyoruz



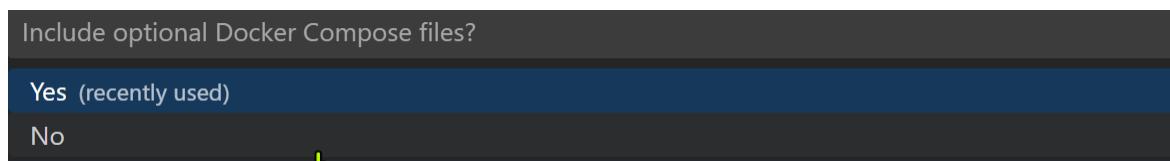
Üzerinde çalışacağımız js projesini seçiyoruz



Dinlenecek Portunu veriyoruz



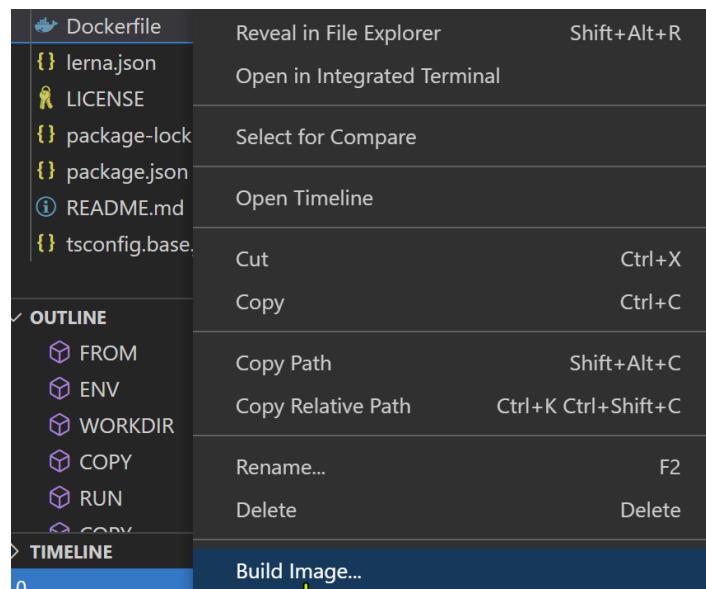
Docker Compose olmadığını No seçiyorum



Göründüğü gibi saniyeler içinde docker file yazdı.

```
dantalion-master > 🐳 Dockerfile > ⚙ FROM
1  FROM node:14-alpine
2  ENV NODE_ENV=production
3  WORKDIR /usr/src/app
4  COPY ["package.json", "package-lock.json*", "npm-shrinkwrap.json*", "./"]
5  RUN npm install --production --silent && mv node_modules ../
6  COPY .
7  EXPOSE 3389
8  RUN chown -R node /usr/src/app
9  USER node
10 CMD ["npm", "start"]
11
```

Tek tuşla build'de edebiliriz. (Arka planda dockerımızın çalıştırmayı unutmayalım)



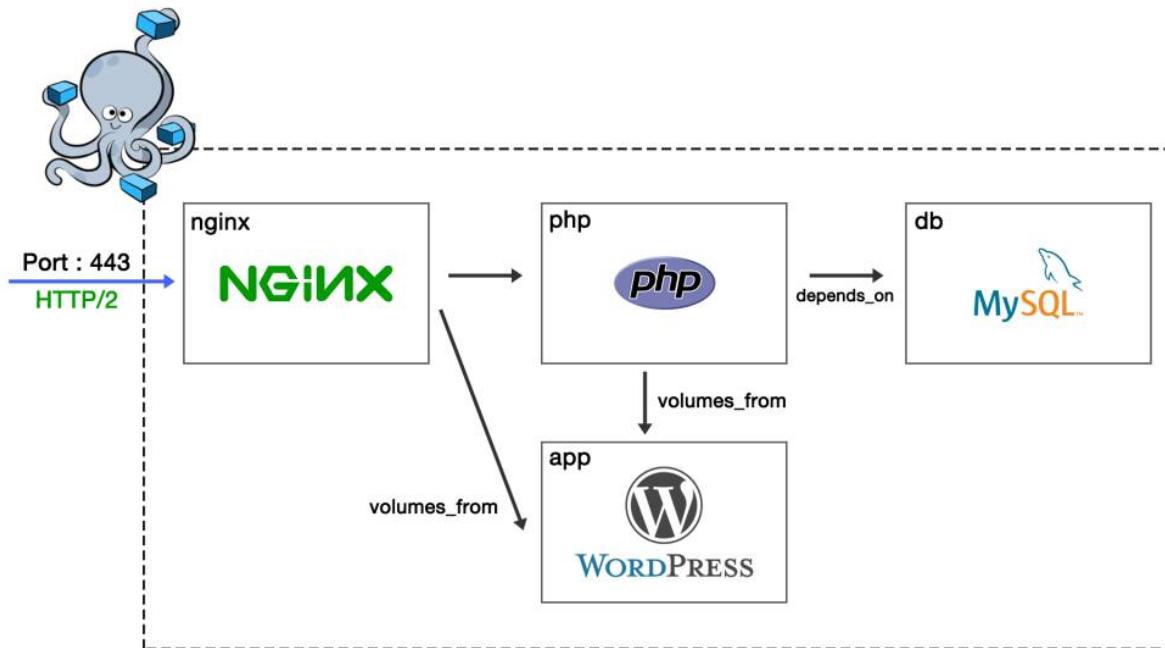
İmajı çalıştırıp başlattığımızda sorunsuz şekilde ayağa kaldırabiliriz (tabi bende hata Verdi :github üzerinden çektiğim bi projeydi bir incelemek lazım bi yemeğe gidip geliyorum :)

## Docker Compose

Docker compose özetçe multi container ihtiyacımız olan bir projede docker uygulamalarını birbirleyi ile tanımlama ve çalışma işlemlerini tek bir text dosyasında sağlayan bir araç diyebiliriz.

Docker compose oluşturmak için kullanılan text dosyası YAML formatındadır. Versiyon konfigrasyon dosya adı docker-compose.yml'dır. Aşağıdaki gibi örnek bir multi container yapıdaki mimaride tek bir docker compose dosyası bulunmaktadır.

Konfigrasyon dosyasında bir değişiklik yapılmadığı sürece, mimari üzerinde yeni bir container ekleme çıkarma bağlama vs işlemleri yapılmaz.



Sistem üzerinde docker compose yüklü olup olmadığını control etmek için >docker-compose version komutu çalıştırabiliriz. Version çıktısı veriyor ise yüklüdür.

Yüklü değil ise kurulum için docs.docker destek alabilirsiniz. [Overview | Docker Documentation](#)

### Docker-Compose Kurulumu

Ubuntu üzerine docker-compose kurulumu

```
> curl -SL https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-  
linux-x86_64 -o /usr/local/bin/docker-compose  
  
>sudo chmod +x /usr/local/bin/docker-compose  
  
>sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose  
  
>docker-compose version
```

## Docker Compose Komutları

```
root@node1225-ubuntu:~# docker compose version
Docker Compose version v2.10.2
root@node1225-ubuntu:~# docker compose

Usage: docker compose [OPTIONS] COMMAND

Docker Compose

Options:
  --ansi string            Control when to print ANSI control characters ("never"|"always"|"auto") (default "auto")
  --compatibility
  --env-file string        Specify an alternate environment file.
  -f, --file stringArray   Compose configuration files
  --profile stringArray    Specify a profile to enable
  --project-directory string Specify an alternate working directory
                            (default: the path of the, first specified, Compose file)
  -p, --project-name string Project name

Commands:
  build      Build or rebuild services
  convert    Converts the compose file to platform's canonical format
  cp         Copy files/folders between a service container and the local filesystem
  create     Creates containers for a service.
  down       Stop and remove containers, networks
  events     Receive real time events from containers.
  exec       Execute a command in a running container.
  images     List images used by the created containers
  kill       Force stop service containers.
  logs       View output from containers
  ls         List running compose projects
  pause      Pause services
  port       Print the public port for a port binding.
  ps         List containers
  pull       Pull service images
  push       Push service images
  restart   Restart service containers
  rm         Removes stopped service containers
  run        Run a one-off command on a service.
  start     Start services
  stop      Stop services
  top        Display the running processes
  unpause   Unpause services
  up         Create and start containers
  version   Show the Docker Compose version information
```

### **docker-compose up**

(Bu komut docker-compose file içerisinde servislerin başlatılmasını eğer servisler yok ise oluşturduktan sonra başlatılmasını sağlamaktadır. Default olarak aranan dosya docker-compose.yml dosyasıdır.

**>docker compose up –help** / ile kullanılan parametrelere erişmek için –help parametresini kullanabilirsiniz.

**>docker compome up -f** / eğer farklı bir isim ile kaydedilmiş ise -f parametresini kullanabiliriz  
(**docker compose up -f ....yml**)

**>docker compose up -no-recreate** / Var olan servislere dokanmadan yeni servisleri ayağa kaldırınmak için no-recreate parametresini kullanabiliriz.

**>docker compose up –force-recreate** / Bu parametre ilede servisleri yeniden oluşturmaya zorlamak için kullanılır

**>docker compose up –remove-orphans** / Bu parametre ile eski kullanılmayan yani .yml dosyası içerisinde içermeyen servisleri kaldırır.

### **>docker-compose down**

Bu komut ile .yml üzerinde tüm çalışan container ve networkler durdurularak kaldırılır.

**>docker-compose down –help**/komutu ile parametreleri inceleyebilirsiniz.

**>docker-compose down –rmi all /** parametresi ile .yml ile sisteme indiriler imaj dosyalarında silinir

**>docker-compose ps**

Docker-compose ile oluşturulan servisler listelenmektedir.

**>docker-compose exec**

Docker-Compose ile çalışan container içerisinde komut çalıştırmak için kullanılır

**>docker-compose build**

Docker-Compose dosya içerisinde belirtilen dockerfile build etmek için kullanılır

**>docker-compose stop**

Docker-Compose komutu down komutunun aksine ile çalışan container silmeden durdurabiliriz. Tüm servisleri durdurmak için docker-compose stop, Belirtilen servisler için stop ‘dan sonra servis adını belirtmeliyiz.

**>docker-compose start**

Docker-Compose stop komutu ile durdurulan servisleri başlatmak için kullanılır

**>docker-compose run**

Docker-Compose run komutu ile servis üzerinde komut çalıştırılabilme imkanı sunmaktadır, --help parametresi ile parametrelerini inceleyebiliriz

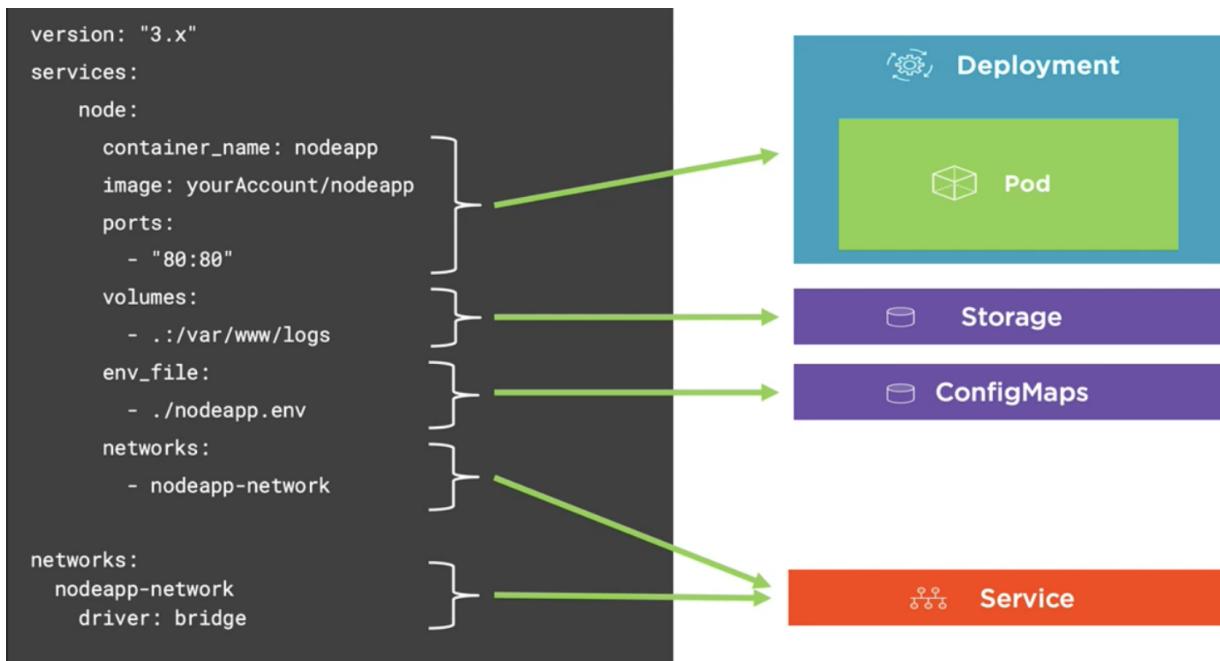
**>docker-compose rm**

Docker-Compose rm komutu ile durmuş servisleri silebiliriz, yine –help parametresi ile detaylarına bakabiliriz

**>docker-compose logs**

logs komutu ile containerların oluşturmuş olduğu logları inceleyebiliriz

Örnek bir docker compose yaml dosyasını incelersek,



**Docker Compose yml yazarken sık kullanılan komut türlerini incelersek;**

**Image:** oluşturulmasını istediğimiz containerin imaj bilgisi vereceğimiz komuttur.  
(image:ubuntu:14.0)

**Ports:** Container oluştururken kullandığımız gibi port mapping için kullanırız ilk value host ikinci value container portunu belirtmekte(ports:"8080:8080")

**Environment:** Yine aynı şekilde ortam değişkenlerini tanımlamak için kullanırız.(DB\_USER: admin DB\_PASSWORD:test123)

**Networks:** Networks komutunu kullanarak containerlarımızın oluşturduğumuz network içerisinde dahil edilmesi sağlar.

**Build:** Daha önce oluşturduğumuz dockerfile dosyasını işleme almak için kullanılır. (local dizin "build: .", localdizindeki etc klasörü altındaki x dockerfile adı dosya için;  
"build: ./etc  
dockerfile: x-dockerfile"

**volumes:** Container içerişine data volume yada bind volume ataması için kullanılır.

**Links:** Links komutu ile ilgili containerlar arasında link atılmasını yani erişmesini sağlar  
(links: container:container)

**Command:** Komut çalıştırılmak için kullanılır.

**Expose:** Containerların belirtilen portlar üzerinde servis yayılmasını sağlar, Networks'den farklı host üzerinde yayılama yapmaz, Containerlar arasında bağlantı için kullanılır

**DNS:** dns komutu kullanarak containerlara dns tanım girebilriiz

**Working\_dir:** Bu parametre ile container içerisinde çalışma dizini değiştirmek için kullanılır

**User:** Yine bildiğimiz gibi container içerisinde default user atama için kullanılır.

**Depends\_on:** Bu komut ile container servisleri önceliklendirmek için kullanılır, Örneğin bir projede ilk başlatılması gereken servisler, önceliklendirilmesi gereken servisler var ise kullanılır.

**Volumes\_From:** Farklı containerlar içerisindeki volumeleri birbirlerine bağlamak için kullanılır

**Restart:** Bir containerin yeniden başlatma yapılandırmasını düzenlenebiliriz

### Docker Compose Örnekleri

Docker-compose kullanarak nginx ayağa kaldırıralım. Versiyonumuzu belirip servisimize bir isim veriyoruz. Image ile container imajını belirtiyoruz. Ports ile listening portunu veriyorum.

Kaydedip çıkış yaptığımda mevcut dizinimde

```
>docker-compose up komutunu çalıştırıyorum ve docker-compose ile oluşturduğum container ayakta.
```

```
version: '3'

services:
  web:
    container_name: myweb
    image: nginx:latest
    ports:
      - "8080:80"
```

Docker-compose ls parametresi ile dockerhostum üzerinde composeleri listeleyebilirim. Oluşturduğumuz örnek en basiti, multi containerlar için kullanılır demistik birden çok containeri yönetmek için docker-compose oluşturur ve ayağa kaldırırız.

Bir diğer örneğimiz bir wordpress uygulaması ayağa kaldırırmak.

Bunun için ihtiyacım olanlar, wordpress containeri, wordpree için mysql containeri ve mysql yönetimi için phpMyAdmin containeri olacaktır.

Bu containerlar birbiri ile bağlı çalışıp ayağa kalkacaktır.

Dockerhostum üzerinde mywordpress adında bir klasör oluşturuyorum, ardından docker-compose-yml dosyası oluşturup komutlarımı yazmaya başlıyorum.

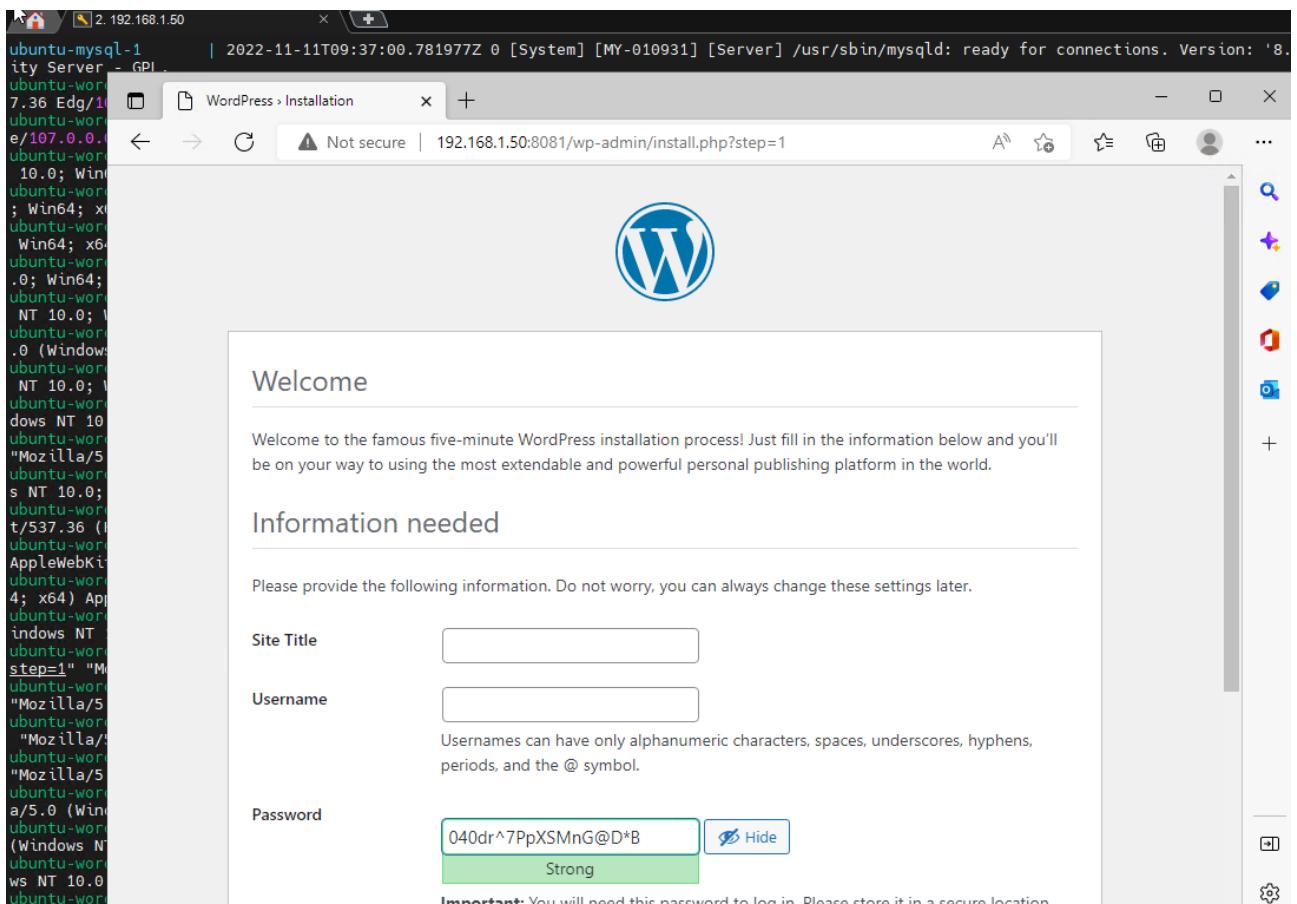
```
services:  
    #version değerini giriyorum, mysql sürümünü 8.0 olarak belirtiyorum. ardından mysql klasörünü volume olarak atıyorum.  
    #environmentleri giriyorum.  
mysql:  
    image: mysql:8.0  
    volumes:  
        - mysql_data:/var/lib/mysql  
    restart: always  
    environment:  
        MYSQL_ROOT_PASSWORD: root  
        MYSQL_DATABASE: wordpress  
        MYSQL_USER: admin  
        MYSQL_PASSWORD: wordpress  
    volumes:  
        - ./mysql:/var/lib/mysql  
        #bu aşamada mysql projemin db yönetimi için phpmyadmin imajı oluşturuyorum. Bağlılığı belirtmek için depends_on komutunu kullanıyorum.  
        #environmentleri belirterek lintening port ataması yapıyorum.  
phpmyadmin:  
    image: phpmyadmin/phpmyadmin  
    depends_on:  
        - mysql  
    environment:  
        PMA_HOST: mysql  
        PMA_PORT: 3306  
        PMA_ARBITRARY: 1  
    restart: always  
    ports:  
        - 8180:80  
        #son olarak wordpress imagını oluşturuyorum, yine bağıllık belirtiyorum. ardından port atması yapıyorum. environment değerlerini giriyorum  
wordpress:  
    depends_on:  
        - mysql  
    image: wordpress:latest  
    ports:  
        - "8081:80"  
    restart: always  
    environment:  
        WORDPRESS_DB_HOST: mysql:3306  
        WORDPRESS_DB_USER: admin  
        WORDPRESS_DB_PASSWORD: wordpress  
        WORDPRESS_DB_NAME: wordpress  
    volumes:  
        - ./wordpress:/var/www/html  
  
volumes:  
    mysql_data: {}  
    #son olarak kullandığım volume için atama compose içinde oluşturuyorum.  
    #çalışma tamamlandı docker-compose up komutu ile çalıştırılabilir
```

Docker compose dosyamı bu şekilde kaydediyorum, ardından >docker compose up komutumu çalıştırarak, docker-compose ayağa kaldırıyorum.

Sorunsuz şekilde çalıştı, docker ps diyerek çalışan containerlarımı görebilirim.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
b99ca17d9052	phpmyadmin/phpmyadmin	"./docker-entrypoint.s..."	About a minute ago	Up 14 seconds	0.0.0.0:8180→80/tcp, :::8180→80/tcp
e41a111873bd	wordpress:latest	"./docker-entrypoint.s..."	About a minute ago	Up 14 seconds	0.0.0.0:8081→80/tcp, :::8081→80/tcp
9ea3d16050ee	mysql:8.0	"./docker-entrypoint.s..."	About a minute ago	Up 15 seconds	3306/tcp, 33060/tcp

Tarayıcımda kontrol ettiğimde wordpress sayfama ulaşıyorum. Pyhmyadmin ile giriş yaptığında veritabanı oluşturabildiğimi mysql container yazdığını görmek mümkün.



## Docker Swarm

Docker swarm docker'ın geliştirdiği bir orkestrasyon aracıdır. Yani birden fazla docker hostlu ortamımızda, çalışan docker hostları, servisleri ve containerları yönetmek, cluster yönetimini sağlamak için kullanılan bir orkestrasyon aracıdır

Dockerswarm, containerlarımız için bir high availability ortamı sağlamaktadır. Bir docker hostumuzun crash olması durumunda fiziksel yada yazılımsal bir sorun olması halinde üzerinde bulunan tüm containerlar etkilenecek buda bizim için kriz yaratacaktır.

Docker Swarm ile, hostlarımızın arasında bir cluster yapısı oluşturulabilir, Kesintisiz hizmet için bu yapıyı kurabiliriz

Docker bize 2 mod olarak gelmektedir, Bunlar single mode ve swarm mode'dur. Single mod şimdije kadar kullandığımız standalone yapısıdır. Swarm mode docker host üzerinde inactive şekilde yüklü gelmektedir.

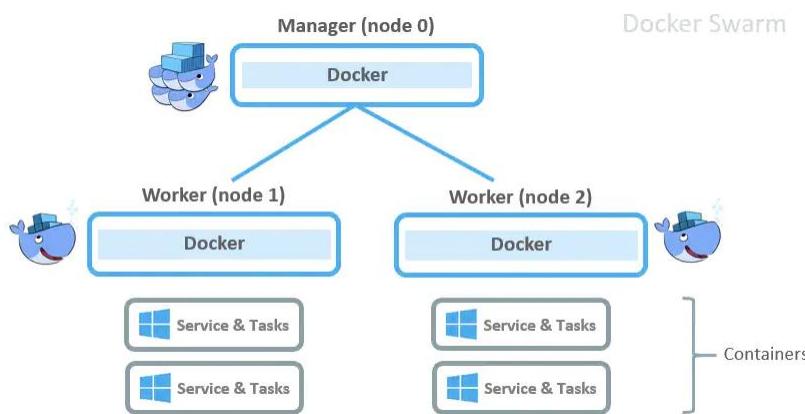
Single mode olarak gelen tüm servisleri, bir swarm yapısı oluşturmak için swarm servisini aktif etmeliyiz, böylece bir cluster yapısı oluşturacak, tüm donanım kaynakları havuz halinde toplanacaktır.

Swar içerisindeki bir docker node üzerinde meydana gelen bir sorunda üzerinde bulunan containerları farklı node'lar üzerine aktarılacaktır.

Altarnetif olarak Kubernetes de günümüz orkestrasyon teknolojilerinde yer alan bir araçtır. Çalışma prensipleri benzer fakat sağladıkları koşul ve özellikler farklıdır.

Farklı orkestrasyon deginecek olursak bunlar,

1. Swarmkit, üzerinde çalıştığımız
2. Kubernetes, en popülerlerinden
3. Amazon ECS
4. Azure Container Service gibi tooları sayabiliriz.



Swarmkit üzerinde 2 mode yapılandırması mevcuttur bunlar manager ve worker mode'lardır. Manager node swarmkit'in ve diğer workerların yönetiminden sorumludur. Ve swarmkit üzerinde serfitikasyon yönetimi yapmaktadır, Worker ise clusterin nodelerindendir. Swarmkit üzerinde birden fazla manager mode olabilir.

Bir swarmkit yapısında birden fazla manager node var ise bu nodular 2 ye ayrılmaktadır. Leader manager node ve Follower Manager node.

Leader manager node, swarmkit'in kurulumunu yapan node'dur. Tüm yapı leader node üzerinden yönetilir. Eğer leader manager node üzerinde bir hata alınırsa ortamdaki follower manager nodlardan birtanesi leader manager node'un yerini alır.

Çoklu node'lu bir ortamda, leader node üzerinde yapılan bir aksiyon yada değişiklik follower node'lar üzerinde onay alınmaktadır. Bu onay işleminin ardından cluster üzerinde işler yürümektedir.

Worker Node'ları ise leader ve manager node'ları dinlemekte aynı zamanda workerlar arasında bağlantı sağlamaktadır.

Docker Host üzerinde swarmkit pasif geliyor demistik bunu aktif etmek için

**>Docker-swarm init** komutunu kullanabiliriz.

## Docker Swarm Komutları

Tüm komutları listelemek için docker host üzerinde **>docker swarm** komutunu çalıştırabiliriz. Komut detaylarını incelicek olursak

**>docker swarm init** (init komutuyla ortamımızda yeni bir swarm oluşturabiliriz. –help ekleyerek parametrelerini listelemek mümkün.)

**>docker swarm init –adversite-addr** (--adversite-addr parametresi ile yayın yapılacak adresi belirtebiliriz ör, **>docker swarm init –adversite-addr 10.10.100.2**)

Init komutundan sonra docekr hostumuz artık lead manager modunda çalışacaktır. Komut satırında worker'ları eklemek için bize bir token yaratacaktır.

**>docker swarm join-token worker veya manager** (Bu komut dockerswarm oluşturduğumuz lead manager üzerinde çalıştırıldığında bizlere token bilgisi verecektir. Sisteme yeni bir worker yada yeni bir manager eklemek istersen token bilgisini bu şekilde alabiliriz. Worker ve manager tokenları farklıdır.)

**>docker swarm join** (swarm ortamına worker, manager node eklemek için kullanılır. –help kullanarak farklı parametrelerini görebiliriz, bir manager node eklemek istersek **>docker swarm join –token “tokenid”** eklememiz yeterlidir.)

**>docker swarm leave** (swarm ortamına ekli manager veya worker node üzerinde kullandığımızda swarm ortamından çıkar)

**>docker swarm update** (Update komutu swarm ortamımızdaki settingleri kontrol etmek için kullanız --help parametresi ile detaylarını inceleyebiliriz)

## Docker Node Komutları

**>docker node ls** (Bu komut swarm üzerinde bulunan nodları listelemek için kullanılır, Node ile beraber status bilgileri ve (manager (leader - follower), worker) modlarını ve version bilgilerini göstermektedir.)

ID'den sonra gelen yıldız ise komutu çalıştığımız ilgili node olmaktadır.

**>Docker node promote** (Bu komut worker nodumuzu manager olarak atamamızı sağlar.)

>Docker node demote (bu komut'da promote tam tersi manager nodumuzu worker mode almamızı sağlar.)

Yukarı promote ve demote node mod değişikliğini >docker node update –role string parametresi ile yapabiliriz.

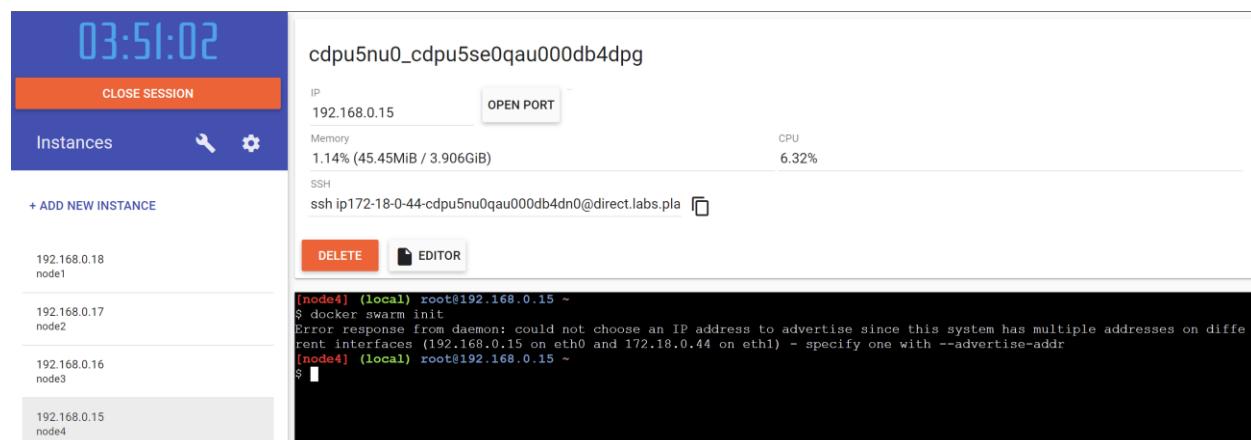
>docker node update –availability ...(drain, pause, ative) “node name” ile node statusunu değiştirebiliriz.

Docker node ps “node name” ps komutu ile node üzerindeki task'ları listelemektedir.

## Docker Swarm Kurulumu

Labratuvardan giriş yaptım ve 4 adet instance oluşturduğum 2 worker 2 master için bir swarm ortamı oluşturacağım. Bunun için ilk nodumun üzerinde docker swarm init komutu ile swarm aktif etmek istiyorum bana bir error veriyor.

Hatanın nedeni instance üzerinde birden fazla NIC bulunmakta, bu niclerden swarmın kullanacağı interface seçmeliyim bunun için aşağıdaki komutu kullanıyorum.



>docker swarm init –advertise-addr 192.168.0.15

Swarmı aktif etti ve bana tokenimi Verdi. Verdiği token node eklemek içindir, Manager token farklıdır.

Worker Node için > docker swarm join-token worker

Manager Node için > docker swarm join-token manager

```
$ docker swarm init --advertise-addr 192.168.0.15
Swarm initialized: current node (9lcca74iz9dv44p2fkj7i84v9) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5ko6td8bcpqmaa8dggc5bmpct1yom3emxded18nt06vc33d2g7-evt0rrmyaw6v6pey13vnzv8yc 192.168.0.15:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Docker info komutu ile swarm bilgileri inceleyebiliriz.

Docker swarm join komutları manager ve worker tokenlarını alıp note ediyorum, node ekleyeceğimiz adımda kullanmak için.

```
[node4] (local) root@192.168.0.15 ~
$ docker swarm join --token worker
To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5ko6td8bcpqmaa8dggc5bmpctlyom3emxded18nt06vc33d2g7-evt0rrmyaw6v6peyl3vnzv8yc 192.1
68.0.15:2377

[node4] (local) root@192.168.0.15 ~
$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-5ko6td8bcpqmaa8dggc5bmpctlyom3emxded18nt06vc33d2g7-8fltbrmhjhjgynxks6cq dugxux 192.1
68.0.15:2377
```

192.168.0.16 docker hostuma gelip swarma dahil etmek için direkt olarak manager tokenimi kullanıyorum.

Aynı şekilde diğer 2 worker hostuma gelip worker node eklemek için worker tokenimi kullanıyorum ve eklediğini görebiliyorum.

manager

```
[node3] (local) root@192.168.0.16 ~
$ docker swarm join --token SWMTKN-1-5ko6td8bcpqmaa8dggc5bmpctlyom3emxded18nt06vc33d2g7-8fltbrmhjhjgynxks6cq dugxux 1
8.0.15:2377
This node joined a swarm as a manager.
[node3] (local) root@192.168.0.16 ~
```

Worker

```
[node2] (local) root@192.168.0.17 ~
$ docker swarm join --token SWMTKN-1-5ko6td8bcpqmaa8dggc5bmpctlyom3emxded18nt06vc33d2g7-evt0rrmyaw6v6peyl3vnzv8yc 192.168.0.15
:2377
This node joined a swarm as a worker.
[node2] (local) root@192.168.0.17 ~
```

Ardından manager node gelip

Docker node ls komutu çalıştırarak swarm ortamındaki nodelarımı görüntüleyebiliyorum.

```
[node4] (local) root@192.168.0.15 ~
$ docker node ls
ID          HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS   ENGINE VERSION
tro8d3gr7bb04vf1uo853xnia  node1  Ready  Active
hkdp95j9fgykzu6ev9k5pan  node2  Ready  Active
7q9g4qq4mxj3wdj4v0bsdb56n  node3  Ready  Active
9lca74iz9dv44p2fkj7i84v9 *  node4  Ready  Active
[node4] (local) root@192.168.0.15 ~
```

Docker info komutu ile kontrol ettiğimde swarmın aktif olduğunu teyit edebiliyorum.

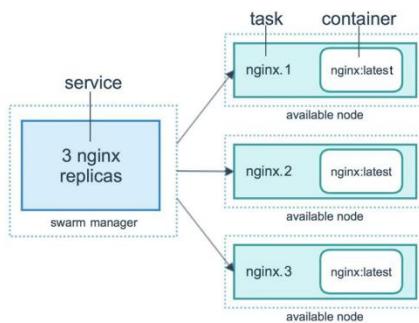
```
Swarm: active
NodeID: 86ov50ouxf08dxpcm8nrnmnp1
Is Manager: true
ClusterID: pumb1bh4igs1z2o3mity7o6ne
Managers: 2
Nodes: 4
```

Manager node üzerine gelip node update –role komutu çalıştırarak worker node'larımından birini manager olarak atıyorum ve görüldüğü gibi işlem başarılı, tekrardan worker olarak değiştiyorum.

```
ubuntu@manager-->lead:$ docker node ls
ID                      HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
5rqv7eits8ytu934hwaveaido  manager-2  Ready  Active        Reachable      20.10.21
86ov50ouxf08dpcm8nrrnnpi *  manager---lead  Ready  Active        Leader         20.10.21
83n2rmez97fztrukf5bpclukk  worker1   Ready  Active        Reachable      20.10.21
op1bt4sm9cc2f4wndv1vr085c  worker2   Ready  Active        Reachable      20.10.21
ubuntu@manager-->lead:$ docker node update --role manager 83n2rmez97fztrukf5bpclukk
83n2rmez97fztrukf5bpclukk
ubuntu@manager-->lead:$ docker node ls
ID                      HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
N
5rqv7eits8ytu934hwaveaido  manager-2  Ready  Active        Reachable      20.10.21
86ov50ouxf08dpcm8nrrnnpi *  manager---lead  Ready  Active        Leader         20.10.21
83n2rmez97fztrukf5bpclukk  worker1   Ready  Active        Reachable      20.10.21
op1bt4sm9cc2f4wndv1vr085c  worker2   Ready  Active        Reachable      20.10.21
ubuntu@manager-->lead:$ |
```

Worker node'lardan birini çıkarmak istersek, node üzerine gelip > docker swarm leave komutunu çalıştırmalıyız, ardından manager node üzerinde kontrol ettiğimizde down durumda olduğunu göreceğiz.

>Docker node rm "node id" ile node swarm üzerinden silebiliriz.



Docker swarm üzerinde, docker host da yönettiğimiz gibi container yönetimi bulunmamaktadır. Container oluşturma ve yürütme işlemleri task'larla oluşturulmaktadır.

İhtiyacımız olan uygulamalar için docker swarm üzerinde bir servis oluştururuz ve bu servisi swarm task'a dönüştür.

Biz replicasını belirleyebiliriz, arka planda hangi node üzerinde çalışracagını kendisi seçer. Eğer replica belirtmezsek default olarak kendisi 1 replica oluşturacaktır.

Görselde göründüğü gibi nginx için 3 replica belirtmiş ve 3 container oluşturmuş, bunu bi task haline getirim worker node'lar üzerinde senkron çalışmasını sağlayacaktır.

Node'lardan birine erişelemez durum olduğunda otomatik olarak eksik kalan container oluşturulur ve senkron şekilde devam eder.

## Docker Swarm Üzerinde Servis Oluşturma

İlk servisimizi oluşturalım, container ayağa kaldırımk için kullandığımız komutlar gibi kullanacağımız komutlar.

Docker service create –name myweb -p 8080:8080 nginx

```
ubuntu@manager-->lead:$ docker service create --name myweb -p 8080:8080 nginx
image nginx:latest could not be accessed on a registry to record
its digest. Each node will access nginx:latest independently,
possibly leading to different nodes running different
versions of the image.
```

Manager node üzerinde servisimi >docker service ls komutu ile kontrol ettiğimde ayakta olduğu görebiliyorum. Ve göründüğü gibi tek replica

```
ubuntu@manager-2:~$ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
kv5ww4birez9  myweb    replicated  0/1      nginx:latest *:8080->8080/tcp
```

Docker service ps myweb komutunu kullanarak hangi node üzerinde çalıştığını görebilirim.

Peki birden fazla replicakalı bir servis ayağa kaldırırsak –replica parametresini kullanmalıyız.

```
docker service create --replicas 10 nginx
```

```
[manager1] [local] root@192.168.0.28 ~
$ docker service create --replicas 10 nginx
ctuvzi7y5vl8rw9jvnws0xcm
overall progress: 0 out of 10 tasks
1/10: preparing
2/10: preparing
3/10: preparing
4/10: preparing
5/10: preparing
6/10: preparing
7/10: preparing
8/10: preparing
9/10: preparing
10/10: preparing
```

Docker service ls diyerek servisimi kontrol ediyorum ve 10 replica çalıştığını görüyorum. Ps komutu ile task'ların hangi nod'lar üzerinde bulundugunu görüntüleyebiliriyorum.

```
[manager2] [local] root@192.168.0.26 ~
$ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
b2anl07zg03g  my      replicated  0/1      web-2:latest
ctuvzi7y5vl8  vigorous_booth  replicated  0/10    nginx:latest
[manager2] [local] root@192.168.0.26 ~
$ docker service ps vigorous_booth
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
          PORTS
3va74qizlpsh  vigorous_booth.1  nginx:latest  manager3  Running     Preparing 11 seconds
ago
v44i9evovc58  \_ vigorous_booth.1  nginx:latest  manager2  Shutdown   Rejected 16 seconds a
go  "No such image: nginx:latest@...@"
ei139szjwgm2  vigorous_booth.2  nginx:latest  manager1  Ready      Preparing 3 seconds a
go
yx94awzxne4x  \_ vigorous_booth.2  nginx:latest  manager2  Shutdown   Rejected 4 seconds ag
o  "No such image: nginx:latest@...@"
gacjad9hw4w4  \_ vigorous_booth.2  nginx:latest  manager3  Shutdown   Rejected 11 seconds a
go  "No such image: nginx:latest@...@"
r7xfrcci4g8p  \_ vigorous_booth.2  nginx:latest  manager3  Shutdown   Rejected 41 seconds a
go  "No such image: nginx:latest@...@"
akvwnbzhtgtf  vigorous_booth.3  nginx:latest  manager1  Ready      Preparing 3 seconds a
no
```

Oluşturulan replicalar üzerinde daha fazla replica etmek istersem kullanacağım komut

```
$ docker service scale vigorous_booth=12
vigorous_booth scaled to 12
overall progress: 0 out of 12 tasks
1/12: preparing
2/12: preparing
3/12: preparing
4/12: preparing
5/12: preparing
6/12: preparing
7/12: preparing
8/12: preparing
9/12: preparing
10/12: preparing
11/12: preparing
12/12: preparing
```

>**docker service scale "servisname"=value**

Göründüğü üzere task'l 12 çıkarılmış ve node'lar üzerinde paylaştırmış durumda. Aynı şekilde azaltmada yapabiliyoruz.

Altarnatif komut ise update komutudur.

>**docker service update --replicas 8 vigorous\_booth** komutunu kullanarak replica sayısını güncelleylebilirim.

```
$ docker service update --replicas 8 vigorous_booth
vigorous_booth
overall progress: 8 out of 8 tasks
1/8: No such image: nginx:latest@sha256:e209ac2f37c70c1e0e9873a5f7231e91dcd83fd...
2/8:
3/8: No such image: nginx:latest@sha256:e209ac2f37c70c1e0e9873a5f7231e91dcd83fd...
4/8: running [=====>]
5/8: running [=====>]
6/8: running [=====>]
7/8: running [=====>]
8/8: running [=====>]
```

Örneğin bir servis oluştururken belirlediğimiz node üzerine çalışmasını ve çalışmamasını sağlamak için servis yazarken –constraint komut yardımıyla bu işlemi mümkün kılar.

>Docker service create –constraint “node.hostname==node1” nginx (node 1 üzerinde servisi oluştursun)

>Docker service create –constraint “node.hostname!=node1” nginx (node 1 üzerinde servisi oluşturmasın)

>Docker service create –constraint “node.role==manager” nginx (node 1 üzerinde servisi oluştursun)

Ör:

```
$ docker service create --constraint node.hostname==worker1 nginx
jso3y0kznqk6989qh6mq900km
overall progress: 0 out of 1 tasks
1/1: preparing [=====>]
```

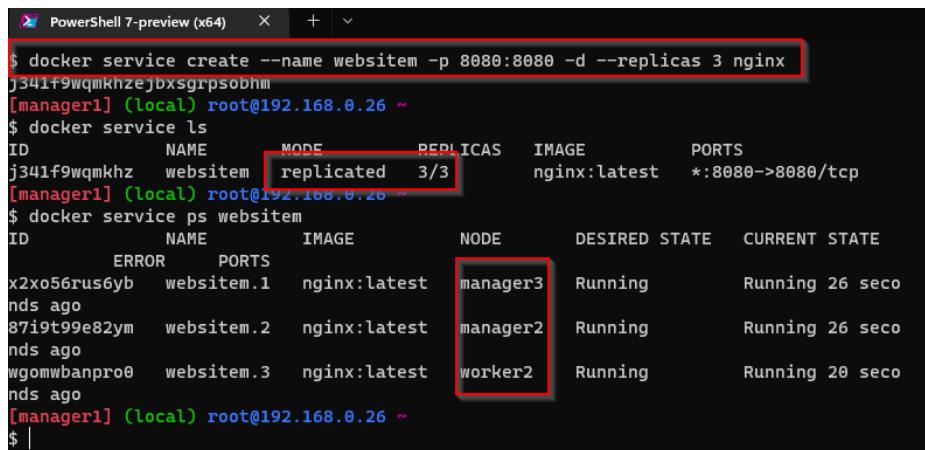
Docker service ps “service name” göründüğü üzere worker1 üzerinde servisimi çalışmakta.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
pdy8iv6bkkuq	mynginx.1	nginx:latest	worker1	Running	Running 12 seconds ago		

Docker service inspect “service name” komutum ile servis detaylarına bakabilirim

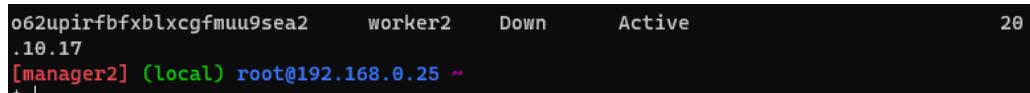
Docker service rm “service name” komutum ile servisimi silebilirim

Farklı bir örnek olarak websitem isminde 3 replikalı nginx servisi oluşturuyorum.  
Gördüğünüz üzere manager 3, manager 2, worker2 node’ları üzerine dağıtılmış.  
Worker node’mu shutdown ediyorum.



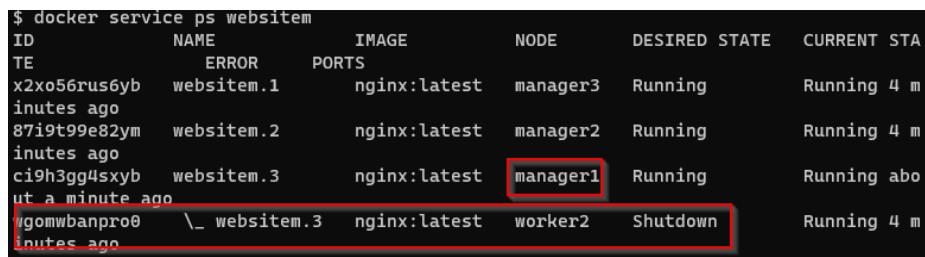
```
$ docker service create --name websitem -p 8080:8080 -d --replicas 3 nginx
j341f9wqmkhz [manager1] (local) root@192.168.0.26 ~
$ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
j341f9wqmkhz  websitem  replicated  3/3      nginx:latest *:8080->8080/tcp
[manager1] (local) root@192.168.0.26 ~
$ docker service ps websitem
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
ERROR      PORTS
x2xo56rus6yb  websitem.1  nginx:latest  manager3  Running       Running  26 seco
.10.17
87i9t99e82ym  websitem.2  nginx:latest  manager2  Running       Running  26 seco
.10.17
wgomwbanch0  websitem.3  nginx:latest  worker2   Running       Running  20 seco
.10.17
[manager1] (local) root@192.168.0.26 ~
$ |
```

Gördüğünüz üzere down durumda.



```
o62upirfbfxblxcmuu9sea2  worker2  Down  Active  20
.10.17
[manager2] (local) root@192.168.0.25 ~
+ |
```

Docker service ps komutu ile tekrar detaylara baktığında worker2 nodumun down durumda olduğunu, yerine ise auto olarak manager1 eklendiği görüyorum.



```
$ docker service ps websitem
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STA
TE          ERROR      PORTS
x2xo56rus6yb  websitem.1  nginx:latest  manager3  Running       Running  4 m
inutes ago
87i9t99e82ym  websitem.2  nginx:latest  manager2  Running       Running  4 m
inutes ago
ci9h3gg4sxyb  websitem.3  nginx:latest  manager1  Running       Running  about a minute ago
wgomwbanch0  \_ websitem.3  nginx:latest  worker2   Shutdown     Running  4 m
.10.17
[manager1] (local) root@192.168.0.26 ~
+ |
```

## Docker Swarm Monitor Tool

Docker swarm visualizer toolu, swarm ortamımızdaki servisleri monitor etmek için hazırlanmış, grafik arayüzlü bir monitor araç tooludur.

Bu araç swarm ortamında çalışmaktadır. İndirme ve yükleme işlemi için aşağıdaki linkte yönlendirmeler ile servis içerisinde olan bu toola erişebiliriz.

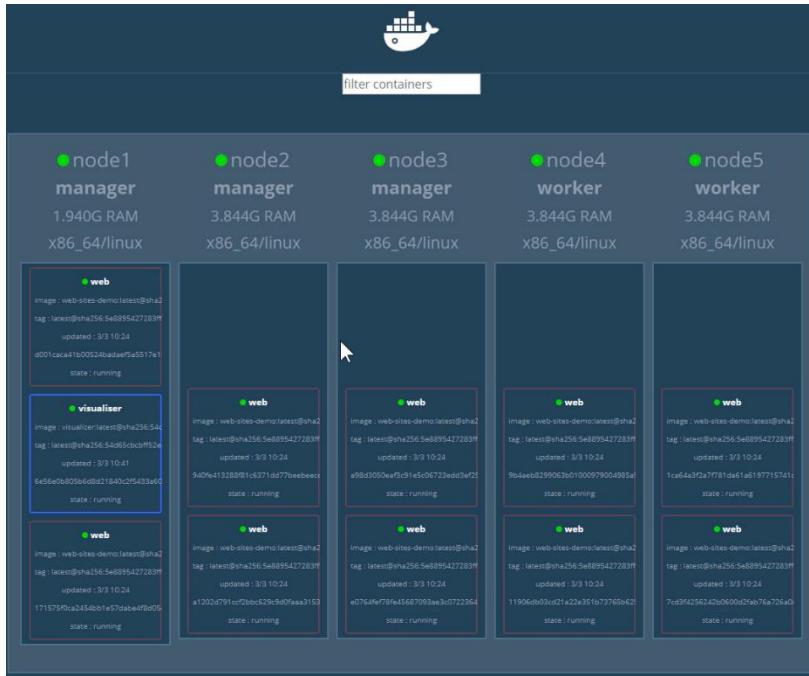
Görselde göründüğü üzere, üzerindeki servis dağılımları, node bilgileri gibi bilgiler yer almaktadır.

<https://github.com/dockersamples/docker-swarm-visualizer>

Writer: @HalilGÖKSEL

Ek olarak daha önce kısaca incelediğimiz portainer.io servisi üzerindede swarm monitoru ve servis yönetimi oldukça mümkün.

Üzerinde çalışarak test edebilirsiniz.



Bir diğer tool ise Node'larımızın kaynaklarını monitor ettğimiz sematext tooludur.

Sematext.com sayfası üzerinden kurulum ve ürün detayları ile ilgili bilgiler edinebiliriz. Sadece dokcer değil bir çok uygulamayı monitor edebileceğimiz bir araçtır.

Yine aynı şekilde swarm üzerinde bize verilen servis komutlarını çalıştırarak servisimizi ayağa kaldırabiliriz. Monitor için app.sematext.com portal üzerinden yapılmaktadır, onprime üzerinde çalışan bir araç olmadığını belirteyim.

## Docker Swarm Overlay Network

Overlay network, birden fazla docker host olan sistemde aynı network üzerinden birbirleriyle veri akışını sağlamak için kullanılan network modelidir.

Sadece docker üzerinde değil bir çok infra altyapısında bu modeli duyabilirsiniz.

>`docker network ls` komutu ile swarm ortamındaki node'larımın birbirleriyle iletişime geçtiği overlay network driver'ını görebiliyorum. Swarm üzerindeki tüm node'lar bu overlay network üzerinden konuşmaktadır.

```
[manager1] (local) root@192.168.0.26 ~
$ docker network ls
NETWORK ID      NAME          DRIVER      SCOPE
98a969de1150    bridge        bridge      local
2a52c5cc2726    docker_gwbridge  bridge      local
79c158505ebe    host          host       local
shr9x65p128h    ingress      overlay     swarm
eb57b5f7debe    none         null       local
[manager1] (local) root@192.168.0.26 ~
```

Writer: @HalilGÖKSEL

Swarm ortamımızda yeni bir overlay network oluşturup, yazdığımız servislerin bu network üzerinde koşmasını sağlayabilriz.

Network eklemek için

```
>docker network create --driver overlay "mynetwork"
```

>`Docker service create --network mynetwork ....` Diyerek eklediğimiz servise dahil edebiliriz.

overlay ağına bağlı bir web servisimizde, overlay networkinden ör: “10.10.10.0/24” tüm IP blogundan yayinallyadığımız servise erişmek mümkün. Servisler üzerindeki port farklılığı olduğu için aynı IP kullanıp erişmekte sorun olmayacağından emin olabiliriz. Bu özelliğin adı Routing mesh olarak geçmektedir. Swarm üzerindeki node’ların hepsi ingress / overlay kullanmakta, ve bunun üzerinden haberleşmektedir.

Biz 2. Bir overlay network oluşturup, bir servis üzerine konumlandırdığımızda aslında node üzerinde 2 overlay kullanmış olur. 1. Default olarak üzerinde bulunan , 2 sadece erişim verdigimiz node’lar üzerinde bulunan overlay network olur. Inspect ile incelediğimizde iki ayrı network CIDR karşılaşacaktır, Ingress overlay network’den özel oluşturduğumuz oveylay network’e yönlendirme sağlamaktadır.

Bu routing mesh özelliği, node IP blogunda üzerinde servis bulunmayan herhangibir IP üzerinden istek gönderdiğimde bu özellik sayesinde, oluşturduğum overlay ağına yönlendirme yapmaktadır.

## Docker Swarm Rolling Update

Rolling Update işlemi swarm üzerinde replica olarak çalışan bir servisini, kesinti olmadan yeni bir image file güncellenmesidir. Bu version güncellemesinde servis üzerinde bir kesinti yaşanmamaktadır.

Servis güncelleme işlemi yaparken update komutundan yararlanacağız, Sıfır kesinti ile versiyon güncellemesi için update komutunun yanına 2 farklı parametre kullanacağız. Duruma göre farklı parametrelerde kullanabiliriz.

`--update-parallelism` (aynı anda kaç task’ın güncelleneceği belirleyecek parametre)

`-update-delay` (güncelleme işlemi yapacak tasklar arasındaki süre belirten parametre)

```
Docker service update --image haligoksel/myweb2 (yeni version repository) --update-parallelism 2(task adeti) --update-delay 20s(süre) myweb (servis adı)
```

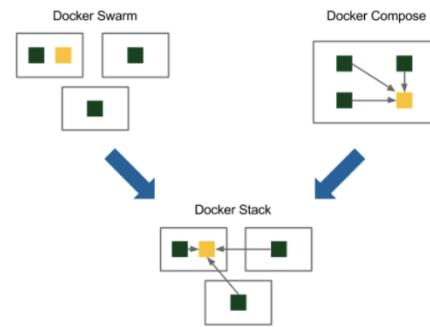
```
[manager2] (local) root@192.168.0.6 ~
$ docker service update --image halilgoksell/myweb:tahname --update-parallelism 2 --update-delay 10s webswarm
image halilgoksell/myweb:tahname could not be accessed on a registry to record
its digest. Each node will access halilgoksell/myweb:tahname independently,
possibly leading to different nodes running different
versions of the image.

webswarm
overall progress: 0 out of 10 tasks
1/10: assigned [=====] ]]
2/10: assigned [=====] ]
3/10:
4/10:
5/10:
6/10:
7/10:
8/10:
9/10:
10/10:
service update paused: update paused due to failure or early termination of task lxa9k6sdparspl1x91i68ulhe
[manager2] (local) root@192.168.0.6 ~
$ |
```

Göründüğü üzere mevcut servisim üzerinde kesinti yaşamadan yeni version geçişini tasklar üzerinde yapmış bulunmaktayız.

## Docker Stack

Docker swarm üzerinde oluşturduğumuz örneklerdeki gibi oluşturulan servis replica edilerek her bir node üzerinde task olarak çalışmaktadır. Ayağa kaldırıldığımız bu servis sadece uygulamadan ibaret.



Docker swarm stack ise txt dosyası üzerinde çoklu servis oluşturmaya ve yönetmeye yaramaktadır. Benzer bir örnek containerlar için docker compose yazmak gibi diyebiliriz.

Yaml dilinde yazılan stack yapısı, tek bir dosya ile swarm ortamında servislerimizi ayağa kaldırabiliriz.

Compose için yazılmış bir dosyayı, eklemeler yaparak stack için uygun hale getirebiliriz.

Default dosya adı **docker-stack.yml** olarak geçmektedir.

### Docker Stack Komutları

Stack ile ilgili tüm komutlara [Compose specification | Docker Documentation](#) adresinden erişebiliriz.

Deploy parametresi altındaki komutlar txt içerisinde kullanacağımız parametrelerdir.

Docker swarm stack dosyası içerisinde kullanılan komutları incelersek;

**Replicas** (oluşturulan servisin kopya sayısını belirler)

**Mode** (servisimizin replica mi yoksa global mi olduğunu belirtir.)

**Label** (etiket ataması yaparız)

**Placement** (Sınırlamalar için kullanırız (ör; bu servis manager üzerinde çalışmasın diyebiliriz))

**Resources** (kaynak kısıtlaması yapabiliriz)

En çok kullanılan docker swarm stack komutlarını incelersek;

**>Docker stack deploy –compose-file (kısalmet -c) docker-stack.yml “stack name”** (Docker swarm stack ayaga kaldırırmak için komutu kullanırız)

**>Docker stack ls** (swarm üzerindeki stack'leri listelemek için kullanırız.)

**>Docker stack ps** (stack üzerindeki taskları inceleyebiliriz.)

**>Docker stack service “stack name”** (stack üzerindeki servisleri listelemek için kullanırız)

Örnek uygulama

Github üzerinden bir örnek uygulama üzerinden gittim, swarm manager nodelerimden birinde docker-stack.yml dosyası oluşturdum ve yml içeriğini kopyaladım, çalıştığım uygulama linked yer verdiğim gibi

<https://github.com/dockersamples/example-voting-app/blob/master/docker-stack.yml> (oylama uygulaması)

deploy için;

**>docker stack deploy -c docker-stack.yml “service adı”** komutu calistirdım

```
[manager1] (local) root@192.168.0.26 ~
$ docker stack deploy -c docker-stack.yml app
Creating network app_frontend
Creating network app_default
Creating network app_backend
Creating service app_worker
Creating service app_visualizer
Creating service app_redis
Creating service app_db
Creating service app_vote
Creating service app_result
[manager1] (local) root@192.168.0.26 ~
```

Kontrol ettiğimde servislerim ayakta, docker stack ps komutum ilede node'lar üzerindeki taskları inceleyebiliyorum.

```
[manager1] (local) root@192.168.0.26 ~
$ docker stack ls
NAME      SERVICES      ORCHESTRATOR
app       6             Swarm
```

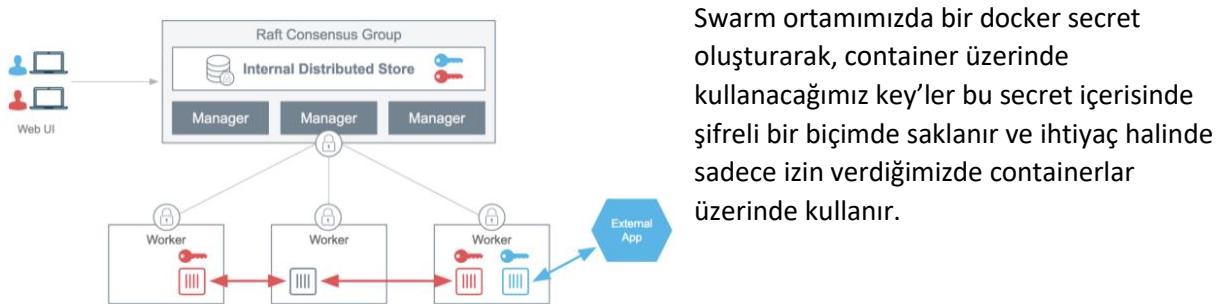
Aşına olunması için üzerinde bolca Pratik yapmak gerekecek, dökümanı daha fazla uzun tutmak istemediğimden bu kadar yeterli olduğunu düşünüyorum. Gerisi sizde :)

## Docker Secret

Şimdiye kadar yaml formatında yazılan compose file ve docker file içerisinde geçen, imajlar üzerinde belirlediğim şifreler, açık olarak yaml dosya içerisinde yazmakta buda ortamımız için güvenlik zafiyeti oluşturmaktadır.

Docker secret ile kontainerlerin ihtiyaç duygu , user – pass, TSL serfitika key, SSH key gibi değerleri yönetmek, bu değerleri şifreli biçimde saklamaktadır. Sadece docker swarm ortamında kullanılmaktadır.

İçerik karakter boyutu 500 kb'dır, docker secret komutları ile yönetimi bulunmaktadır.



Swarm ortamımızda bir docker secret oluşturarak, container üzerinde kullanacağımız key'ler bu secret içerisinde şifreli bir biçimde saklanır ve ihtiyaç halinde sadece izin verdığımızde containerlar üzerinde kullanır.

Linux üzerinde /run/secrets altında saklanmaktadır

Windows üzerinde C:\Programdata\docker\Secrets altında saklanmaktadır.

## Docker Secret Komutları

Docker host üzerinde docker secret komutunu kullandığımızda kullanacağımız komutlar listelenmektedir.

>Docker secret create "secret name" "secret içerisinde aktarılacak text dosya yolu" ile secret oluştur ve belirlediğimiz txt dosyası ile şifremizi secret üzerine şifreli yerleştirmiş oluruz.

Bir diğer yolu ise echo ile bir şifre belirtirip secret oluşturmak

```
>echo "şifre" | docker secret create "secret name" -
```

>docker secret ls ile secretleri listeleriz.

```
[manager1] (local) root@192.168.0.8 ~
$ vim dbsifre.txt
[manager1] (local) root@192.168.0.8 ~
$ cat dbsifre.txt
passwd
[manager1] (local) root@192.168.0.8 ~
$ docker secret create db-passwd dbsifre.txt
z9ovma15gem89df41h9bclt92
[manager1] (local) root@192.168.0.8 ~
$ docker secret ls
ID                      NAME      DRIVER    CREATED        UPDATED
z9ovma15gem89df41h9bclt92  db-passwd          8 seconds ago  8 seconds ago
[manager1] (local) root@192.168.0.8 ~
```

```
[manager1] (local) root@192.168.0.8 ~
$ echo "passwd" | docker secret create db-passwd2 -
nqe4uivj4c58vt6fgnpfm321n
[manager1] (local) root@192.168.0.8 ~
$ docker secret ls
ID          NAME      DRIVER    CREATED      UPDATED
z9ovma15gem89df41h9bclt92  db-passwd  About a minute ago  About a minute ago
nqe4uivj4c58vt6fgnpfm321n  db-passwd2  6 seconds ago   6 seconds ago
```

Oluşturma adımları bu kadar şimdi bunu nasıl kullanacağımıza bakalım.

Servis oluştururken secret kullanmak istersek

```
>docker service create --secret "secret adı" nginx
```

Oluşturulmuş bir servis üzerinde secret eklemek istersek,

```
>docker service update --secret-add "secret adı" "servis adı"
```

Çalışan servis üzerinden secret silmek için

```
>docker service update --secret-rm "secret name" "servis adı"
```

Güncellemek için,

```
>docker service update --secret-rm "secret adı" \ --secret-add "secret adı" servis adı
```

## Docker Secret Örnekleri

Passwd adında bir secret oluşturdum

```
$ docker secret ls
ID          NAME      DRIVER    CREATED      UPDATED
gtxdabalvb95jep3l5qacec9q  passwd    10 minutes ago  10 minutes ago
```

Ardından swarm üzerinde bir servis yaratıyorum ve secret kullanıyorum

```
$ docker service create --name myweb --secret passwd -p 8080:80 nginx
tlcbhl8e31ujecvu6r1j55hh
overall progress: 1 out of 1 tasks
1/1: running  [=====]
verify: Waiting 2 seconds to verify that tasks are stable...
```

Oluşturulan nginx container üzerinde bağlandığında secret'imin run/secrets dizinen ulaşlığını görüyorum.

```
[manager1] (local) root@192.168.0.7 ~
$ docker container ps
CONTAINER ID  IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
90b2c3000851  nginx:latest  "/docker-entrypoint..."  7 minutes ago  Up 7 minutes  80/tcp    myweb.1.oke800oaq83b4jtu6800g59t4
[manager1] (local) root@192.168.0.7 ~
$ docker container exec -it myweb.1.oke800oaq83b4jtu6800g59t4 /bin/bash
root@90b2c3000851:/# cat /run/secrets/passwd
admin
root@90b2c3000851:/# |
```

Farklı bir örnek,

Postgre database servislerimiz oluşturmak için ilgili komutları yazıyoruz, secretimizi kullanmak için servis içerisinde ekliyoruz ve environment üzerinde secret yolunu belirtiyoruz.

Containerlarımı kontrol ettiğimde secret oluşturulmuş ve DB servislerimde etkin olduğunu görmekteyim.

```
[manager2] (local) root@192.168.0.5 ~
$ echo "useradmin" | docker secret create db-user -
jhy3x6afacmry3mduxmkquier
[manager2] (local) root@192.168.0.5 ~
$ echo "dbadmin" | docker secret create db-admin -
rngyv42bjadtmtwbqfqzbgzy6
[manager2] (local) root@192.168.0.5 ~
$ docker secret ls
ID                      NAME          DRIVER      CREATED        UPDATED
pgrd9jvd6rytmmnbwyq5yim9r  database-password    11 minutes ago  11 minutes ago
irp7iambypnsr7pkvsulty46w  database-user     12 minutes ago  12 minutes ago
rngyv42bjadtmtwbqfqzbgzy6  db-admin       8 seconds ago   8 seconds ago
jhy3x6afacmry3mduxmkquier  db-user        27 seconds ago  27 seconds ago
gtxdobelvb95jep3l5qacec9q  passwd         52 minutes ago  52 minutes ago
[manager2] (local) root@192.168.0.5 ~
$ docker service create --name mydatabase \
> --secret db-user \
> --secret db-admin \
> -e POSTGRES_USER_FILE=/run/secrets/db-user \
> -e POSTGRES_PASSWORD_FILE=/run/secrets/db-admin \
> postgres:latest
```

Bu örneğimizde **docker-compose** içerisinde secret kullanımını göreceğiz. Bunun için docker hostum üzerinde secret olarak oluşturulacak 2 adet passwd içeren txt dosyası oluşturuyorum. Ardından docker compo yazmaya başlıyorum.

```
[manager2] (local) root@192.168.0.14 ~
$ vim db_admin.txt
[manager2] (local) root@192.168.0.14 ~
$ vim db_user.txt
[manager2] (local) root@192.168.0.14 ~
$ vim docker-compose.yml
```

Writer: @HalilGÖKSEL

Mariadb imajım içerisinde secret kullandığımı belirtip secret adını ve dosya yolunu veriyorum.

Environment içerisinde user pass bilgilerini run/secrets/.. üzerinden almasını sağlıyorum.

```

1 version: '3.7'
2 services:
3     db:
4         image: mariadb
5         volumes:
6             - db_data:/var/lib/mysql
7         secrets:
8             - db_user
9             - db_admin
10        environment:
11            MYSQL_USER_FILE: /run/secrets/db_user
12            MYSQL_PASSWORD_FILE: /run/secrets/db_admin
13            MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_admin
14            MYSQL_DATABASE: db
15
16 secrets:
17     db_user:
18     file: ./db_user.txt
19     db_admin:
20     file: ./db_admin.txt
21
22 volumes:
23     db_data:

```

Docker compose up komutum ile compose ayağa kaldırıyorum. Container üzerindeki secrets'ları kontrol ediyorum. DB bağlanmak için exec komutunu kullanabiliriz.

```

[manager2] (local) root@192.168.0.14 ~
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS          NAMES
9308bbc7fb9   mariadb    "docker-entrypoint.s..."   About a minute ago   Up About a minute   3306/tcp       root-db-1
[manager2] (local) root@192.168.0.14 ~
$ docker-compose ps
NAME           COMMAND          SERVICE          STATUS          PORTS
root-db-1      "docker-entrypoint.s..."  db            running        3306/tcp
[manager2] (local) root@192.168.0.14 ~
$ docker-compose exec db ls /run/secrets
db_admin  db_user

```

Docker swarm stack için bir örnek yapsaydık durum aynı olacaktı, dosya içerisinde kullandığımız secrets parametreleri stack içinde aynı kullanabiliriz. Daha sonra swarm ortamımızda servisimi çalıştırabiliriz.

**Dökümanım bu kadar faydası olduysa ne mutlu bana, sağlacakla (: @HalilGÖKSEL**