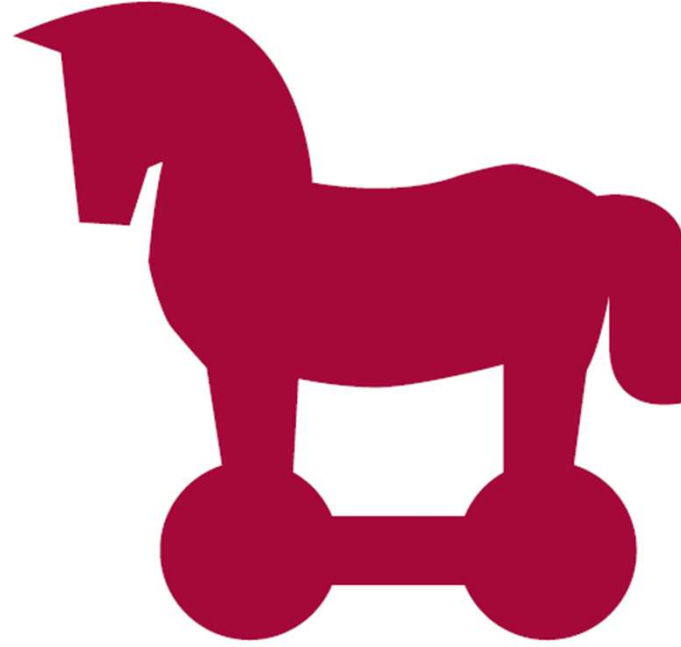


GANTEK ACADEMY

40+ YILLIK TECRÜBE İLE



GANTEK

Docker

Docker Kurulumu - Özet

- Docker'ın eski sürümleri varsa sistemden kaldırmamız gerekmektedir.
- Sistemin en son güncellemeleri almasına emin olmak için «**sudo apt update**» komutu verilmelidir.
- Docker'ın çalışması için gerekli ek paketler yüklenmeli ve kurulumları yapılmalıdır.
- Ubuntu 20.04 ve 22.04 için farklı kurulum işleri bulunmaktadır.
- Güncellemelerin alınması için Docker Kod Deposunun, APT (veya diğer paket yönetimi sistemleri) **sources.list** dosyasına eklenmesi gereklidir.
- Docker'dan gelecek paketleri açabilmek için Docker'ın açık şifre anahtarının (public key) da **/etc/apt** dizinine kaydedilmesi gereklidir.
- Docker için aşağıdaki paketler yüklenmelidir:
 - docker-ce**
 - docker-ce-cli**
 - containerd.io**

<https://docs.docker.com/engine/install/ubuntu/>

Kurulum (Ubuntu 20.04)

- İşletim sistemin en son güncellemeleri aldığından emin olalım
`sudo apt update`
- Gerekli paketleri kurmalıyız
`sudo apt install apt-transport-https ca-certificates curl
gnupg-agent software-properties-common -y`
- Güncellemeleri docker sitesinde alabilmemiz için, GPG anahtar kelimesini güncellemelerin alınacağı siteler arasına eklememiz gerekli
 - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -`
- GPG anahtarının eklendiğinden emin olmak için aşağıdaki komutu kullanın ve hata vermediğinden ve Docker için kurulu olduğunu görün
 - `sudo apt-key fingerprint 0EBFCD88`

<https://docs.docker.com/engine/release-notes/>

Kurulum (Ubuntu 20.04)

- Docker kod deposunu (repository) eklememiz gerekli

```
$ sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs)  
stable"
```
- Bütün hazırlıklar tamam ve artık Docker'ın kurulumuna başlayabiliriz.

```
$ sudo apt install docker-ce docker-ce-cli containerd.io -y
```
- Yukarıdaki komut belirli bir süre alacak ve Docker kurulumu gerçekleştirilecektir. Hangi sürümün yüklendiğini ve alt bileşenlerin sürümlerini görmek için «**docker version**» komutu verilebilir.
- Docker konteynerlerinin çalıştırılması için **sudo** yetkisi gerekir. Docker'ın **sudo** gerekmeden çalıştırılması için, şu komutlar verilmelidir:

```
$ sudo groupadd docker  
$ sudo usermod ${USER} -aG docker  
$ su - ${USER}  
$ id -nG
```

Kurulum (Ubuntu 22.04)

- İşletim sistemin en son güncellemeleri aldığından emin olalım
`$ sudo apt update`
- Gerekli paketleri kurmalıyız
`$ sudo apt install apt-transport-https ca-certificates curl software-properties-common -y`
- Güncellemeleri docker sitesinde alabilmemiz için, GPG anahtar kelimesini güncellemelerin alınacağı siteler arasına eklememiz gerekli
`$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg`
- Docker reposunu sistemimize tanıtalım
`$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`

<https://docs.docker.com/engine/release-notes/>

Kurulum (Ubuntu 22.04)

- Docker reposunun adresini girmiştik. Oradan paket listelerini güncelleyelim. Docker'ın en son sürümü de listemize girecek.
`$ sudo apt update`
- Ubuntu deposu yerine Docker reposundan kurulumu yapacağımızı belirteceğiz:
`$ apt-cache policy docker-ce`
- Docker'ı kuralım
`$ sudo apt install docker-ce`
- Yukarıdaki komut belirli bir süre alacak ve Docker kurulumu gerçekleştirilecektir. Hangi sürümün yüklendiğini ve alt bileşenlerin sürümlerini görmek için «**docker version**» komutu verilebilir.
- Docker konteynerlerinin çalıştırılması için kullanıcımıza **sudo** yetkisi gerekmektedir. Bu amaçla şu komutlar verilmelidir:
`$ sudo usermod -aG docker ${USER}`
`$ su - ${USER}`

Test

- Docker'ın kurulumunu sınamak için test konteynerini deneyebiliriz.

docker run hello-world

- Docker lokal olarak bulamadığı konteyneri Docker Hub adlı imaj/konteyner sitesinden indirecek ve çalıştıracaktır. Eğer çalışmazsa, Docker kurulumunda bir sorun var demektir. İşlemleri tekrarlamak gerekmektedir.

```
$ docker run hello-world
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

```
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.

```
To try something more ambitious, you can run an Ubuntu container with:
```

```
$ docker run -it ubuntu bash
```

```
Share images, automate workflows, and more with a free Docker ID:
```

```
https://hub.docker.com/
```

```
For more examples and ideas, visit:
```

```
https://docs.docker.com/get-started/
```

```
$
```

Bu komut sonrasında, eğer, «<https://registry-1.docker.io/v2/>: unauthorized» ve benzeri bir hata alırsanız, Docker hub'a bağlanamadınız demektir. Docker'ı yeniden kurmalısınız.

hello-world imajı repodan indirildi, /var/lib/docker/ dizini altındaki dizinlerden birinde açıldı, çalıştırıldı ve sonlandırıldı.

Alpine Linux

- Alpine Linux sadece 5 MB'lık bir yer tutan Linux dağıtımıdır. **musl**, **libc** ve **BusyBox** kütüphanelerini kullandığından diskte çok az yer kaplamaktadır.
- Docker sitesinden imaj halinde indirilerek konteyner olarak çalıştırılabilmektedir.
- Ancak, aşağıdaki komut verildiğinde, Alpine Linux konteynerinin çalıştığını ve sonlandığını görebiliriz (komut çalıştıktan sonra konteyner çalışmasına devam etmemekte ve durmaktadır).
- Bunun yerine **--interactive** opsiyonuyla Alpine Linux konteynerinin konsola bağlı kalarak, komutlarımıza cevap verecek şekilde interaktif olarak çalışmasını sağlamak istersek, verilecek komut budur

```
docker run --interactive alpine
```

https://hub.docker.com/_/alpine

<https://github.com/docker/labs/blob/master/beginner/chapters/alpine.md>

Alpine Linux

- İnteraktif modda, Alpine konteyneri sonlanmamakta ve kabuğa (Shell) komut verilebilmektedir.

```
$ docker run --interactive alpine
ps
PID  USER  TIME  COMMAND
   1  root   0:00  /bin/sh
  11  root   0:00  ps
ls
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

--interactive opsiyonu kısaca **-i** olarak girilebilir.

--interactive opsiyonunda komut istemcisi görülmez çünkü kabuk programı (/Bash/Sh/zsh vb) çalışmamaktadır.

ls komutu verildiğinde, **ls** programı çalıştırılarak içinde bulunan dizinin içeriği gösterilir.

Alpine Linux Konteyneri (**--tty** opsiyonu)

- Görüldüğü gibi, kabuğa benzemeyen bir ara yüz bulunmaktadır. Daha uygun bir ara yüz için şu komut kullanılabilir:

```
docker run --interactive --tty alpine
```

```
$ docker run --interactive --tty alpine
/ # ls
bin      etc      lib      mnt      proc     run      srv      tmp      var
dev      home     media    opt      root     ssh      sys      usr
/ # ps
PID      USER     TIME     COMMAND
   1   root         0:00   /bin/sh
   8   root         0:00   ps
/ # exit
```

--tty opsiyonuyla kabuk programı çalışır ve komut istemi görülür. Ya da kısaca **-it** komutu verilebilir.

- Görüldüğü gibi bir terminal penceresi gibi görünen bir ara yüzle Alpine çalışır durumdadır.
 - exit** komutu bu ara yüzden çıkılmasını sağlayacaktır.

TTY nedir?

- TTY, «**T**ele **T**ype writer» kelimelerinden türetilmiş bir terimdir ve elektrikli klavye/yazıcı anlamına gelmektedir.
- İlk tele yazıcılar, 1800'lü yılların ikinci yarısında borsadaki verilerin uzak mesafelere telefon kabloları üzerinden gönderilmesi için kullanılan ASCII temelli klavye sistemleridir.
- Tele yazıcıların ilk kullanıldığı dönemlerde bilgisayarlar yoktu ve insanlar tarafından veri giriş yapılan klavyeler olarak işlev görmektedir.
- TTY kavramı, özetle, elektriksel ortamda veri aktarımı yapılan, gelenleri yazıya dökabilen, klavye üzerinden sağlanan kullanıcı arayüzü sistemi demektir.



TTY nedir?

- 1900'lü yılların ortalarında ilk bilgisayar sistemleri ortaya çıktığında ve çok kullanıcı olmaya başladığında, kullanıcıların sistemlerle iletişim kurabilmesi için, zaten kullanımda olan tele klavye/yazıcılar kullanıldı.
- Ancak farklı tele klavye/yazıcı sistemleri bulunmaktaydı. Bunların standartlaşması gerekiyordu.
- 1970'li yılların sonunda video terminallerin çıkışında, baud-hızının değiştirilmesi, akış kontrolü, kontrol kodları (ekranı sil, satır başına git, vb) gibi daha önce olmayan özelliklerin bulunduğu VT-100 standardı geliştirildi.
- Günümüzde, fiziksel tele klavyeler bulunmamaktadır. Linux'te, artık, klavye ve ekranla çalışan emüle edilmiş TTY'ler kullanılmaktadır.

DEC VT100

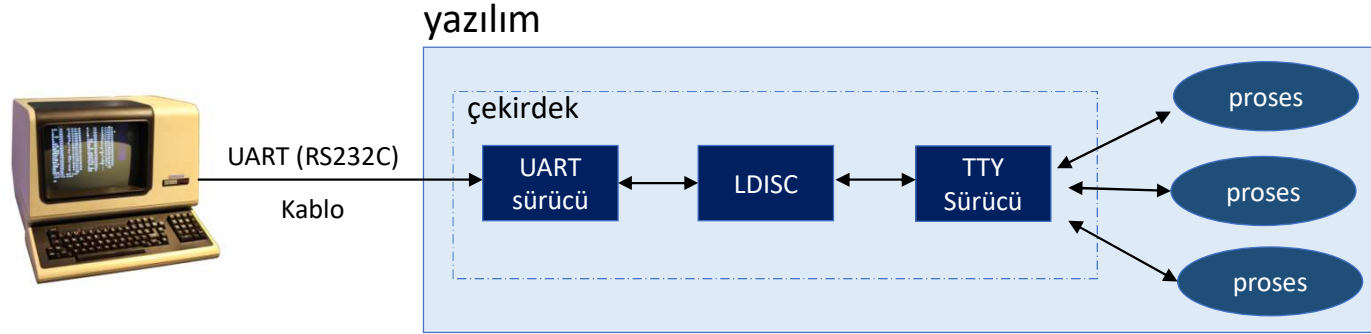


TTY nedir?

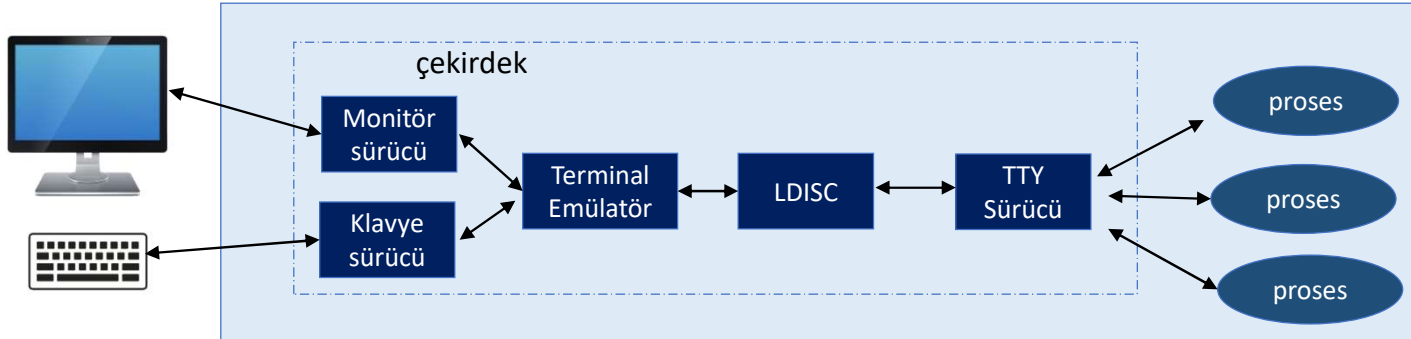
- TTY alt sistemi, Linux'un ve genelde UNIX'un çalışması ve etkin kullanımı için önemlidir.
- Konsol, Linux makinenin ana kontrol arayüzüdür. Çekirdekten gelen mesajlar buraya yazılır, mesajlar burada gösterilir.
- Altı adet varsayılan sistem konsolu (TTY) bulunmaktadır.
- Ctrl-Alt-F1-F6 ile erişilir. Ctrl-Alt-F2 Ubuntu'da grafik ekrana girer.

TTY nedir?

- Fiziksel TTY'ler söz konusu olduğunda yapı şöyleydi:



- Masaüstü sistemlerde şu hale geldi.



TTY nedir?

- TTY, özetle terminal anlamına gelir ve metin bazlı komutların ve mesajların aktarıldığı giriş/çıkış ortamıdır.
- Linux'te 6 adet konsol (TTY terminal) bulunur. Bunlar arasında Ctrl-Alt-F1...Ctrl-Alt-F6 tuşları ile geçiş yapılabilir. Ctrl-Alt-F2, grafik masaüstüne döner.

```
Ubuntu 20.04.4 LTS ubuntu tty5
ubuntu login: adminpc
Password:
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.15.0-41-generic x86_64)

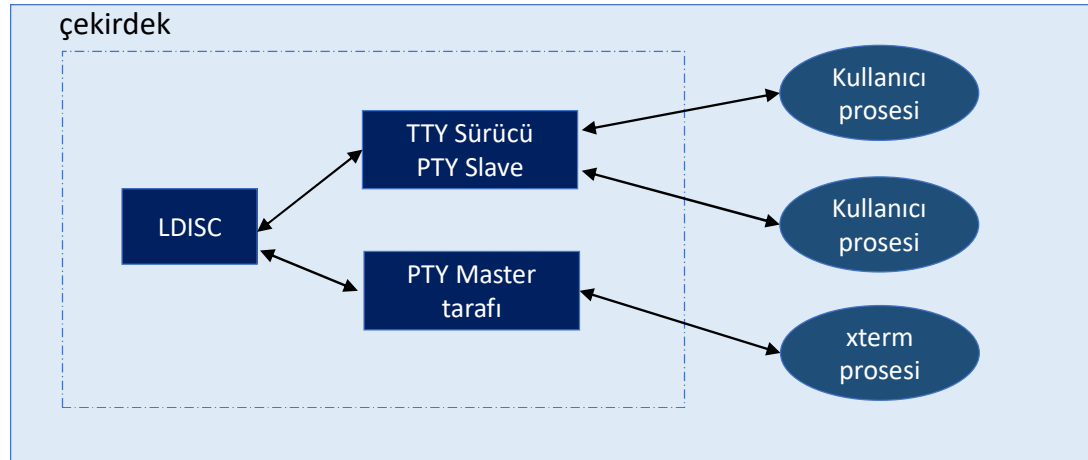
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Fri Jul 22 16:39:54 +03 2022 on tty5
Chicken Little only has to be right once.
adminpc@ubuntu:~$ ps l
F  UID      PID     PPID  PRI  NI       VSZ   RSS   WCHAN  STAT TTY      TIME COMMAND
4  1000    2009     1949   20   0  166756  6500  do_sys Ssl+  tty2     0:00 /usr/lib/gdm3/gdm-x-session
4  1000    2012     2009   20   0  297032  68868  ep_pol S1+   tty2     1:03 /usr/lib/xorg/Xorg vt2 -dis
0  1000    2067     2009   20   0  190952  13784  do_sys S1+   tty2     0:00 /usr/libexec/gnome-session-
0  1000    2564     2556   20   0  13620   5192  do_sel Ss+   pts/0    0:00 bash
4  1000    8275     8188   20   0  13744   5176  do_sel S+    tty3     0:00 -bash
4  1000    9021     8969   20   0  13744   5228  do_wai S     tty5     0:00 -bash
0  1000    9030     9021   20   0  14156   3212  -      R+    tty5     0:00 ps l
adminpc@ubuntu:~$ _
```

TTY nedir?

- Masa üstü sistemlerde tek konsol bulunmasına gerek duyulmamaktadır. İstenirse birden fazla arayüz de oluşturulabilmektedir.
- Terminal programları üzerinden, çok sayıda terminal emüle edilebilmektedir. Bunlara «pseudo-TTY» yani **pty** adı verilmektedir. **pts** ise «Pseudo TTY Slave» anlamına gelir.



<https://www.golinuxcloud.com/difference-between-pty-vs-tty-vs-pts-linux/>

«pts» ve «tty»

- «ps 1» komutu, çalışan prosesleri ve bağlı oldukları TTY'leri gösteren bir komuttur.

```
$ ps 1
```

F	UID	PID	PPID	PRI	NI	VSZ	RSS	WCHAN	STAT	TTY	TIME	COMMAND
4	1000	2009	1949	20	0	166756	6500	do_sys	Ss1+	tty2	0:00	/usr/lib/gdm3/gdm-x-
4	1000	2012	2009	20	0	297236	69584	ep_pol	S1+	tty2	1:11	/usr/lib/xorg/Xorg v
0	1000	2067	2009	20	0	190952	13784	do_sys	S1+	tty2	0:00	/usr/libexec/gnome-s
0	1000	2564	2556	20	0	13620	5192	do_wai	Ss	pts/0	0:00	bash
4	1000	9021	8969	20	0	13744	5228	do_sel	S+	tty5	0:00	-bash
4	1000	11490	11198	20	0	13744	5116	do_sel	S+	tty3	0:00	-bash
0	1000	11951	2564	20	0	14156	1076	-	R+	pts/0	0:00	ps 1

«pts», grafik masaüstünden «xterm» programı üzerinden bu proseslerin çalıştığını söylemektedir.

«tty3» Ctrl-Alt-F3'le 3 nolu konsolu açılarak sisteme doğrudan giriş yapıldığı ve bu prosesin çalıştırıldığını göstermektedir.

Ubuntu'da «tty2» grafik ekranın bulunduğu TTY konsoludur.

«**tty**» komutu

- «**tty**» komutu, hangi TTY'ye bağlı olduğumuzu görmek için kullanılır.
- Eğer istenirse, yönlendirme «>» işaretiyle, o TTY'ye bir string yazılabilir.

```
$ tty  
/dev/pts/1  
$ echo merhaba > /dev/pts/1  
merhaba
```

Konteyner prosesi ne zaman sonlanır?

- Konteynerdeki ana prosesin numarası (yani PID'si), namespace özelliği sayesinde konteyner için PID 1 olarak gösterilir.
- Linux işletim sistemi türevlerinde PID'si 1 olan proses (**init** veya **systemd**) sona erdiğinde makine sonlanır.

```
$ docker run -it alpine  
/ # ps  
PID    USER      TIME  COMMAND  
   1   root         0:00  /bin/sh  
   8   root         0:00  ps
```

- 1 numaralı proses sonlanınca, Docker, konak işletim sistemi üzerinde konteyner kontrol grubundaki bütün prosesleri sonlandırır.
- Diğer deyişle konteynerin sağlığı tek bir prosese bağlıdır. Bu nedenle bir konteynerin içinde birden fazla önemli prosesin bulunması, konteynerin sağlığını belirleme açısından sıkıntılıdır.

«namespace»

- Her konteyner oluşturulduğunda bir PID «namespace» oluşturulur ve konteyner prosesleri bu alanda çalıştırılır. Bu şekilde prosesler PID'si 1'den başlar.

```
$ docker run -dit alpine
c3cffd8749290df40574fc7c5b0e565830c5c146960964a9f33146e161db969c
$ sudo lsns -t pid
[sudo] password for adminpc:
      NS TYPE NPROCS   PID USER COMMAND
4026531836 pid       322     1 root /lib/systemd/systemd auto noprompt splash --system --deserialize 18
4026533189 pid         1 57710 root /bin/sh
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS          NAMES
c3cffd874929   alpine    "/bin/sh"               5 minutes ago   Up 5 minutes           jolly_rubin
$ docker stop c3cffd874929
c3cffd874929
$ sudo lsns -t pid
      NS TYPE NPROCS   PID USER COMMAND
4026531836 pid       328     1 root /lib/systemd/systemd auto noprompt splash --system --deserialize 18
```

Kalıcı Konteynerler

- Önceki örneklerde görüldüğü gibi, konteynerler aksi belirtilmediği zaman çalışmasını tamamlayıp çıkış yapmaktadır.
- Eğer konteynerlerin kalıcı bir şekilde Docker ortamında çalıştırılması isteniyorsa, konteynerlerin daha farklı opsiyonlarla çalıştırılmaları gereklidir.
- Bu durum sunucu uygulamalarında, sunucunun devamlı çalışması istendiğinde gereklidir. Bunun için kalıcılık opsiyonu yani «**-dt**» opsiyonları kullanılmalıdır (Burada **d**: detached, **t**: tty anlamına gelmektedir)
- Ayrıca, eğer konteynerimiz bir hata veya başka bir nedenle durursa yeniden başlatılabilmesi için gerekli olan «**--restart**» opsiyonunun farklı özelliklerle birlikte kullanılması iyi olacaktır.

Kalıcı Konteynerler

- Verilecek komut şöyle olabilir (**--name** opsiyonuyla konteynere isim vermek ve ID yerine bu adı kullanarak işlem yapabilmek için **sunucu2** ismi veriyoruz ve çıktı olarak)

```
$ docker run -dt --restart always --name sunucu2 alpine
```

```
$ docker run -dt --restart always --name sunucu2 alpine
a4053d9f493116773746f3e00cd58417d95758ae4f1ff896a75e13ae00736b70
```

- «**docker ps -a**» komutuyla, Docker'ın bu konteyneri arka planda çalıştırdığı görülebilir

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a4053d9f4931	alpine	"/bin/sh"	About a minute ago	Up About a minute		sunucu2
34ea13761ce0	debian	"bash"	5 hours ago	Exited (0) 3 hours ago		vibrant_lovelace
c6355a4d3b79	debian	"bash"	5 hours ago	Exited (0) 5 hours ago		quirky_saha
d679ba107bc6	alpine	"/bin/sh"	25 hours ago	Exited (0) 24 hours ago		cool_margulis
cd83ae0cc41b	hello-world	"/hello"	26 hours ago	Exited (0) 26 hours ago		hopeful_torvalds

<https://docs.docker.com/config/containers/start-containers-automatically/>


Kalıcı Konteynerin Durdurulması

- Kalıcı olarak başlatılan «**alpine**» konteynerinin arka planda çalıştığı «**docker ps -a**» komutuyla görülebilir.
- Eğer çalışır («**up**») durumda olan konteyner durdurulmak istenirse, «**docker ps -a**» komutuyla elde edilen listede «**CONTAINER ID**» kolonundaki ID'si «**docker stop konteyner-ID/isim**» yazılarak ilgili konteyner durdurulabilir.

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
a4053d9f4931   alpine    "/bin/sh" 2 minutes ago    Up 2 minutes           sunucu2
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
a4053d9f4931   alpine    "/bin/sh" 2 minutes ago    Up 2 minutes           sunucu2
34ea13761ce0   debian    "bash"    5 hours ago     Exited (0) 3 hours ago         vibrant_lovelace
c6355a4d3b79   debian    "bash"    5 hours ago     Exited (0) 5 hours ago         quirky_saha
d679ba107bc6   alpine    "/bin/sh" 25 hours ago     Exited (0) 24 hours ago        cool_margulis
cd83ae0cc41b   hello-world "/hello"  26 hours ago     Exited (0) 26 hours ago        hopeful_torvalds
$ docker stop sunucu2
```

«IMAGE ID»

- «**docker images**» komutu, sisteme yüklenmiş olan imajları göstermektedir. Her imaj farklı bir «**IMAGE ID**» değeriyle ifade edilir. Docker'da verilen komutların bir kısmında «**IMAGE ID**» değerinin verilmesi gerekmektedir.



\$ docker images					
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE	
nginxtry	latest	ala52aec2d42	2 days ago	142MB	
alppyth	1.0.1	77b792ed5fd8	2 days ago	68MB	
ubuntu-guncel	latest	df531d64f1e5	3 days ago	171MB	
debian	stable	fd388d9cf0ba	2 weeks ago	124MB	
ubuntu	latest	d2e4e1f51132	4 weeks ago	77.8MB	
alpine	3.15	0ac33e5f5afa	7 weeks ago	5.57MB	
hello-world	latest	feb5d9fea6a5	8 months ago	13.3kB	

- «**IMAGE ID**» olarak görülen 12 heksadesimal (16'lık) dijitten oluşan değer, o imajı ifade eden daha büyük 256 bitlik bir sayının son 12 dijitidir.
- Aşağıdaki komutla imajla ilgili ayrıntılı bilgi ve 256 bitlik «**hash**» (SHA256) sonucu burada görülebilir.

docker inspect <IMAGE-ID-DEGERI>

Konteyner Bilgileri

- Herhangi bir konteynerle ilgili bilgi (örneğin versiyon bilgisi, içinde hangi yazılımların çalıştığı, vb) alınması için konteyner ID'si verilerek «**docker inspect**» komutu kullanılmalıdır. «**inspect**», imajla ilgili bütün bilgileri JSON formatında listeleyecektir.

```
$ docker inspect 0ac33e5f5afa
[
  {
    "Id": "sha256:0ac33e5f5afa79e084075e8698a22d574816eea8d7b7d480586835657c3e1c8b",
    "RepoTags": [
      "alpine:3.15"
    ],
    "RepoDigests": [
      "alpine@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2022-04-05T00:19:59.912662499Z",
    "Container": "b714116bd3f3418e7b61a6d70dd7244382f0844e47a8d1d66dbf61cb1cb02b2b",
    "ContainerConfig": {
      "Hostname": "b714116bd3f3",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
```

Docker imaj mimarisi

imajlar ve konteynerler

- Docker imajları çalışabilir bir konteyneri oluşturmak için gerekli her türlü bilgiyi içeren şablon veri yapılarıdır.
- Konteynerler oluşturulurken, bu veri yapısındaki komutlar işlenir.
- **Birden fazla katman içerirler ve katmanlı bir yapıya sahiptirler.**
- Son katman dışında, bütün katmanlar sadece okunabilir ve değiştirilmez (neden? Çünkü bir katman bir önceki katmana bağlıdır)
- Konteyner üzerinde yaptığımız her türlü değişiklik, kurduğumuz programlar, vb son katmanda değişikliğe sebep olur.
- İmaj ve konteyner arasındaki en önemli fark, konteyner üzerinde çalışma sırasındaki değişikliklerin yazıldığı R/W katmandır.

docker info

- «**docker info**» komutu, sistemde yüklü Docker yazılımı ve konfigürasyon dosyalarıyla ilgili bilgiler verir.

```
$ docker info
Client: Docker Engine - Community
Version: 24.0.6
Context: default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version: v0.11.2
    Path: /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version: v2.21.0
    Path: /usr/libexec/docker/cli-plugins/docker-compose

Server:
Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
Images: 1
Server Version: 24.0.6
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
Logging Driver: json-file
Cgroup Driver: systemd
Cgroup Version: 2
```

Genel bilgiler:
Docker sürümü
Hangi eklentilerin yüklü olduğu gibi bilgiler bulunur.

Sistemde yüklü imajların sayısı, kaç konteynerin çalışmış veya çalışıyor olduğunu gösterir.

Hangi «Control Group» sürücüsünün kullanıldığını gösterir.

docker info

- «**overlayFS**», Docker konteynerlerin içindeki dosya sistemi için gerekli bir sürücüdür.

```
$ docker info
Client: Docker Engine - Community
Version: 24.0.6
Context: default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version: v0.11.2
    Path: /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version: v2.21.0
    Path: /usr/libexec/docker/cli-plugins/docker-compose

Server:
Containers: 1
  Running: 0
  Paused: 0
  Stopped: 1
Images: 1
Server Version: 24.0.6
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
Logging Driver: json-file
Cgroup Driver: systemd
Cgroup Version: 2
```

OverlayFS, bir imajdan gelen dosyaların, çalıştırıldığında konteynerdeki dosyalarla birleştirilerek bir dosya sistemi oluşturulmasını sağlayan bir Linux özelliğidir.

docker info

- «**overlayFS**», Docker konteynerlerin içindeki dosya sistemi için gerekli bir sürücüdür.

```
Plugins:
Volume: local
Network: bridge host ipvlan macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runc.v2 runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 61f9fd88f79f081d64d6fa3bbl1a0dc71ec870523
runc version: v1.1.9-0-gccaecfc
init version: de40ad0
Security Options:
  apparmor
  seccomp
    Profile: builtin
  cgroupns
Kernel Version: 6.2.0-32-generic
Operating System: Ubuntu 22.04.3 LTS
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 3.781GiB
Name: srv1
ID: 063a2595-88ec-49d0-ac69-2367fa055e1a
Docker Root Dir: /var/lib/docker
Debug Mode: false
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false
```

Bir konteynerin ana dosya sisteminde nerelere erişebileceği apparmor ile belirlenebilir ve güvenlik seviyesi artırılabilir.

Konak sistemle ilgili bilgiler

İmajlar

- **hub.docker.com** adresine bakıldığında çok sayıda işletim sistemi imajı göreceksiniz. Debian, Ubuntu, Alpine, Centos, Amazon Linux, Fedora, Oracle Linux, Rocky Linux, Arch Linux, vb
- **SORU:** Buradan indireceğimiz bütün işletim sistemi konteynerleri Ubuntu işletim sistemi üzerinde çalışıyorsa, bu kadar işletim sistemi imajına ne gerek var????
- Bu işletim sistemi imajlarının hepsi kendi imajlarınızı oluşturmak için baz imajlar olarak kullanılır.
- Uygulamanız eğer Debian'da geliştirildiyse, Debian işletim sisteminin yapısını isteyebilir. Bu imajdaki konfigürasyon dosyaları, dosya sistemi ve kütüphaneler Debian tarzındadır ama konak işletim sistemi üzerinde çalışır.
- Yazılımlarınızı yükleyeceğiniz imajlar için kesinlikle baz imajla başlamanız ve ilk katman olarak belirlemeniz gerekecektir.

İmajlardaki katmanların görüntülenmesi

- İmajlarla ilgili ayrıntılı bilgi almak için **dive** programı kullanılabilir.
- Programı Firefox ile indirin ve kurun
`https://github.com/wagoodman/dive#installation`
- **dive** programı, Docker hub'daki herhangi imajı (örneğin alpine) indirerek katmanlarını size gösterebilir.
`dive alpine`
`dive ubuntu`
- **dive** programı Docker kullanmadan, komut satırından çalıştırılarak imajdaki sorunların bulunması için bir araç olarak kullanılabilir.

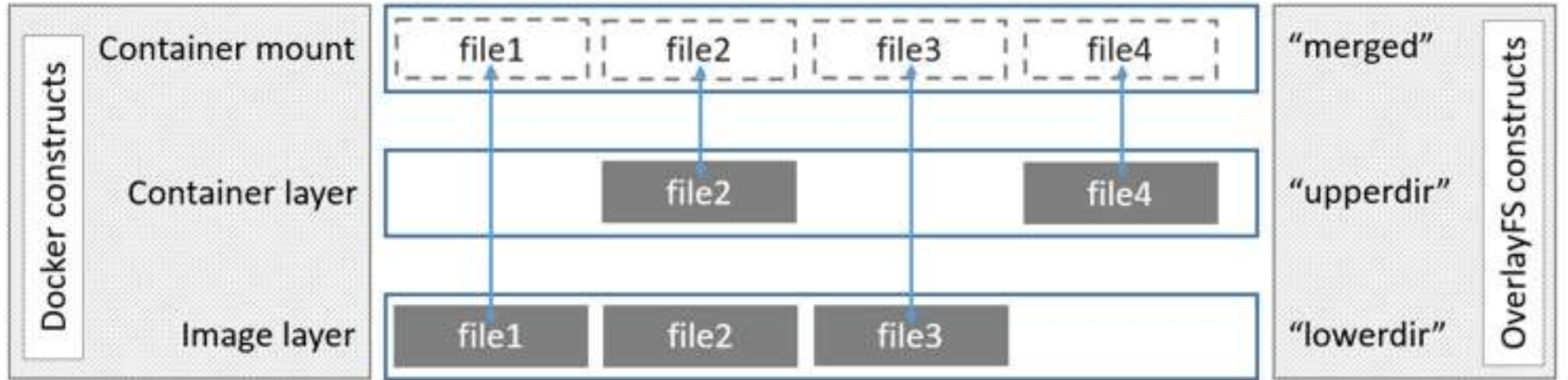
**GANTEK
ACADEMY**
40+ YILLIK TECRÜBE İLE

Layers			Current Layer Contents			Filetree	
Cmp	Size	Command	Permission	UID:GID	Size	Filetree	
			-rwxrwxrwx	0:0	0 B	bin → usr/bin	
	75 MB	FROM 56e8812f2c55944	drwxr-xr-x	0:0	0 B	boot	
4.3 kB		set -eux; groupadd -r postgres --gid=999; useradd -r -g postgres --uid=999 --home	drwxr-xr-x	0:0	0 B	dev	
10 MB		set -ex; apt-get update; apt-get install -y --no-install-recommends gnupg	drwxr-xr-x	0:0	172 kB	etc	
4.2 MB		set -eux; savedAptMark="\$(apt-mark showmanual)"; apt-get update; apt-get inst	-rw-----	0:0	0 B	.pwd.lock	
25 MB		set -eux; if [-f /etc/dpkg/dpkg.cfg.d/docker]; then grep -q '/usr/share/loc	-rw-r--r--	0:0	3.0 kB	adduser.conf	
4.0 MB		set -eux; apt-get update; apt-get install -y --no-install-recommends libn	drwxr-xr-x	0:0	100 B	alternatives	
0 B		mkdir /docker-entrypoint-initdb.d	-rw-r--r--	0:0	100 B	README	
4.0 kB		set -ex; key='B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8'; export GUNUPGHOME="\$	-rwxrwxrwx	0:0	0 B	awk → /usr/bin/mawk	
300 MB		set -ex; export PYTHONDONTWRITEBYTECODE=1; dpkgArch="\$(dpkg --print-archi	-rwxrwxrwx	0:0	0 B	awk.1.gz → /usr/share/man/man1/mawk.1.gz	
60 kB		set -eux; dpkg-divert --add --rename --divert "/usr/share/postgresql/postgresql.conf.	-rwxrwxrwx	0:0	0 B	builtins.7.gz → /usr/share/man/man7/bash-builtins.7.gz	
Layer Details			-rwxrwxrwx	0:0	0 B	nawk → /usr/bin/mawk	
			-rwxrwxrwx	0:0	0 B	nawk.1.gz → /usr/share/man/man1/mawk.1.gz	
			-rwxrwxrwx	0:0	0 B	pager → /bin/more	
			-rwxrwxrwx	0:0	0 B	pager.1.gz → /usr/share/man/man1/more.1.gz	
Tags: (unavailable)			-rwxrwxrwx	0:0	0 B	rmt → /usr/sbin/rmt-tar	
Id: 56e8812f2c55944a36ed248f081b8e12d4f638557a5fc027060bdf5846a8597a			-rwxrwxrwx	0:0	0 B	rmt.8.gz → /usr/share/man/man8/rmt-tar.8.gz	
Digest: sha256:d310e774110ab038b30c6a5f7b7f7dd527dbe527854496bd30194b9ee6ea496e			-rwxrwxrwx	0:0	0 B	which → /usr/bin/which.debianutils	
Command:			-rwxrwxrwx	0:0	0 B	which.1.gz → /usr/share/man/man1/which.debianutils.1.g	
#(nop) ADD file:a1398394375faab8dd9e1e8d584eea96c750fb57ae4ffd2b14624f1cf263561b in /			-rwxrwxrwx	0:0	0 B	which.dei.gz → /usr/share/man/de/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.es1.gz → /usr/share/man/es/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.fr1.gz → /usr/share/man/fr/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.it1.gz → /usr/share/man/it/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.ja1.gz → /usr/share/man/ja/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.pl1.gz → /usr/share/man/pl/man1/which.debianutil	
			-rwxrwxrwx	0:0	0 B	which.sl1.gz → /usr/share/man/sl/man1/which.debianutil	
Image Details			drwxr-xr-x	0:0	79 kB	apt	
Image name: postgres			drwxr-xr-x	0:0	3.3 kB	apt.conf.d	
Total Image size: 418 MB			-rw-r--r--	0:0	399 B	01autoremove	
Potential wasted space: 9.8 MB			-rw-r--r--	0:0	182 B	70debconf	
Image efficiency score: 98 %			-rw-r--r--	0:0	754 B	docker-autoremove-suggests	
			-rw-r--r--	0:0	1.2 kB	docker-clean	
Count			-rw-r--r--	0:0	481 B	docker-gzip-indexes	
6	4.7 MB	/var/cache/debconf/templates.dat	-rw-r--r--	0:0	269 B	docker-no-languages	
4	3.1 MB	/var/cache/debconf/templates.dat-old	drwxr-xr-x	0:0	0 B	auth.conf.d	
6	609 kB	/var/lib/dpkg/status-old	drwxr-xr-x	0:0	0 B	keyrings	
6	609 kB	/var/lib/dpkg/status	drwxr-xr-x	0:0	0 B	preferences.d	
			drwxr-xr-x	0:0	443 B	sources.list.d	
</							

<https://github.com/wagoodman/dive#installation>

OverlayFS

- İmajdan gelen dosyalar ve dizinlerle (**lowerdir**), konteyner seviyesinde oluşan dosyalar (**upperdir**) birleştirilerek, **merged** katmanı oluşturulmaktadır.
- Altta ve üstte aynı isimli bir dosya olduğunda üstteki gösterilir.



Neden OverlayFS kullanılıyor?

- Disk ve bellek alanından tasarruf için OverlayFS kullanılır.
- Bir sistem üzerinde çalışan aynı imajdan türetilmiş konteynerler varsa, bunlar disk alanında 0B kullanır.
- Sanal boyut olarak büyük boyutlarda olabilir.
- 10 konteyner aynı imajdan türetildiğinde, eğer konteynerin **rw** katmanlarda tuttuğu bir şey yoksa (örneğin disk alanına konteyner bir şey yazmıyorsa), kullanılan disk alanı sadece tek bir konteynerin kapladığı alan olacaktır.
- Docker, OverlayFS sayesinde disk alanını optimum şekilde kullanmakta ve konteynerleri etkin şekilde çalıştırmaktadır.

Docker Komutları

run, attach, exec, update

Temel Komutlar

- Docker versiyonunun öğrenilmesi
`docker --version`
- Docker daemon'unun çalışma durumunu görmek:
`sudo systemctl status docker`
- Docker içindeki konteynerlerin listesini görmek:
`docker ls -a`
- Bir konteynerin tek seferlik çalıştırılması
`docker run konteynerismi`
- Bir konteynerin interaktif şekilde çalışmasının sağlanması
`docker run --interactive --tty konteyneradı`
- Bir konteyner çalıştırdıktan sonra listeden çıkarılabilir (Yani «`docker ls -a`» o konteyneri göstermeyecektir)
`docker run --rm konteyner`

`--interactive -i` olarak gösterilebilir. `--tty` ise kısaca `-t` olabilir.
Birleştirilerek, `-ti` veya `-it` olarak da kullanılabilir.

Komutlar

- Yeni bir Docker imajı Docker Hub'tan yüklemek için
`docker pull imajismi`
- Docker Hub içinde bir imajı aramak:
`docker search imajismi`
- Docker Hub'dan indirilmiş imajların listesini görmek
`docker images`
- Docker hub'dan çekilmiş imajlardan birini silmek (imajın çalışan veya durdurulmuş bir konteynerle ilgisi varsa, ilk önce o konteynerin «**docker stop**» ardından «**docker rm**» komutuyla silinmesi gereklidir)
`docker rmi <IMAGE ID>`

Komutlar

- Sistemdeki aktif imajların listesini görmek
`docker ps`
- Sistemdeki aktif veya pasif imajların tamamını (all) listelemek
`docker ps -a`
- Sistemde çalışan konteynerlerin bellekte kapladığı alanları görmek
`docker ps -s`
- Sistemde çalışan konteynerlerin sadece ID'lerini (quiet) alt alta listeler
`docker ps -q`
- En son çalışan (last) konteyneri gösterir
`docker ps -l`
- Sistemde son çalışan 10 tane konteyneri listeler
`docker ps -n 10`

docker run

- Belirlenen bir imajı belleğe alarak opsiyonlarla belirlenen şekilde çalıştırır. Çok kullanılan opsiyonlar şunlardır:
- «**--interactive**» veya «**-i**» : konteynere bağlı olunmasa da STDIN'i açık bırakır.
- «**--detach**» veya «**-d**»: konteyneri arka planda çalıştırır. Konteynerin ID'sini ekrana yazar ve komut istemi görünür.
- «**--tty**» veya «**-t**» : konteynere pseudo-TTY (konsol) tanımlar.
- «**--name**»: konteynere bir isim atar.
- «**--hostname**» : konteynere host ismi atar.
- «**--restart**» : konteyner prosesi kapandığında veya çıkış yaptığında, yapılacak işlemleri belirtir.
- «**--rm**» konteyner sonlandığında konteyneri otomatik olarak siler.

«-it» ve «-dit» opsiyonları

- «**docker run**» komutuyla kullanılan «-it», «-dt», «-dit» opsiyonları arasında önemli farklar var. Bunların öğrenilmesi, Docker ile yapılacak işleri hızlandıracaktır:
 - «-i» (interactive): konteyner çalıştırılınca **STDIN**'i açık bırakır ve konteynerde komut çalıştırılabilir. Bash gibi programlar **STDIN**'i kullanır ama bazıları kullanmaz. Bu opsiyonla başlatılan konteynerlerde arka planda çalışan program eğer **STDIN** kullanıyorsa ekranda yazılar akar ve loglar görülebilir. Yoksa herhangi bir çıktı izlenmez.
 - «-d» (detached): Docker'ın çalıştırılan konteyner prosesinin çıkış yapmasını beklemesiyle alakalıdır. Beklemeden **STDOUT**'u bırakır. **STDIN**'i açık tutar.
 - Bash programı **STDIN** kapandığında sonlanır. «-i» opsiyonu olmazsa hemen çıkış yapar. «-d» ve «-i»nin birbirine ortogonal opsiyonlar oldukları düşünülebilir. Ancak birbiriyle çelişmezler.
 - «-t» opsiyonu konteyner için **pseudo-tty** ve dolayısıyla bir kabuk başlatır. «-it» ve «-i» arasındaki fark, ilkinin kabuk komut istemini (prompt) göstermesidir. Diğeri, komut istemini («**root@342878732 %**» gibi) göstermez ama komutları kabul ederek çalıştırır.

«-it» ve «-dit» opsiyonları

- «**docker run**» komutuyla kullanılan «-it», «-dit», «-dit» opsiyonları arasında çeşitli farklar bulunur.
- «-it» (interactive-TTY): konteyner çalıştırılınca BASH komut istemini (prompt) görüntüler ve kullanıcının komut girmesine izin verir. Bu süreç içinde kullanıcı Bash veya diğer kabuk programları üzerinden konteynerde komut çalıştırabilir ama «**exit**» komutunu verdiğinde kabuk kapanacak ve konteyner de sonlanacaktır. Konteyneri tekrar çalıştırmak için «**docker start konteynerID**» komutunu vermek gereklidir.
- «-dit» (detached-interactive-TTY): . Bu opsiyonla Docker, Bash'in **STDIN**'ini tekrar bağlanmak için açık bırakır. «**docker run**» hemen çıkış yapar fakat daha sonra «**docker attach**» komutuyla «**docker run -it**» komutunda olduğu gibi konteynerdeki Bash komut istemine tekrar bağlanılabilir.
- «-it» opsiyonuyla konteynerin TTY'sine bağlandığınızda, konteyneri kapatarak çıkmak için «**Ctrl-C**» kullanılır. Kapatmadan çıkmak için «**Ctrl-P**» ve «**Ctrl-Q**» kullanılır.

«-a» standart akımlara bağlantı

- «**docker run**» komutu «-it» opsiyonuyla çalıştırıldığında , **STDIN**, **STDOUT** ve **STDERR** akımlarına bağlı kalınır.
- Eğer sadece tek bir akıma bağlanmak isteniyorsa, «-a» komutu kullanılmalıdır. Örneğin sadece **STDERR** akımına bağlı kalmak ve hata mesajlarını görmek isterseniz şu komutu verebilirsiniz:

```
$ docker run -it -a stderr ubuntu
```

- Eğer oluşturduğunuz bir konteyner (**benimkont**) içine bir dosyayı, konteyner içinde çalışan bir programa (örneğin, **yapilandir.sh** programına) **PIPE** yapmak isterseniz ve bu şekilde konteynerde bazı yapılandırma işleri yapmak isterseniz, şöyle bir komut kullanabilirsiniz:

```
$ cat dosya | docker run -i -a stdin benimkont  
yapilandir.sh
```

Duran konteyneri yeniden başlatma

- **--restart** opsiyonundan sonra verilecek opsiyonlar şunlardır:
 - **no**: konteyner durduğunda otomatik olarak yeniden çalıştırmaz
 - **on-failure**: eğer konteyner bir hata sonucu durur ve hata kodu olarak 0 olmayan bir çıkış değeri üretirse
 - **always**: konteyner durduğunda her zaman yeniden başlat
 - **unless-stopped**: **always** komutuna benzer ve konteyner durduğunda yeniden çalıştırılmaz hatta Docker daemon'u yeniden çalışsa da başlatılmaz.
- Aşağıdaki komutla (**docker update**, konteynerin parametrelerini değiştirmek için kullanılabilir), Docker'da çalışan konteynerlerin tamamı kendiliğinden durmadıkları sürece yeniden çalıştırılacaktır.

```
docker update --restart unless-stopped $(docker ps -q)
```

docker run - kaynak sınırlamaları

- «**docker update**» ile kullanılabilecek ve konteynerin kullandığı kaynakları sınırlandırmak için kullanılacak opsiyonlar şunlardır:
 - «**--cpus**» : kaç tane CPU kullanabileceğini belirtir.
 - «**--memory**»: bellek limiti
 - «**--kernel-memory**»: çekirdek bellek limiti (çekirdek bellek swap yapılamayan proses belleğidir. Docker v20.10'da çeşitli sorunlar sebebiyle bu opsiyon kaldırılmıştır)
 - «**--memory-swap**»: toplam bellek ve swap miktarı. «**-1**» değerinin verilmesi limitsiz swap anlamına gelir.
 - «**--cpu-shares**» : Docker'da bir CPU'nun göreceli değeri 1024 olarak kabul edilir. Bu değeri referans alarak en fazla ne kadar CPU kullanacağı belirlenebilir.
 - «**--blkio-weight**»: Block cihazlarda G/Ç ağırlığı. 10 ile 1000 arasında olabilir. 0 bu sınırlandırmayı iptal eder.

<https://docs.docker.com/engine/reference/commandline/update/>

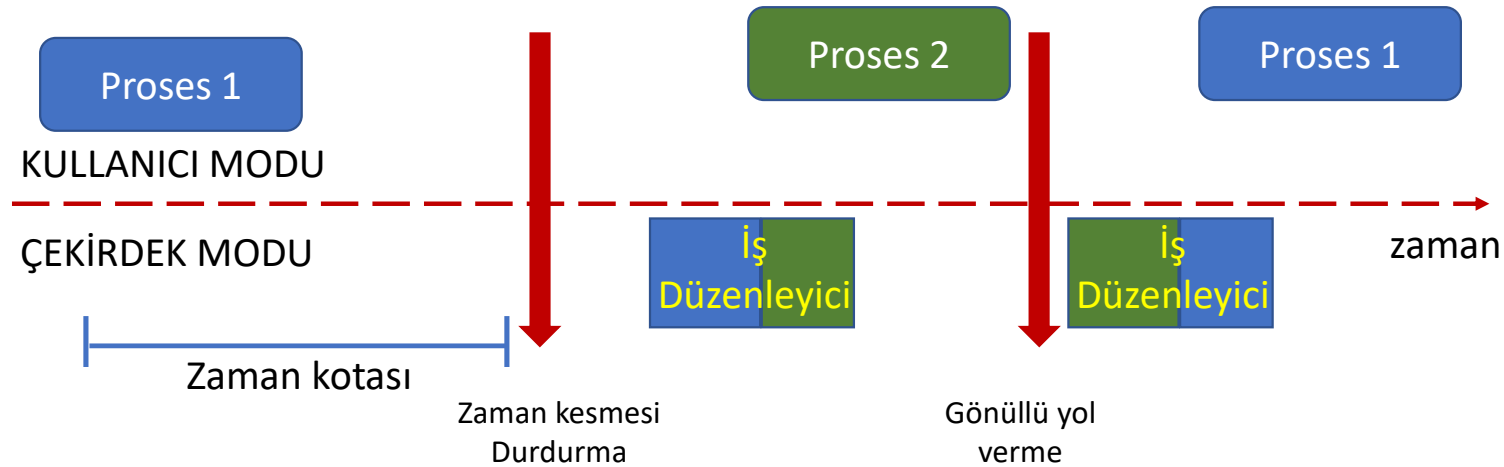
İş zamanlayıcı (scheduler)

- Sınırlı sayıda CPU ve bilgisayar kaynağı bulunan sistemlerde, çekirdek, kaynakları birbiriyle yarışan prosesler arasında paylaştırmaktadır.
- İş düzenleyici, çekirdeğin bir parçasıdır ve prosesler arasında CPU (veya CPU'ların) zamanını paylaştıran modüldür.
- Çekirdek, bir prosesi kısa bir zaman aralığı (time slice) süresince veya bir G/Ç işlemini beklemesi gerekene kadar çalıştırır ve bütün proseslerin paralel çalıştığı illüzyonunu yaratır.
- «**preemption**» işletim sistemlerinde önemli bir kavramdır ve bir prosesin durdurulması anlamına gelir. «**preemption**» genelde zaman belirli zaman aralıklarında çalışan zamanlayıcı kesmeleri sayesinde gerçekleştirilir.

Proses İş Zamanlayıcı

- Durdurma (**preemption**)

- Çok prosesli sistemlerde, iş zamanlayıcı (scheduler) sistemlerinde genellikle kullanılan yöntemdir.
- Her bir prosesin zaman kotası bulunur. Linux'te **preemption** 250 Hz'lik (4 ms'de bir) bir zamanlayıcı kesmesi tarafından gerçekleştirilir.
- Bazı prosesler, bunlara gerçek zamanlı (RT) denir ki, durdurulmazlar.



CFS

- Ingo Molnar tarafından tanıtılan CFS (Completely Fair Scheduler) algoritması Linux çekirdeğinin v2.6.23 sürümünde kullanıldı.
- CFS, İşletim Sistemi terminolojisinde *weighted fair queueing* (WFQ) adı verilen bir alitmadan türetilmiştir.
- CFS, CPU zamanını **epoch** adı verilen dilimlere ayırır.
- CFS’de çekirdek, her prosesin çalışma süresini, proseslerin önceliklerine göre önceden belirleyerek, her **epoch** için belirli bir çalışma süresini sağlamak için ağırlık atamaktadır. Kısacası, proseslerin gelecekte ne kadar çalışacakları önceden planlanmaktadır.
- Her **epoch** içinde, bu ağırlıklara göre bütün prosesler önceden planlanan sürelerde çalıştırılmakta ve bu şekilde prosesler arasında adalet sağlanmaktadır.
- Ağırlıkların hesaplanmasında bazı istisnai durumlara göre de uyarlama yapılmakta ve bu şekilde sistemde çok fazla proses bulunduğunda sistemin çalışması sekteye uğramamaktadır.

<https://www.youtube.com/watch?v=5WtnnzpwEuA>

CFS

- Basit şekilde, CFS'in çalışma şekli şöyledir: «**Vaktinin çoğunu uyuma modunda geçiren prosesler, öncelikleri artırılarak CPU'da çalıştırılırlar**». Bu şekilde bütün proseslere çalışma için daha adil (fair) bir ortam sunulur.
- **Soru:** Linux'un iş zamanlayıcı kodu iyi çalışıyor mu?
- **Cevap:** İlk 10 yılda CFS'in genel olarak (%99) iyi çalıştığı bilinmektedir. 2016'da bazı özel durumlarda bazı işlemci çekirdeklerinin boşa kalabildiği ve CPU zamanının uygun şekilde kullanılamadığı fark edilmiştir. 2016'da CFS'e yama yapılmış ve performansı iyileştirilmiştir:
 - <https://blog.acolyer.org/2016/04/26/the-linux-scheduler-a-decade-of-wasted-cores/>
- **Soru:** Linux'te sadece CFS mi kullanılıyor? Başka algoritmalar yok mu?
- **Cevap:** 2009'da Con Kolivas «**Brain F*ck Scheduler**» isimli iş zamanlayıcıyı önerdi. BFS, CFS'e göre daha basit bir tasarımdı ve CFS'e göre daha başarılı olduğu yerler vardı. Ancak Kolivas, BFS'in Linux çekirdeğine eklenerek, çekirdek bünyesinde geliştirilmesini istemedi. Şu anda yamalarla kullanılabilir.

<http://ck.kolivas.org/patches/bfs/bfs-faq.txt>

docker run - kaynak sınırlamaları

- «**docker run**» ile kullanılabilecek diğer opsiyonlar şunlardır:
 - «**--cpu-period**» : CPU CFS (Completely Fair Sched) periyodunu sınırlandırır.
 - «**--cpu-quota**»: CPU CFS kotası koyar
 - «**--cpu-rt-period**»: mikrosaniye olarak CPU'daki gerçek zamanlı periyodu sınırlandırır.
 - «**--cpu-rt-runtime**»: mikrosaniye olarak CPU'daki gerçek zamanlı çalışma periyodunu sınırlandırır.
 - «**--cpuset-cpus**» : Konteynerin hangi CPU'larda çalışacağını belirtir. Örneğin, «0-3», «0,1» gibi.
 - «**--cpuset-mems**» : Konteynerin hangi MEM'lerde çalışacağını belirtir. Örneğin, «0-3», «0,1» gibi.
 - «**--pids-limit**» : pids limitini ayarlar (-1 limitsiz demektir)
 - «**--memory-reservation**» : «**soft memory**» sınırlandırır.

docker run

- «**--cpu-period=<değer>**» opsiyonu CFS zamanlayıcı periyodunu ayarlamak için kullanılır. Varsayılan olarak 100000 mikrosaniyedir. Kullanıcıların çoğu bu parametreyi değiştirmez ve «**--cpus**» opsiyonunu tercih eder.
- «**--cpu-quota=<değer>**» : konteyner'e CFS kotası koyar. Varsayılan değeri 100.000 mikrosaniyedir. Genellikle kullanılmaz ve «**--cpus**» opsiyonu tercih edilir.
- «**--cpuset-cpus**»: sistemde birden fazla CPU veya CPU çekirdeği varsa, konteynerin çalıştırılacağı CPU veya çekirdek numarası belirtilebilir. Örneğin, «**--cpuset-cpus 2-3**», konteynerin çalışması için 3. ve 4. çekirdeklerin kullanılması gerektiğini belirtir.
- «**--cpu-shares**»: normal şartlarda yeterince CPU kapasitesi varsa, konteynerler kullanabildikleri kadar CPU zaman kaynağını kullanırlar. Bu opsiyon, 1024 değerinde olan bir CPU'nun ne kadarının kullanılabileceğini belirtir. İleri seviyeli bir opsiyon olduğundan yaygın şekilde kullanılmamaktadır.

docker run - örnekler

- Sadece 1 CPU'nun bulunduğu bir ortamda, konteynerin saniyede bir CPU'nun %50'ni kullanması isteniyorsa:
`$ docker run -it --cpus=".5" ubuntu bash`
- CPU'nun %50'sini kullanmak için diğer alternatif şudur (**docker update** komutuyla da kullanılabilir):
`$ docker run -it --cpu-period=100000 --cpu-quota=50000 ubuntu bash`
- Eğer konak makinedeki Linux çekirdeğinin gerçek zamanlı zamanlayıcı (real-time scheduler) özelliği etkin hale getirilmişse, «**--cpu-rt-runtime**» opsiyonu kullanılabilir.
- Konteynerin en fazla 300 Mbayt bellek kullanabilmesi için
`$ docker run -it --memory 300MB ubuntu bash`

docker run --device

- Docker'ın «**volume**» özelliği, konaktaki bir dizinin konteyner tarafından kullanılmasını sağlar.
- Ancak, «**volume**» ile **/dev/*** dizini altındaki bir diskin veya cihazın kullanılması durumunda bu özellik çalışmaz ve erişim hatası verir.
- **--device** opsiyonuyla herhangi bir disk alanı konteynere bağlanabilir ve konteynerin bu diski kullanabilmesi sağlanabilir.
- Bu özellik, veritabanı yönetim sistemi gibi ayrı bir disk alanında (LUN, LVM, vb bölütleri) DB dosyalarının tutulması istendiğinde faydalıdır. Bu alandaki dosyalara yazılan veriler kaybolmaz ve daha sonra başlatılan konteynerler tarafından da görülebilirler.
- **--device** ile sabit diske erişim güvenlik riski yaratabilir ve bu durumdan kaçınılmalıdır. **Aslında yapılması gereken, bir diske mount edilmiş bir dizine --device opsiyonunu kullanmadan --volume opsiyonuyla erişmektir.**

docker attach komutu

- «**-dit**» opsiyonuyla çalıştırılan bir konteynerin konsoluna bağlanmak için **attach** opsiyonu kullanılabilir (**attach -i** ve **-t** opsiyonlarıyla ilgilidir):

docker attach <konteynerID>

- Konteyner içindeki programların **STDOUT**'a yazdıkları bu şekilde görülebilir. **nginx**, bağlantı gerçekleştiğinde bunu **STDOUT**'a yazmaktadır. (**Ctrl-C** ile çıkabilirsiniz veya **Ctrl-P** ve **Ctrl-Q** ile)

```
$ docker attach e0a08575bbd8
172.17.0.1 - - [26/May/2022:11:35:28 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:35 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:40 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:41 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:42 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:54 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:35:55 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:36:20 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:100.0) Gecko/20100101 Firefox/100.0" "-"
2022/05/26 11:36:23 [error] 32#32: *10 open() "/usr/share/nginx/html/favicon.ico" failed (2: No
such file or directory), client: 172.17.0.1, server: localhost, request: "GET /favicon.ico
HTTP/1.1", host: "127.0.0.1:49154", referer: "http://127.0.0.1:49154/"
172.17.0.1 - - [26/May/2022:11:36:23 +0000] "GET /favicon.ico HTTP/1.1" 404 153
"http://127.0.0.1:49154/" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:100.0) Gecko/20100101
Firefox/100.0" "-"
172.17.0.1 - - [26/May/2022:11:36:26 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:100.0) Gecko/20100101 Firefox/100.0" "-"
```


`docker attach` komutu

- «**attach**» komutuyla konsoluna bağlanan konteynerin **stdin/stdout/stderr**'den çıkmak için kullanılan «**Ctrl-C**» tuşuyla birlikte, konteyner de durmaktadır.
- «**Ctrl-C**» ile sonlandırılan konteyner «**docker start ID**» komutuyla tekrar çalıştırılabilir.

```
^C2022/05/26 11:37:29 [notice] 1#1: signal 2 (SIGINT) received, exiting
2022/05/26 11:37:29 [notice] 31#31: exiting
2022/05/26 11:37:29 [notice] 31#31: exit
2022/05/26 11:37:29 [notice] 32#32: exiting
2022/05/26 11:37:29 [notice] 32#32: exit
2022/05/26 11:37:29 [notice] 1#1: signal 17 (SIGCHLD) received from 31
2022/05/26 11:37:29 [notice] 1#1: worker process 31 exited with code 0
2022/05/26 11:37:29 [notice] 1#1: signal 29 (SIGIO) received
2022/05/26 11:37:29 [notice] 1#1: signal 17 (SIGCHLD) received from 32
2022/05/26 11:37:29 [notice] 1#1: worker process 32 exited with code 0
2022/05/26 11:37:29 [notice] 1#1: exit
```

docker attach «ctrl-P» «ctrl-Q»

- Bildiğiniz gibi «**attach**» komutuyla konsoluna bağlanılan konteynerin **STDIN/STDOUT/STDERR** çıkmak için kullanılan «**Ctl-C**» komutuyla konteyner de durmaktadır.
- Bunu engellemenin diğer bir yolu, «**docker attach**» ile konteynerin konsoluna bağlandıktan sonra «**Ctrl-P**» ve ardından «**Ctrl-Q**» tuşlarına basılmasıdır. Bu şekilde «**read escape sequence**» mesajı vererek, konteynerle bağlantıyı koparacak ama konteyner çalışmaya devam edecektir.
- Tuşlar, «**docker run -it**» komutunda da geçerlidir.

```
$ docker run -dit --name ubu02 ubuntu
585a92bb74d22b6f22a8fb85a9b0766448dbc1975e6ba3221a60e7e4c3147233
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
585a92bb74d2   ubuntu   "bash"    40 seconds ago   Up 38 seconds           ubu02
6d013a34f1db   alpine   "/bin/sh" 11 hours ago   Up 11 hours             alp05
$ docker attach 585a92bb74d2
root@585a92bb74d2:/# ls
bin    dev    home  lib32  libx32  mnt    proc   run    srv    tmp    var
boot  etc    lib   lib64  media   opt    root   sbin   sys    usr
root@585a92bb74d2:/# read escape sequence
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
585a92bb74d2   ubuntu   "bash"    3 hours ago   Up 3 hours           ubu02
6d013a34f1db   alpine   "/bin/sh" 13 hours ago   Up 13 hours           alp05
```



docker attach --detach-keys

- «**--detach-keys**» opsiyonuyla yapılandırma yapılarak «**Ctrl-C**» tuşunun, konteyner kapanmadan sadece konsol bağlantısını kopartmak için kullanılacağı belirtilebilir.
- Aşağıdaki komutla, «**Ctrl-C**» sadece konsol bağlantısını kopartacak ve konteyner çalışmaya devam edecektir. İstenirse farklı tuş kombinasyonları da kullanılabilir.

docker attach --detach-keys="ctrl-C" ID

```
$ docker run -dit --name ubu02 --network deneag ubuntu
585a92bb74d22b6f22a8fb85a9b0766448dbc1975e6ba3221a60e7e4c3147233
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
585a92bb74d2   ubuntu   "bash"    40 seconds ago   Up 38 seconds           ubu02
6d013a34f1db   alpine   "/bin/sh" 11 hours ago   Up 11 hours             alp05
$ docker attach --detach-keys="ctrl-c" ubu02
root@585a92bb74d2:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib   lib64  media  opt  root  sbin  sys  usr
root@585a92bb74d2:/# read escape sequence 
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
585a92bb74d2   ubuntu   "bash"    3 hours ago   Up 3 hours           ubu02
6d013a34f1db   alpine   "/bin/sh" 13 hours ago   Up 13 hours          alp05
```

docker attach

- «**attach**» komutuyla kullanılabilecek opsiyonlar şunlardır:
 - **--detach-keys**: konteynerden **STDIN/STDOUT**' undan ayrılmak için «**Ctrl-P**» ve «**Ctrl-Q**» yerine istenen bir tuş kombinasyonunun verilmesini sağlar. (Bunu örnekleriyle gördük)
 - **--no-stdin**: Bu opsiyon verildiğinde, konteynerin **STDIN**'ine bağlanılmaz ama **STDOUT**'a bağlanılır. Bu şekilde konteynerdeki programların **STDOUT**'a yazdığı loglar veya mesajlar görülebilir. **Ctrl-C** tuşuna basılsa da, **STDIN**'den bu sinyal gitmediği için konteyner çalışır durumda kalır.
 - **--sig-proxy** : klavyeden girilen sinyallerin doğrudan konteynere gönderilmesini sağlar.

```
$ docker run -dit --name kont99 ubuntu top -b
18b2081ee2e262f8e232a365929856f0c291a68cd8270917e52edba6147d9311
adminpc@ubuntu:~$ docker attach --no-stdin kont99
```

--no-stdin
opsiyonuyla sadece
STDOUT'a
bağlanılır.

```
top - 11:13:38 up 16:26,  0 users,  load average: 0.19, 0.13, 0.05
Tasks:  1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3.0 us,   1.6 sy,   0.0 ni,  95.2 id,   0.0 wa,   0.0 hi,   0.2 si,   0.0 st
MiB Mem :  9683.6 total,  6100.8 free,   1351.4 used,   2231.4 buff/cache
MiB Swap:  1873.4 total,  1873.4 free,     0.0 used.  7970.6 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	7180	2836	2508	R	0.0	0.0	0:00.19	top^C

```
adminpc@ubuntu:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
18b2081ee2e2	ubuntu	"top -b"	28 seconds ago	Up 25 seconds		kont99
585a92bb74d2	ubuntu	"bash"	18 hours ago	Up 18 hours		ubu02

Ctrl-C

docker exec

- «**docker exec**», çalışan bir konteyner içinde istenen bir komutun çalıştırılması için kullanılır.
- «**docker exec**» ile belirtilen komut sadece konteyner içindeki ana proses (PID=1) çalışırken çalıştırılabilir. Eğer konteyner yeniden başlatılırsa, komut sonlanır ve yeniden başlatılmaz.
- Komut, konteynerde varsayılan dizinde çalışır. «**--workdir**» opsiyonuyla komutun çalışacağı dizin değiştirilebilir.
- Komutta tek bir program ismi olmalıdır. Tırnak içinde bileşik birden fazla programı (örneğin **&&** ile zincir hale getirilmiş komutlar), «**docker exec**» çalıştırmaz. Örneğin, şu komut çalıştırılmayacaktır:

```
docker exec -it cont01 "ls -laF && echo $PATH"
```
- Ama bu komutlar şu şekilde çalıştırılabilir:

```
docker exec -it cont01 sh -c "ls -laF && echo $PATH"
```

docker exec

- «**docker exec**» komutunda kullanılan opsiyonlar şunlardır:
 - «**--detach**» veya «**-d**»: komutu konteynerde geri planda çalıştırır. (**docker run** komutundaki **-d** opsiyonu gibi)
 - «**--detach-keys**» : konteynerin **STDIN**'ininden çıkmak için kullanılan tuş kombinasyonunu değiştirir.
 - «**--env**» veya «**-e**» : çevre değişkenlerini değiştirir.
 - «**--env-file**» : belirtilen dosyadan çevre değişkenlerini okur.
 - «**-i**» : interaktif (**docker run**'daki opsiyon)
 - «**--privileged**» : komutu genişletilmiş yetkilerle çalıştırır. Örneğin, «**mount**» komutu normal yetkilerle kullanılamaz. Ama bu opsiyonla kullanılabilir.
 - «**--tty**» veya «**-t**»: **pseudo-TTY**
 - «**--user**» veya «**-u**» : komutun çalıştırılacağı kullanıcı (kullanıcı adı veya UID olabilir. Kullanıcı formatı şu şekildedir: **<isim|uid>[:<grup|gid>]**)
 - «**--workdir**» veya «**-w**» : konteyner içindeki komut çalışma dizinini değiştirir.

<https://docs.docker.com/engine/reference/commandline/exec/>

docker exec örnekler

- Aşağıdaki örnekler «**Ubuntu**» tipi konteynerler için geçerlidir.
- bir konteyner içinde «**touch**» komutuyla dosyanın tarihini güncelleyelim ve yoksa yeni bir dosya oluşturur.

```
$ docker exec -d kontey01 touch /tmp/deneme
```
- Konteyner içinde komut verilebilen ve çıktıları görülebilen interaktif **Bash** kabuğu çalıştır.

```
$ docker exec -it kontey02 bash
```
- VAR isimli bir çevre değişkenine 1 değerini yükleyip **Bash** kabuğunu interaktif olarak çalıştıracamız. VAR değişkeni sadece şimdi çalıştırdığımız **Bash** içinde geçerli olacak ve konteyner genelinde geçerli olmayacaktır.

```
$ docker exec -it -e VAR=1 kontey03 bash
```
- Durmuş bir konteynerde komut çalıştırıldığında «**FATA[0000] Error response from daemon**» mesajı verilir.

docker update

- «**docker update**» veya «**docker container update**» komutu konteynerin yapılandırmasını dinamik olarak değiştirebilir.
- Kullanımı şöyledir (birden fazla konteyner belirtilebilir):
`docker update [OPSİYONLAR] KONT1 [KONT2...]`
- Bu komut, çeşitli opsiyonlarıyla birlikte konteynerlerin gereksiz kaynak kullanımını (bellek, CPU sayısı/kapasitesi, sınırlandırmak ve konaktaki tüm kaynakları tüketmesini engellemek için kullanılır.
- «**docker run**» komutunda kullanılan opsiyonların büyük kısmı «**docker update**» için de kullanılmaktadır.
- Tek bir komutla, çalışan veya durmuş olan bir konteynerin kaynak kullanım oranlarını sınırlandırabilirsiniz. Sadece, kullanılan çekirdek bellek miktarını değiştiren «**--kernel-memory**» opsiyonu dışında, diğer sınırlandırmalar çalışan konteynerler üzerinde uygulanabilmektedir.
- «**docker update**» Windows'ta çalışmamaktadır.

<https://docs.docker.com/engine/reference/commandline/update/>

Docker imajlarının tar dosyası
olarak kaydedilmesi ve yerel
«registry» kurulması

Çevrimdışı imaj yükleme

- Her «**docker pull**» komutu, Docker'ın **hub.docker.com** ile ilişkili imaj deposuna ulaşarak istenen imajların indirilmesini sağlar.
- Ancak bazı durumlarda, (örneğin kurumsal sistemlerde güvenlik tedbirleri sebebiyle), Docker'ın resmi **registry** sistemine erişilemeyebilir.
- Böyle bir durumda, Docker yüklü başka bir sistemde indirilen imajlar, **tar** dosyası haline getirilerek bir şekilde iç taraftaki ağda bulunan makineye kurulabilir.
- Diğer bir yöntem de ağda bulunan proxy (HTTP veya HTTPS Proxy olabilir) üzerinden imajın indirilmesidir.

```
$ HTTP_PROXY="http://proxy.example.com:80/" docker pull ubuntu  
$ HTTPS_PROXY="https://proxy.example.com:443/" docker pull ubuntu
```

<https://stackoverflow.com/questions/60734760/how-to-pull-docker-images-inside-restricted-network>

<https://docs.docker.com/config/daemon/systemd/#httphttps-proxy>

İmaji tar dosyası haline getirme

- Herhangi bir imaji Docker registry'den indirerek aşağıdaki komutlarla TAR dosyası haline getirebilirsiniz.

```
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:9b8dec3bf938bc80fbe758d856e96fdfab5f56c39d44b0cff351e847bb1b01ea
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest
$ docker save -o ubuntu_son.tar ubuntu
$ ls -la ubuntu_son.tar
-rw----- 1 adminpc adminpc 80357376 Eki  3 13:15 ubuntu_son.tar
```

- Eğer isterseniz, daha sonra bunu sunucunuza yükleyebilirsiniz:

```
$ docker load -i ubuntu_son.tar
Loaded image: ubuntu:latest
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	8ca4688f4f35	4 days ago	7.34MB
ubuntu	latest	3565a89d9e81	8 days ago	77.8MB
debian	latest	2657a4a0a6d5	13 days ago	116MB
postgres	latest	2d74f8a2591c	2 weeks ago	417MB
hello-world	latest	9c7a54a9a43c	5 months ago	13.3kB

İmaji etiketleme

- Eğer isterseniz, yüklediğiniz imaja yeni bir etiket de atayabilirsiniz.

```
$ docker image tag 3565a89d9e81 ubuntu_son:latest
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine	latest	8ca4688f4f35	4 days ago	7.34MB
ubuntu	latest	3565a89d9e81	8 days ago	77.8MB
→ ubuntu_son	latest	3565a89d9e81	8 days ago	77.8MB
debian	latest	2657a4a0a6d5	13 days ago	116MB
postgres	latest	2d74f8a2591c	2 weeks ago	417MB
hello-world	latest	9c7a54a9a43c	5 months ago	13.3kB

Yerel «**registry**» oluşturma

- Eğer isterseniz, imajlarını depolamak ve çekmek için yerel bir registry kullanabilirsiniz.
- Buradaki en pratik çözüm, «**registry**» isimli imajı Docker registry'den çekmek ve **-v** opsiyonuyla çalıştırarak imajları tutacağı lokal dizini belirlemektir.
- Bu amaçla yapmanız gerekenler aşağıdaki sitede anlatılmaktadır. Konu bütünlüğü açısından bu kısım kursiyerimize bırakılmıştır.

<https://docs.docker.com/registry/deploying/#run-a-local-registry>

Docker İmajının Değiştirilmesi

Docker İmajının Değiştirilmesi

- Docker Hub'dan indirdiğiniz veya çalıştırdığımız konteynerler üzerinde değişiklik yapılabilir ve bunların üstüne yeni yazılımlar/paketler ekleyerek işlevlerini geliştirilebilirsiniz.
- Özellikle yazılım geliştirme işlerinde, yeni geliştirilen veya yeni sürümü çıkarılan yazılım, konteynerlerin içine kaydedilebilir ve gerekirse içindeki kütüphanelerde değişiklikler yapılabilir.
- Güncelleme işleri otomasyon araçlarıyla yapılabildiği gibi, konteynere bağlanarak elle de güncelleme gerçekleştirilebilmektedir.
- Bu bölümde, konteynerlerin nasıl elle güncelleneceğini öğreneceğiz.
- Güvenilir bir güncelleme ve konteyner oluşturma işleri için otomasyon uygulamalarının kullanılması «DevSecOps» anlamında önemlidir.

Çalışma: Ubuntu İmajının uyarlanması

- Yapacağımız işlemler:

1. «Docker hub» içinde bulunan Ubuntu imajını «**docker pull**» komutuyla indireceğiz ve «**docker images**» komutuyla **IMAGE-ID**'sini bularak **Ctrl-C** ile kopyalayacağız.
2. Ubuntu imajının üzerinde «**/bin/bash**» kabuk programını interaktif şekilde çalıştırmak üzere «**docker run -it IMAGE-ID bin/bash**» komutunu vereceğiz. Diğer bir deyişle konteynere komut yazabilmek için orada çalışan Bash kabuğuna bağlanacağız.
3. Konteyner içinde **root** yetkilerine sahibiz ve çeşitli paketleri imaj içinde «**apt install**» komutuyla kurabiliriz.
4. Paketleri kurduktan sonra verdiğimiz «**exit**» komutuyla kabuktan çıkacağız.
5. «**docker commit**» komutuyla değiştirdiğimiz imajın farklı bir isimde kopyasını çıkaracağız.


Çalışma: Ubuntu İmajının değiştirilmesi

- Docker Hub'dan ubuntu imajını indirelim:

docker pull ubuntu

```
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
125a6e411906: Pull complete
Digest: sha256:26c68657ccce2cb0a31b330cb0be2b5e108d467f641c62e13ab40cbec258c68d
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

- «**docker images**» komutuyla, Ubuntu imajının yüklenip yüklenmediğini ve «**IMAGE ID**» değerini görebilirsiniz. **Ctrl-C** ile kopyalayın.



```
$ docker images
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
ubuntu        latest    d2e4e1f51132  2 seconds ago  77.8MB
alpine        latest    e66264b98777  2 days ago    5.53MB
hello-world   latest    feb5d9fea6a5  8 months ago  13.3kB
```

<https://phoenixnap.com/kb/how-to-commit-changes-to-docker-image>

Çalışma: Ubuntu İmajının değiştirilmesi

- Ubuntu imajını üzerinde **bash** kabuk programıyla çalıştırarak, interaktif şekilde ulaşacağız ve ardından **vim** paketini «**apt-get install vim**» komutuyla kuracağız.

```
$ docker run -it imaj bin/bash
```

```
$ docker run -it ubuntu bin/bash  
root@63d083377f28:/#
```

- Konteyner üzerinde paket kurabilmek için ilk önce, «**apt-get update**» komutunun verilmesi gereklidir. Bu komut, paket indirme ve kurmak için gerekli veri yapılarını oluşturur.
- «**apt-get install vim**» komutuyla vim paketi indirilerek kurulabilir.

«apt-get update»

```
$ docker run -it ubuntu bin/bash
root@63d083377f28:/# apt-get update
Get:1 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [109 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [99.8 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 Packages [1792 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [17.5 MB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [68.8 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [156 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [153 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [4653 B]
Get:11 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 Packages [164 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [266 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [4653 B]
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [157 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [121 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [265 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [1202 B]
Fetched 21.2 MB in 6s (3574 kB/s)
Reading package lists... Done
```

«apt-get install vim»

```
root@63d083377f28:/# apt-get install vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libexpat1 libgpm2 libmpdec3 libpython3.10 libpython3.10-minimal libpython3.10-stdlib libreadline8 libsodium23
  libsqlite3-0 media-types readline-common vim-common vim-runtime xxd
Suggested packages:
  gpm readline-doc ctags vim-doc vim-scripts
The following NEW packages will be installed:
  libexpat1 libgpm2 libmpdec3 libpython3.10 libpython3.10-minimal libpython3.10-stdlib libreadline8 libsodium23
  libsqlite3-0 media-types readline-common vim vim-common vim-runtime xxd
0 upgraded, 15 newly installed, 0 to remove and 4 not upgraded.
Need to get 14.5 MB of archives.
After this operation, 61.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 libexpat1 amd64 2.4.7-1 [90.7 kB]
Get:2 http://archive.ubuntu.com/ubuntu jammy/main amd64 libmpdec3 amd64 2.5.1-2build2 [86.8 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy/main amd64 libpython3.10-minimal amd64 3.10.4-3 [809 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy/main amd64 media-types all 7.0.0 [25.5 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 readline-common all 8.1.2-1 [53.5 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/main amd64 libreadline8 amd64 8.1.2-1 [153 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy/main amd64 libsqlite3-0 amd64 3.37.2-2 [643 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy/main amd64 libpython3.10-stdlib amd64 3.10.4-3 [1830 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/main amd64 xxd amd64 2:8.2.3995-1ubuntu2 [51.1 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy/main amd64 vim-common all 2:8.2.3995-1ubuntu2 [81.5 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy/main amd64 libgpm2 amd64 1.20.7-10build1 [15.3 kB]
Get:12 http://archive.ubuntu.com/ubuntu jammy/main amd64 libpython3.10 amd64 3.10.4-3 [1951 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy/main amd64 libsodium23 amd64 1.0.18-1build2 [164 kB]
Get:14 http://archive.ubuntu.com/ubuntu jammy/main amd64 vim-runtime all 2:8.2.3995-1ubuntu2 [6825 kB]
Get:15 http://archive.ubuntu.com/ubuntu jammy/main amd64 vim amd64 2:8.2.3995-1ubuntu2 [1724 kB]
Fetched 14.5 MB in 33s (444 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
```

Docker Ağ Köprüsü Sorunu

- Eğer Ubuntu imajında, «**apt-get update**» komutu çalışmıyorsa ve ağa bağlanmakta zorluk çektiğine dair ifadeler varsa, büyük bir olasılıkla **Docker'ın ağ köprüsü (bridge) çökmüştür.**

```
root@4fba1810f6b6:/# apt-get update
Err:1 http://archive.ubuntu.com/ubuntu focal InRelease
      Temporary failure resolving 'archive.ubuntu.com'
Err:2 http://security.ubuntu.com/ubuntu focal-security InRelease
      Temporary failure resolving 'security.ubuntu.com'
Err:3 http://archive.ubuntu.com/ubuntu focal InRelease
      Temporary failure resolving 'archive.ubuntu.com'
```

- Docker'ın çalıştığı konakta (host) şu komutların çalıştırılması gerekir:
`sudo apt-get install bridge-utils`
`sudo pkill docker`
`sudo iptables -t nat -F`
`sudo ifconfig docker0 down` **veya** `sudo ip link delete docker0`
`sudo brctl delbr docker0`
`sudo systemctl restart docker`
- Bu durumda, Docker'ın yeniden ağ köprüsünü (bridge) oluşturması zorlanmıştır ve «**apt-get update**» komutu tekrar verildikten sonra «**apt-get install vim**» denilmelidir.

Çalışma: Ubuntu İmajının değiştirilmesi

- «**Ubuntu**» konteynerinden «**exit**» komutuyla çıkın.
- «**docker ps -a**» komutuyla, üzerinde değişiklik yaptığınız imajın konteyner ID'sini bulunuz.

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
63d083377f28	ubuntu	"bin/bash"	13 minutes ago	Exited (130) 52 seconds ago		objective_dijkstra
d277bf460276	alpine	"/bin/sh"	2 hours ago	Up 2 hours		

- «**docker ps -a**» komutuyla, Ubuntu'nun **IMAGE-ID**'sinden oluşturulan **CONTAINER-ID**'yi kullanarak, konteynerde yapılan değişiklikleri ve eklemeleri de alarak yeni bir imaj oluşturmak için aşağıdaki komutu kullanabiliriz:

```
docker commit 63d083377f28 ubuntu-guncel
```

```
$ docker commit 63d083377f28 ubuntu-guncel
sha256:df531d64f1e511b65ca5efa677e41fc3cf8888b2588dd770e08ffb4b434ed19
```

İmaj ismi

- Dönen SHA256 değerinin ilk 12 dijiti, **image ID** olarak kullanılabilir. «**docker images**» komutuyla yeni oluşturulan **ubuntu-guncel** imajının ID'si görülebilir.

Docker ağ sistemi

Docker Ağ Sistemi

- Docker konteynerleri birbirlerinden izole şekilde çalışmaları gerekmekte birlikte, genellikle birbirleriyle ağ üzerinden haberleşmeleri gerekir.
 - Ağ üzerinden TCP/IP haberleşmesi kurabilirler
 - Disk üzerindeki dosyaları paylaşabilirler.
 - NFS, Samba gibi uygulamalarla dosya sistemi paylaşımı yapabilirler
 - Konteyner uygulaması bir veritabanı konteynerine ulaşip veri sorgulayabilir veya yazabilir.
- Konteynerler belirli bir ağ servisini verme konusunda ideal uygulamalardır:
 - Apache, Nginx, vb
 - Node, JBoss, Spring Boot, vb
 - MongoDB, PostgreSQL, vb
- Bütün bu işlevleri gerçekleştirmek için Docker'a sanal bir ağ oluşturması ve her bir ağ için de ağ köprüsü (network bridge) oluşturması istenebilir.
- Aynı ağ içindeki konteynerlerin birbirleriyle haberleşmeleri ve İnternet'e bağlanmaları sağlanabilir.

<https://www.tutorialworks.com/container-networking/>

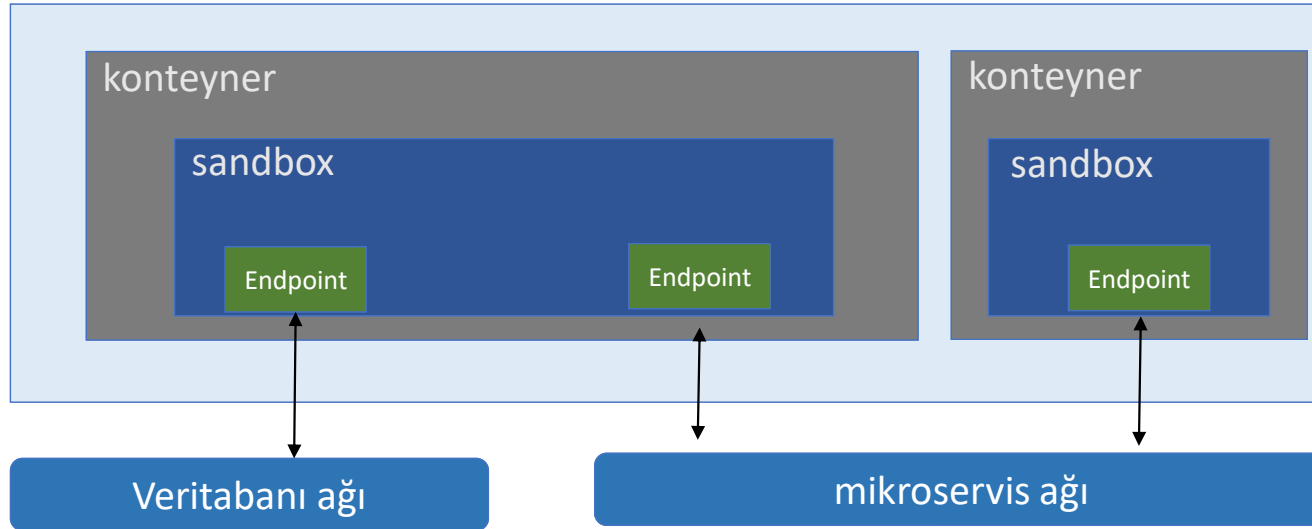
Konteyner Ağ Modeli

- Docker ağ sisteminde üç farklı bileşen bulunur:
 - Konteyner Ağ Modeli (CNM: Container Network Model): ağ tanımlama modelidir.
 - **libnetwork**: CNM'in gerçekleştirilmesini sağlar.
 - Sürücüler: ağ topolojilerinin uygulanmasını sağlar
- Ağ sürücüleri
 - **bridge** (köprü): link katmanında bir veya fazla konteynerin bağlanmasını sağlar. Diğer ağlardaki konteynerlerin bağlanmasını engeller.
 - **host**: konak
 - **overlay**: Docker Swarm uygulamalarında, farklı konaklara dağılmış ağların bir araya getirilmesini sağlar
 - **macvlan**: herhangi bir konteynere MAC adresi verilmesini sağlar. Bu şekilde fiziksel bir cihaz olarak çalışabilir. Ayrıca VLAN'lara bağlanabilir.
 - **none**: ağ sisteminin çalışmasının engellenmesi
 - Ağ «plugin»leri: çeşitli uygulamalarla gelen ek özellikler

CNM (Container Network Model)

- Konteyner Ağ Modelinde üç bileşen bulunur:
 - **sandbox**: ağ yığıtını izole eder.
 - **endpoint**: sanal ağ ara yüzleridir (virtual NIC). **Sandbox**'ların ağa bağlantı noktalarıdır. Bir konteynerde, her biri ayrı bir ağa bağlanan birden fazla **endpoint** bulunabilir.
 - **network** (ağ): Sanal ağ ara yüzü üzerinden **sandbox**'un bağlandığı ağ yapısıdır.

Docker host



docker0

- **docker0**, Docker kurulduğunda, konteynerleri fiziksel ağ adaptörüne bağlamak için oluşturulan sanal ağ köprüsüdür.
- «**ip link show**» veya «**ifconfig**» komutlarıyla görülebilir.
- Yeni bir konteyner oluşturulduğunda **docker0**'da tarif edilen IP numaralarından birini alarak çalışmaya başlar.



```
$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:d6ff:fe9a:a408 prefixlen 64 scopeid 0x20<link>
    ether 02:42:d6:9a:a4:08 txqueuelen 0 (Ethernet)
    RX packets 4893 bytes 202177 (202.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7669 bytes 36140694 (36.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.211.139 netmask 255.255.255.0 broadcast 192.168.211.255
    inet6 fe80::8c20:dd5a:c76d:79d8 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:b1:0d:f2 txqueuelen 1000 (Ethernet)
    RX packets 151031 bytes 221474065 (221.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 25662 bytes 1666142 (1.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Docker Ağ Sistemi

- Docker, konteynerlerin bağlanabileceği farklı sanal ağ sistemleri oluşturabilmektedir.
- Aynı ağ sistemine bağlanan konteynerler birbirlerini görerek haberleşebilmektedir.
- Docker, eğer başka bir ağa atanmadıkları sürece çalıştırılan konteynerleri varsayılan «**bridge**» adlı ağa ekler ve ona göre IP adresi olarak ağa bağlanmalarını sağlar.
- Ağları listelemek için «**docker network ls**» komutu kullanılır. Görüntülenen «**bridge**» isimli ağ, varsayılan ağdır.



```
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
275b9ffa6c55    bridge    bridge      local
9e46cf32e837    host      host        local
71255d3cd5b6    none      null        local
```

Ağ Özellikleri

- **bridge**, Docker'da çalışan bütün konteynerlerin ağa çıkışını sağlayan köprüdür. Bunun özellikleri JSON formatında şu komutla görülebilir.

`$ docker network inspect bridge`

- JSON çıktısı, Docker'daki **bridge** ağıyla ilgili çeşitli bilgilerin görölmesini sağlar.
- **IPAM** (IP Address Management) bölümünde, **bridge** üzerinden hangi IP adreslerinin konteynerlere atanacağı, konteynerlere hangi IP adreslerinin ulaşabileceği, vb bir çok bilgi bulunur.

IPAM

```
$ docker network inspect bridge
```

```
[
  {
    "Name": "bridge",
    "Id": "275b9ffa6c55900ec50a0213f546fb57db59a47060a64211e2bca9ba6f102231",
    "Created": "2022-05-26T01:20:41.303299272-07:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
  },
]
```

Driver olarak «**bridge**»
kullanıyor.

Konteynerlere verilen IP'ler
172.17.0.2-172.17.255.254
arası olacak.

Konteynerlere verilen IP'ler

- Docker altında çalışan konteynerlere verilen IP'ler ve MAC adresleri görülebilir.

```
"Containers": {  
  "061fdaf613d8a741d54b0ca10968d568a2146ef54b4c2e9d22cc109f3b17090d": {  
    "Name": "sunucu5",  
    "EndpointID":  
    "6bf91e751a7ba76666b2fc4936683c95ed9b662f881067f8bdd2d517a687f351",  
    "MacAddress": "02:42:ac:11:00:04",  
    "IPv4Address": "172.17.0.4/16",  
    "IPv6Address": ""  
  },  
  "12aefa40b82a45baa6192e49552df2f3f9a279ee12d76620928eca07e9fb81c3": {  
    "Name": "sunucu4",  
    "EndpointID":  
    "28c5354679360285bc26842281892020966985f586c7cb508a0",  
    "MacAddress": "02:42:ac:11:00:03",  
    "IPv4Address": "172.17.0.3/16",  
    "IPv6Address": ""  
  },  
  "d277bf460276335b7bd6d69faaf9ce6d147c5ad1af22d4cf9881a2f4cf0344e2": {  
    "Name": "sunucu3",  
    "EndpointID":  
    "4ed9793c30e582b0a7c190be5c8cb42de3e121e525d854790aa271b367fa0ff1",  
    "MacAddress": "02:42:ac:11:00:02",  
    "IPv4Address": "172.17.0.2/16",  
    "IPv6Address": ""  
  }  
},
```

Kont.1

Kont.2

Kont.3

Bu ağa bağlı olan bütün konteynerler, birbirlerine «--name» ile belirlenmiş adları üzerinden erişebilirler.

docker network create

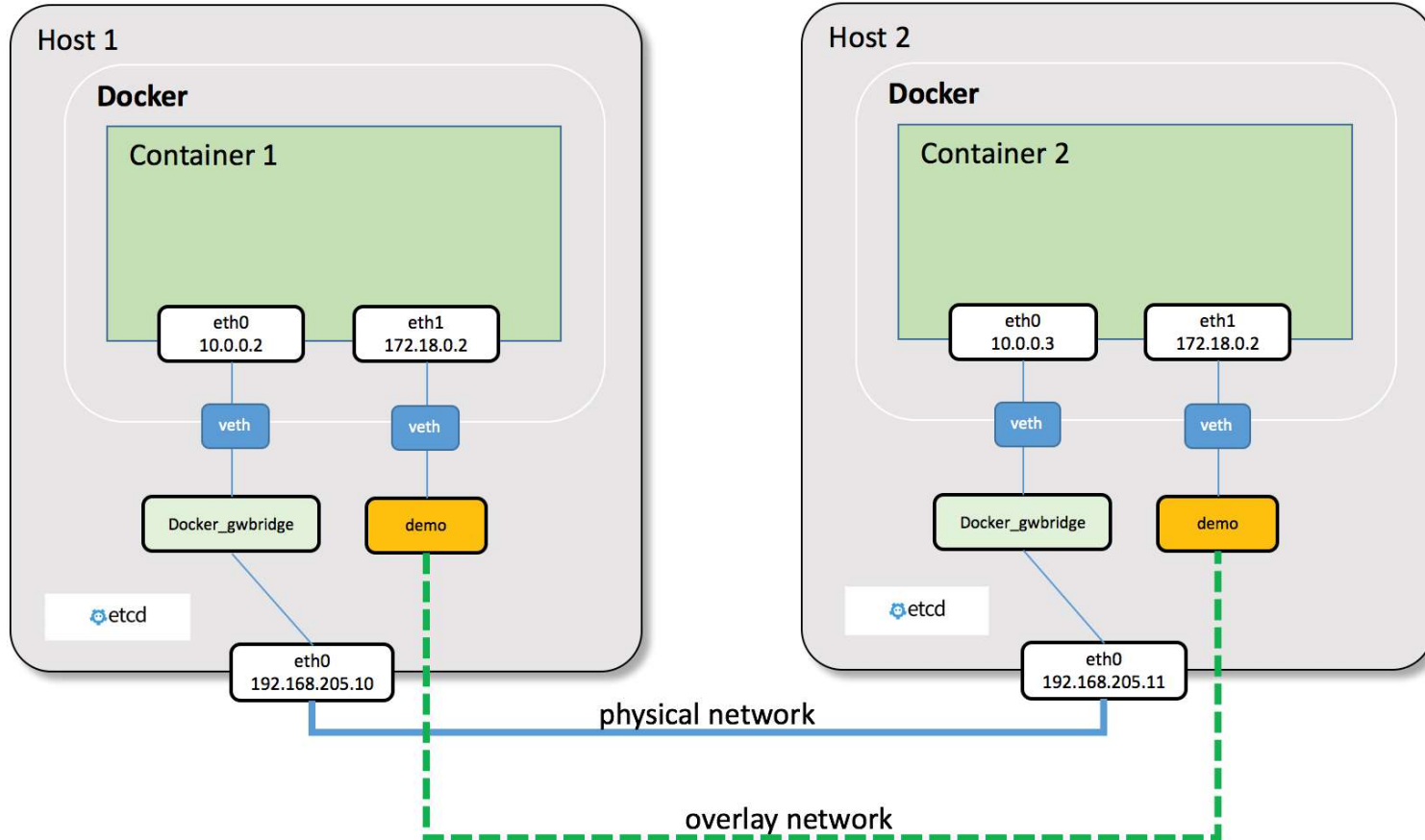
- Yeni bir oluşturmak için kullanılan «**docker network create**» komutunun kullanımı şöyledir:
\$ **docker network create** [OPSİYONLAR] AĞ-İSMİ
- Normal şartlarda **--driver** (veya **-d**) opsiyonu, kullanılacak sürücüyü belirler. Sürücü olarak «**bridge**» veya «**overlay**» sürücüleri Docker ile gelmektedir.
\$ **docker network create --driver bridge yeniag**
- Ancak sürücü belirtilmezse, Docker tarafından varsayılan olarak «**bridge**» sürücüsü kullanılır. «**bridge**» ağları, tek bir konak üzerinde konteynerlere bağlanan ve «**docker0**» üzerinden dış ağa bağlanan izole ağlardır.

docker network create

- Eğer, üzerinde Docker çalıştıran birden fazla konakta çalışan konteynerlerin bağlı olduğu bir ağ oluşturulacaksa, «**--driver**» olarak «**overlay**» kullanılmalıdır. Ancak «**overlay**» ağları oluşturabilmek için bazı özel koşulları sağlamak gerektiğinden kurulumu zor olabilir.
- Docker, «**bridge**» ve «**overlay**» sürücüleri tanımlı bir şekilde gelmektedir. Bunlar dışında da sürücüler vardır:
 - **host**: konteynerin doğrudan konağın (host) IP numarası ve portları üzerinde hizmet vermesini sağlar.
 - **ipvlan**: IPv4 temelli VLAN oluşturmak ve IP temelli yönlendirme yapmak için kullanılır.
 - **macvlan**: bir konteynere MAC adresi atamak ve Docker daemon'un MAC adreslerine göre yönlendirme yapması isteniyorsa kullanılır.
 - **none**: konteynerler için ağ özelliklerinin kapatılmasını sağlar. Bu sürücü genellikle, konteynerin özel bir sürücü kullanması istediğinde kullanılır.

https://docs.docker.com/engine/reference/commandline/network_create/

«overlay» ağı



«--network host»

- Bu opsiyon, konteyner ve Docker'ın üzerinde çalıştığı konak arasındaki ağ izolasyonunu kaldırarak, konağın (host) ağına doğrudan bağlanmasını sağlar.
- «**host**» ağındaki konteynere ayrı bir IP numarası verilmez ve konağın IP'sini kullanır. Konak üzerinde bir proses gibi çalışır. Konteynerde açık olan portlar, konaktaki aynı numaralı portlar üzerinden ağa bağlanır.
- «**host**» ağına bağlı bir konteynerin ağ özellikleri boştur.

```
"Networks": {  
  "host": {  
    "IPAMConfig": null,  
    "Links": null,  
    "Aliases": null,  
    "NetworkID": "9e46cf32e8379adb5548e468af",  
    "EndpointID": "73dc1f7b02e3a1af3ba2ba26ba",  
    "Gateway": "",  
    "IPAddress": "",  
    "IPPrefixLen": 0,  
    "IPv6Gateway": "",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "MacAddress": "",  
    "DriverOpts": null  
  }  
}
```

Bir «**nginx**» konteyneri host ağına bağlandığında, 80 numaralı port hostun da 80 numaralı portu olur.

Docker Ağ Komutları

- **docker network connect:** konteyneri bir ağa bağlar. Eğer konteyner başka bir ağda çalışıyorsa, yeni bir «**endpoint**» ekleyerek yeni ağa bağlar ve eskisini de korur.
- **docker network create:** yeni ağ oluşturmak
- **docker network disconnect:** konteyneri ağdan çıkartmak
- **docker network inspect:** ağla(rla) ilgili detaylı bilgi
- **docker network ls:** ağları listelemek
- **docker network prune:** kullanılmayan ağları temizle
- **docker network rm:** ağ(lar)ı silme

https://docs.docker.com/engine/reference/commandline/network_connect/

Opsiyonlar

- «**docker network inspect bridge**» komutu verildiğinde ağla ilgili görülen JSON dosyasındaki «**Options**» bölümünde çeşitli detaylar görülebilir.
 - **default_bridge**: "true" : varsayılan ağa ana çıkış köprüsü olup olmadığı
 - **host_binding_ipv4**: "0.0.0.0": bütün ara yüzlerden bu köprüye geldiğini göstermektedir.
 - **name**: "docker0": **docker0**, Docker paketleri kurulunca oluşturulan ana NIC'in (Network Interface Card) adıdır.

```
$ docker network inspect bridge
...
...
"Options": {
    "com.docker.network.bridge.default_bridge": "true",
    "com.docker.network.bridge.enable_icc": "true",
    "com.docker.network.bridge.enable_ip_masquerade": "true",
    "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
    "com.docker.network.bridge.name": "docker0",
    "com.docker.network.driver.mtu": "1500"
},
```

Opsiyonların değiştirilmesi

- Sürücü ile ilgili parametrelerin değiştirilmesi için «**docker network create**» komutunda «**--opt**» opsiyonu, «**Options**» kısmındaki parametreleri değiştirmemizi sağlayabilir.
- Örneğin, «**bridge**» sürücüsü ile oluşturacağımız ağın MTU değerini 9216 bayt olarak değiştirmek için şu komutu kullanabiliriz.

```
$ docker network create -d bridge  
--subnet=10.11.0.0/16 --opt  
com.docker.network.driver.mtu=9216 --opt  
encrypted=true benimagim
```

Docker Ağ Sistemi

- Çalışan bir konteynerin hangi IP numarasını aldığını görebilmek için «**docker inspect konteynerID**» komutu kullanılabilir.
«**"IPAddress": "172.17.0.4"**» gibi bir satır konteynerin aldığı IP adresin **172.17.0.4** olduğunu gösterir.

```
"Networks": {  
  "bridge": {  
    "IPAMConfig": null,  
    "Links": null,  
    "Aliases": null,  
    "NetworkID": "275b9ffa6c55900ec50a0213f546fb57db59a47060a64211e2bca9ba6f102231",  
    "EndpointID": "6bf91e751a7ba76666b2fc4936683c95ed9b662f881067f8bdd2d517a687f351",  
    "Gateway": "172.17.0.1",  
    "IPAddress": "172.17.0.4",  
    "IPPrefixLen": 16,  
    "IPv6Gateway": "",  
    "GlobalIPv6Address": "",  
    "GlobalIPv6PrefixLen": 0,  
    "MacAddress": "02:42:ac:11:00:04",  
    "DriverOpts": null  
  }  
}
```

Yeni Ağ Oluşturma

- Aynı ağ sistemine bağlanan konteynerler birbirlerini ağda görerek haberleşebilmektedir. Diğer ağlara erişemezler.
- Güvenlik açısından, konteynerleri gruplayarak, her bir grubu da farklı sanal ağlarda toplamak iyi bir güvenlik tedbiri olabilir.
- Aynı ağ içinde yer alan konteynerler, birbirleriyle «**--name**» opsiyonuyla verilmiş adları üzerinden haberleşebilir.
- «**vtb-ag**» isimli yeni sanal ağ oluşturmak için verilmesi gereken komut şudur:

```
$ docker network create vtb-ag
```

```
$ docker network create vtb-ag
0459dcffffcb18b5c18e91dbbf9c9cf9c748f03de530de12c622c8cc294eef1dd
$ docker network ls
NETWORK ID          NAME       DRIVER      SCOPE
275b9ffa6c55        bridge    bridge      local
9e46cf32e837        host      host        local
71255d3cd5b6        none      null        local
0459dcffffcb1      vtb-ag    bridge      local
```


Yeni ağın özellikleri

\$ docker network inspect vtb-ag

```
$ docker network inspect vtb-ag
[
  {
    "Name": "vtb-ag",
    "Id": "0459dcffffcb18b5c18e91dbbf9c9cf9c748f03de530de12c622c8cc294eef1dd",
    "Created": "2022-05-26T03:32:23.83040316-07:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

Docker'ın varsayılan ağı olan «**bridge**» ağındaki IP numaralarından farklı IP numaralarının, bu ağa atanmış olduğu görülebilir.

Ağın «**internal**» yani dahili bir ağ olmadığı belirtiliyor. Diğer bir deyişle, bu ağa dış ağdan erişilebilir ve ping atılabilir.

Subnet ve GW vererek Yeni Ağ Oluşturma

- «**docker network create**» komutu yalın bir şekilde ve opsiyonlar belirtilmeden yeni bir ağ oluşturmak için kullanılırsa, hangi IP aralığının kullanacağı ve IP numarası dağıtacağını Docker'ın kendisi belirlemektedir.
- Bazen, oluşturulan ağın IP numaralarını belirlemek gerekebilir. Bu durumda, «**network create**» komutu «**--subnet**» ve «**--gateway**» opsiyonlarıyla birlikte çağrılmalıdır.
- Sadece «**subnet**» belirtilerek ve «**gateway**» tanımlanmadan da ağ oluşturulabilir. Bu durumda Docker uygun bir IP numarasını geçit (gateway-GW) olarak atayacaktır.
- Aşağıdaki komut, **10.10.0.0/24** alt ağında **10.10.0.1**'i ağ geçidi olarak tanımlamaktadır:

```
docker network create --subnet 10.10.0.0/24 --gateway  
10.10.0.1 ms-ag
```

Yeni Ağ Oluşturma

- «**docker network create**» komutu için belirlenebilecek opsiyonlar şunlardır:
 - subnet** : Ağın hangi IP alt ağında yer alacağı
 - gateway**: ağ geçidinin IP'si
 - driver**: hangi sürücü (bridge, overlay, vb) kullanılacağı
 - ip-range**: hangi aralıkta IP dağıtılacağı
 - label** : bilgi için etiket
- Örneğin: **10.10.*.*** ile başlayan alt ağ ve **10.10.0.1** geçit IP'si kullanılacak ve **10.10.8.1**'den itibaren **10.10.8.254**'e kadar konteynerlere IP dağıtılacaksa şöyle bir komut verilebilir:

```
$ docker network create --subnet 10.10.0.0/16 --gateway 10.10.0.1 --ip-range=10.10.8.0/24 --driver=bridge --label=mikrosrv ms-ag
```

Yeni Ağ Oluşturma

- Yarattığımız **ms-ag** isimli ağın özellikleri «**docker inspect ms-ag**» komutunu verebiliriz.

```
$ docker inspect ms-ag
[
  {
    "Name": "ms-ag",
    "Id": "667a033fc0edf8fffb1adc055d209a8a1d6588cd25478214adc068f30434ffec",
    "Created": "2022-07-19T12:42:51.295182826+03:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "10.10.0.0/16",
          "IPRange": "10.10.8.0/24",
          "Gateway": "10.10.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {
      "mikrosrv": ""
    }
  }
]
```

Driver olarak «**bridge**» varsayılan olduğu için seçilmiştir.

Oluşturduğumuz ağa bağlanacak konteynerlere verilerek IP numaraları **10.10.0.0/16** olduğu görülebilir.

Konteyneri Ağa Bağlama

- Çalıştırılacak bir konteyneri bir ağa ekleyerek çalıştırmak için kullandığınız «**docker run**» komutuna «**--net <ağın_ismi>**» opsiyonunun eklenmesi gereklidir.
- Eğer çalışan bir konteyner «**ms-ag**» ağına bağlamak için «**docker network connect ms-ag <konteyner-ID>**» komutu kullanılabilir. Daha önce bağlı olduğu ağ yanında, «**ms-ag**» ağına da bağlanacaktır. Eski ağından çıkarmak için «**disconnect**» komutu kullanılmalıdır.
- «**docker inspect**» komutunda konteynerin ID'i verildiğinde çalışan konteynerin aldığı yeni **gateway** ve IP adresi görülebilir.



```
"Networks": {
  "ms-ag": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "43ac09fea671"
    ],
    "NetworkID": "667a033fc0edf8fffb1adc055d209a8a1d6588cd25478214adc068f30434ffec",
    "EndpointID": "aa11540221ec326bad617df0f4a5205dfe844c40af3e597ca3c52d33fb14fc2f",
    "Gateway": "10.10.0.1",
    "IPAddress": "10.10.8.0",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "02:42:0a:0a:08:00",
    "DriverOpts": null
  }
}
```

Konteynere statik IP verilmesi

- Bazı durumlarda, konteyneri çalıştırırken belirli bir IP alarak çalışması istenebilir. Aynı ağda birden fazla konteyner ortaklaşa çalışıyorlarsa ve birbirlerinde bulunan sunuculara IP/port üzerinden bağlanıyorlarsa bunlara statik IP adresi verilmesi iyi olacaktır.
- Bu şekilde, örneğin, bir konteyner veritabanı sunucusunun IP adresinin **10.10.8.7** olacağını bilirken, veritabanı sunucusu da **nginx** (web) sunucusunun **10.10.8.5**'te olduğunu bilmelidir.
- Konteyneri çalıştırırken, belirli bir ağa bağlanmasını (**servisnet**) ve statik bir IP numarası (**10.10.8.5**) verilmesi aşağıdaki komutla gerçekleştirilebilir (**ÖNEMLİ: ağı oluştururken --subnet opsiyonunun kullanılarak ağdaki IP numaralarının belirlenmesi gerekir. Yoksa konteynere statik IP verilemez**):

```
$ docker network create --subnet=172.99.0.0/16 servisnet  
$ docker run -d --name websunucu --ip 172.99.0.11 --network servisnet nginx
```

Çalışan konteynerin ağa bağlanması

- ★ Konteynerler «**docker run**» komutuyla başlatılırken «**--network**» opsiyonuyla sadece tek bir ağa bağlanabilirler. Konteynerler çalıştırıldıktan sonra diğer bir ağa bağlanmalıdır.
- Çalışan bir konteyner varsa ve bunu diğer bir ağa bağlamak isterseniz, «**docker network connect <ağ_adı> <konteynerID>**» komutuyla bunu gerçekleştirebilirsiniz.
- Bir konteyneri ağdan çıkarmak için «**docker network disconnect <ağ_adı> <konteynerID>**» komutu kullanılmalıdır.
- Bir konteyneri dahili bir ağa bağladıktan sonra varsayılan «**bridge**» ağından çıkarılırsa, bu konteynere dışarıdan erişilemez. Bu konteynere sadece bu dahili ağdaki diğerleri erişebilir.

Konteyner DNS sunucu

- Konteynerleriniz, varsayılan olarak konak isimlerini çözebilmek için DNS sunucusu olarak **/etc/resolv.conf** dosyasında tanımlı olanı kullanır.
- Konteynerinizin farklı bir DNS adresini kullanmasını istiyorsanız (örneğin **8.8.8.8** Google'un DNS'i kullanılabilir), «**--dns**» opsiyonuyla istenen DNS sunucusu belirtilebilir.

```
$ docker run -dt --dns 8.8.8.8 nginx
```

- Konteynerinizin konak ismi genellikle konteynerin ID'sidir. Eğer değiştirmek isterseniz, «**--hostname**» opsiyonuyla konteynerin konak (host) ismi değiştirilebilir. Konteyner içindeki «**/etc/hostname**» dosyası içine bu konak ismi konulacaktır.

```
$ docker run -dt --hostname denemehost 8.8.8.8 nginx
```


Proxy kurulumu

- Konteynerler, sistemde bulunan bir **HTTP**, **HTTPS** veya **FTP** vekil (Proxy) sunucusu üzerinden web'e bağlanıyorsa, sistem iki şekilde yapılandırılabilir.
- Birinci yöntemde, `~/ .docker/config.json` dosyasının oluşturulması ve aşağıdaki içeriğin eklenmesi gereklidir. Eğer istenirse, «**noProxy**» opsiyonuyla, hangi sitelere erişimin vekil üzerinden yapılmayacağı da belirlenebilir.

```
{
  "proxies":
  {
    "default":
    {
      "httpProxy": "http://192.168.1.12:3128",
      "httpsProxy": "http://192.168.1.12:3128",
      "noProxy": "*.deneme.com, .turkiye.gov.tr, 127.0.0.0/8"
    }
  }
}
```

Virgülle ayrılmış şekilde, hangi sitelere erişimin vekil üzerinden yapılmayacağı belirtilebilir. Burada IP numaraları için CIDR notasyonu kullanılabilir

<https://docs.docker.com/network/proxy/>

Proxy kurulumu

- İkinci yöntemde, imajı oluştururken veya konteyneri çalıştırırken «**--env**» opsiyonuyla vekil sunucu tanımı yapılabilir. «**HTTP_PROXY**», «**HTTPS_PROXY**», «**FTP_PROXY**» ve «**NO_PROXY**» değişkenlerine konulan IP numaraları vekil sunucu tanımlarına eklenecektir.

```
$ docker run -it --name denekon01 --env  
HTTP_PROXY="http://10.10.4.56:8080" nginx
```

```
$ docker run -it --name denekon02 --env  
NO_PROXY="127.0.0.0/8, 10.10.4.81,  
*.deneme.com" ubuntu
```

```
$ docker run -it --name denekon02 --env  
FTP_PROXY="ftp://10.0.2.1" jenkins
```

<https://docs.docker.com/network/proxy/>

--add-host opsiyonu

- Bazen DNS'te olmayan isimler için IP numarası atanması gerekir. Linux'te bu tür durumlar için **/etc/hosts** dosyası kullanılır ve IP numaralarına isim verilmesi burada sağlanır.
- En kolay yol, konteynerlerde **/etc/hosts** dosyasına ekleme yapmadan, «**docker run --add-host**» opsiyonuyla konak ismiyle (hostname) IP numarasının eşlenmesidir.
- Örneğin, Docker'ın çalıştığı konağın IP numarasını «**ip address show**» komutuyla görelim ve **alpine** konteynerini «**-it**» opsiyonuyla çalıştırarak içinde «**ping docker**» komutunu verelim.

```
$ ip address show ens33 | grep "inet "  
    inet 192.168.211.139/24 brd 192.168.211.255 scope global noprefixroute ens33  
$ docker run --rm -it --add-host=docker:192.168.211.139 alpine  
/ # ping -c 2 docker  
PING docker (192.168.211.139): 56 data bytes  
64 bytes from 192.168.211.139: seq=0 ttl=64 time=0.693 ms  
64 bytes from 192.168.211.139: seq=1 ttl=64 time=0.531 ms  
  
--- docker ping statistics ---  
2 packets transmitted, 2 packets received, 0% packet loss  
round-trip min/avg/max = 0.531/0.612/0.693 ms  
/ #
```

Konağın IP adresini
öğrenip, «**docker**» ismini
veriyoruz.