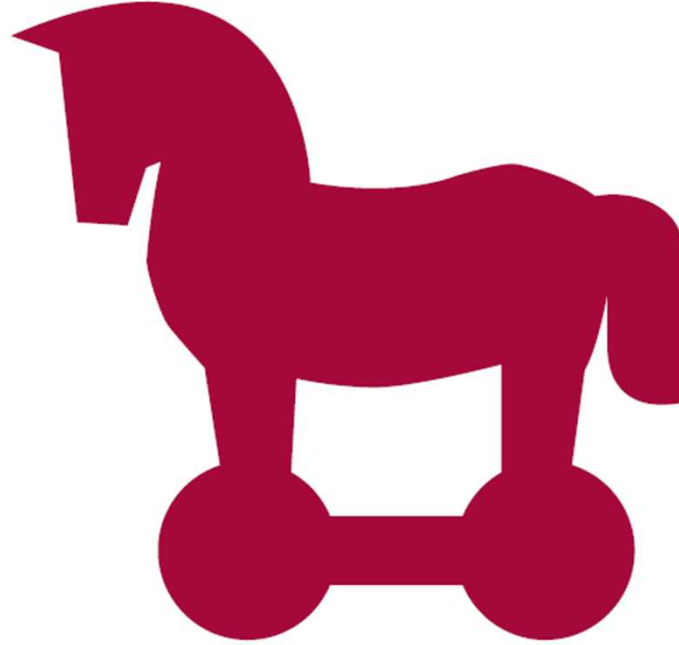


GANTEK ACADEMY

40+ YILLIK TECRÜBE İLE



GANTEK

Docker örnek (nginx)

nginx

- Yüksek başarımlı açık kaynak kodlu bir web sunucusudur.
- **nginx**, ayrıca yük paylaşım sistemi, ters proxy ve web önbellek içinde de kullanılır.
- Bu örnekte, docker komut satırı programıyla Docker Hub'dan **nginx** imajının son versiyonunun indirilerek arka planda çalıştırılmasını sağlayacağız (**-P** opsiyonu, konteynerin bütün portlarının hosta aktarılmasını ve 49153'ten itibaren Docker'in belirlediği port numaraları üzerinden erişilmesini sağlar).

docker container run -P -d nginx

- «**inspect**» komutuyla, konteynerin hangi IP numarasıyla çalıştığını göreceğiz.
- Konak (host) üzerinden konteynerdeki web sunucusuna **curl** programıyla erişerek ana web sayfasını görüntüleyeceğiz.

nginx

- **docker container run -P -d nginx**
 - **-P:** **nginx** 80 numaralı portu kullanacak ama konakta (host) aynı port kullanıldığında çakışma olacağından konaktaki başka bir port üzerinden erişilebilecek. Genellikle 49153'ten başlayan portlar konteynerlerin dışarıya açılan portlarına bağlanmaktadır.
 - **-d:** «**detached mode**» : arka planda çalışır vaziyette durması sağlanmakta ve konsoldan ayrılması engellenmektedir.


```
$ docker container run -P -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
214ca5fb9032: Pull complete
66eec13bb714: Pull complete
17cb812420e3: Pull complete
56fbf79cae7a: Pull complete
c4547ad15a20: Pull complete
d31373136b98: Pull complete
Digest: sha256:2d17cc4981bf1e22a87ef3b3dd20fbb72c3868738e3f307662eb40e2630d4320
Status: Downloaded newer image for nginx:latest
e0a08575bbd87def175b0af41f68561a9669782111ae1147c9aed7d9225b0b2e
```

docker container run -P

- -P opsiyonu, konteynerde kullanılan 80 numaralı TCP portunu, aynı zamanda konağa da başka bir port üzerinde bağlar.
- «**docker ps**» komutuyla, 80 numaralı portun hangi konak (host) portuna bağlandığı görülebilir.
- Aşağıdaki örnekte, 49154 numaralı host portunun, konteynerin 80 numaralı portuna bağlandığı görülebilir.

```
$ docker ps
```

CONTAINER ID	IMAGE	STATUS	PORTS	NAMES
e0a08575bbd8	nginx	Up 51 seconds	0.0.0.0:49154->80/tcp, :::49154->80/tcp	jovial_shtern
7883afb4e39c	ubuntu	40 minutes ago	Up 40 minutes	sunucu6
061fdaf613d8	alpine	About an hour ago	Up About an hour	sunucu5
12aefa40b82a	ubuntu	About an hour ago	Up About an hour	sunucu4
d277bf460276	alpine	3 hours ago	Up 3 hours	sunucu3



nginx web sunucusuna bağlanma

- **nginx** konteynerinin hangi IP'de çalıştığını ve port bilgilerini görmek için «**docker inspect <ID>**» komutu kullanılabilir. Konteyner ID değeri «**docker ps**» komutundan öğrenilebilir.

```
"NetworkSettings": {  
  "Bridge": "",  
  "SandboxID": "536d7fe26329bd219f3225cc983d8de216d96accc4763f943",  
  "HairpinMode": false,  
  "LinkLocalIPv6Address": "",  
  "LinkLocalIPv6PrefixLen": 0,  
  "Ports": {  
    "80/tcp": [  
      {  
        "HostIp": "0.0.0.0",  
        "HostPort": "49154"  
      },  
      {  
        "HostIp": ":::",  
        "HostPort": "49154"  
      }  
    ]  
  },  
  "SandboxKey": "/var/run/docker/netns/536d7fe26329",  
  "SecondaryIPAddresses": null,  
  "SecondaryIPv6Addresses": null,  
  "EndpointID": "00d7c72234b39c6418f98148928ec2ef2b6a165201e52b08",  
  "Gateway": "172.17.0.1",  
  "GlobalIPv6Address": "",  
  "GlobalIPv6PrefixLen": 0,  
  "IPAddress": "172.17.0.5",  
}
```

Konteynerin 80
nolu portu
konakta 49154
olarak görülecek.

0.0.0.0 adresi, bu sunucuya
herhangi bir IP adresinden
gelinebileceğini ve bağlantının
kabul edileceğini belirtir.

curl ile web sitesine ulaşım

- **curl** komutu, komut satırından web sitesine bağlanarak bilgi almak için kullanılan bir programdır.
- IP numarasını öğrendikten sonra **curl http://IP/** komutuyla web sayfası indirilebilir:

```
$ curl http://172.17.0.5/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```


curl ile web sitesine ulaşım

- **-P** komutuyla, konteynerdeki 80 nolu port konaktaki (host) bir porta bağlanmıştır.
- Eğer istenirse, «**curl http://127.0.0.1:portno/**» yazılarak da HTML sayfası görüntülenebilir.

```
$ curl http://127.0.0.1:49154
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

konteyner logları

- Konteyner çalışırken STDOUT'a yazılan loglar üretir. STDOUT'a yazılan logları Docker kaydeder.
- Konteyner'in ürettiği logların görülebilmesi için «**docker logs <konteyner-ID>**» komutu verilebilir.

```
$ docker container logs e0a08575bbd8
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/05/26 11:16:38 [notice] 1#1: using the "epoll" event method
2022/05/26 11:16:38 [notice] 1#1: nginx/1.21.6
2022/05/26 11:16:38 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/05/26 11:16:38 [notice] 1#1: OS: Linux 5.13.0-44-generic
2022/05/26 11:16:38 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/05/26 11:16:38 [notice] 1#1: start worker processes
2022/05/26 11:16:38 [notice] 1#1: start worker process 31
2022/05/26 11:16:38 [notice] 1#1: start worker process 32
172.17.0.1 - - [26/May/2022:11:30:49 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
172.17.0.1 - - [26/May/2022:11:33:37 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
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" "-"
```

Konteyner hakkında bilgiler

- Konteynerlerin ne kadar kaynak harcadıkları bazen test yapılırken önemli olabilir.
- «**docker container stats**» komutuna konteynerin ID'si verilerek çağrılırsa, CPU ve bellek kullanımı anlık görülebilir. Ctrl-C ile çıkılır.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
7883afb4e39c	sunucu6	0.00%	1.496MiB / 7.73GiB	0.02%	9.14kB / 0B	0B / 0B	1
061fdaf613d8	sunucu5	0.00%	1.062MiB / 7.73GiB	0.01%	4.78kB / 0B	0B / 0B	1
12aefa40b82a	sunucu4	0.00%	1.496MiB / 7.73GiB	0.02%	4.78kB / 0B	0B / 0B	1
d277bf460276	sunucu3	0.00%	1.07MiB / 7.73GiB	0.01%	6.44kB / 280B	0B / 0B	1

- Konteynerlerde hangi proseslerin çalıştığı bazen incelenmesi gereken konulardan biridir.
- «**docker container top**» komutuna konteynerin ID'si verilerek çağrılırsa, CPU ve bellek kullanımı anlık olarak görülebilir.

\$ docker container top e0a08575bbd8							
UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	11920	11898	1	14:42	?	00:00:00	nginx: master process nginx -g daemon off;
systemd+	11963	11920	0	14:42	?	00:00:00	nginx: worker process
systemd+	11964	11920	0	14:42	?	00:00:00	nginx: worker process

Konteyner içinde program çalıştırma

- Konteyner içinde belirli bir programın çalıştırması gerekebilir.
- Bu amaçla aşağıdaki komut kullanılabilir:
`docker exec -it ID komut`
- Örneğin, **nginx** konteynerinde ana HTML sayfası görüntülenmek istendiğinde aşağıdaki komut kullanılabilir:

`docker exec -it ID cat /usr/share/nginx/html/index.html`

```
$ docker exec -it e0a08575bbd8 cat /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Konteynere dosya kopyalama

- Konteyner içindeki belirli bir dosyanın değiştirilmesi için belirli bir programın çalıştırması gerekebilir. Yeni **index.html**'yi konakta üretelim.
- Bu amaçla aşağıdaki komut kullanılabilir:
`docker container cp yeni.html IMAGEID:/usr/share/nginx/html/index.html`
- Curl programıyla bağlanarak güncellenen **index.html** görülebilir.


```
$ docker container cp index.html e0a08575bbd8:/usr/share/nginx/html/index.html
$ curl http://172.17.0.5/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>YENİ SAYFA -- YENİ SAYFA</h1>
<p>BU SAYFA DEGISTI</p>

<p><em>BU SAYFA DEGISTI.</em></p>
</body>
</html>
```

konteyner TCP portları

- Konteynerlerde güvenlik nedeniyle aksi belirtilmediği sürece bütün TCP portları kapalıdır.
- **nginx** konteynerinde, **nginx** sunucusu çalıştığı için 80 numaralı portun açıktır ve imaj tanımında bu bulunmaktadır. «**docker image inspect nginx**» yazıldığında, «**Expose Ports**» ifadesinde 80 numaralı portun açık olduğu görülebilir.
- İmajdan konteyner oluşturulduğunda, otomatik olarak konteynerde TCP 80 portu erişilebilir hale getirilir.

```
"Config": {  
  "Hostname": "",  
  "Domainname": "",  
  "User": "",  
  "AttachStdin": false,  
  "AttachStdout": false,  
  "AttachStderr": false,  
  "ExposedPorts": {  
    "80/tcp": {}  
  },  
  "Tty": false,  
  "OpenStdin": false,  
  "StdinOnce": false,  
}
```



konteyner TCP portları

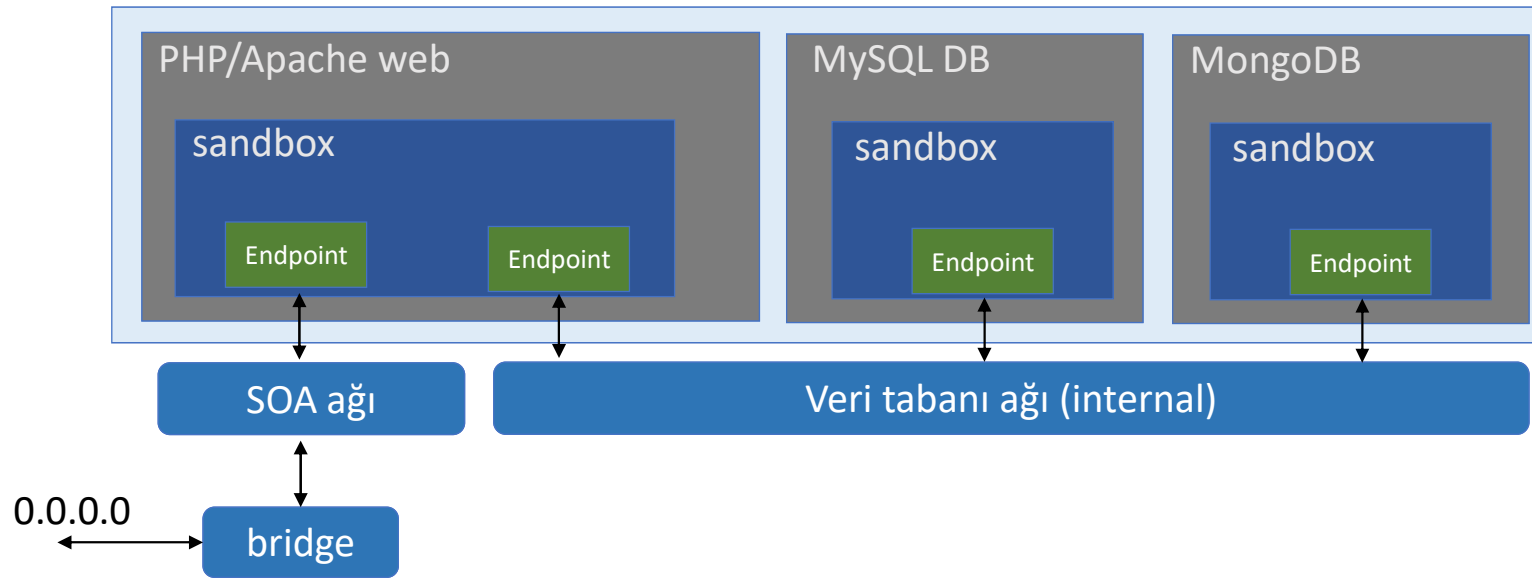
- Konteyner çalışırken, istenen bir TCP portu «**docker run -d --expose 5400 nginx**» yazıldığında, otomatik açılan 80 numaralı portun yanında 5400 portu da açılır. Hangi portların açık olduğu, «**docker ps -a**» komutuyla görülebilir.
- Eğer konteynerdeki 80 numaralı TCP portunun konakta (host) 8080 olarak kullanılabilmesi isteniyorsa, aşağıdaki komut kullanılmalıdır:
`docker container run -d -p 8080:80/tcp nginx`
- Birden fazla port «**-p**» opsiyonlarıyla ayrı ayrı belirtilerek host üzerindeki port numaraları belirlenerek açılabilir.
- «**docker container port ID**» komutuyla, konteynerde hangi port eşlemelerin yapıldığı görülebilir.

```
$ docker container port e0a08575bbd8  
80/tcp -> 0.0.0.0:49155  
80/tcp -> :::49155
```

Docker ÖRNEK
dahili (internal) ağ

Dış ağa bağlı olmayan ağ oluşturmak

- Dış ağa bağlı olmayan konteynerler güvenlik düşüncesiyle tercih edilen yapılardır.
- Örneğin veritabanı yönetim yazılımları bulunan konteynerleri dış ağa bağlamamak ve konak üzerindeki ağda tutmak tercih edilen durumlardan biridir.
- Eğer veritabanı sunucularına sadece web sunucusu ulaşıyorsa, web sunucusu hem iç ağa hem de dış ağa bağlamak, güvenli bir sistem oluşturulmasına katkıda bulunur.



Dış ağa bağlı olmayan ağ oluşturmak

- Fiziksel sunucu/ağ sistemlerinde, veritabanı sunucularının (veya dışarıya açılması istenmeyen sunucuların), dış ağdan yalıtılması için farklı yöntemler kullanılmaktadır.
 - İzole edilecek fiziksel sunucuları farklı bir ağ cihazına bağlayarak, firewall'un bir bacağına bağlayarak, IP veya port bazında firewall cihazı üzerinde yazılacak kurallarla yalıtmak
 - Fiziksel ağ cihazlarında VLAN tanımı yaparak, aynı fiziksel ağ anahtarına bağlansalar da MAC adresi bazında bunları farklı ağlarda göstermek
 - Web sunucusunun ikinci bir Ethernet ucu varsa, veritabanı sunucularına bağlamak
 - vb

Dış ağa bağlı olmayan ağ oluşturmak

- Docker mimarisinde, bütün konteynerlerin aynı konak üzerinde çalıştığı ve Docker çalıştırma ortamının sanal ağları işlettiği unutulmamalıdır.
- Bazı konteynerlere dışarıdan erişimin engellenmesi için «**bridge**» sürücüsünün kullanılacağı ve dahili (internal) nitelikte olacak bir ağ oluşturulması gerekebilir:

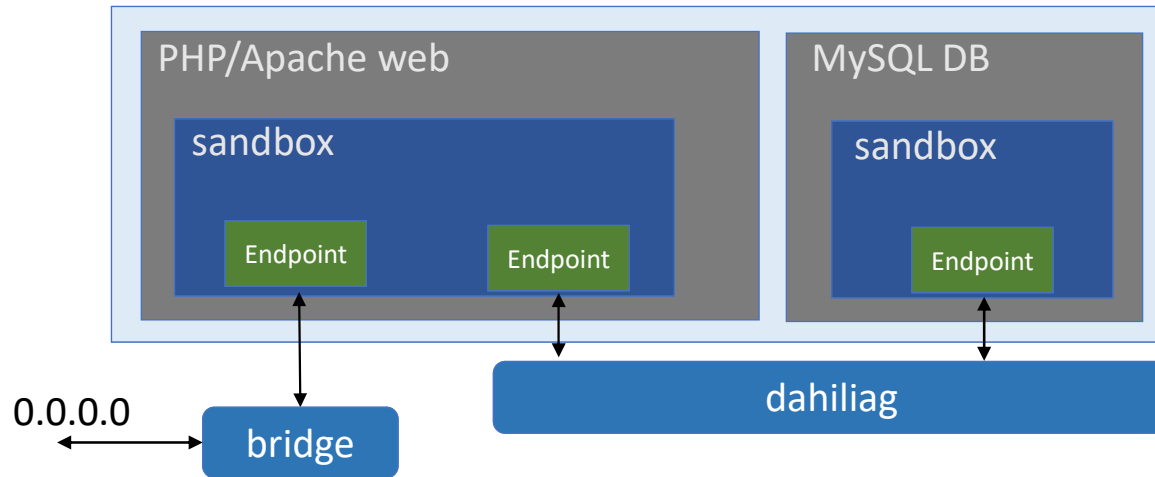
```
docker network create -d bridge --internal dahiliag
```

- Bu şekilde oluşturulacak «**dahiliag**» isimli ağa MySQL v5.7 sunucusu ekleyelim.

```
docker container run -d --name mysql_db -e  
MYSQL_ROOT_PASSWORD=9sifresifre1 --network dahiliag  
mysql:5.7
```

ÖDEV

- Üzerinde çalıştığımız **PHP/Apache** konteynerini, hem dış ağda hem de **dahiliag**'a bağlayın.
- MySQL konteynerini **dahiliag**'a «**connect**» edin. Eğer dış ağa bağlıysa bunu «**disconnect**» edin.



Docker depolama/disk

Konteynerlerde Depolama

- Konteynerlerde iki çeşit depolama bulunur
 - Kalıcı olmayan depolama: Lokal depolama olarak bilinir ve konteynerde kök dizini ve altındaki dizinlerde bulunan dosyalardır. Konteynerle birlikte gelir ve konteyner oluşturulduğunda bu kalıcı olmayan depolama alanı da yeniden oluşturulur.
 - Kalıcı depolama: sabit disk alanlarında bulunan ve konteynerlerin dışarıdan erişebildikleri alanlardır.

<https://docs.docker.com/storage/>

«**mount**» tipleri

- Docker dört tür «**mount**» tipini desteklemektedir.
 - «**volume**» **mount**:
 - «**bind**» **mount** işlevine göre daha çok özelliği vardır.
 - Docker tarafından yönetilir ve başka proseslerin buradaki dosyaları değiştirmesine izin verilmez.
 - Birden fazla konteyner aynı «**volume**»u kullanabilir.
 - Bulut veya başka bir konaktaki dizin **volume** olarak kullanılabilir.
 - «**bind**» **mount**:
 - Konaktaki bir dizinin konteynerle paylaşılması esasına dayanır.
 - Konak makinenin dosya sistemini kullanılır.
 - «**tmpfs**»: Konteynerin yaşam döngüsü boyunca kullandığı ve kalıcı olmayan verinin tutulduğu alandır.
 - «**named pipe**»: Docker konak ve konteyner arasındaki iletişim için kullanılır. Örneğin, bir konteyner içindeki üçüncü parti bir aracın «**named pipe**» üzerinden Docker API tarafından bağlanması.

«**volume mount**»

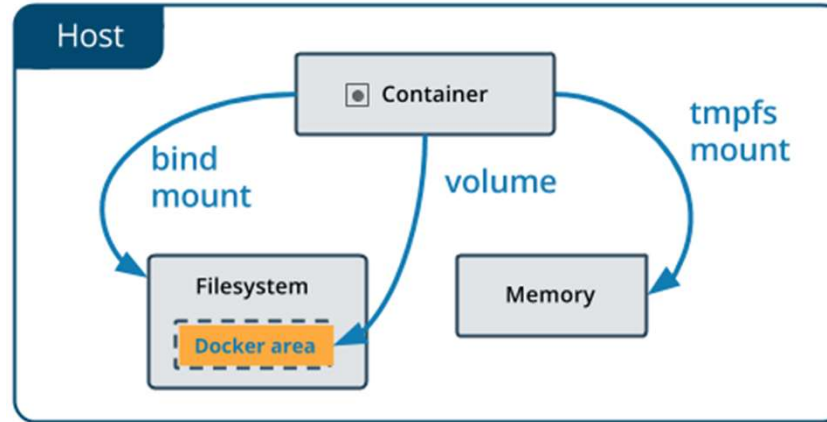
- **volume mount**: Docker tarafından oluşturulur ve yönetilir.
 - «**volume**» oluşturulduktan sonra bir dizinde veya Docker konakta depolanır.
 - Bir konteynere «**volume mount**» yapıldığında, bu dizin konteynere kurulur.
 - «**volume**»ların özelliği bunların konaktan izole edilmesidir.
 - Bir «**volume**» birden fazla konteynere yerleştirilebilir ve birbirleriyle haberleşme için kullanılabilir.
 - Belirli bir «**volume**»u kullanan çalışan bir konteyner bulunmasa da Docker'da kullanılabilir durumdadır.
 - «**volume**» sürücüsü (driver) tarafından yönetildiğinde, «**volume**»lar bulutta veya başka bir konakta tutulabilir veya şifreli bir şekilde depolanabilir.

«**bind mount**»

- «**bind mount**», Docker'ın ilk yıllarından beri bulunmaktadır.
- Lokal bir dizinin veya dosyanın, konteynerdeki bir dizinle veya dosya ile eşlenmesini gerçekleştirir.
- Örneğin, konak tarafında **/home/adminpc/dizin** isimli bir dizin konteynerdeki **/etc/deneme** dizini olarak gösterilebilir.
- Bu dizine konteyner tarafından veya konak tarafından eklenen herhangi bir dosya her iki tarafta da görülebilir halde olacaktır.
- Konteynere dosya aktarmak (örneğin yapılandırma dosyaları) için kullanılan yöntemlerden biridir.

★ Hangisi? «**bind**» mı «**volume**» mu?

- Depolama alanının Docker tarafından yönetilmesini ve buna sadece konteynerler tarafından erişilmesini istiyorsanız «**volume**» türünü seçmelisiniz. Eğer veriler farklı Docker konaklarda veya bulutta bulunuyorsa, «**volume**» en iyi çözümdür. Ayrıca «**volume**» veri yedeklenebilir (back-up) veya diğer bir Docker hosta aktarılabilir (migration).
- Eğer depolama alanının yani dizinin diğer prosesler tarafından değiştirilebilmesini isterseniz veya konaktaki dizin üzerinden konteynerlere dosya transferi yapmak amaçlıyorsanız «**bind**» türü en doğru seçimdir. Örneğin **nginx** konteynerinde, **nginx** konfigürasyon dosyasını konteynerde kullanmak isterseniz, «**bind**» hızlı bir çözüm sunar.



<https://docs.docker.com/storage/#good-use-cases-for-volumes>

<https://docs.docker.com/storage/#good-use-cases-for-bind-mounts>

«**bind**» ve «**mount**»

- Docker'da konteyner çalıştırılırken verilen parametrelerden biri «**--mount**» parametresidir.
- Konteyner çalıştırılırken verilen «**mount**» parametresinde «**type**» olarak bind belirtilirse, «**source**» konaktaki dizini ve «**target**» da konteynerdeki dizini belirtmektedir:

```
docker container run -d --mount type=bind,  
source=/home/adminpc/dizin,target=/etc/nginx
```

```
adminpc@ubuntu:~$ mkdir dizin  
adminpc@ubuntu:~$ docker container run -d --name deneme1 --mount type=bind,source=/home/adminpc/diz  
in,target=/etc/deneme nginx  
89fc95b865be7f0ce90766e0da813fe54e08cd5d747469f9db7d09926dc60c05
```

«bind» ve «mount»

- Konteynerde yapılan «bind mount» işleminin tanımı, «docker container inspect ID» komutuyla görüntülenen yapılandırma bilgileri arasında görülebilir.

`docker container inspect ID`

```
"Mounts": [  
  {  
    "Type": "bind",  
    "Source": "/home/adminpc/dizin",  
    "Destination": "/etc/deneme",  
    "Mode": "",  
    "RW": true,  
    "Propagation": "rprivate"  
  },  
]
```

«bind» ve «mount»

- Konteyner ve konak tarafından paylaşılan dizinlerde yapılan değişiklikler, dosya eklemeleri, dosyalardaki değişiklikler, her iki tarafta da görülebilir.
 - `/home/adminpc/dizin/de config.txt` diye `nano` programıyla yeni bir metin dosyası oluşturalım ve içine bir şeyler yazalım ve kaydedelim.
 - «`docker container exec -it deneme1 /bin/bash`» komutuyla, daha önce çalıştırarak ismini `deneme1` koyduğumuz konteynerimizde `/bin/bash` kabuğu çalıştıralım ve kabuğa bağlanalım ve `/etc/deneme/config.txt` dosyasını görüntüleyelim.

```
adminpc@ubuntu:~$ nano dizin/config.txt
adminpc@ubuntu:~$ docker container exec -it deneme1 /bin/bash
root@89fc95b865be:/# cat /etc/deneme/config.txt
name aaaa
surname bbbb
root@89fc95b865be:/#
```

«**volume mount**»

- «**volume**»ların «**bind**» türünden temel farklılığı, «**volume**»ların Docker tarafından yönetilmesi ve **volume**'un **/var/docker/volumes** dizininde tutulmasıdır.
- Birden fazla konteyner tarafından kullanılmasının daha güvenli olduğu söylenmektedir.
- Bir volume oluşturulabilmesi için «**docker volume create**» veya «**docker container run**» komutunun kullanılması gerekir.
- Eğer belirtilen dizin yoksa, Docker bunu oluşturacaktır.

```
docker container run -d --name nginxvol3 -v  
/home/adminpc/htmldir:/usr/share/nginx/html nginx
```

«volume mount»

- «volume»lar «**docker volume create**» komutuyla da oluşturulabilir (**/var/lib/docker/volumes** altında):
docker volume create htmlvol
- Konteyner çalıştırılırken daha önce oluşturulan **volume**'la ilişkilendirilebilir.
docker container run -d --name nginxvol3 --mount type=volume,htmlvol:/usr/share/nginx/html nginx
- Volume ile ilgili bilgi istenirse, «**docker volume inspect htmlvol**» komutu verilebilir.

nginx HTML dosyası değişikliği

- Bu şekilde **htmlvol**'un altındaki dosyalarda **nginx**'in kullandığı HTML dosyaları da değiştirilebilir. **htmlvol**'un içindeki **nginx** ana HTML dosyasının yeri:

`/var/lib/docker/volumes/htmlvol/_data/index.html`

- Burada dosya «**sudo nano**» ile değiştirilebilir.

```
adminpc@ubuntu:~$ sudo docker volume create htmlvol^C
adminpc@ubuntu:~$ sudo docker container run -d --name htmltry1 -v htmlvol:/usr/share/nginx/html nginx
1d4881083c1ab31733165ce1ba3ec2744701bc910b97200e4f10ca601811e04b
adminpc@ubuntu:~$ sudo ls /var/lib/docker/volumes
backingFsBlockDev  deneme  htmlvol  metadata.db
adminpc@ubuntu:~$ sudo ls /var/lib/docker/volumes/htmlvol
_data
adminpc@ubuntu:~$ sudo ls /var/lib/docker/volumes/htmlvol/_data
50x.html  index.html
adminpc@ubuntu:~$ sudo nano /var/lib/docker/volumes/htmlvol/_data/index.html
```

```

!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>DEGDEGDEG</h1>

```

Read 18 lines

^G Get Help	^O Write Out	^W Where Is	^K Cut Text	^J Justify	^C Cur Pos
^X Exit	^R Read File	^_ Replace	^U Paste Text	^T To Spell	^_ Go To Line

nginx HTML dosyası değişikliği

- Değiştirilen HTML sayfasının görüntülenmesi için, «**docker inspect**» ile konteynerin IP'si bulunur ve bu IP için «**curl http://IPNO/**» yazılarak, değiştirilen web sayfasının görüntülenmesi sağlanabilir.

```
adminpc@ubuntu:~$ curl http://172.17.0.2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>DEGDEGDEG</h1>

<p>HELLO</p>

<p><em>Thank you.</em></p>
</body>
</html>
```

Ne yaptık?

- Konteynerde **nginx** web sunucusunun kullandığı HTML dizini için bir «**volume**» oluşturduk.
- Bu şekilde, **nginx** sunucusundaki HTML sayfasını, konakta «**/var/lib/docker/volumes/htmlvol/_data**» dizini üzerinden değiştirebildik.
- Değiştirdiğimiz HTML sayfasının, **nginx** konteyneri tarafından sunulduğunu **curl** programıyla gördük.

«**ephemeral**» kavramı

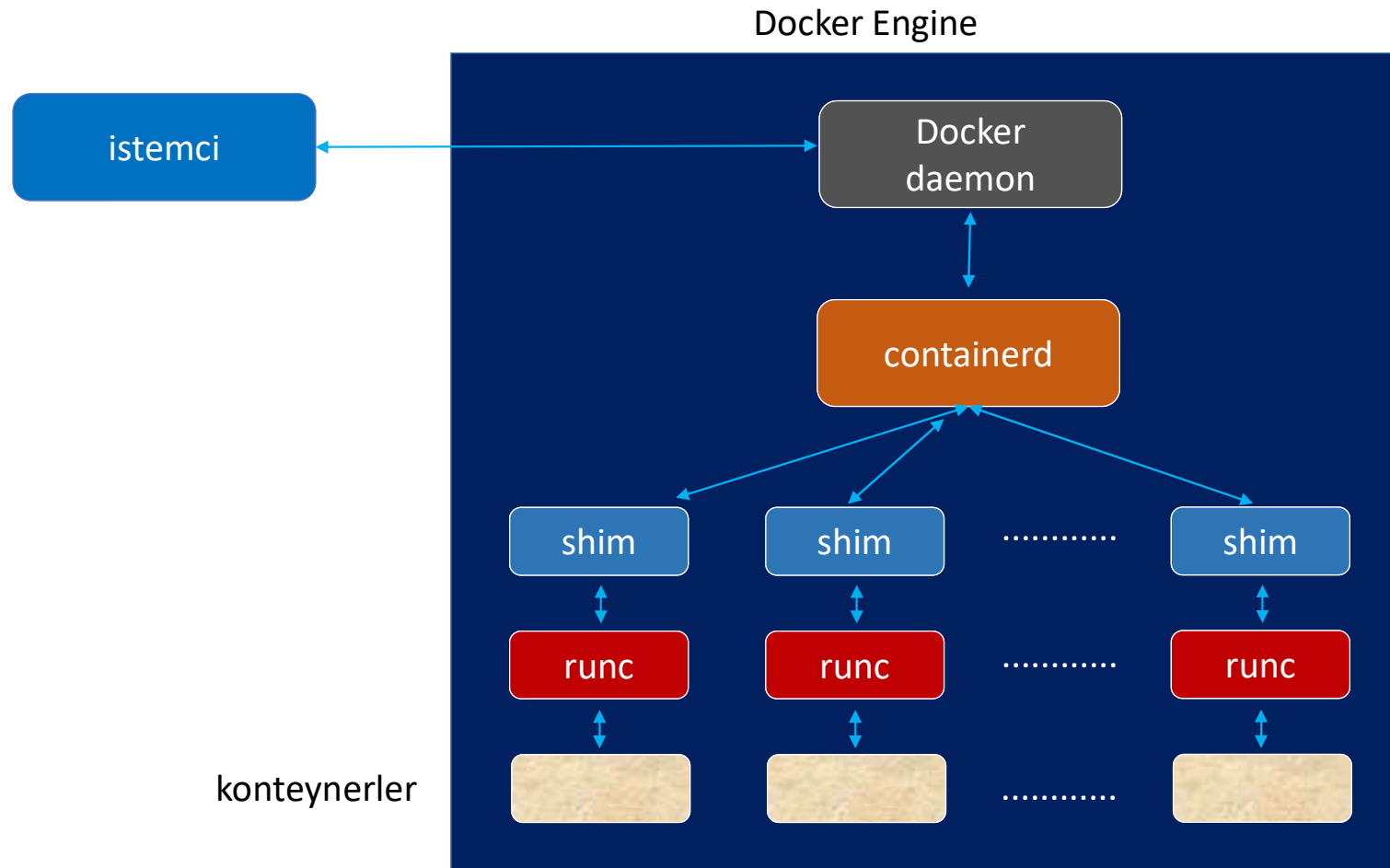
- Bir konteyner içinde yer alan «**read-only**» olmayan katmanlar (**OverlayFS** içindeki) konteyner kapandıktan sonra varlığını sürdüremez.
- Konteyner çalışırken oluşturulan veya içine veri yazılan dosyalar, konteyner çalışmasını durdurduktan sonra kaybolur
- Konteynerlerin (podların) «**ephemeral**» olması, Kubernetes'in de temel çalışma ilkesini oluşturur.
- Ancak bir «**volume**» oluşturularak, dosyalar burada oluşturulursa, oluşturulan dosyalar «**volume**» ortadan kaldırılmadığı sürece kalıcı olarak depolanabilir.
- Konteynerlerin «**ephemeral**» oluşu, konteynerler içinde veritabanı yönetim sistemleri çalıştırıldığı zaman sorun yaratır. «**volume**» içine kaydedilmeyen veritabanı dosyaları, konteyner kapandığında veya bozulduğunda kaybolabilir.

Docker mimarisi

Docker Nasıl Çalışır?

- Komut satırında çalışan «**docker**» programı, REST API kullanarak Docker daemon'a komutları gönderir.
- Docker daemon komutları alır.
- Docker daemon «**containerd**»yi çalıştırarak yeni bir konteyner çalıştırır.
- Docker daemon gRPC (Google'un geliştirdiği RPC – Remote Procedure Call) kullanır ve **containerd**'ye komutları iletir.
- **containerd** imajdan çalıştırılabilir bir konteyner üretir ve **runc**'yi çalıştırır (fork) ve konteyneri çalıştırması için iletir.
- **runc** Linux çekirdeğindeki özellikleri (**namespaces**, **cgroups**, vb) kullanarak konteyneri yapılandırır ve bir proses olarak çalıştırmaya başlar. **runc** OCI'nin koyduğu spesifikasyona uyar ve konteynerleri çalışmak üzere hazırlar.
- Konteyner sonlandığında, **runc** de sonlanır.

Docker Mimarisi - basit



containerd

- Docker, sistemindeki konteyner çalışma ortamını ayırarak, **containerd** adı altında yeni bir proje olarak endüstriye sundu.
- Proje dahilinde, Docker konteynerlerinin nasıl çalıştırılacağı, düşük seviyede depolamaya nasıl erişileceği, imajların nasıl transfer edileceği gibi çözümler bulunuyordu
- Proje başlatıldıktan bir süre sonra, Docker projeyi, endüstrinin kullanımına açmak ve daha da geliştirilmesini sağlamak amacıyla, CNCF vakfına (Cloud Native Computing Foundation) devretmiştir.
- **containerd**, Kubernetes gibi «orquestrasyon» sistemlerinin doğrudan Docker'ı kullanmak yerine, konteynerlerdeki düşük seviyeli Docker öğelerine erişmesini ve daha etkin bir şekilde yapılandırarak çalıştırmasını kolaylaştırmıştır.
- **containerd**, konteynerlerin yaşam döngüsünü yönetir. Başlamasını, durmasını, duraklamasını ve silinmesini gerçekleştirir.
- İmajların «**push**» ve «**pull**» işlemlerini gerçekleştirir.

```
docker run -it <imajNO>
```

- Yukarıdaki komutu verdiğimiz zaman ne oluyor?
 - docker komut satırı programı REST API üzerinden ne yapılacağını Docker daemon'a aktarır.
 - Docker daemon komutları alır ve **containerd**'yi çalıştırarak yeni bir konteyner oluşturmasını ister.
 - **containerd** OCI'nin standartlarında tarif edilen bir veri yapısı (JSON formatında) hazırlar. Konteynerin çalışması için gerekli her türlü önemli bilgi burada yer alır.
 - **runc** prosesine bu OCI verisini iletir.
 - **runc** işletim sistemi çekirdeğindeki **namespaces** ve **cgroups** özelliklerini kullanarak konteyneri (çalışacak programlar, kök dizin yapısı, ağ arayüzü, IP numaraları, vb) oluşturur. Konteyner prosesi başlatılır ve **runc** işini bitirir ve çıkar. **shim** prosesi artık ebeveyn prosesi olur.

shim

- **shim, containerd**'nin başlattığı alt proseslerden biridir
 - containerd kullanıcıların yapılandırma yapıp yeniden başlattıkları bir proses. shim'i çalıştırmasının sebebi, konteynerlerin containerd yeniden başlatılsa da çalışmasının sağlanması.
 - **shim** olmadan konteyner çalışmaz.
- Containerd, **runc** prosesini çalıştırdıktan sonra konteyner oluşturulunca **runc** çıkış yapar ve **shim** konteyner prosesinin ebeveyn prosesi haline gelir (konteyneri çalıştıran **shim** prosesidir).
- Bu durumda çok sayıda konteyner çalıştıran sistemlerde **runc** prosesinin çok sayıda kopyası olmamasını sağlar.
- **shim**, konteynerin STDIN ve STDOUT gibi akımların idaresini eline alır ve onları açık tutar. Eğer kullanıcı tarafından **containerd** yeniden başlatılırsa, çalışan konteynerler **shim** üzerinden çalışmalarını sekteye uğramadan yürütürler.
- **shim** ayrıca konteynerin çalışma durumunu, çıkış yapıp yapmadığını Docker daemon'a iletir.

Güvenlik Sorunu

- Konteynerler, Linux çekirdeğindeki bazı özellikleri kullanarak çalıştırılmaktadır. Çekirdekte olabilecek hatalar ve sorunlar, konteynerlerin çalıştırılmasında da risk yaratabilmektedir.
- Diğer yandan, konteyner çalışma ortamlarında (mesela containerd) olan sorunlar da aynı şekilde risk yaratabilir.
- 2020'de **containerd-shim** API'sinin bir güvenlik açığı olduğu duyuruldu.
- Konteynerler, **containerd** ve **containerd-shim** prosesleri tarafından idare edilir.
- Docker komutunun çalıştırılması sırasında «**--net=host**» opsiyonunun kullanılması konteynerin **shim** API üzerinden root yetkilerine sahip olabildiği tespit edilmiştir.
- SORUN KISA BİR SÜRE İÇİNDE ÇÖZÜLDÜ AMA BU TÜR PROBLEMLERİN GELECEKTE DE ÇIKMASI OLASIDIR.

<https://blog.aquasec.com/cve-2020-15257-containerd-shim-api-vulnerability>

Docker içinde Docker çalışır mı?

- Elimizde ne var bakalım:
 - Docker, işletim sistemi seviyesinde bir sanallaştırma uygulamasıdır.
 - Çalışan konteyner, bir proses olarak sadece belirli dosyalara erişme ve çalıştırma izni vardır. Konteyner bir proses olarak **jailroot**, **cgroups**, **cpusets**, **namespaces** özellikleri sayesinde izole bir ortamda çalıştırılır.
- Docker konteyneri içinde Docker çalışabilmesi mümkündür. Ancak ilk başlarda önerilmemekteydi:
 - <https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/>
- Ancak, Petazzoni, 2020'de fikrini değiştirdi ve Docker içinde Docker çalışabilir dedi. Şu anda mümkündür. Örneğin K3D, minikube, Rancher, vb sistemler bu Docker içinde Docker konteynerleri olarak çalışır.

Nginx – «load balancer» ve «Reverse Proxy» uygulamaları

«Load Balancer» nedir?

- İstemcilerden gelen RESTful API veya web isteklerini, iç taraftaki web sunucularına ileten ve havuzda bulunan bu sunucular arasında yük paylaşımı yapmasına «load balancing» denir.
- Yük paylaşımı dışında bir de web sunucularını ağ üzerinden doğrudan bağlantı kurmasını engelleyerek güvenliği sağlar.
- Ayrıca, havuzda yer alan bir web sunucusunun güncellenmesinden önce yük paylaşımcısında yapılan yapılandırma değişikliğiyle verilen hizmetin kesintiye uğramasının önüne geçilebilir.
- İstemcilerin SSL ile sunuculara bağlandığı bir ortamda, LB cihazı, SSL ile gelen talepleri karşılayabilir, SSL sonlandırıcı olarak çalışarak iç tarafta sunuculara SSL'siz şekilde talepleri iletebilir.

- devam

Dockerfile ile amaca özel imaj oluşturma

«**Dockerfile**» nedir?

- «**Dockerfile**», belirli spesifikasyona sahip bir Docker imajını oluşturmak için kullanılan komutları içeren bir dosyadır.
- Bilindiği gibi, imajlar salt-okunabilir katmanlardan oluşur.
- Üst üste yığılı katmanların her biri bir «**Dockerfile**» komutuna karşılık gelir ve her komut bir önceki katmanın üzerinde gerçekleştirilerek yeni bir katman oluşur.
- İmajlar, «**docker build**» komutuyla oluşturulur ve birbiri ardına yürütülen komut satırlarıyla kullanıcılar otomatik olarak imajı oluştururlar.
- Docker, imaj oluştururken «**FROM**», «**COPY**», «**RUN**», «**ENV**», «**EXPOSE**», «**CMD**» gibi komutların çalıştırılmasını sağlar.

«**Dockerfile**»

- «**Dockerfile**»'da kullanılan komutların nasıl çalıştığını öğrenmek, imaj oluşturmak için yeterli değildir.
- Eğer imajınızda bir yazılım bulunuyorsa, bunun bir sunucu sistemde nasıl kurulacağını önceden bilmeniz gereklidir. Örneğin Apache üzerinde PHP bir sistem kurmak için daha önce bu tür bir PHP kurulumunu yapmış olmalısınız.
- Ancak, **Dockerfile** okunması, değiştirilmesi ve uyarlanması kolay bir araç olduğundan, kullanılabilecek çok sayıda örnek çeşitli web sitelerinde bulunabilir.
- Bir imajın oluşturulması uzun zaman almaz ve farklı denemeler için zaman kalır.

«Dockerfile»

- İmajı, komut satırında kullandığınız «**docker**» programı oluşturmamaktadır. İmajın kurulması, Docker daemon tarafından gerçekleştirilir.
- Yapılması gereken ilk iş, boş bir dizin oluşturmak ve içine «**Dockerfile**» ismindeki imajın oluşturulması için gerekli komutları içeren metin dosyasını koymaktır. Dosya içinde tarif edilen ve imajın oluşturulmasında kullanılacak veya eklenecek dosyalar varsa, bunları da bu dizin içinde tutmak gereklidir.
- **Dockerfile**'ın içinde bulunduğu dizinde «**docker build**» komutunu kullanmakla birlikte, eğer istenirse dosya sisteminde belirli bir yerde bulunan **Dockerfile** da çalıştırılabilir:

```
$ docker build -f /dosya/dizini
```
- Oluşturulacak imajın etiketlenmesi (isimlendirilmesi) için **-t** opsiyonu kullanılabilir. Bu şekilde oluşturulacak imajın versiyonu da belirlenebilir:

```
$ docker build -t kafkasunucu2:1.0.1 .
```

Örnek «Dockerfile»

«apt-get update»
komutundan sonra
apache2 yükle.

Docker Hub'dan
ubuntu imajını al ve
baz olarak kullan.

```
FROM ubuntu:latest  
RUN apt-get update && apt-get install -y apache2  
EXPOSE 80 443  
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]  
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

80 ve 443 nolu TCP
portları açılacak

Bu dizinler için
sunucuda VOLUME'lar
oluşturulmalı

«docker build»

- «**docker build**» komutuyla **Dockerfile** işlendiğinde, ilk olarak dosyanın formatında hata olup olmadığı kontrol edilir.
- Eğer, dosyada herhangi bir sorun yoksa, **Dockerfile**'da tarif edilen imaj oluşturulur.

```
adminpc@ubuntu:~/deneme$ docker build -t kafkasunucu2:1.0.1 .
Sending build context to Docker daemon 6.482MB
Step 1/7 : FROM node:12-alpine
---> 9756efdda869
Step 2/7 : RUN apk add --no-cache python2 g++ make
---> Using cache
---> b33da50d1e67
Step 3/7 : WORKDIR /app
---> Using cache
---> dc0df1785b05
Step 4/7 : COPY . .
---> Using cache
---> ee03abc6ef5f
Step 5/7 : RUN yarn install --production
---> Using cache
---> 6a6829185518
Step 6/7 : CMD ["node", "src/index.js"]
---> Using cache
---> cae1199b5fcc
Step 7/7 : EXPOSE 3000
---> Using cache
---> bfb0141b9595
Successfully built bfb0141b9595
Successfully tagged kafkasunucu2:1.0.1
```

Moby Buildkit

- Docker'ın 18.09 sürümünden sonra, imajın oluşturulması için moby/buildkit projesi kullanılmaya başlanmıştır. Bu proje sayesinde
 - Oluşturma sırasında kullanılmayan aşamaların tespiti ve uygulanmaması
 - Oluşturma süreçlerinin paralel halde yürütülmesi
 - Farklı zamanlarda yapılan imaj oluşturma işlerinde, önceki sonuçların kullanılması (yeni oluşturmada sadece değişiklik yapılan dosyaların kullanılması, vb)
 - Kullanılmayan dosyaların tespit edilmesi
 - Yeni parametre eklenmesi
 - v.b.

özellikler sağlanmıştır.

<https://github.com/moby/buildkit>

«**docker buildx**»

- «**Docker Buildx**», Moby Buildkit'in işlevlerini tam olarak destekleyen komut satırı eklentisidir.
- «**docker build**» komutuna benzemekle birlikte, ek özellikleri de bulunmaktadır. Örneğin farklı platformlar için (amd64, linux/arm64, linux/amd64, vb) imaj oluşturabilmektedir.

<https://docs.docker.com/build/buildx/>

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

Dockerfile - katmanlar

- Docker dosyasında komutlar, imaja katman olarak eklenmektedir.
- Komut formatı şöyledir:

```
# comment
```

```
KOMUT argümanlar
```

- Katman ekleyen komutlar şunlardır:
 - **FROM** : Docker Hub'da bulunan bir imajı baz imaj olarak belirler. Bütün Dockerfile dosyaları bununla başlamalıdır.
 - Örnek: **FROM ubuntu:18.04**
 - **COPY**: Docker'ın istemcisindeki dizindeki dosyaların imaja aktarılmasını sağlar
 - Örnek: **COPY . .**
 - **RUN**: Bir komutun çalıştırılmasını sağlar
 - Örnek: **RUN echo 'hello World'**
 - **CMD**: Konteyner içinde ilk çalıştığında hangi komutların çalıştırılacağını belirler.
 - Örnek: **CMD systemctl enable httpd**

FROM

- «**Dockerfile**» bir metin dosyası olduğundan, herhangi bir metin editöründe oluşturulabilir.
- «**FROM**», eklemek için hangi imajın baz alınacağını belirler

```
FROM [--platform=<platform>] <image> [AS <name>]
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```
- Docker dosyaları «**FROM**» komutuyla **başlamalıdır**.
- «**FROM scratch**» Docker Hub'daki minimal imaja karşılık gelir.
- «**FROM alpine:latest**», Alpine Linux'un son sürümünün baz imaj olarak kullanılacağını belirtir. Eğer istenirse, baz imaj için sürüm de belirtilebilir. Örneğin, «**FROM python:3.9**»
- Aynı **Dockerfile** ile birden fazla imaj oluşturuluyorsa, dosyada birden fazla «**FROM**» satırı bulunabilir.

RUN

- Docker dosyalarında «**RUN**» imajın tarif ettiği makinenin üzerinde çalıştırılacak uygulamaları belirtir:
- Kabuk içinde çalıştırılabilir şekilde yazılabilir:

```
RUN <komut>
```

- Komut ve parametreler dizin içinde belirtilir:

```
RUN ["executable", "param1", "param2"]
```

- **RUN** komutuyla oluşan durum, o andaki katman imajına eklenir ve sonuçlar bir sonraki katmandaki imajda kullanılır.

- Örnek:

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

```
RUN ["/bin/bash", "-c", "echo hello"]
```

ADD

- **ADD** komutu, imajın içine bazı dosyaların kopyalanmasını sağlar.
- Kopyalama yeri, **Dockerfile**'da tanımlanan **WORKDIR**'e göreceli olarak gerçekleştirilebilirken tam dosya yeri de tarif edilebilir.
- Kullanımı şu yapılabilir:

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```
- Konakta **Dockerfile**'ın bulunduğu dizindeki **deneme.txt** dosyası **WORKDIR**'e relatif olarak kopyalanabilir.

ADD deneme.txt gorecelidizin/

- Tam dosya adresi / ile belirtilebilir.

ADD config.cfg /etc/deneme/

- İzinler verilebilir:

- **ADD --chown=55:mygroup files* /somedir/**
- **ADD --chown=bin files* /somedir/**
- **ADD --chown=1 files* /somedir/**
- **ADD --chown=10:11 files* /somedir/**

<https://docs.docker.com/engine/reference/builder/#add>

COPY

- Dockerfile'ın içinde bulunduğu dizinde imaja kopyalanması için tutulan dosyalar «**COPY**» komutuyla imaja eklenir.
- **ADD** komutuyla işlevsel olarak benzerdir ancak **COPY** komutu tercih edilmelidir.
- **ADD** komutunda lokal TAR dosyasının açılması ve uzak URL'den dosya indirilmesi gibi özellikler bulunur.
- Eğer **Dockerfile**'ın içinde bulunduğu dizinde kopyalanması istenmeyen dosyalar varsa **.dockerignore** dosyası içinde belirtilebilir.

«Dockerfile» directive

- «Dockerfile» içinde bulundukları yerden sonraki komutların işlenmesini etkileyen ve oluşturma işini yapan Docker Daemon'a verilen komutlardır:

`# directive=değer`

- Bir Dockerfile içinde birden fazla olabilir.
- Eğer `#` işaretinden sonra «**directive**» kelimesi yoksa, satır yorum (comment) olarak alınır.

LABEL

- İmaj içine meta-veri eklenmesi için kullanılır.
- Her **LABEL** aslında bir anahtar ve değer çiftinden oluşur.
- Örneğin:

```
LABEL "com.example.vendor"="ACME Incorporated"  
LABEL com.example.label-with-value="foo"  
LABEL version="1.0"
```

- Eğer istenirse birden fazla **LABEL** olabilir:

```
LABEL multi.label1="value1"    multi.label2="value2"  
other="value3"
```

- Etiketler (label) imaj içinde «**docker image inspect**» komutuyla görülebilir. Genellikle, sürüm numarası, kimin tarafından ne zaman oluşturulduğu gibi meta-verilerin etiketler sayesinde imaja eklenmesi, imajların yönetilmesi anlamında faydalı olur.

<https://docs.docker.com/engine/reference/builder/#label>

WORKDIR

- **WORKDIR**, **RUN**, **CMD**, **ENTRYPOINT**, **COPY**, **ADD** gibi komutlar için çalışma dizinini belirler.
- Eğer **WORKDIR** komutu olmasa da **Dockerfile** işlenirken oluşturulur.
- **Dockerfile** içinde birden fazla yerde **WORKDIR** çağrılabilir. Tanımlandığı andan sonraki komutlar için dizini belirler.

```
ENV DIRPATH=/usr/share/nginx/html
WORKDIR $DIRPATH
COPY index.html .
RUN mkdir _data
WORKDIR $DIRPATH/_data
RUN touch data.db
```

EXPOSE

- **EXPOSE** komutu, Docker'a konteynerin belirli portları dinleyeceği bilgisini verir.
- Kullanımı şöyledir:
`EXPOSE <port> [<port>/<protocol>...]`
- Konteynerde TCP (veya UDP) olarak hangi portların dinleneceği **bilgisi** sağlanır ama bunların dinlenmesini gerçekleştirecek TCP/IP sunucu altyapısını Docker kurmaz. Bu kullanıcının sorumluluğundadır.
- Gerçek anlamda portların dışa açılması ancak «**docker run -p**» komutuyla sağlanabilir. Dockerfile içindeki tanımın da üzerine bu şekilde yazılabilir (override).
- Örnek:
`EXPOSE 80/tcp`
`EXPOSE 80/udp`
- Konteyner'in çalışma esnasında belirli portların dışarı açılması istenirse şu komut kullanılabilir:
`docker run -p 80:80/tcp -p 80:80/udp`

ENV

- **ENV** komutu, imaj içinde çevre (environment) değişkeni tanımlayarak değer atamamızı sağlar.
- Kullanımı şu yapılabilir:
`ENV <key>=<value> ...`
`ENV <key> <value>`
- Bu değişkenler, konteyner çalıştırıldığında içinde çalışma anında tanımlıdır ve komut satırlarında kullanılabilir.

```
ENV PG_MAJOR=9.3
ENV PG_VERSION=9.3.4
RUN curl -SL https://example.com/postgres-$PG_VERSION.tar.xz | -xJC /usr/src/postgres
ENV PATH=/usr/local/postgres-$PG_MAJOR/bin:$PATH
```


VOLUME

- **VOLUME** komutu konteynerler arasında paylaşılabilen bir «**volume**» oluşturmak için kullanılır.
- İki farklı kullanımı vardır:
`VOLUME ["/usr/share/nginx/html"]`
`VOLUME /var/log`
- Aşağıdaki «**dockerfile**» «**docker run**» komutu çalıştırıldığında, **/myvol** isimli bir «**mount**» noktası oluşturur ve **greetings** adlı dosyayı yeni oluşturulan **volume**'a kopyalar.

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

CMD

- **CMD**, oluşturulacak imajın işlevini gerçekleştirebilmesi için çalıştırılacak komutu içerir.
- **Dockerfile**'ın içinde sadece bir «**CMD**» satırı olabilir.
- Şu şekillerde yazılabilir:

```
CMD ["executable","param1","param2"]  
CMD ["param1","param2"]  
CMD command param1 param2
```
- Örnek olarak şu satırlar verilebilir:

```
CMD echo "This is a test." | wc -  
CMD ["/usr/bin/wc","--help"]
```
- **CMD**, benzer bir işlevi olan **ENTRYPOINT** komutuyla beraber kullanılabilir.

ENTRYPOINT

- **ENTRYPOINT** komutu, bir konteynerin çalışabilir hale gelmesini sağlar.
- Kullanımı şöyledir:

```
ENTRYPOINT ["executable", "param1", "param2"]  
ENTRYPOINT command param1 param2
```

```
FROM debian:stable  
RUN apt-get update -y && apt-get install -y apache2  
EXPOSE 80 443  
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"]  
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

- Eğer istenirse, «**docker run --entrypoint**» opsiyonuyla farklı bir giriş noktası tanımlanabilir.

<https://docs.docker.com/engine/reference/builder/#exec-form-entrypoint-example>

CMD ve ENTRYPOINT

- Bu iki komut da bir konteyner çalışırken hangi programın çalıştırılacağını belirlemektedir.
- Bu komutların kullanımı için bazı kurallar bulunur:
 - **Dockerfile**'da **CMD** veya **ENTRYPOINT** komutlarından en az bir tanesi olmalıdır.
 - **ENTRYPOINT** konteyner çalışabilir bir program olarak kullanıldığında tanımlanmalıdır.
 - **CMD** komutu **ENTRYPOINT**'te tanımlanan komutun varsayılan argümanlarını belirtmek için kullanılır.
 - «**docker run**» komutunda verilen alternatif opsiyonlarla **CMD** komutu iptal edilebilir ve başka kod çalıştırılabilir.

CMD ve ENTRYPOINT

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

<https://docs.docker.com/engine/reference/builder/#understand-how-cmd-and-entrypoint-interact>

USER

- **USER** komutu, bir imaj çalıştırıldığında (**RUN**, **CMD** ve **ENTRYPOINT** komutlarında çalıştırılan kodların hangi kullanıcı tarafından çalıştırılacağını belirtir.
- Kullanımı şöyledir:

```
USER <user>[:<group>]
```

```
USER <UID>[:<GID>]
```

```
FROM microsoft/windowsservercore
# Create Windows user in the container
RUN net user /add patrick
# Set it for subsequent commands
USER patrick
```

«Dockerfile» örnekleri

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
RUN apt-get update
RUN apt-get install -y curl nginx build-essentials
```



En son sürümlerin
alınabilmesi için

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
    curl \
    nginx \
    build-essentials
```

Tek bir satır halinde yazılması,
«**CACHE boosting**»

ÖRNEK 1: İmaj Oluşturma

- Ev dizininizde «**mkdir deneme2**» diyerek bir dizin oluşturunuz.
- Ardından deneme2'ye gidin ve «**mkdir htmldir**» ile bir dizin daha oluşturun.
- Aşağıdaki HTML dosyasını **htmldir** dizini altında «**index.html**» olarak kaydedin.

```
<!DOCTYPE html>
<html>
<body>
<h2>An Unordered HTML List</h2>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
<h2>An Ordered HTML List</h2>
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
</body>
</html>
```

```
adminpc@ubuntu:~$ tree deneme2/
deneme2/
├── Dockerfile
├── htmldir
│   └── index.html
1 directory, 2 files
```


ÖRNEK 1: İmaj oluşturma

- Aşağıdaki **Dockerfile**'ı «**deneme2**» dizinin içine kaydedin

```
FROM nginx:latest
COPY htmldir /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Dizinin içinde «**nginxtry**» imajını oluşturmak için «**docker build -t nginxtry .**» komutunu verin (sondaki noktayı unutmayın).

```
adminpc@ubuntu:~/deneme2$ docker build -t nginxtry .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM nginx
---> f2f70adc5d89
Step 2/3 : COPY htmldir /usr/share/nginx/html
---> 8ec9b114a9f3
Step 3/3 : EXPOSE 80
---> Running in 472ec0a26b68
Removing intermediate container 472ec0a26b68
---> 6aa365e331f9
Successfully built 6aa365e331f9
Successfully tagged nginxtry:latest
```

ÖRNEK 1: Oluşturulan imaj nerede?

- Oluşturulan imaj, **Dockerfile** dosyasının olduğu yerde değil Docker dizinlerine konulmaktadır. Yeni oluşan imajı görmek için, «**docker images**» komutu kullanılabilir.

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginxtry	latest	a1a52aec2d42	2 days ago	142MB
alppyth	1.0.1	77b792ed5fd8	2 days ago	68MB
apinepyth	1.0.1	548710822fd5	2 days ago	52.7MB
ubuntu-guncel	latest	df531d64f1e5	3 days ago	171MB
alpine	latest	e66264b98777	5 days ago	5.53MB
nginx	latest	de2543b9436b	11 days ago	142MB
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

ÖRNEK 2: **postgresql** Dockerfile

- Üzerinde **postgresql** bulunan bir imaj oluşturmak için aşağıdaki **dockerfile** kullanılabilir.

```
FROM komljen/ubuntu
MAINTAINER Alen Komljen <alen.komljen@live.com>

ENV PG_VERSION 9.3
ENV USER docker
ENV PASS SiHRDZ3Tt13uVVyH0ZST

RUN \
    echo "deb http://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main" \
        > /etc/apt/sources.list.d/pgdg.list && \
    curl -sL https://www.postgresql.org/media/keys/ACCC4CF8.asc \
        | apt-key add - && \
    apt-get update && \
    apt-get -y install \
        postgresql-${PG_VERSION} \
        postgresql-contrib-${PG_VERSION} && \
    rm -rf /var/lib/apt/lists/*

COPY start.sh start.sh

RUN rm /usr/sbin/policy-rc.d
CMD ["/start.sh"]
```

<https://github.com/komljen/dockerfile-examples/blob/master/postgres/Dockerfile>

ÖRNEK 2: postgresql Dockerfile

- Gerekli olan **start.sh** de Dockerfile'ın bulunduğu dizinde yer almalıdır.

```
#!/usr/bin/env bash
#=====
#
#   AUTHOR: Alen Komljen <alen.komljen@live.com>
#
#=====
CONF="/etc/postgresql/${PG_VERSION}/main/postgresql.conf"
HBA="/etc/postgresql/${PG_VERSION}/main/pg_hba.conf"
DATA="/var/lib/postgresql/${PG_VERSION}/main"
POSTGRES="/usr/lib/postgresql/${PG_VERSION}/bin/postgres"
#-----
echo "host all all 0.0.0.0/0 md5" >> "$HBA"
echo "listen_addresses='*'" >> "$CONF"
#-----
echo "Creating superuser: ${USER}"
su postgres -c "${POSTGRES} --single -D ${DATA} -c config_file=${CONF}" <<EOF
CREATE USER $USER WITH SUPERUSER PASSWORD '$PASS';
EOF
#-----
echo "Starting postgres:"
exec su postgres -c "${POSTGRES} -D ${DATA} -c config_file=${CONF}"
#=====
```

<https://github.com/komljen/dockerfile-examples/tree/master/postgres>

Diğer Dockerfile Örnekleri

- Dockerfile örnekleri aşağıdaki sitelerden indirilebilir:
 - <https://github.com/dockersamples>
 - <https://github.com/prometheus/prometheus>

İmajların Depolanması ve Yüklenmesi

- Oluşturulan imajlar **tar** dosyası halinde diske depolanabilir ve başka bir yere transfer edilebilir.
- Bu amaçla kullanılan komutlar şunlardır:
`docker image save <IMAJ> --output <DOSYA.tar>`
`docker image load --input <DOSYA.tar>`
- Örneğin «**docker image save nginxtry -o deneme.tar**» komutuyla elde edilen dosyanın boyutu 146 MB'tır.

İmaj Oluşturma Kılavuzu

1. Güncel kalabilmek için hazır imajları kullanın. «**FROM**» opsiyonuyla belirli bir sürümü işaret edin kullanın. Bu tür imajların güvenlik güncellemesi genellikle sahipleri tarafından yapılmaktadır.
2. İmajları etiketleyin ve etiketlerde geriye yönelik uyumluluk sağlayacak yeni etiketler oluşturun. Örneğin, `imaj:v1` ile oluşturduğunuz bir projede, ileride **`imaj:v2`** çıkardığınız zaman imajlar güncellenmeyebilir. Bu nedenle, **`imaj:latest`** isimli ikinci bir etiket kullanın. **`imaj:latest`** --> **`imaj:v1`**'i işaret ederken, yeni imaj çıktığında **`imaj:latest`** --> **`imaj:v2`**'yi işaret edince, bütün konteynerler güncellenecektir.
3. Bir konteyner içinde birden fazla daemon bulundurmayın. Bir konteynerin içinde bulunan ve ana işlevi tamamlayan daemon'a **`sidecar`** adı verilir. Örneğin, **`httpd`** yanında **`sshd`** bulunmasın. Zira birden fazla konteyneri ortaklaşa çalıştırabilirsiniz.
4. Bir konteynerin konsoluna bağlanarak yeni yazılım yükleme yapıldığında, «**`sudo apt-get autoremove`**» veya «**`yum clean all -y`**» komutlarıyla geçici dosyaların silinmesini sağlayın

https://docs.openshift.com/container-platform/3.11/creating_images/guidelines.html

İmaj Oluşturma Kılavuzu

5. İmaj içindeki daemon'ların kullanacağı çevre değişkenlerini (**JAVA_HOME**, **MYSQL_USER**, vb) her zaman **Dockerfile** içinde belirtin
6. Varsayılan şekilde gelen şifreleri her zaman değiştirin.
7. Konteynerlere **SSHD** kurulumu yapmayın. Zira, **ssh**'a gerek olmadan «**docker exec**» komutuyla konteyner konsoluna bağlanma imkanı bulunmaktadır.
8. Konteynerler içinde metaveriler bulunsun ve bu şekilde çalışan konteynerler hakkında çalışma anında bilgi alınabilsin.
9. Loglama **STDOUT**'a yazılmalı. Docker **STDOUT**' a giden bütün logları merkezi olarak toplayarak bir yere depolamaktadır.

İmaj Kullanım Kılavuzu

10. **Dockerfile** yazarken, **ADD** komutunu dosyanın en sonuna koymaya çalışın.

```
FROM foo  
ADD dosya.txt /test/dosya.txt  
RUN apt -y install paket && apt autoremove
```

İmaj oluşturulması sırasında «dosya.txt» değişse de üstteki komutun önbellekteki hali kullanılacağı için oluşturma süresi kısılacaktır.

«dosya.txt» her değiştiğinde, «apt» komutu yeniden çalışacak ve imaj oluşturma süresi uzayacaktır.

```
FROM imaj  
RUN apt -y install paket && apt autoremove  
ADD dosya.txt /test/dosya.txt
```

Çıkabilecek arızalar

- Dockerfile'ın işlenmesi sürecinde çeşitli sorunlar çıkabilir.
- Bazen, paketleri indirdiği siteleri bulamadığını gösterebilir.

```
$ sudo docker build -t myjenkins-blueocean:2.387.1-1 .
[+] Building 41.6s (5/10)
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                  0.0s
=> [internal] load build definition from Dockerfile            0.0s
=> => transferring dockerfile: 718B                             0.0s
=> [internal] load metadata for docker.io/jenkins/jenkins:2.387.1 1.5s
=> CACHED [1/7] FROM docker.io/jenkins/jenkins:2.387.1@sha256:0944e182 0.0s
=> ERROR [2/7] RUN apt-get update && apt-get install -y lsb-release 40.0s
-----
> [2/7] RUN apt-get update && apt-get install -y lsb-release:
#0 15.31 Err:1 http://deb.debian.org/debian bullseye InRelease
#0 15.31   Temporary failure resolving 'deb.debian.org'
#0 15.31 Err:2 https://packagecloud.io/github/git-lfs/debian bullseye InRelease
#0 15.31   Temporary failure resolving 'packagecloud.io'
#0 27.60 Err:3 http://deb.debian.org/debian-security bullseye-security InRelease
```

- En iyi çözüm, Docker'ın yeniden başlatılmasıdır. Bu amaçla aşağıdaki komut kullanılmalıdır:

```
$ sudo /etc/init.d/docker restart
```

Docker tag (etiketler)

- Bu konu daha detaylı anlatılacak

Docker Hub

Docker Hub

- Docker Hub, Docker imajlar için bir dağıtım noktası olarak işlev görmektedir.
- Docker Hub'da
 - Resmi imajlar (PostgreSQL, Ubuntu, MongoDB, vb)
 - Ücretli destek sağlanan imajlar (çeşitli firmalar tarafından farklı amaçlarla çıkarılmakta ve kurulumları/yapılandırmaları için destek sağlanmaktadır)
 - DevOps ekiplerinin iç işlerinde kullanmak için oluşturdukları imajlar
 - Ücretsiz/açık veya gizli depolar
 - İmaj oluşturmak için «build» bölümü ve imaj oluşturulduğunda bildirim yapması için webhook hizmetleri.

Docker Hub

- Temel imajlar için bir açık depo olarak işlev görür.
- DevOps süreçlerinde kullanılan önemli bir işlevi vardır:
 - Örneğin eğer üzerinde bir web sunucusu olan bir konteynere ihtiyaç duyarsanız, temel imajı Docker Hub'dan indirerek üzerinde modifikasyon yapabilirsiniz ve kendi imajınızı oluşturabilirsiniz.
 - Örneğin, PostgreSQL sunucusu olan bir imajı indirerek ve kendi veritabanınızı yükleyerek kendinize özel bir imaj hazırlayabilirsiniz.
- Sorun?
 - PostgreSQL sunucusunun çok sayıda versiyonu ve dolayısıyla farklı imajları olabilir. Bunu nasıl halledeceğiz?
 - Docker Hub, bu amaçla bize bazı seçenekler sunuyor.

Docker Hub : Ubuntu İmajı

- Her imajla birlikte kullanılabilen etiketleri bulunmaktadır. Örneğin Ubuntu imajının bulunduğu sayfada görülen etiketler şunlardır:

- 18.04 , bionic-20220315 , bionic
- 20.04 , focal-20220316 , focal , latest
- 21.10 , impish-20220316 , impish , rolling
- 22.04 , jammy-20220315 , jammy , devel
- 14.04 , trusty-20191217 , trusty
- 16.04 , xenial-20210804 , xenial

- «**docker pull ubuntu:21.10**» denildiğinde, Ubuntu'nun 21.10 versiyonu çekilir.
- En son sürün çekilmek istenirse, «**docker pull ubuntu:latest**» denilebilir. Etiket belirtilmezse en son sürüm indirilir.

Docker push'u anlat