Docker Dokümantasyonu

Docker'ın Tarihi

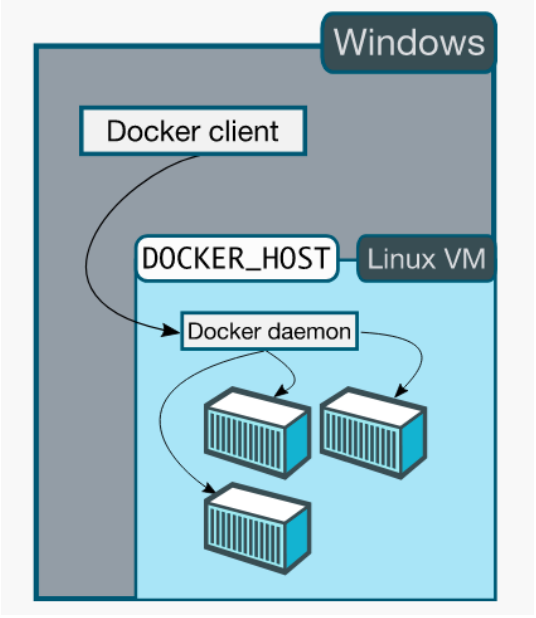
Yer, PyCon. 2013 yılının Mart aylarıydı. Docker, ilk defa bir topluluk önünde sergilenecekti. Docker'ın fikir babası olan Solomon Hykes'ın, teknoloji dünyasını değiştirmek için yalnızca 5 dakikası vardı. Yabancıların prensibi işte, 5 dakikası dolduğu için thaaaank you cümleleriyle ve yarım yamalak yazabildiği ""hello wowlrd" komutuyla birlikte sahneden uzaklaştı.

İşte computer history müzesine kaldırılması gereken o video:
https://www.youtube.com/watch?v=wW9CAH9nSLs

2008 yılında linux containers(LXC) yapısı linux kernela eklenmiştir. LXC, tek bir Linux çekirdeği kullanarak bir kontrol ana bilgisayarında birden çok yalıtılmış Linux sistemini çalıştırmak için işletim sistemi düzeyinde bir sanallaştırma yöntemidir.

Docker ise 2013 yılında LXC üzerine kurulmuş ve LXC'nin zengin mirasını kullanmıştır. Docker LXC'de manuel olarak yapılan işlemleri paketleyerek standardize etmiştir. Docker, LXC'nin sunduğu kapsamlı fonksiyonları ve konfigürasyonları detaylarından arındırmıştır. Farklı olarak image oluşturma, image paylaşma, farklı containerların aynı imageları kullanabilmesi gibi yenilikler geliştirilmiştir.

Docker, linuxta daha hızlı çalışmaktadır. Mac OS ve Windowsta ise bir linux sanal makinesi üzerinden çalışmaktadır. Bu sebeple daha farklı bir sistemi vardır. Aşağıdaki görsel windows ve mac için geçerlidir.




```
msaidzengin@msaidzengin:~$ docker version
Client: Docker Engine - Community
 Version:           20.10.1
 API version:       1.41
 Go version:        go1.13.15
 Git commit:        831ebea
 Built:             Tue Dec 15 04:34:59 2020
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.1
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       f001486
  Built:            Tue Dec 15 04:32:40 2020
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.3
  GitCommit:        269548fa27e0089a8b8278fc4fc781d7f65a939b
 runc:
  Version:          1.0.0-rc92
  GitCommit:        ff819c7e9184c13b7c2607fe6c30ae19403a7aff
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

İşletim sistemi içinde bir linux vm bulunur. Onun içerisinde Docker daemonlar bulunur. Linuxta ise direk olarak linux daemonlarla haberleşir. Bu sebeple daha hızlıdır. Daemon ve linux, CLI (command line interface) ile haberleşir.

Sağdaki resimde ise linuxta docker version komutu sonucu görüntülenmektedir. Hem Client hem de Server kısmında OS/Arch olarak linux/amd64 yazmaktadır. Bu komut windows veya mac'te çalışmış olsaydı, client kısmı farklı olacaktı. Fakat server kısmı yine linux yazacaktı.

Containerlar bir nevi virtual machine gibi çalışır. Farklı olarak, container'lar milisaniyeler içerisinde başlatılabilir, istenen herhangi bir anda duraklatılabilir (Pause), tamamen durdurulabilir (Stop) ve yeniden başlatılabilirler.

1. Kurulum

- Docker kullanımı için öncelikle bilgisayara docker kurulması gereklidir.
- Her işletim sistemi için ayrı olan indirme prosedürüne aşağıdaki linkten ulaşılabilir. https://www.docker.com/get-started

Örneğin Ubuntu için takip edilmesi gereken talimatlar :

- https://www.docker.com/get-started adresinden linux seçilir.
- Docker Engine - Ubuntu (Community) seçilir ve https://hub.docker.com/editions/community/docker-ce-server-ubuntu sayfasına ulaşılır.
- Sayfaya ulaştıktan sonra en alttaki https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/ bağlantının yönlendirdiği sayfaya gidilir ve buradaki talimatlar takip edilerek kurulum yapılır.

Kurulum işlemi tamamlandıktan sonra docker versiyonunu kontrol etmek için :

- docker -v komutu çalıştırılır.

```
msaidzengin@msaidzengin:~$ docker -v
Docker version 20.10.0, build 7287ab3
```

Kurulumun sorunsuz tamamlandığını görmek için bir örnek çalıştırılır :

- docker run hello-world komutu çalıştırıldığında docker hub üzerinden bir image çekilerek bir container ayağa kaldırılır.
- Bunun sonucunda terminal ekranında Hello from Docker! yazısı gördüğümüzde kurulum işlemi başarıyla tamamlanmış demektir.

```
msaidzengin@msaidzengin:~$ sudo docker run hello-world
[sudo] password for msaidzengin:

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/
```

2. Hello World Denemesi

Dockerfile :
docker-compose :
Docker-hub :

sudo docker-compose up
sudo docker-compose up -d

Example *Dockerfile*

FROM acidgenomics/miniconda3
ADD hello.py /
CMD ["python3", "./hello.py"]

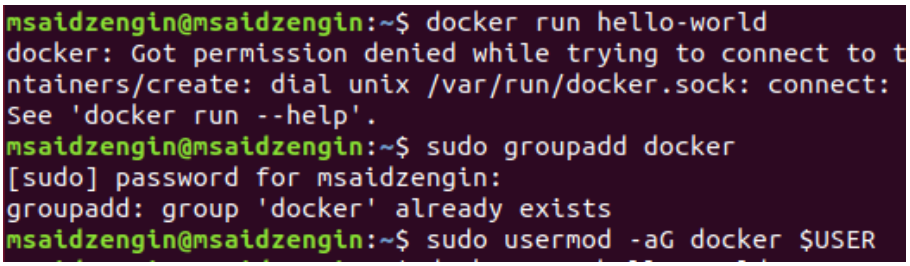
Example *hello.py* file
print("hello world")

Build komutu : *sudo docker build -t mini:v2 .*
Çalıştırma komutu : *sudo docker run mini:v2*

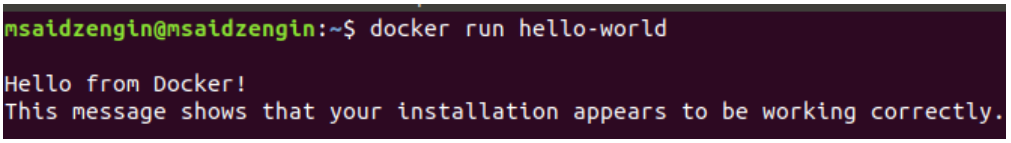
3. Docker User Tanımlama

Docker kullanmak için yönetici izni gereklidir. Sadece Docker kullanım izni vermek için yeni bir kullanıcı tanımlanabilir. Kullanıcı root user olmadan sadece Docker yöneticisi olur.

sudo groupadd docker
sudo usermod -aG docker $USER
(Kullanıcı hesabını kilitleyip tekrar açmak gerekir.)
docker run hello-world



Bilgisayara reset atılır ve artık sudo komutuna ihtiyaç yoktur.



4. Docker Otomatik Başlatma

Sistem başlangıcında Docker'ı otomatik olarak başlatmak için
sudo systemct1 enable docker

5. Container ve Local Haberleşmesi

docker run --name <filename> -v /var/www:/usr/share/mlflow:ro -p -d <image name>

80/tcp portunu 332768 şeklinde dışarıya açıyor.
0.0.0.0:332768

6. Örnek Komutlar

sudo docker build -t [name]:[version] .
sudo docker build -t [name]:[version] /filename
sudo docker images
sudo docker image ls
sudo docker ps
sudo docker ps -a
sudo docker image rm [image-id]
sudo docker image rmi [image-id]
sudo docker image rm -f [image-id]
sudo docker run [container-id]

sudo docker run -p 5000:5000 mlflow (dış hat:iç hat)
sudo docker run --name [name] -d -p 5000 [image-name]

```
msaidzengin@msaidzengin:~/Desktop/dockermlflow$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS                     PORTS                    NAMES
69435bfee6f4   examplemlflow  "/bin/sh -c 'mlflow …"   23 seconds ago      Up 22 seconds              0.0.0.0:49153->5000/tcp  mlflow1
7855655b1b35   examplemlflow  "--name mlflow1 -d -…"   48 seconds ago      Created                    5000/tcp                 affectionate_chandrasekhar
5ae953c2fa27   examplemlflow  "/bin/sh -c 'mlflow …"   12 minutes ago      Up 12 minutes              0.0.0.0:5000->5000/tcp   pedantic_fermat
677f7824a051   91df54a9c648   "python3 ./hello.py"     37 minutes ago      Exited (0) 37 minutes ago                           inspiring_liskov
aae1c7ff36b0   8df89dd209f0   "python ./hello.py"      40 minutes ago      Created                                             naughty_leavitt
a556785a6a11   8df89dd209f0   "python ./hello.py"      40 minutes ago      Created                                             sleepy_fermat
6ec566ec9256   bda27a013ab2   "/bin/sh -c 'python …"   53 minutes ago      Exited (2) 53 minutes ago                           sweet_mendel
5336b067eea3   hello-world    "/hello"                 About an hour ago   Exited (0) About an hour ago                        suspicious_leakey
2c312c508dd6   hello-world    "/hello"                 2 hours ago         Exited (0) 2 hours ago                              quizzical_morse
```

docker start -a [container-id] container'ı tekrar çalıştır ve outputu terminale bağla
docker start [container-id] container'ı tekrar çalıştır fakat outputu bağlama
docker log [container-id] container'ın loglarını(tüm eski outputlarını) ekrana bas

docker rm [container-id] komutu ile eğer silemiyorsak -f kullanarak silebiliriz. Veya
docker stop[container-id] kullanarak önce durdurup sonra silebiliriz.
docker tag [image-id] [rapo-name]:[version] tag, repo:version eklemek için kullanılır.
Örn:
docker build deneme .
docker tag a7488… msaidzengin/deneme:0.1

Bu iki komut satırı aslında şuna denktir:
docker build -t msaidzengin/deneme:0.1 .

docker login komutu ile login olabilirsiniz. Daha sonra hazırladığınız image'ı dockerhub'a gönderebilirsiniz.

```
msaidzengin@msaidzengin:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: msaidzengin
Password:
WARNING! Your password will be stored unencrypted in /home/msaidzengin/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
msaidzengin@msaidzengin:~$
```

(Parola base64 ile config dosyasına kayıt edildi. Bu şekilde durması çok hoş olmadı)
docker push [repo-name]:[version]
komutu ile docker hesabınıza public olarak gönderebilirsiniz ve herkesin kullanımına açabilirsiniz.

7. Docker Cheat Sheet

Komut	Açıklaması
`docker images`	Lokal registry'de mevcut bulunan Image'ları listeler
`docker ps`	Halihazırda çalışmakta olan Container'ları listeler

`docker ps -a`	Docker Daemon üzerindeki bütün Container'ları listeler
`docker ps -aq`	Docker Daemon üzerindeki bütün Container'ların ID'lerini listeler
`docker pull <repository_name>/<image_name>:<image_tag>`	Belirtilen Image'ı lokal registry'ye indirir. Örnek: `docker pull gsengun/jmeter3.0:1.7`
`docker top <container_id>`	İlgili Container'da `top` komutunu çalıştırarak çıktısını gösterir
`docker run -it <image_id|image_name> CMD`	Verilen Image'dan terminal'i attach ederek bir Container oluşturur
`docker pause <container_id>`	İlgili Container'ı duraklatır
`docker unpause <container_id>`	İlgili Container `pause` ile duraklatılmış ise çalışmasına devam ettirilir
`docker stop <container_id>`	İlgili Container'ı durdurur
`docker start <container_id>`	İlgili Container'ı durdurulmuşsa tekrar başlatır
`docker rm <container_id>`	İlgili Container'ı kaldırır fakat ilişkili Volume'lara dokunmaz
`docker rm -v <container_id>`	İlgili Container'ı ilişkili Volume'lar ile birlikte kaldırır
`docker rm -f <container_id>`	İlgili Container'ı zorlayarak kaldırır. Çalışan bir Container ancak `-f` ile kaldırılabilir
`docker rmi <image_id|image_name>`	İlgili Image'ı siler
`docker rmi -f <image_id|image_name>`	İlgili Image'ı zorlayarak kaldırır, başka isimlerle Tag'lenmiş Image'lar `-f` ile kaldırılabilir
`docker info`	Docker Daemon'la ilgili özet bilgiler verir
`docker inspect <container_id>`	İlgili Container'la ilgili detaylı bilgiler verir
`docker inspect <image_id|image_name>`	İlgili Image'la ilgili detaylı bilgiler verir
`docker rm $(docker ps -aq)`	Bütün Container'ları kaldırır
`docker stop $(docker ps -aq)`	Çalışan bütün Container'ları durdurur
`docker rmi $(docker images -aq)`	Bütün Image'ları kaldırır
`docker images -q -f dangling=true`	Dangling (taglenmemiş ve bir Container ile ilişkilendirilmemiş) Image'ları listeler
`docker rmi $(docker images -q -f dangling=true)`	Dangling Image'ları kaldırır
`docker volume ls -f dangling=true`	Dangling Volume'ları listeler

`docker volume rm $(docker volume ls -f dangling=true -q)`	Danling Volume'ları kaldırır
`docker logs <container_id>`	İlgili Container'ın terminalinde o ana kadar oluşan çıktıyı gösterir
`docker logs -f <container_id>`	İlgili Container'ın terminalinde o ana kadar oluşan çıktıyı gösterir ve `-f` follow parametresi ile o andan sonra oluşan logları da göstermeye devam eder
`docker exec <container_id> <command>`	Çalışan bir Container içinde bir komut koşturmak için kullanılır
`docker exec -it <container_id> /bin/bash`	Çalışan bir Container içinde terminal açmak için kullanılır. İlgili Image'da /bin/bash bulunduğu varsayımı ile
`docker attach <container_id>`	Önceden detached modda `-d` başlatılan bir Container'a attach olmak için kullanılır

8. Docker nginx Kurulumu Örneği

Komut: *docker run -p 8080:80 nginx*

```
msaidzengin@msaidzengin:~$ docker run -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6ec7b7d162b2: Pull complete
cb420a90068e: Pull complete
2766c0bf2b07: Pull complete
e05167b6a99d: Pull complete
70ac9d795e79: Pull complete
Digest: sha256:4cf620a5c81390ee209398ecc18e5fb9dd0f5155cd82adcbae532fec94006fb9
Status: Downloaded newer image for nginx:latest
/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: Configuration complete; ready for start up
```

Komut çalıştığında nginx latest indirilmiş ve çalışmış oldu. http://localhost:8080 da çalıştı.

Şimdi yeni bir terminal açıp bunu inceleyelim.

```
msaidzengin@msaidzengin:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                    NAMES
918466906e1b   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:8080->80/tcp   agitated_ramanujan
```

```
msaidzengin@msaidzengin:~$ docker exec -it 918 /bin/bash
root@918466906e1b:/# ls
bin   dev                  docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d  etc                   lib   media  opt  root  sbin  sys  usr
```

Komut: *docker exec -it 918 /bin/bash*

Bu komut ile container'da bir Bash Shell açabiliriz. -i interaktif terminali -t ise terminalin buraya attach olmasını sağladı. 918 ise container-id'nin başlangıcı.

```
root@9184669o6e1b:/#
root@9184669o6e1b:/# cat /usr/share/nginx/html/
50x.html    index.html
root@9184669o6e1b:/# cat /usr/share/nginx/html/index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    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
```

Dockerı kill edelim.

```
msaidzengin@msaidzengin:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                    NAMES
918466906e1b   nginx     "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes   0.0.0.0:8080->80/tcp   agitated_ramanujan
msaidzengin@msaidzengin:~$ docker kill 918
918
msaidzengin@msaidzengin:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
msaidzengin@msaidzengin:~$ docker ps -a
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS                       PORTS      NAMES
918466906e1b   nginx           "/docker-entrypoint.…"   7 minutes ago    Exited (137) 7 seconds ago              agitated_ramanujan
6caf670ae5da   hello-world     "/hello"                 31 minutes ago   Exited (0) 27 minutes ago               tender_maxwell
```

9. Docker görüntüsü

Docker görüntüsü, container'ınızı tanımlayan salt okunur bir şablondur. Görüntü, kodunuzun gereksinim duyduğu kitaplıklara ve bağımlılıklara yönelik tüm tanımlar dahil olmak üzere çalışacak kodu içerir. Docker container, başlatılmış (çalışan) bir Docker görüntüsüdür. AWS, Docker görüntülerinin depolanması ve hızla alınmasına yönelik Amazon Elastic Container Registry (ECR) adlı görüntü kayıt defterini sunar.

10. Docker ile sanal makineler arasında ne fark vardır?

Sanal makineler (VM) sunucu donanımını sanallaştırırken (doğrudan yönetme gereksinimini ortadan kaldırma), container'lar bir sunucunun işletim sistemini sanallaştırır. Docker, container'lara yönelik bir işletim sistemidir (veya çalışma zamanı). Docker Altyapısı container çalıştırmak istediğiniz her sunucuya yüklenir ve container'ları oluşturmak, başlatmak veya durdurmak için kullanabileceğiniz bir dizi basit komut sağlar.

11. Yazılım Geliştirirken Docker Nasıl Kullanılır

Projeye başlarken, dağıtırken ve proje sonunda Docker görüntüsü oluşturmak mantıklıdır. Projenin her versiyonu için bir Docker görüntüsü kaydedilebilir. Fakat geliştirme esnasında sürekli olarak Docker image oluşturmak, her kod değişikliğinde build almak mantıklı değildir.

Production için, her versiyonda eksiksiz bir Docker oluşturmak isteriz. Fakat geliştirme ortamında bunun yerine hızlı yinelemelere izin verecek şekilde tasarlanmalıdır. Kodunuzu, her değişiklikte yeni bir image oluşturmak yerine, bind mount'ları kullanarak başlatılan bir container ile paylaşabilirsiniz. Docker CLI kullanarak yeni bir container başlatırken lokal bir "./source_dir" dizinini nasıl bağlayabileceğiniz aşağıda açıklanmıştır:

$ *docker run -it -v "$(pwd)/source_dir:/app/target_dir" ubuntu bash*
veya
$ docker run -it --mount "type=bind,source=$(pwd)/source_dir,target=/app/target_dir" ubuntu bash

Örnek docker-compose.yml dosyası:

```
version: '3'
services:
  example:
    image: ubuntu
    volumes:
      - ./source_dir:/app/target_dir
    command: touch /app/target_dir/hello
```

If you want to save on typing, consider using docker-compose and a docker-compose.yml files to configure your Docker containers. Here's an example docker-compose.yml file mounting a local directory:

```
version: '3'
services:
  example:
    image: ubuntu
    volumes:
      - ./source_dir:/app/target_dir
    command: touch /app/target_dir/hello
```

The example above will create a new file in the shared folder and exit the container when you run `docker-compose up`.

Container'da yukarıdaki gibi proje dizininde çalışan bir bind mounts kullanırsanız, Docker görüntüsünü uzun süre yeniden kullanabilirsiniz. Container çalıştığında yerel dizindeki dosyalar kopyalanır. Böylece görüntüyü bir kere build etmeniz yeterli olur. Python paketleri gibi yüklü kütüphaneler değişene kadar kullanabilirsiniz.

Kod dizinini koyuyor olmanız, görüntüye kod ekleyemeyeceğiniz anlamına gelmez. Görüntünüzü requirements.txt dosyasıyla oluşturmak, Python bağımlılıklarını ona dayalı olarak kurmak güzel bir çözümdür.

Container içinde bir development server çalıştırırsanız, mounted kod dosyalarındaki değişiklikleri sürekli olarak alır. Yeniden başlatmanıza gerek yoktur. Klasör paylaşılır ve lokal olarak yaptığınız değişiklikler dockerize işlemi tarafından hızlı bir şekilde fark edilir.

Docker görüntüsünün yeniden oluşturulma sayısını azaltmak, dockerize developent workflowunuzu hızlandırmaya yönelik ilk adımdır. Deneyiminizi daha iyi hale getirmek için kullanabileceğiniz epeyce teknik ve püf noktası vardır. Bu sadece ilk adımdır.

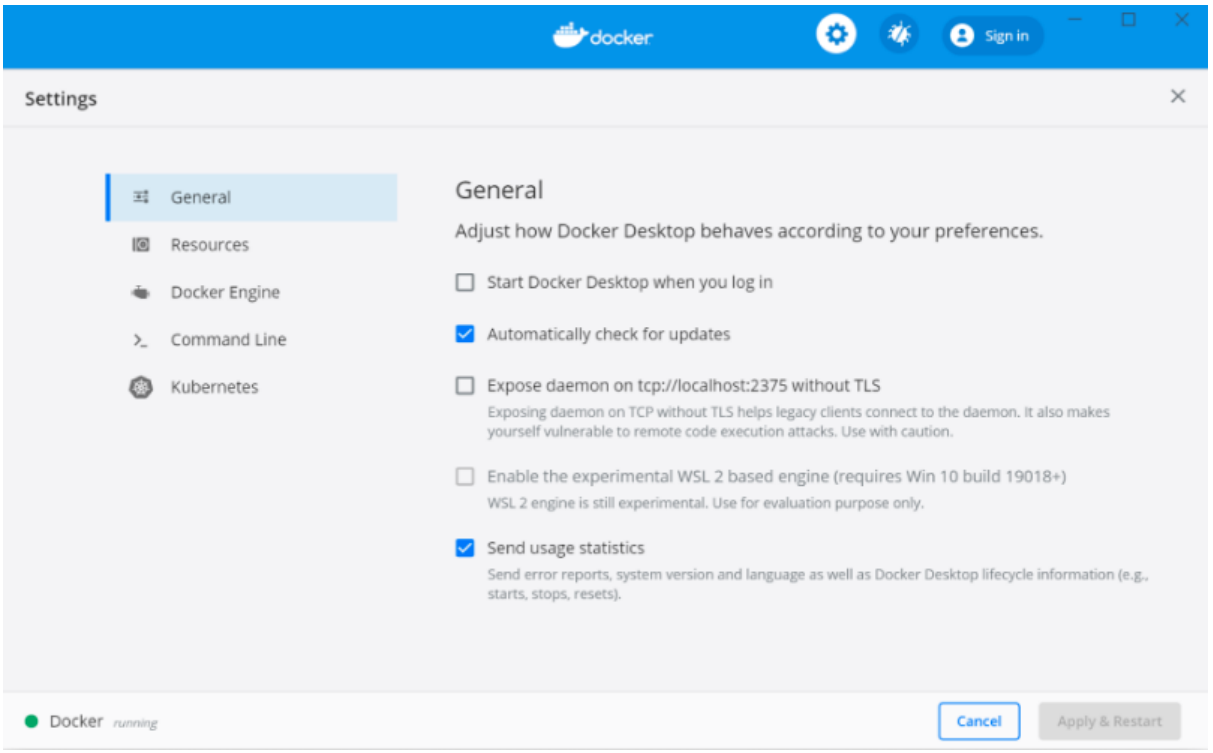
Sonuç olarak, her kod değişikliği için Docker görüntünüzü yeniden oluşturmanıza gerek yoktur. Kodunuzu development containerına eklerseniz, her kod değişikliğinde yeni bir görüntü oluşturmanız ve daha hızlı yineleme yapmanız gerekmez.

Kaynak: https://vsupalov.com/rebuilding-docker-image-development/

12. Docker Desktop

Windows ve mac işletim sistemleri için native masaüstü Docker uygulaması bulunmaktadır. Bu uygulama komut satırında yapılan işlemleri görsel hale getirmektedir. Linux için böyle bir uygulama henüz yoktur.

Kaynak: https://www.docker.com/products/docker-desktop
https://hub.docker.com/search?q=&type=edition&offering=community&platform=desktop



13. Docker / VM Karşılaştırması

Kıyas türü	VM	Docker
OS	Tam işletim sistemi	Küçültülmüş işletim sistemi imajı
Izolasyon	Yüksek	Daha düşük
Çalışır hale gelmesi	Dakikalar	Saniyeler
Versiyonlama	Yok	Yüksek
Kolay paylaşılabilirlik	Düşük	Yüksek

Dockerfile komutları

FROM	- kullanılacak base
MAINTAINER	- sunan kisi ve maili
RUN	- build işlemi sırasında eklenecek paketler
CMD	- imageda çalışacak default komut belirlenir.
ENTRYPOINT	-
EXPOSE	-
ADD	- görüntüye dosya ekler, pcden veya intten
COPY	- görüntüye pcden dosya ekler
WORKDIR	- working directory değiştirir
HEALTHCHECK	-

FROM ubuntu:latest
MAINTAINER isim soyisim <abcd@gmail.com>
RUN apt-get install -y iputils-ping
CMD ["executable", "param1", "param2"]
CMD ["/bin/ping", "8.8.8.8"]
CMD ["param1", "param2"]
ENTRYPOINT ["ping"]
CMD ["8.8.8.8"]
CMD ping 8.8.8.8
ENTRYPOINT ["executable", "param1", "param2"]
ENTRYPOINT ["/bin/ping"]
docker run gsengun/myubuntu:0.2 8.8.8.8
ENTRYPOINT command param1 param2
ADD ["./BuildDir/", "/app/bin"]
ADD ["https://curl.haxx.se/download/curl-7.50.1.tar.gz", "/tmp/curl.tar.gz"]
WORKDIR /app/src
ADD ["./BuildDir/", "binaries/"]

Örn:

```
ENTRYPOINT [ "/bin/ping" ]
CMD ["-help"]
```

Burada container'ın çalıştıracağı executable /bin/ping olarak belirlenmiştir ve kullanıcı bir parametre sağlamadığında -help parametresi verilmesi sağlanmıştır, dolayısıyla parametre verilmeden Container'ın çalıştırılmak istendiği durumda Image'ın kullanımı özetlenecektir.

Parametre verilmezse şu şekilde sonuç dönecektir:

```
docker run abcd/myubuntu:0.3
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
        [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
        [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
        [-w deadline] [-W timeout] [hop1 ...] destination
```

Parametre verilirse:

```
docker run abcd/myubuntu:0.3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=37 time=0.197 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=0.334 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=37 time=0.333 ms
```

Örn:

```
HEALTHCHECK --interval=10s --timeout=60s --retries=1 CMD curl -f http://localhost:9876/ || exit 1
```

Örnekte 10 saniyede bir koşturulan curl -f http://localhost:9876 komutu -curl ile yerel olarak 9876 portunda çalıştırılan web sunucudan ana sayfayı isteyen komut- ile maksimum 60 saniyede bir çıktı alıp alamadığına bakar eğer alamıyorsa Container'ın health status'unu unhealthy olarak işaretler, eğer cevap alırsa container'ı bu kaz success ile işaretler.

Docker'ın içini açma ve terminali bağlama komutu:

docker run -it --entrypoint bash <image-name-or-id>

Referanslar ve Kaynaklar:

https://medium.com/swlh/alpine-slim-stretch-buster-jessie-bullseye-bookworm-what-are-the-differences-in-docker-62171ed4531d

https://vsupalov.com/rebuilding-docker-image-development/ (Docker)

https://docs.docker.com/storage/bind-mounts/ (Bind Mounts)

https://gokhansengun.com/docker-nedir-nasil-calisir-nerede-kullanilir/ (Docker Nedir)

https://tahiroglu.com/post/145827965207/docker-medeniyeti#_=_ (Docker Nedir)

https://www.youtube.com/watch?v=3N3n9FzebAA (Docker Tanıtım)

https://gokhansengun.com/docker-yeni-image-hazirlama/ (Docker)

https://gokhansengun.com/docker-compose-nasil-kullanilir/ (Docker Compose)

Docker’ın içinde kodu çalıştırma komutu
python -m deneme.__main__
python3 -m ai.__main__

docker build -t stmdeneme:1.0 .
docker run -it --entrypoint bash stmdeneme:1.0
Docker dosyası çalıştırılır. Daha sonra sistem çalışır.

14.12.2020
Rabia Bilgücü
M. Said Zengin