**DOCKER NOTES**

1. EC2’ya baglan ya da desktopta Docker’i ac.

(Burada flask app’inin oldugu klasore gel ve virtual environment’i aktive et.

2. sudo yum update -y (EC2 makineye baglandiktan sonra genel bir guncelleme komutu)

# Install the most recent Docker Community Edition package. (Bu komut AWS’de EC2’da iken bir amazon makineye docker install etmek icin)

3. sudo amazon-linux-extras install docker -y

# Starting a docker service

4. sudo systemctl start docker

# Enable docker service so that docker service can restart automatically after reboots.

5. sudo systemctl enable docker

# Check if the docker service is up and running.

6. sudo systemctl status docker

(Bu koddan sonra active yazisini yesil olarak gormemiz gerekiyor)

# Check the running containers

7. sudo docker ps

# - Add the `docker` group to the `ec2-user` to run docker commands without using `sudo`.

8. sudo usermod -aG docker ec2-user

# - Normally, the user needs to re-login into bash shell for the group `docker` to be effective, but `newgrp` command can be used activate `docker` group for `ec2-user`, not to re-login into bash shell.

9. newgrp docker

# from now on we don't have to type sudo check running containers again

10. 'docker ps' --> same command with 'docker container ls'

# show running and stopped containers (butun container’lari gormek icin)

11. docker ps -a

# docker image list (mevcut docker image’larinin listesi)

13.docker image ls

# pull an image --> show from docker hub as well --> tags etc.

14. docker pull ubuntu # --> latest is the default tag

16. docker info

# Create a container from an image

# only required argument is image name --> ubuntu

# if the image does not exist on the machine, it is pulled from the registry

17. docker run -it --name myubuntu ubuntu # --> docker container

# -it --> opens a virtual terminal session in docker container

# - interactive terminal

* docker run -it ubuntu (burada ubuntu yerine mevcut image’lardan birinin ismini yazarak calistiriyoruz)

18. python --version (image icinde python var mi yok mu)

19. apt-get update -y (image son haline guncelleme) (nasil ec2’yu ilk actigimizda sudo yum update -y ile update islemi gerceklestirdiysek burada da ubuntu image’i iceren container’i update ediyoruz)

20. apt-get install python3 (bu komut ile python3 ubuntu image’inin icine indirilmis olur)

21. apt-get install python3-pip (container’in icine pip indirmek icin, container icinde nasil python direct olarak yoksa pip de yok. Python kutuphanelerini install edebilmek icin pip’I yuklememiz gerekiyor)

22. terminali kapat --> container will be gone

23. cat /etc/os-release (image icerisindeki mevcut OS hakkinda bilgi icin)

24. pip3 list (mevcut kurulu python paketlerini goruntulemek icin)

25. pip install pandas (normal bir bilgisayara kurulum yapiyormus gibi kurulum yapabilirim)

26. exit ile image’dan cikis yapabilirim.

### Eger terminal’i kapatirsam docker baglantim bitecegi icin tekrar docker ps dedigimde calismayacak ve benden sudo docker ps yazmami isteyecek. Bunu engellemek icin uzantisi sh olan bir dosya olusturulur, ornegin ”docker\_runner.sh”. Icerisine:

sudo yum update -y

sudo amazon-linux-extras install docker -y

sudo service docker start

sudo usermod -aG docker ec2-user

newgrp docker

systemctl enable docker

yazarak kaydeder ve terminalde ”sh docker\_runner.sh” komutunu calistirdigimizda yukaridaki tum komutlari tek tek calistiracagi icin sadece docker ps komutunu kabul edecektir.

27. biz bir image’i calistirdigimizda onunla ilgili bir container da olusmus oluyor. Cikis yaptiktan sonra tekrar container’i calistirmak icin:

docker start container\_id/container name (burada container ismi, id’si ya da id’nin ilk uc hanesi yazilabilir)

docker attach container\_id (bu komut ile container’a geri donus saglanir, bu sayede container icindeki image da acilmis olur)

container’in icinden cikmak icin exit yazabilirsiniz ya da ctrl d.

# REMOVING A CONTAINER

28. docker rm XXX # XXX --> CONATINER ID's first 3 chars

docker rm container\_name/container\_id --> same thing

29. docker ps -a --> no container w same name

30. history --> summarize all the commands we had so far

# Detached mode’da calistirmak icin

31. docker run -it -d --name myubuntu ubuntu --> this command does not attach the container so we cannot go inside the container immediately but docker run -it --name myubuntu ubuntu --> attach the container immediately

(-d ile container’I terminalden bagimsiz hale getiriyoruz. Terminali kapatsak bile container calismaya devam edecek)

32. docker ps

33. # Going inside the docker container

Once calistirmamiz gerekiyor

1. docker start container\_id

1. docker exec -it xxx bash

34. # Exit edilmis bir container calistirma

docker start xxx

docker attach xxx

# Exit Container

35. 'CTRL + D' or 'exit'

36. docker ps

37. 'docker start xxx' and ('docker exec -it xxx bash' or 'docker attach xxx')

38. docker stop xxx

39. docker start xxx

40. docker run -it -d -p 80:80 nginx

(bu komut ile docker once nginx adinda bir image var mi diye bakiyor, yoksa direkt pull islemi gerceklestiriyor, hem interactive hem de detached mod’da yapiyor bu islemi. Ayrica nginx’in calismasi icin 80 portuna ihtiyac var. Bunun icin 80:80 diyerek benim local’deki 80 portumu onun 80 portuna esitliyorum, boylece benim local’deki 80 portumu kullanmasini sagliyorum. Nginx image’ini indirirken ayni zamanda bunun icinde calisacagi bir container’i da aktif hale getirmis oluyor.)

Burada nginx’e localhost ile ulasabildigim icin onun uzerinde degisiklik de yapabilmeme mumkun. cd /usr/share/nginx/html/ komutu ile index.html dosyasina erisip bu dosya uzerinde yaptigim tum degisiklikler nginx sayfasina direct olarak yansiyor.

41. apt-get install vim -y

42. ls -l --> list files in docker container

# Remove Docker image

43. docker rmi xxx --> cannot remove stop containers’ images

# --rm flag --> removes a container when it stops

docker image rm image\_id

44. docker rm xxx

# VOLUME MOUNTING

45. docker rm container\_id (durmus vaziyetteki bir container’i silmek icin)

46. docker stop container\_id (calisir vaziyetteki bir container’i silmek icin)

47. docker rm -f container\_id (hangi pozisyonda olduguna bakmaksizin bir container’i zorla silmek icin)

(bir container olusturdugumuzda bu container icindeysek yaptigimiz dosyalari doldurabilecegimiz bir volume olusturacagiz, bu container sonradan silinse bile bu dosyalar volume icinde kalacagi icin kaybolmayacak, sonrasinda baska bir container ile iliskilendirerek kullanmaya devam etmek de mumkun olacak)

48. docker volume create clarus-vol (bir volume olusturmak icin)

49. docker volume ls (mevcut volume’lari gorebilmek icin)

50. docker run -dit -v clarus:/app –name mycontainer2 ubuntu

(bu komut ile bir ubuntu image’ini calistirmis olduk. Eger boyle bir image zaten varsa onu calistirdi, yoksa pull edecek. Ayrica bunu detached mod’da ve interactive olarak calistirdik. Ayrica -v ile clarus volume’una iliskilendirdik ve bu volume icerisinde varsa app klasorune ulasmasini istedik, yoksa bu klasoru kendisi kuracak. Yani burada clarus volume’u ile app klasorunu birbiriyle iliskilendirmis oldum. Bu iliskilendirilen klasorun icine atilan bilgiler ayni zamanda volume’a da atilmis oldugu icin container sonradan silinse de volume’daki bilgiler silinmiyor.

sudo ls /var/lib/docker/volumes/myvolume/\_data

(burada myvolume benim volume’um)

51. docker exec -it container\_id bash

(bu komut ile yukarida olusturdugum container’a baglaniyorum)

52. ls -l (burada ls yaptigimizda app klasorunun olustugunu goruruz.)

53. docker volume inspect volume\_name (bu komut ile secilmis olan volume hakkinda detayli bilgi sunuluyor)

## Su ana kadar hazir image’lar kullaniyorduk, simdi kendi image’imizi olusturacagiz.

Image olusturmak icin dockerfile’i nerede build edeceksek orada bir dockerfile olusturuyoruz.

Bunun icin touch Dockerfile yapiyoruz ve sonrasinda nano Dockerfile ile icine giriyoruz.

Orada:

FROM ubuntu (ubuntu bilgisayar kullaniyorsam buraya ubuntu)

RUN apt-get update -y

RUN apt-get install python3 -y

RUN apt-get install python3-pip -y

RUN pip3 install -U pip

RUN mkdir /app

COPY requirements.txt /app/requiremements.txt

RUN pip3 install -r /app/requirements.txt

WORKDIR /app (kurulacak image’i nereye kursun)

COPY . /app (ilk nokta bulunulan yerdeki unsurlari /app ile working dir’e kopyala)

CMD python3 app.py

Bunu yazip nano’yu kaydediyoruz ve kapatiyoruz. Sonra “docker build -t flaskapp .” diyerek calistiriyoruz. Burada flaskapp adinda bir image’i yukaridaki icerigi kuracak sekilde olusturmus oluyoruz. Nokta ile o anda bulundugumuz klasorde dockerfile’i ara diyoruz.

Boylece kendi hazirladigimiz bir uygulamayi dockerize etmis oluyoruz.

Bu image’i olusturduktan sonra

docker run -dit --name mynewcontainer -p 82:80 flaskapp dersem bu image’i kendi olusturacagi bir container’in icinde calistirmaya baslayacak. Eger burada streamlit bir app calistirsaydim 80:8501 diyebilirdim.

* Image olusturduktan sonra bunu docker hub’a ya da aws’nin elastic container repository’ye yukleyebiliyoruz. Oradan da ilgili kisilerin bu image’a ulasmasi saglanabiliyor.
* Image olustuktan sonra image’I olustururken kullandigimiz file’larda herhangi bir degisiklik yaparsak image’i tekrar olusturmamiz gerekiyor.

Export PATH=”buraya uyari mesajindaki path yazilir:$PATH”

**DOCKER NOTES FROM PRE-CLASS**

docker run -dit --name clarusway ubuntu (bu kod bir ubuntu image’inin calistigi bir container olusturacak ve container’in adini clarusway olarak atayacak. Once local’de bu image var mi diye bakacak yoksa docker hub’dan bu image’I pull yapacak)

docker stop container\_id (to stop a container)

docker start container\_id (to start a container)

docker attach container\_id (to attach to a container)

docker inspect container\_id (to retrieve detailed information about a container)

docker rm container\_id (to delete a container)

docker pause container\_id/name

docker unpause container\_id/name

docker volume create volume\_name (to create a volume)

docker volume inspect volume\_name (to retrieve detailed info about the volume)

docker container run -it -v volume\_name