Docker

Docker version

Docker info

Docker cli => docker cli kullanılabilcek komutlar

docker container run hello Word (docker (ne yönetmek istiyorsan o ) girmek istediğimiz komut imaj) => container oluşturma

docker run = docker container run (yeni sistem)  
  
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...] => container oluşturma

docker container run --name ilkcontainer ozgurozturknet/app1  
  
  
not: her container imajından,o imajdan bir container yarattığımız zaman varsayılan olarak çalışması için ayarlanmış bir uygulama vardır. Bu uygulama çalıştığı sürece container ayakta kalır uygulama çalışamayı bıraktığı zaman container kapatılır.

Docker container logs ef251646c28(id) => container loglari

Docker ps – docker ps –a = docker container ls - docker container ls –a   
  
docke container rm id => container silme

Docker container rm –f id => container çalışsa bile silmeye zorlar

Docker container exec –it websunucu sh => çalışan kontainera bağlanır (sh shall’e bağlan anlamında)

İt => --interaactive --tty = container’a interactive modda bağlan

Ls –l => folder üzerindeki dosyları listeler

Docker container prune => sistemde çalışmayan bütün containlerlari siler.

docker image prune –a => sistemdeki bütün imageleri siler

docker image pull alpine => dockerhub üzerindeki alpine imajını çektik

**Volume :**

docker volume create ilkvolume => volüme oluşturma  
docker volume inspect ilkvolume => volüme detayları

docker container run -it -v ilkvolume:/uygulama alpine sh

-it ile interaktif bağlantı yapıyoruz.

v volüme bağlıyoruz

hangi klasöre bağlamak isyorsak onu yazıyoruz

Volüme adı : container içindeki klasör

/ hangi klasöre bağlamak istiyorsak onu yazıyoruz

Alpine imajından oluştur   
sh – shall aç

Docker volüme ls = volumeleri veririr

docker container run -it -v ilkvolume:/deneme2 centos sh = ro ile read only olarak yeni bir container oluşturduk

docker container run --rm -it ozgurozturknet/adanzyedocker sh=> --rm (container kapatıldığı zaman sil)

docker container run --rm -it -v deneme1:/test ozgurozturknet/adanzyedocker sh

docker container run --rm -it -v deneme1:/usr/src/myapp ozgurozturknet/adanzyedocker sh => boş volume’i içinde dosya olan bir klasöre mount ettik

**Bind Mounts:**

Local de dosya oluşturarak bunu mount ediyoruz örnek olarak bir web projemiz var nginx ile publish ediyoruz normal şartlarda her bir değişiklikte yeni bir imaj yaratarak yeni bir container oluşturup index.html güncelleyceğiktin ancak bind mount ile anlık değişikliği görüp bu zahmetten kurtuluyoruz.

Sadece kendi sistemlerimizde development yaparaken kullandığımız bir özelliktir.

Docker container run –d –p 80:80 –v c:\adanzyedocker\kisim3\bolumm28\websitesi:/ usr/share/nginx/html   
  
ilk kısımda bizim local pc deki konum : sonrasında ise mount edilecek yeri belirttik.

Notlar :  
 Docker'da bir volume'ü silmek için şu komutu kullanabilirsiniz:

docker volume rm <volume\_adı>

olume'ü silmeden önce listelemek isterseniz şu komutu kullanabilirsiniz:

docker volume ls

Volume'ü kullanan konteynerleri durdurun ve silin:

docker ps -a --filter volume=<volume\_adı>

Bu komut, belirtilen volume'ü kullanan konteynerleri listeleyecektir. Konteynerleri durdurmak ve silmek için:

docker stop <konteyner\_adı>

docker rm <konteyner\_adı>

Konteynerler durdurulup silindikten sonra volume'ü silmek için:

docker volume rm <volume\_adı>

Docker'da bir imajı silmek için şu komutu kullanabilirsiniz:

docker rmi <image\_adı>

İmajları listelemek için şu komutu kullanabilirsiniz:

docker images

Belirli bir imajın üzerinde kullanılmadığından emin olduktan sonra silmek için:

docker rmi <image\_adı>

Birden fazla imajı aynı anda silmek isterseniz, imaj adlarını veya ID'lerini aralarına boşluk koyarak belirtebilirsiniz:

docker rmi <image\_adı1> <image\_adı2>

Eğer imajın silinmesini zorlamak istiyorsanız ve imaj kullanılmıyorsa, -f (force) parametresini ekleyebilirsiniz:

docker rmi -f <image\_adı>

Docker Hub veya belirttiğiniz bir Docker registry'den imajı indirir. İmajı çekmek için aşağıdaki komutu kullanabilirsiniz:

docker pull <image\_adı>

Belirli bir etiket (tag) ile imaj çekmek istiyorsanız, etiket adı ile birlikte kullanabilirsiniz. Örneğin, nginx imajının alpine etiketli sürümünü çekmek için:

docker pull nginx:alpine

Docker'da bir konteyneri arka planda çalıştırmak (detach mode) için -d (detach) bayrağını kullanarak konteyneri oluşturabilirsiniz. Aşağıdaki komut, bir konteyneri arka planda çalıştırmak için kullanılır:

docker run -d <image\_adı>

Bu komut, nginx imajını arka planda çalıştırırken, konteynerin 80 numaralı portunu ana makinenin 8080 numaralı portuna yönlendirir.

docker run -d -p 8080:80 nginx

docker exec komutu ile konteyner içinde bir terminal açabilirsiniz. Genellikle /bin/bash veya /bin/sh shell'ini kullanırsınız:

docker exec -it <konteyner\_adı\_veya\_id> /bin/bash

veya daha hafif bir shell olan /bin/sh:

docker exec -it <konteyner\_adı\_veya\_id> /bin/sh

 -i: Interaktif mod, yani kullanıcı girdisini alır.

 -t: Terminal tipinde bir ortam sağlar.

 <konteyner\_adı\_veya\_id>: Bağlanmak istediğiniz konteynerin adı veya ID'si.

Örneğin, my-container adlı bir konteynerde bash shell açmak için:

docker exec -it my-container /bin/bash

docker container exec –it websunucu sh -> web sunucu isimli conatiner’a bağlandık

docker rm –f ${docker ps –qa} tüm containerları siler

docker ps –aq -> tüm container iamjlarını verir

docker attach b3(container id ) -> çalışan container’a bağlanır

ctrl+p+q -> containerdan geçici olarak çıkar

Docker Network :

Docker network ls

Docker inspect bridge [network\_name]

İnspect => detaylı bilgi

Ctrl p+q => containerla bağlantıyı keser

Bridge network : birden fazla ağdan tek bir birleşik ağ yapmaya yarıyor.Bridge network makine üzerinde docker0 adında sanal network interface yaratı ve onada 172.17.0.1/16 ip adresini verir. Biz yeni bir container oluştuduğumuzda network vermezsek var sayılan olarak bu network atanır. Docker bu container 172.0.0.1/16 network’ünden sıraki ıp ‘yi atar.Gateway olarak da docker0 interface’nin ip adresini verir.Container dış dünya ile haberleşeceği zaman paketi buraya teslim eder. Daha sonra bu paket ana esas network interface’sine aktarılır ve nereye gidecekse gider. Bridge network’deki bir container’ın dışarın bağlanılabilmesi için dışarıya açık olaması lazım bu durmda port publish yapılır.

Eğer kullanıcı tanımlı bir bridge networke bağlığı olduğu için isimler üzerinden haberleşebilirler başka bir ifade ile basit dns hizmeti alınır.

-p host\_port : container\_port

Host network : buna bağlı containerların hiçbir network izolasyonu yoktur çalıştıkları sistemin ağ alt yapısını kullanırlar.

None network : container oluşturup herhangi ağa bağlanması istmediğimz durumlarda kullanılır.

Not: aynı bridge networkte bulunan containler ip üzerinden haberleşebilir ancak isim üzerinden haberleşemez çünkü dns çözücü yoktur.

Kendi bridge networkümüzü oluştumak istersek :

Docker network create (isim)

docker container run –dit = -d –it birleşim direkt –dit olarak kullanıdık .

docker container run -dit --name websunucu --net kopru1 ozgurozturknet/adanzyedocker sh => bu komutta –net ile istedğimiz networku belirtiyoruz. Dit (git containerı oluştur onunla bağlantıyı kur ama detec olarak)

Docker attach (container\_name) => ile containerla attach olabiliriz.

docker network create --driver=bridge --subnet=10.10.0.0/16 --ip-range=10.10.10.0/24 --gateway=10.10.10.10 kopru2 =>(ip-range= subnet içinde dağıtacağı ip aralığı ne olacak onu belirler)

kullanıcı tanımlı bridge networklere önemli bir özelliği de containler çalışırken bağlanabilirler komutu ;

docker network connect kopru2 (container name yada id)

docker network disconnect kopru2 (container name yada id) ile de bağlantıyı kesebiliriz

not : bir containerı birden fazla networke de bağlayabiliriz

network silme : docker network rm kopru2

not : üzerinde container varsa network silinmez ilk önce container silinme sonra network silinebilir

docker logging :

STDIN-STOUT-STDERR

Stdın=Klavyeden giriş içeren uygulamanın giriş akışıdır.

Stdout=genellikle bir uygulamanın normal çıktısıdır.

Stderr= hata mesajı göndermek için kullanılan çıktıdır.

Docker Logs

Docker logs [id]

Docker logs –details con1 (daha detaylı logları gösterir)

Not: bazen logalara zaman damgası vurulmaz bu gibi durumlarda –t opsiyonunu kullanırız.

Docker logs –t con1

Log doslarının içinde belirli bir zamana uygun filitreleme yapmak istersek

docker logs --until 2024-09-25T13:22:38.729691482Z con1

bu komutta logları göster ama zaman damgası belirtiğimiz yere kadar göster.

Eğer belirtilen saatten sonraki logları görmek istiyorsak

Docker logs –-since 2024-09-25T13:22:38.729691482Z con1

Son oluşturulmuş belirtilen sayıda logu görmek istiyorsak

docker logs --tail 3 con1

şuana kadar olan loğlar ve canlı olarak o anda oluşan logları görmek istiyorsak

docker logs -f con1 (-f = follow)

docker container run –log-driver splunk nginx -> splunk driverı ile tüm loglar spunkun merkezi sunucusuna yölendiriyoruk ve daha gelişim işlemler yapabilirdik

Not: varsayılan olarak json-file driverı kullanılır.

Docker stats-top :

Containera girmede o containerda hangi processler çalıltığını görmemizi sağlar:

docker top con1

containerların ne kadar cpu ve memory kullanığını network ıo ne durumda alınlık olarak bilgi sahibi oluruz:

docker stats [id yada ad ] eğer isim belirtmezek sistemdiki tüm containerlar hakkında bilgi verir.

Container cpu ve memory limitleri:

Container oluşturuken ne kadar cpu memory kullanmasını sınırlayabiliriz eğer bunu yapmazsak her container varsayılan olarak sistem kaynaklarına sınırsız şekilde erişebilir. Bu yapmamız önemlidir.

docker container run -d --memory=100m ozgurozturknet/adanzyedocker

eğer bir aşım olursa containerın durmaması için başka belirtilen alan kadar kullanım yapabilir:

docker container run -d --memory=100m --memory-swap=200m ozgurozturknet/adanzyedocker

not : b=byte m=mb g=gb anlamındadır.

Cpu -> core olarak bölümleye biliriz. Ayarlamaları karışık olduğundan bu şeklide yapıyoruz.

docker container run -d --cpus="1.5" ozgurozzturknet/adanzyedocker -> sistemde ne kadar core varsa 1.5 tanesini kullancak total logical processing’in 1,5 gücü

docker contaiener run -d --cpus="1.5" --cpuset-cpu="0,3" ozgurozzturknet/adanzyedocker -> cpu 0 ve 3 kullan git sadece cpu 0 cpu3 kullanabil. Cpu kısıtlama yapmış oluyoruz.

Enviroment Variables:

Sistem bazında geçerli olan ve heryerden çağırılan değişkenlerdir.

get-childItem Env: sistemdeki tüm enviroment variabler listeler.

Belirttiğimiz bir değeri görmek istersek :$Env: ve değeri giriyoruz

$Env:windir

Kendimiz bir enviroment varible tanımlamak istersek :

$Env:test="denemedir"

Linux da ise :

Tüm envler : printenv

Belirtiğimiz bir taneyi çağırmak istersek : echo $Term

Biz tanımlamak istersek : export test=”denemedir”

Env’ler

İmaj yada containerler oluştururken tanımlanır.

Env oluştururken büyük küçük uyumuna dikkat etmek gerekir.

docker container run -t --env VAR1=deneme1 --env VAR2=deneme2 ubuntu bash

bu şekilde var1 ve var2 adında iki tane env tanımladık .

çalışıtığı host üzerindende bir env tanımlayıp containerdan erişilebilir yapabiliriz:

docker container run -t –env TEMP ubuntu bash

burada görüldüğü sistem üzerinde olan temp adında bir env oluşturuk bir bir değer vermedik otomatik sistem üzerinden alacak.

Bir diğer yöntem eğer tanımlamamız gereken birden fazla env varsa :

docker container run -t –env-file .\env.list ubuntu bash

(--env-file dosya\_ismi)

bir dosya yolunu ve adını verirsek dosyayı alır ve dosya içindeki bütün değerleri alıp env yapıcak ve containera set edecek.

Image ve Registry

Dokcer image pull ozgurozturknet/adanzyedocker -> image çektik.

Default tag latest’tir

Docker file : docker imajları docker file dediğimiz talimatlardan oluşur.imaj oluşturmak için öncelikle docker file oluşturup içine talimatları yazar sonrada bu dosyadan docker imajı oluşutururuz.

Docker iamge pull grc.io/Google-conatiners/busybox => google’in iamge registerisinden imaj çektik

Docker file Oluşturma :

**From :** Oluşturulacak imajın hangi iamajdan oluşturulacağını belirten talimat dockerfile içerinde geçmesi mecburi tek talimat budur.mutlaka olamalıdır.

Ör: From ubuntu:18.04

**Run:**imajdan oluşturulurken shell’de bir komut çalıştırmak istersek bu talimatı kullanılır.Örneğin apt-get-install xxx ile xxx isimli uygulamasının bu imaja yüklenmesi sağlanabilir.

Ör: RUN apt-get update

**Workdir:** cd xxx komutuyla ile istediğimiz klasöre geçmek yerine bu taliamtı kullanarak istediğimiz klasöre geçer ve oradan çalışmaya devam ederiz

Ör: WORKDIR /usr/src/app

**Copy:**imaj içine dosya klasr kopyalamak için kullanırız.

Ör: COPY/source/user/src/app

**Expose:**Bu imajdan oluşturulacak containerların hangi portlar üstünden erişilebileceğini yani hangi portların yayınlanacağını bu talimatl belirtiriz.

Ör: EXPOSE 80/tcp

**Cmd:** Bu imajdan container oluşturulduğu zaman varsayılan olarak çalıştırılması istediğimiz komutu bu talimat ile belirlersiniz.

Ör:CMD java hello

**Healthcheck :**Bu talimat ile docker’a bir containerın hala çalışıp çalışmadığını kontrol etmesini söyleyebiliriz.Docker varsayılan olarak container içerinde çalışan ilk processi izler ve o çalıştığı sürece container çalışamaya devam eder.Fakat process çalışsa bile onun düzgün işlem yapıp yapmadığını bakmaz. HEALTHCHECK ile buna bakabilme imkanına kavuşuruz.

Ör: HEALTHCHECK --interval=5m –timeout=3s CMD curl -f <http://localhost/||> exit 1

Örnek : Docker file oluşturma

EntryPoint ve Cmd Farkı :

* **ENTRYPOINT**: Konteynerin her zaman çalıştırması gereken ana komutu belirler. Değiştirilemez (ancak --entrypoint ile override edilebilir).
* **CMD**: Varsayılan komut veya argümanları belirler. Ancak docker run sırasında farklı bir komut verilirse geçersiz olur.

Ör :

CMD Örneği (Geçersiz Olabilir) =>

dockerfile :

FROM ubuntu

CMD ["echo", "Merhaba!"]

Sh:

docker run myimage # Çıktı: Merhaba!

docker run myimage ls # Çıktı: (CMD geçersiz olur, "ls" çalışır)

ENTRYPOINT Örneği (Değiştirilemez) =>  
df:

FROM ubuntu

ENTRYPOINT ["echo"]

Sh:

docker run myimage Merhaba! # Çıktı: Merhaba!

docker run myimage ls # Çıktı: ls (echo'nun argümanı olarak geçer)

 **ENTRYPOINT**: Komut zorunlu, her zaman çalışır.

 **CMD**: Varsayılan komuttur, ancak değiştirilebilir.

Not :

1. Her iki Docker imajında bir CMD veya ENTRYPOINT talimatı bulunmalıdır.
2. Her iki talimat da bu imajdan container oluşturulduğunda çalıştırılacak uygulmayı belirtmemizi sağlar.
3. ENTRYPOINT iler girilen komut runtime’da yani container çalışırken değiştirilmez CMD ile yazılan ise runtime’da değiştirilebilir.
4. ENTRYPOINT ve CMD aynı anda kullanılırsa CMD yazılan ENTRYPOINT talimatında yazılana parametre olarak eklenir.

EXEL ve SHELL Form Farkı :

Exec Form Shell Form

* CMD ["java", "uygulama"] CMD java uygulama

1. Eğer komut Shell formunda girilirse, Docker bu imajdan container yaratıldığı zaman bu komutu varsayılan shell’i çalıştırarak onun içerisinde işler. Bu nedenle container’da çalışan 1. process yani pid 1 bu shell process’i olur.
2. Eğer komut Exec formunda girildiyse, Docker herhangi bir shell çalıştırmaz ve komut direkt process olarak çalışır ve container’ın pid 1’i o process olur.
3. Exec formunda çalıştırılan komutlar herhangi bir shell process’i çalışmadığı için **Environment Variable** gibi bazı değerlere erişemez. Bunu göz önünde bulundurmak gerekir.
4. Eğer **Entrypoint** ve **CMD** birlikte kullanılacaksa, Exec form kullanılmalıdır. Shell formu kullanıldığında, CMD’deki komutlar **Entrypoint**'e parametre olarak aktarılmaz.