|  |  |  |  |
| --- | --- | --- | --- |
| **Container Logs** | | | |
| **docker logs [container ismi]** | Container ın loglarını göster | | |
| **docker logs--untill [zaman damgası]** | Şu saate kadar oluşan logları göster | | |
| **docker logs --since [zaman damgası]** | Şu saatten sonra olan logları göster | | |
| **docker logs --tail [satır sayısı] [container ismi]** | Yazılan son satır sayısının loglarını gösterir | | |
| **docker logs --f con1** | Canlı olarak logları gösterir, log akışı devam eder. | | |
| **docker info** | Logging driverları gösterir | | |
| **docker container run –log-driver [driver ismi] [image ismi]** | Logların yazılacağı formatını değiştirmek | | |
| **Docker Stats ve Top** | | | |
| **docker top [container ismi]** | Container içine girmeden hangi process ler çalışıyor | | |
| **docker stats** | Tüm containerların memory/cpu/storage durumlarını gösterir | | |
| **docker stats [container adı]** | Containerın memory/cpu/storage durumlarını gösterir | | |
| **Container cpu ve Memory Limitleri** | | | |
| **docker run -d –memory=100[m,g,k,b] [image]** | Container ın kullanacağı max memory limit(100) | | |
| **docker run -d –memory=100[m,g,k,b] --memory-swap=200[m,g,k,b] [image]** | Container ın kullanacağı max memory limiti(100) aşarsa 200 mb a kadar swap alanı kullanma opsinoyu verdim | | |
| **docker container run -d --cpus=”[1,5]” [image]** | Container core ların 1,5 tanesini kullan | | |
| **docker container run -d --cpus=”[1,5]” --cpuset-cpus=”[0,3]” [image]** | Container core ların 0 ve 3 numaralı core u kullan | | |
| **Environment Variables** | | | |
| **det-childItem Env:** | (Windowsda)Sistemdeki tüm env ları göster | | |
| **$Env:[env]** | (Windowsda)Belli bir env değerini gösterir | | |
| **$Env:[env]=”[değer]”** | (Windowsda)Env a değer atamak | | |
| **printenv** | (Linux)Sistemdeki tüm env ları göster | | |
| **echo $[env]** | (Linux) Belli bir env değerini gösterir | | |
| **export [env]= ”[değer]”** | (Linux)Env a değer atamak | | |
| **Docker Enviroment Variables** | | | |
| **docker container run -it --env [variable]=degeri [image]** | Docker containerda değişken yarat | | |
| **docker container run -it -e[variable]=degeri [image]** | Docker containerda değişken yarat | | |
| **docker container run -it --env [host env] [image]** | Host makine üzerindeki env değerini conatiner a atamak | | |
| **docker container run -it --env-file [dosya\_ismi] [image]** | Dosya içindeki tüm değerleri env olarak tanımlar ve container a set etmek | | |
| **docker container run --env [env\_ismi]=[deger] [image]** | Container içinde yaratılmış bir env ı degerini değiştirerek kullanabilmek: Örneğin testveritabanı olan veri tabanı ismimi prod olarak değiştirerek prod veri tabanımda kullabiliyorum  Docker container run --env veritabanı=testveritabanı.pizzadukkanı.com ozgurozturknet/env1  Docker container run --env veritabanı=prod.pizzadukkanı.com ozgurozturknet/env1 | | |
| **Docker Image Isımlendirme ve Tag Yapısı** | | | |
| **docker image pull [repositry]** | Sonuna bir şey eklemezsek Default olarak tag sonuna :latest ekleyerek yaratıyor | | |
| **docker pul [repositry]:[tag]** | Image ın tag ını değiştirerek istediğimiz version u seçebiliriz | | |
| **Docker Image Oluşturma (Temel Manuel)** | | | |
| **FROM [imaj]:[tag]** | From ile oluşturulacak olan imajın hangi imajdan oluşturulacağını belirten talimat. | | |
| **RUN komut** | İmaj oluşturulurken shell’de bir komut çalıştırmak istersek bu talimat kullanılır. Örneğin apt-get install xxx ile xxx isimli uygulamanın bu imaja yüklenmesi sağlanabilir. Örneğin RUN apt-get update | | |
| **WORKDIR klasör\_lokasyonu** | cd xxx komutuyla istediğimiz klasöre geçmek yerine bu talimat kullanılarak istediğimiz klasöre geçer ve oradan çalışmaya devam ederiz. | | |
| **COPY kaynak\_hedef** | İmaj içine dosya veya klasör kopyalamak için kullanırız. | | |
| **EXPOSE port** | Bu imajdan oluşturulacak containerların hangi portlar üstünden erişilebileceğini yani hangi portların yayınlanacağını bu talimatla belirtirsiniz. | | |
| **CMD komut** | Bu imajdan container yaratıldığı zaman varsayılan olarak çalıştırılmasını istediğiniz komutu bu talimat ile belirlersiniz  Örneğin: CMD java merhaba | | |
| **HEALTCHECK komut** | Bu talimat ile docker a bir konteynerin hala çalışıp çalışmadığını kontrol etmesini söyleyebiliriz. Docker varsayılan olarak container içerisinde çalışılan ilk processi izler ve o çalıştığı sürece container çalışmaya devam eder. Fakat process çalışsa bile onun düzgün işlem yapıp yapmadığına bakmaz. HEALTHCHECK ile buna bakabilme imkanına kavuşuruz.  Örneğin: HEALTHCHECK –interval=5m –timeout=3s curl –f http://localhost/||exit1 | | |
| **FROM ubuntu:18.04**  **COPY . /app**  **RUN make/app**  **CMD python/app/app.py** | Ubuntu base olarak kullanılıyor. Sonra python ile yazılmış app klasörüne copyalanıyor. Make komutu ile uygulama derleniyor. Sonra container başlatıldıktan sonra bu uygulamanın başlatılması için talimat veriliyor. | | |
|  | Docker File oluşturmak.  Uygulamanın olduğu klasör içine gelip yeni bir file yaratıyoruz. “Dockerfile” isimli bir file yaratıyoruz. Bu isimle olacak. Bu file ın içine giriyoruz ve adımları tek tek uyguluyoruz.  Image yaratıyoruz. İşletim sistemi kur.  1.Adım: FROM ubuntu:18.04  İşletim sistemini güncelle  2.Adım: RUN apt-get update –y  Java Env sisteme gir  3.Adım: RUN apt-get install default-jre –y  Uygulamayı sisteme kopyala. WORKDIR ile Merhaba isimli klasöre geçiş yaptım. COPY ile /myapp dosyasındaki uygulamaları bulunduğumuz klasöre kopyala.  4. Adım:  WORKDIR/merhaba  COPY /myapp .  Sistem başlatıldığında uygulamanın başlatılması için ayar yap. CMD yapıyoruz ve uygulamanın çalışması için hangi komut olduğunu docker formatında giriyorum. Java boşluk merhaba komutunu gir demek.  5.Adım: CMD [“java”, “merhaba”] | | |
| **docker image build –t [tag adı] –f [docker file ismi] .** | Docker file dan bir image yaratmak için.  Build diyorum image oluşturmak için. –t diyorum tag atamak için ve tag ın adını giriyorum. Sonra –f ile docker file in ismini giriyorum. İsmi docker file ise Dockerfile ın bulunduğu klasörde çalışıyorsak –f ye gerek yok. Sonuna “.” Girersek build contexin burası dockerfile bir komut sorarsa burada ara diyoruz. | | |
| **docker image history [image adı]** | İmage ın layerları görünüyor | | |
| **docker image push [image adı]** | Docker hub a update etmek | | |
| **Docker Imga Oluşturmak-2** | | | |
|  | Image oluştururken java, python gibi prog yüklemek yerine bunların yüklenmiş olduğu resmi image ları çekmek gerekir. Bu adımların zaten daha ayrıntılısını resmi image ında yapmışlar güvenlik açıkları vs kapatmışlar.  Bir uygulamanın resmi image ını docker hubdan öncelikli olarak bulmak gerekir. | | |
| **FROM python:alpine**  **COPY . /app**  **WORKDIR /app**  **RUN install –r requirements.txt**  **EXPOSE 5000**  **CMD python ./index.py** | Docker file.  Expose dışardan 5000 portu ile vu container a bağlanılabilsin. | | |
| |  | | --- | | **FROM node:10** | | **# Create app directory** | | **WORKDIR /usr/src/app** | | **# Install app dependencies** | | **# A wildcard is used to ensure both package.json AND package-lock.json are copied** | | **# where available (npm@5+)** | | **COPY package\*.json ./** | | **RUN npm install** | | **# If you are building your code for production** | | **# RUN npm ci --only=production** | | **# Bundle app source** | | **COPY server.js .** | | **EXPOSE 8080** | | **CMD [ "node", "server.js" ]** | | Package.json dosyasında bu dosya içinde temel meta data bilgieleri, yadığımız uygulamanın bağımlı olduğu modülleri tanımlarız. Json dosyasındaki Depencies: uygulamanın bağlı olduğu modülleri gösterir.  Server.js dosyası da java dilinde yazılmış web sitesi uygulamasıdır.  Bu uygulamyı image haline getirmek için :  Form ile Node:10 image ını çekiyoruz.  Workdir ile app klasörüne geç diyoruz.  Daha sonra bu klasörün içindeki json ile biten ve packaage ile başlayan tüm dosyaları buraya at.  RUN ile npm install ile packaege ile bize söylenilen depencies leri kuruyoruz.  COPY ile server.js . dosyasını kopyalıyoruz.  EXPOSE ile 8080 portunu expose ediyoruz.  CMD komutunun çalışmasını sağlıyoruz. | | |
| # Bu örnek uygulama https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md adresinden alınmıştır  # base imaj olarak nginx resmi imajını kullanıyoruz --  # https://github.com/nginxinc/docker-nginx/blob/5971de30c487356d5d2a2e1a79e02b2612f9a72f/mainline/buster/Dockerfile  **FROM nginx:latest**  # label ile de key value pair olarak bu imajla ilgili bilgileri giriyoruz. her iki talimat da imajın içindeki sistemle alakalı değil.  # her ikisi de imajın metadatasında saklanıyor. imajla alakali bilgiler.  **LABEL maintainer="Ozgur Ozturk @ozgurozturknet"**  **LABEL version="1.0"**  **LABEL name="hello-docker"**  # bir adet env variable tanımlıyoruz. Daha sonra bunu runtime'da değiştireceğiz  **ENV KULLANICI="Dunyali"**  # nginx debian-slim imajı baz alınarak oluşturulmuştur.  # bu imajda geriye doğru uyumluluk sorunları var ve o nedenle update olmuyor.  #Bu url'leri sources.list'e ekleyerek update olabilmesini sağlıyoruz  **RUN printf "deb http://archive.debian.org/debian/ jessie main\ndeb-src http://archive.debian.org/debian/ jessie main\ndeb http://security.debian.org jessie/updates main\ndeb-src http://security.debian.org jessie/updates main" > /etc/apt/sources.list**  # Sisteme güncellemeleri geçiyoruz  **RUN apt-get update**  # debian-slim'de bir çok tool mevcut değil. Bizim curl kullanmamız gerektiği için curl kuruyoruz  **RUN apt-get install curl -y**  # nginx web sayfalarını /usr/share/nginx/html folder'ında barındırıyor. O nedenle bu folder'a geçiyoruz  **WORKDIR /usr/share/nginx/html**  # web sitemizin açılış sayfası olan Hello\_docker.html dosyasını buraya kopyalıyoruz.  **COPY Hello\_docker.html /usr/share/nginx/html**  # sistemin düzgün çalıştığını ve nginx daemon'ının web sitesini publish etmekte bir sorun yaşamadığını test ediyoruz  **HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 CMD curl -f http://localhost/ || exit 1**  # bu imajdan container yaratıldığı zaman çalışmasını istediğimiz komutu buraya giriyoruz  # /usr/share/nginx/html klasörüne geçiyor ve sed aracılığıyla Hello\_docker.html dosyasının içerisindeki  # Kullanici kelimesini $KULLANICI env variable'ının değeriyle Hostname kelimesini de $HOSTNAME ile değiştiriyor ve  # dosyanın içeriğini index.html'e adlı yeni bir dosyaya yazıyoruz.  # ardından nginx daemon'ı çalıştırıyoruz  **CMD sed -e s/Kullanici/"$KULLANICI"/ Hello\_docker.html > index1.html && sed -e s/Hostname/"$HOSTNAME"/ index1.html > index.html ; rm index1.html Hello\_docker.html; nginx -g 'daemon off;'** | Adımlarda değişiklik yapıldığı zaman cash den kullanılmasını istiyorsak değiştirilmeyecek olan dosyaları mümkün olduğunca ist tarafa almak gerekir ki her seferinde tekrar tekrar indirme olmasın. Bir adımda değiştirilen bir dosyanın tekrar build edilemsi demek o değişitirilen adımdan sonraki tüm adımların yeniden build edilmesi demek.  İnterval=30s: Ne kadar aralıklarla kontrol yapmak istediğimizi belirliyoruz.  Timeout: 10s içinde sonuç dönmezse fail olarak belirle  Start-period=5s: healtcheck i 5s olarak belirle hemen başlama yoksa fail verebilir  Retries=3: 3 deneme sonucunda olumlus olursa unhealthy olarak belirlesin  CMD curl -f <http://localhost/>: Bu komutun çalışıp çalışmadığını check ediyoruz işte asıl olay kontrol ettiğimiz komut bu.  || exit 1: eğer ilk komut çalışmazsa exit 1 komutunu veriyor yani hata komutu üret anlamına geliyor yani komutu öldürüyorum  **HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 CMD curl -f http://localhost/ || exit 1**  Burada 4 komut aynı anda çalışıyor.semi columnlarla birlikte.  Bu ilk komut ile hello docker html dosyasını tara burada kullanıcı kelimesi görürsen bunu KULLANICI env değer ile değiştir ve bunu index1.html dosyasına kaydet. Sonra and and giriyorum ve bunun içinde hostname arat hostname in değerini hostname in env değeri ile değiştir ve index.html doyasının içine kaydet  **CMD sed -e s/Kullanici/"$KULLANICI"/ Hello\_docker.html > index1.html && sed -e s/Hostname/"$HOSTNAME"/ index1.html > index.html**  İhtiyacım olmana index ve hellod docker dosyalarını siliyorum ve son olarak da nginx –g ile daemon off komutunu çalıştırıyorum. Daemon off komunutunu çalıştır.  **rm index1.html Hello\_docker.html; nginx -g 'daemon off;'**  Bu komutların hepsini tek komutta çalıştırmamın sebebi. Image oluşturma aşamasında yapmak istemediğim için. Eger öncesinde run komutu ile yapsaydım son çalıştırılan containerın host nameine o ismi vermemiş olurdu. İmage a o ismi vermiş olurdu.  Komutları işlerken her seferinden container oluşturarak diğer komuta geçiyor. Öncesinde run komutu ile oluştursam image dan container oluşturulduktan sonra hostname verilmesini istiyorum. | | |
| #labelların 3 tanesini tek bir talimatta birleştiriyorum  **LABEL maintainer="Ozgur Ozturk @ozgurozturknet" version="1.0" name="hello-docker"**  #3 run talimatı nın arasına && koyarak tek talimata indirdim. Daha düzgün görünmesi için back slash “\” koyabilirim.  **RUN printf "deb http://archive.debian.org/debian/ jessie main\ndeb-src http://archive.debian.org/debian/ jessie main\ndeb** [**http://security.debian.\**](http://security.debian.\)  **org jessie/updates main\ndeb-src http://security.debian.org jessie/updates main" \**  **> /etc/apt/sources.list && apt-get update && apt-get install curl -y**  **#** Daha düzgün görünmesi için back slash \ koyabilirim.  **CMD sed -e s/Kullanici/"$KULLANICI"/ Hello\_docker.html > index1.html && sed -e s/Hostname/"$HOSTNAME"/ index1.html > index.html ; \**  **rm index1.html Hello\_docker.html; nginx -g 'daemon off;'** | Oluşturulan image ın 12 komutu var. Bu komutlar her biri ayrı layerlar demek. Bunu azaltma için bir takım işlemler yapmak istiyorum.  Labelları birleştiriyorum | | |
| **docker container run –d –p 80:80 --name hd2 –e KULLANICI=”Sezgin ERDEM” ozgurozturknet/hello-world** | KULLANICI env ön tanımlı olarak merhaba dünyalı iken bu komutla onu değiştirebiliyorum  Expose komutu koymadım ama 8080 portundan bu container a giriş yapabiliyorum peki bu nasıl oluyor? Nginx base imajdan geçer. Tek istisnası bunu manuel olarak overwrite etmek gerekir | | |
| **ADD VE COPY TALİMATLARI** | | | |
| **ADD** | Copy ile aynı işi yapar yanı dosya ya da klasör kopyalarsınız. Fakat ADD bunun yanında dosya kaynağının bir url olmasına da izin verir. Ayrıca ADD ile kaynak olarak bir .tar dosyası belirtilirse bu dosya imaja.tar olarak sıkıştırılmış haliyle değil de açılarak kopyalanır. Ör. ADD https://wordpress.org/latest.tar.gz./temp  Uzaktaki bir web sunucundan kopyalamak için web kullanmak gerekir ama başka bir yerden çekerse sıkıştırılmış haliyle kalır açmaz. | | |
| **ENTRYPOINT VE CMD FARKI** | | | |
| **ENTRYPOINT** | Entrypoint runtim da değiştirilmez. İmaj yaratırken cmd yazacağıma entrypoint ile imajı oluştururken dockerfile içinde. Entrypoint ile oluşturulan container da runtime da yeni bir komut çalıştır dediğimizde değiştirmez.  Mesela docker run container ls dediğimizde ls komutunu çalıştırmaz.  Dockerfile içinde hem entrypoint hem cmd olursa ne olur? CMD içinde yazılanları entrypoint içine parametre olarak ekler.  Mesela bir dockerfile yazalım  FROM centos:latest  ENTRYPOINT [“ping”, “127.0.0.1”]  Bundan docker image build –t pingimaj –f .\Dockerfile.Centos #yarattık  Docker run pingimaj #container yarattık  FROM centos:latest  ENTRYPOINT [“ping”]  CMD [“127.0.0.1”] yazarsam burada da ping komutunun arkasına 127.0.0.1 komutu çalışacak demek, parametre ataması o demek.  Yeniden imaj oluştuğumda herhangir bir şey değişmemiş gibi oldu. Fakat bu ne işimize yarar CMD de girilen runtime da değiştirilebilir.ama  Docker run pingimaj 8.8.8.8 #diye yaratırsam bunu pingler | | |
|  | 1. Her docker imajında bir CMD veya ENTRYPOINT talimatı bulunmalıdır  2. Her iki talimat da bu imajdan container yaratıldığında çalıştırılacak uygulamayı belirtmemizi sağlar.  3. ENTRYPOINT ile girilen komut runtime’da yani container çalıştırılırken değiştirilemez. CMD ile yazılan ise runtime’da değiştirilebilir  4. ENTRYPOINT ve CMD aynı anda kullanılırsa CMD’de yazılan ENTRYPOINT talimatında yazılana parametre olarak eklenir. | | |
| **EXEC ve SHELL Form farkı** | | | |
| #Exec form  **CMD [“java”, “uygulama”]**  #Shell Form  **CMD java uygulama** | 1. Eğer komut shell formunda girilirse Docker bu imajdan container yaratıldığı zaman bu komutu varsayılan shell’i çalıştırarak onun içerisinde girer. Bu nedenle containerda çalışan 1. Process yani pid1 bu shell process’i olur.  2. Eğer komut Exec formunda girildiyse Docker herhangi bir shell çalıştırmaz ve komut direk process olarak çalışır ve container’ın pid1’i o process olur  3. Exec formunda çalıştırılan komutlar herhangi bri shell processi çalıştırmadığı için Environment Variable gibi bazı değerlere erişemezler. Bunu göz önünde bulundurmak gerekir.  4. Eğer Entrypoint ve CMD birlikte kullanılacaksa Exec form kullanılmaldır. Shell formu kullanıldığında CMD’deki komutlar ENTRYPOINT’e parametre olarak aktarılmaz. | | |
| **FORM centos:latest**  **ENV TEST=”bu bir denemedir”**  #exec formu  **CMD [“echo”, “$TEST”]** | Cmd exec formunda yazılmıştır  Bu docker file dan yaratılan bir containerda ortam değişkenleri gibi şeyleri okuyamaz ve “$TEST” yazısını ekrana verir. | | |
| **FORM centos:latest**  **ENV TEST=”bu bir denemedir”**  #exec formu  **CMD echo $TEST** | Cmd shell formundan yazılan bir docker file ile oluşturulan containerdaki değişkenleri ekranda “bu bir denemedir” olarak verir. Çünkü shell formunda bu komut bize container içerisinde bir shell çalıştır ve o shell in içerisinde bu komutu gir demek. Shell i çalıştırmadığın zaman erişemiyorsun.  Özellikle bazı durumlarda yanlış komutu girmek enviroment ları çekemeyecek anlamına gelmektedir. | | |
| **MultiStage Build** | | | |
| **FROM mcr.microsoft.com/java/jdk:8-zulu-alpine**  **COPY /source/usr/src/uygulama**  **WORKDIR /usr/src/uygulama**  **RUN javac uygulama.java**  **CMD [“java”, “uygulama”]**  Docker container run –name javauygulama javasdk #container yarattım  Docker cp java uygulama:/usr/src/uygulama . #kopyaladım  cd .\uygulama #ile bu uygulamanın içine giriyorum  #yeni bir docker file yaratıyorum  **FROM mcr.microsoft.com/java/jre:8-zulu-alpine #sadece jre kurmanın benim için yeterli jdk gereksiz şeyler var**  **COPY /uygulama /uygulama**  **WORKDIR /uygulama**  **RUN javac uygulama.java**  **CMD [“java”, “uygulama”]**  **docker image build –t javajre –f .\Dockerfile.jre .** #build ettim böylelikle mb ı yarıyarıya düştü sdk gibi bütün dosyalar yok. Gereksizleri yüklemedim. | Image ın boyutunu küçülmek için docker file dan bir java uygulaması build ettim ama bunun mb çok yüksek. | | |
| **FROM mcr.microsoft.com/java/jdk:8-zulu-alpine AS derleyici #sadece jre kurmanın benim için yeterli jdk gereksiz şeyler var**  **COPY /uygulama /uygulama**  **WORKDIR /uygulama**  **RUN javac uygulama.java**  **FROM mcr.microsoft.com/java/jre:8-zulu-alpine #ikinci bir from komutu gidiyorum ama bunu jre den başlatmak istiyorum**  **WORKDIR /uygulama**  **COPY --from=derleyici /usr/src/uygulama . #bir üstteki dosyaların stage ı buraya kopyalayabiliyorum**  **CMD [“java”, “uygulama”]** | Bu imajı automatize etmek istiyorum. Bütün yukarıdaki aşamaları manuel yapmak istemiyorum.  #Bu komutlarla build ettiğimizde çok aşamalı bir docker image yarattığımızda yaratılan image son from talimatı ile girilen imagelardan oluşur. Bundan önceki aşamaların tek amacı son aşamadan önceki image lara kopyalar oluşturmaktır. Son aşama image lara dahil edilmezler. | | |
| **ARG** | | | |
| **FROM ubuntu:latest**  **WORKDIR /gecici**  **ADD https://www.python.org/ftp/python/3.7.6/Python-3.7.6.tgz .**  **CMD ls –al**  **Docker image build –t uygulama:3.7.6 .**  **FROM ubuntu:latest**  **WORKDIR /gecici**  **ADD VERSION**  **ADD https://www.python.org/ftp/python/$(VERSION)/Python-$(VERSION).tgz .**  **CMD ls –al**  **Docker image build –t x1 –build-arg VERSION=3.7.1**  **Docker image build –t x1 –build-arg VERSION=3.8.1** | Bu file dan bir imaj yaratıyoruz.  Ancak bir de bu uygulamanın 3.8.1 versiyonunu yaratmak istiyorum.  ARG komutu ile VERSION adında bir değişken atıyorum. Bu ARG komutu ile dockerfile içinde değişken oluşturabiliyorum. Bunu ne gibi durumlarda kullanabilirim: farklı version kurmak istediğimde kulllanabilirim. Eğer varsayılan değer olarak VERSION=3.7 atarsam bir değer girmediğimde 3.7 versionunu build eder. | | |
| **Docker Commit** | | | |
| **Docker container run –it --name con1 ubuntu:latest bash**  **Apt-get update #içine girdim güncelledim**  **Mkdir temp/ #tempe gittim**  **Cd /temp**  **Apt-get install wget #wget i kurdum**  **Wget** [**https://wordpress.org/latest.tar.gz**](https://wordpress.org/latest.tar.gz) **#dosya çektim**  **Exit**  **Docker commit con1 ozguruzturknet/con1:latest # ozgurozturknet/con1:latest bu image ın tag idir.**  **Docker commit -c ‘CMD [“java”, “uygulama”]’ ozguruzturknet/con1:ikinci #-c ile CMD komutunu bu aşamada girebiliyoruz.** | Docker image oluşturmada ikinci yöntem docker container çalıştırdıktan ve içindeki tüm istediğimiz değişiklikleri yaptıktan sonra bu container ı commit komutunu kullanarak image oluşturmak. | | |
| **DOCKER SAVE – LOAD** | | | |
| **Docker save ozgurozturknet/con1:latest –o con1imaj.tar #save komutu ozgurozturknet/con1:latest adlı image ı con1imaj dosyası haline getirdi**  **Docker load -i .\con1imaj.tar #yükleyeceğim yere giderim ve load komutu ile daha önc save lediğim imajın adını yazarım** | İnternetin bulunmadığı ortamlarda save komutu ile sistemdeki dosyaı tar dosyası olarak kaydedebiliyoruz daha sonra bu dosyayı int erişimi olmayan makineye kopyalayabilir daha sonra load komutu ile yükleyebiliriz. | | |
| **DOCKER REGISTRY** | | | |
| **Docker image ozgurozturknet/hello-app:latest**  **Docker image tag ozgurozturknet/hello:latest 127.0.0.1:5000/hello-app:latest #image ı taglayarak localime aldım.**  **Docker imgae push 127.0.0.1:5000/hello-app:latest #127.0.0.1:5000 repositry sine gönderildi.** | Docker registry imajını docker hub dan indiriyoruz. Bundan bir container yaratıyoruz. 5000 portunu publish ettiğimizde url sonuna /\_catalog dediğimizde içindeki reposity leri görebiliyoruz.  Bir imaja başka bir tag atadığımızda aslında onun nereye konulacağının göstergesi anlamına gelmektedir. | | |
| **DOCKER COMPOSE** | | | |
| **Docker-compose –version #version kontrolü** | Compose, çoklu Docker uygulamalarını tanımlamak ve çalıştırmak için bir araçtır. Compose ile uygulamanızın hizmetlerini yapılandırmak için bir YAML dosyası kullanırsınız. Ardından, tek bir komutla, tüm hizmetleri yapılandırmadan oluşturur ve başlarsınız. | | |
| **#elimizdeki yaml dosyası ile**  **Docker-compose up –d #yaml ın bulunduğu klasöre gitmek gerekiyor –d ile yaparsak arkada işlemleri yürütüyor** | Imajları çekiyor.  Imajlardan sonra servisleri yaratıyor. Containerları başlatıyor.  Cloudformation gibi otomatik olarak buradan bakıp istediğim işlemleri yapıyor. | | |
| **docker-compose down** | Bu yaml file içindeki her şeyi temizleniyor. Devam ediyorsa stop ediyor. Volume ları silmiyo | | |
| **Docker-compose --help** | Yardım | | |
| **Docker-compose config #dosyayı aşağıdan yukarıya doğru gösterir** | Kullanıla yaml dosyasını shell de görünür | | |
| **Docker-compose images** | #oluşturulan servislerin hangi imagelarla oluşturulduğunu gösterir. | | |
| **Docker-compose logs** | Docker compose ile oluşturulan servislerin loglarını gösterir | | |
| **Docker-compose exec websrv ls -al** | Websrv conatiner ın altında ls –al komutunu çalıştırabiliriz | | |
| YAML | | | |
|  | Dosyaynın adı “docker-compose.yml” olmalı. | | |
| **version: "3.7"**  **services:**  **veritabani: #container**  **image: mysql:5.7 #image ismi**  **restart: always #kapanırsa her seferinde yeniden başlat**  **volume: #containerlara bu volumelerin takılmasını istiyorum**  **- verilerim:/var/libmysql #ilk volume**  **enviroment: #env varialble tanımlıyorum #servislerin birbiri ile konuşmasını sağlamak için gereklidir**  **MYSQL\_ROOT\_PASSWORD: somewordpress**  **MYSQL\_DATABASE: wordpress**  **MYSQL\_USER: wordpress**  **MYSQL\_PASSWORD: wordpress**  **network: #servislerin birbiri ile konuşabilmesi için network tanımlıyorum**  **- wpnet**    **wordpress:**  **image: wordpress:latest**  **depends\_on:**  **- veritabani #öncelikle veri tabanının ayağı kalkması lazım ki wordpress çalışsın yani bu oluşturduğum servis wordpress e bağlı demek bu başlamadan diğerini başlatma demek**  **restart: always**  **ports: #port tanımlıyorum**  **- "80:80"**  **enviroment:**  **WORDPRESS\_DB\_HOST: veritabani:3306**  **WORDPRESS\_DB\_USER: wordpress**  **WORDPRESS\_DB\_PASSWORD: wordpress**  **WORDPRESS\_DB\_NAME: wordpress**  **volumes:**  **verilerim: #verilerim diye bir volume yarat ki ben yukarda yarattığım volume e bağlayabileyim.**  **networks: #network u burada oluşturuyorum yukarıda bağlıyorum**  **wpnet: #wpnet isimli network oluştur**  **driver: bridge #driver ı bridge olsun** | #compose un verisonu en yüksek kullanılmalı(hangiversionu kullanacağına docs.docker/compose adresinden bakabilirsiniz.)   * Servives altında container larımı tanımlıyorum. | | |
|  | Yaml dosyasının içine yazılan build komutu ile dockerfile dan image oluşturulabilirsiniz.  Docker compose ile oluşturulan bir imajı güncellemek için bir değişiklik yaptıktan sonra docker build komutunu ile build ediyoruz ve sonrasında docker compose up ile yeniden oluşturuyoruz. | | |
| **CONTAINER ORCHESTRATION** | | | |
| Container orchestration containerların dağıtımını yöntetimini ölçeklendirilmesini ve ağ oluşturmasını otomatikleştirme ve zamanlama işlemlerine denilir. Yüzlerce veya binlerce container’ı bir arada yönetmek için kullanıdığımız temel yazılımlardır. | | | |
| **DOCKER SWARM** | | | |
| Docker engine e entegre bir container orchestration çözümüdür. Bir docker ana bilgisayar havuzunu tek bir sanal ana bilgisayara dönüştürür. | | | |
| **Docker swarm init** | Engine mod dan swarm moduna geçirmek | | |
|  | Swarm devreye alındığında menager node a talimat veriliyor ve o bizim adımıza talimatları uyguluyor. Bir sistemde arıza çıktığında swarm bunu otomatik olarak destekliyor. Yeni manager node lar yükeleyebiliriyoruz. | | |
| Swarm birden fazla manager node destekler ve bu sayade yüksek erişilebilirlik sağlar. Bir manager sorun olursa diğer manager devreye girer ve iş yürümeye devam eder.  Bu manager nodelardan yalnızca bir tanesi lider olarak seçilir ve tüm yönetim lider tarafından yapılır. Diğer manager nodelar pasif durumdadır. Siz pasif manager node’lardan birine bir komut verip iş yapmasını isterseniz bu sadece proxy görevi ve komutu lider node’a iletir.  Ortamda birden fazla manager olduğu durumlarda bir adet lider seçilmelidir. Swarm bunu otomatik halleder ve bunun için Raft Consensus algoritmasını kullanır. Raft algoritması lider seçimi için kuralları belirler. Örneğin ortamda 5 manager olan bir durumda lider node bir şekilde erişilemez hale gelirse belirli bir zaman sonra kalan 4 node kendi aralarında oylama yaparak lider belirler. Ve artık swarm cluster’ın yönetimi bu lider tarafından sağlanır.  Raft consensus algıritması maksimüm (N-1)/2 sayıda replikanın devre dışı kalmasını tolere eder. Örneğin 5 swarm manager olarak kurgulanan bir yapıda 2 manager devre dışı kalırsa kalan 3 node kendi aralarında anlaşarak çalışmaya devam edecektir. Fakat 3 node devre dışı kalırsa kalan 2 node çoğunluğu sağlayıp anlaşamayacakları için sorun çıkacak ve management altyapısı çalışmayacaktır.  Raft algoritmasının düzgün çalışabilmesi ve lider seçiminin sorunsuz olabilmesi için ortamın her zaman tek sayıda manager nodela kurulmuş olması gerekir. bu nedenle Docker Swarm 1 ya da 3 ya da 5 ya da 7 manager ile kurulmalıdır. 9-11 diye devam ettiririlir ama 7’den fazla manager olduğu durumlarda ortamda daha fazla sorun çıkacaktır. | | | |
| **docker swarm init –adversite-addr 192.168.0.13** #bu sistemde yeni bir docker cluster oluştur. Advertise address olarak da 192.168.0.13 ü kullanmak istiyorum.  **docker info** #docker swarm kısmının aktif olduğunu görebiliriz.  **Docker swarm join-token manager #bir makinayı bu cluster a eklemem için girmem gereken koumutu bana gösteriyor ekranda.**  **Docker swarm join –toke SWMTKN…. #ekranda çıkan komut(manager yapmak için)**  **Docker swarm join –toke SWMTKN…. #ekranda çıkan komut(manager yapmak için)**  **Docker swarm join-token worker #worker olarak eklemek için hangi komutu girmem gerektiğini soruyorum o da ekranda bir komut gösteriyor.**  **Docker swarm join –token SWMTKN… #ekranda çıkan komut(worker yapmak için)**  **Docker swarm join –token SWMTKN… #ekranda çıkan komut(worker yapmak için)**  **Docker node ls #clusterla ilgili bilgi veriyor**  **#manager 1 e bağlandım**  **Docker service create --name test --replica=5 –p 8080:80 nginx #servis yarat ismi ne olsun test olsun. Toplamda altında 5 tane container olsun.8080 portundan 80 e publish et bunu da nginx den yarat. Bu işi tüm worker lara dağıttı. Onlarda bu işleri yaratmaya başladı**  **Docker service ps test #burada oluşturulan servicelerin test sonuçlarını gör(managerlar da worker gibi davranır onların üzerinde de conteyler yaratıldır. Porduction ortamında bu tavisye edilmez bunu kısıtlıyoruz. Manager sadece managerlık yapsın isteriz.)**  **Docker ps #conteiner ları görüyoruz**  **#diyelim ki bu workerlardan biri bozulsun diye bir senaryomuz olsun (disk bozulsun, veya worker bozulsun)**  **Docker service ps test #5 container yarattıktan sonra manager ın işi bitmedi. Gözlüyor sürekli sorun çıktı node5 de sorun çıktı kapandı sistemde 4 conteiner kalınca hemen devreye giriyor ve yeni conteiner yaratıyor.Müsait olan bir node da yaratıyor otomatikman.** | | DockerPlay Lab da 5 tane makine yarattım. Ardından docker swarm init –adversite-addr 192.168.0.13 komutunu girdim.  3 tane manager 2 tane worker umuz olacak şekilde senaryomuz var.  Makinaya tıklayarak hangi makinannın ne olmasını istiyorsak komutları makinanın içine giriyoruz. | |
| **Docker Service** | | | |
| Docker service create --name test --network abc –publish 8080:80 –replica=10 –update-delay=10s –update-paralelism=2 ozgurozturkalistirma:kirmizi | | Ozgurozturkalistirma:kirmizi isimli imajdan bir servis yaratılacak içinde 10 tane container çalışacak abc network una bağlı olacak ve 8080 portu bu container ların 80 portuna yönlendirilecek eğer ben bu servisi güncellersem her güncelleme aynı anda 2 task üzerinden yürütülecek ve her güncelleme arasında 10 sn bekleyeceksin.  Docker swarm bu tanımı alıyor. Etcd veri tabanına kaydediyor. Sonra bu cluster çapında uygulanıyor ve current state ile desired state eşitleniyor. Ve başlıyor container yaratmaya. İstenen durum ile mevcut durum arasında bir fark olursa bir sorun çıktı mesela 9 container oldu bu durumda istenen durumla mevcut durumu eşitlemeye çalışıyor ve 1 container daha yaratmaya çalışıyor(deklaratif mode). Bunun alternatifi (imperatif mode) bir komut girersiniz ve manuel yapmaya çalışırsınız. | |
|  | | Swarm altında temelde iki mode oluştursunuz. Varsayılan olan replicated mode dur.  **Replicated**: Oluşturmak istediğiniz servisin kaç replica içereceğini belirtirsiniz. Swarm uygun olan nodelar üstünde o sayıda replica oluşturur.  **Global**: oluşturmak istediğiniz servisin kaç replica içereceğini belirlemezsiniz. Swarm altındaki her node üstünde 1 replica oluşturulur. | |
| **Docker swarm init --advertise-addr 192.168.0.23** #docker swarm manager haline getirdim  **Docker swarm join –token SWMTKEN-1…. #**çıkan worker ekleme komutu  **Docker swarm join –token SWMTKEN-1….** #çıkan worker ekleme komutu  **Docker node ls** #cluster ları görüyorum  #tüm komutlar manager üzerinden girilir(workerlara girilirse error verir)  **Docker service create --help** #create komutunun nasıl olduğu görülebilir(run ile aynı)  **Docker service create --name test ngnix** #herhangi bir replica sayısı belirtmediğim için varsayılan olarak 1 task seçildi. Belirlenen adette task görev yaratılır. Aslında bu task ara katman. Task eşittir container diye düşünebilirisiniz. İlerde belki script olarak da dizayn edilebilir. Task oluşturulurken preparing aşamasına geçti sonra start etti. Task ı oluşturabileceği worker a söyledi ve hazırladı ve sonra start etti. Node müsait olana kadar prepare etti.  **Docker service** #bu komutla kullanılan alt komutları gösterir.(docker container test komutuna benzer)  **Docker service ps test** #istediğimiz service in task durumuna bakabiliyorum.  **Docker service inspect test** #test service inin detaylarını görebiliyorum.  **Docker service logs test** #test service inin altında üretilen logları gösterir  **Docker service scale test=3** #desire scale imi 3 olsun diyorum 2 tane daha container ekledi  **Docker service scale test=2** #desire imi 2 olsun diyorum 1 i siliniyor.  **Docker service rm test** #service siliyorum  **Docker service create --name glb --mode=global nginx** #global tipte service yaratmak için, glb isminde global mode da ngnix imajından oluştur. Kaç makine varsa hepsinde birer tane yaratır.  **Docker service ps glb** #gld olarak yaratılan servisleri gösterir | |  | |
| **Docker Service Update** | | | |
| **Docker network create –d overlay over-net #network oluştur**  **Docker service create --name websrv --network over-net -p 8080:80 --replica=10 ozgurozturknet/web:v1** #service oluştur  **Docker service ls** #servicelere bakıyorum  docker service ps websrv #servicelere bakıyorum  #yeni yarattığım image ımı v2 olan versiyonunu müşterilerim bu servislere bağlanırken kesinti olmaksızın güncellenmesini istiyorum  **docker service update --help** #opsiyonları gösteriyor (yeni env ekleniyor, çıkarılıyor containerları update edebiliyorum, güncelelyebiliyorum)  **docker service update ps websrv** #update yapıldığında öncelikle containerı siler sonra yeni containerı yaratacak mevcutta yapmaz. yeni bir container oluşturur update edilen version ile. Updata sırasını kendimiz belirleyebiliyoruz. geçişler arasında bir delay vermek istiyorsak bunu --update-delay duration komutu ile yapabiliriz.  --update-paralelism unit containerları tek tek değil de gireceğiniz sayıya göre 2 şer olarak da update edebilir.  **docker service update --detach --update-delay 5s --update-parallelism 2 --image ozgurozturknet/web:v2 websrv** #image ile update et, bu işlemi 2 task üzerinde yap ve her işlemden sonra 5 sn bekle ve arka planda yap  **docker service ps websrv** #yapılan işlemleri görebiliyorum  #bu aşamada sorun yaşarsam  **docker service rollback --detach websrv #bir önceki v1 e geri dönecek** | |  | |
| **Docker Secret** | | | |
| Containerlarda plain text olarak tutmamızın güvenlik zafiyeti yaratabileceği kullanıcı adı ve şifre gibi verileri secret objeleri şeklinde encrypted olarak transfer edebiliriz. | | | |
| **docker swarm init** #docker engine tek node lu oluşturdum  **#secret yaratmanın iki yolu var birincisi:**  **notepad kullaniciadi.txt #dosya içine ozgurozturk değerini giriyorum**  **notepad sifre.txt #Password olarak bir değer giriyorum**  **docker secret create kullanici\_adi .\kullaniciadi.txt** #kullanici\_adi secret ini oluştur değer olarak da kullanici\_adi değerini ata  **docker secret create sifre .\sifre.txt** #sifreyi oluştur sifre.txt olarak ata  **docker secret ls** #secretları göster  **docker secret inspect sifre** #aytıntısını gör  #secret yaratmanın ikinci şekli  **echo “bu bir denemedir” | docker secret create deneme -** #bu bir denemedir değerini “deneme” isimli secret ı oluşturarak ata  **docker secret ls** #secretları gör  #secretlara servislerde nasıl ulaşabilirim  **docker service create -d --name secretdeneme -secret kullanici\_adi --secret sifre --secret deneme ozgurozturknet/web**  **docker service ps secretdeneme**  **docker exec -it [container id] sh**  #run dosyasının altında dosyanın adı secret in ismidir. değeri de verilen değerdir. kaç tane secret oluşturdu isen o kadar file vardır. böylelikle encrypt bir service oluşturdum. bunu da containerın içine gömdüm. böylelikle ben burada çalışan uygulamaya injekte edebilirim. mesela veri tabanına bağlanırken bilgileri buradan çek diyebilirim. Böylelikle env olarak girmeme gerek kalmadı. güvenlik açıklarından sakınabildim  #diyelim ki secret ı güncellemek istiyorum bunun için. içeriğini değiştiremiyorsunuz.  **echo “basodaskp” | docker secret create sifre2** #yeniden oluşturuyorum  **docker service update --secret-rm sifre --secret-add sifre2 secretdeneme** #eskisini silerek yenisini koymak gerekiyor | |  | |
| **Docker Stack** | | | |
| Docker compose un swarm ortamında yapılmasına imkan sağlayan hizmettir. compose ile bazı farklarla aynı dosya kullanılıyor.  Compose dosyasının en az 3 versiyonu olması gerekiyor.  Engine versiyonuna uygun olmayan şeyleri burada yaratamıyoruz.(bridge network, build gibi)  Bölüm 79 da-68 dekinin aynısı | | | |
| version: '3.7'  services:  mysqldb:  image: ozgurozturknet/webdb  environment:  MYSQL\_DATABASE: proje  MYSQL\_USER: projemaster  MYSQL\_PASSWORD: master1234  MYSQL\_ROOT\_PASSWORD: master1234  networks:  - webnet  websrv:  image: ozgurozturknet/webkayit #build komutu burada yok imagden var  deploy: #ilerde servis update edilirse nasıl olacağını önceden giriyorum  replicas: 3  update\_config:  parallelism: 2  delay: 10s  order: stop-first  depends\_on:  - mysqldb  ports:  - "80:80"  restart: always  networks:  - webnet  environment:  DB\_SERVER: mysqldb  DB\_USERNAME: projemaster  DB\_PASS: master1234  DB\_NAME: proje  networks:  webnet:  driver: overlay #bridge dan overlay yaptık | | docker stack #bu komutla hangi komutlar olacağını gösterir  **docker stack deploy -c .\docker-compose.yml ilkstack** #stackden oluştur dosyanın adını ve yolunu girmek gerekir.  **docker stack ls** #stackleri listeliyor  docker stack ps ilkstack #bu stack altındaki servislerin tasklarını gösterir  docker stack rm ilkstack #direk sormadan siler | |
| **Overlay Network (Driver)** | | | |
| Docker swarm altında bu driver ile yeni bir network oluşturabiliyoruz. Tüm cluster genelindeki tüm nodelar üzerinden erişilebiliyor. Bu nodelar hangi networkte olursa olsun birbiri ile haberleşebiliyorlar.  Bir swarm cluster oluşturulduğu zaman ingress adında bir overlay network de otomatik olarak oluşturulur. Siz aksini belirtmediğinzi sürece oluşturduğunuz servisler bu overlay networke bağlanır.  Biz istersek aynen user defined bridge networkde yaptığımız gibi user defined overlay network de yaratabiliriz.  Overlay networklerin temel yönetim katmanının haberleşme altyapısı ecryptedır. Fakat sizin buraya bağlandığınız containerların birbirleriyle iletişimi varsayılan olarak encrypted değildir. Siz overlay network yaratırken --opt encrypted opsiyonunu kullanarak bu trafiğin de encrypted olmasını sağlayabilirsiniz. Fakat bu network trafiğini biraz yavaşlatacaktır.  Aynı overlay network’e bağlı servislerin containerları birbiriyle herhangi bri port kısıtlaması olmaksızın haberleşebilirler ve sanki yanı ağdaymış gibi çalışırlar.  Swarm altında yaratılan servisler aynı overlay network üzerinde birbirlerine servis isimleriyle ulaşabilirler. Docker burada hem dns çözümlemesi hizmeti hem de load balancing hizmeti sunmaktadır. Swarm altında overlay ile yaratılan networkler servis isimleri ile birbirlerini çağırabilirler.  Docker burada hem dns çözümlemesi hizmeti hem de load balancer hizmeti vermektedir.  Overlay network üzerinde aynı zamanda port publish de edebiliriz ve Docker Swarm overlay networklerde ingress routinh mesh destekler. Siz port publish edip Docker host üstünden o porta erişirseniz Docker o host üstünde o portun publish olduğu bir container bulunmasa bile bir host’a trafiği yönlendirecek ve cevap verecektir. | | | |
| **# 3 manager 2 workerlı sistemi ayağa kaldırdım. Dockerlabdaki template ile**  **# iki servis oluşturacağız web ve db adında**  **Docker network ls** #networkleri listeledim arkada yaratılan docker\_gwbridge network var containerlerın bridge ına denktir. İngress network u de yaratılmıştır. Yeni serviste hiçbir ağ belirtilmezse burada ingress overlay ağına bağlanıyor. Ama bunu kullanmıcaz kendimiz overlay oluşturucaz.  **docker network create -d overlay over-net** #manager üzerinden yaratıyorum  **docker network ls** #listele  **docker network inspect over-net** #ayrıntılarına bak  **docker service create --name web --network over-net -p 8080:80 –replica=2 ozgurozturknet/web** #containerları oluştur  **docker sevice ls** #listele  **docker service ps** #kaç task çalışıyor 3 task(container)  #swarm üzerinde bir port pubslih i yaptığınızda tüm cluster üzerinde bu port publish ediliyor. Container olmasa da o 8080 den engine üzerinden bağlanılabiliyor. Manager o isteği alıyor başka bir node a yönlendiriyor işte buna routing mash diyoruz.  **docker service create --name db --network over-net ozgurozturknet/fakedb** #db service yarattım  **docker service ps db** #db nerde çalışıyor bakıyorum (manager 2 deymiş)  **docker container exec –it container numarası sh** #containera bağlandım  **ping web** #web service ine erişebildim  #service in ismini söylerseniz aynı networkdeki makinalarla konuşabilir. Web deyince VIP ıne gidiyor ve oradan ulaşıyor.  **curl** [**http://web**](http://web)#komutunu her girdiğimde başka bir containerdan cevap geliyor  **curl** [**http://web**](http://web)#komutunu her girdiğimde başka bir containerdan cevap geliyor  **curl** [**http://web**](http://web)#komutunu her girdiğimde başka bir containerdan cevap geliyor  **#bu da overlay hizmetinin load balancer hizmeti olarak çalıştığının da göstergesi.** | | 5 nodeluk docker swarm kuracağım ve overlayer oluşturacağım. 2 adet swarm servis yaratacağım bunlardan web adında olanı 3 replikaya sahip olacak db adında olanı tek replikaya sahip olacak. Her iki servisi de aynı overlayer a bağlayacağım böylece container lar birbiri ile haberleşecek. Web servisini de dış dünyaya publish edeceğim. 8080:80 yapacağım. Hostların 8080 portuna gittiğimde ilgili makine hanginde ise ona yönlendirecek.Routing machine nedir onu göreceğiz. | |
| **WINDOWS CONTAINERS** | |  | |
| windows 2016 veya 2019 docker engine yükleyeceksiniz. Docker Entrprise olmak zorunda dockerhubdan yüklenecek. bu işletim sistemlerinden birine sahipseniz extara bir ücret ödenmiyor. 2010 için de Docker desktopdan windows container a geçiş yapmanız gerekir. | | | |
| Windows, kullanıcılarının oluşturabileceği dört temel container image tipi sunar.  Her image tipi, Windows işletim sisteminin farklı boyutta ve farklı miktarda Windows API seti taşıyan halidir  ----  Windows Server Core: Supports Traditional .Net Framework Applications | https://hub.docker.com/\_/microsoft-windows-servercore  Nano Server: Built for .Net Core Applications. | https://hub.docker.com/\_/microsoft-windows-nanoserver  Windows: Provides the full Windows Api set. | https://hub.docker.com/\_/microsoft-windows  Windows IoT Core: Purpose-built for IoT applications. | https://hub.docker.com/\_/microsoft-windows-iotcore  -----  Hangi image hangi iş için uygun?  • Uygulamanız için tam .NET framework gerekiyor mu? Bu sorunun cevabı evet ise, Windows Server Core'u hedeflemelisiniz.  • .NET Core tabanlı bir Windows uygulaması mı oluşturuyorsunuz? Bu sorunun cevabı evet ise, Nanoserver'ı hedeflemelisiniz.  • IoT uygulaması mı oluşturuyorsunuz? Bu sorunun cevabı evet ise, IoT Core'u hedeflemelisiniz.  • Windows Server Core image'inde uygulamanızın ihtiyaç duyduğu bir kütüphane ya da dependency eksik mi? Bu sorunun cevabı evet ise, Windows'u hedeflemeye çalışmalısınız. Bu image diğer temel image'lerden çok daha büyüktür, ancak çekirdek Windows kitaplıklarının çoğunu (GDI kitaplığı gibi) taşır.  -----------------------------------------------------  Bir .Net Framework veya .Net Core tabanlı bir uygulamayı veya bir web uygulaması için IIS'yi mi kullanmak istiyorsunuz?  O zaman Server Core alıp üstüne bu araçları yüklemekle boşuna vakit kaybetmeyin.  Microsoft temelde .net yüklü olan hem .Net Framework hem de .Net Core temel image'lerini (hem SDK hem de Runtime) yayınlar. Kendi .net image'lerinizi sıfırdan oluşturmak yerine her zaman bu dotnet temel image'lerini kullanın.  .NET Framework:  - .NET Framework SDK | https://hub.docker.com/\_/microsoft-dotnet-framework-sdk/  - .NET Framework Runtime | https://hub.docker.com/\_/microsoft-dotnet-framework-runtime/  - ASP.NET Web Forms and MVC | https://hub.docker.com/\_/microsoft-dotnet-framework-aspnet/  - Windows Communication Foundation (WCF) | https://hub.docker.com/\_/microsoft-dotnet-framework-wcf/  .NET Core:  - .NET Core SDK | https://hub.docker.com/\_/microsoft-dotnet-core-sdk/  - .NET Core Runtime | https://hub.docker.com/\_/microsoft-dotnet-core/  - ASP.NET Core Runtime | https://hub.docker.com/\_/microsoft-dotnet-core-aspnet/  IIS  - Server Core IIS | https://hub.docker.com/\_/microsoft-windows-servercore-iis  ---------------------------------------------------------  Windows, container işletim sistemi sürümünün host işletim sistemi sürümüyle eşleşmesini gerektirir. Daha yeni bir Windows derlemesine dayalı bir container çalıştırmak istiyorsanız, eşdeğer bir ana makine derlemesine sahip olduğunuzdan emin olun. Örneğin, ana bilgisayar Windows Server 2019 ise ve Server Core image'ini kullanmak istiyorsanız, image'iniz mcr.microsoft.com/windows/servercore:ltsc2019 olmalıdır. Ancak, host makineniz Windows Server 2019 ise, image'iniz mcr.microsoft.com/windows/servercore:ltsc2016 olmalıdır.  ---------------------------------------------------------  Escape yönergesi, Dockerfile'daki karakterlerden kaçmak için kullanılan karakteri ayarlar. Belirtilmezse, varsayılan kaçış karakteri ters eğik çizgi \ olur. Kaçış karakterini backsitck `olarak ayarlamak özellikle Windows'ta kullanışlıdır, burada \ dizin yolu ayırıcısıdır.  Dockerfile (# escape=`) komutuyla başlamalıdır. Bu, backstick'i bir kaçış karakteri olarak tanımlayacaktır ve ters eğik çizgi karakterini Dockerfile'da bir dosya yolu olarak kullanabileceğiz.  ---------------------------------------------------------  Shell ile ilgili uyumsuzlukları önlemek için PowerShell'i varsayılan shell olarak ayarlayın.  Dockerfile dosyanızdaki aşağıdaki talimatı kullanın. (P.S. Powershell, Nanoserver temel image'ine dahil değildir)  SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop';"] | | | |
|  | |  | |
|  | |  | |
| **Linux Shell Command** | | | |
| **echo $SHELL > deneme.txt** | | | SHELL in çıktısını deneme.txt içerisine yaz |
| **&** | | | Script in arka tarafta çalışmasına izin veriyor ki ben consolu kullanmaya devam edeyim. |
| **|** | | | Birinci komutun çıktısının ikinci komutta kullanılmasını sağlar. |
| **grep** | | | Bir karakter ya da kelimeyi aratmak |
| **; (komut1; komut2; komut3)** | | | Aynı satırdan birden fazla komut çalıştırmaya izin verir |
| **ls ; date** | | | Hem ls i verir hem de date i verir. |
| **&& (komut1 && komut2)** | | | Birden çok komutun birleştirmesini yapar. İki komut arasına and koymak demek. Birinci hatalı ise ikinciyi çalıştırmaz. |
| **|| (komut1 || komut2)** | | | Or işareti. Birinci çalıştı ise ikinciyi çalıştırma. Komut1 çalıştırmazsa komut2 yi çalıştır. |
|  | | |  |
|  | | |  |
|  | | |  |

***Docker Cheat Sheet (Türkçe)***

* ***Container***

**Container çalıştırma:**

docker container run image:tag

Örnek: docker container run nginx:latest

**Detach modda container çalıştırma (-d):**

docker container run -d image:tag

Örnek: docker container run -d nginx:latest

**Varsayılan uygulama yerine başka uygulama ile container başlatma:**

docker container run image:tag uygulama

Örnek: docker container run ubuntu:latest ping 127.0.0.1

**Container’a bir isim vererek çalıştırma**

docker container run --name isim image:tag

Örnek: docker container run --name container1 -d nginx:latest

**Çalışan bir container içerisinde başka bir uygulama çalıştırma:**

docker container exec container\_id|ya da|container\_ismi uygulama

Örnek: docker container exec 12a793b3fec0 ping 127.0.0.1

**Çalışan bir container’a shell bağlantısı oluşturma:**

docker container exec -it container\_id|ya da|container\_ismi sh

Örnek: docker container exec -it 12a793b3fec0 sh

**Container’ı detach modda ve shell bağlantısı ile oluşturma (dit):**

docker container run -dit image:tag sh

Örnek: docker container run -dit nginx:latest sh

**Detach modda ve shell bağlantısı ile oluşturulmuş container’a bağlanma:**

docker attach container\_id|ya da|container\_ismi

Örnek: docker attach 12a793b3fec0

**Container durdurma:**

docker container stop container\_id|ya da|container\_ismi

Örnek: docker container stop 12a793b3fec0

**Container silme:**

docker container rm container\_id|ya da|container\_ismi

Örnek: docker container rm 12a793b3fec0

**Çalışan containerı silme (-f):**

docker container rm -f container\_id|ya da|container\_ismi

Örnek: docker container rm -f 12a793b3fec0

**Container kapatıldığı zaman aynı zamanda silinmesi (-rm):**

docker container run -rm image:tag

Örnek: docker container run -rm nginx:latest (-rm ile container kapandığı zaman otomatik olarak silinmesini söylüyoruz)

**Container ile ilgili detayları inceleme:**

docker container inspect container\_id|ya da|container\_ismi

Örnek: docker container inspect 12a793b3fec0

**Sistemdeki tüm containerları (çalışan ve durdurulmuş) silme**

docker container rm -f $(docker ps -aq)

**Sistemdeki çalışan containerları listeleme:**

docker container ls ya da

docker container ps

**Sistemdeki tüm containerları listeleme:**

docker container ls -a ya da

docker container ps -a

**Çalışan container’daki processleri listeleme:**

docker top container\_id|ya da|container\_ismi

Örnek: docker top 12a793b3fec0

**Çalışan container’ın Cpu, Ram, I/O kullanımını görme:**

docker stats container\_id|ya da|container\_ismi

Örnek: docker stats 12a793b3fec0

**Container’ın memory kullanımını sınırlama (–memory, --memory-swap):**

docker container run --memory=rakam(b,k,m,g) --memory-swap=rakam(b,k,m,g) image:tag

Örnek: docker container run --memory=1g --memory-swap=2g nginx:latest (memory-swap ile swap alanı da tanımlayabiliriz.b=byte k=kilobyte m=megabyte g=gigabyte)

**Container’ın cpu kullanımını sınırlama (–cpus, --cpuset-cpus):**

docker container run --cpus="core\_adeti" image:tag

Örnek: docker container run --cpus="3" nginx:latest (sistemden kaç core’a erişebileceğini belirledik)

docker container run --cpuset-cpus="core\_numarası" image:tag

Örnek: docker container run --cpuset-cpus="0,4" nginx:latest (sistemdeki hangi corelara erişebileceğini belirledik)

**Container’a enviroment variable tanımlama:**

docker container run --env enviroment\_variable=değeri image:tag

Örnek: docker container run --env VAR1=deneme1 --env VAR2=deneme2 nginx:latest

**Containerdan hosta ya da tam tersi dosya kopyalama:**

docker cp container\_id|ya da|container\_ismi:path host\_path

Örnek: docker cp 12a793b3fec0:/usr/src/uygulama/ .

* ***Image***

**Docker CLI aracılığıyla registery’de oturum açma:**

docker login registery\_url

Örnek: docker login localhost:8080

**Sisteme bir imaj çekme:**

docker image pull image:tag

Örnek: docker image pull nginx:latest

**Docker hub’a (ya da başka bir repository) image gönderme:**

docker image push repository/image:tag

Örnek: docker image push ozgurozturknet/adanzyedocker:latest

**Mevcut bir imaja yeni tag ekleme**

docker image tag image:tag yeniimage:tag

Örnek: docker image tag nginx:latest ozgurozturknet\nginx:v1

**Image ile ilgili detayları inceleme:**

docker image inspect image:tag

Örnek: docker image inspect nginx:latest

**Image layerlarını listeleme:**

docker image history image:tag

Örnek: docker image history nginx:latest

**Dockerfile kullanarak yeni bir imaj yaratma:**

docker image build -t image:tag .

Örnek: docker image build -t ozgurozturknet\merhaba-dunya:latest . (Dockerfile dosyası komutun çalıştırıldığı folder’da bulunmalı)

**Image oluştururken build arg kullanma:**

docker image build --build-arg arg=deger -t image:tag .

Örnek: docker image build --build-arg VERSION=3.7.1 -t nginx:latest .

**Sistemdeki tüm imageleri listeleme:**

docker image ls

**Sistemden bir imajı silme:**

docker image rm image:tag

Örnek: docker image rm nginx:latest

**Containerdan image yaratma:**

docker commit container\_id|ya da|container\_ismi image:tag

Örnek: docker commit 12a793b3fec0 ozgurozturknet/img:latest

**Image’i bir dosyaya kaybetmek ve kaydedilmiş bir dosyadan image oluşturmak:**

docker save image:tag -o dosyaadi.tar

Örnek: docker save ozgurozturknet/img:latest -o image.tar

docker load -i dosyaadi.tar

Örnek: docker load -i imagecon1.tar

* ***Volume***

**Volume oluşturma:**

docker volume create volume\_ismi

Örnek: docker volume create ilkvolume

**Volume ile ilgili detayları inceleme:**

docker volume inspect volume\_id|ya da|volume\_ismi

Örnek: docker volume inspect ilkvolume

**Sistemdeki tüm volumeleri listeleme:**

docker volume ls

**Volume’u container’a bağlama (-v):**

docker container run -v volume\_ismi:container\_icindeki\_path image:tag

Örnek: docker container run -v ilkvolume:/var/www/html image:tag

**Volume’u container’a sadece okunur şekilde bağlama (:ro):**

docker container run -v volume\_ismi:container\_icindeki\_path:ro image:tag

Örnek: docker container run -v ilkvolume:/var/www/html:ro image:tag

**Host üstündeki bir klasör ya da dosyayı bind mount olarak bağlama:**

docker container run -v host\_klasör\_path:container\_icindeki\_path image:tag

docker container run -v c:\websitesi:/usr/share/nginx/html nginx:latest

**Volume silme:**

docker volume rm volume\_ismi

Örnek: docker volume rm ilkvolume

* ***Network***

**Kullanıcı tanımlı bridge network oluşturma (bridge):**

docker network create --driver=bridge network\_ismi

Örnek: docker network create --driver=bridge kopru

**Kullanıcı tanımlı bridge network oluşturma (ip bilgilerini belirleyerek):**

docker network create --driver=bridge --subnet=cidr --ip-range=cdir --gateway=ip\_adresi network\_ismi

Örnek: docker network create --driver=bridge --subnet=10.10.0.0/16 --ip-range=10.10.10.0/24 --gateway=10.10.10.10 kopru

**Sistemdeki tüm volumeleri listeleme:**

docker network ls

**Volume ile ilgili detayları inceleme:**

docker network inspect network\_ismi

Örnek: docker network inspect kopru

**Container’ı varsayılan dışında bir network’e bağlayarak çalıştırma:**

docker container run --network network\_ismi image:tag

Örnek: docker container run --network kopru nginx:latest

**Çalışan bir container’ı başka bir network’e bağlama:**

docker network connect network\_ismi container\_id|ya da|container\_ismi

Örnek: docker network connect kopru 12a793b3fec0

**Çalışan bir container’ın bağlı olduğu networkle bağlantısını kesme:**

docker network disconnect network\_ismi container\_id|ya da|container\_ismi

Örnek: docker network disconnect kopru 12a793b3fec0

**Port publish ederek bir container çalıştırma (-p):**

docker container run -p host\_portu:container\_portu/tcp\_yada\_udp image:tag

Örnek: docker container run -p 8080:80 -p 53:53/udp nginx:latest

* ***Logging***

**Container tarafından oluşturulan logları görmek:**

docker logs container\_id|ya da|container\_ismi

Örnek: docker logs 12a793b3fec0

**Container tarafından oluşturulan logları uzun formatta detaylı görmek:**

docker logs --details container\_id|ya da|container\_ismi

Örnek: docker logs --details 12a793b3fec0

**Container tarafından oluşturulan logları belirli tarih aralığında görmek:**

docker logs --since tarih\_saat --until tarih\_saat container\_id|ya da|container\_ismi

Örnek: docker logs --since 2020-01-13T11:34:43.154304300Z 12a793b3fec0 (since verilen andan itibaren olanları, until ise verilen ana kadar olanları listeler)

**Container tarafından oluşturulan logların belirli sayıda son oluşanlarını görmek:**

docker logs --tail sayı container\_id|ya da|container\_ismi

Örnek: docker logs --tail 10 12a793b3fec0 (son 10 log çıktısını listeler)

**Container tarafından oluşturulan logları anlık olarak izlemek:**

docker logs -f container\_id|ya da|container\_ismi

Örnek: docker logs -f 12a793b3fec0 (loglar oluştukça ekranda gözükecektir. Ctrl-C ile bağlantı kesilebilir)