DOCKER

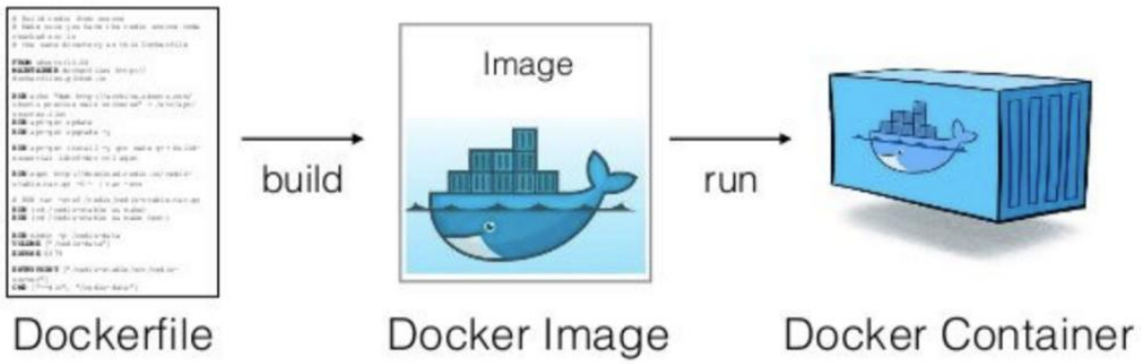
1- Docker Nedir?

Docker açık kaynaklı bir sanallaştırma platformudur.

Docker öncesi sanallaştırmada bir uygulamanın ayağa kaldırılması için gerekli tüm dosyaların yüklü olması gerekiyordu. (örn: Kütüphaneler...) Misafir işletim sistemi ayağa kaldırılmalı sonrasında konfigürasyonlar belirlenmeli sonra kütüphaneler yüklenmeli.

Uygulamalar misafir işletim sistemi üzerinde çalışır.

Docker ile Docker Engine bizim yerimize sanallaştırma işlemini bizim yerimize yapar. Uygulamalar direk Docker Engine üzerinde çalışır. Docker ile bir uygulamayı ayağa kaldırırken bir misafir işletim sistemine ihtiyaç duymayız. Uygulamayı ayağa kaldırmak için gerekli kütüphanelerin exelerin yüklenmesi yeterli. Uygulamanın gereklerini içerisinde bulunduran yapıya conteiner denir.



Bir uygulamamızı Docker bünyesinde ayağa kaldırabilmek(dockerize yapmak) için yukarıdaki adımları gerçekleştirmemiz gerekiyor.

- İlk adım prolemizin root klosörü içerisinde oluşturacağımız bir Dockerfile dosyası oluşturmaktır. (Bu dosyanın bir uzantısı yoktur.) Bu dosya içerisine talimatlar yazarız. Bu talimatlar uygulamamızın nasıl bir ortamda çalışacağı ile ilgili talimatlardır.
- Bu adım sonrası docker buid komutu ile bir image oluştururuz. Bu image, Dockerfile içeriğine doğrultusunda oluşturulur. Docker Image bir layer(katmanlar) topluluğudur. Dockerfile içerisindeki her bir satır için bir layer oluşur. (Dört satır varsa dört katmanlı bir image oluşur.) Docker Image static bir dosyadır. Yani sadece okunabilir bir dosyadır. Çalıştırılabilir bir dosya değildir.

- Uygulamamızı çalıştırabilmek için docker run komutu ile bir container ayağa kaldırırız. Container, Docker Imagelerin çalışabilir instance leridir. Containerleri başlatabilir, durdurabilir, silebiliriz.

Docker Image üzerinden istediğimiz kadar container ayağa kaldırabiliriz.

DockerCLI

DockerCLI, Docker ile haberleşmemizi sağlayan bir komut satırıdır.

Bilgisayarımıza Docker kurduğumuzda iki ana tool kurulur.

- Server tarfında docker deamon
- Client tarafında docler CLI

Dacler deamon tek iken birden fazla client olabilir. (Docker CLI bu client lerden biridir.)

Docker CLI a bir komut yazılınca bu komut docker deamon a API isteğinde bulunur. Docker deamon bu API isteği üzerinden yapacağı işlemi gerçekleştirir.

Docker Registry nedir?

Image lerimizi kaydedebileceğimiz bir depo olarak tanımlanabilir.

Bunun yararı sürekli yeni Image ler yazmak yerine var olan Image lerden gerekli olanı hazır olarak alıp kullanabiliriz.

Registry hizmeti sunan farklı kuruluşlar vardır.(Docker, Hub Azure Container, AWS Container...)

Docker Yararları

Docker öncesi tam bir işletim sistemi ayağa kaldırırken Docker sonrası ihtiyaçlarımıza uygun mini bir işletim sistemi ayağa kaldırırız.

Öncesi izolasyon işletim sistemi bazında sonrasında ise container bazındadır.

Docker ile uygulamanın çalışır hale gelme süresi kısaldı.

Docker ile uygulama geliştirirken kolay ve kısa sürede versiyonlama imkanı bulduk.

Docker sonrası uygulama paylaşım süresi kısaldı.

Docker bilgisayarlar arası versiyondan işletim sisteminden bağımsız bir ortam sağlıyor.

Notlar...

Windows işletim sistemine sahip bilgisayarlarda Docker kurduğumuzda hem Linux container ları hemde Windows container ları çalıştırabiliriz. Fakat Linux türevi bir işletim

sistemine sahip bir bilgisayarda Docker kurduğumuzda sadece Linux conteıner ları çalıştırabiliriz.

Docker .dll dosyasını kullanır.

Bir uygulamayı Dockerize yaparken bu uygulamanın dosya yapısını bilmek bizim için önemli. Dosya yapısını bilmek bu işlemi yaparken bize kolaylık sağlar.

Container ler Imagelerin çalışabilen intance larıdır.

Container ın çalışabilmesi için ona bir .dll vermemiz gerekir. .(Sebebi bir proje içerisinde uygulamamızı çalıştırmak isiyorsak o proje içerisinde uygulamamızın .dll ini kullanırız.)

Docker'ın doğal ortamı Linux'tur.

Her bir Image, mini bir işletim sistemidir.

- Environmentlar, bir uygulamayı sunucuda restart etmeden o uygulamanın davranışını değiştirmemizi sağlar.

push ve pull komutları Docker Hub ile haberleşir. Pull komutu Docker Hub üzerindeki herhangi bir Imageyi çeker. Push komutu Docker Hub üzerindeki repository'leri public ya da private olarak Image'larımıza gönderebiliriz.

Kendi Image'larımızı Docker Hub'a atıp daha sonra Docker Hub'a girerek başka serverlardan bu Image'ları çekebiliriz.

(Ctrl + C sunu –sig-proxy=false)

Konteyner silmek için ilk olarak durdurup sonra rm komutu ile silmeliyiz ya da çalışan konteyneı rm komutuyla silmeliyiz.(docker rm konteyner_adı –force)

Eğer SDK varsa uygulamamız lokalde çalışır.

Dockerfile dosyasında ne kadar çok layer yani satır varsa o kadar performanslı bir uygulama yapmış oluruz.

Docker ortam için en küçük birim conrainerdır.

.Net Core Concole Uygulamasını nasıl Dockerize yaparız?

- Uygulamayı hazırla.

Docker da bir image oluşturabilmek için uygulamayı publish etmemiz gerekiyor.

- Uygulamayı publish et.
- Dockerfile dosyasını yaz.

Dockerfile dosyasını oluşturma:

Projeye sağ tık yap – Add – Docker Support – İşletim sistemi seç

İmage gereklerine göre Dockerfile içerisini yazıyoruz

Dockerfile:

FROM mcr.microsoft.com/dotnet/core/runtime:3.1(image yolu) : Hazır image kullanabilmek için FROM komutu kullanılıt.

WORKDIR /app : Image içerisinde App adında bir klasör oluşturur. Bu komut yazılan dosya adında bir dosya varsa o dosyaya gider. Yoksa yazılan adda yeni bir dosya oluşturur.

COPY bin/Release/net8.0/publish(kopyalanacak dataların olduğu yerin yolu) /app/(dataların kopyalanacağı yerin yolu) : App klasörü içerisine uygulamanın publish edilmiş datasını atar. Yani bilgisayarımızdaki konsol uygulamasından image içerisine dosya gönderir.

ENTRYPOINT ["dotnet" , "UdemyConsoleDocker.dll"] : Bir Imageden bir Container ayağa kalktığı zaman çalışacak olan kodu belirler. Yani uygulamayı çalıştıran komuttur.

Bu Dockerfile dosyasına göre 4 katmanlı bir yapı oluşur. Dockerfile çalıştırıldığındada Docker bir yazılabilen katman oluşturur ve toplamda 5 katmanlı bir yapı oluşmuş olur.

- Image dosyasını oluştur.

Dockerfile içerisinde belirtilen talimatlara göre bir Image oluşturabilmek için üç konuya dikkat etmeliyiz.

1. İşletim sistemi üzerinde Docker çalışıyor olmalı.
2. Docker da sign in olmalıyız.
3. Dockerin hangi tür kontainerları çalıştıracağını kontrol etmeliyiz.(Windows container/linux container)

- Docker çalışınca Docker CLI aracılığı ile Docker a Image oluşturma ile ilgili bir komut yazabiliriz.

- **docker build -t udemyconsoleapp(Image adı ,küçük harfler kullanılmalı) .(Nokta ile Dockerfile dosyasını kendisi bulur.):** Image oluşturma komutu
- **docker images :** Docker içerisindeki Image leri listeleme

Artık Container oluşturmak için hazırız.

- Container oluştur.

Container oluşturduğumuzda uygulamamız artık ayakta ve çalışıyor olacak.

- **docker –name udemyconsole_container(container adı) udemyconsoleapp(hangi Image den container oluşturulacaksa o Image nin adı):** Docker oluşturma komutu

- **docker ps -a :** Mevcut Container ları listeleme
- **docker ps :** Çalışan Container ları listeleme
- **docker start udemyconsole_container(çalıştırılacak container adı) :** Container ı çalıştırma
- **docker stop udemyconsole_container(durdurulacak container adı) :** Var olan Container ı durdurma

Elimizdeki Image den istediğimiz kadar Container oluşturabiliriz.

Docker Container içerisindeki çıktıyı nasıl görürüz?

Consol açıyoruz. Consol da nerede olduğumuz önemli değil.(Çünkü herhangi bir buid işlemi gerçekleştirmeyeceyiz.)

- **docker attach udemyconsole_container(bağlanılacak container adı) :** Container a bağlanma

Docker CLI Komutları

- **docker run –name myapp(oluşturulacak container adı) udemyconsoleapp(hangi image üzerinden oluşturulacaksa o imagenin adı) :** docker create + docker start + docker attach
- **docker run –rm –name myapp(oluşturulacak container adı) udemyconsoleapp(hangi imsage üzerinden oluşturulacaksa o imagenin adı):** container rm parametresi eklenerek oluşturulursa container durdurulduğu zaman durdurulan container aynı zamanda silinir.
- **docker rm udemyconsole_container(silinecek container adı) :** durdurulmuş bir container ı silme
- **docker rmi udemyconsoleapp(silinecek image adı) :** Image yi silme komutu (Image yi silmeden önce silinecek image ye bağlı container ların silinmesi gerekiyor.)
- **docker build -t udemy_conso_leapp(Image adı ,küçük harfler kullanılmalı):v1(etiket) .(Nokta ile Dockerfile dosyasını kendisi bulur.):** oluşturulan sImageye etiket(tag) verme
- **docker rm udemyconsole_container(silinecek container adı) --force: Çalışan** bir container ı silme
- **docker rmi udemyconsoleapp(silinecek image adı) -–force:** Image ye bağlı container varsa ve durmuş ise Image yi silme komutu
- **docker pull mcr.microsoft.com/dotnet/sdk(çekilecek Imagenin yolu):** Docker Hub sitesinden istediğimiz bir Image'ı çekme
- **docker tag busybox(referans alınacak Image adı) seyma5/udemyrepository(hangi Repository'e aImage atılacaksa o repo adı):v1(tag):** Imageve tag atamak Yeni oluşturulan Image ilk Imaageyi referans alır.

- **docker push seyma5/udemyrepository(hangi Repository'e aImage atılacaksa o repo adı):v1(tag):** Docker Hub üzerindeki repository'leri(Docker tarafında Image'ların tutulduğu yer) public ya da private olarak Image'larımıza gönderme
- **docker run -d –name container_adı -p 5002:80 image_adı:versiyon :** İlgili konteyner ayağa kaldırılır fakat içeirsine bağlanmaz
- **docker build –no-cache -t image_name:version .(Dockerfile dosyasını nerede bulacağı):** Yeni image oluştururken tüm layerlar sıfırdan başlatılır.
- **docker rm $(docker ps -a -q):** Tüm containerları siler.

Asp.Net Core Uygulamasını nasıl Dockerize yaparız?

1- Uygulama oluştur.
2- Uygulamayı Publish et.
3- Projenin Dockerfile dosyasını oluştur ve içeriğini yaz.
4- Image dosyasını oluştur.
5- Container oluştur.
   Container oluştururken -p parametresi ile işletim sisteminin portu ile container portunu eşitlememiz gerekiyor. (docker run -p 5000:80 image_adı)

.Net Core CLI nedir?

.Net Core CLI, bilgisayara bir sdk kurduğumuzda otomatik olarak gelir. Bir komut satır arayüzüdür.

Bu CLI ile core uygulamaları oluşturabiliriz, var olan uygulamaların build ve publih işlemlerini gerçekleştirebiliriz.

Neden Dockerfile içerisinde .Net Core CLI komutları kullanmak isteriz?

- Her image yapmak istediğimizde projeyi elle publish yapmak zorunda kalmayız. Çünkü image içerisinde komutla publish yapma işlemini gerçekleştiririz.

Docker MultiStage Build

Uygulamayı inşa ederken birden fazla base imagee den yararlanıyorsak buna multistage denir.

Mutlaka kullanmaya çalışmalıyız.

Neden birden fazla base image kullanılır?

- Var olan image nin boyutunu küçültmemizi sağlar.

.dockerignore dosyası?

Copy komutu kullanırken biz bazı klasör ve dosyaları image içerisine kesinlikle kopyalama diyebiliriz.

Proje root unda .dockerignore adında bir dosya oluştururuz. Bu dosyaya kopyalanmasını istemediğimiz dosyaları belirtebiliriz.

.dockerignore faydaları:

- Image oluşma süresini kısaltır.
- Image boyutu azalır.

Docker Volume

Docker konteynırlar tarafından üretilen dataları kalıcı kılmamızı sağlayan bir yöntemdir.

Datayı kalıcı kılma yöntemleri:

- bind mount : İşletim sistemi üzerindeki bir dosyaya konteynırlar tarafından üzertilen dataların kaydedilmesi işlemidir. Artık bir konteynırın ürettiği datayı başka bir konteynırda erişebilecek.
- volume : Docker CLI tarafından yönetilen ve Docker ın kendi içerisinde bir alanı oluşturmasıdır. Uzak, cloud ortamdaki Docker volume lerede veri gönderebiliriz. Hem Linux konteynırlarda hem de Windows konteynırlarda kullanılabilir.
- tmpfs mount : Konteynırlar tarafından üretilen veriler işletim sistemi üzerindeki memory de kaydedilir.

❖ Volume

Konteynırlar içerisinde bir volume, bir dosya bağlamadığımız zaman her konteynır içerisinde bir dosya kaydettiği zaman kendi dosya sistemi çerisinde kaydetmiş olur. Yani konteynırlar birbirinden bağımsız olmuş olur.

Konteynırlar içerisindeki datalar kalıcı değildir. Fakat volume kullanırsak bu datalar kalıcı hale gelir.

Hem Linux için hem de Windows konteynırlar için volume oluşturabiliriz.

docker volume create valıme_name: İlk olarak docker içerisinde bir volume oluştururuz.

docker run -d -p 5000:4500 –name conteyner_name -v volume_name:bağlanacak dosya yolu image_name : Konteynır oluşturur ve oluşturulan konteynırın volume dosyasını yolu verilen dosya ile bağlar ve volume dosyasının verilerini bağlanan dosyanın verileri ile aynı yapar.

docker volume rm volume_name : önceden oluşturulan volume silinir.

❖ Bind Mount

Docker ın data kaydedeceği klasöre erişebilmesi için öncelikli olarak Docker a izin vermek gerekiyor. İlk olarak dosya paylaşım işlemini yapmalıyız.

Bind mount ta işletim sisteminde herhangi bir yere data kaydı yapabiliriz.

docker run -d -p 5000:4500 –name konteynır_name –mount type=bind,source="İşletim sistemindeki dosya yolu",target="Docker içerisindeki kayıt yapılan dosya yolu" İmage_name : Sonrasında İşletim sisteminde kayıt yapacağımız dosya ile Docker üzerinde kayıt yapacağımız dosyayı birbirine eşitlememiz gerek. Böylece Docker da ilgili dosyaya kayıt yapınca buraya kayıt yapmayacak, işletim sisteminde seçtiğimiz dosyaya kayıt yapmış olacağız. Konteynır ayağa kalkarken İşletim sisteminde bağladığımız dosya içerisindeki dataları çeker.

Docker Environment

Asp.Net Core tarafında bir uygulama oluşturulduğunda default olarak üç environment oluşur.

Uygulama exe de çalıştığı zaman yani sunucudan çalıştığı zaman ortam default olarak production oluyor. Local de çalıştırdığımız zaman ortam default olarak development oluyor.

Konteynır içerisinde çalıştığında default olarak production olur. Ortamı istersek development olarak da ayağa kaldırabiliriz.

!!! Biz production ortamda bir hata aldığımızda hatanın detayını göremeyiz. Eğer ortamımız development ise hatanın detayını görebiliriz.

docker run -p 5000:4500 –env ASPNETCORE_ENVIRONMENT=PRODUCTION –name container_name image_name: Container oluşturulurken ortamı production olarak verildi.

docker run -p 5000:4500 –env ASPNETCORE_ENVIRONMENT=DEVELOPMENT –name container_name image_name: Container oluşturulurken ortamı development olarak verildi.

- Environment ile ortam belirleriz.
- Environment bize appsettings.json içerisinde tanımlamış olduğumuz değişkenleri ezme imkanı sunar. (Sabit değişkenler appsettings.json içerisinde tutulur.)

<none> image?

Docker bir önbellek yani cach leme mekanizması oluşturmak için bu image leri kullanır. İstersek nu <none> image leri silebiliriz. Silmek de faydalıdır.

docker images -f "dangling=true": Sadece <none> image leri listeler.

docker rmi $(docker images -f “dangling=true” -q): Id’si alınan tüm <none> image ler silinir.

Çok katmanlı Asp.Net Core(mvc) projesini dockerize yapmak

Her katmanı ayrı ayrı COPY almalıyız. Çünkü ilgili katmanda değişiklik olduğunda tüm projenin kopyası yerine ilgili katmanın kopyası alınır.

Publish yapılırken her katman değil sadece web projesinin olduğu katman publish yapılır. Sebebi sadece web projesinin ayağa kalkmasıdır.

Unit Test içeren Asp.Net Core(mvc) projesini dockerize yapmak

dotnet test: Test projesini çalıştırıp sonuçlarını görebiliriz.

Image oluşturma konutu öncesi test koyarsak ve test başarısız olursa image oluşturulmaz.

Docker Compose

Bilgisayarımıza Docker kurduğumuzda bir Docker CLI birde Docker Compose CLI gelir. Docker Compose CLI bir tooldur. Bu tool biden faza container ve imageyi kolay bir şekilde yönetmemizi sağlar. Formatı YAML’dır.

docker compose version: Docker’ın güncel versionu

Visual Studio DockerFile

- Expose

  Bu komut olmasa sadece imagenin içerisindeki uygulamaya erişiriz. Docker içerisindeki farklı bir container içerisindeki microservice bizim microservicemize erişemez.

Bu komut ile Docker içerisinde çalışan bu container içrisinde çalışan bu serviceye başka container içerisinde çalışan microserviceler belirtilen porttan bu serviceye erişebilir.

- Expose ve -p yoksa dışa ve içe açılamaz.
- Sadece expose varsa içe açılır fakat dışa açılmaz
- Sadece -p varsa dışa ve içe açılır.

- Copy

```
COPY ["MicroService1.API.csproj", "MicroService1.API/"]
RUN dotnet restore "./MicroService1.API/MicroService1.API.csproj"
COPY . ./MicroService1.API/
```

Şeklinde iki ayrı copy kullanmamızın sebebi, projede en küçük bir değişiklikte tüm projenin kopyalanmasını önlemektir. Bu ayrım sayasinde projeye packet ekleme ya da

.csproj'da bir değişiklik olmadığı sürece proje cache'den alınır. Bu iki durumun olması durumunda tüm proje sıfırdan kopyalanır.

- Publish

  Proje publish edilince iki dosya oluşur. Projenin .exe ve .dll dosyası oluşmuş olur. Docker .dll dosyasını kullanır.

  .exe linux da çalışmaz. Sadece Windows da çalışır.

docker.compose.yml

Docker compose, birden fazla imagelarımızı-containerları tek bir komut ile ayağa kaldırmamızı sağlar.

docker compose build: Image oluşturur.

docker compose create: Image oluştuysa container oluşturur. Containerları çalıştırmaz.

docker compose start: Containerları çalıştırır.

docker compose stop: Çalışan containerları durdurur.

docker compose rm: Durmuş containerları siler.

docker compose up: (build + recreate + start) Imageleri sıfırdan build eder. Containerları sıfırdan oluşturur ve containerları başlatır.

docker compose up -d: (build + recreate + start) Imageleri sıfırdan build eder. İçerisine bağlanmadan containerları sıfırdan oluşturur ve containerları başlatır.

docker compose down: (stop + remove) Containerları durdurur ve siler. Up metodu ile ayağa kalkan volume varsa onlarıda siler. Up komutu ile ne ayağa kalkarsa down metodu ile silinir.

docker compose pause: Containerın memorysi serbest bırakılmaz.Yani memoryde bir değer tutuluyorsa geri unpause yapıldığında bu değer kaybolmaz. Stop yapıldığında memory serbest bırakılır. Yani memoryde tutulan değer varsa bu değer kaybolur.

docker compose exec service_name /bin/bash: İsmi girilen servisin komut satırına bağlanır.

docker compose up –scale service_name:intance_sayısı: Bir containerın birden fazla intance ayağa kaldırılabilir ve sayıları da ayrı ayrı belirtilebilir. Belirli bir port aralığı verirsek bazen port çakışması olabiliyor.

docker compose push: Herhangi bir container depolama servisine image gönderebiliriz. Docker compose da ne kadar image varsa hepsini tek bir komutla göndeririz.

- Docker composede her containerın bir localhostu vardır.