İşletim Sistemi

CPU: işlemci

RAM: Görevi üzerinde geçici bir süre bilgi tutmak, bunu CPU'ya aktarmak ve CPU'dan gelen veriyi geçici olarak tutmaktır.xaz

Anakart(Motherboard): Tüm ünitelerin birbirleri ile haberleşmelerini sağlayan elektronik yolların olduğu cihaz. Bilgisayarı oluşturan tüm bileşenleri birbirine bağlar.

BIOS(Basic Input-Output System): Veri deposudur. Bilgisayarın açma kapama düğmesine basıldığı anda yapılan ilk işlem bu depo üzerinde saklanan uygulamanın çalışmasıdır. Bu uygulamanın ilk yaptığı şey anakarta takılı tüm cihazları test etmektir. Hangi porta hangi cihaz takılı düzgün çalışıp çalışmadığını test eder. Ardından harddiske gider ve işletim sistemini RAM'e kaydeder ve çalıştırır. Yetkiyi işletim sistemine devreder.

Kernel-Çekirdek

Bilgisayardaki uygulamaların çalışmasına izin veren yazılımdır. Kısacası uygulamalar ile bilgisayar donanımı arasında köprü görevi görür.

Kernel, işletim sisteminin kendisidir. Ana görevi kurulu olduğu bilgisayarın tüm kaynaklarını yönetmek ve bu kaynaklarla uygulamalar arasında köprü görevi oluşturmaktır.

Sanallaştırma

Sunucu israfı sorununa karşılık geliştirilen bir teknolojidir.

Temelde bir fiziksel cihaz üzerine birden fazla sanal makine kurmamıza imkan sunan, kaynak dağıtımını ve ortak kaynak kullanımını sağlayan sisteme verilen isimdir.

Container

Konteyner içerisinde uygulamanın çalışması için gerekli tüm ek paketler bulunur.

Docker

2013 senesinde ortaya çıktı.

Docker, uygulama geliştirmek, dağıtmak ve çalıştırmak için oluşturulan bir platformdur.

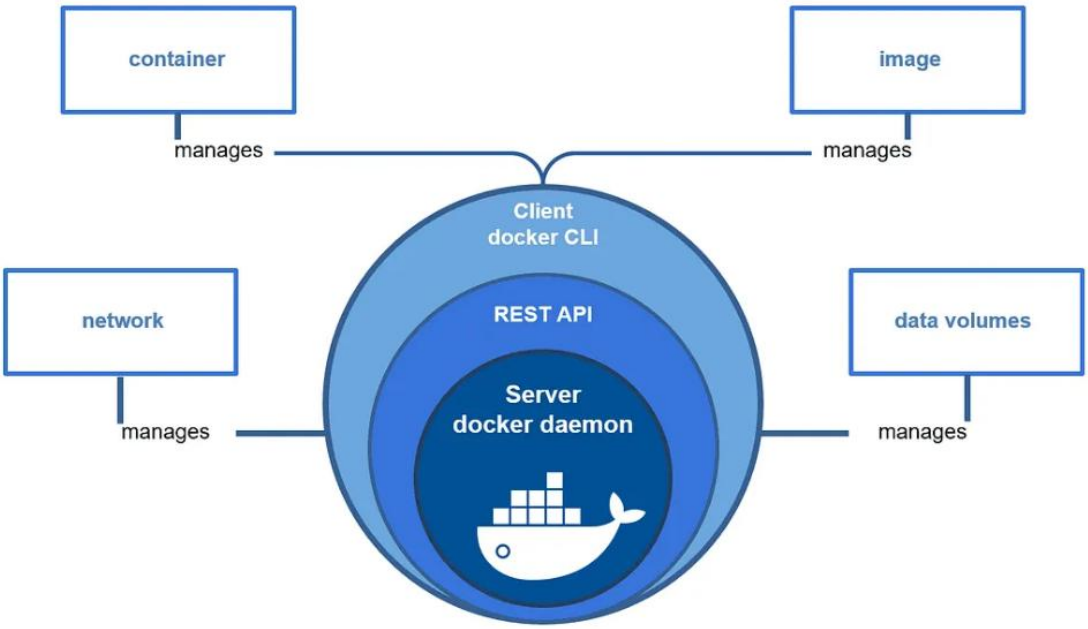
- Uygulamayı altyapıdan bağımsız kılar.
- Yazılım üretim ve dağıtım sürecimizi hızlandırır.
- Docker ile altyapımızı yönetebiliriz.

Docker Engine

Docker platformunun kalbidir.

1- Docker CE
(ücretsiz)
2- Docker EE
(ücretli)

Docker Engine = Docker Daemon + Docker Rest API + Docker CLI



- Docker Daemon

Imageler, konteynerlar, networkler ve volümeler gibi Docker objelerini yaratmamızı ve yönetmemizi sağlar. Bir Rest API ile dış dünya ile görüşebilir.

- Docker Rest API

Diğer programlar, Docker Daemon ile Rest API aracılığı ile konuşur ve ona ne yapması gerektiğini söyler. En önemli kullanıcısı Docker CLI'dır.

- Docker CLI

Docker komut satırıdır.

Image

Bir uygulamanın ve o uygulamanın çalışması için gerekli tüm ek kütüphane ve diğer öğelerin paketlenmiş haline denir.

Sadece okunabilir bir dosyadır.

İçerisinde Kernel yoktur. Çünkü Image ler üzerinde koştukları işletim sisteminin Kernel ını kullanır.

Docker Image: Bir temel işletim sistemini alıp üzerine yaptığımız her değişikliğin ayrı bir dosya katmanı olarak tutulduğu ve en sonunda tüm bu katmanların tek bir bütün gibi çalışmasının sağlandığı Docker objesidir.

Container

Oluşturulan Image nin çalışır halidir.

Hangi sisteme hangi Docker kurulacak?

Linux → Docker Engine CE

Mac OsX → Docker Desktop for Mac

Windows 10 Pro-Ent-edu → Docker Desktop for Windows

Windows 7-8 ve 10 Home → Docker Toolbox

Docker Hub

Imag' leri depoladığımız yerlere Image Registry denir. Bu depoların en bilineni Docker Hub'dır. Docker Engine 'nin varsayılan olarak baktığı alandır.

İki amacı vardır:

1- Docker'ın resmi Image'lerini burada tutmak.
2- Kullanıcıların kendi yaptıkları Image'leri depolamsı.

Docker'la ilgili dökümantasyon hariç her şeye Docker Hub'dan ulaşabiliriz.

Docker CLI (Command Line Interface)

Docker yüklendiğinde otomatik olarak yüklenen ve Docker Engine'i yönetmemizi sağlayan komut satırı arayüzüdür.

Docker CLI'da kullanılabilecek kodlar üç ana başlığa ayrılır.

1- Options

   Docker Daemon'a bağlanırken kullanabileceğimiz çeşitli opsiyonlar bulunur.

2- Management Commands
3- Commands

Volume

Conteynerlar uzun ömürlü değildir. Fakat verilerin uzun ömürlü olması gerekir. Bu sorunu çözmek için volume geliştirilmiştir.

Container silinsede volume silinmez.

Bir volumeyi birden fazla containera bağlayabiliriz.

Bind Mounts

Sadece Local Development, yani sadece kendi bilgisayarımızda geliştirme yaparken kullanmalıyız.

Bind Mounts, Docker Engine nin yüklü olduğu host sistemlerdeki bir klasör ya da dosyayı container içerisine map etmeye yarayan yöntemin adıdır.

Volume atar gibi bind mount oluşturabiliriz.

docker container run -it -v Bilgisayardaki_dosya:container_içerisinda_bağlanacak_klasör image_name sh: Yeni bir container oluşturur ve buna bind mount ile bilgisayardaki bir dosyayı container içerisinde istediğimiz klasöre bağlar ve sh bağlantısı yapar.

Docker Plugin-Diriver Sistemi

Dacker “batteries include but removable: pilleri içerisinde ama isterseniz değişebilirsiniz” mantığını yürütmüştür. Firmanın ürün geliştirme politikası budur.

- Depolama
- Günlük dosyalama
- Şifreleri saklayabilecek altyapı
- Network

Docker bu dört alanda hem kendisi diriver sunar hem de firmaların kendi diriverlarını kullanma imkanı sunar.

Plugins: Docker’da varsayılan diriverları gösterir.

Docker Network Driver

Docker arka planda onlarca farklı servisten oluşmasına ve altyapısal olarak karmaşık olmasına rağmen kullanımda yıllardır alıştığımız ve bildiğimizin dışında bir sistem sunmaz.

Container’ın temel ağ kullanımının bir Linux sunucunun ağ kullanımından farkı yoktur.

Tüm iletişim altyapısını Docker Network objeleri ile sağlarız.

Docker Network Driver:

1- Bridge(Köprü)

Varsayılan driverdır. Biz container oluştururken aksini belirtmezsek bu bridge networküne bağlanır. Ya da biz yeni Docker network objesi oluştururken bir diriver belirtmezsek varsayılan olarak bridge alınır.

2- Host

Bazı durumlarda sistemde çalışan containerın arada herhangi bir izolasyon olmadan üzerinde koştuğu hostun direkt bir parçası olmasına ihtiyaç duyulur. Yani containerın üstünde çalıştığı ağ kartına yapışması gerekir. Arada herhangi bir paket çevirimi olmaması gerekir. Bu durumda host driver kullanılır.

Her sistemde host adında bir docker ağ objesi bulunur ve bu host driver ile oluşturulmuştur.

3- MacVlan

Bu driver ile oluşturulan bir docker network objesine bağlı docker containerlara direkt MAC adresi atayarak mevcut ağa bağlı birer fiziksel cihaz gibi davranmaları sağlanabilir. Docker network trafiğini containera bu MAC adresi üzerinden yönlendirir.

Nat veya ip routing yapmadan containerın ağ ile haberleşmesi sağlanır.

4- None

Container hiçbir şekilde ağ bağlantısına sahip olmasın istenirse bu driver ile oluşturulan docker objesine bağlanmamız yeterli.

5- Overlay

Ayrı hostlar üstünde çalışan containerların aynı ağda çalışıyormuş gibi çalışması istendiğinde Overlay ile oluşturulan networkler sayesinde sağlarız.

Docker kurulumu ile üç adet network objesi gelir.

- bridge
- host
- none

Biz varsayılan olarak gelenlerden kullanmak zorunda değiliz. Yeni network objeleri oluşturup kullanabiliriz. None ve host için yeni objeler oluşturmamıza gerek yok. Fakat bridge için yeni network objesi oluşturabiliriz. Bu bize network izolasyonu sağlar.

Kullanıcı tanımlı bridge network'e bağlı containerlar birbirleriyle isimler üzerinden haberleşebilirler. DNS çözümlemesi sağlar. Yani aynı bridge üzerine bağlı iki container isimleri üzerinden de haberleşebilir.

Port Publish

Dış dünyadan containerımızdaki bir servise ulaşmak istersek bu containerın dış dünyaya açık olması gerekir. Yani dış dünya ile iletişim içerisinde olması gerekir. Bunu da port publish denilen işlem ile yaparız.

Örn: **-p 80:80 / --publish 80:80**

-p host_port(host üzerinde açmak istediğimiz port):container_port(container içerisinde dinleyen port)

Bir belirtme yapmazsak varsayılan tcp protokolünde port açmış oluruz. Eğer udp protokolünde port açmak istiyorsak belirtmeliyiz. **(-p 53:53/udp**)

Logging

Tüm servis ve uygulamalar, bizim gördüğümüz ana ekranlar dışında da birçok kayıt tutar. Adım adım hangi işlemleri gerçekleştirdiğini, bu işlemleri gerçekleştirirken hangi adımları attığını ve eğer bu süreçte bir hata oluştuysa bunu ve daha birçok detayı bizlerin erişebileceği şekilde kaydederler. Buna logging yani günlük kaydı tutma diyoruz.

İşletim sistemleri kendi alt servisleri ile ve genel davranışları ile ilgili çeşitli seviyede bilgiyi bizlerin erişebileceği şekilde düzenli olarak kaydederler.

Linux → root/var/log klasörüne kaydedilir.

Windows → Event viewer altından erişilebilecek şekilde Windows Logs altında kaydedilir.

Linux sistemlerde uygulamalar çalıştığında bu uygulamalara giriş çıkış, stdin()-stdout()-stderr() adında üç genel akış aracılığı ile sağlanır.

Stdin() → Klavyeden ya da başka bir uygulamadan giriş içerebilen uygulamanın giriş akışıdır.

Stdout() → Bir uygulamanın normal çıktısıdır.

Stderr() → Hata mesajı göndermek için kullanılan çıktıdır.

Environment Variables / Ortam Değişkenleri

Değişkenler(variables) bize dinamik uygulamalar yapma imkanı sunar.

Environment variables, işletim sisteminin tamamında geçerli olan ve her yerden çağırılabilen isimlerdir. Yani işletim sistemi bazında tanımlanır.

Windows:

Get-ChildItem Env: Sistemdeki tüm environmeent variablesları listeler.

$Env:env_name: Belirtilen environment değeri gösterilir.

$Env:environment_name="environment_değer": Yeni bir environment oluşturulur.

Linux:

printenv: Sistemdeki tüm environmeent variablesları listeler.

echo $environment_name: Belirtilen environment değeri gösterilir.

export environment_name="environment_değer": Yeni bir environment oluşturulur.

- Environment variables, container ortamlarında image oluşturulurken ya da container oluşturulurken tanımlanan değerlerdir.

docker container run -it –env env_name=env_deger ubuntu bash: Container oluştururken env oluşturup container içerisine giriş yaptık.

Environment oluştururken büyük küçük harfe dikkat etmeliyiz. Çünkü var1 ile VAR1 aynı şeyler değildir.

İstersek containerın çalıştığı sistem içerisinde bulunan environmentıda container içerisine gönderebiliriz.

docker container run -it --env TEMP ubuntu bash: Container oluştururken sistemde bulunan TEMP environmentını container içerisine gönderir.

Environmentları bir dosyaya atıp o dosyayı container oluştururken bu dosya üzerinden tanımlama yapabiliriz.

docker container run -it --env-file .\env.list ubuntu bash: Containerı oluştururken environmentları dosya üzerinden tanımlarız. Dosya içerisindeki tüm environmentları container içerisine aktarır.

docker container run --env veritabani=prod.pizzadukkani.com ozgurozturknet/env1: Oluşturduğumuz container veritabani adındaki environment aracılığı ile değerleri alır.

Docker Image

Docker image, uygulamamızın paketlenmiş haline verilen isimdir. Uygulamamızı image olarak paketliyoruz. Daha sonra bundan containerlar oluşturuyoruz. Containerı çalıştırdığımızda uygulamamız çalışıyor.

İmage Registry, Docker imageleri depolayabildiğimiz ortamlardır. (örn: Docker Hub) Public ortamlar olduğu gibi private ortamlarda olabilir.

- Docker Image İsimlendirme Yapısı

Docker imagelerine verilen isim o imajın depolandığı yeri de belirtir.

Her objenin benzersiz bir ID'si vardır ve Docker objeler bu ID'ler üzerinden tanınır.

Docker Image isimleri üç temel kısımdan oluşur.

1- Registry URL : Imagenin durduğu ya da duracağı registry url'i belirtir.
2- Repository: Imagenin ne olduğunu ve kimin tarafından oluşturulduğunu belirtir.
3- Tag: Versionu belirtir. Aynı repositoryden birden fazla imageye ulaşmayı sağlar. Biz tag vermezsek Docker varsayılan olarak latest tagını verir.

Registry URL / Repository : Tag(örn1: docker.io/ozgurozturknet/adanzyedocker:latest örn2: docker.io/Ubuntu:18.04)

Aynı ID'ye birden fazla tag verebiliriz.

- Docker Hub

Varsayılan Image registrydir.

- DockerFile

Docker imageler, DockerFile talimatnamesine göre yaratılır. Bir nevi Docker imagenin kodudur.

Docker Image Oluşturma

Docker Image oluşturmak için ilk önce Dockerfile adında bir dosya oluşturmalıyız. Dockerfile bize Docker Image yaratma imkanı sunan ve kendine özel talimatları olan bir text dosyasıdır.

FROM | Oluşturulacak imajın hangi imajdan oluşturulacağını belirten talimat. Dockerfile içerisinde geçmesi mecburi tek talimat budur. Mutlaka olmalıdır. Ör: FROM ubuntu:18.04 LABEL | İmaj metadata'sına key=value şeklinde değer çiftleri eklemek için kullanılır. Örneğin team=development şeklinde bir etiket eklenerek bu imajın development ekibinin kullanması için yaratıldığı belirtilebilir. Ör: LABEL version:1.0.8 RUN | İmaj oluşturulurken shell'de bir kom ut çalıştırmak istersek bu talimat kullanılır. Örneğin apt-get install xxx ile xxx isimli uygulamanın bu imaja yüklenmesi sağlanabilir. Ör: RUN apt-get update WORKDIR | cd xxx komutuyla ile istediğimiz klasöre geçmek yerine bu talimat kullanılarak istediğimiz klasöre geçer ve oradan çalışmaya devam ederiz.

Ör: WORKDIR /usr/src/app

USER | gireceğimiz komutları hangi kullanıcı ile çalıştırmasını istiyorsak bu talimat ile onu seçebiliriz.
Ör: USER poweruser

COPY | İmaj içine dosya veya klasör kopyalamak için kullanırız
Ör: COPY /source /user/src/app

ADD | COPY ile aynı işi yapar yani 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

ENV | Imaj içinde environment variable tanımlamak için kullanılır
Ör: ENV TEMP_FOLDER="/temp"

ARG | ARG ile de variable tanımlarsınız. Fakat bu variable sadece imaj oluşturulurken yani build aşamasında kullanılır. Imajın oluşturulmuş halinde bu variable bulunmaz. ENV ile imaj oluşturulduktan sonra da imaj içinde olmasını istediğiniz variable tanımlarsınız, ARG ile sadece oluştururken kullanmanız gereken variable tanımlarsınız.
Ör: ARG VERSION:1.0

VOLUME | Imaj içerisinde volume tanımlanamızı sağlayan talimat. Eğer bu volume host sistemde varsa container bunu kullanır. Yoksa yeni volume oluşturur.
Ör: VOLUME /myvol

EXPOSE | Bu imajdan oluşturulacak containerların hangi portlar üstünden erişilebileceğini yani hangi portların yayınlanacağını bu talimatla belirtirsiniz.
Ör: EXPOSE 80/tcp

ENTRYPOINT | Bu talimat ile bir containerın çalıştırılabilir bir uygulama gibi ayarlanabilmesini sağlarsınız.
Ör: ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]

CMD | Bu imajdan container yaratıldığı zaman varsayılan olarak çalıştırmasını istediğiniz komutu bu talimat ile belirlersiniz.
Ör: CMD java merhaba

HEALTHCHECK | Bu talimat ile Docker'a bir konteynerin hala çalışıp çalışmadığını kontrol etmesini söylebiliriz. Docker varsayılan olarak container içerisinde çalışan 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.
Ör: HEALTHCHECK --interval=5m --timeout=3s CMD curl -f http://localhost/ || exit 1

SHELL | Dockerfile'ın komutları işleyeceği shell'in hangisi olduğunu belirtiriz. Linux için varsayılan shell ["/bin/sh", "-c"],Windows için ["cmd", "/S", "/C"]. Bunları SHELL talimatı ile değiştirebiliriz.
Ör: SHELL ["powershell", "-command"]

docker image build -t seyma5/merhaba . : Dockerfile'ın uzantısı yoks, yazılan bir Dockerfile'den image oluşturma.

docker image build -t seyma5/merhaba -f Dockerfile .txt . : Dockerfile'ın bir uzantısı varsa, yazılan bir Dockerfile'den image oluşturma.

docker image history image_name: Imagenin geçmişini görürüz.

- Biz bir uygulamayı, container imagesi haline çevirirken resmi image kullanmalıyız. Bu daha kolay ve daha güvenlidir.

docker container run --rm -p 80:5000 image_name: Container oluşturur ve durunca silinir.

- Dockerfile içerisinde talimatların sıralaması önemlidir. Değişmeyecek talimatlar üste değişebilecek talimatlar alta yazılmalı. Mümkün olduğu kadar çok önbelleğin kullanılması sağlanmalıdır. Bu sayede image oluşturma süreleri de kısalır.

1- **Exec Form**

Komutları parantezler şeklinde yazma formudur.

Komut shell formunda girilirse, Docker image oluşturduğunda bu komutu varsayılan shell'i çalıştırarak onun içerisine girer.

Herhangi bir Shell çalışmadığı için bazı değerlere erişemezler.

Eğer Entrypoint ve CMD kullanılacaksa Exec form kullanılmalıdır.

2- **Shell Form**

Direkt komut olarak yazma formudur.

Komut bu formda girildiyse herhangi bir Shell çalıştırmaz ve komut direkt olarak çalıştırılır.

Shell formu kullanıldığında CMD'deki komutlar ENTRYPOİNT'e parametre olarak aktarılmaz.

Multi-stage Build

Docker tarafından 2017 senesinde yayınlanan Multi-stage build özelliği bizim imaj yaratma aşamasını kademelere bölmemize ve ilk kademede oluşturduğumuz imaj içerisindeki dosyaları bir sonraki kademede oluşturacağımız imaja kopyalayabilmemize imkan sağlar.

- Bu sayede imaj boyutumuzun küçülmesine imkan tanır.

Build ARG

Dockerfile içerisine ARG VERSİON yazıp alttaki kodla farklı versionda image kolaylıkla oluşturabiliriz.

docker image build -t x2 --build-arg VERSİON=3.8.1 .

Arg sadece image oluştururken kullanılır.

- Image oluşturmak için sadece Dockerfile ve build yöntemlerini kullanmak zorunda değiliz. Bir başka yöntemde container çalıştırıp içerisinde istenilen değişiklikleri yapıp daha sonrada docker commit ile bu containerı image olarak kaydetmektir.

  docker commit con1 seyma5/con1:latest: con1 containerını, seyma5/con1:latest tagli image yapar.

  docker commit -c 'CMD {"jave", "uygulama"}' con1 seyma5/con1:ikinci: -c ile artı özellikler verebiliriz.

Docker Save-Load

docker save seyma5/con1:latest -o con1imaj.tar: Sistemde bulunan bir imageyi .tar dosyası olarak kaydedebiliyoruz.

docker load -i .\con1imaj.tar: Bu dosyayı internet erişimi olmayan makineden kopyalayabilir ve load ile bu .tar dosyasından image oluşturabiliriz.

Kendi Docker Reegistry'mizi oluşturma

docker pull registry: registry imagesini çekme

docker run -d -p 5000:5000 --restart always --name registry registry: Yeni bir regirtry containerı oluşturmak

docker image tag ozgurozturknet/hello-app:latest 127.0.0.1:5000/hello-app:latest: var olan imageye yeni tag vermek

docker image push 127.0.0.1:5000/hello-app:latest: local registrye imageyi atmak

http://127.0.0.1:5000/v2/_catalog adresinden atılan imagelere bakılır

Docker Komutlar

- **docker container run –name container_name image_name :** Yeni bir konteyner oluşturma
- **docker container run -d -p 80:80 İmage_name :** Yeni konteynerı oluşturur ve arka planda çalıştırır.
- **docker logs container_name:** Oluşturulan containerın loglarını görme
- **docker exec -it docker_name sh:** Çalışan Container'ın Shell'ine bağlanır.
- **docker start container_name:** Durdurulan konteynır tekrar çalıştırılır.
- **docker container prune:** Sistemde çalışmayan tüm container ler silinir.
- **docker image prune -a:** Sistemde kullanılmaya tüm image ler silinir.
- **docker volume create volume_name:** Yeni bir volume oluşturur.
- **docker volume inspect volume_name.** Var olan volumenin ayrıntılarını gösterir.
- **docker container run -it -v volume_name:container_içerisinda_bağlanacak_klasör image_name sh:** Yeni bir container oluşturur ve buna var olan bir volumeyi container içerisinde istediğimiz klasöre bağlar ve sh bağlantısı yapar.

- **docker container run -it -v volume_name:container_içerisinda_bağlanacak_klasör:ro image_name sh:** Yeni bir container oluşturur ve buna var olan bir volumeyi container içerisinde istediğimiz klasöre bağlar ve sh bağlantısı yapar. Fakat bağlanan klasör sadece okunur.
- **docker container run --rm -it ozgurozturknet/adanzyedocker sh:** Yeni bir container oluşturulur ve içine sh ile bağlanır. Sonrasında kapatıldığında container silinir.
- **docker container run image_name varsayılan_olarak_başlatılacak_uygulama**: Yeni bir container oluşturur ve varsayılan olarak belirtilen uygulama çalışır.
- **docker network ls:** Sistemde bulunan network nesnelerini listeler
- **docker network inspect network_obje_name:** Bize yazılan network objesinin tüm özelliklerini verir.
- **docker container run -it --name container_name --net network_driver mage_name sh:** Bir container oluşturur ve onu yazılan network drivera bağlar. Oluşturulan container içerisine bağlanır.
- **docker network create bridge_name:** kullanıcı tanımlı bridge network oluşturma
- **docker network create –**driver host: host network oluşturma
- **docker container run -dit --name container_name --net network_name image_name sh:** yeni bir container oluşturur bunu verilen networke bağlar ve bağlantıyı arka planda kurar çıkar
- **docker attach container_name:** verilen containera bağlanır
- **docker network create --driver=bridge --subnet=10.10.0.0/16 --ip-range=10.10.10.0/24 --gateway=10.10.10.10 network_name:** Bir network objesi oluştururken kendi belirlediğimiz ip aralığı ile oluşturduk.
- **docker network connect network_name containet_name:** çalışan containerı verilen network objesine bağlar
- **docker network disconnect network_obje_name container_name:** Çalışan containerın network objesi ile bağlantısını kesme
- **docker network rm network_obje_name:** Network objesini siler. Fakat bu objeye bağlı bir container varsa silme işlemini yapmaz. Hata verir.
- **docker logs container_name**: stdout() ve stderr()'a gönderilen tüm mesajları listeler. Container silinene kadar oluşan tüm loglar listelenir.
- **docker logs --details container_name:** Daha fazla detaylı log listesini gösterir.
- **docker logs -t container_name:** Logları zaman damgalı gösterir.
- **docker logs --until hangi_zamana_kadar container_name:** Belirli bir zaman kadar olan loglar listelenir.
- **docker logs --since hangi_zamandan_sonra container_name:** Belirtilen zamandan sonra oluşan loglar listelenir.
- **docker logs –tail son_kaç_satır container_name**: Belirtilen sayı kadar oluşan son satırları listeler.
- **docker logs -f container_name:** Logları canlı olarak izleriz.
- **docker container run --log-driver log_driver_name image_name:** Containerın log driverını istediğimiz şekilde belirleyebiliriz.
- **docker top container_name:** Containera bağlanmadan hangi proceslerin çalıştığını görürüz.
- **docker stats:** Containera bağlanmadan sistemin ne kadar CPU, RAM, disk ve network kullandığını görmek için kullanılır. Docker hostu üzerinde çalışan tüm containerları listeler ve yeniler.

- **docker stats container_name:** Sadece belirtilen containerın ne kadar CPU, RAM, disk ve network kullandığını görmek için kullanılır.
- **docker container run --memory=memory_miktarı -d image_name:** Memory limitli container oluşturma
- **docker container run --memory=memory_miktarı –memory-swap=swap_miktarı -d image_name:** Memory limitli container oluşturma ve yedek memory alanı oluşturma
- **docker container run -d --cpus="1.5" image_name:** Container oluştururken ne kadar CPU kullanacağını belirleme. Toplam cpu'nun 1.5 tanesini kullan.
- **docker container run -d --cpuset-cpus="0,3 image_name:** Container oluştururken ne kadar CPU kullanacağını belirleme. Sistemdeki CPU0 ve CPU3'ü kullan.

Notlar...

Bilgisayar aracılığı ile yaptığımız şey uygulama çalıştırmaktır.

- Imajdan bir container oluşturduğumuzda varsayılan olarak çalışması için ayarlanmış bir uygulama vardır. Bu uygulama çalıştığı sürece container ayakta kalır. Uygulama çalışmayı bıraktığında container da kapatılır.
- Container oluşturulduğunda birden fazla uygulama içerisinde olabilir. Fakat olomatik çalışması için sadece bir uygulama ayarlanabilir. Bu uygulamanın yanında başka uygulamalarda çalıştırılabilir.
- Containerlar tek bir uygulama çalıştırmak için oluşturulur.

Conteynır içerisine bağlanıp değişiklik yapabiliriz.

Containerlar üzerinde çalıştıkları sistemin CPU ve memory kaynaklarını herhangi bir kısıtlama olmadan kullanabilirler. Biz bu kullanımlara sınırlar koyabiliriz.

Image'da sadece okunabilir katmanlar bulunur. Image üzerine boş bir yazılabilir katman eklendiğinde container oluşturulmuş olur.

Image ➔ R/O Layers

Container ➔ R/O Layers + R/W Layer

Container içerisinde yaptığımız tüm değişiklikler bu yazılabilir katmana yazılır ve sadece değişikliğin yapıldığı container için geçerlidir. Docker image'ı değişmez.

Docker depolama altyapısında, birleşim dosya sistemi adını verdiği yapıyı kullanır. Docker Image!leri mevcut bir base image üzerine inşa edilir.

Aynı Image'den oluşmuş olsada iki container birbirinden bağımsız yapılardır. Birinde yapılan değişiklik diğerini etkilemez.

-it = --interactive –tty

-dit = -d + -it bağlantıyı arka planda kur

Ctrl + PQ: Container ile bağlantıyı keser fakat containerı kapatmaz.

Windows ve Mac kurulu bir bilgisayara Docker kurunca, Docker arkada içerisinde küçük bir Linux yüklü sanal makinayı ayağa kaldırıp Docker Daemon'u bunun içerisine kurar. Client'ı ise direkt işletim sisteminin içerisine kurar ve ikisinin birbiri ile konuşabileceği şekilde ayarlama yapar.

Container oluştururken biz bir isim vermez isek Docker kendi bir isim verir. Bu ismin ilk kısmı İngilizce sıfatlardan ikinci kısım ise bilişim dünyasındaki önemli şahsiyetler ve bilim insanlarının isminden oluşur.

Çalışan bir conteynır direk silinemez. Ya silmek için ilk olarak durdurup sonra silmeliyiz ya da -f parametresinide kullanarak silmeliyiz.

Docker bir Linux konteynır platformudur.

Sunucu – Server

- Hizmet sunar
- Güçlü ve yüksek kapasiteli işlem gücü
- Uzun süreli, kesintisiz ve çoklu isteklere cevap vermek üzere tasarlanmıştır

İstemci – Client

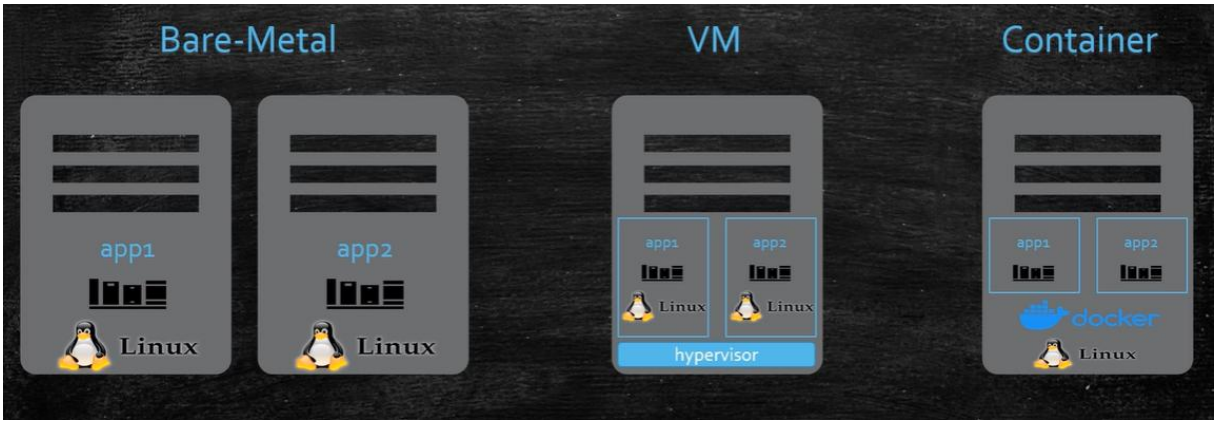
- Hizmet kullanır
- Görece düşük kapasiteli işlem gücü
- Tek bir kullanıcıya hizmet vermek için tasarlanmış, uzun süreli ve kesintisiz çalışma öncelikli değildir

Bilgisayarlar temelde matematiksel işlemler yapan cihazlardır.

Bir yazılım işletim sistemine gerek duymadan direk bilgisayara yüklenerek çalışabilir.

İşletim sistemi = Uygulamalar + Kullanıcı Arayüzü + Kernel

Bare-Metal → VM → Container



VM tam bir işletim sistemi barındırır. Fiziksel makinayı sanallaştırırız. İşletim sistemi izolasyonu sağlar.

Container de işletim sistemi bulunmaz. Uygulamayı sanallaştırır. Uygulama izolasyonu sağlar.

Docker Compose

Bizim birden fazla containerden oluşan uygulamaları tanımlamamıza ve ayağa kaldırıp çalıştırmamıza yarar. Compose ile Uygulamamızı yapılandırmak için YAML dosyası kullanılır. Yani Docker Compose bize çoklu container sistemlerini YAML dosyası olarak tanımlamamızı sağlar.

YAML, insanların kolayca okuyup anlayabilmeleri için oluşturulmuş bir ver serilizasyon dilidir. Genelde konfigürasyon dosyaları oluşturmak için bu dil kullanılır.

Compose prod. Ortam için kullanışlı değıldir. Geliştirme aşamasında kendi bilgisayarımızda kullanmamız için idealdir.

Docker Compose, Docker Engine ile yüklü gelmez. Ayrıca indirip yüklememiz gerek. Eğer Docker Desktop kullanıyorsak burada yüklü gelir.

docker-compose up -d: Docker compose yaml dosyasını çalıştırır.

docker-compose down: Docker compose yaml dosyası ile kurulan şeyleri geri alır.

- Down ile sadece volume silinmez. Docker compose içerisinde, Dockerfile ile oluşturulan imagede silinmez.

Docker Compose = Docker Compose YAML + Docker Compose CLI

- Docker Compose CLI komutları sadece docker-compose.yml dosyasının olduğu klasörün içerisindeyken çalışır.
- Docker Compose için service=container ‘dır.

docker-compose config: Docker compose yaml içeriğini listeler.

docker-compose images: Oluşturulan servislerin hangi imageden oluşturulduğunu listeler.

docker-compose logs: Ayaktaki bütün servislerin loglarını gösterir.

docker-compose exec service_name komut_name: Docker compose ile oluşturulan servisin içerisinde komut çalıştırabiliriz.

Docker compose oluştururken, Dockerfile kullanarak compose içerisinde yeni bir image oluşturabiliriz.

Docker Orchestration

Containerların dağıtımını, yönetimini, ölçeklendirilmesini ve ağ oluşturmasını otomatikleştirme ve zamanlama işlemlerine denir.

Çok sayıda containerı bir arada yönetmek için kullandığımız temel yazılımlardır.

Container Orchestration Araçları:

- docker SWARM
- MESOS
- kubernetes

Kubernetes

Borg → Omega → Kubernetes

(Proprietary) (Proprietary) (open-source)

Google daha sonra kubernetes'i CNCF'e bağışladı.

Docker Swarm

Docker Engine'e entegre gelir. Kullanımı ve yönetimi kubernetes'e göre oldukça kolay ve basittir.

Docker ana bilgisayar havuzunu tek bir sanal ana bilgisayara dönüştürür.

Docker Swarm ortamı hazırlamak için gerekenler ve hazırlama aşamaları:

- Sunucuların birbirleri ile hızlı ağ bağlantısı olmalı.
- TCP port 2377(Cluster yönetimi)
- TCP ve UDP port 7946 (Nodeler arası iletişim)
- UDP port 4789 (Overlay Network için)
- İşletim sistemleri kurulu olmalı.
- Docker Engine'nin son sürümü kurulmalı.
- Bir sunucuda **docker swarm init** komutunu vermeliyiz. Bu makine tüm docker ortamını yönetir. Bu makine manager node olur.
- Daha sonra tek tek tüm makinalara bağlanıp swarm modunu devreye almalıyız. Bu makinalar worker node olur.

- **docker swarm init** komutunu girdiğimiz anda, Docker bu sistemi Engine moddan alıp Swarm moduna geçirir.

Manager node, worker node ile haberleşirken bu trafik şifrelenmiştir.

- Raft Algoritması lider seçimi için kuralları belirler. Raft algoritması maksimum (N-1)/2 sayıda replikanın devre dışı kalmasını tolere eder. Bu algoritmanın çalışması için her zaman tek sayıda manager node ile kurulması gerekir.

docker swarm init –advertise-addr node_ip: Swarm modunu aktif hale aldık.

docker swarm join-token manager: Manager node yapmak için gereken komutu verir.

docker swarm join-token worker: Worker node yapmak için gereken kodu verir.

docker node ls: Sistemdeki nodeler hakkında bilgi verir.

docker service create –name service_name –replicas=5 -p 8080:80 image_name: Servis oluştururken kaç tane container olsun(5 container oluşturulur) onu belirleriz.

docker service ps service_name: İsmi verilen servisin detaylarını listeler.

- Varsayılan olarak manager nodelerde birer worker nodedir.

reboot -f: Node kapatma komutu.

docker service ls: Docker'da çalışan servisler listelenir.

docker service scale service_name=adet: Adı girilen servisi büyütür ya da küçültürüz. Verilen sayıda task yapar.

docker service rm service_name: Servisi silme.

- Docker Swarm Cluster'da oluşturabileceğimiz en temel obje servislerdir.

Docker Service Mode

- Replicated
  Oluşturmak istediğimiz servisin kaç replica içereceğini belirtiriz. Swarm uygun olan nodelar üstünde o sayıda replica oluşturur.
- Global
  Oluşturmak istediğimiz servisin kaç replica içereceğini belirtmeyiz. Swarm altındaki her node üstünde 1 replica oluşturulur.

docker service create --name service_name –mode=global image_name: Global modda service oluşturma.

- Swarm altında yaratılan servisler aynı overlay network üzerinde birbirlerine servis isimleriyle ulaşabilir. Docker burada hem dns çözümlemesi hismeti hem de koad balancing hizmeti sunmaktadır.

docker service update –datach –update-delay 5s –update-parallelim 2 –image yeni_image_name:version service_name : Servisteki imageyi yeni versionuyla güncelleme

docker service rollback –detach service_name: Bir önce ki update ettiğimiz şeyleri geri alır.

Docker Secret

Containerlarda plain text olarak tutmamızın güvenlik zafiyeti yaratabileceği kullanıcı adı ve şifre gibi verileri secret objeleri şeklinde encrypted olarak transfer edebiliriz.

secret oluşturma yolu 1:

notepad kullaniciadi.txt

notepad sifre.txt

docker secret create kullanici_adi .\kullaniciadi.txt

docker secret create sifre .\sifre.txt

secret oluşturma yolu 2:

echo "bu bir denemedir" | docker secret create secret_name - : Girilen echo secret olarak kaydedilir.

docker service create -d --name service_name --secret kullanici_adi --secret sifre --secret deneme image_name: Secretı service ekleme

- Eklenen secretlar /run/secrets içerisine eklenir.(docker exec -it service_id sh komutu ile bağlanıp bakılabilir.)

docker service update –secret-rm secret_1 –secret-add secret_2 service_name: Secretı değiştirme

- Secret içeriği değiştirilmez. Yeni oluşturup eskisini silip yenisini yüklemeliyiz.

Docker Stack

Birden fazla servisi tek bir konfigürasyon dosyası altında oluşturmamızı sağlar.

docker stack deploy -c .\docker-compose.yml ilkstack : Compose dosyasından stack oluşturma