Docker-Notes.md

Docker Kurulum

Docker Ubuntu kurulum

sudo apt-get update -y sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

```
sudo apt-get install -y docker-ce \
                        docker-ce-cli \
                        containerd.io \
                        docker.io
sudo usermod -aG docker ubuntu
newgrp docker
sudo systemctl start docker
sudo systemctl enable docker
```

Docker AWS Linux kurulum:

```
sudo yum update -y
sudo yum install -y docker
sudo usermod -aG docker ec2-user
newgrp docker
sudo systemctl start docker
sudo systemctl enable docker
```

Docker Compose kurulum

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

- Docker Compose'a executable iznini ver:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Docker Komutları

```
docker version

docker info

docker "OPTIONS" inspect "OPTIONS_parametre"

docker run --name "verilen_isim" "image_adı"

docker container logs "container_id"

docker container run -d -p 80:80 nginx

docker container rm "id1" "id2" "id3"

docker container exec -it "websunucu" sh

docker container prune
```

`docker image pull -q "image_adı"` "q" stands for quiet. Suppresses the verbose output

`docker ps -l` command displays only the last container to exit.

`docker sytem prune` Makinadaki tüm çalışmayan container, kullanılmayan network, image ve build cache'leri siler.

`docker image prune` Dangling image'ları siler(dangling image means that you've created the new build of the image, but it wasn't given a new name. So the old images you have becomes the "dangling image".)

`docker image prune -a` unused ve dangling'leri siler. "image" yerine "container", "network" ve "volume" yazılabilir.

Docker notes

- `docker image pull "image_name"` komutunu kullandığımızda image orjinal değilse 2 katman iner. İlk katman orjinal image, ikinci katman da ilk imajın üstüne yapılan değişikliğin katmanıdır.
- `ctrl+p+q` ile container kapatılmadan çıkılır.

- Default'da açılan portların hepsi TCP olarak açılır, aksi `-p 80:80/udp` yazılarak belirtilmeli. "-p" parametresi ile sadece bir port belirtilirse externalport otomatik belirlenir.
- `-d ve -it beraber -dit olarak da kullanılabilir. Açılan termimal arkaplanda açılır.`
- `docker cp container:/usr/src/app destination` Container içinden dosya almak için kullanılır.

Volume Commands

```
docker volume create "volume_name"
docker volume inspect "volume_name"
docker container run -it -v volume_name:/uygulama alpine sh
docker container run -it -v volume_name:/uygulama:ro alpine sh
#"ro" stands for read only. you can't chance anything in this volume.
docker container run --rm -it nginx sh
#"rm" command will delete container when you stop it.
```

Volume Notes

- Eğer bir volume mount edildiği klasör mevcut değilse bu klasörü yaratır. Ve o anda volume içerisinde hangi dosyalar varsa bu klasörde de o dosyaları görürsünüz. Boşsa boş görürsünüz.
- Eğer bir volume imaj içerisinde bulunan mevcut bir klasöre mount edilirse:

A: Klasör boşsa o anda volume içerisinde hangi dosyalar varsa bu klasörde de o doşyaları görürsünüz.

B: Klasörde dosya varsa ve volume boşsa bu sefer o klasördeki dosyalar volume'e kopyalanır.

C: Klasörde dosya var ya da yok fakat volume'de dosyalar varsa yani volume boş değilse, bu sefer siz o klasörün içerisinde volume'de ne dosya varsa onu görürsünüz.

Bind Mount

bind mount local'deki bir directory'yi container'a mount etmeyi sağlar. Devolopment yaparken kullanılır.

örn.:

```
docker container run -d -p 80:80 -v "local_directory:container_mount_point" nginx
# while creating a bind mount use the "-v" parameter just like volumes.
```

Log Commands

```
docker logs "container_name"
docker logs --details "container_name" # Daha çok detay için.
docker logs -t "container_name"  # Zaman detayı için.
docker logs --help
docker logs --until "log_time" "container_name"
# Belirtilen zamana kadar oluşan loglar listelenir.
docker logs --since "log_time" "container_name"
# Belirtilen zamandan itibaren oluşan loglar listelenir.
docker logs --tail "tail_number" "container_name"
# Belirtilen sayıya göre son loglar listelenir.
docker logs -f "container_name"
# Gerçek zamanlı takip(-f) edilir.
docker container run --log-driver "logging_driver" "image"
# Container belirtilen logging driver ile çalışır.
```

Log Notes

- loglar `/dev/fd/` altındaki stdin(0), stdout(1) ve stderr(2)'den alınır.
- `-f` stands for follow.
- Default logging driver "json-file" dır. Docker awslogs, fluentd, gcplogs, gelf, journald, json-file, local, logentries, splunk ve syslog driver'larını da destekler. İstenilen driver default olarak seçilebilir.

Env-Variable Comands

```
docker container run -it --env Var1=deneme1 ubuntu sh
#her variable için --env ya da -e paremetresi ayrı ayrı verilmeli
docker container run -it --env TEMP ubuntu sh
# Host'dan variable aktarmak için.
```

Nots

- Environment variable'lar case sensitive'dir.
- Host'un Environment'ları da container'a aktarılabilir. `--env "buraya host'daki variable'nın adı."`
- Çoklu variable tanımla için `--env-file "file_name_path"` kullanılabilir.

Network Commands

```
docker network inspect "network_name"
docker container run -it --name deneme1 --net host nginx sh
# "--net" ile bağlanılacak network'ü seçtik.
docker network create --driver=host "network_name"
# Network türü belirtilmezse network default'da bridge oluşur.
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"
docker network connect "network_name" "container_name_or_id"
# Container'ı network'e bağlamak için.
docker network disconnect "network_name" "container_name_or_id"
# Container'ı network'den disconnect etmek için.
```

Notes

- Aynı network'e bağlı tüm container'lar birbirlerinin ismini çözebilirler. Kullanıcı tanımlı bridge'lerde DNS hizmeti bulunur.
- Varsayılan dışında ip aralıkları tanımlanabilir.
- Containerlar çalışır durumdayken de kullanıcı tanımlı bridge networklere bağlanıp, bağlantıyı kesebilirler.
- Containerlar arası network izolasyonu sağlamak istersek ayrı bridge networkler yaratarak bunu sağlayabiliriz.
- Oluşturulan Network'ün subnet, ip range ve gateway'i belirtilebilir: `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"`
- Kullanıcı tanımlı bridge network'lere bağlı olan bir container başka bir network'e bağlabilir. Container default bridge'e bağlıysa bu işlem gerçekleşmez.
- Container'a bağlı olan bir network silinemez.("-f" dahil) Containerların silinmesi veya durdurulması gerekir. Network silme işlemi bağlı olan container'ların durdurulması yolu ile gerçekleştiyse, o container'lar tekrar start edilemez ve başka bir network'e bağlatılamaz. Network ID kullanılarak aynı network yaratılırsa container tekrar çalışabilir.

Resource Commands

```
docker tap "container_name"
# Belirtilen container içindeki process'leri listeler

docker stats "container_name"
# Belirtilen container'ın kaynak kullanımını gerçek zamanlı olarak listeler.
#Komutta container belirtilmez ise çalışan tüm container'ler listelenir.

docker container run -d --memory=100m "container_name"
# Ram sınırlandırmak için. (megabayt=m, gigabayt=g, kilobayt=k)
```

```
docker container run -d --memory=100m --memory-swap=100m "container_name"
# Ram Swap miktarı için.

docker container run -d --cpus="1.5" "container_name"
# CPU sınırlandırmak için.

docker container run -d --cpus="2" --cpuset-cpus="0,3" "container_name"
# Kullanılması istenilen CPU core'ları belirtmek için.
```

Dockerfile

- `FROM` Dockerfile içinde zorunlu olan tek komuttur. Image'ın hangi Imaj'dan oluşturulucağı belirtilir.
- `RUN` Oluşturulucak olan Image'ın Komut satırında çalıştırılmak istenen komut yazılır. Örn.: `RUN apt-get install -y curl`
- `WORKDIR` Default olarak çalışılmak istenen istenen path belirtilir. Belirtilen yol yoksa oluşturulur.
- `COPY` Image içine kopyalanmas istenen dosya belirtilir. Örn.: `COPY ./deneme.py /home/container/`
- `EXPOSE` Image'dan oluşturulacak container'ların hangi portlardan yayın yapacağı belirtilir.
- `CMD` Container oluşturulduktan sonra default olarak çalıştırılması istenen komut girilir. `RUN` 'dan farkı ise `RUN` Image'ın oluşturulması için gereken bir veya birden fazla komutu kapsarken, `CMD` o image'dan yaratılmış olan container'ın içinde default olarak çalışacak komut içindir.
- `HEALTHCHECK` Belirtilen parametrelere gör container'ın sağlık durumunu sorgular. Örn.: `HEALTHCHECK --interval=5m --timeout=3s`
- Dockerfile'da yapılan tüm değişiklikler, yapılan değişiklikten sonraki katmanları da etkiler. Öncesi için cache'den işlem yapar.
- `ADD` `COPY` 'ile aynı işlemi yapar. Farklı olarak web sunucudan kopyalama yapar. Dosya sıkıştırılmışsa hedefe açarak atar.
- `ARG` Dockerfile içinde değişken tanımlamaya yarar. "ARG" build aşamasında kullanılır. "ENV" `container run` aşamasında. Örn.:

```
FROM ubuntu:latest
WORKDIR /gecici
ARG VERSION
```

```
ADD https://www.python.org/ftp/python/${VERSION}/Python-${VERSION}.tgz .
CMD ls -al
# Built aşamasında "ARG" değeri verilir.
docker image build -t app:ARG --build-arg VERSION=3.7.1 .
```

Commands

```
docker image build -t deneme/merhaba .
# Komut Dockerfile'ın olduğu directory'de çalıştırılır.
docker image build -t deneme/merhaba -f "dosya_adı" .
# Dockerfile'ın adı farklı ise veya komut Dockerfile'ın oldu directory'de değilse
# "-f" verilerek dosya ve yolu belirtilir.
docker image tag "source_image" "new_image_tag"
# Image'a tag vermek için.
```

- docker image history "image name" image'nin katmanları/geçmişini gösterir.
- `-t` ile image'a bir tag verilir.

Docker commit ile image oluşturma

`docker commit "container_name" "image_name":latest` `docker commit` komutunu verirken de Dockerfile komutları girilebilir: `docker commit -c 'WORKDIR /app/' "container_name" "image_name"`

- Image'ları save ve load etmek için:

```
docker save "image_name" -o "OutputName.tar"
docker load -i ./"saveName"
```

Multi-stage Build

- Image oluşturma işlemi sırasında birden fazla image oluşturulabilir ve bunlar arasında etkileşim sağlanabilir. Buna "Multi-stage Build" denir. Aşağıda verilen örnekte bir Dockerfile'dan iki image(stage) oluşturulmuştur. İlk stage'de Java source code'u build(compile) edilmiş, ikinci stage'de ise bu build çekilip yeni bir image oluşuturulmuştur.

```
# Java code'unun Build işlemi için jdk imajından base image oluşturuluyor
FROM mcr.microsoft.com/java/jdk:8-zulu-alpine AS derleyici
# Working directory seçiliyor.
WORKDIR /usr/src/uygulama
# W.D. içine java kaynak kodu atılıyor.
COPY /source /usr/src/uygulama
# Java kaynak code'u build ediliyor.
```

```
RUN javac uygulama.java

# jre base image seçiliyor.
FROM mcr.microsoft.com/java/jre:8-zulu-alpine
WORKDIR /uygulama
#derleyici ismi verilen build stage'inde oluşturulmuş olan build edilmiş code,
#yeni stage'e alınıyor.
COPY --from=derleyici /usr/src/uygulama .
CMD ["java", "uygulama"]
```

Docker Compose

Installation

- curl ile compose dosyasını indir:

```
DOCKER_CONFIG=${DOCKER_CONFIG:-$HOME/.docker}
mkdir -p $DOCKER_CONFIG/cli-plugins
```

```
curl -SL https://github.com/docker/compose/releases/download/ \
v2.5.0/docker-compose-linux-x86_64 -o $DOCKER_CONFIG/cli-plugins/docker-compose
```

- Dosyayı executable yap: `chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose`
- Version check: `docker compose version`

Notes

- version-2'de hyphen(-) yerine 'space' verilebilir. Örn.: " `docker-compose` `docker compose` "
- Oluşturulan servisler "bulunduğu-klasörün-adı_Verilen-ad" şeklinde adlanır.

Commands

```
docker compose up -d
docker compose down # oluşan tüm servisler silinir.
docker compose build
# compose aşamasında Dockerfile ile image oluşturulduysa "docker compose up"
# komutu tekrarlandığında hali hazırda yarılmış olan image kullanılır.
#Dolayısıyla yapılan değişiklikler image'a uygulanmaz.
#Bunun için bu komut tekrar girilerek image yeniden oluşturulur.
```

Örnek docker-compose.yml:

```
docker-compose for wordpress website
version: "3.7"
services:
  db-server:
    image: mysql:5.7
    restart: always
    volumes:
      - datas:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    networks:
      - wpnet
  wordpress:
    image: wordpress:latest
    depends_on:
      - db-server
    restart: always
    ports:
      - "80:80"
    environment:
      WORDPRESS_DB_HOST: db-server:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    networks:
      - wpnet

volumes:
  datas:

networks:
  wpnet:
    driver: bridge
```

Docker Swarm

Commands

```
# Swarm'ı aktive etmek için:
docker swarm init --advertise-addr 172.31.21.200

# Manager token'ını almak için:
```

```
docker swarm join-token manager

# Worker token'ını almak için:
docker swarm join-token worker

# Örnek bir container oluşturma komutu:
docker service create --name test --replicas=5 -p 8080:80 nginx

docker service rm "service_name"
```

Scaling

```
docker service scale test=3
```

Update and Rollback

```
docker service update --detach -- update-delay 5s --update-parallelism 2 --image worc
# update edilir.

docker service rollback --detach websrv
## update geri alınır.
```

Secret

docker create secret "name" "./file_name" or echo "password" | docker secret create "secret_name" -

- secret'lar container içindeki "/run/secrets" directory'sindedir.

docker service update --secret-rm "old_secret" --secret-add "new_secret" "secret_name" "service_name"

Swarm notes

- Ports: 2377/TCP Cluster yönetimi için 7946/TCP-UDP Node'lar arası iletişim 4789/UDP Overlay Network
- İletişim etcd ile şifrelenmiştir.
- Manager sayısı her zaman 1, 3, 5 ,7 ... şeklinde tek sayı olarak ilerler.
- ideal Manager sayısı 3 olmakla birlikte, 7 Manager'den sonrası sağlıksızdır.
- Kubernetes'in aksine defaul'da Manager'ler da worker görevi görür

- Overlay Network'ler ile aynı ağda olmayan makineler aynı service'e alınabilir.
- Default'da ingress Overlay network kullanılır.