|  |  |
| --- | --- |
| **DOCKER BASİC** |  |
| docker run -i -t ubuntu bash | Container başlat interaktif modda yani container shelline bağlan -d yada bağlanma -i interaktif -t shell tahsis |
| docker run -it --name techpro ubuntu bash | Containera isim ver |
| docker ps -a ya da docker container ls -a | Containerları listele (-a inaktifleride listele) |
| docker start romantic\_colden yada stop | Container başlat yada durdur. |
| docker images or docker image ls | İmageları listele |
| docker rm container ID veya name | # inaktif containerlari siler |
| docker rm -f container ID veya name | # statusu ne olursa olsun containeri siler. |
| docker container prune  docker rm -f $(docker ps -aq) | # tüm stop etmiş tüm containerları siler.  # hepsini siler |
| docker exec 1db32 ls pwd date  docker exec -it 1db32 bash | Container shelline bağlan ya da komut çalıştır |
| **DOCKER VOLUME** |  |
| docker volume ls | Volume listele |
| docker volume create vol | Volume oluştur |
| docker volume inspect name | Volume incele |
| docker run -it --name techpro -v vol:/data alpine ash | Containera volume bağla vol=volume adı /data mount point |
| docker volume rm vol ---- docker volume prune | Volume silme |
| docker run -d --name nginx-new -p 8080:80 -v /home/ec2-user/webpage:/usr/share/nginx/html nginx | Bind Mount 8080 host 80 container port |
| **DOCKER NETWORK** |  |
| docker network ls | Netwotkleri listele |
| docker network inspect bridge | less  docker network inspect techpronet | Networkleri incele |
| docker container inspect techpro2nd | grep IPAddress | Container Ip incele |
| docker network create --driver bridge techpronet | User defined network oluştur. |
| docker container run -dit --network techpronet --name techpro1st alpine ash | Containerı oluştururken networke bağlama Eğer network belirtmezsek default bridge de oluşur. |
| docker network connect techpronet Container1 | Containerı sonradan networke dahil etme. Container 2 networkte olabilir. |
| docker network rm techpronet | Network sil |
| docker container run --rm -d -p 8080:80 --name ng nginx | --rm: Bu seçenek, konteynerin durdurulduğunda otomatik olarak silinmesini sağlar. |
| None network  docker container run --rm -it --network none --name nullcontainer alpine | Bu, konteynerin dış dünya ile hiçbir şekilde iletişim kuramayacağı anlamına gelir. Konteyner sadece kendi içinde çalışan uygulamalar için kullanılabilir ve dış ağ kaynaklarına erişimi yoktur. |
| Host network  docker container run --rm -dit --network host --name my\_nginx nginx | Konteyner, Docker host'unun ağını doğrudan kullanır ve host'un IP adresi ve portları üzerinden erişilebilir hale gelir. Bu, ağ performansını artırır fakat izolasyonu azaltır. |
| **DOCKER İMAGE** |  |
| docker image ls  docker images | İmage listele |
| docker image pull ubuntu | İmage çek indir |
| docker image inspect ubuntu:18.04 | İmage incele |
| docker search ubuntu | Docker hub image arar |
| docker build -t “<dockerhubusername>/flask-app:1.0” . | İmage oluştur |
| docker run -d --name welcome -p 80:80 <dockerhubusername>/flask-app:1.0 | İmage kullan |
| docker login | Docker hub login |
| docker push <dockerhubusername>/flask-app:1.0 | Docker hub gönder |
| docker build -t "<dockerhubusername>/flask-app:2.0" -f ./Dockerfile-alpine . | Dockerfile isminden farklı bir isim kullanma |
| docker image tag <dockerhubusername>/flask-app:2.0 techpro/flask-app:latest | Tag değiştirme |
| docker image rm 497 | İmage sil |
| **DOCKER COMPOSE** |  |
| docker-compose up | Docker compose çaılıştırı. Containerlar ayağa kalkar |
| docker-compose ps | Docker compose ları listele |
| docker-compose help | Docker compose yardım |
| docker-compose images | Docker compose imageları listele |
| docker-compose config | Docker compose config bilgileri |
| docker-compose down | Docker compose siler. Containerlar silinir imagelar kalır |
| docker-compose up --build | İmageları günceller. Sistem çalışmaya devam eder. |

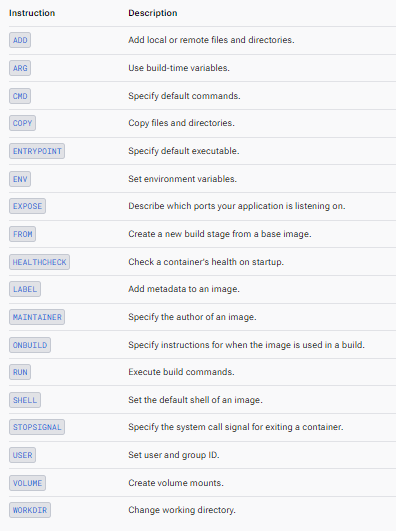
**DOCKERFİLE ARG/ENV VE CMD/ENTRYPOİNT İLİŞKİSİ**

|  |  |
| --- | --- |
| **ARG** değerlerinin Docker imajının oluşturulması sırasında kullanılırken, **ENV** değerlerinin oluşturulan imajdan türetilen konteynerlerde kullanılmasıdır. | |
| FROM python:3.8-slim-buster  ARG APP\_PORT=8000  ENV APP\_ENV=testing  EXPOSE $APP\_PORT  RUN echo $APP\_PORT  RUN echo $APP\_ENV | docker build --build-arg variable\_name=new\_value -t image\_name .  docker build --build-arg APP\_ENV=production --build-arg APP\_PORT=8000 -t myapp . |
| Docker ARG ve ENV, Dockerfile içinde değişkenleri tanımlamak için kullanılan iki farklı komuttur. Ancak, aralarındaki temel fark, ARG komutlarının container oluşturulduğunda ve sadece "docker build" komutu ile kullanılabildiği iken, ENV komutları container oluşturulduktan sonra ve "docker run" komutu ile kullanılabilir. | |

|  |  |
| --- | --- |
| Docker ENTRYPOINT komutu, konteyner başlatıldığında çalıştırılacak komutu belirtir. Ayrıca, docker run komutunda belirtilen komutlar ENTRYPOINT komutunun parametreleri olarak eklenir. Bu, konteyneri çalıştırırken ENTRYPOINT komutunun parametrelerini değiştirmek için docker run komutunu kullanmanızı sağlar. | |
| Docker CMD komutu, konteyner başlatıldığında çalıştırılacak varsayılan komutu belirtir. Bu komut, docker run komutunda belirtilen komutlar tarafından geçersiz kılınabilir. | |
| FROM nginx:latest  ENTRYPOINT ["echo", "deneme"] | docker build -t my-nginx .  docker run -d -p 80:80 my-nginx  docker run -d -p 80:80 my-nginx ls |

|  |  |
| --- | --- |
| FROM nginx:latest  CMD ["echo", "deneme"]  CMD ls | docker build -t my-nginx .  docker run -d -p 80:80 my-nginx  docker run -d -p 80:80 my-nginx ls  docker run -it my-nginx bash |

|  |  |
| --- | --- |
| **SHELL VE EXEC FORM** | |
| CMD ["executable", "param1", "param2"] | Exec form-env variable erişemez-direk process başlatır |
| CMD command param1 param2 | Shell form-env variable erişebilir. Shell açıp orada icra eder. |

**DOCKERFİLE COMMAND**