A’dan Z’ye Docker Eğitimi

Bilgisayar Bileşenleri ve Kavramları

Bilgisayar, birçok temel bileşenin bir araya gelmesiyle çalışır. İşte bu bileşenlerin temel işlevleri:

- **CPU (İşlemci)**: Bilgisayarın beyni olarak görev yapar ve tüm hesaplamaları gerçekleştirir. Örneğin, mouse’u sağa sola oynatmak birçok matematiksel işlemin sonucudur.
- **RAM**: Bilgiyi geçici olarak saklar ve CPU’ya aktarır. Bu geçici depolama, hızlı veri erişimi sağlamak için kullanılır.
- **Anakart**: Tüm bileşenlerin birbirleriyle iletişim kurmasını sağlar. Diğer tüm donanım bileşenleri anakarta bağlanır ve haberleşme bu bağlantılar üzerinden gerçekleşir.
- **Harddisk**: Kalıcı depolama birimidir. Veriler ve işletim sistemi burada saklanır.
- **GPU**: Görsel hesaplamaları yapar. Tüm işlemleri görsel olarak monitöre aktaran bileşendir.

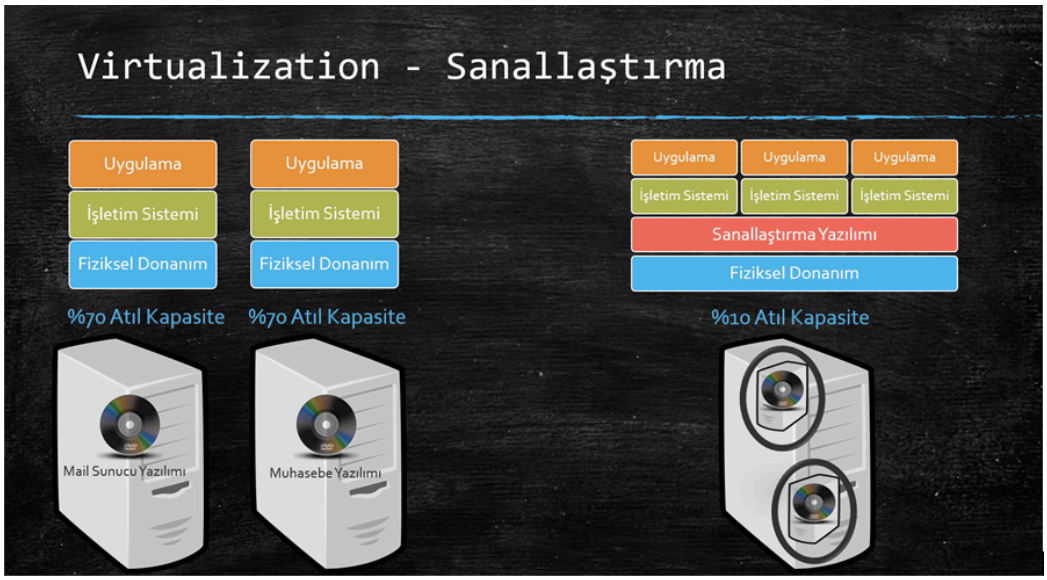
Bilgisayarın Başlatılması ve BIOS İşlevi

- **BIOS**: Bilgisayarın açma/kapama düğmesine basıldığında ilk olarak **BIOS** devreye girer. BIOS, anakartta bulunan ve bilgisayarı başlatan temel bir yazılımdır. Bilgisayar açıldığında, BIOS anakarta bağlı tüm bileşenleri kontrol eder.
- **BIOS ve İşletim Sisteminin Yüklenmesi**: BIOS, harddiskte işletim sistemini arar ve bulduğunda bu işletim sistemini RAM’e yükler. Ardından bilgisayarı başlatır ve kontrolü işletim sistemine devreder.

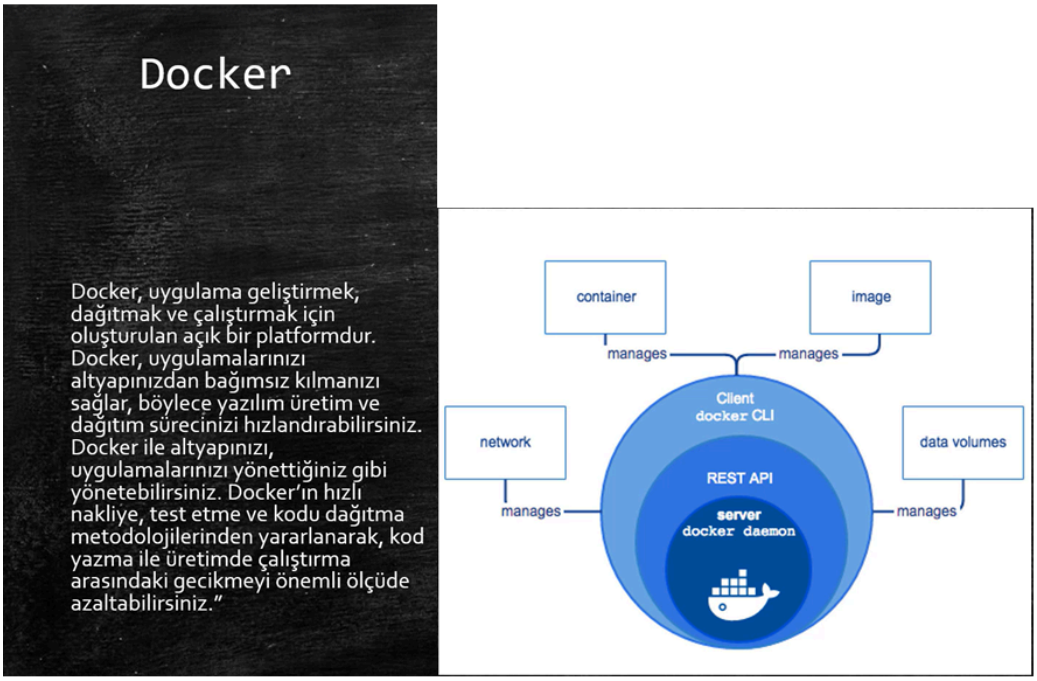
Diğer Temel Kavramlar

- **İşletim Sistemi (OS)**: Kullanıcının bilgisayarda uygulama çalıştırmasını sağlayan temel yazılımdır. Tüm donanım ve yazılım kaynaklarını yönetir.
- **Kernel**: İşletim sisteminin çekirdeği olarak görev yapar. Bilgisayarın donanım kaynaklarını (CPU, RAM, harddisk vb.) yönetir ve bu kaynaklar ile uygulamalar arasında köprü görevi görür.
- **Server ve Client**:
  - **Server (Sunucu)**: Hizmeti sağlayan sistemdir.
  - **Client (İstemci)**: Bu hizmeti kullanan sistemdir. Örneğin, Facebook sunucu üzerinden hizmet sağlarken, kullanıcı arama yaparken istemci tarafında işlemi gerçekleştirir.

SANALLAŞTIRMA TEKNOLOJİSİNE GEÇİŞ



DOCKER TEKNOLOJİSİ



DOCKER ENGINE KAVRAMI

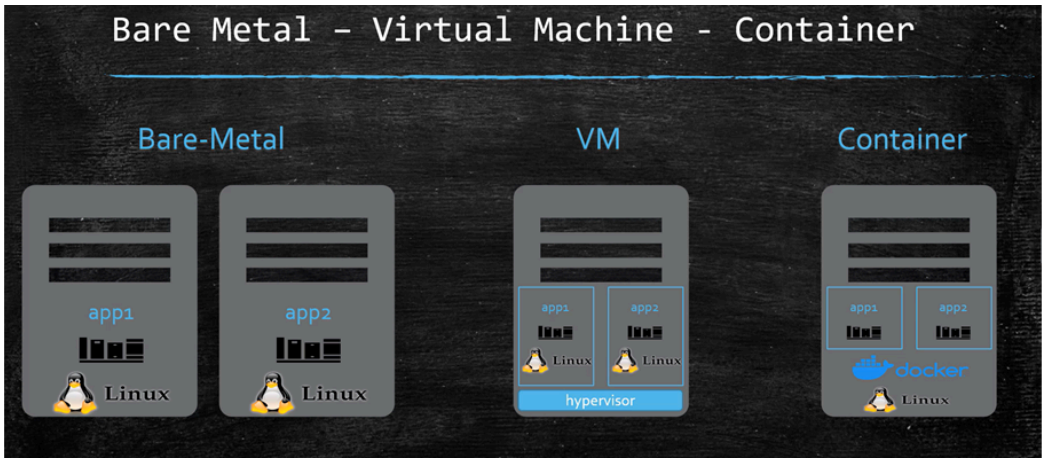
Docker Engine, Docker'ın çalışmasını sağlayan temel bileşenlerin bütünüdür. Üç ana bileşenden oluşur:

- **Docker Daemon**: Docker objelerini (Image, Container, Network ve Volume) yaratmamızı ve yönetmemizi sağlar. Docker arka planda çalışan bir servis olarak çalışır ve container işlemlerini yönetir.
- **REST API**: Docker Daemon, dış dünya ile iletişim kurmak için bir **REST API** sunar. Bu API, Docker objelerini oluşturmak, listelemek, silmek gibi işlemler için kullanılabilir.
- **Docker CLI**: Docker Komut Satırı Arayüzü (CLI), kullanıcıların Docker ile etkileşime geçmesine olanak tanır. `docker run`, `docker build`, `docker ps` gibi komutlar Docker CLI üzerinden çalıştırılır.

IMAGE VE CONTAINER

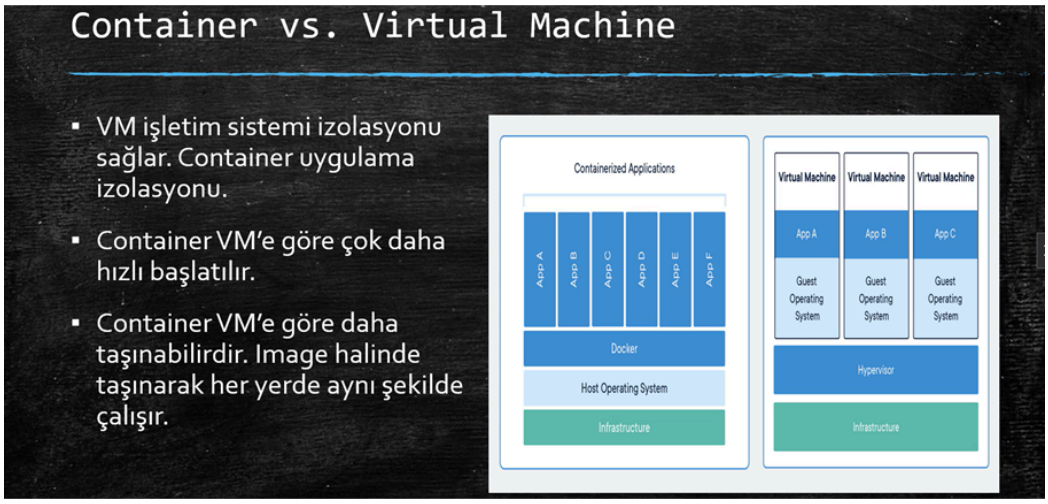
Docker dünyasında, Image ve Container kavramları en temel yapı taşlarıdır.

- **Image**: Bir Docker Image, bir uygulamanın ve çalışması için gerekli tüm kütüphane ve bağımlılıkların paketlenmiş halidir. Docker Image, uygulamayı bağımsız bir ortamda çalıştırmaya hazır hale getirir. Image, yalnızca okunabilir bir şablondur ve değiştirilemez.
- **Container**: Container, yaratılmış bir Image'ın çalışır halidir. Image, bir şablon olarak depolanır ve bu şablondan istediğimiz kadar container oluşturup çalıştırabiliriz. Container, Image'ın yalnızca okunabilir olan şablonuna yazılabilir bir katman ekleyerek çalışan bir kopyasını oluşturur.



CONTAINER VS VM (Virtual Machine)

- **Sanal Makine (VM)**, tam bir işletim sistemi barındırır ve donanımı sanallaştırır. Buna karşılık, **Container** işletim sistemini değil yalnızca uygulamayı sanallaştırır; bir işletim sistemi üzerinde çalışır ve uygulamayı izole eder.
- **Container**'lar, **VM**'lere göre çok daha hızlı başlatılabilir. (VM'in başlatılması, bilgisayarın açılma sürecine benzer ve daha uzun sürerken, container'lar çok daha kısa sürede hazır hale gelir.)
- **VM**'leri paketleyip başka bir sistemde çalıştırmak mümkündür, ancak bu süreç ayar ve yapılandırma gerektirir ve karmaşık olabilir. Ancak, bir uygulamayı **Container Image** olarak paketlediğimizde, bunu üzerinde **Docker Engine** yüklü herhangi bir makineye kolayca taşıyabiliriz ve birebir aynı şekilde çalıştırabiliriz.



Docker CLI (Command Line Interface)

Docker CLI, komut satırı üzerinden Docker işlemlerini gerçekleştirmek için kullanılır. PowerShell veya terminal aracılığıyla çeşitli Docker komutları verilebilir.

Docker Versiyon ve Bilgi Görüntüleme

- **Docker Versiyonu Görüntüleme**: Docker sürümünü görmek için aşağıdaki komutu kullanabilirsiniz:

```
docker version
```

- **Docker Bilgilerini Görüntüleme**: Docker hakkında ayrıntılı bilgi almak için aşağıdaki komut kullanılır:

  ```
  docker info
  ```

  Bu komutla, konteyner durumu hakkında aşağıdaki bilgilere ulaşabilirsiniz:

  - **Containers**: 1
  - **Running**: 0
  - **Paused**: 0
  - **Stopped**: 1

Docker Komutları

Docker CLI'da kullanabileceğiniz ana komutlar:

- **Temel Komutlar**:

  ```
  run, exec, ps, build, pull, push, images, login, logout, search, version, info
  ```

- **Komut Yardımı**: Belirli bir komut hakkında bilgi almak için `-help` parametresi ekleyin. Örneğin:

  ```
  docker container --help
  ```

Docker Container Temelleri

Her konteyner bir Docker imajından türetilir ve bu imaj, çalışması için bir uygulama barındırır. Bu uygulama çalıştığı sürece konteyner ayakta kalır; uygulama sonlandığında konteyner kapanır.

- **Yeni Bir Konteyner Çalıştırma**: Aşağıdaki komut, `ilkcontainer` adında yeni bir konteyner oluşturur ve çalıştırır:

  ```
  docker container run --name ilkcontainer sibacode/adanzyedockerdeneme
  ```

  - Eğer belirtilen imaj yerel sistemde yoksa Docker Hub üzerinden otomatik olarak indirilir ve konteyner oluşturulur.

- **Arka Planda Çalıştırma ve Port Yönlendirme**: Konteyneri arka planda çalıştırmak ve port yönlendirmesi yapmak için aşağıdaki komut kullanılır:

```
docker container run -d -p 80:80 sibacode/adanzyedockerdeneme
```

Bu komut, konteyneri 80 numaralı port üzerinden çalıştırır.

- **Çalışan Bir Konteynere Bağlanma**: Çalışan bir konteynere bağlanmak ve interaktif modda komut çalıştırmak için:

  ```
  docker container exec -it websunucu sh
  ```

  Bu komut, `websunucu` adındaki konteynere bağlanarak `sh` komutunu çalıştırır. `-it` parametresi interaktif modda bağlantı sağlar.

Docker Çok Katmanlı Dosya Sistemi

Docker'da dosya sistemi katmanlı bir yapıya sahiptir:

- **Image**: Sadece okunabilir katmanlardan oluşur (Read-Only Layer, R/O).
- **Container**: Yazılabilir katman içerir (Read-Write Layer, R/W).

Sık Kullanılan Diğer Docker Komutları

- **Kullanılmayan Konteynerleri Temizleme**: Sistemden çalışmayan tüm konteynerleri silmek için:

  ```
  docker container prune -a
  ```

- **İmaj Çekme (Pull)**:
  - Alpine imajını indirmek için:

    ```
    docker image pull alpine
    ```

  - `ozgurozturknet/hello-app` imajını indirmek için:Eğer `hello-app` imajı, `alpine` imajına bağlıysa ve `alpine` imajı önceden indirildiyse, Docker mevcut `alpine` imajını kullanır ve gereksiz indirmeleri önler.

    ```
    docker image pull ozgurozturknet/hello-app
    ```

Docker Volume - Container Dışı Veri Saklama

Veriyi container içinde tuttuğumuzda, container silindiğinde veriler de kaybolur. Ancak, Docker Volume kullanarak veriyi container dışında tutabiliriz. Böylece container silinse bile veri kaybı yaşanmaz.

Docker Volume Nedir?

- Docker Volume'ler, container ve image gibi birer Docker objesidir.
- Bir volume, birden fazla container'a bağlanabilir ve bu sayede veriyi container'lar arasında paylaşabiliriz.

Docker Volume Kullanımı:

1. **Volume Oluşturma:**

   ```
   docker volume create ilkvolume
   ```

   Bu komut ile `ilkvolume` adında bir volume yaratılır.

2. **Volume Detaylarını Görüntüleme:**

   ```
   docker volume inspect ilkvolume
   ```

   Bu komut ile `ilkvolume` volume'ünün detaylarını görebiliriz.

3. **Volume'u Container'a Bağlama:**

   ```
   docker container run -it -v ilkvolume:/uygulama alpine sh
   ```

   Bu komut ile `ilkvolume`, container içerisindeki `/uygulama` klasörüne bağlanır.

4. **Volume'u Salt Okunur Bağlama:**

   ```
   docker container run -it -v ilkvolume:/uygulama:ro alpine sh
   ```

   Bu komut, `ilkvolume` volume'ünü `/uygulama` klasörüne yalnızca okunabilir (read-only) modda bağlar. Bu durumda dosya yazılamaz.

Volume'lerin Özellikleri:

- Container ve volume ayrı objelerdir, yani container silinse bile volume varlığını korur.

Boş veya Dolu Volume Mount Edildiğinde:

1. Eğer bir volume, mount edileceği klasör container içinde mevcut değilse, bu klasör otomatik olarak oluşturulur. O anda volume içinde hangi dosyalar varsa, bu klasörde de o dosyalar görünür.

2. Eğer volume, image içerisinde bulunan mevcut bir klasöre mount edilirse:
    - **Klasör Boşsa:** Volume içinde hangi dosyalar varsa, bu klasörde de o dosyalar görünür.
    - **Klasörde Dosya Varsa ve Volume Boşsa:** Klasördeki dosyalar volume'e kopyalanır.
    - **Klasörde Dosya Olsun veya Olmasın, Volume Doluysa:** Klasörde yalnızca volume içindeki dosyalar görünür.

Terminal Uygulama Açıklamaları

1. **Container Oluşturma ve Volume Kullanımı:**
    - `docker volume create deneme1`: Yeni bir `deneme1` adlı volume oluşturuldu.
    - `docker container run --rm -it -v deneme1:/test ozgurozturknet/adanzyedocker sh`: Yeni bir container oluşturulup `deneme1` volume'ü `/test` klasörüne mount edildi. Bu klasörde volume içeriği görülebilir.
2. **Volume İçeriğini Güncelleme:**
    - `docker container run --rm -it -v deneme1:/xyz alpine sh` komutuyla `deneme1` volume'ü `/xyz` klasörüne mount edildi ve yeni bir dosya (`adanzye.txt`) oluşturuldu.
    - Bu dosya daha sonra `deneme1` volume'üne bağlanan diğer containerlarda da göründü, böylece volume'ün veri paylaşımı sağlanmış oldu.
3. **Volume İçerik Kontrolü:**
    - `docker container run --rm -it -v deneme1:/usr/src/myapp ozgurozturknet/adanzyedocker sh` komutuyla `deneme1` volume'ü tekrar mount edildiğinde, daha önce oluşturulan `adanzye.txt` dosyasının volume içinde mevcut olduğu görüldü.

Bu detaylar, Docker volume'lerin kullanımına yönelik tüm temel özellikleri kapsamaktadır ve verdiğiniz terminal çıktıları bu özelliklerin pratikte nasıl işlediğini göstermektedir.

Bind mounts, Docker'ın bir dosya veya dizini host (sunucu) dosya sisteminden bir container içindeki belirli bir konuma doğrudan bağlamanızı sağlayan bir özelliktir. Bind mount'lar, volume'lere göre daha düşük seviyeli bir depolama çözümü sunar ve daha doğrudan dosya paylaşımı sağlar.

Bind Mount Temel Özellikleri

1. **Host Dosya Sisteminden Erişim:** Bind mount'lar, host (yani Docker'ın çalıştığı bilgisayar) dosya sisteminde mevcut bir dosya veya dizini kullanır. Bu, container ve host arasında veri paylaşmanın hızlı ve etkili bir yoludur.

2. **Direkt Dosya Bağlantısı:** Bir bind mount oluşturduğunuzda, belirtilen host dizini veya dosyası container içinde belirtilen bir konuma bağlanır. Bu sayede, host sistemde yapılan değişiklikler anında container içinde de görünür ve tersi de geçerlidir.
3. **Dinamik Veri Güncelleme:** Host üzerindeki dosyalar, bind mount aracılığıyla container içinde anında güncellenir. Örneğin, geliştirme sırasında kod değişikliklerinin anında container'da görünmesi için bind mount kullanılabilir.
4. **Kapsam ve Güvenlik Farklılıkları:** Bind mount'lar volume'lere göre daha esnektir ancak potansiyel güvenlik riskleri taşıyabilir, çünkü host sistemin herhangi bir dosya veya dizini container'a bağlanabilir. Volume'ler ise yalnızca Docker'ın yönetim alanında depolanan verileri paylaşır ve genellikle belirli kullanım senaryoları için daha güvenlidir.

Bind Mount Kullanımı

Bind mount'u çalıştırmak için `docker run` komutunda `-v` veya `--mount` seçenekleri kullanılır. Örneğin:

```
docker run -it --rm \
  -v /host/dizin/yolu:/container/dizin/yolu \
  image_ismi
```

Bu komutun açılımı şu şekildedir:

- `/host/dizin/yolu`: Host üzerinde bulunan ve container içine bağlanacak dosya veya dizinin yolu.
- `/container/dizin/yolu`: Container içinde host dosyasının veya dizinin görünmesini istediğiniz yol.

Örnek:

```
docker run -it --rm \
  -v /home/mustafa/proje:/app \
  python:3.8-slim \
```

Bu komutta, host makinedeki `/home/mustafa/proje` dizini container içindeki `/app` dizinine bind edilir. Bu şekilde, `proje` dizininde yaptığınız değişiklikler container'da `/app` dizininde görünür.

Bind Mount ile Volume Arasındaki Farklar

Özellik	Bind Mount	Volume
Veri Kaynağı	Host dosya sisteminde belirli bir yol	Docker'ın yönettiği özel alan
Dayanıklılık	Container silinse bile aynı dosya ve dizin kullanılır	Container'dan bağımsızdır, silinmez

Kullanım Alanı	Geliştirme ortamları, anlık dosya paylaşımı	Uygulama verileri, kalıcı veri
Güvenlik	Host dosya sistemine tam erişim verir	Daha güvenli ve izole bir depolama
Yedekleme	Host'un yedeği alınmalıdır	Docker volume yedekleme sistemi

Bind mount, özellikle geliştirme sırasında değişikliklerin container içinde hemen görünmesi gerektiğinde çok faydalıdır. Ancak, güvenlik ve yedekleme ihtiyacı olan senaryolarda volume tercih edilir.

Bind mount kullanarak nginx container'da host sistemdeki dosyaları bağlamak için aşağıdaki adımları izleyebiliriz. Bu örneklerde host üzerindeki bir dizini nginx container içindeki `/usr/share/nginx/html` dizinine bind mount ile bağlayacağız.

1. **Host üzerinde bind mount için kullanılacak bir dizin oluşturma:**

   ```
   mkdir C:\Users\mustafa.senlik\Toren
   ```

2. **Host dizinine bir `index.html` dosyası ekleme veya düzenleme:**

   Bu dosyada kendi içeriğinizi oluşturabilirsiniz:

   ```
   echo "<html><body><h1>Hello from Bind Mount!</h1></body></html>" > C:\Users
   \mustafa.senlik\Toren\index.html
   ```

3. **Nginx container'ını bind mount ile çalıştırma:**

   ```
   docker container run -d -p 80:80 -v C:/Users/mustafa.senlik/Desktop/Toren:/usr/sha
   re/nginx/html nginx
   ```

   Bu komut ile:

   - `-v C:/Users/mustafa.senlik/Desktop/Toren:/usr/share/nginx/html`: Host dizinini container içindeki Nginx'in varsayılan html klasörüne bağlar.
   - `p 80:80`: Container'ın 80 numaralı portunu host'a yönlendirir.

4. **Container içinde `index.html` dosyasını doğrulama:**

   ```
   docker exec -it 44e4e6813e8f sh
   ```

   Container içinde aşağıdaki komutlarla dosya içeriğini görebilirsiniz:

```
cd /usr/share/nginx/html
ls
cat index.html
```

Bu işlemler sonrasında, `http://localhost` adresine gittiğinizde oluşturduğunuz `index.html` içeriğini görebilirsiniz. Bind mount sayesinde host dizinindeki `index.html` dosyasında yaptığınız değişiklikler anında container içinde de görünür olacaktır.

Docker CLI Kullanımı ve Denemeler

Genel Komutlar:

- `docker ps -a`: Tüm konteynerleri listelemek için kullanıldı.
- `docker image ls`: Tüm mevcut imajları listelemek için kullanıldı.
- `docker volume ls`: Tüm mevcut volümleri listelemek için kullanıldı.

Konteyner ve İmaj Silme:

- Tüm konteynerler silindi: `docker rm $(docker ps -aq)`
- Tüm imajlar silindi: `docker rmi $(docker images -q)`
- Tüm kullanılmayan volümler silindi: `docker volume rm $(docker volume ls -q)`

İmaj Çekme (Pull):

- Başarılı indirme komutları:
  - `docker pull centos`
  - `docker pull alpine`
  - `docker pull nginx`
- Hatalı komutlar ve açıklama:
  - `docker pull nginz alpine centos http:alpine` → Birden fazla argüman verilemediği için hata alındı.
  - `docker pull ozgurozturnet/adanzyedocker/hello app` → Hatalı isimlendirme veya giriş gereksinimi nedeniyle hata alındı.

Konteyner Çalıştırma ve Yönetme:

- Çalıştırma:
  - `docker container run -d httpd:alpine` → Arka planda çalıştırıldı.
  - `docker container run -it --name birinci alpine` → Etkileşimli modda çalıştırıldı.

    - `docker container run -it -v alistirma1:/test birinci sh` → Volüm bağlanarak çalıştırıldı.
- Hata yönetimi:
    - `docker exec -it websunucu` komutunda en az iki argüman gerektiği için hata verildi.

Volume Bağlama ve Kullanım:

- `docker volume create alistirma1`: Yeni bir volüm oluşturuldu.
- `docker container run -it --name birinci -v alistirma1:/test alpine sh`: `alistirma1` volümü `/test` dizinine bağlanarak çalıştırıldı.
- Dosya erişim testleri:
    - Volümün bağlandığı dizinde dosya oluşturma ve erişim testi yapıldı (`abc.txt` dosyasına `echo` ile veri eklendi).
    - `docker container run -it --name ucuncu -v alistirma1:/test:ro alpine sh`: Sadece-okunur (read-only) modda bağlanarak dosya oluşturulmaya çalışıldı, ancak "Read-only file system" hatası alındı.

Docker Network Driver

Docker, arka planda birçok farklı servis ve altyapısal karmaşıklığa sahip olmasına rağmen, kullanıcı deneyimi açısından alışık olduğumuz sanal makine mantığından farklı bir sistem sunmaz. Docker konteynerleri, bir sanal makine ağ altyapısında nasıl davranıyorsa benzer şekilde davranır. Özetle, bir Docker konteynerin ağ kullanımı ile bir Linux sunucunun ağ kullanımı arasında büyük bir fark yoktur.

Bu aynılığı sağlamak için tüm iletişim altyapısını Docker network objeleri üzerinden yönetiyoruz. Bu network objeleri, çeşitli driver'lar aracılığıyla yaratılır ve böylece networklere farklı özellikler kazandırılabilir.

Aşağıda en sık kullanılan Docker network driver türleri açıklanmıştır:

1. **Bridge**
    - Bridge driver, Docker konteynerleri arasında iletişim kurmak için varsayılan olarak kullanılır. Her bir konteyner, aynı bridge network içinde otomatik olarak birbirine erişebilir.
    - Özellikle tek bir host üzerinde çalışan konteynerlerin birbirleriyle haberleşmesi gerektiğinde kullanılır. Konteynerler, bu network aracılığıyla kendi özel IP adreslerini alır ve dış dünya ile belirli kurallar çerçevesinde iletişim kurabilir.
2. **Host**
    - Host driver, konteyneri direkt olarak host makinenin ağ kartına bağlar. Bu durumda, konteyner hostun IP adresini ve port numarasını kullanır.

- Özellikle performans gerektiren durumlarda veya hostun ağ yapılandırmasını birebir paylaşmak gerektiğinde tercih edilir. Ancak, güvenlik açısından dikkatli kullanılmalıdır, çünkü konteyner, hostun network katmanına tam erişim kazanır.

3. **Macvlan**
   - Macvlan driver, her konteyner için ayrı bir MAC adresi oluşturur ve konteyneri doğrudan fiziksel network cihazına bağlar. Bu sayede konteyner, host dışında kendi bağımsız ağ kimliğine sahip olur.
   - Özellikle ağda her konteynerin kendi IP ve MAC adresine sahip olması gerektiğinde (örneğin, ağ yönetim araçları ile doğrudan entegrasyon yapıldığında) tercih edilir.
4. **None**
   - None driver, konteynere herhangi bir ağ konfigürasyonu uygulamaz. Böylelikle konteyner, hiçbir network yapılandırmasına sahip olmaz.
   - Genellikle izole edilmiş test ortamlarında veya özel ağ konfigürasyonlarına ihtiyaç duyulan durumlarda kullanılır.
5. **Overlay**
   - Overlay driver, birden fazla Docker hostu arasında konteynerlerin haberleşmesini sağlar. Bu driver, genellikle Docker Swarm gibi orchestrasyon çözümlerinde kullanılır.
   - Overlay ağı, birden çok host üzerinde çalışan konteynerlerin güvenli bir şekilde haberleşmesine olanak tanır ve özellikle çoklu host dağıtımlarında tercih edilir.

Docker Network Kullanımı ve İnceleme Komutları

Docker'da bir network objesini ve konteynerlerin bu networklere nasıl bağlandığını anlamak önemlidir. `docker network ls` komutu kullanıldığında, mevcut ağları görebiliriz. Örneğin:

```
NETWORK ID     NAME      DRIVER    SCOPE
8057cf58b839   bridge    bridge    local
8b7fa1c8f39c   host      host      local
97cddb2e9bb4   none      null      local
```

Bu listede, Docker tarafından varsayılan olarak oluşturulan üç network türü (bridge, host, none) görülür. Eğer bir konteyner oluştururken hangi network'e bağlanacağını belirtmezsek, Docker otomatik olarak konteyneri `bridge` network'üne bağlar.

Inspect Komutu

Konteyner, image ve volume gibi Docker objeleri hakkında detaylı bilgi almak için `inspect` komutu kullanılır:

- Örneğin, bir konteyner hakkında bilgi almak için:

```
docker container inspect <container_adı>
```

- Bir network hakkında detaylı bilgi almak için:

```
docker network inspect <network_adı>
```

Bridge Network'te Konteynerler Arası İletişim

Aynı `bridge` network'e bağlı konteynerler, birbirleriyle iletişim kurabilir. Örneğin, iki konteyneri aynı bridge network'e bağlayarak, kendi özel IP adresleri üzerinden haberleşmelerini sağlayabiliriz.

Host Network Kullanımı

Eğer bir konteynerin `host` network'üne bağlı olarak çalışması isteniyorsa, aşağıdaki komut kullanılabilir:

```
docker container run -it --name deneme1 --net host ozgurozturknet/adanzyedocker sh
```

Bu komutla konteyner, `host` network'üne bağlı olarak çalışır. `docker network inspect host` komutu ile bu network hakkında bilgi alınabilir. Bu durumda konteyner, çalıştığı makinenin IP adresini kullanır ve doğrudan host network yapısına bağlanır.

None Network Kullanımı

Bir konteynerin herhangi bir network'e bağlı olmadan çalışması istendiğinde, `none` network seçeneği kullanılabilir:

```
docker container run -it --name deneme1 --net none ozgurozturknet/adanzyedocker sh
```

Bu komutla oluşturulan konteyner, hiçbir network adresine bağlı olmayacaktır. Böylelikle tamamen izole bir network yapısı sağlar ve yalnızca kendine erişimi vardır.

Publish Port Nedir?

Docker'da bir container içinde çalışan uygulamanın, dış dünyadan erişilebilir hale getirilmesi için **publish port** özelliği kullanılır. Bu özellik, host makinenin bir portunu container içindeki bir porta yönlendirme işlemidir.

Örneğin, bir web sunucusu çalıştırıyorsanız ve bu sunucu container içinde **80** portunda çalışıyorsa, dış dünyadan bu sunucuya erişmek için container içindeki **80** portunu, host

makinenin bir portuna (örneğin **8080**) yönlendirebilirsiniz.

Publish Port Kullanımı

Docker'da bir portu publish etmek için `-p` veya `--publish` bayrağı kullanılır. Bu bayrak, şu şekilde çalışır:

```
docker container run -d -p [HOST_PORT]:[CONTAINER_PORT] image_nam
```

- **HOST_PORT**: Host makinenizde kullanılacak port numarası.
- **CONTAINER_PORT**: Container içinde erişilecek port numarası.
- **image_name**: Çalıştırılacak Docker imajının adı.

Örnek:

Bir Nginx container'ını çalıştırarak 80 portunu host makinenizde 8080 portuna yönlendirmek için:

```
docker container run -d -p 8080:80 nginx
```

Bu komut, aşağıdaki işlemleri gerçekleştirir:

1. **Nginx** çalıştırılır ve container'ın **80** portunda yayın yapar.
2. Host makinenizin **8080** portu, container'ın **80** portuna yönlendirilir.
3. Tarayıcınızda `http://localhost:8080` adresine giderek Nginx'in varsayılan sayfasına ulaşabilirsiniz.

Custom Docker Network ve Container İletişimi

Docker, container'lar arasında izole ve güvenli iletişim sağlamak için **custom network** oluşturma ve container'ları bu ağlarda çalıştırma olanağı sunar. Bu örnekte bir özel ağ oluşturup container'lar arası isim bazlı iletişimi gerçekleştireceğiz.

Network Oluşturma

```
docker network create kopru1
```

- **Amaç**: `kopru1` adında bir özel **bridge** network oluşturur.
- **Bridge Network**:
  - Container'lar arasında özel bir iletişim kanalı sağlar.

- İsim bazlı iletişim (DNS çözümleme) desteklenir.
- Container'lar yalnızca aynı network'e bağlı olan diğer container'larla iletişim kurabilir.

Network Detaylarını Görüntüleme

```
docker network inspect kopru1
```

- **Amaç**: `kopru1` network'ünün özelliklerini ve bağlı container'ları kontrol etmek.
- Görülen Bilgiler:
  - **Subnet**: Ağın IP aralığı.
  - **Container Listesi**: Bu network'e bağlı container'lar ve IP adresleri.

Container Oluşturma ve Ağa Bağlama

1. İlk Container (Web Sunucu)

```
docker container run -dit --name websunucu --net kopru1 ozgurozturknet/adanzyedock
er sh
```

- **Komut Açıklaması**:
  - `-name websunucu`: Container'ın adını `websunucu` olarak belirler.
  - `-net kopru1`: Container'ı `kopru1` ağına bağlar.
  - `ozgurozturknet/adanzyedocker`: Kullanılan Docker imajı.
  - `dit`: Container'ı arka planda (detached), etkileşimli (interactive) ve terminal (tty) modda çalıştırır.

---

2. İkinci Container (Database)

```
docker container run -dit --name database --net kopru1 ozgurozturknet/adanzyedocker
sh
```

- **Komut Açıklaması**:
  - Bu container da `kopru1` ağına bağlanır.
  - Aynı ağda olduğu için, `websunucu` ile iletişim kurabilir.

3.Container'a Bağlanma

```
docker attach websunucu
```

- **Amaç**: `websunucu` container'ına bağlanarak etkileşimli komutlar çalıştırmak.
- Container içinde kullanılan kabuk: **sh**

İsim Bazlı İletişim

4.Database Container'ına Ping Atma

```
ping database
```

- **Sonuç**:
  - `websunucu` container'ından `database` container'ına isim üzerinden erişim sağlanır.
  - Bu, Docker'ın **DNS çözümleme** özelliği ile mümkün olur.

---

1. Docker Logs

- **Amaç:** Container'ın loglarını (uygulama çıktısı ve hatalar) görmek.
- **Komut:**

```
docker logs [OPTIONS] CONTAINER
```

- **Sık Kullanımlar:**
  - Son logları takip et: `docker logs -f container_name`
  - Belirli sayıda log göster: `docker logs --tail 10 container_name`

---

2. Docker Top

- **Amaç:** Container içinde çalışan işlemleri listelemek.
- **Komut:**

```
docker top CONTAINER
```

- **Ne Gösterir?**
  - PID, CPU kullanımı, kullanıcı ve komut bilgileri.

- **Örnek:** `docker top websunucu`

---

3. Docker Stats

- **Amaç:** Container'ların canlı kaynak kullanımını izlemek.
- **Komut:**

```
docker stats [OPTIONS] [CONTAINER...]
```

- **Gösterilen Bilgiler:**
  - CPU ve bellek kullanımı, ağ ve disk I/O, işlem sayısı.
- **Tüm Container'ları İzle:**

```
docker stats
```

- **Belirli Bir Container:**

```
docker stats container_name
```

Özet:

- `logs`: Logları görüntüler.
- `top`: Container içindeki işlemleri listeler.
- `stats`: Canlı kaynak kullanımını gösterir.

Docker Kaynak Kısıtlamaları

1. Bellek Sınırlama

```
docker container run -d --memory=100m ozgurozturknet/adanzyedocker
```

- Maksimum 100 MB RAM kullanır. Sınır aşılırsa container kapanır.

---

2. Swap ile Bellek Sınırlama

```
docker container run -d --memory=100m --memory-swap=200m ozgurozturknet/adanz
yedocker
```

- 100 MB RAM + 100 MB Swap = Toplam 200 MB bellek sınırı.

3. CPU Kullanımı Sınırlama

```
docker container run -d --cpus="1.5" ozgurozturknet/adanzyedocker
```

- Maksimum 1.5 CPU çekirdeği kullanır.

4. Belirli CPU Çekirdeklerini Kullanma

```
docker container run -d --cpus="1.5" --cpuset-cpus="0,3" ozgurozturknet/adanzyedocker
```

- 1.5 CPU ile çalışır ve sadece 0 ve 3 numaralı çekirdekleri kullanır.

Özet:

- `-memory`: RAM sınırı.
- `-memory-swap`: RAM + Swap toplam sınırı.
- `-cpus`: CPU çekirdeği sınırı.
- `-cpuset-cpus`: Belirli çekirdekleri kullanma.

Docker Image İsimlendirme Yapısı

Docker image'larının isimlendirme ve tag yapısı şu şekildedir:

```
docker.io/<kullanıcı-adı>/<repository>:<tag>
```

- **docker.io**: (Opsiyonel) Docker'ın varsayılan kayıt defteri (registry). Belirtilmezse `docker.io` varsayılan olarak kullanılır.
- **<kullanıcı-adı>**: Docker Hub üzerindeki kullanıcı adı veya organizasyon adı.
- **<repository>**: İlgili uygulamanın veya projenin adı.
- **<tag>**: İmage'in versiyon bilgisini tanımlayan bir etikettir. Belirtilmezse varsayılan olarak `latest` kullanılır.

Örnekler

1. **Varsayılan registry ile:**

```
docker.io/mustafasenlik/adanzyedocker:latest
```

   - **docker.io**: Varsayılan registry.
   - **mustafasenlik**: Kullanıcı adı.
   - **adanzyedocker**: Repository.
   - **latest**: Varsayılan tag.

2. **Tag belirtilerek:**

```
mustafasenlik/uygulama:koyu
```

   - **mustafasenlik**: Kullanıcı adı.
   - **uygulama**: Repository.
   - **koyu**: Özel tag.

3. **Tag belirtilmezse:**

```
mustafasenlik/uygulama
```

   - **latest** tag'i varsayılan olarak kullanılır.

Docker Image Çekme Komutları

Belirli bir repository ve tag ile image çekmek için şu komutlar kullanılır:

1. **Varsayılan tag ile:**

```
docker image pull ozgurozturknet/adanzyedocker
```

   - Bu komut, `latest` tag'li image'i çeker.

2. **Belirli bir tag ile:**

```
docker image pull ozgurozturknet/adanzyedocker:v1
docker image pull ozgurozturknet/adanzyedocker:v2
```

   - Bu komutlar, sırasıyla `v1` ve `v2` tag'li image'leri çeker.

Tüm Versiyonları Görmek

Çekilen image'lerin tüm versiyonlarını görmek için şu komut kullanılır:

```
docker image ls
```

Örnek Çıktı:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ozgurozturknet/adanzyedocker	latest	abc123def	2 days ago	150MB
ozgurozturknet/adanzyedocker	v1	ghi456jkl	1 week ago	145MB
ozgurozturknet/adanzyedocker	v2	mno789pqr	3 weeks ago	160MB

Dockerfile Nedir?

Dockerfile, Docker imageleri oluşturmak için kullanılan ve kendine özgü kuralları olan bir betik dosyasıdır. Dockerfile'da tanımlanan talimatlar doğrultusunda bir Docker image oluşturulur.

Dockerfile Talimatları

1. FROM

- Dockerfile'da **olmazsa olmaz** bir talimattır.
- Hangi baz image'den başlanacağını belirtir.
- Söz Dizimi:

  ```
  FROM <image>:<tag>
  ```

- Örnek:

  ```
  FROM ubuntu:18.04
  ```

2. RUN

- Shell üzerinde komut çalıştırmak için kullanılır.
- Genellikle yazılımları yüklemek veya ayarları yapmak için kullanılır.
- Söz Dizimi:

  ```
  RUN <komut>
  ```

- Örnek:

```
RUN apt-get update && apt-get install -y python3
```

3. COPY

- Host makineden, imaj içine dosya veya klasör kopyalamak için kullanılır.
- Söz Dizimi:

```
COPY <kaynak> <hedef>
```

- Örnek:

```
COPY ./app /app
```

4. EXPOSE

- Container'ın hangi port üzerinden iletişim kuracağını belirtir.
- Sadece bir bilgi amaçlıdır, container'ı çalıştırırken bu portu yayınlamak için `p` veya `P` kullanılmalıdır.
- Söz Dizimi:

```
EXPOSE <port>/<protokol>
```

- Örnek:

```
EXPOSE 80/tcp
```

5. CMD

- Container çalıştırıldığında varsayılan olarak çalıştırılacak komutu belirtir.
- Söz Dizimi:

```
CMD ["<komut>", "<parametre1>", "<parametre2>"]
```

- Örnek:

```
CMD ["python3", "/app/app.py"]
```

6. HEALTHCHECK (Opsiyonel)

- Container'ın sağlığını kontrol etmek için kullanılır.
- Söz Dizimi:

```
HEALTHCHECK [OPTIONS] CMD <komut>
```

- Örnek:

```
HEALTHCHECK CMD curl -f http://localhost:8080 || exit 1
```

Örnek Dockerfile

Aşağıda basit bir Python uygulaması için örnek bir Dockerfile verilmiştir:

```
# Baz imaj olarak Ubuntu kullanılıyor
FROM ubuntu:18.04

# Gerekli dosyalar kopyalanıyor
COPY ./app /app

# Gerekli bağımlılıklar yükleniyor
RUN apt-get update && apt-get install -y python3 python3-pip && pip3 install -r /app/req
uirements.txt

# Uygulamanın çalışacağı port belirtiliyor
EXPOSE 8080

# Container başlatıldığında çalışacak komut
CMD ["python3", "/app/app.py"]
```

Dockerfile’dan Docker Hub’a Image Push Etme Süreci (Adım Adım Notlar)

Bu notlar, Dockerfile oluşturma aşamasından başlayarak Docker image’inizi Docker Hub’a yükleyene kadar geçen tüm adımları içerir. Her adımı detaylı bir şekilde açıklıyoruz.

---

1. Dockerfile Oluşturma

Docker container'ınızı oluşturmak için bir `Dockerfile` oluşturun. Bu dosya, container'ın nasıl inşa edileceğini tanımlar.

- **Dosya Adı:** `Dockerfile`
- **İçerik:**

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8080

CMD ["python","app.py"]
```

---

2. Uygulama Kodu (app.py)

Container içinde çalışacak bir Python dosyası oluşturun.

- **Dosya Adı:** `app.py`
- **İçerik:**

```
print("Merhaba Dünya!")
```

---

3. Docker Image Oluşturma

Dockerfile ve `app.py` dosyasının bulunduğu dizinde terminali açarak image oluşturun.

- **Komut:**

```
docker build -t python-merhaba-dunya:latest .
```

- **Açıklama:**
  - `t python-merhaba-dunya:latest`: Image'e isim (`python-merhaba-dunya`) ve etiket (`latest`) verir.
  - `.`: Dockerfile'ın bulunduğu dizini işaret eder.

---

4. Docker Hub'a Giriş Yapma

Docker Hub'da oturum açmak için aşağıdaki komutu çalıştırın:

- **Komut:**

```
docker login
```

- **Gerekli Bilgiler:**
  - Kullanıcı adı: `mustafasenlik`
  - Şifre: Docker Hub hesabınızın şifresi.

---

5. Docker Image'i Etiketleme

Image'inizi Docker Hub'a uygun bir isimle etiketlemeniz gerekir.

- **Komut:**

```
docker tag python-merhaba-dunya:latest mustafasenlik/adanzyedockerdeneme:latest
```

- **Açıklama:**
  - `python-merhaba-dunya:latest`: Yerel image'in adı.
  - `mustafasenlik/adanzyedockerdeneme:latest`: Docker Hub deposu ve etiketi.

---

6. Docker Image'i Push Etme

Etiketlenen image'i Docker Hub'a yükleyin.

- **Komut:**

```
docker push mustafasenlik/adanzyedockerdeneme:latest
```

- **Açıklama:**
  - `mustafasenlik/adanzyedockerdeneme:latest`: Docker Hub'da kullanacağınız image'in tam adı.

---

7. Yüklemeyi Kontrol Etme

Docker Hub'da `mustafasenlik/adanzyedockerdeneme` deposuna giderek image'in yüklendiğinden emin olun.

- Web adresi: https://hub.docker.com/repository/docker/mustafasenlik/adanzyedockerdeneme

---

8. Başka Bir Sistemden Çekme

Başka bir sistemde bu image'i kullanmak için `docker pull` komutunu çalıştırın.

- **Komut:**

```
docker pull mustafasenlik/adanzyedockerdeneme:latest
```

Dockerfile'da `COPY` ve `ADD` Arasındaki Farklar

1. **Temel İşlev:**
   - `COPY`: Dosyaları yerel sistemden konteyner imajına kopyalar.
   - `ADD`: `COPY` ile benzer şekilde çalışır ancak ek özelliklere sahiptir.
2. **Ek Özellikler:**
   - `ADD` ile bir **URL'den dosya** indirilebilir:
     - Örnek: `ADD https://example.com/file.tar.gz /app/`
   - `ADD` sıkıştırılmış bir dosyayı açabilir:
     - Örnek: `ADD file.tar.gz /app/` (Dosya otomatik açılır).
3. **Öneri:**
   - **Yalnızca kopyalama gerekiyorsa `COPY` kullanın**: Daha net ve belirgin.
   - **Ek özelliklere ihtiyaç varsa `ADD` kullanın**: Örneğin, bir URL'den indirme veya sıkıştırılmış dosya açma işlemi için.
4. **Performans ve Güvenlik:**
   - `COPY` daha öngörülebilirdir ve genellikle tercih edilir.
   - `ADD`'nin otomatik dosya açma özelliği beklenmeyen sonuçlara yol açabilir.

✔ **Özet:**

`COPY`: Sadece yerel dosyaları kopyalar.

`ADD`: URL'den indirme ve sıkıştırılmış dosyaları açma gibi ek özellikler sağlar.

CMD ve ENTRYPOINT Arasındaki Farklar

1. **CMD:**
   - **Varsayılan komut** tanımlar.

- Kullanıcı tarafından **tamamen geçersiz kılınabilir.**
- Örnek:

```
CMD ["python", "app.py"]
```

2. **ENTRYPOINT:**
   - **Sabit başlangıç komutu** tanımlar.
   - Kullanıcı yalnızca **argüman ekleyebilir, komut değiştirilemez.**
   - Örnek:

```
ENTRYPOINT ["python", "app.py"]
```

3. **Birlikte Kullanım:**
   - `ENTRYPOINT` sabit komut; `CMD` ise varsayılan argüman sağlar.
   - Örnek:

```
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Multi-Stage Build Nedir?

Multi-Stage Build, Dockerfile'da birden fazla aşama (stage) tanımlayarak daha küçük ve optimize edilmiş Docker imajları oluşturmayı sağlayan bir tekniktir. Bu yöntem, özellikle derleme aşamalarını ayırarak yalnızca gerekli dosyaları final imajına dahil etmek için kullanılır.

Multi-Stage Build: Aşama 1 ve Aşama 2 Tablosu

Satır	Aşama 1 (builder)	Aşama 2 (Final)
Base Image	`FROM python:3.10-slim as builder`	`FROM python:3.10-slim`
Çalışma Dizini	`WORKDIR /app`	`WORKDIR /app`
Gerekli Araçların Kurulumu	`RUN apt-get update && apt-get install -y gcc`	(Bu aşamada araç kurulumu yapılmaz.)
Bağımlılıkların Yüklenmesi	`COPY requirements.txt ./` ve `RUN pip install -r requirements.txt`	`COPY --from=builder /usr/local/lib/python3.10/site-packages /usr/local/lib/python3.10/site-packages`

Kodların Kopyalanması	`COPY . .`	`COPY --from=builder /app /app`
Port Açma	(Bu aşamada port açılmaz.)	`EXPOSE 5000`
Çalıştırma Komutu	(Bu aşamada çalıştırma yapılmaz.)	`CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]`

`BUILD ARG` Nedir?

`BUILD ARG`, Dockerfile'da **build sırasında değişken tanımlamak** için kullanılır. Sadece build sırasında geçerlidir ve `--build-arg` ile değer atanır.

Örnek:

```
ARG VERSION=3.10
FROM python:$VERSION
```

Build Komutu:

```
docker build --build-arg VERSION=3.9 -t myimage .
```

Sonuç: `VERSION=3.9` ile Python 3.9 kullanılarak imaj oluşturulur.

`docker save` ve `docker load` Nedir?

Docker'da `docker save` ve `docker load` komutları, imajları dosya olarak kaydetmek ve geri yüklemek için kullanılır. Bu komutlar özellikle imajları taşımak, yedeklemek veya farklı bir makinede kullanmak için kullanışlıdır.

`docker save`

- **İşlev:** Bir veya birden fazla Docker imajını bir arşiv dosyasına kaydeder.
- **Amaç:** İmajları yedeklemek veya başka bir ortama taşımak.
- **Kullanım:**

  ```
  docker save -o <dosya_adı.tar> <image_name:tag>
  ```

  - Örnek:Bu komut, `myimage:latest` imajını `myimage.tar` dosyasına kaydeder.

    ```
    docker save -o myimage.tar myimage:latest
    ```

`docker load`

- **İşlev:** `docker save` ile kaydedilmiş bir imajı arşivden yükler.
- **Amaç:** Kaydedilen bir imajı başka bir ortamda veya aynı sistemde geri yüklemek.
- **Kullanım:**

```
docker load -i <dosya_adı.tar>
```

  - Örnek:Bu komut, `myimage.tar` dosyasındaki Docker imajını sisteme yükler.

```
docker load -i myimage.tar
```

Farklılıkları:

Komut	Amaç
`docker save`	Docker imajını bir dosyaya kaydeder.
`docker load`	Kaydedilmiş dosyadan Docker imajını yükler.

Docker Registry Nedir?

Docker Registry, Docker imajlarının saklandığı ve paylaşıldığı bir **depolar sistemidir**. Geliştiriciler, kendi oluşturdukları Docker imajlarını bir registry'e yükleyebilir (push) ve başka bir sistemde indirip (pull) kullanabilir.

Öne Çıkan Özellikler:

1. **İmaj Depolama ve Yönetimi:**
   - Docker Registry, imajların versiyonlarını ve etiketlerini (tag) yönetir.
   - Örneğin: `myimage:1.0`, `myimage:latest`.
2. **Kendi Registry'inizi Kurabilirsiniz:**
   - Docker Hub dışında kendi özel (private) registry'nizi kurabilirsiniz.
3. **En Popüler Örnek:**
   - **Docker Hub:** Varsayılan halka açık Docker Registry.
   - Diğer popüler örnekler:
     - **Amazon ECR**

- **Google Container Registry (GCR)**
- **Azure Container Registry (ACR)**

Bilgisayara Docker Registry Kurulumu (Kısa)

1. **Registry İmajını İndirin:**

```
docker pull registry:2
```

2. **Registry'yi Başlatın:**

```
docker run -d -p 5000:5000 --name my-registry registry:2
```

3. **İmaj Gönderme (Push):**
   - Etiketleyin:

```
docker tag myimage localhost:5000/myimage:1.0
```

   - Gönderin:

```
docker push localhost:5000/myimage:1.0
```

4. **İmaj Çekme (Pull):**

```
docker pull localhost:5000/myimage:1.0
```

5. **Registry Durumu Görüntüleme:**

```
curl http://localhost:5000/v2/_catalog
```

Docker Compose Nedir?

Docker Compose, birden fazla container'ı bir arada yönetmek ve uygulamaların çalışma ortamlarını tanımlamak için kullanılan bir araçtır. Özellikle mikroservis mimarilerinde birden fazla servisin birlikte çalışması gerektiğinde çok işe yarar. `docker-compose.yml` dosyası kullanılarak servisler, ağlar ve hacimler kolayca tanımlanır.

- **Avantajları:**
  - Birden fazla container'ı tek komutla başlatma (`docker-compose up`).
  - Servislerin yapılandırmalarını kodla tanımlama (örn: portlar, çevre değişkenleri, hacimler).
  - Local ve production ortamlarında aynı yapılandırmayı kullanarak tutarlılığı sağlama.

Komutlar Arası İş Akışı

1. **Proje Başlatma:**

```
docker-compose up -d
```

2. **Servis Durumlarını Kontrol Etme:**

```
docker-compose ps
```

3. **Servis Loglarını Görüntüleme:**

```
docker-compose logs -f
```

4. **Servisleri Güncelleme veya Yeniden Başlatma:**

```
docker-compose down
docker-compose up -d
```

5. **Sorun Giderme İçin Container İçinde Komut Çalıştırma:**

```
docker-compose exec [service_name] bash
```

Docker Compose CLI, mikroservis ortamlarında ya da birden fazla container ile çalışan projelerde container'ların tüm yaşam döngüsünü yönetmeyi kolaylaştırır.

Docker Compose Güncelleme Süreci

Eğer `docker compose down` komutuyla container'ları durdurup kodda bir güncelleme yaparsanız, ardından `docker compose up` komutunu çalıştırmanız yeterli olmayabilir. Bunun nedeni, Docker

Compose'un mevcut imajları kullanmaya devam etmesidir. Koddaki değişikliklerin geçerli olmasını sağlamak için şu adımları izlemelisiniz:

1. **Güncellenmiş İmajı Oluşturma**

   ```
   docker compose build
   ```

2. **Container'ları Yeniden Başlatma**

   ```
   docker compose up
   ```

Bu işlem, kod değişikliklerinin Docker container'larında uygulanmasını sağlar.

Container Orchestration (Konteyner Orkestrasyonu) Nedir?

Container Orchestration, container'ların dağıtımını, yönetimini, ölçeklendirilmesini ve ağ oluşturulmasını otomatikleştirme sürecidir. Büyük ölçekli uygulamalarda, yüzlerce veya binlerce container'ı verimli bir şekilde yönetmek için kullanılır.

Container Orchestration Araçları

1. Kubernetes

- Google tarafından geliştirilmiş ve en yaygın kullanılan konteyner orkestrasyon aracıdır.
- **Başlıca Özellikler:**
  - Otomatik ölçeklendirme (auto-scaling).
  - Kendini iyileştirme (self-healing) özellikleri.
  - Gelişmiş ağ çözümleri (ingress, service).
  - Rolling updates desteği (kesintisiz güncellemeler).

2. Docker Swarm

- Docker'ın yerleşik orkestrasyon çözümüdür.
- **Başlıca Özellikler:**
  - Hafif ve hızlı bir kurulum sağlar.
  - Docker Compose ile kolay entegrasyon.
  - Daha küçük ve orta ölçekli projeler için uygundur.

Özellik	Docker Compose	Docker Swarm

Kapsam	Tek makine	Çoklu makine (cluster)
Kullanım Amacı	Geliştirme ve test	Üretim ve ölçekleme
Ağ	Lokal ağ	Overlay network
Yedekleme	Elle başlatma	Otomatik yedekleme
Ölçeklendirme	Sadece lokal	Cluster genelinde
Yönetim	Basit	Karmaşık

Docker Swarm'da Manager Node ve Worker Node

Manager Node

- **Görevi:** Kümenin yönetiminden ve görevlerin dağıtımından sorumlu.
- **Özellikleri:**
  - Lider node, görevleri planlar ve diğer node'ları koordine eder.
  - Swarm komutları yalnızca manager node’da çalıştırılır.
  - Hata toleransı için birden fazla manager node olabilir.

Worker Node

- **Görevi:** Manager node tarafından atanan görevleri (tasks) yerine getirir.
- **Özellikleri:**
  - Sadece görev çalıştırır, küme yönetimi yapmaz.
  - Manager node’dan talimat alarak container’ları çalıştırır.

Manager Node ve Worker Node Arasındaki Farklar

Özellik	Manager Node	Worker Node
Görev	Küme yönetimi ve görev planlama	Verilen görevleri yerine getirme
Liderlik	Lider node seçimi ve küme denetimi	Yok
Komut Çalıştırma	Swarm ile ilgili komutlar burada çalıştırılır	Swarm komutları çalıştırılamaz
Konteyner Çalıştırma	Evet (opsiyonel)	Evet

Docker Swarm Modu ve Node Yönetimi

Swarm Modunu Aktif Etmek

Docker Engine üzerinde **Swarm modunu aktif etmek** için aşağıdaki komutu kullanabilirsiniz:

```
docker swarm init --advertise-addr 192.168.0.13
```

- Bu komut, Swarm modunu devreye alır ve **ilk manager node** oluşturulur.

Cluster'a Node Ekleme

1. **Yeni bir Manager Node eklemek için:**

   ```
   docker swarm join-token manager
   ```

   - Bu komut, cluster'a bir **manager node** eklemek için gerekli olan token'ı sağlar.
   - Token'ı farklı bir node üzerinde çalıştırarak o node'u **manager** olarak ekleyebilirsiniz.

2. **Yeni bir Worker Node eklemek için:**

   ```
   docker swarm join-token worker
   ```

   - Bu komut, cluster'a bir **worker node** eklemek için gerekli olan token'ı sağlar.
   - Token'ı farklı bir node üzerinde çalıştırarak o node'u **worker** olarak ekleyebilirsiniz.

Node'ları Listeleme

Cluster içerisindeki tüm node'ları listelemek için:

```
docker node ls
```

- Bu komut, **manager** ve **worker** node'ların bir listesini gösterir.

Varsayılan Davranış

- **Manager node'lar**, varsayılan olarak aynı zamanda birer **worker node** olarak görev yapar. (Yani, yönetim işlevlerinin yanı sıra container çalıştırabilirler.)

Docker Service

Docker Swarm Cluster'da oluşturulabilecek en temel obje **Service**'dir.

Örnek Komut:

```
docker service create \
  --name test \
```

```
--network abc \
--publish 8080:80 \
--replicas=10 \
--update-delay=10s \
--update-parallelism=2 \
mustafasenlik/dockerdeneme:v4
```

Desired State (Deklanmış Durum)

Bu komutla aşağıdaki durumlar belirtilmiştir:

1. **Konteyner İmajı:** `mustafasenlik/dockerdeneme:v4` imajından yaratılacak.
2. **Replika Sayısı:** Sistem genelinde **10 container** çalışacak.
3. **Ağ:** `abc` isimli bir network'e bağlı olacak.
4. **Port:** Port yönlendirmesi yapılacak: **8080:80**
5. **Güncelleme Stratejisi:**
   - Aynı anda maksimum **2 task** güncellenecek.
   - Her güncelleme arasında **10 saniye** bekleme süresi olacak.

Current State (Mevcut Durum)

- Yukarıdaki özelliklerle **10 container** çalışmaktadır.
- Eğer mevcut durum ile deklare edilen durum arasında farklılık olursa (örneğin, yalnızca 9 container çalışıyorsa), Swarm devreye girer ve şu işlemleri yapar:
  - **Mevcut Durum** ile **İstenen Durumu** eşitlemeye çalışır.
  - Müsait olan bir node'da bir container daha çalıştırır ve replika sayısını belirtilen duruma getirir.

Docker Service Mode (Servis Modu)

Docker servislerinin çalışma modunu belirtmek için iki farklı yöntem bulunmaktadır:

1. **Replicated Mode:**
   - Belirli bir replika sayısı tanımlanır (örneğin, 10 replika).
   - Swarm, uygun olan nodelar üzerinde belirtilen sayıda replika çalıştırır.
2. **Global Mode:**
   - Replika sayısı belirtilmez.

- Swarm, kümedeki her node üzerinde **1 replika** çalıştırır.

Docker Service Komutları ve Container Orchestration Notları

Servis Yönetimi

1. **Servis Altındaki Taskları Görüntüleme**

   ```
   docker service ps test
   ```

   Bu komut, `test` servisinin altında çalışan taskları gösterir.

2. **Servis Detaylarını Görüntüleme**

   ```
   docker service inspect test
   ```

   Bu komut, `test` servisinin detaylarını gösterir.

3. **Servis Loglarını Görüntüleme**

   ```
   docker service logs test
   ```

   `test` servisi altında 20 container çalışıyorsa, bu komutla tüm containerların loglarını topluca görebilirsiniz.

Container Orchestration ile Ölçeklendirme

- Bir websitesi yüksek trafik aldığında, mevcut container sayısını artırarak sistemin performansını iyileştirmek mümkündür.
- Bu işlem için **Container Orchestration** araçları kullanılır.
- **Docker Swarm** ile ölçeklendirme şu şekilde yapılabilir:

1. **Container Sayısını Artırma**

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

   Bu komut, `test` servisine 2 adet daha container ekleyerek toplamda 3 container çalıştırır.

2. **Container Sayısını Azaltma**

```
docker service scale test=2
```

Bu komut, çalışan container sayısını 2'ye düşürür.

3. **Servisi Silme**

```
docker service rm test
```

Bu komut, `test` servisini siler. Ancak **onay istemeden doğrudan silme işlemi yaptığı** için dikkatli kullanılmalıdır.

Global Modda Servis Oluşturma

- Global mod, her bir node üzerinde **tek bir container çalıştırmak** için kullanılır.
- Örnek komut:Bu komut, tüm node'larda birer adet `nginx` container çalıştırır. Eğer herhangi bir container kapanırsa, sistem hemen yenisini oluşturur. Çünkü **global modda**, her node'da bir container çalışması zorunludur.

```
docker service create --name glb --mode=global nginx
```

Docker Swarm ve Overlay Network Notları

Overlay Network Nedir?

- Swarm cluster oluşturulduğunda, otomatik olarak `ingress` adında bir overlay network oluşturulur.
- Eğer bir network belirtmezseniz, oluşturduğunuz servisler bu overlay networke bağlanır.

Overlay Network Özellikleri

1. **Şifreleme**
   - Overlay networkler, yönetim katmanında şifreli bir haberleşme altyapısına sahiptir.
   - Ancak, bu **varsayılan olarak aktif değildir**. Şifrelemeyi etkinleştirmek için:**Not:** Şifreleme aktifleştirildiğinde network trafiği biraz yavaşlayabilir.

     ```
     docker network create -d overlay --opt encrypted my-overlay
     ```

2. **Servisler Arası İletişim**

- Aynı overlay network'e bağlı olan servislerin container'ları **herhangi bir port kısıtlaması olmadan** haberleşebilir.
- Bu container'lar, aynı fiziksel ağda çalışıyormuş gibi davranır.

3. **DNS Çözümleme**
   - Swarm altında oluşturulan servisler, aynı overlay network üzerindeyse **servis isimleriyle birbirlerine ulaşabilirler.**

Overlay Network Kullanımı

1. **Network Oluşturma**

```
docker network create -d overlay over-net
```

2. **Network Detaylarını Görüntüleme**

```
docker network inspect over-net
```

3. **Servis Oluşturma ve Overlay Network Bağlantısı**

   Örneğin, 3 adet replika içeren bir web servisi oluşturalım:

```
docker service create --name web --network over-net -p 8080:80 --replicas=3 ozgu
rozturknet/web
```

   - Bu servis, `over-net` networküne bağlanır.
   - `p 8080:80` parametresi ile 8080 portundan erişim sağlanır.

4. **Servis Detaylarını Görüntüleme**

```
docker service ps web
```

   - Örneğin, 1 worker ve 1 manager node üzerinde container bulunmayabilir. Ancak, o node'un **8080 portuna giderseniz**, sistemin çalıştığını görürsünüz.
   - Bu özelliğe **Swarm Routing Mesh** denir.
     - Herhangi bir node'a gelen istek, container'ın bulunduğu bir node'a yönlendirilir.

5. **DB Servisi Oluşturma**

```
docker service create --name db --network over-net ozgurozturknet/fakedb
```

6. **Container İçinden Servisler Arası Haberleşme**
    - DB container'ına bağlanıp `web` servisine ping atabilirsiniz:

      ```
      docker exec -it "db_container_idsi" sh
      ping web
      ```

    - `ping web` komutu, `web` servisine ait bir VIP (Virtual IP) döndürür. Bu VIP, **DNS çözümlemesi** ve **load balancing** işlemlerini destekler.

Docker Service Update ile Servis Güncelleme

Bir Docker servisini yeni bir imaj ile güncellemek için aşağıdaki komutu kullanabiliriz:

```
docker service update --detach --update-delay 5s --update-parallelism 2 --image ozgurozturknet/web:v2 websrv
```

Parametrelerin Anlamı:

1. `-detach`: Güncelleme işlemini arka planda gerçekleştirir. Terminali bekletmez.
2. `-update-delay 5s`: Her güncelleme görevinden sonra **5 saniye** bekle.
3. `-update-parallelism 2`: Aynı anda **en fazla 2 görev** (task) güncelle.
4. `-image ozgurozturknet/web:v2`: Güncellemede kullanılacak yeni imaj. Burada `v2` versiyonu kullanılıyor.
5. `websrv`: Güncellenecek olan servis adı.

Açıklama:

- Bu komut, `websrv` servisini `ozgurozturknet/web:v2` imajı ile günceller.
- Aynı anda en fazla **2 görev** güncellenir ve her bir güncelleme tamamlandıktan sonra **5 saniye** beklenir.
- Güncelleme arka planda çalışır.

Docker Service Rollback ile Güncellemeyi Geri Alma

Eğer güncelleme sırasında bir sorun çıkarsa veya eski versiyona dönmek isterseniz aşağıdaki komutu kullanabilirsiniz:

```
docker service rollback --detach websrv
```

Parametrelerin Anlamı:

1. `-detach`: Rollback işlemini arka planda gerçekleştirir.
2. `websrv`: Rollback yapılacak servis adı.

Açıklama:

- Bu komut, `websrv` servisini **bir önceki versiyona** geri alır.
- Rollback işlemi güncelleme sırasında sorun yaşandığında servisin önceki durumuna kolayca dönmeyi sağlar.

Docker Secrets, Docker'da gizli verileri (örneğin, şifreler, API anahtarları, SSL sertifikaları) güvenli bir şekilde yönetmek ve hizmetler arasında paylaşmak için kullanılan bir mekanizmadır. Özellikle **Docker Swarm** modunda çalışırken kullanılır ve bu gizli bilgiler, yalnızca yetkilendirilmiş konteynerler tarafından erişilebilir hale getirilir.

Docker Secrets'in Özellikleri:

1. **Güvenli Depolama**: Gizli bilgiler Docker Engine üzerinde şifrelenmiş şekilde saklanır.
2. **Sınırlı Erişim**: Sadece izin verilen hizmetler bu bilgilere erişebilir. Diğer konteynerler veya düğümler erişemez.
3. **Kolay Kullanım**: Bir secret oluşturmak ve bunu bir servise bağlamak oldukça basittir.
4. **Bellekte Yönetim**: Gizli bilgiler, çalışan konteynerin belleğinde plaintext olarak tutulur, dosya sisteminde kaydedilmez.

Docker Secrets Kullanımı

1. Secret Oluşturma

`docker secret create` komutu ile bir secret oluşturabilirsiniz:

```
echo "my_password" | docker secret create my_secret -
```

Bu komutla `my_password` değerinde bir secret oluşturulur ve `my_secret` adı verilir.

2. Secret'i Servise Bağlama

Bir secret'i bir Docker hizmetine bağlamak için aşağıdaki gibi bir komut kullanabilirsiniz:

```
docker service create --name my_service --secret my_secret my_image
```

Bu komut, `my_secret` adlı secret'i `my_service` adındaki hizmete bağlar.

3. Secret'i Konteynerde Kullanma

Bir hizmet, bağlı olan secret'lere `/run/secrets/` dizini üzerinden erişebilir. Örneğin:

```
cat /run/secrets/my_secret
```

Bu komut, secret'in değerini konteyner içinde okumak için kullanılır.

4. Secret Listeleme

Oluşturulan secret'leri listelemek için:

```
docker secret ls
```

5. Secret Silme

Bir secret'i silmek için:

```
docker secret rm my_secret
```

Docker Secrets'in Avantajları

- Hassas bilgilerin güvenli bir şekilde taşınmasını sağlar.
- Gizli bilgilerin konteyner dışında şifreli saklanması güvenliği artırır.
- CI/CD süreçlerinde şifre yönetimini kolaylaştırır.

Docker Stack Nedir?

Docker Stack, birden fazla servisi tanımlayıp yönetmek için kullanılan bir Docker Swarm özelliğidir. **Docker Compose** dosyalarını Swarm ortamında çalıştırarak, uygulamanızın servislerini bir arada tanımlamanıza ve dağıtmanıza olanak tanır.

Docker Stack'in Avantajları:

1. **Çok Servisli Uygulamaların Yönetimi:**

Birden fazla servisten oluşan uygulamalarınızı tek bir yapılandırma dosyasıyla tanımlayabilirsiniz (örneğin: web, database, cache gibi).

2. **Kolay Dağıtım:**

   Tüm servisleri tek bir komutla dağıtabilir ve Swarm cluster'ınızda ölçeklendirebilirsiniz.

3. **Compose Dosyasıyla Entegrasyon:**

   Docker Stack, **Docker Compose** dosyalarını kullanır. Yani, `docker-compose.yml` dosyanız hazırsa, bunu Swarm ortamında kolayca kullanabilirsiniz.

4. **Servislerin İzlenmesi ve Yönetimi:**

   Her bir servis için loglama, ağ bağlantıları ve ölçeklendirme gibi özellikleri kolayca yönetebilirsiniz.

Docker Stack Komutları ve Kullanımı

1. Docker Compose Dosyası Hazırlama

- Örnek `docker-compose.yml` dosyası:

```
version: "3.9"
services:
  web:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
    networks:
      - app-net

  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
    networks:
      - app-net

networks:
```

```
app-net:
  driver: overlay
```

- `deploy` kısmı, Swarm için özel ayarları içerir (örneğin: `replicas` ile container sayısı).
- `networks` kısmında overlay network kullanarak servislerin birbirleriyle haberleşmesini sağlıyoruz.

---

2. Docker Stack ile Servislerin Dağıtımı

- Aşağıdaki komutla Docker Stack'i dağıtabilirsiniz:

```
docker stack deploy -c docker-compose.yml my-stack
```

  - `c`: Compose dosyasını belirtir.
  - `my-stack`: Stack'in adı (örneğin: `my-stack`).

---

3. Docker Stack Komutları

- **Tüm Stack'leri Listeleme**

```
docker stack ls
```

- **Bir Stack'teki Servisleri Görüntüleme**

```
docker stack services my-stack
```

- **Bir Stack'in Detaylarını Görüntüleme**

```
docker stack ps my-stack
```

- **Bir Stack'i Kaldırma**

```
docker stack rm my-stack
```

Docker Stack ile Docker Compose Arasındaki Farklar

Özellik	Docker Compose	Docker Stack
Kapsam	Tek bir makine üzerinde çalışır.	Swarm cluster'da çalışır.

Orchestration	Orchestration yoktur.	Swarm özellikleri kullanılır.
Network	Bridge veya host network kullanır.	Overlay network kullanır.
Dağıtım	`docker-compose up` ile çalıştırılır.	`docker stack deploy` ile çalıştırılır.