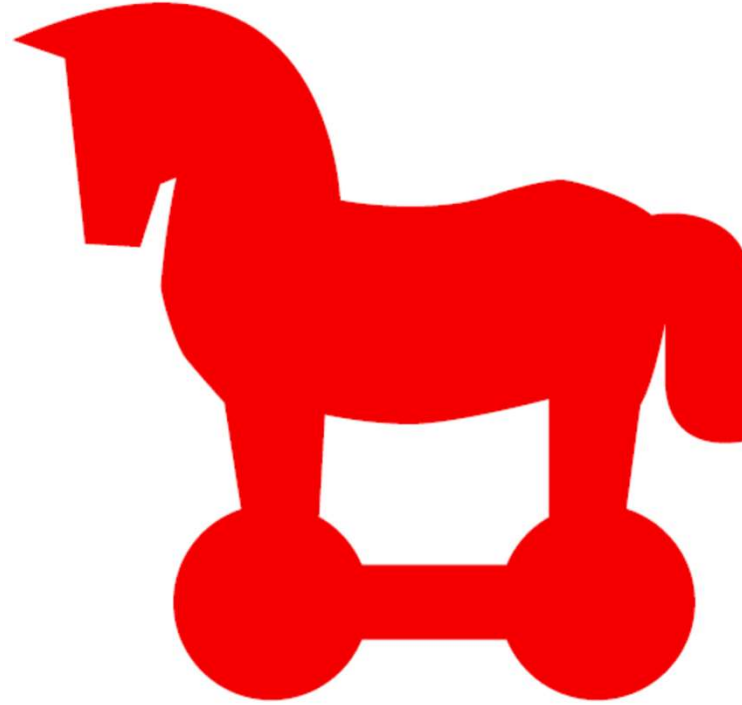


GANTEK
INTEGRATING FUTURE



GANTEK
ACADEMY

40 YILLIK TECRÜBE İLE

Docker Compose

Büyük Ölçekli Sistemler

- Şu ana kadar gördüğümüz bilgiler ışığında Docker konteynerlerini oluşturup, her birini çalıştırıp sonuçlarını görebiliyoruz. Her bir Docker komutu tek bir konteyneri ilgilendiriyordu. Onlarca konteynerin birbiriyle haberleşecek şekilde bir arada çalıştırılmasının gerektiği bir ortam düşünün.
- Özellikle mikroservis mimarilerinde, birbirileriyle haberleşen, sanal ağ içinde çalışan onlarca hatta yüzlerce konteyner olabilir. Uğraşmanız gereken yüzlerce mikro iş olabilir.
 - Bir programı oluşturan mikroservislerin çalıştığı konteynerlerin bir anda ayağa kaldırılacağını düşünün.
 - Volume yapıları
 - Sanal ağlar
- Aynı mikroservis yapısının, başka bir konakta ayağa kaldırılacağını düşünün.
 - Sistemi çalıştırmak için ne yapacaksınız?
 - Tekrar aynı işlemleri tekrar mı edeceksiniz?
 - Kabuk betikleri iş görür mü?

Docker Compose

- **Docker Compose**, çok sayıda konteynerin bir arada çalıştığı bir sistemde, tüm yapıyı otomatik olarak çalıştırmak için kullanılan bir programdır.
- «**Docker Compose**» aslında bir konteyner orkestrasyon yazılımıdır.
- Kubernetes, birden çok konakla ilgilenirken, **Docker Compose**, tek bir konak üzerinde konteynerlerin bir arada çalıştırılmasını otomatik hale getirir.
- Yapılması gereken tek şey, sistemi tarif eden ve çalıştırılması için gerekli adımları gösteren YAML dosyasını oluşturabilmek.

Docker Compose

- Docker Compose üç adımda kullanılır:
 - Kullandığınız konteynerleri **Dockerfile** ile oluşturulabilecek şekle getirin.
 - Hangi «**volume**» kullanılacak?
 - Hangi portlar açılacak?
 - Hangi ağ üzerinden hizmet sunacak?
 - Uygulamanızdaki konteynerlerin birbirleriyle nasıl çalışacağını **docker-compose.yml** dosyasında tanımlayın.
 - «**docker-compose up**» komutunu vererek tüm uygulamanın çalışabilir hale getirilmesini sağlayın.

docker-compose.yml

- Dosyanın genel yapısı şöyledir:
 - Başlatma, durdurma ve yeniden kurma hizmetleri
 - Çalışan hizmetlerin durumunun görülmesi
 - Çalışan servislerin log çıktılarının akıtılması
 - Servis üzerinde çalışacak komutlar

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - ../code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Docker Compose

- **Docker Compose**, ortamı izolasyon altında tutabilmek için bir proje ismi kullanır.
- Proje isimleri, aynı yapının farklı proje isimleriyle kopyalarının bir sistem üzerinde çalışmasını sağlamaktadır.
- Daha önce çalışan projelerden kalan «**volume**»lar varsa **Compose** onu da kopyalamaktadır.
- Sadece değişen konteynerler, servisler yeniden başlatıldığında güncellenmekte, değişmeyenler ön bellekte tutulmaktadır.

Docker Compose nerede kullanılır?

- Kubernetes, Docker Compose uygulamasının yerine tercih edilen bir sistemdir. Ancak çeşitli yazılım geliştirme projelerinde konteyner test ortamının kurulması için faydalıdır.
- Genel olarak faydalı olduğu alanlar şunlardır:
 - Yazılım geliştirirken, bir uygulamanın izole bir ortamda çalışmasının sınanması
 - CI (Continuous Integration) için otomatik test ortamlarının oluşturulması. Test programlarının sınanacağı sistemin kurularak çalıştırılması
 - Tek bir konak sistem üzerinde çalışacak birden fazla konteynerden oluşan sistemlerin kurulması

<https://docs.docker.com/compose/#common-use-cases>

Docker Compose Kurulumu

- Docker Compose'un kurulumu için ilk önce Docker'ı kurmalısınız.
- «**sudo apt update**» komutunu çalıştırın
«**sudo apt install docker-compose**» komutuyla Docker Compose'u yükleyin. **Ancak bu yöntemle en son sürümü yüklenmeyebilir.**
- Docker Compose'un en son sürümünün ne olduğunu öğrenmek ve bunu yüklemek için Firefox'ta (veya diğer bir web izleyicisinde) **<https://github.com/docker/compose/>** adresine gidiniz ve sağ tarafta «**Releases**» adlı bölümde sağda gösterilen yere tıklayınız.

Report repository

3 years ago
1 months ago
last week
last year
last year
last month
3 years ago
1 months ago
2 weeks ago

Releases 205

v2.22.0 Latest 3 weeks ago

+ 204 releases

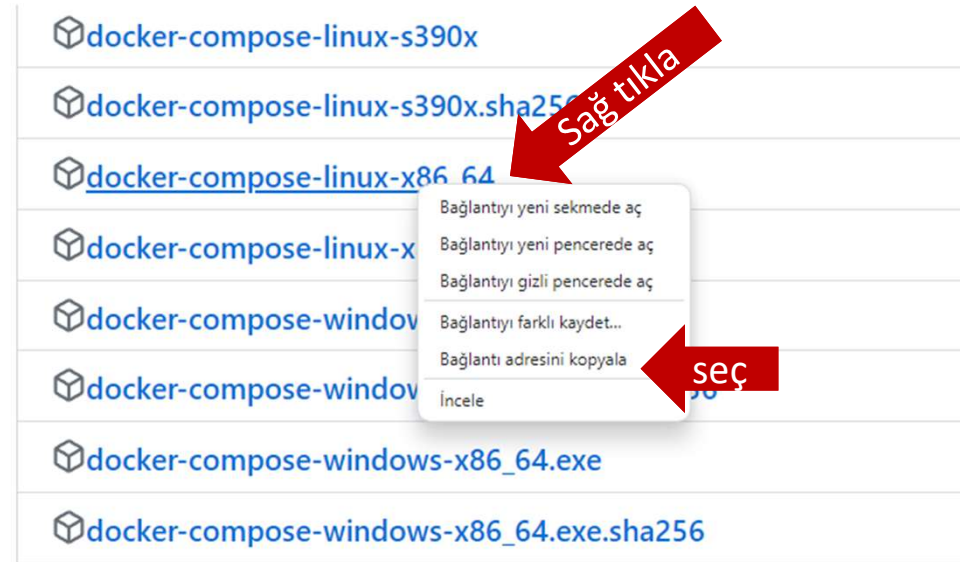
Packages

No packages published

<https://github.com/docker/compose/>

Docker Compose Kurulumu

- Docker Compose'un son sürümünün v2.22 olduğunu gördük ve Github sayfasında aşağıya giderek indirilebilir sayfaları göreceğiz.
- İşletim sisteminize ve işlemci tipinize uygun bir dosyanın linkini kopyalayıp «**curl**» komutuyla indirmelisiniz ve bu dosyayı **/usr/local/bin** dizinine «**sudo**» komutuyla kaydetmeniz gerekmektedir.



<https://github.com/docker/compose/releases/tag/v2.22.0>

Docker Compose Kurulumu

- Vereceğimiz komut şöyle olacaktır:

```
$ sudo curl -L  
"https://github.com/docker/compose/releases/download/v2.2  
2.0/docker-compose-linux-x86_64" -o  
/usr/local/bin/docker-compose
```

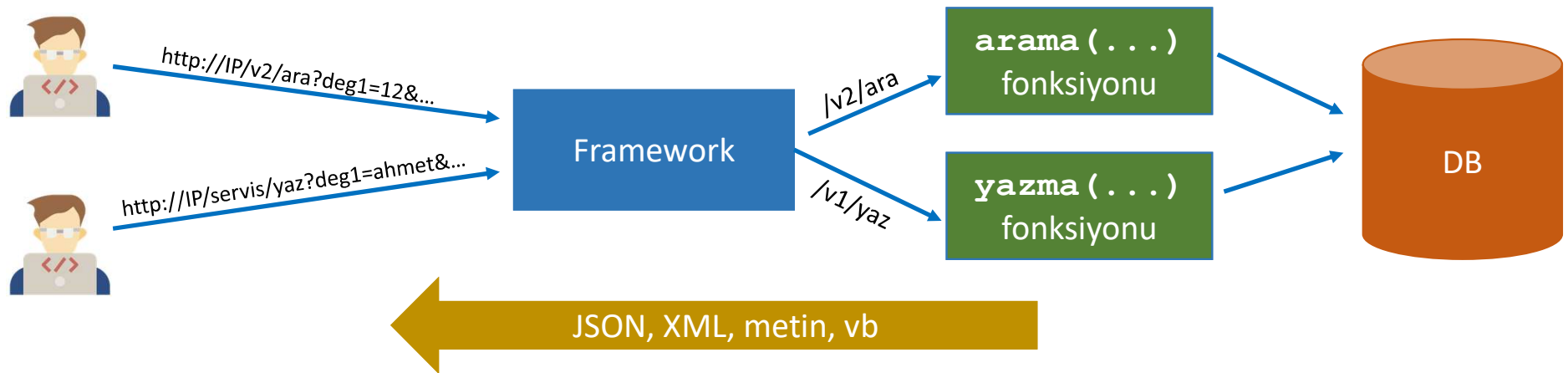
- Ardından da dosyanın izinlerini «**sudo chmod +x /usr/local/bin/docker-compose**» komutuyla değiştirin.

```
$ sudo curl -L  
"https://github.com/docker/compose/releases/download/v2.22.0/docker-compose-  
linux-x86_64" -o /usr/local/bin/docker-compose  
  % Total      % Received % Xferd  Average Speed   Time    Time     Time  Current  
                                 Dload  Upload   Total   Spent    Left   Speed  
  0     0     0     0     0     0     0     0  --:--:--  --:--:--  --:--:--     0  
100 56.8M  100 56.8M    0     0 5368k      0  0:00:10  0:00:10  --:--:-- 6065k  
$ sudo chmod +x /usr/local/bin/docker-compose  
$ docker-compose version  
Docker Compose version v2.22.0
```

<https://github.com/docker/compose/releases/tag/v2.22.0>

REST API mikroservisleri nasıl yazılır?

- REST API mikroservislerinin yazılması için HTTP protokolüne cevap veren yazılımların kullanılması gereklidir.
- Bu yazılım GET, PUT, DELETE vb HTTP v2 komutlarına yanıt vermeli ve verilen argümanlara göre istenen servis için yazılan kodu çalıştırmalı ve sonucu JSON, XML, metin, vb türde geri döndürmelidir.



REST API yazmak için hangi «framework»?

- REST API yazmak için, öncelikli olarak hangi dili kullanacağınızı belirlemeniz lazım ve buna göre «framework» seçmelisiniz.
- Her framework birbirine benzemez. Bazıları daha kapsamlıdır.
- Örneğin, bazıları farklı veritabanlarına bağlantıyı sağlayabilirken, diğerleri sınırlı sayıda özelliğe sahiptir.
- Bazı framework'ler tek iplikle (thread) hizmet sağlarken bazıları çok iplikli uygulamalara izin verir.
- Yaygın şekilde kullanılan «framework»ler kullanılırsa, destek almak kolaylaşır.
- Tavsiyemiz, kolay öğrenme imkanı olan, veritabanı bağlantılarına izin veren, kimlik doğrulama yapılabilen bir «framework»un tercih edilmesidir.

Pyhton'da hangi «framework»?

- Eğer Python ile yazılmış «framework» arıyorsanız, aşağıdaki seçenekler en fazla kullanılanlar arasında yer alır:
 - Django (kapsamlı bir kütüphanedir. ORM veya ORM olmayan veri kaynaklarına erişebilir. aktif bir topluluk tarafından geliştirilmektedir. Kimlik doğrulama hizmeti sunar. Red Hat bu projeyi desteklemektedir. Kapsamlı bir sistem olduğundan öğrenmesi zor olabilir)
 - Flask (Hızlı şekilde REST API hizmetleri oluşturmak için kullanılabilir. Çok az yer kaplar ve sağladığı hizmetler azdır. Çok kolay öğrenilir)
 - FastAPI (Yüksek performanslı bir «web framework»tür. Kolay kod yazılabilir ancak geliştirici sayısı dar olduğundan destek alımında sorun yaşanabilir)
 - Pyramid (Modüler bir mimariye sahiptir. Tasarım örüntülerine (MVC, MVP, vb) göre kod yazılabilir. Desteği zayıftır)
 - Falcon
 - Bottle
 - Eve
 - Sanic

REST API yazmak için hangi «framework»?

Framework	Performance	Community support	Suitable for
Django REST	It's a heavy-weight medium performing framework as it has a lot of components and dependencies	It has a huge community support	Building full-fledged web application.
Flask RESTful	Lightweight, fast performance framework	Massive community support	Building backend business logic.
FastAPI	Fast, asynchronous framework	Medium community support which is steadily growing	Building backend business logic
Pyramid	Full stack, medium performing framework, and its speed depend on its add-on libraries.	Comparatively smaller community	Building full-fledged web application.
Falcon	High performing framework	Comparatively smaller community	Building REST API interface and microservices backend.
Bottle	Fast, simple, light-weight framework	Smaller community support	Building REST API interface with backend business logic
Eve	Fast and it's built on top of the Flask framework.	Small community	Building REST API interface with backend logic.
Sanic	Fast, asynchronous framework	Medium-sized community support.	Building REST API interface with backend logic.
Tornado	The medium-performing framework can handle multiple connections at the same time.	Medium-sized community support	Building REST API interface with backend logic.

Java'da hangi «framework»?

- Java en yaygın programlama ortamlarından biridir.
- Java'daki framework'lerin en çok kullanılanları sırasıyla şunlardır:
 - **Spring MVC:** Yazılması gereken kodlar ağıdalıdır ve bu ortamda programlama kolay öğrenilmez. Spring Enterprise Java Uygulama Geliştirme ortamına aşinaysanız, kullanmanız gereken framework budur. Spring ortamı aşırı XML kullanımıyla meşhurdur ama web sunucusu olmadan çalışması büyük bir artıdır. JAX-RS standardıyla uyumsuzdur.
 - **Play:** Django, Ruby on Rails, ASP.NET MVC gibi ortamlara benzer bir yapısı vardır. Hızlı bir şekilde kod geliştirilebilir. Ancak mimarisi karmaşıktır, geriye yönelik uyumluluk bulunmamaktadır. Her zaman son sürümün kullanılması gerekir.
 - **Blade:** MVC framework'tür. Hafif bir ortamdır. Hızlıdır. Ancak kod yazımı oldukça zordur çünkü dokümantasyon zayıftır.
 - **Grails:** Java üzerinde çalışan Groovy dilinde yazılmıştır. Ayrıntılı bir dokümantasyonu vardır.
 - **Dropwizard:** hafif bir framework'tür. Veritabanı bağlantısı için dışarıdan kütüphane kullanılmalıdır. Kodlama bu nedenle karmaşık olabilir. ORM desteklenmez.

Node.js nedir?

- Açık kaynak kodlu, farklı işletim sistemi platformlarını destekleyen ve 2009'dan itibaren geliştirilmekte olan bir sunucu platformudur.
- Node.js aslında back-end Javascript çalıştırma ortamıdır ve Google'un 2008'da açık kaynak kodlu olarak yayınladığı V8 JavaScript motoru üzerinde çalışır. Sunucu tarafında Javascript kodu çalıştırmaktadır.
- Olayları yakalayan (event loop) bir döngüye ve düşük seviyeli I/O için bloklu olmayan (non-blocking) bir kütüphaneye sahiptir. Tek iplikte çalışır. Apache gibi çok iplikli (multithread) bir yapıya sahip değildir.
- Node.js, sunucu tarafındaki uygulamaların yazılmasını oldukça kolaylaştırır.
- Günümüzde yaygın olarak kullanılan bir çok IDE tarafından (Atom, Brackets, JetBrains WebStorm, Microsoft Visual Studio, NetBeans, Nodeclipse, Enide Studio, Visual Studio Code gibi) desteklenmektedir.

Springboot nedir? Spring'ten farkı nedir?

- Springboot, «backend» tarafında mikroservis geliştirmek için kullanılan bir «framework»tür ve Spring üzerinde çalışır. Spring ise web uygulaması yazmak için kullanılır.
- Springboot, Spring «framework» üzerinde çalışarak tek başına çalışan REST API uygulamalarını hızlı bir şekilde geliştirmek için kullanılır.
- Springboot Tomcat ve Jetty gibi sunucularla birlikte gelir ve az sayıda satırla çok fonksiyonlu uygulamalar hazırlanabilir.
- Spring framework'te uzun programlar yazmak gerekirken, Springboot'ta az satırda çok iş yapılabilir.
- Springboot'un en önemli özelliği otomatik konfigürasyondur. Spring'te ise konfigürasyon biraz ağdalıdır.

Python Flask nedir?

- Flask, Python'da web sunucu uygulamalarını kolaylıkla geliştirmek için kullanılan bir uygulamadır.

<https://flask.palletsprojects.com/en/3.0.x/>

- En basit şekilde bir Flask uygulaması şu şekilde yazılabilir:

hello.py



```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

- Komut satırından «**flask run**» denilerek web uygulaması ayağa kaldırılabilir. Ancak önceden ortam değişkeni olan **FLASK_APP**'a hangi Python programını çalıştıracağını belirtmek gereklidir:

```
export FLASK_APP=hello
flask run
```

<https://flask.palletsprojects.com/en/3.0.x/quickstart/#a-minimal-application>

Python Flask

- «**flask run**» komutu çalıştırıldığında, flask 5000 portundan gelen taleplere yanıt dönmektedir. Flask WSGI uygulamasıdır. HTTP taleplerine cevap verir.
- WSGI (Web Server Gateway Interface) sunucusu Python kullanarak web uygulaması geliştirmek için kullanılır.

```
$ export FLASK_APP=hello
$ flask run
* Serving Flask app 'hello'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [09/Oct/2023 14:30:30] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2023 14:30:30] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [09/Oct/2023 14:32:01] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Oct/2023 14:32:03] "GET / HTTP/1.1" 200 -
```

Redis nedir?

- Redis, açık kaynak kodlu anahtarlar ve buna karşılık gelen değerlerin tutulduğu bir veritabanı olarak düşünülebilir. Redis, Redis Labs (<http://redis.io>) adlı şirket tarafından geliştirilmektedir.
- Redis, anlam olarak «Uzak Sözlük Sunucusu» yani «**RE**mote **D**ictionary **S**erver» anlamına gelmektedir.
- Uygulamalarda, bazı verilerin merkezi olarak tutulması için çok sık kullanılan bir hizmetidir.
- Farklı tür veri yapılarını (string, list, set, sorted set, stream, vb) tutabilir ve istendiğinde verilen anahtarla bu veri yapılarına erişilebilir.
- Yaygın olarak kullanıldığı yerler (use-case) şunlardır:
 - Gerçek zamanlı veri depolama
 - Önbellek, durum ve oturum bilgileri depolama
 - Akım (streaming) ve mesajlaşma

Redis'in Özellikleri

- **Redis'in** kullanışlı özellikleri bulunur:
 - Depoladığı veri yapılarını bellekte tutar ve bellekten hızlı şekilde sunar ama depolanan veriyi diskte de tutar.
 - Programlama arayüzü (API) sunar
 - Lua dilinde sunucu tarafında betik programlar yazılabilir.
 - Redis'e ek özellikler tanımlamak için C, C++ ve Rust dillerinde API sağlar
 - Kümelenebilir bir yapıdır. Çok sayıda konakla ortak çalışabilir.
 - Kümelenmiş ortamlarda replikasyon ve hata durumunda tekrar çalıştırma gibi opsiyonlar sunar.

Redis - Sandbox

- Redis'le, herhangi bir bilgisayara kurulum yapılmadan, **<https://try.redis.io/>** adresinde deneme yapılabilir.
- Bu adresteki **TUTORIAL** bölümü, Redis'le ilgili özet bilgi sunmakta ve veritabanı sunucusuna bağlantı kurulması, anahtar-değerlerin depolanması ve diğer özelliklerle ilgili bilgi vermektedir.
- Redis'le ilgili herhangi bir kurulum yapmadan önce bu sitede Redis'e alışmak ve komutları deneyerek nasıl kullanabileceğinizi planlamak faydalı olacaktır.

Ubuntu üzerinde kurulum

- İlk önce, APT için paket listelerini oluşturmak gereklidir

```
$ curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg
```

```
$ echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/redis.list
```
- Ardından, sisteme **redis** kurulabilir:

```
sudo apt-get update
```

```
sudo apt-get install redis
```
- Kurulumun gerçekleştiğini test etmek için **redis-cli** programı kullanılabilir.

```
$ redis-cli ping
PONG
$
```

<https://redis.io/docs/getting-started/installation/install-redis-on-linux/>

Redis TCP arayüzü

- Redis, 6379 numaralı TCP portundan bağlantıları bekler.
- Anahtar-değer ikilileri bu arayüz üzerinden veritabanına eklenebilir.
- Eğer istenirse, «**telnet 127.0.0.1 6379**» komutuyla, Redis arayüzüne bağlanılabilir.

```
$ telnet 127.0.0.1 6379
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
ping
+PONG
set giris selam
+OK
get giris
selam
```

redis-cli

- **redis-cli**, Redis'in komut satırı arayüzüdür ve **/usr/bin** dizininde yer almaktadır.
- **redis-cli**, telnet üzerinden bağlantıya göre daha iyi bir ara yüz imkanı sağlar.
- **redis-cli** eğer istenirse, interaktif modda veya komut satırı modunda kullanılabilir:

```
$ redis-cli  
127.0.0.1:6379> incr sayi  
(integer) 1  
127.0.0.1:6379> incr sayi  
(integer) 2  
127.0.0.1:6379> exit
```

İnteraktif mod

```
$ redis incr sayi  
3
```

Komut satırı modu

redis-cli komut satırı modu

- **redis-cli**, Bash üzerinden komut gönderilmesini ve çıktısının bir dosyaya yönlendirilmesini sağlamaktadır.

```
redis-cli inc sayi > cikti.txt
```

- **redis-cli**, Redis'in bulunduğu konağın IP adresi (-h opsiyonuyla) ve portuna (-p opsiyonuyla) bağlanmamızı sağlamaktadır.

```
redis-cli -h 192.168.1.22 -p 10201
```

- **redis-cli** ile pipe özelliklerini kullanabilirsiniz.

```
$ cat > komutlar.txt  
INCR sayi  
INCR sayi2  
GET sayi2
```

```
$ cat komutlar.txt | redis-cli
```

komutlar.txt
isimli içinde REDIS
komutları bulunan bir
dosya oluşturuyoruz.

komutlar.txt
içindeki komutları
redis-cli'ye
gönderiyoruz.

Redis içindeki veritabanları

- Redis içinde, numaralarla belirlenen farklı veritabanları bulunmaktadır. Bu veritabanları birbirinden farklıdır ve aynı isimli değişkenler olsa da içerikleri farklıdır.
- Redis ilk kurulduğunda varsayılan olarak 16 farklı veritabanı oluşturur ve «**redis-cli -n <veritabanı-no>**» ile hangi veritabanı üzerinde işlem yapılacağı belirtilebilir.

```
$ redis-cli FLUSHALL
OK
$ redis-cli -n 1 INCR a
(integer) 1
$ redis-cli -n 1 INCR a
(integer) 2
$ redis-cli -n 2 INCR a
(integer) 1
```

FLUSHALL bütün
anahtar-değer
ikililerinin
sıfırlanmasını sağlar

ilk Docker Compose uygulaması

İlk «**Docker-Compose**» Uygulamamız

- **Flask**'e dayalı bir web uygulaması olacak.
- Web uygulama konteynerinin adı
«**proje/microservis_flask**»
http://konteyner-IP:5000
- Docker Hub'dan hazır bir **redis** mod imajı alacağız ve üzerinde değişiklik yapmadan kullanacağız.
- Flask'te çalışan Python uygulaması, **Redis** sunucusuna 6379 no'lu port üzerinde ulaşarak «**hits**» isimli bir veri yapısını alacak ve web sayfasında gösterecek ve ardından «**hits**»i bir artıracak.
- Diğer bir deyişle bir sonraki sefer web sayfasına girildiğinde, web sayfasında gösterilen bu değerin bir fazlası olması sağlanacak.

İlk «Docker-Compose» Uygulaması

- İlk Docker Compose uygulamamız için, ilk önce ev dizinimizde **composetest** isimli bir dizin oluşturalım.

```
mkdir composetest  
cd composetest
```

- Bu dizinde **app.py** programını ve **requirements.txt** dosyalarını oluşturun.

app.py

```
import time  
import redis  
from flask import Flask  
  
app = Flask(__name__)  
cache = redis.Redis(host='redis', port=6379)  
  
def get_hit_count():  
    retries = 5  
    while True:  
        try:  
            return cache.incr('hits')  
        except redis.exceptions.ConnectionError as exc:  
            if retries == 0:  
                raise exc  
            retries -= 1  
            time.sleep(0.5)  
  
@app.route('/')  
def hello():  
    count = get_hit_count()  
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

requirements.txt

```
flask  
redis
```

<https://docs.docker.com/compose/gettingstarted/>

Dockerfile

- Uygulamada **redis** sunucusunu hazır şekilde Docker Hub'dan alacağız ama **flask** çalıştıran web sunucusunu oluşturmak için **Dockerfile**'a ihtiyaç duyuyoruz. **Dockerfile**'ı **composetest** dizinine ekliyoruz.

```
FROM python:3.10-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache build-base musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Compose dosyası

- «**docker-compose.yml**» dosyası, uygulama içinde hangi servislerin olacağını söylüyor. «**composetest**» dizinine konulacak.

```
services:
  redis:
    image: redislabs/redismod
    container_name: redis
    ports:
      - '6379:6379'
  flask:
    build: .
    container_name: flask
    image: proje/mikroservis_flask
    ports:
      - "8000:5000"
    volumes:
      - .:/code
    depends_on:
      - redis
```

Doğrudan imaj
olarak Docker
Hub'dan alınacak

Dockerfile'dan
oluştur (build)

Konakla
konteyner 8000
TCP portundan
bağlanacak

«**docker image
ls**» komutuyla listede
bu adla görülecek.

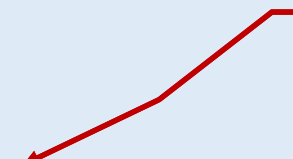
«docker-compose build»

NOT: Çalışan bütün konteynerleri kapatın «**docker kill \$(docker ps -q)**» ve ardından «**docker rm \$(docker ps -a -q)**» diyerek kaldırın. Çünkü çakışma olabilir!

```
$ docker-compose build
[+] Building 0.7s (11/11) FINISHED                                docker:default
=> [flask internal] load build definition from Dockerfile        0.0s
=> => transferring dockerfile: 299B                               0.0s
=> [flask internal] load .dockerignore                          0.0s
=> => transferring context: 2B                                     0.0s
=> [flask internal] load metadata for docker.io/library/python:3.10-alpine 0.5s
=> [flask 1/6] FROM docker.io/library/python:3.10-alpine@sha256:fe57859a64e9d9ade56dc63daffc 0.0s
=> [flask internal] load build context                          0.0s
=> => transferring context: 516B                                   0.0s
=> CACHED [flask 2/6] WORKDIR /code                             0.0s
=> CACHED [flask 3/6] RUN apk add --no-cache build-base musl-dev linux-headers 0.0s
=> CACHED [flask 4/6] COPY requirements.txt requirements.txt     0.0s
=> CACHED [flask 5/6] RUN pip install -r requirements.txt        0.0s
=> [flask 6/6] COPY . .                                         0.0s
=> [flask] exporting to image                                    0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:18ea1da5d984d3839aec49cfae6ee5df5ee29f709a6b33de9dbcc5f042504acf 0.0s
=> => naming to docker.io/library/composetest-flask             0.0s
```

«docker-compose up»

```
$ docker-compose up
[+] Building 0.0s (0/0)
[+] Running 2/2
    ✓ Container redis Created
    ✓ Container flask Recreated
Attaching to flask, redis
redis | 1:C 09 Oct 2023 13:58:24.125 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis | 1:C 09 Oct 2023 13:58:24.125 # Redis version=6.2.6, bits=64, commit=00000000, modified=0,
pid=1, just started
redis | 1:C 09 Oct 2023 13:58:24.125 # Configuration loaded
redis | 1:M 09 Oct 2023 13:58:24.126 * monotonic clock: POSIX clock_gettime
redis | 1:M 09 Oct 2023 13:58:24.128 * Running mode=standalone, port=6379.
redis | 1:M 09 Oct 2023 13:58:24.128 # Server initialized
.....
flask | * Serving Flask app 'app.py'
flask | * Debug mode: off
flask | WARNING: This is a development server. Do not use it in a production deployment. Use a
production WSGI server instead.
flask | * Running on all addresses (0.0.0.0)
flask | * Running on http://127.0.0.1:5000
flask | * Running on http://172.18.0.3:5000
flask | Press CTRL+C to quit
flask | 172.18.0.1 - - [09/Oct/2023 13:58:38] "GET / HTTP/1.1" 200 -
flask | 172.18.0.1 - - [09/Oct/2023 13:58:40] "GET / HTTP/1.1" 200 -
```



Her bağlandıkça
yeni bir satır
görülecek

Test

- «**docker-compose up -d**» komutundaki **-d** opsiyonuyla oluşturulan konteyner sisteminin geri planda çalışması sağlanabilir. Bu durumda loglara erişmek için «**docker-compose logs -f**» komutu kullanılabilir.
- «**curl http://localhost:8000**» komutuyla, **redis**'teki değerin artırıldığını görebilirsiniz.

```
$ curl http://127.0.0.1:8000
Hello World! I have been seen 3 times.
$ curl http://127.0.0.1:8000
Hello World! I have been seen 4 times.
$ curl http://127.0.0.1:8000
Hello World! I have been seen 5 times.
$ curl http://127.0.0.1:8000
Hello World! I have been seen 6 times.
$ curl http://127.0.0.1:8000
Hello World! I have been seen 7 times.
```

İmajlar

- İki sunucuyu (**redis** ve **proje/mikroservis_flask**) birbirine bağlayarak sonuca ulaştık.
- Yapılan işlerin çalışması için bir imajın oluşturulması gerekiyor.
- Bu imajlar nerede? «**docker image ls**» komutuyla görülebilir.

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
proje/mikroservis_flask	latest	88a64dd56bc1	4 minutes ago	312MB
composetest-flask	latest	18ea1da5d984	13 minutes ago	312MB
myapp	latest	3b3f87e5d2ea	6 days ago	230MB
ubuntu-benim	latest	4beeb61d8a17	6 days ago	181MB
ubuntu-guncel	latest	0d9d293590a3	6 days ago	72.8MB
alpine	latest	8ca4688f4f35	10 days ago	7.34MB
redislabs/redismod	latest	88923bcac4ad	15 months ago	1.68GB

İkinci Docker Compose uygulaması

İkinci «**Docker-Compose**» Uygulaması

- Konteyner ortamında Wordpress kuracağız ve bunu da MySQL'e bağlayacağız.
- Bu amaçla iki servis oluşturmamız gerekli:
 - **db**: MySQL 5.7 kurulacak ve kullanıcı adı/şifreyi belirlememiz gerekli.
 - **wordpress**: **db** servisine bağlanacak ve veritabanı olarak oradaki veritabanını kullanacak.
- Daha önce çalıştırdığımız konteynerler varsa (eğer 80 no'lu portu kullanıyorsa) bunları durdurmanız gerekebilir.

<https://docs.docker.com/samples/wordpress/>

docker-compose.yml

db servi
MySQL

Wordpress
servisi

```
version: "3.9"

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
  wordpress_data: {}
```

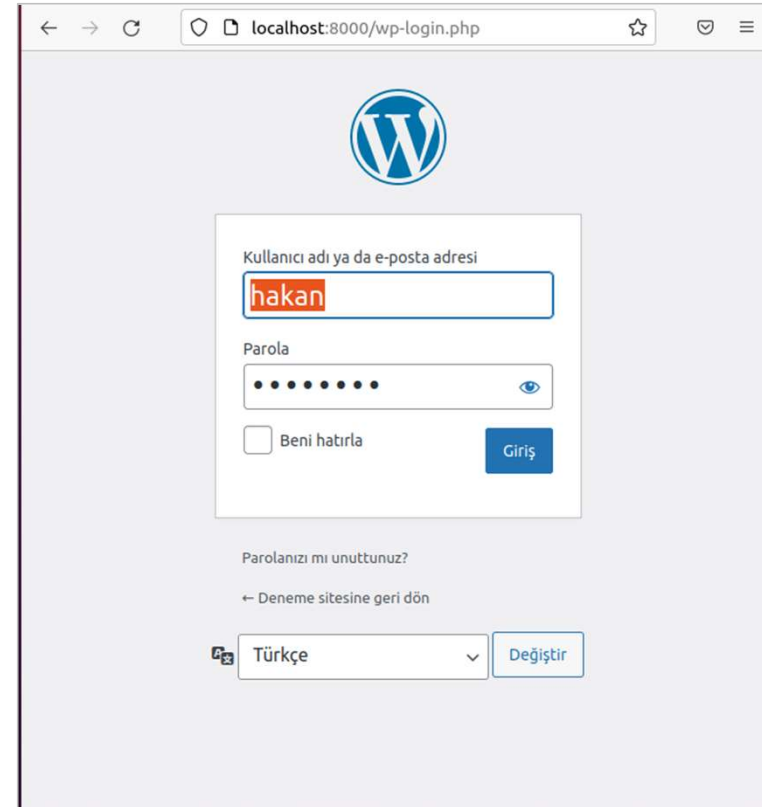
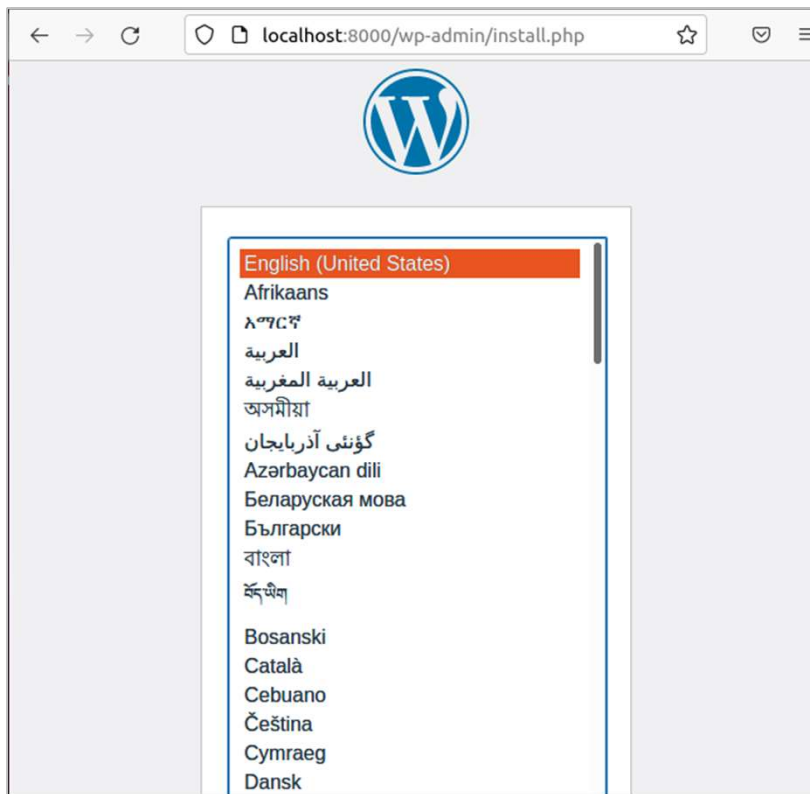
«**docker-compose up -d**»

- «**docker-compose.yml**» dosyasını **\$HOME** dizin altında deneme dizinini oluşturarak bunun içine kaydedin.
- «**docker-compose up -d**» komutu çalıştırıldığında, tüm sistem derlendikten sonra **-d** opsiyonu sayesinde geri planda çalıştırılmaktadır.
- Daha önce çalışan konteynerler varsa durdurulur ve silinir.
- Firefox'ta «**http://localhost:8000**» komutuyla Wordpress'e erişebilirsiniz.

```
$ docker-compose up -d
[+] Building 0.0s (0/0)
docker:default
[+] Running 2/2
✓ Container deneme-db-1          Started          0.0s
✓ Container deneme-wordpress-1   Started          1.2s
```

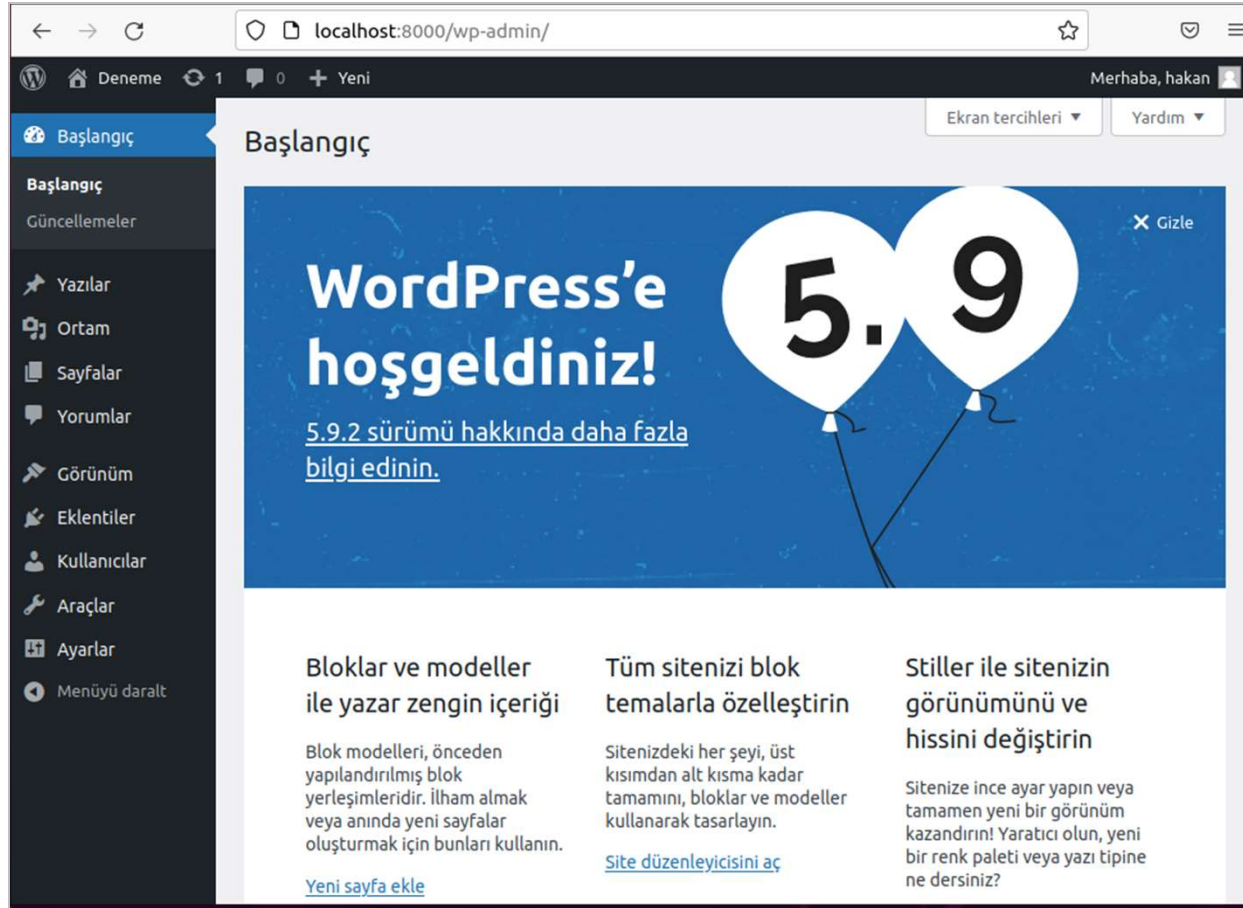
Firefox üzerinde Wordpress

- Firefox'ta **http://localhost:8000** ile Wordpress çalışmaya başlar.



Firefox üzerinde Wordpress

- Tanımlanan kullanıcı adı/şifre girildikten sonra Wordpress'in ana sayfasına ulaşılır.



Docker üzerinde PostgreSQL


Docker ve Kubernetes üzerinde PostgreSQL

- Gelecekte baskın duruma geleceği düşünülen bulut uygulamaları için PostgreSQL'i «konteyner» veya «pod» olarak çalıştırmak gerekmektedir.
- Replikasyonla birbirine bağlı konteynerlerin, Docker'ın ve Kubernetes'in sağlamış olduğu «Volume» özelliğiyle aynı veri dosyalarına erişmesi mümkündür.
- Ayrıca, Kubernetes'te «Service» kavramıyla bu sunucuların farklı bilgisayarlarda (node) çalışsalar bile aynı ağ içinde yer alabilirler.
- Pod/konteyner sayısının artırılmasıyla, yatay ölçeklemenin büyütülmesi mümkündür.
- Mikroservis mimarisi üzerinde tüm veritabanının bir kısmını bünyesinde bulunduran PostgreSQL sunucuların çalıştırılması senaryosu yakın gelecekte mümkün görünmektedir.

Docker üzerinde PostgreSQL çalıştırma

- Docker üzerinde PostgreSQL'in çalıştırılabilir.
- «**docker run**» komutuyla PostgreSQL imajı indirilerek, **POSTGRES_PASSWORD** ve **POSTGRES_DB** çevre değişkenlerinin değerleri verilerek konteyner çalıştırılabilir.

```
$ docker run --name test_postgres -e POSTGRES_PASSWORD=selamlar -e POSTGRES_DB=testdb -d postgres
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
f03b40093957: Pull complete
9d674c93414d: Pull complete
de781e8e259a: Pull complete
5ea6efaf51f6: Pull complete
b078d5f4ac82: Pull complete
97f84fb2a918: Pull complete
5a6bf2f43fb8: Pull complete
f1a40e88fea4: Pull complete
4be673794a1a: Pull complete
9d72f84fb861: Pull complete
5d52569da92e: Pull complete
5d48fbe991ff: Pull complete
4ae692d11ad3: Pull complete
Digest: sha256:31c9342603866f29206a06b77c8fed48b3c3f70d710a4be4e8216b134f92d0df
Status: Downloaded newer image for postgres:latest
f4c7d0ff55721cfcfeafdad1f067a0628ade8128e24759f864d9b455277fe28c
```



postgres kullanıcısının şifresi olacak

Postgres imajı «detached» modda çalıştırılacak.

Docker üzerinde PostgreSQL çalıştırma

- «**docker ps**» komutu, Docker'da çalışan konteynerlerin ID numaralarını ve konteyner isimlerini gösterir.
- Bütün «**docker**» komutları bu ID'ler ile çalışır.

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
f4c7d0ff5572   postgres "docker-entryp..."    9 seconds ago Up 4 seconds  5432/tcp     test_postgres

$ docker logs f4c7d0ff5572
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.utf8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Etc/UTC
.....
```


«**docker logs ID**»,
postgres konteyneri çalıştırılınca
oluşan loglara erişilmesini sağlar.

Konteynerler ID'leri üzerinden
takip edilir.

Docker üzerinde PostgreSQL çalıştırma

- «**docker exec -it**» komutu, Docker'da çalışan konteynere komut gönderilerek çalıştırılmasını sağlar. Gönderdiğimiz «**psql -U postgres -d testdb**» komutuyla çeşitli SQL deyimleri çalıştırabiliriz.

```
$ docker exec -it test_postgres psql -U postgres -d testdb -c "create table Test (Id integer, Name text);"
CREATE TABLE
$ docker exec -it test_postgres psql -U postgres -d testdb -c "insert into Test values (1, 'Bob');"
INSERT 0 1
$ docker exec -it test_postgres psql -U postgres -d testdb -c "select * from Test;"
 id | name
----+-----
  1 | Bob
(1 row)
$ docker kill test_postgres
test_postgres
$ docker rm test_postgres
test_postgres
```



Konteyneri öldürdüğümüzde, oluşturulan testdb'i de kaybolur çünkü testdb konteyner içinde oluşturulmuştur.

Docker üzerinde PostgreSQL çalıştırma

- «**docker exec -it test_postgres psql -U postgres**» komutu, **psql** programını çalıştırarak, konsolumuza bağlar ve istediğimiz komutu verebiliriz.

```
$ docker exec -it test_postgres psql -U postgres
psql (15.3 (Debian 15.3-1.pgdg110+1))
Type "help" for help.
postgres=# \c testdb
You are now connected to database "testdb" as user "postgres".
testdb=# \dt
          List of relations
 Schema | Name | Type  | Owner
-----+-----+-----+-----
 public | test | table | postgres
(1 row)
testdb=# \q
$
```

psql programından çıktığımızda,
konteyner çalışmasına devam eder.

Küme (cluster) ne demek?

- SQL standardı, veritabanı kümesi kavramını belirtirken katalog kümesi deyimini kullanır.
- Bir veritabanı kümesi, bir grup veritabanı çalıştıran sunucunun tek bir veritabanı sunucusu üzerinden yönetilerek kullanılmasıdır.
- Dosya sisteminde, bir veritabanı kümesi, kümedeki bütün sunucular tarafından kullanılan veritabanı dosyalarına tek bir dizin üzerinden erişir.
- Kümedeki her bir sunucu lokal dizini olarak gözüken **/usr/local/pgsql/data** dizinindeki dosyalara ulaşırken, ağ üzerinden (NFS, SAMBA, vb yöntemlerle) aynı dosyalara erişmektedir. Buna veri dizini veya veri alanı adını vereceğiz.

Docker'da «**volume**» kavramı

- Herhangi bir konteyner durdurulduğunda, içindeki veriler de kaybolur. Bu durum konteynerlerin «**ephemeral**» olmasıdır ve içinde tuttuğu veri de konteynerle birlikte silinir.
- Docker, verilerin konteyner dışında tutulması için «**volume**» özelliği kullanılabilir. Oluşturulan bir «**volume**», konteyner içindeki bir dizine bağlanabilir. Örneğin, postgres konteynerinde veritabanı dosyalarının bulunduğu **/var/lib/postgresql/data** dizini, oluşturulan «**volume**»a bağlanırsa, konteyner çalışırken oluşan veri dosyaları bu dizin üzerinden «**volume**»a yazılır ve bu şekilde verinin kalıcılığı sağlanır.
- Docker «**volume**»ları kendi bünyesinde tutar ve yeni bir postgres konteyneri başlatılırken bu «**volume**»u kullanması sağlandığında, verilerin kalıcılığı sağlanmış olur.

Docker'da «**volume**» ile PostgreSQL çalıştırma

- «**docker create volume**» komutu, Docker'ın bünyesinde yeni bir «**volume**» oluşturmak için kullanılır.

pgdata alanını konteynerdeki
/var/lib/postgresql/data
dizinine bağladık.

```
$ docker volume create pgdata
pgdata
$ docker run -v pgdata:/var/lib/postgresql/data --name test_postgres -e
POSTGRES_PASSWORD=password -e POSTGRES_DB=testdb -d postgres
90541d5eb5f3c4b7447dc416ce3e3889b313a5a86d1eb193fe504aed8fd02d4f
$ docker ps
CONTAINER ID   IMAGE          COMMAND                                     CREATED        STATUS
PORTS         NAMES
90541d5eb5f3   postgres      "docker-entrypoint.s..."  10 seconds ago Up 6 seconds
5432/tcp      test_postgres
$ docker exec -it test_postgres psql -U postgres -d testdb -c "create table Test (Id
integer, Name text); insert into Test values (1, 'Bob');"
CREATE TABLE
INSERT 0 1
$ docker stop test_postgres
test_postgres
$ docker rm test_postgres
test_postgres
```

Konteyneri durdurup sildik.
ACABA VERİLER SAKLANACAK MI?

Konteynerdeki postgresQL
sunucusunda yeni bir tablo
oluşturduk ve yeni bir kayıt ekledik.

Docker'da «**volume**» ile PostgreSQL çalıştırma

- «**docker run**» komutuyla, **pgdata** «**volume**»unu kullanarak **postgres** imajından **test_postgres2** isimli yeni bir konteyner başlatacağız.
- Yeni konteyner, eski konteynerin veri dosyalarını **/var/lib/postgresql/data** dizininde göreceği için, **testdb** veritabanını ve **Test** tablosunu aynı şekilde kullanabilecek.

Önceki konteynerde olduğu gibi **pgdata** alanını konteynerdeki **/var/lib/postgresql/data** dizinine bağladık.

```
$ docker run -v pgdata:/var/lib/postgresql/data --name test_postgres2 -e  
POSTGRES_PASSWORD=password -e POSTGRES_DB=testdb -d postgres  
432acf48ab17e33260523e325273583b136e409e9c5f9066523f7acd5254e522  
$ docker exec -it test_postgres2 psql -U postgres -d testdb -c "select * from Test;"  
 id | name  
----+-----  
  1 | Bob  
(1 row)
```

Önceki konteynerde yazdığımız verilere ulaşabildik mi? EVET!!!

Ne öğrendik?

- Docker'da oluşturduğumuz «**volume**», bir konteyner tarafından oluşturulan veritabanı dosyalarının bulunduğu **/var/lib/postgresql/data** dizinine eşleştirildiğinde ve veritabanı şifresi olarak aynısı kullanıldığında, yeni bir konteyner eski dosyaları kullanabilir ve veri kaybı oluşmaz.
- Benzer bir mantık, Kubernetes'te de vardır. K8s'de oluşturulan «**volume**», podlar arasında ortak şekilde kullanılabilir ve bir pod kapansa bile diğer pod devreye alındığında, veritabanına ve tablolara benzer şekilde erişebilir ve hizmet verebilir.
- **SORU:** veritabanı dosyalarını, oluşturduğumuz «**volume**» üzerinden iki veya daha fazla konteyner/pod ile paylaştığımızda, daha fazla performans elde edebilir miyiz? HEM HAYIR HEM EVET!!!!

Docker-Compose ile Küme oluşturma (clustering)

Bu örneğimizde, iki PostgreSQL örneğini birbirinden izole ama replikasyonla bağlı şekilde çalıştıracamız.

Docker-compose ile postgresQL kümesi

- **Docker Compose**, bir konteyner orkestrasyon aracıdır.
- Amacımız, **Docker Compose** kullanarak **master** ve **slave** isimli postgresQL yüklü iki konteyner oluşturmak ve konteynerlerden biri **master** olarak çalışırken diğerini sadece **read-only** çalıştırmaktır.
- Bu amaçla, **deneme** isimli bir dizin oluşturacağız ve aşağıdaki dizin yapısını kuracağız.

```
$ tree .  
.  
├── docker-compose.yml  
├── master  
│   ├── data  
│   └── init.sql  
└── slave  
    └── data  
  
4 directories, 2 files
```

Hangi PostgreSQL imajı?

- Bütün PostgreSQL imajları replikasyonu desteklememekte ve imaj üzerinde değişiklik yapılması gerekmektedir.
- Örneğin, PostgreSQL'in resmi **postgres** isimli imajının replikasyonu desteklemediği bilinmektedir.
- Bu nedenle bitnami'nin PostgreSQL imajının kullanılması daha iyi olacaktır çünkü resmi PostgreSQL imajında bir konteyneri replikasyonla **standby** olarak çalıştırma olanağı bulunmamaktadır.
- Bitnami PostgreSQL imajı **root** yetkisi olmadan çalışabilmektedir. Bu tür imajların güvenlik katmanları daha sıkıdır ve bu nedenle Kubernetes ortamında da çalışabilmektedir:

<https://hub.docker.com/r/bitnami/postgresql>

Bitnami PostgreSQL imajının çevre değişkenleri

- **POSTGRESQL_REPLICATION_MODE**: Replikasyon modu. master ve slave. Varsayılan değer bulunmuyor.
- **POSTGRESQL_REPLICATION_USER**: Master düğüm oluşturulurken yaratılan replikasyon kullanıcısı.
- **POSTGRESQL_REPLICATION_PASSWORD**: Replikasyon kullanıcısının şifresi
- **POSTGRESQL_REPLICATION_PASSWORD_FILE**: Replikasyon kullanıcısının şifresinin tutulduğu dosya.
- **POSTGRESQL_MASTER_HOST**: master düğümün IP'si veya host ismi
- **POSTGRESQL_MASTER_PORT_NUMBER**: Master düğümün port numarası. Varsayılan olarak 5432.

Docker-compose ile postgresQL kümesi

- **deneme** isimli bir dizin oluşturup aşağıdaki yapıyı oluşturuyoruz.

```
$ mkdir deneme
$ mkdir deneme/master
$ mkdir deneme/master/data
$ mkdir deneme/slave
$ mkdir deneme/slave/data
$ tree deneme
deneme
├── master
│   └── data
└── slave
    └── data

4 directories, 0 files
```

Docker-compose ile postgresQL kümesi

- **deneme/master/init.sql** isimli dosyaya aşağıdaki ifadeleri ekliyoruz.

```
$ cat > deneme/master/init.sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

- Bir sonraki amacımız, deneme dizininin altında **docker-compose.yml** dosyasını oluşturmak.

Docker-compose ile PostgreSQL kümesi

- **deneme/master/init.sql** isimli dosyayı oluşturuyoruz.

```
$ cat > deneme/docker-compose.yml
version: '3'

services:
  master:
    image: bitnami/postgresql:latest
    environment:
      POSTGRES_PASSWORD: selamlar
      POSTGRES_REPLICATION_MODE: master
      POSTGRES_REPLICATION_USER: repuser
      POSTGRES_REPLICATION_PASSWORD: selamlar

    volumes:
      - ./master/data:/var/lib/postgresql/data
      - ./master/init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
    restart: always

  slave:
    image: bitnami/postgresql:latest
    environment:
      POSTGRES_PASSWORD: selamlar
      POSTGRES_REPLICATION_MODE: slave
      POSTGRES_REPLICATION_USER: repuser
      POSTGRES_REPLICATION_PASSWORD: selamlar
      POSTGRES_MASTER_HOST: master
      POSTGRES_MASTER_PORT: 5432
    volumes:
      - ./slave/data:/var/lib/postgresql/data
    restart: always
```

```
$ tree deneme
deneme
├── docker-compose.yml
├── master
│   ├── data
│   └── init.sql
└── slave
    └── data

4 directories, 2 files
```

Docker-compose ile postgresQL kümesi

- «**docker-compose up**» komutu, **master** ve **slave** isimli iki konteyneri çalıştırarak birbirine bağlayacak.

```
$ docker-compose up -d
Creating network "deneme_default" with the default driver
Creating deneme_master_1 ... done
Creating deneme_slave_1 ... Done
```

- «**docker-compose exec master psql -U postgres**» yazılarak **master** düğüme bağlanılabilir. Bu konteynerde, tablolara hem yazma hem de okumanın izin verilmektedir. «**slave**»de sadece okuma temelli komutlar uygulanabilir.

```
$ docker-compose exec master psql -U postgres
Password for user postgres:
psql (15.3)
Type "help" for help.

postgres=# CREATE DATABASE testdb;
CREATE DATABASE
postgres=# CREATE TABLE kullanıcı (ID int, name text);
CREATE TABLE
postgres=#
```

Diğer Uygulamalar

- Çeşitli Docker-Compose örnekleri için aşağıdaki sitelerden yararlanılabilir:
 - Bir çok örnek için
★ <https://github.com/docker/awesome-compose>
 - Rails
 - <https://docs.docker.com/samples/rails/>
 - Django
 - <https://docs.docker.com/samples/django/>
 - Oy uygulaması
 - <https://github.com/dockersamples/example-voting-app>
 - MongoDB, Node.js
 - <https://www.bogotobogo.com/DevOps/Docker/Docker-Compose-Node-MongoDB.php>

Docker üzerinde node.js temelli REST API konteyneri ve yük paylaşımı

Konular

- node.js
- REST API
- Load Balancer ne demek?
- NGINX

Node.js nedir?

- Node.js, açık kaynak kodlu ve olay temelli asenkron Javascript çalışma ortamı olarak, Node.js, ölçeklenebilir ağ uygulamaları geliştirme amacıyla kullanılır.
- V8 JavaScript Chrome motoru kullanır.
- Node.js tek bir iplik (thread) olarak çalışır ve çok iplikli yapıda değildir.
- Ağ üzerinden gelen talepler, sonsuz bir döngü içinde bekleyen bir program parçası tarafından karşılanır ve talep hangi porttan geldiyse, o portla ilgili callback çağrılarak işlem yapılır.
- Basit uygulamalar için, örneğin HTTP üzerinden REST API uygulamaları için basit bir çözüm sunar. Kilitlenme veya takılma (deadlock) gibi bir risk bulunmamaktadır.

Node.js nasıl kurulur?

- İlk önce sistemi güncellemek gerekir.

```
$ sudo apt update -y  
$ sudo apt upgrade -y  
$ sudo apt install curl
```
- Curl kurulduktan sonra, Node.js'in paket sunucusunu sistemimizdeki paket sunucu listesine ekleyelim:

```
$ curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E  
bash -
```
- Node.js'i kuralım:

```
$ sudo apt-get install -y nodejs
```
- NPM'i (javascript betik dili için geliştirilmiş bir paket yönetim sistemi) kuralım:

```
$ sudo apt-get install -y npm
```
- Express paketini (<https://expressjs.com/>) kuralım. Bu paket programlarımızda kullanılacak.

```
$ npm install express --save
```

Node.js programı

- Aşağıdaki örnek, Node.js programlarına güzel bir örnektir.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Node.js programını çalıştırmak

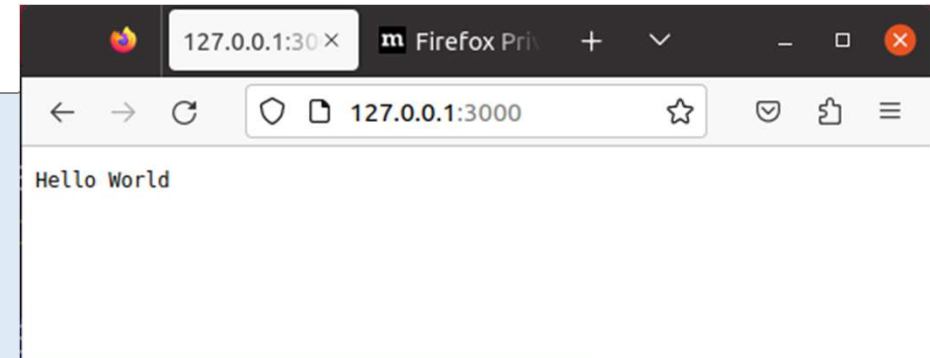
- Önceki slayttaki programı, **deneme.js** dosyası adı altında ev dizinimize kaydedelim ve «**node deneme.js**» komutuyla çalıştıralım. Firefox üzerinden «**http://127.0.0.1:3000/**» URL'ini verelim ve «**Hello World**» mesajının görüntülendiğini görelim.

```
$ cat > deneme.js
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
$ node deneme.js
Server running at http://127.0.0.1:3000/
```



Node.js ile REST API nasıl yazılır?

- Node.js ile REST API yazmak oldukça kolaydır.
- Şöyle bir uygulama yazalım.
 - **http://IP-adresi/listUsers** : URL'i çalıştırınca sistemdeki kullanıcıları gösterebilirsin. Kullanıcılar, **users.json** dosyasında bulunmaktadır. Bu URL, **users.json** dosyasını HTTP üzerinden geri göndermektedir. Bu amaçla HTTP GET komutu kullanılacaktır.
 - **http://IP-adresi/addUser** : URL'i çalıştırdığımız zaman, POST komutuyla yeni kullanıcının ismini ve diğer özelliklerini **users.json** dosyasına ekler.

users.json dosyası

```
{
  "user1" : {
    "name" : "hakan",
    "password" : "password1",
    "profession" : "professor",
    "id": 1
  },
  "user2" : {
    "name" : "deniz",
    "password" : "password2",
    "profession" : "student",
    "id": 2
  },
  "user3" : {
    "name" : "zeliha",
    "password" : "password3",
    "profession" : "teacher",
    "id": 3
  }
}
```


listUsers URI

server.js

```
var express = require('express');
var app = express();
var fs = require("fs");

app.get('/listUsers', function (req, res) {
  fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
    console.log( data );
    res.end( data );
  });
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

8081 portunu dinliyor!!!
Bu portu Dockerfile içinde
EXPOSE etmeliyiz

https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm

<https://expressjs.com/en/guide/routing.html>

addUser URI

server.js

```
var express = require('express');
var app = express();
var fs = require("fs");

var user = {
  "user4" : {
    "name" : "ahmet",
    "password" : "password4",
    "profession" : "teacher",
    "id": 4
  }
}

app.post('/addUser', function (req, res) {
  // First read existing users.
  fs.readFile( __dirname + "/" + "users.json", 'utf8', function (err, data) {
    data = JSON.parse( data );
    data["user4"] = user["user4"];
    console.log( data );
    res.end( JSON.stringify(data));
  });
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

https://www.tutorialspoint.com/nodejs/nodejs_restful_api.htm

Konteyner olarak nasıl çalıştırırız?

- Node.js'te yazdığımız programları Konteyner içinde çalıştırmak için, **Dockerfile** oluşturmamız gereklidir.

```
FROM node:8.11-slim

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json .

RUN npm install

COPY ./ .

EXPOSE 8081

CMD ["npm", "start"]
```

package.json dosyası, Node.js projesinin başlangıç noktasıdır. Proje hakkında bütün metaverileri içermesi gereklidir.

npm (Node Package Manager) package.json içindeki metaverilere göre yapıyı kurar.

package.json içinde «start» olarak tanımlanmış JS programını çalıştırır.

package.json

- Node.js projesi için gerekli meta verileri bu dosyada tanımlanmalıdır.

```
{  
  "name": "gantek_rest_api",  
  "version": "1.0.1",  
  "description": "Gantek Akademi",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js"  
  },  
  "author": "Gantek Akademi",  
  "license": "GPL v2",  
  "dependencies": {  
    "express": "^4.16.4"  
  }  
}
```

«npm start» denildiğinde
çalıştırılacak komut

Express web framework
yüklenmeli ve versiyonu
4.16'dan daha yeni olmalıdır.

Dockerfile'dan imaj oluşturma

- Bir dizin oluşturalım ve içine şu dosyaları kopyalayalım:
`users.json`
`Dockerfile`
`server.js`
`packages.json`
- «`sudo docker build -t gantek-rest-api .`» komutuyla `gantek-rest-api` imajını oluşturacağız.
- Ardından «`docker run`» komutuyla konteyneri çalıştıracacağız.

Dockerfile'dan imaj oluşturma

- Node.js projesi için gerekli meta verileri bu dosyada tanımlanmalıdır.

```
$ sudo docker build -t gantek-rest-api .  
[+] Building 7.7s (11/11) FINISHED  
=> [internal] load build definition from Dockerfile 0.0s  
=> == transferring dockerfile: 397B 0.0s  
=> [internal] load .dockerignore 0.0s  
=> == transferring context: 2B 0.0s  
=> [internal] load metadata for docker.io/library/node:8.11-slim 1.4s  
=> [1/6] FROM docker.io/library/node:8.11-slim@sha256:682383b9e173828b78 0.0s  
=> [internal] load build context 0.0s  
=> == transferring context: 354B 0.0s  
=> CACHED [2/6] RUN mkdir -p /usr/src/app 0.0s  
=> CACHED [3/6] WORKDIR /usr/src/app 0.0s  
=> [4/6] COPY package.json . 0.0s  
=> [5/6] RUN npm install 6.0s  
=> [6/6] COPY ./ . 0.0s  
=> exporting to image 0.1s  
=> == exporting layers 0.1s  
=> == writing image sha256:6c47ba6deb26cef5b5f5594c250b0d98b3f0ae2642dd8 0.0s  
=> == naming to docker.io/library/gantek-rest-api 0.0s  
$ sudo docker images  
REPOSITORY TAG IMAGE ID CREATED SIZE  
gantek-rest-api latest 6c47ba6deb26 35 seconds ago 186MB  
alpine latest b2aa39c304c2 3 weeks ago 7.05MB  
hello-world latest feb5d9fea6a5 17 months ago 13.3kB
```

«Dockerfile» içinde tanımlı
imajı oluşturacağız

Oluşturulan imaj, «docker
images» komutuyla
görülebilir.

İmajı çalıştırma

- «**docker run**» komutu, oluşturduğumuz imajı çalıştırmak için kullanılabilir.

```
$ sudo docker run -it -p 8080:8081 --name=gantekrest7 gantek-rest-api
```

```
> gantek_rest_api@1.0.1 start /usr/src/app  
> node server.js
```

Example app listening at http://:::8081

Konakta 8080 numaralı porttan erişilebilecek. Konteynerin içinde 8081 portu EXPOSE durumda.

ÖDEV

- Aşağıdaki siteden örneklerden birini seçin ve uygulayın.
 - <https://github.com/docker/awesome-compose>

podman
buildah
skopeo

podman nedir?

- Podman, Docker gibi, OCI konteynerlerinin çalıştırılması için kullanılan bir konteyner motorudur.
- podman, daemon'u olmayan bir konteyner motorudur.
- Linux sistemleri üzerinde OCI (Open Containers Initiative) konteynerleri çalıştırmak ve yönetmek için kullanılır.
- **En önemli özelliği (Docker'ın aksine) konteynerleri «root» yetkisi olmadan çalıştırılabilmesidir.** Bu şekilde, yönetici yetkisi olmayan sistemlerde de konteynerler kullanılarak sistem geliştirilebilir. Yazılım geliştirme işlemleri için faydalıdır.
- Podman CLI (Command Line Interface), Docker komut satırı uygulamasının çok benzeridir ve aynı opsiyonları kullanır. Bazı sistem yöneticileri **alias** tanımlayarak **docker** komutunu **podman'a** bağlarlar.

<https://developers.redhat.com/blog/2020/11/19/transitioning-from-docker-to-podman>

<https://developers.redhat.com/blog/2020/09/25/rootless-containers-with-podman-the-basics>

podman nedir?

- Podman'ı 22.04 üzerinde kurabiliriz. 20.04'te kurulmayacaktır.

```
$ sudo apt update
```

```
$ sudo apt install podman
```

- **podman** komutu, **docker** komut satırı programıyla uyumludur (**docker** yerine **podman** kullanabilirsiniz)

```
$ podman -v
```

```
podman version 3.4.4
```

- **podman**'ın en önemli özelliklerinden biri, **root** yetkisi olmadan konteynerleri çalıştırılabilmesidir. Ayrıca, daemon yüklenmesine gerek bulunmamaktadır.

podman

- Podman'ın imajları indireceği yeri belirtmek için **/etc/containers/registries.conf** dosyasına «**sudo nano /etc/containers/registries.conf**» komutuyla aşağıdaki satırları ekleyerek kaydetmek gereklidir.

```
[registries.search]
registries=["registry.access.redhat.com", "registry.fedoraproject.org", "docker.io"]
```

- «**podman pull ubuntu**» diyerek ilk imajımızı çekelim.

```
$ podman pull ubuntu
Resolved "ubuntu" as an alias (/etc/containers/registries.conf.d/shortnames.conf)
Trying to pull docker.io/library/ubuntu:latest...
Getting image source signatures
Copying blob 37aaf24cf781 done
Copying config 3565a89d9e done
Writing manifest to image destination
Storing signatures
3565a89d9e81a4cb4cb2b0d947c7c11227a3f358dc216d19fc54bfd77cd5b542
```

buildah nedir?

- OCI veya Docker uyumlu imaj oluşturmak için kullanılabilen açık kaynak kodlu komut satırı programıdır.
- **buildah** ile aşağıdaki işler yapılabilir:
 - Sıfırdan veya var olan bir imajı kullanarak çalışan bir konteyner oluşturabilir.
 - Çalışan bir konteynerden veya **Dockerfile** içindeki talimatlar kullanılarak bir imaj oluşturabilir.
 - Oluşturulan imajlar geleneksel Docker imajı olabildiği gibi OCI imaj biçiminde de olabilir.
 - Çalışan bir konteynerin **root** dosya sistemi, konak makineye monte (**mount**) edilebilir ve üzerinde istenen değişiklikler yapılabilir. Değiştirilen **root** dosya sistemi kullanılarak farklı bir imaj oluşturulabilir.
 - Lokal bir konteynerin ismini değiştirebilir.

Buildah nasıl kurulur?

- **Podman** ve **Buildah** birbirini tamamlayan yazılımlardır. **Buildah** imajları oluştururken **podman** bunları çalıştırmaktadır. **Buildah** da **podman** gibi, çalıştırılması için sistem yöneticisi yetkisine sahip olunmasına gerek bulunmamaktadır.
- **Buildah**, **podman** API'sını kullanmaktadır.
- **Buildah** ancak Ubuntu v20.10'dan sonraki sürümlere kurulabilmektedir.

```
$ sudo apt-get -y update
```

```
$ sudo apt-get -y install buildah
```
- **Buildah** Red Hat Enterprise Linux v7.4'den sonraki sürümlerde çalışabilmektedir.

Skopeo nedir?

- Docker daemon'unu kullanmadan, imajları doğrudan kopyalama, silme, imzalama ve inceleme/onaylama işlerini yapabilen komut satırı programıdır.
- Skopeo, 2020'de Red Hat tarafından duyurulmuştur.
- En önemli özelliklerinden biri, kullanıcının, Skopeo ile imajları, **sistem yöneticisi yetkilerine sahip olmadan** bir depodan diğerine aktarabilmesidir.
- OCI standartlarında oluşturulmuş olanları veya geleneksel Docker imajlarını kullanabilir.
- **docker.io** kütüphanesinden (image registry) bir imaj lokale kaydedilebilir ve lokalde bulunan bir imaj kullanıcı adı ve şifreyle uzaktaki kütüphaneye aktırılabilir.
- Uzaktaki bir imajı, lokal makinenize çekmeden (**pull**) incelemenize (**inspect**) ve içindeki katmanları görmenizi sağlar.
- Ubuntu üzerinde kurulum için Ubuntu'nun v20.10'dan sonrası gereklidir (diğer bir deyişle Ubuntu 20.04'de çalışmamaktadır)

<https://github.com/containers/skopeo>

«skopeo inspect»

```
$ skopeo inspect docker://registry.fedoraproject.org/fedora:latest
{
  "Name": "registry.fedoraproject.org/fedora",
  "Digest": "sha256:655721ff613ee766a4126cb5e0d5ae81598e1b0c3bcf7017c36c4d72cb092fe9",
  "RepoTags": [
    "24",
    "25",
    "26-modular",
    ...
  ],
  "Created": "2020-04-29T06:48:16Z",
  "DockerVersion": "1.10.1",
  "Labels": {
    "license": "MIT",
    "name": "fedora",
    "vendor": "Fedora Project",
    "version": "32"
  },
  "Architecture": "amd64",
  "Os": "linux",
  "Layers": [
    "sha256:3088721d7dbf674fc0be64cd3cf00c25aab921cacf35fa0e7b1578500a3e1653"
  ],
  "Env": [
    "DISTTAG=f32container",
    "FGC=f32",
    "container=oci"
  ]
}
```


«skopeo inspect --config»

```
$ skopeo inspect --config docker://registry.fedoraproject.org/fedora:latest | jq
{
  "created": "2020-04-29T06:48:16Z",
  "architecture": "amd64",
  "os": "linux",
  "config": {
    "Env": [
      "DISTTAG=f32container",
      "FGC=f32",
      "container=oci"
    ],
    "Cmd": [
      "/bin/bash"
    ],
    "Labels": {
      "license": "MIT",
      "name": "fedora",
      "vendor": "Fedora Project",
      "version": "32"
    }
  },
  "rootfs": {
    "type": "layers",
    "diff_ids": [
      "sha256:a4c0fa2b217d3fd63d51e55a6fd59432e543d499c0df2b1acd48fbe424f2ddd1"
    ]
  },
  "history": [
    {
      "created": "2020-04-29T06:48:16Z",
      "comment": "Created by Image Factory"
    }
  ]
}
```

JSON dosyalarını
okunabilir hale
getirir.

Docker Swarm

- Docker stack is a collection of services that make up an application in a specific environment
- Docker Container is a runtime instance of Docker Image. Docker Service can run one type of Docker Images on various containers locating on different nodes to perform the same functionality. Docker Stack consists of multiple Docker Services.
- <https://betterprogramming.pub/how-to-differentiate-between-docker-images-containers-stacks-machine-nodes-and-swarms-fd5f7e34eb9f>
- <https://nickjanetakis.com/blog/docker-tip-23-docker-compose-vs-docker-stack>

Docker Stack