FLASK RESTPLUS APİ GELİŞTİRME

Mustafa CİN

İÇİNDEKİLER

DOCKER

Docker, yazılım geliştiriciler ve sistemciler için geliştirilen açık kaynaklı bir sanallaştırma platformudur. Docker ile Linux, Windows ve MacOSX üzerinde Linux ve Windows sanal konteynırlar (makineler) çalıştırabiliriz. Bu platform sayesinde web sistemlerinin kurulumunu, testini ve dağıtımını kolaylıkla gerçekleştirilebiliriz. Docker'ın kullanım alanı çok geniştir. Sadece website veya sanal makine için değil veri tabanı sistemi veya birçok uygulama da docker üzerinde çalıştırılabilinir.Geliştirilen yazılım projelerinde docker kullanmak da oldukça avantaj sağlayacaktır.

Docker ile sanal konteynırlar oluşturur.Oluşturulan her konteynırın imajı da olur.O imaj sayesinde başka sistemlerde aynı konteynır çalıştırılır. Docker ile geliştirilen uygulamalara port atayarak istenilen port'ta çalışması sağlanabilir. Geliştirmiş olduğumuz uygulamanın farklı ortamlarda sorunsuz çalışmasını istiyorsak docker oldukça işimize yarayacaktır.Kullanıcı dostu ve düşük ram tüketimi sayesinde büyük firmalar da docker kullanmaya başlamıştır.

Docker Container ve Docker Image

Docker, LXC sanallaştırma mekanizması üzerine kuruludur. Bir Docker imajı, container denilen birimlerde çalıştırılıyor. Oyunculara rol vermek gibi düşünebiliriz. Her bir container bir süreç (process) kullanıyor. Bir makinede gücüne bağlı olarak binlerce docker konteynırı birden çalışabiliyor. Konteynır imajları ortak olan sistem dosyalarını paylaşıyorlar. Dolayısıyla disk alanından tasarruf ediliyor. Ancak klasik sanal makine sistemlerinde (VMWare, VirtualBox vs) her bir uygulama için ayrı işletim sistemi ve kütüphane dosyaları ayrılmak zorunda.

Docker Hub

Docker Hub, oluşturulan docker imajlarının yüklenip ortak kullanıma açıldığı bir repodur (depo). Public ve Private olmak üzere iki tür repo vardır. Public olarak yayınlanan imajlar herkes tarafında görülebilmekte ve kullanılabilmektedir. Örnek vericek olursak sürekli güncel sürümü yayınlanan Wordpress imajını indirip kolayca kullanabilir, kurulum sıkıntısı olmaz. Popüler birçok uygulamanın (mysql, apache, nginx, ghost, vs) hazırda ücretsiz imajları mevcut.

Docker Avantajları

- Az RAM tüketir ve oldukça hızlıdır.
- Paket sorunları vermez.Her türlü ortamda çalıştırılır.
- Sanal makinelerden web sunucuya kadar oldukça geniş bir alanı vardır.
- Belgelerimizi imaj olarak yükleyerek farklı ortamlarda da ulaşabiliriz.

Docker Komutları

docker start <CONTAINER ID> =>konteynır başlatır.

docker stop <CONTAINER ID> =>Konteynır durdurur

docker inspect <container_id> =>Konteynır hakkında detaylı bilgi verir.

docker rm -vf <CONTAINER ID> => Konteynır siler.

docker pause <container_id> => Konteynırı duraklatır.

docker rmi $(docker images -aq) => Bütün imajları siler.

docker –version => Docker versiyonu belirtir.

docker search postgresql => postgresql arar.

docker pull postgres => En son güncellemeyi indirir.

sudo docker run --name postgres -e POSTGRES_PASSWORD=123456 -d -p 5432:5432 -v $HOME/satleca/docker/postgresql/data:/var/lib/postgresql/data postgres

=>Postgresql'i çalıştırır.

sudo docker exec -it postgres psql -U postgres =>Docker üzerinden Postgresql'e bağlanır.

sudo docker exec -it postgres bash => Sanal makinede postgresql çalıştırır.

sudo docker run -it ubuntu bash =>Sanal ubuntu çalıştırır.

```
sudo docker run -p 9000:80 --name pgadmin4  \
    -e "PGADMIN_DEFAULT_EMAIL=admin" \
    -e "PGADMIN_DEFAULT_PASSWORD=0" \
    -d dpage/pgadmin4
```

=>PGadmin kurmak.

docker ps =>Çalışan imajları gösterir.

docker images =>Sistemde var olan imajları listeler.

docker logs <container_id> =>İlgili Container'ın terminalinde o ana kadar oluşan çıktıyı gösterir

Scrum Nedir? Scrum Rolleri Nelerdir?

Scrum, kompleks yazılım süreçlerinin yönetilmesi için kullanılır. Bunu yaparken bütünü parçalayan; tekrara dayalı bir yöntem izler. Düzenli geri bildirim ve planlamalarla hedefe ulaşmayı sağlar. Bu anlamda ihtiyaca yönelik ve esnek bir yapısı vardır. Müşteri ihtiyacına göre şekillendiği için müşterinin geri bildirimine göre yapılanmayı sağlar. İletişim ve takım çalışması çok önemlidir. 3 temel prensip üzerine kurulmuştur;

- Şeffaflık; Projenin ilerleyişi, sorunlar,gelişmeler herkes tarafından görülebilir olmalıdır.
- Denetleme; Projenin ilerleyişi düzenli olarak kontrol edilir.
- Uyarlama; Proje, yapılabilecek değişikliklere uyum sağlayabilmelidir.

Roller

Pig Roller; Scrum sürecine dahil olanlar yani projede asıl işi yapan kişilerdir. Bunlar Scrum Master, Product Owner, Geliştirme Takımı'dır.

1) Product Owner; Geliştirme takımı ve müşteri arasındaki iletişimi sağlar. Projenin özelliklerini tanımlar. Sprint'i iptal yetkisine sahiptir. Sprint neden iptal edilmek istenebilir? Hızla değişen ortamlarda bir sprint'e alınan işlerin iş birimi için önemi kalmamış olabilir ya da sprint'e alınan işlerden daha önemli işler ortaya çıkabilir. İş sahibi bunu görüp sprint'i iptal etmek isteyebilir.

2) **Scrum Master**; Scrum kurallarını, teorilerini ve pratiklerini iyi bilir ve takımın bu kurallarını uygulamasından sorumlu kişidir. Takımın yöneticisi değildir. Takımı rahatsız eden, verimli çalışmalarını engelleyen durumları ortadan kaldırır.

3) Geliştirme Takımı; Bir Sprint'e alınan bütün işleri tamamlayacak özelliklere sahip kişilerdir. Kişilerin tek bir görevi yoktur, çapraz görev dağılımı yaparlar, herkes her şeyi yapabilir konumdadır. 5–7 kişi arasında değişir. Projenin geliştirilmesi ile ilgili sorumluluk geliştirme takımına aittir.

Chicken Roller; Scrum'ın işleyişinde aktif olarak yer almayan kişilerdir. Müşteriler, satıcılar gibi.

Postgresql Veri Tabanı

PostgreSQL, veri tabanları için ilişkisel modeli kullanan ve SQL standart sorgu dilini destekleyen bir veri tabanı yönetim sistemidir.Postgresql , aynı zamanda iyi performans veren, güvenli ve geniş özellikleri olan ücretsiz ve açık kaynak kodlu bir veri tabanıdır. Güvenlik alanı üzerine birçok başarılı yönü vardır.PostgreSQL bu bağlamda veri tabanının hızlı olmasından ziyade güvenli olmasına önem vermiştir.Mysql veri tabanıyla karşılaştırıldığında daha yavaş olmasına karşın çok daha güvenli olması da PostgreSQL veri tabanına olan eğilimi arttırmıştır.

PostgreSQL veri tabanı 1996'dan bu yana bağımsız şekilde ve sadece gönüllülerin çabalarıyla geliştirilmiştir. Hiçbir kuruma bağlı olmadan geliştirilen veri tabanı, özgür yazılım felsefesinin en önemli ürünlerinde biri haline gelmiştir.

Temel SQL Komutları

\l+ => veritabanlarını listeler.

\c template => template isimli veri tabanına giriş yapar.

\d =>Veri tabanındaki tabloları listeler.

create database mydb; => mydb isminde veri tabanı oluşturur.

create user admin with encrypted password 'sifre'; =>admin adında kullanıcı oluşturur.Parola olarak sifre atandı.

grant all privileges on database mydb to admin; =>Admin kullanıcısını mydb veri tabanında yetki sahibi yapar.

CREATE TABLE sinif (id serial PRIMARY KEY , isim CHAR(30), pass char(20));

=>sinifi isminde veri tabanı oluşturur.Id sürekli artarak devam eder.Veri tabanına kelime olarak 30 karakter sınırında isim ve 20 karakter sınırında pass şeklinde değerler girilebilir.

INSERT INTO sinif (id, isim, pass) VALUES (1, 'mustafa', 'sifrem123');

=>Oluşturulan sinif ismli tabloya id , isim ve sifre atar.

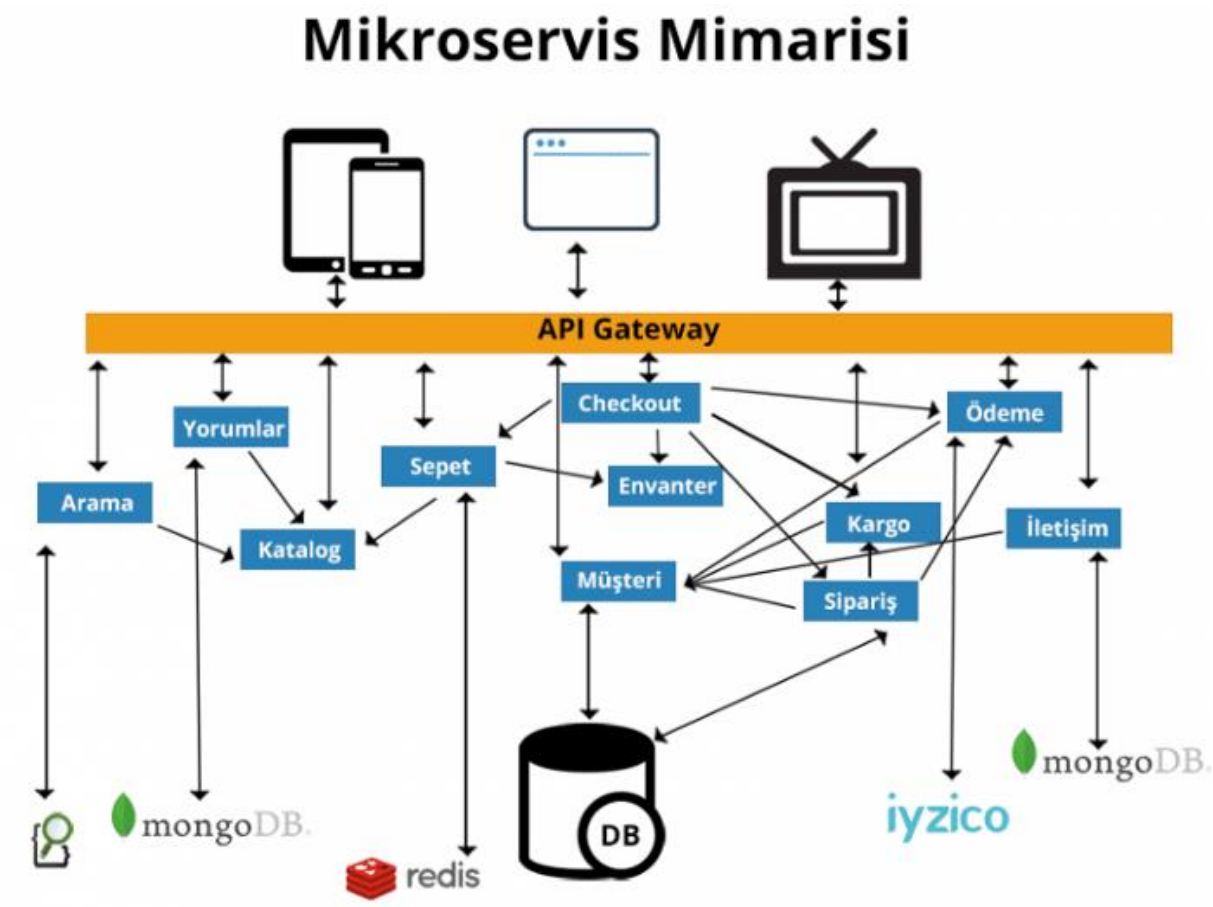
SELECT * FROM sinif; =>"sinif" adındaki tablonun içindeki verileri listeler.

DROP TABLE sinif; =>Oluşturulan "sinif" isimli tabloyu siler.

Mikroservis Mimarisi

Mikroservis en kısa tabiriyle , küçük , otonom ve bir arada çalışan servislerdir. Bütün bir sistemin, her biri bağımsız olarak çalışan ve açık protokoller vasıtasıyla birbiri ile iletişim kuran küçük servislere ayrılmasıdır. Monolitik sisteme bir alternatif olarak geliştirilmiştir.Gelişmiş yazılım projeleri ve sistemlerde her bir sistemi kendi içinde denetlenebilir şekilde kılarak yazılım ekiplerine oldukça avantajlar sağlamaktadır.

Mikroservisler birbirinden bağımsız ve tek bir işe odaklanmış uygulamalar olduklarından, her bir servisi farklı bir programlama dili ile geliştirmek mümkün. Bu da uygulamanın bir programlama diline olan bağımlılığını ortadan kaldırıyor. Mikroservislerin en büyük faydası mimarinin en baştan kurulmaya çalışmaması, ürün geliştikçe mimarinin gelişmesidir. Ekibe sonradan katılan bir kişinin bu kadar büyük bir yapıya, mimariyi öğrenmesi yerine, bu kişinin hangi dilde ve hangi DB ortamında işi yapmasını biliyor ise bu ortamda bu ufak iş mantığını geliştirme imkanı vermesidir. Aynı zamanda bu atomik yapıların sırasını ve hiyerarşini değiştirme imkanı vermesi, uygulamanın daha esnek olmasını sağlar.



FLASK RESTPLUS

Flask-RESTPlus, hızlı bir şekilde REST API'leri oluşturmak için destek ekleyen bir Flask uzantısıdır. Flask, Python işlevlerinin API'ler olarak gösterilmesini sağlar . Flask-RESTPlus, minimum kurulumla en iyi uygulamaları teşvik eder. API'nizi tanımlamak ve belgelerini doğru şekilde ortaya çıkarmak için Swagger UI desteği sağlar.

Flask rest_plus indirme:

pip install flask

pip install flask-restplus

FLASK RestPLUS İle Servis Geliştirilmesi (GET,POST,PUT)

Flask uygulaması için import etme:

```
from flask import Flask
from flask_restplus import Api, Resource, fields
from werkzeug.contrib.fixers import ProxyFix
```

Flask uygulamasını tanımlamak ve alınacak değerleri belirmek:

```
app = Flask(__name__)
app.wsgi_app = ProxyFix(app.wsgi_app)
api = Api(app, version='1.0', title='TodoMVC API',
          description='A simple TodoMVC API',
          )

ns = api.namespace('todos', description='TODO operations')
```

Api'de kullanıcıdan istenilen değerlerin istenmesi.Tarayıcıda Model sekmesinde bulunur.

```
todo = api.model('Todo', {
    'id': fields.Integer(readonly=True, description='The task unique identifier'),
    'name': fields.String(required=True, description='The task details'),
    'surname': fields.String(required=True, description='The task details'),
    'age': fields.Integer(required=True, description='The task details')
})
```

GET,PUT ve POST isteklerinin işleme gireceği ana sınıf.Daha aşağıda her istek bu sınıf içerisinde düzenlenir.

```
class TodoDAO(object):
    def __init__(self):
        self.counter = 0
        self.todos = []

    def get(self, id):
        for todo in self.todos:
            if todo['id'] == id:
                return todo
        api.abort(404, "Todo {} doesn't exist".format(id))

    def create(self, data):
        todo = data
        todo['id'] = self.counter = self.counter + 1
        self.todos.append(todo)
        return todo

    def update(self, id, data):
        todo = self.get(id)
        todo.update(data)
        return todo

DAO = TodoDAO()
```

GET İsteği:

```
@ns.route('/')
class Liste(Resource):
    '''Post isteginde bulunur ve verileri listeler.'''
    @ns.doc('Veriler listesi')
    @ns.marshal_list_with(veri)
    def get(self):
        '''Taslaklari listeler.'''
        return TAM.veriler
```

POST isteği:

```
    @ns.doc('create_veri')
    @ns.expect(veri)
    @ns.marshal_with(veri, code=201)
    def post(self):
        '''Create a new task'''
        return TAM.create(api.payload), 201
```

PUT isteği:

```
@ns.route('/<int:id>')
@ns.response(404, 'Todo not found')
@ns.param('id', 'The task identifier')
class Veri(Resource):

    @ns.expect(veri)
    @ns.marshal_with(veri)
    def put(self, id):
        '''Verileri günceller. '''
        return TAM.update(id, api.payload)
```

Uygulamayı çalıştıran ana main fonksiyonu:

```
if __name__ == '__main__':
    app.run(debug=True)
```

Flask RestPLUS Api Kullanarak SQLAlchemy ile PostgreSQL Veritabanı Yönetimi

Python kütüphanesi olan SQLAlchemy kütüphanesi kullanılarak postgresql veritabanı yönetilebilir. Flask RestPLUS kullanıldığı için bu uygalamada flask restplus içerisinde postgresql veritabanı bağlantısı yapılacaktır. Bağlantıyı yapmak için sqlalchemy kütüphanesi kullanılacaktır.

SQLAlchemy kütüphanesini import etme:

```
from sqlalchemy import create_engine, Table, Column, Integer, String, MetaData
from flask_sqlalchemy import SQLAlchemy
```

Bağlanılacak veritabanını belirleme:

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = "postgresql://postgres:toor@localhost:5433/postgres"

db = SQLAlchemy(app)
```

Veritabanında oluşturulacak tabloyu ve o tabloya eklenecek kolonları belirler.Örneğin alttaki şekilde oluşturulan tabloda id isminde int sıra numarası var.Her değer arttıkça id değişecektir.İsim ve soyisim ise self char olarak maksimum 200 karakter olarak belirlendi.Yaş kolonu ise int olarak kullanıcı tarafından girilen değerler arasındadır.:

```
class users_api(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(200))
    surname = db.Column(db.String(200))
    age = db.Column(db.Integer())

    def __init__(self, name, surname, age):
        self.name = name
        self.surname = surname
        self.age = age
```

Post isteğinde veritabanına değerler ekleme.Kullanıcı name,surname ve age değerlerini JSON formatında göndermelidir.İd değeri sistem tarafından atanır. “**db.session.create_all()”** fonksiyonu kullanılarak veritabanına yeni bir kayıt eklenmiş olur.:

```
def post(self):
    '''Create a new task'''
    new_user = users_api(api.payload['name'], api.payload['surname'], api.payload['age'])
    db.session.add(new_user)
    db.create_all()
    db.session.commit()

    return DAO.create(api.payload), 201
```

Delete şeklinde veritabanındaki bir veriyi silmek.Alttaki kod örneğinde id isteniyor.İstenilen id değeri **db.session.delete()** fonksiyonu kullanılarak siliniyor:

```
def delete(self, id):
    '''Delete a task given its identifier'''
    DAO.delete(id)
    sil = users_api.query.filter_by(id=id).first()
    db.session.delete(sil)
    db.session.commit()
    return '', 204
```

Put methodu ile veritabanında değerler değiştirilebilinir. İlk olarak id değeri girilir. Girilen id değerine ait name, surname ve age değerleri isteğe bağlı olarak güncellenebilinir.

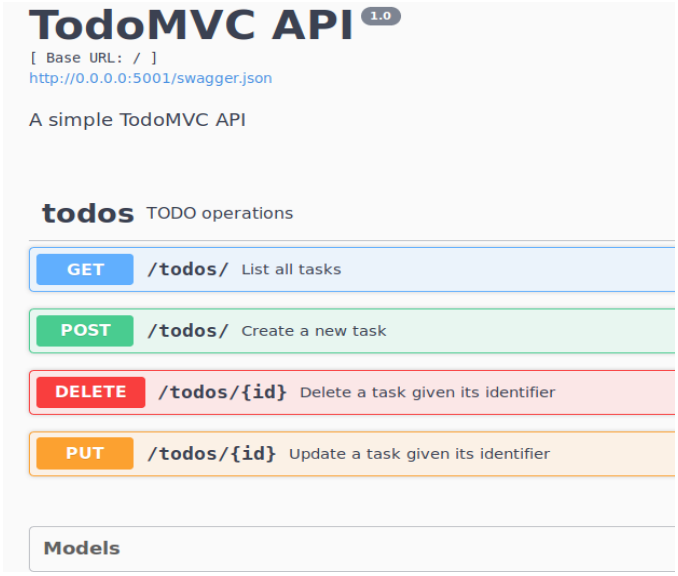
```
def put(self, id):
    '''Update a task given its identifier'''

    data = users_api.query.filter_by(id=id).first()
    data.name = api.payload['name']
    data.surname = api.payload['surname']
    data.age = api.payload['age']
    db.session.commit()

    return DAO.update(id, api.payload)
```

Uygulamanın Çalıştırılması ve Testi

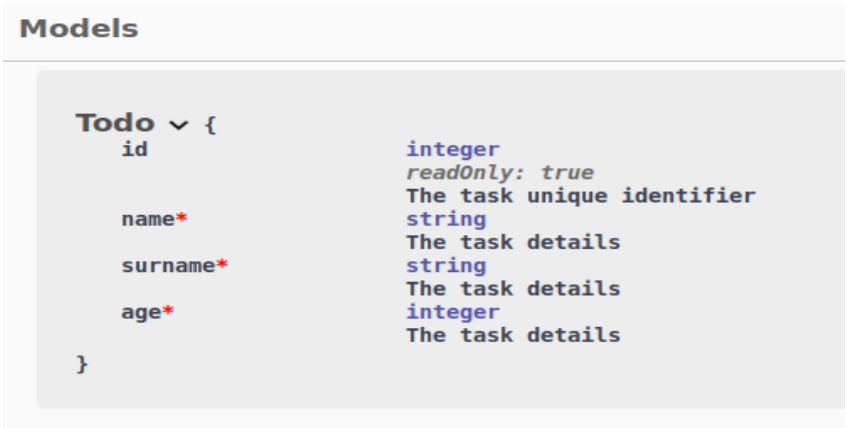
Uygulamanın çalıştırılması:

```
satleca@kali:~/proje/web$ python3 app.py
app.py:12: DeprecationWarning: 'werkzeug.contrib.fixers.ProxyFix' has moved t
port is deprecated as of version 0.15 and will be removed in 1.0.
  app.wsgi_app = ProxyFix(app.wsgi_app)
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deplo
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
 * Restarting with stat
/home/satleca/proje/web/app.py:12: DeprecationWarning: 'werkzeug.contrib.fixe
y_fix.ProxyFix'. This import is deprecated as of version 0.15 and will be rem
  app.wsgi_app = ProxyFix(app.wsgi_app)
 * Debugger is active!
 * Debugger PIN: 217-105-042
```

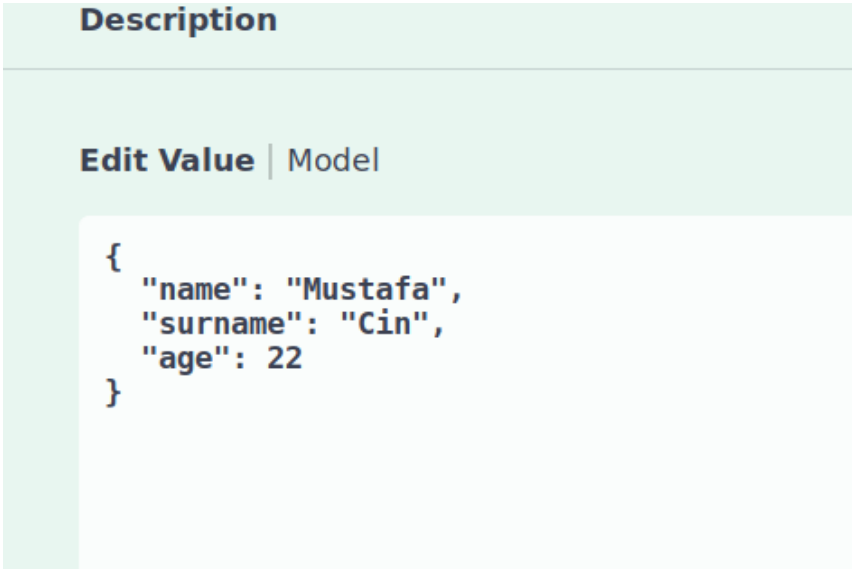
Uygulamayı web tarayıcıda çalıştırma :



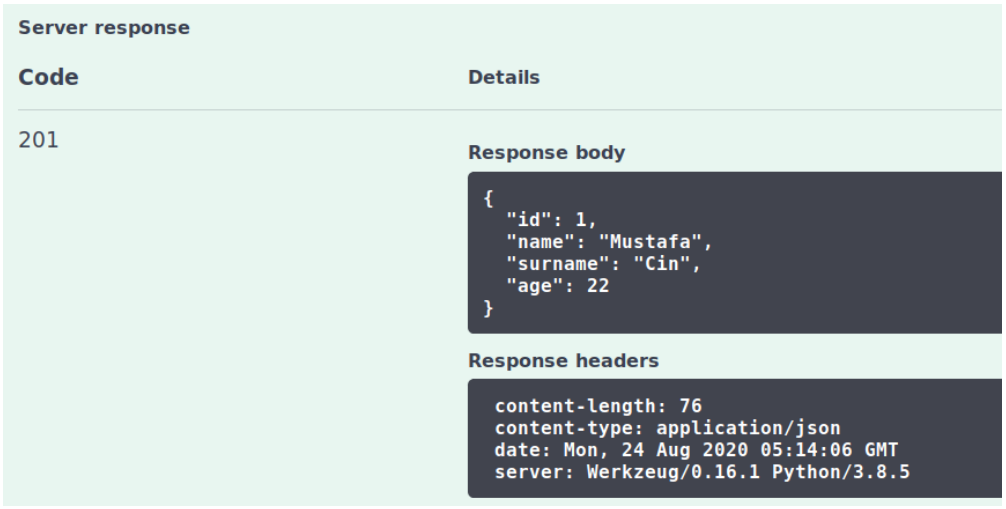
Models (Yapılması gereken işlemleri anlatan kısım) :



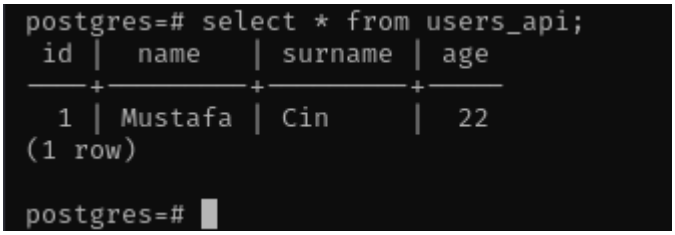
Json formatında POST isteğinde bulunma :



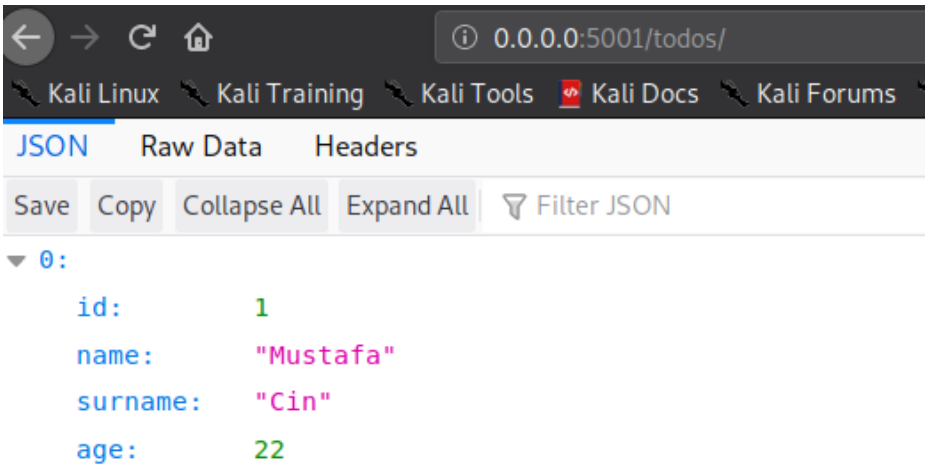
Server yanıtı :



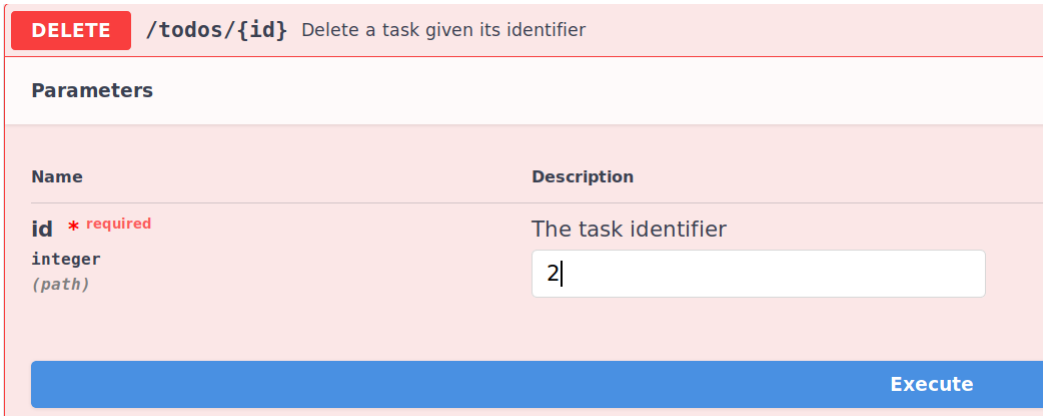
Veritabanındaki değişiklik :



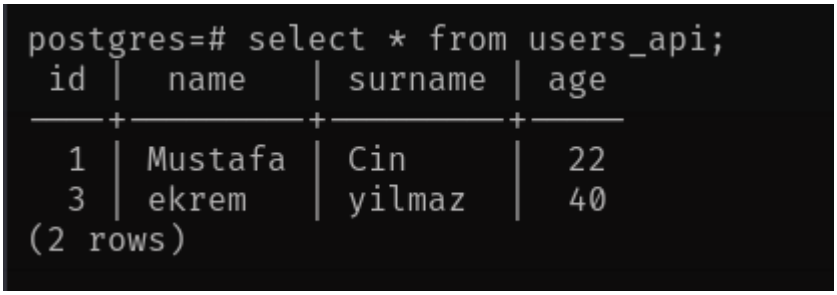
Uygulama her bir kayıtı kendi dizininde tutar :



Kayıt silmek için silinecek kayıtın id değerinin girilmesi gerekmektedir :

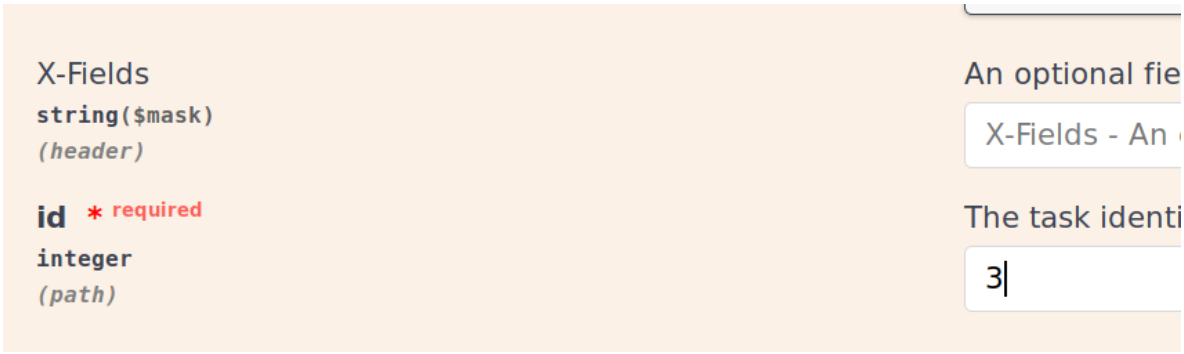


İd değeri 2 olan kayıt hem veritabanından hem de websitesinden silindi :



Kayıtlarda değişiklik yapmak için PUT methodu kullanılır. İlk olarak eklemek istediğimi kayıt yazılır ardından id değeri girilir:

```
{
  "name": "erol",
  "surname": "özkan",
  "age": 28
}
```



Veritabanındaki değişiklik:

```
postgres=# select * from users_api;
 id |  name   | surname | age
----+---------+---------+-----
  1 | Mustafa | Cin     |  22
  3 | erol    | özkan   |  28
(2 rows)
```

Yapılan tüm işlemler uygulama tarafından kayıt altına alınmaktadır :

```
127.0.0.1 - - [24/Aug/2020 08:11:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:57] "GET /swaggerui/swagger-ui.css HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:57] "GET /swaggerui/droid-sans.css HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:57] "GET /swaggerui/swagger-ui-bundle.js HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:57] "GET /swaggerui/swagger-ui-standalone-preset.js HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:58] "GET /swaggerui/favicon-16x16.png HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:11:58] "GET /swagger.json HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:14:06] "POST /todos/ HTTP/1.1" 201 -
127.0.0.1 - - [24/Aug/2020 08:14:40] "GET /todos/ HTTP/1.1" 200 -
127.0.0.1 - - [24/Aug/2020 08:15:57] "POST /todos/ HTTP/1.1" 201 -
127.0.0.1 - - [24/Aug/2020 08:16:16] "POST /todos/ HTTP/1.1" 201 -
127.0.0.1 - - [24/Aug/2020 08:16:45] "DELETE /todos/2 HTTP/1.1" 204 -
127.0.0.1 - - [24/Aug/2020 08:16:58] "DELETE /todos/2 HTTP/1.1" 404 -
127.0.0.1 - - [24/Aug/2020 08:18:50] "PUT /todos/3 HTTP/1.1" 200 -
```

Uygulamayı Docker Üzerinde Çalıştırma

Docker konteynır yapısı sayesinde birçok servisi çalıştırabilmektedir. Aynı zamanda bireysel kullanıcıların ve yazılım ekiplerinin geliştirmiş olduğu uygulamaları da çalıştırabilmektedir. Docker'ın avantajlarından ilk bölümde bahsedilmişti. Docker Build ile herhangi bir uygulamanın docker imajı alınır. Bu imaj istenildiği zaman istenildiği port'ta çalıştırılır. Bu imaj Docker Hub'a push edilerek başka bilgisayarlarda da çalıştırılması sağlanır.

Docker imajını almak için ilk önce requirements.txt dosyasını üretmek gerekmektedir. Çünkü docker sanal bir konteynırdır. Bu konteynırda hali hazırda kütüphaneler indirilmiş değildir. Bu uygulamada python üzerinden işlem yapılacağı için kullanılacak olan python kütüphanelerini uygun kütüphane sürümleriyle birlikte yazılmalıdır.

Geliştirilen uygulama için requirements.txt :

```
satleca@kali:~/proje/web$ cat requirements.txt
Flask==1.1.1
flask-restplus==0.13.0
flask-sqlalchemy==2.4.0
sqlalchemy==1.3.18
werkzeug==0.16.0
psycopg2-binary==2.8.5
```

Docker imajı oluşturmak için Dockerfile kullanılmalıdır. Dockerfile dosyası docker konteynırının yapması gerek işlemleri belirler.

```
satleca@kali:~/proje/web$ cat Dockerfile
FROM ubuntu:18.04

RUN apt-get update -y && \
    apt-get install -y python3-pip python-dev

COPY . /app
WORKDIR /app

RUN pip3 install --upgrade pip
RUN pip3 install -r requirements.txt

EXPOSE 5001

ENTRYPOINT ["python3"]
CMD ["app.py"]
```

Docker İmajının oluşturulması:

```
satleca@kali:~/proje/web$ source venv/bin/activate
(venv) satleca@kali:~/proje/web$ sudo docker build --tag proje .
[sudo] password for satleca:
Sending build context to Docker daemon  7.699MB
Step 1/9 : FROM ubuntu:18.04
 ---> 6526a1858e5d
Step 2/9 : RUN apt-get update -y &&     apt-get install -y python3-pip python-dev
 ---> Using cache
 ---> 8375a3842862
Step 3/9 : COPY . /app
 ---> Using cache
 ---> 7db2672af7c0
Step 4/9 : WORKDIR /app
 ---> Using cache
 ---> 3058db340795
Step 5/9 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> 63da8a045946
```

```
Successfully built c0084d49fb61
Successfully tagged proje:latest
```

Docker imajının çalıştırılması :

```
(venv) satleca@kali:~/proje/web$ sudo docker run --name proje -p 5001:5001 proje
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 719-962-666
```

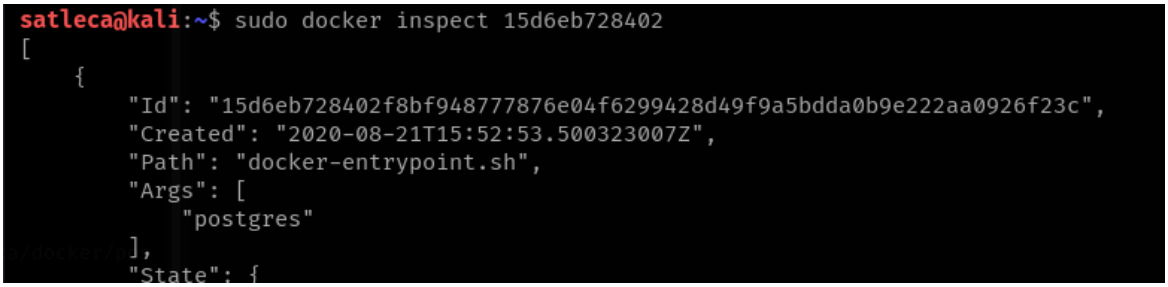
Geliştirilen docker üzerinde çalıştı. Girilen değerler postgresql veritabanına yazılması gerekmektedir. Şimdi docker üzerinde postgresql veritabanı çalıştırmalı ve girilen değerleri postgresql veritabanına eklemeliyiz.

Docker üzerinde postgresql veritabanı çalıştırma:

```
satleca@kali:~$ sudo docker exec -it postgres psql -U postgres
Error response from daemon: Container 15d6eb728402f8bf948777876e04f6299428d49f9a5bdda0b9e22
is not running
satleca@kali:~$ sudo docker start 15d6eb728402f8bf948777876e04f6299428d49f9a5bdda0b9e222aa0
15d6eb728402f8bf948777876e04f6299428d49f9a5bdda0b9e222aa0926f23c
satleca@kali:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
            PORTS                    NAMES
38026a361254        proje               "python3 app.py"         34 minutes ago      Up 34
minutes     0.0.0.0:5001->5001/tcp   proje
15d6eb728402        postgres            "docker-entrypoint.s…"   43 hours ago        Up 21
seconds     0.0.0.0:5432->5432/tcp   postgres
```

Docker flask restplus üzerinde geliştirilen uygulama ile postgresql veritabanı arasında bağlantı yapmak için ip adresini doğru girmek gereklidir.

sudo docker ps komutunu yazarak çalışan konteynırlar görüntülendi. Postgres konteynırının ip adresini tespit etmek için:



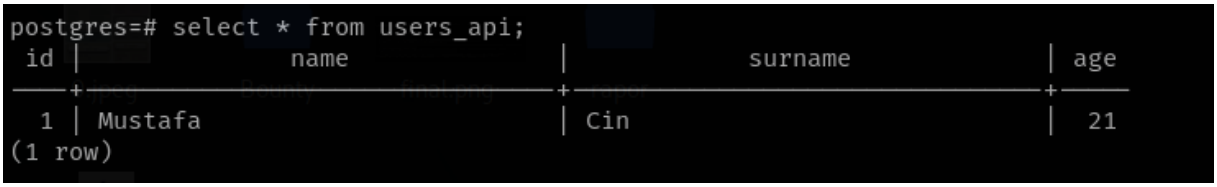
Docker inspect, girilien konteynır id'ye ait detaylı bilgiler verir.Bu bilgilere konteynır ip adresi de dahil.

İp adresini app.py dosyasında doğru şekilde girdikten sonra uygulama çalıştırılır ve restplus api üzerinden yapılan veritabanı işlemleri docker üzerindeki postgresql veritabanı üzerinde yapılır.

Değerleri girmek:

Yanıtımız:

Veritabanındaki "**users_api**" tablosu:



Docker-Compose İle Mikroservisler Çalıştırma

Docker Compose, docker uygulamalarının tanınmasını ve çalıştırılmasını sağlayan bir docker aracıdır. Geliştirilen birçok ve bir mikroservisin tek bir çatı altında çalıştırılmasını sağlar. Kullanım kolaylığı açısından uygulama geliştiricilerin oldukça kullandığı bir araçtır.

Docker Compose kullanmak için **docker-compose.yml** dosyasına ihtiyacımız var. Bu dosya Docker Compose için konfigürasyon dosyası görevini üstlenir. Docker Compose bu konfigürasyon dosyası içindekileri baz alarak çalışmaktadır.

Docker compose indirme:

Konfigürasyon dosyasını yazarken ilk olarak versiyon bilgisi verilir. Ardından her mikroservis için ayrı ayrı özellikler belirtilir.

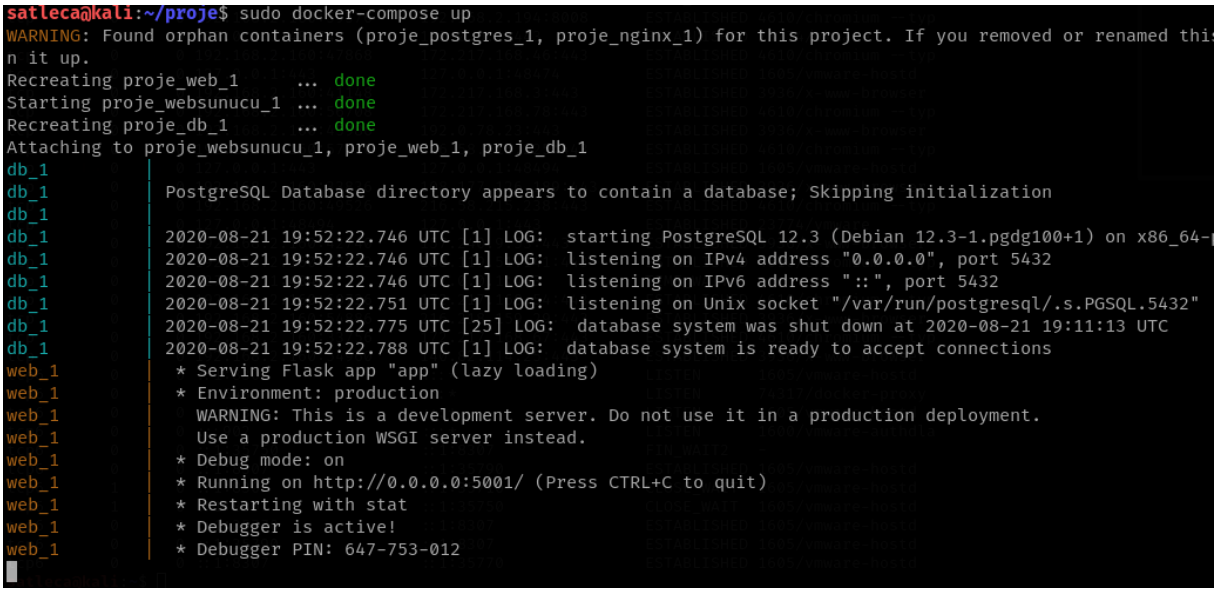
docker-compose.yml dosyası:

```
satleca@kali:~/proje$ cat docker-compose.yml
version: '3.8'

services:
    web:
        build: ./web
        ports:
          - "5001:5001"

    db:
        image: postgres:latest
        volumes:
          - /var/lib/postgresql/data
        ports:
          - "5432:5432"
        environment:
          - "POSTGRES_HOST_AUTH_METHOD=trust"

    websunucu:
        image: nginx:latest

        ports:
          - "80:80"
```

Docker Compose build ile imaj alma:

```
satleca@kali:~/proje$ sudo docker-compose build
[sudo] password for satleca:
db uses an image, skipping
websunucu uses an image, skipping
Building web
Step 1/9 : FROM ubuntu:18.04
 ---> 6526a1858e5d
Step 2/9 : RUN apt-get update -y &&     apt-get install -y python3-pip python-dev
 ---> Using cache
 ---> 8375a3842862
Step 3/9 : COPY . /app
 ---> cf9c9fa61ab4
Step 4/9 : WORKDIR /app
 ---> Running in 71e5efab0734
```

Docker Compose ile imajı çalıştırma:

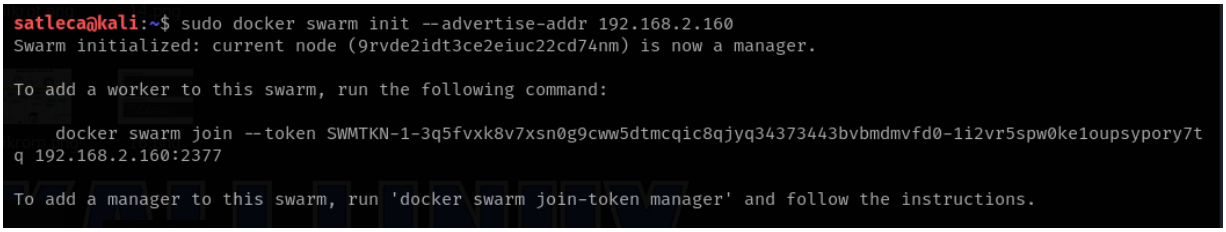


Docker Compose ile postgresql veritabanı, flask restplus uygulaması ve nginx web sunucusu tek bir çatı altında çalıştırılmış oldu. “**sudo docker-compose stop**” yazarak uygulamayı durdurabiliriz. Mikroservislerin birlikte çalıştırılması gerektiği zaman Docker Compose bize oldukça kolaylıklar sağlamaktadır. Tek komutla çalıştırıp tek komutla durdurmak gibi diyebiliriz.

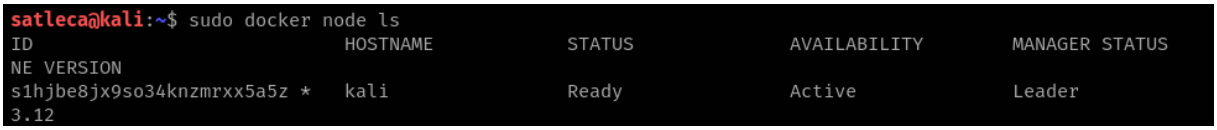
Uygulamayı Docker Swarm İle Çalıştırma

Docker Swarm, Docker Compose’ye yapı ve amaç olarak benzerlikleri bulunur. Swarm farklı olarak düğüm yapısı kullanır. Her bir imaj için bir düğüm denirse bu düğümlerin birbirleriyle iletişimini sağlar.

Docker swarm çalıştırmak için ilk olarak bağlanmak gerekir. Bağlantı da token sayesinde olur. Token almak ve bağlanmak için :



Swarm’a bağlandıktan sonra düğüm oluşur:



Docker Swarm üzerinde uygulamayı çalıştırma :

```
satleca@kali:~/proje$ sudo docker stack deploy --compose-file docker-compose.yml stack
Creating network stack_default
Creating service stack_web
Creating service stack_db
Creating service stack_websunucu
```

Docker Stack kullanarak swarm düğümümüz üzerinde uygulamayı çalıştırdık. Bu işlemi sorunsuz yapabilmek için **docker-compose.yml** dosyasında ufak bir değişiklik yapmamız gerekiyor. Build olarak **./web** dizini kullanılmıştı. Onun yerine ilk olarak oluşturmuş olduğumuz proje isimli imajı eklemek gerekiyor.

```
services:
    web:
        image: proje:latest
        ports:
          - "5001:5001"
```

Docker ps komutu ile imajlara bakılması:

```
satleca@kali:~/proje$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
b7ae556a3f6d        nginx:latest        "/docker-entrypoint.…"   26 seconds ago
7c5d7ddf9d50        postgres:latest     "docker-entrypoint.s…"   30 seconds ago
54ca28fe0cdd        proje:latest        "python3 app.py"         36 seconds ago
```

```
  PORTS               NAMES
  80/tcp              stack_websunucu.1.0gzv9lkzzl0c1ocqx4s1ngzcr
  5432/tcp            stack_db.1.33zwics97s32s5kukqbqwuo6b
  5001/tcp            stack_web.1.ihbidvxqiwjmny3083986tj7x
```

Uygulamanın kaynak kodları:

https://github.com/mustafacin/RestPLUS-PostgreSQL-Docker-Swarm