Görev1

Hizmeti Dağıtma ve Doğrulama Talimatları

1)Gereksinimler

Linux işletim sistemi

Python 3, django ve venv modülü kurulu olmalı.

Uygulamanın bağımlılıkları (requirements.txt) eksiksiz olmalı.

2)Uygulama Kurulumu

Sanal ortam oluşturulması;

python3 -m venv venv

source venv/bin/activate

Bağımlıkların yüklenmesi;

requirements.txt dosyasını kullanarak;

pip install -r requirements.txt

Uygulamanın çalışıp, çalışmadığının kontrol edilmesi;

python manage.py runserver 0.0.0.0:8000

3)Systemd Hizmetinin **Dağıtılması**

Systemd Birim Dosyasının Oluşması;

[Unit]

Description=My Django Application

After=network.target

[Service]

User=busra

WorkingDirectory=/mnt/c/Users/busra.arslan/Desktop/deutsch/deutsch_lernen

ExecStart=/mnt/c/Users/busra.arslan/Desktop/deutsch/deutsch_lernen/venv/bin/python manage.py runserver 0.0.0.0:8000

Restart=always

StandardOutput=append:/mnt/c/Users/busra.arslan/Desktop/deutsch/deutsch_lernen/myapp_output.log

StandardError=append:/mnt/c/Users/busra.arslan/Desktop/deutsch/deutsch_lernen/myapp_error.log

[Install]

WantedBy=multi-user.target

Hizmet dosyasını systemd'ye tanıtmak için;

sudo systemctl daemon-reload

Hizmeti çalıştırmak için;

sudo systemctl start myapp.service

Hizmeti sistem açılışında otomatik başlatmak için;

sudo systemctl enable myapp.service

4)Hizmeti **Doğrulama**

Hizmetin Durumunu Kontrol Edin;

sudo systemctl status myapp.service

Uygulamayı Test Edin;

http://localhost:8000 adresinin erişilebilir olması

Görev 2

**1)Uygulamanın Konteynerleştirilmesi**

Dockerfile kullanılarak Django tabanlı deutsch_lernen uygulaması bir Docker imajına dönüştürüldü.

Projenin kök dizininde yer alan Dockerfile şu şekilde yapılandırıldı:

```
FROM python:3.8-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Docker imajı şu komutla oluşturuldu:

```
docker build -t deutsch_lernen_app:latest .
```

2)Docker Compose ile Yönetim

docker-compose.yml dosyası oluşturularak, uygulamanın NGINX ters proxy ile yönetilmesi ve yük dengelemesi sağlandı.

docker-compose.yml dosyasının içeriği:

```
version: '3.8'

services:
  app:
    image: deutsch_lernen_app:latest
    ports:
      - "8000:8000"
```

```
  volumes:
    - .:/app
  environment:
    - DJANGO_SETTINGS_MODULE=deutsch_lernen.settings
  deploy:
    replicas: 2
    restart_policy:
      condition: on-failure

nginx:
  image: nginx:latest
  ports:
    - "8080:80"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - app
```

3)NGINX ile Ters Proxy ve Trafik Yönlendirme

nginx.conf dosyası, gelen HTTP isteklerini Django uygulamasına yönlendirecek şekilde ayarlandı.

nginx.conf dosyasının içeriği:

```
events { }

http {
  upstream app_servers {
      server deutsch_lernen-app-1:8000;
      server deutsch_lernen-app-2:8000;
  }

  server {
      listen 80;
```

```
        location / {
            proxy_pass http://app_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

4)Docker Swarm ile Replika **Yapılandırması**

2 replika ile yüksek kullanılabilirlik sağlamak için Docker Swarm modunda çalışıldı.

Swarm başlatıldı:

```
docker swarm init
```

Uygulama Swarm modunda dağıtıldı:

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

Çalışan hizmetler kontrol edildi:

```
docker service ls
```

Test ve Doğrulama

Portlar ve erişim yolları:

- Django: http://localhost:8000
- NGINX: http://localhost:8080

Replikaları doğrulama:

```
docker service ps deutsch_lernen_app
```

Görev 3

Kubernetes ile Uygulama Dağıtımı ve Erişim Yapılandırması

1. Docker ile **Uygulamanın Hazırlanması**

- Uygulama için bir Docker imajı oluşturuldu.
- İmaj adı: deutsch_lernen:latest
- kind kullanılarak Kubernetes cluster'ına yüklendi:

```
kind load docker-image deutsch_lernen:latest
```

2. Kubernetes Deployment ve Service **Yapılandırması**

- Deployment oluşturuldu:
  - Uygulamanın iki pod çalıştırması için replicas: 2 ayarlandı.
  - deployment.yml dosyası aşağıdaki şekilde yapılandırıldı:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: deutsch-lernen-deployment
spec:
 replicas: 2
 selector:
  matchLabels:
   app: deutsch-lernen
 template:
  metadata:
   labels:
    app: deutsch-lernen
  spec:
   containers:
   - name: deutsch-lernen-app
    image: deutsch_lernen:latest
```

```
    ports:
    - containerPort: 8000
```

Service yapılandırıldı:

- LoadBalancer tipi kullanıldı.
- service.yml dosyası aşağıdaki şekilde yapılandırıldı:

```
apiVersion: v1
kind: Service
metadata:
  name: deutsch-lernen-service
spec:
  selector:
    app: deutsch-lernen
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
  type: LoadBalancer
```

3. MetalLB ile LoadBalancer **Yapılandırması**

- MetalLB, LoadBalancer tipi için bir ağ IP havuzu sağladı:
    - MetalLB için ConfigMap oluşturuldu:

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
```

```
- name: default
  protocol: layer2
  addresses:
  - 172.24.236.240-172.24.236.250
```

Uygulandı:

kubectl apply -f config.yml

4. Port Yönlendirme ve **Erişim**

- Uygulamayı tarayıcıdan test etmek için WSL ve Windows arasında port yönlendirme yapıldı:
  - Port yönlendirme:

netsh interface portproxy add v4tov4 listenport=80 connectport=80 connectaddress=172.24.224.1

New-NetFirewallRule -DisplayName "Allow Port 80" -Direction Inbound -Protocol TCP -LocalPort 80 -Action Allow

5. Sorun Giderme ve Testler

Podlar, servis ve MetalLB durumları kontrol edildi:

Podlar:

kubectl get pods

Servisler:

kubectl get services

MetalLB logları:

kubectl logs -n metallb-system -l component=speaker

Uygulamayı tarayıcıda test etmek için:

kubectl port-forward service/deutsch-lernen-service 8080:80

Ardından tarayıcıda http://127.0.0.1:8080 adresine girildi.

Görev 4

myapp.service adlı bir systemd servis dosyası yüklendi, ancak servis doğru şekilde çalışmamaktadır. Servis başlatmayı denendiğinde, şu hata mesajını alınıyor:

Job for myapp.service failed because the control process exited with error code. See "systemctl status myapp.service" and "journalctl -xe" for details.

Sorun Tespiti:

Servis durumunu kontrol ettim:

sudo systemctl status myapp.service

Çıktıda şu hatayı fark ettim:

ExecStart=/usr/local/bin/example-app: No such file or directory

Ayrıca journalctl çıktısında, çalışma dizininin (WorkingDirectory) mevcut olmadığını gördüm.

Hata **Kaynağı:**

- ExecStart kısmında belirtilen uygulama dosyası yanlış bir yolu işaret ediyordu.
- WorkingDirectory tanımlanmadığı veya mevcut bir dizini işaret etmediği için servis başlatılamıyordu.

Çözüm:

- myapp.service dosyasını düzenledim ve ExecStart ile WorkingDirectory bölümlerini doğru yollarla değiştirdim:

[Service]

ExecStart=/usr/bin/python3 /home/user/app/manage.py runserver

WorkingDirectory=/home/user/app

Yaptığım değişikliklerden sonra şu komutları çalıştırdım:

sudo systemctl daemon-reload

sudo systemctl restart myapp.service

sudo systemctl enable myapp.service

**Doğrulama:**

- Servis başarılı bir şekilde başlatıldı:

sudo systemctl status myapp.service

Çıktı:

Active: active (running) since Thu 2025-01-09 12:00:00 UTC; 5s ago

Final Servis **Dosyası:**

myapp.service dosyasının düzeltilmiş hali:

[Unit]

Description=My App Service

After=network.target

[Service]

ExecStart=/usr/bin/python3 /home/user/app/manage.py runserver 0.0.0.0:8000

Restart=always

WorkingDirectory=/home/user/app

User=appuser

Environment="ENV=production"

StandardOutput=journal

StandardError=journal

[Install]

WantedBy=multi-user.target

Servis dosyasındaki hatalar düzeltilmiş ve servis başarıyla çalışır duruma getirilmiştir. Bu süreçte ExecStart ve WorkingDirectory gibi kritik parametrelerin doğru yapılandırıldığından emin olunmuştur.