Uygulamalı DevOps Projesi: E-Ticaret Web Uygulaması (Detaylı Anlatım)

Proje Genel Bakış

* Hedef: Bir e-ticaret sitesini (ürün listeleme, sepet ekleme) Python Flask ile geliştirip, DevOps araçlarıyla otomatik test, kalite kontrol ve dağıtım yapmak.
* Araçlar: Git (GitHub), Jenkins, Docker, Ansible, SonarQube, Pytest, Selenium, Prometheus, Grafana, Slack.
* Ortam: Yerel bir bilgisayar veya AWS EC2 (örnekte yerel kullanılacak, ama AWS için notlar eklenecek).
* Süre: Her adım için tahmini süre verilecek.

Adım 1: Proje Planlama ve Kod Yönetimi

Amaç: Kodun sürüm kontrolünü ve ekip iş birliğini sağlamak.  
Süre: ~1 saat

Detaylı Uygulama: (Python yüklü olmalı)

1. Git ve GitHub Kurulumu:
   * Git’in yüklü olduğundan emin olun: git --version (yoksa sudo apt install git ile kurun - Linux için).
   * GitHub’da bir hesap açın ve yeni bir depo oluşturun: e-commerce-app (public veya private).
2. Yerel Depo Oluşturma:
   * Bir klasör oluşturun ve başlatın:

bash

mkdir e-commerce-app

cd e-commerce-app

git init

* + GitHub ile bağlayın:

bash

git remote add origin https://github.com/kullanici-adi/e-commerce-app.git

1. Basit Bir Flask Uygulaması Yazma:
   * Gerekli dosyaları oluşturun:

bash

touch app.py requirements.txt templates/index.html

* + requirements.txt:

text

Flask==2.0.1

* + app.py:

python

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

products = [{'id': 1, 'name': 'Telefon', 'price': 1000}, {'id': 2, 'name': 'Laptop', 'price': 5000}]

@app.route('/')

def home():

return render\_template('index.html', products=products)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

* + templates/index.html:

html

<!DOCTYPE html>

<html>

<body>

<h1>E-Ticaret Sitesi</h1>

<ul>

{% for product in products %}

<li>{{ product.name }} - {{ product.price }} TL</li>

{% endfor %}

</ul>

</body>

</html>

1. Kodun GitHub’a Yüklenmesi:
   * Değişiklikleri ekleyin ve push edin:

bash

git add .

git commit -m "İlk Flask uygulaması"

git push origin main (veya git push origin HEAD:master)

* + Not: İlk push için GitHub’da bir main branch oluşturmanız gerekebilir (web arayüzünden).

1. Branch ile Çalışma:
   * Yeni bir özellik için branch açın:

bash

git checkout -b feature/add-cart

Olası Sorun: GitHub’a push yaparken認証 hatası alırsanız, bir PAT (Personal Access Token) oluşturup kullanın.  
Kalite Katkısı: Kodun izlenebilirliği ve ekip incelemesi sağlanır.

Adım 2: Kod Yazımı ve Birim Testleri

Amaç: Sepet işlevini ekleyip test etmek.  
Süre: ~1.5 saat

Detaylı Uygulama:

1. Sepet Ekleme İşlevi:
   * app.py’yi güncelleyin:

python

cart = []

@app.route('/add/<int:product\_id>')

def add\_to\_cart(product\_id):

product = next((p for p in products if p['id'] == product\_id), None)

if product:

cart.append(product)

return f"{product['name']} sepete eklendi!"

return "Ürün bulunamadı."

1. Birim Testleri Yazma:
   * Test bağımlılıklarını ekleyin: requirements.txt’ye pytest==7.1.2 ekleyin.
   * Test dosyası oluşturun: tests/test\_app.py:

python

import pytest

from app import app

@pytest.fixture

def client():

app.config['TESTING'] = True

with app.test\_client() as client:

yield client

def test\_home\_page(client):

response = client.get('/')

assert response.status\_code == 200

assert b'Telefon' in response.data

def test\_add\_to\_cart(client):

response = client.get('/add/1')

assert response.status\_code == 200

assert b'Telefon sepete eklendi' in response.data

* + Testleri çalıştırın:

bash

pip install -r requirements.txt

pytest tests/

1. Kodun GitHub’a Yüklenmesi:
   * Değişiklikleri commit edin:

bash

git add .

git commit -m "Sepet ekleme ve testler eklendi"

git push origin feature/add-cart

* + GitHub’da bir PR açın ve main’e birleştirin.

Olası Sorun: Test çalışmazsa, Flask’in doğru yüklendiğinden emin olun (pip list).  
Kalite Katkısı: Fonksiyonel doğruluk test edilir.

Adım 3: Sürekli Entegrasyon (CI) Kurulumu

Amaç: Kod değişikliklerini otomatik test etmek.  
Süre: ~2 saat

Detaylı Uygulama:

1. Jenkins Kurulumu:
   * Docker ile Jenkins’i kurun: (docker yüklü olmalı)

bash

docker run -d -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts

* + Tarayıcıda http://localhost:8080’a gidin, ilk şifreyi alın (docker logs <container-id>).

1. Jenkins Yapılandırması:
   * Jenkins’e giriş yapın ve "New Item" > "Pipeline" seçin.
   * Pipeline’ı GitHub ile bağlayın (depo URL’sini ekleyin).
2. Jenkins Pipeline Dosyası:
   * Jenkinsfile’ı kök dizine ekleyin:

groovy

pipeline {

agent any

stages {

stage('Build') {

steps {

sh 'python3 -m venv venv'

sh '. venv/bin/activate && pip install -r requirements.txt'

}

}

stage('Test') {

steps {

sh '. venv/bin/activate && pytest tests/'

}

}

}

}

* + GitHub’a push edin:

bash

git add Jenkinsfile

git commit -m "Jenkins pipeline eklendi"

git push origin main

1. Webhook Ayarı:
   * GitHub’da depo ayarlarından "Webhooks" > "Add webhook" seçin.
   * Payload URL: http://<jenkins-ip>:8080/github-webhook/.
   * Her commit’te Jenkins tetiklensin.

Olası Sorun: Jenkins portu açılmazsa, güvenlik duvarını kontrol edin (sudo ufw allow 8080).  
Kalite Katkısı: Her değişiklik test edilir, hatalar erken yakalanır.

Adım 4: Kod Kalite Analizi

Amaç: Kodun kalitesini denetlemek.  
Süre: ~1.5 saat

Detaylı Uygulama:

1. SonarQube Kurulumu:
   * Docker ile kurun:

bash

docker run -d -p 9000:9000 sonarqube:lts-community

* + http://localhost:9000’a gidin, kullanıcı: admin, şifre: admin (ilk girişte değiştirin).

1. Proje Tanımlama:
   * SonarQube’da "Create Project" > e-commerce-app adıyla bir proje oluşturun.
   * Token oluşturun (örneğin: abc123).
2. Jenkins Entegrasyonu:
   * Jenkins’e SonarQube eklentisi kurun (Manage Plugins).
   * Jenkinsfile’a analiz ekleyin:

groovy

stage('SonarQube Analysis') {

steps {

withSonarQubeEnv('SonarQube') {

sh '. venv/bin/activate && sonar-scanner -Dsonar.projectKey=e-commerce-app -Dsonar.host.url=http://localhost:9000 -Dsonar.login=abc123'

}

}

}

* + sonar-project.properties ekleyin:

properties

sonar.projectKey=e-commerce-app

sonar.sources=.

sonar.exclusions=venv/\*\*, tests/\*\*

1. Analizi Çalıştırma:
   * Commit yapın ve Jenkins’in taramayı çalıştırmasını izleyin.
   * SonarQube’da raporu kontrol edin (örneğin, kod duplicasyonu).

Olası Sorun: SonarQube bağlanmazsa, Jenkins’te SonarQube sunucusunu doğru tanımlayın.  
Kalite Katkısı: Kod okunabilirliği ve güvenlik artar.

Adım 5: Konteynerizasyon

Amaç: Uygulamayı konteynerize etmek.  
Süre: ~1 saat

Detaylı Uygulama:

1. Dockerfile Yazma:
   * Kök dizine Dockerfile ekleyin:

dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

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

1. Docker İmajı Oluşturma ve Test:
   * İmajı oluşturun:

bash

docker build -t e-commerce-app:latest .

* + Çalıştırın:

bash

docker run -p 5000:5000 e-commerce-app:latest

* + Tarayıcıda http://localhost:5000’u kontrol edin.

1. Docker Hub’a Yükleme:
   * Docker Hub’da oturum açın: docker login.
   * İmajı etiketleyin ve push edin:

bash

docker tag e-commerce-app:latest kullanici-adi/e-commerce-app:latest

docker push kullanici-adi/e-commerce-app:latest

Olası Sorun: Port çakışması olursa, başka bir port deneyin (-p 5001:5000).  
Kalite Katkısı: Ortam tutarlılığı sağlanır.

Adım 6: Sürekli Dağıtım (CD)

Amaç: Uygulamayı otomatik dağıtmak.  
Süre: ~1.5 saat

Detaylı Uygulama:

1. Dağıtım Hedefi Belirleme:
   * Yerel bir Docker konteyneri veya AWS EC2 kullanılabilir. Yerel örnek:

bash

docker run -d -p 5000:5000 --name e-commerce-app kullanici-adi/e-commerce-app:latest

1. Jenkins CD Entegrasyonu:
   * Jenkinsfile’a dağıtım ekleyin:

groovy

stage('Deploy') {

steps {

sh 'docker stop e-commerce-app || true'

sh 'docker rm e-commerce-app || true'

sh 'docker run -d -p 5000:5000 --name e-commerce-app kullanici-adi/e-commerce-app:latest'

}

}

* + Commit yapın ve Jenkins’in dağıtımı çalıştırmasını bekleyin.

Olası Sorun: Docker izin hatası alırsanız, sudo kullanın veya kullanıcıyı docker grubuna ekleyin (sudo usermod -aG docker $USER).  
Kalite Katkısı: Hızlı ve güvenilir dağıtım.

Adım 7: Test Otomasyonu

Amaç: Kullanıcı arayüzünü test etmek.  
Süre: ~2 saat

Detaylı Uygulama:

1. Selenium Testi Yazma:
   * requirements.txt’ye selenium==4.5.0 ekleyin.
   * tests/test\_ui.py:

python

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

import time

def test\_add\_to\_cart():

options = Options()

options.headless = True

driver = webdriver.Chrome(options=options)

driver.get('http://localhost:5000')

driver.find\_element\_by\_xpath('//a[contains(@href, "/add/1")]').click()

time.sleep(1)

assert "sepete eklendi" in driver.page\_source

driver.quit()

* + HTML’e bağlantı ekleyin (index.html):

html

<li>{{ product.name }} - {{ product.price }} TL <a href="/add/{{ product.id }}">Sepete Ekle</a></li>

1. Jenkins’e Entegrasyon:
   * Jenkinsfile’a ekleyin:

groovy

stage('UI Test') {

steps {

sh '. venv/bin/activate && python tests/test\_ui.py'

}

}

* + ChromeDriver kurun: sudo apt install chromium-chromedriver.

Olası Sorun: Selenium çalışmazsa, ChromeDriver sürümünü kontrol edin.  
Kalite Katkısı: Kullanılabilirlik doğrulanır.

Adım 8: İzleme ve Geri Bildirim

Amaç: Performansı izlemek.  
Süre: ~2 saat

Detaylı Uygulama:

1. Prometheus Entegrasyonu:
   * requirements.txt’ye prometheus-client==0.11.0 ekleyin.
   * app.py’ye metrik ekleyin:

python

from prometheus\_client import Counter, generate\_latest

requests\_total = Counter('requests\_total', 'Total HTTP requests')

@app.route('/metrics')

def metrics():

requests\_total.inc()

return generate\_latest()

* + Prometheus’u kurun:

bash

docker run -d -p 9090:9090 -v $(pwd)/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus

* + prometheus.yml:

yaml

scrape\_configs:

- job\_name: 'ecommerce'

static\_configs:

- targets: ['host.docker.internal:5000']

1. Grafana Kurulumu:
   * Kurun: docker run -d -p 3000:3000 grafana/grafana.
   * http://localhost:3000’a gidin (kullanıcı/şifre: admin/admin).
   * Veri kaynağı olarak Prometheus’u ekleyin (http://localhost:9090).
   * Bir dashboard oluşturun: requests\_total metriğini görselleştirin.

Olası Sorun: host.docker.internal çalışmazsa, IP adresini manuel girin.  
Kalite Katkısı: Performans izlenir.

Adım 9: İletişim ve Bildirim

Amaç: Ekip farkındalığını artırmak.  
Süre: ~30 dakika

Detaylı Uygulama:

1. Slack Entegrasyonu:
   * Slack’te bir kanal oluşturun: #devops.
   * Jenkins’e "Slack Notification" eklentisi kurun.
   * Jenkinsfile’a ekleyin:

groovy

post {

failure {

slackSend channel: '#devops', message: "Build failed: ${env.JOB\_NAME} #${env.BUILD\_NUMBER}"

}

}

* + Slack’ten webhook URL’sini Jenkins’e ekleyin.

Olası Sorun: Bildirim gelmezse, webhook URL’sini kontrol edin.  
Kalite Katkısı: Hızlı hata çözümü.

Sonuç ve Değerlendirme

* Çıktılar: Çalışan bir e-ticaret sitesi, otomatik pipeline, izleme dashboard’u.
* Metrikler:
  + Hata oranı: %5’in altında.
  + Teslim süresi: < 10 dakika.
  + Performans: Yanıt süresi < 2 saniye.
* Rapor: Her adımın kaliteye katkısını yazın.