UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Pregatiri pentru deploy si productie
Proiect ASO 3

Student: Alexuc Tudor
Grupa: 30642
Îndrumător de laborator: Andrei Bogdan Leucuta

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Cerințele rezolvate

In aceasta etapa presupunem ca site-ul nostru este funct, ional si il pregatim de lansare ın piata. Pentru un deploy cat mai portabil vom utiliza containere Docker.
• (1p) Instalații Docker si creați un Dockerfile pentru aplicația voastra .
• (1p) In aplicatia Django, modificati setarile astfel incat baza de date sa nu mai fie un fisier SQLite, ci o baza de date reala, Postgres sau MySQL. Serverul bazei de date va rula si el ıntr-un container Docker separat. Cele doua servicii (site-ul web si baza de date) vor fi definite ıntr-un fișier docker-compose.yml si vor rula folosind docker-compose.
• (1p) Pana acum, aplicat, ia voastra a rulat ın mod debug, folosind server-ul implicit de la Django. In continuare, vom folosi un server WSGI real, numit gunicorn si un reverse proxy (puteti alege nginx sau apache2) care fie va redirecta request-urile catre site-ul web, fie va servi direct fișierele statice. Creati fisiere docker-compose.dev.yml pentru development/debug (ın care site-ul ruleaza ca si pana acum, cu server-ul de la Django) si dockercompose.prod.yml pentru product, ie, ın care reverse proxy-ul ruleaza într-un container separat.
• (1p) Site-ul nostru trebuie sa poata rula s, i ın mod development s, i ın mod production. Din acest motiv, ın settings.py nu vom mai avea toate setarile hardcodate, ci unele dintre ele vor fi citite din variabile de mediu. Vom pastra variabilele de mediu separat in fis, iere .env.dev s, i .env.prod, care vor fi utilizate de docker-compose pentru development, respectiv productie.
• (1p bonus) Site-ul nostru trebuie sa fie accesibil si prin HTTPS. Pentru aceasta etapa, va trebui sa generati un certificat self-signed (nu puteți obtine unul semnat de o autoritate, decat daca detineti un domeniu) si sa configurati reverse proxy-ul sa il foloseasca. Puteti urma acest tutorial.

Modul de rezolvare

Am urmărit tutorialul pentru instalarea si utilizarea Docker. Am generat requirements.txt si am rulat comenzile de creare a imaginii si a containerului.
De asemenea, pentru setarea unei baze de date in locul fisierului sqlite3 am urmarit pasii de la sectiunea de PostgreSQL de la acelasi tutorial.
https://testdriven.io/blog/dockerizing-django-with-postgres-gunicorn-and-nginx/

Din moment ce aplicatia a fost realizata pe un sistem Windows a trebuit sa inlocuiesc gunicorn cu waitress. Pentru setup-ul necesar am urmarit acest tutorial:
https://www.youtube.com/watch?v=BBKq6H9Rm5g
De asemenea am creat fisiere noi pentru productie (.env.prod & docker-compose.prod.yml). Diferentele dintre docker-ul de productie si cel de development sunt folosirea librariei waitress cu nginx pentru a rula server-ul fata de serverul default Django si rularea acestuia in mod de productie (DEBUG = 0).

```
# pull official base image
FROM python:3.9.6-alpine

# set work directory
WORKDIR ./ASO_Django_Project

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install dependencies
RUN pip3 install --upgrade pip
COPY ./requirements.txt .
RUN pip3 install -r requirements.txt

# copy project
COPY . .
```

Dockerfile pentru docker simplu

```
version: '3.8'

services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./app/:/usr/src/app/
    ports:
      - 8000:8000
    env_file:
      - ./.env.dev
```

docker-compose.yml pentru docker simplu

```
# pull official base imagedb
FROM python:3.9.6-alpine

# set work directory
WORKDIR ./ASO_Django_Project

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install psycopg2 dependencies
RUN apk update \
    && apk add postgresql-dev gcc python3-dev musl-dev

# install dependencies
RUN pip install --upgrade pip
COPY ./requirements.txt .
RUN pip install -r requirements.txt

# copy project
COPY . .
```

Dockerfile pentru containere separate pentru server si DB

```yaml
version: '3.8'

services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./app/:/usr/src/app/
    ports:
      - 8000:8000
    env_file:
      - .env.dev
  db:
    image: postgres:13.0-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=postgres

volumes:
  postgres_data:
```

docker-compose.dev.yml

```yaml
version: '3.8'

services:
  web:
    build: .
    command: waitress-serve --listen=*:8000 project.wsgi:application
    volumes:
      - ./app/:/usr/src/app/
    ports:
      - 8000:8000
    env_file:
      - .env.prod
  db:
    image: postgres:13.0-alpine
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=postgres

volumes:
  postgres_data:
```

docker-compose.prod.yml

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Probleme întâlnite și modul de rezolvare

Pentru setarea Docker-ului nu am intampinat nici o problema, tutorialul oferit fiind foarte folositor. In schimb, pentru setarea unei baze de date in locul fisierului db.sqlite3 din Django au fost cateva mici probleme pe care le-am rezolvat prin schimbarea unor variabile de environment.

Dupa ce am intampinat probleme cu gunicorn si am cautat rezolvarea acestora am realizat ca nu este utilizabil pe un sistem care ruleaza Windows asa ca am cautat alternative si am ajuns la Waitress. Principala problema intampinata in incercarea de folosire a acestei librarii a fost cauzata de setarea variabilelor de environment la pasii anteriori. Din moment ce un run al aplicatie din terminal nu preia fisierele de variabile acestea veneau toate “None”. Dupa ce am realizat de la ce am aceasta problema, am hardcodat inapoi variabilele in settings.py pentru a testa functionarea Waitress si dupa le-am schimbat si am rulat Docker-ul.

Mod de utilizare al site-ului

Cont admin:
- Username: adminuser
- Password: adminpassword

Cont simplu creat cu flow-ul de inregistrare implementat:
- Username: simpleuser
- Password: simplepassword

Cel mai usor mod de a testa functionalitatea e prin a deschide o pagina normala de browser si o pagina incognito deoarece nu poti fi logat pe mai multe conturi in cadrul aceluiasi browser.
Apoi de pe ambele ferestre se intra pe aceeasi camera si se scrie mesaje.
De asemenea, se pot crea doua conturi noi in cazul in care se doreste sa se testeze aplicatia cu tot cu flow-ul de autentificare.

Link Github:
https://github.com/DMyrm/ASO-Django-Chat-App
Branch : `project-3`

Concluzii

Urmarind tutorialul si cautand solutii pentru Windows pentru diversele probleme intampinate am invatat cum sa introduc procesele de WSGI si reverse proxy intr-o aplicatie atat Django dar si in alte tipuri de aplicatii. Din moment ce HTTPS a fost un punct bonus si am mai facut procesul de transformare din http in https intr-o aplicatie Django intr-un proiect de anul trecut, nu am mai considerat necesara parcurgerea acestui punct dar am urmarit tutorialul oferit.