Универзитет „Св. Кирил и Методиј" во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО
Supported by A1

Континуирана интеграција и испорака - CI/CD

Наслов - Title

"Movie Shop" Application

Ментори/Mentors:
проф. д-р Милош Јовановиќ
проф. д-р Панче Рибарски

Студент/Student:
Бранко Георгиев (213077)

Table of Contents

Introduction

My project for the CI/CD course is based on a “Movie Shop” FastAPI-based web application designed to manage a digital movie store. It provides basic CRUD functionalities (Create, Read, Update, Delete).

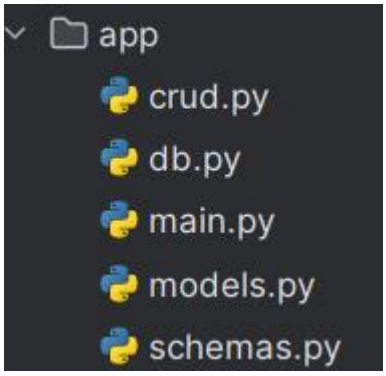
Key features:

- **FastAPI:** A high-performance web framework for seamless API development.
- **PostgreSQL:** A robust and scalable relational database for data storage.
- **Docker:** The application is containerized using Docker.
- **Kubernetes:** The application is orchestrated with Kubernetes for scalability and deployment.
- **ArgoCD:** Ensures continuous deployment and GitOps-driven Kubernetes management.

Technologies and apps

FastAPI Application

The application contains the following python (.py) files:



crud.py: Contains the basic CRUD operations for interaction with database

crud.py

```python
from fastapi import HTTPException
from sqlalchemy.orm import Session
from . import models, schemas


1 usage  Branko Georgiev
def create_movie(db: Session, schema: schemas.MovieSchema):
    movie = models.Movie(**schema.dict())
    db.add(movie)
    db.commit()
    db.refresh(movie)
    return movie


3 usages  Branko Georgiev
def read_movie(db: Session, movie_id: int):
    movie = db.query(models.Movie).filter(models.Movie.id == movie_id).first()
    if not movie:
        raise HTTPException(status_code=404, detail="Movie not found!")
    return movie

```

```python
1 usage  Branko Georgiev
def update_movie(db: Session, movie_id: int, schema: schemas.MovieSchema):
    movie = read_movie(db, movie_id)

    if movie:
        for key, value in schema.dict().items():
            setattr(movie, key, value)

        db.commit()
        db.refresh(movie)
    return movie


1 usage  Branko Georgiev
def delete_movie(db: Session, movie_id: int):
    movie = read_movie(db, movie_id)

    db.delete(movie)
    db.commit()

    return movie

1 usage  Branko Georgiev
def list_all_movies(db: Session):
    return db.query(models.Movie).all()
```

db.py: Database connection setup and session management

```python
import os
from dotenv import load_dotenv
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

load_dotenv()

POSTGRES_USER = os.getenv("POSTGRES_USER")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD")
POSTGRES_DB = os.getenv("POSTGRES_DB")

SQLALCHEMY_DATABASE_URL = os.getenv("SQLALCHEMY_DATABASE_URL")

if SQLALCHEMY_DATABASE_URL is None:
    raise RuntimeError("SQLALCHEMY_DATABASE_URL env variable is missing!")

engine = create_engine(SQLALCHEMY_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

main.py: Initializes and runs the FastAPI application

```python
from fastapi import FastAPI, Depends, HTTPException
import time
from sqlalchemy.orm import Session
from sqlalchemy.exc import OperationalError
from app.db import SessionLocal, engine
from app.models import Base
from app import schemas, crud

app = FastAPI()

for i in range(5):
    try:
        Base.metadata.create_all(bind=engine)
        break
    except OperationalError:
        time.sleep(i + 1)


5 usages  Branko Georgiev
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

```python
Branko Georgiev
@app.delete( path: "/movies/{movie_id}", response_model=schemas.MovieSchema)
def delete_movie(movie_id: int, db: Session = Depends(get_db)):
    db_item = crud.delete_movie(db=db, movie_id=movie_id)
    if db_item is None:
        raise HTTPException(status_code=404, detail="Movie not found!")
    return db_item


Branko Georgiev
@app.get( path: "/movies/", response_model=list[schemas.MovieSchema])
def list_all_movies(db: Session = Depends(get_db)):
    return crud.list_all_movies(db=db)
```

Pull Requests

```python
Branko Georgiev
@app.post( path: "/movies/", response_model=schemas.MovieSchema)
def create_movie(movie: schemas.MovieSchema, db: Session = Depends(get_db)):
    return crud.create_movie(db=db, schema=movie)


Branko Georgiev
@app.get( path: "/movies/{movie_id}", response_model=schemas.MovieSchema)
def read_movie(movie_id: int, db: Session = Depends(get_db)):
    db_item = crud.read_movie(db=db, movie_id=movie_id)
    if db_item is None:
        raise HTTPException(status_code=404, detail="Movie not found!")
    return db_item


Branko Georgiev
@app.put( path: "/movies/{movie_id}", response_model=schemas.MovieSchema)
def update_movie(movie_id: int, movie: schemas.MovieSchema, db: Session = Depends(get_db)):
    db_item = crud.update_movie(db=db, movie_id=movie_id, schema=movie)
    if db_item is None:
        raise HTTPException(status_code=404, detail="Movie not found!")
    return db_item
```

models.py: Contains the database models that define the structure of the data tables

```python
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

3 usages  Branko Georgiev
class Base(DeclarativeBase):
    ...

4 usages  Branko Georgiev
class Movie(Base):
    __tablename__ = 'movies'

    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str]
    rating: Mapped[float]
    director: Mapped[str]
    category: Mapped[str]
    duration: Mapped[int]
    description: Mapped[str]
    release_year: Mapped[int]
```

schemas.py: Defines the Schemas that are used for data validation and serialization.

```python
from pydantic import BaseModel

9 usages  Branko Georgiev
class MovieSchema(BaseModel):
    title: str
    rating: float
    director: str
    category: str
    duration: int
    description: str
    release_year: int
```

Dockerfile and docker-compose.yaml

Docker file - This file sets up a FastAPI application using Python 3.10. It defines /app as the working directory, copies dependencies from requirements.txt, and installs them without caching. The entire app is then copied into the container and port 8000 is exposed for external access.

Dockerfile

```
FROM python:3.10

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Docker-compose.yaml - This file defines a FastAPI app with a PostgreSQL database. The **db** service runs on version 15, with credentials set via environment variables and a health check. The **app** is built from the local directory, connects to the database using a network bridge, and only starts after the **db** is healthy.

docker-compose.yml

```yaml
version: '3.8'
services:
  db:
    image: postgres:15
    restart: always
    container_name: db
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5432:5432"
    networks:
      - fastapi-network
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres"]
      interval: 5s
      timeout: 5s
      retries: 5
```

```yaml
  app:
    build: .
    container_name: fastapi-app
    ports:
      - "8000:8000"
    depends_on:
      db:
        condition: service_healthy
    environment:
      DATABASE_URL: postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:5432/${POSTGRES_DB}
    networks:
      - fastapi-network
networks:
  fastapi-network:
    driver: bridge
```

GitHub Actions

The file **linter-and-docker-ci.yml** is placed in the **.github/workflows/** directory in the GitHub repositorium. This pipeline performs two tasks:

1. **Linter Check (Flake8)**
   - Runs on every push to the master branch and on pull requests.
   - Uses Flake8 to check Python code for styleand syntax issues.
   - Runs on **ubuntu-latest.**

2. **Docker Build & Push**
   - Runs only if the linter job passes.
   - Logs into **Docker Hub** using secrets (**DOCKERHUB_USERNAME** & **DOCKERHUB_TOKEN**).
   - Build a **Docker image** for the application and tags it as latest.
   - Pushed the image to **Docker Hub** for deployment.

brankogeorgiev Update linter-and-docker-ci.yml ✓ 220734f · 2 days ago History

Code Blame 47 lines (36 loc) · 1.13 KB Code 55% faster with GitHub Copilot Raw

```yaml
name: Linter and Docker build pipeline

on:
  push:
    branches:
      - master
  pull_request:

jobs:
  linter:
    name: Run Flake8 Linter
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Set Up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.12"

      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install flake8

      - name: Run Flake8 Linter
        run: flake8 . || true

  docker:
    name: Build & Push Docker Image
    runs-on: ubuntu-latest
    needs: linter  # This ensures Docker push runs only if linter passes

    steps:
      - name: Checkout Code
        uses: actions/checkout@v4

      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin

      - name: Build Docker Image
        run: docker build -t ${{ secrets.DOCKERHUB_USERNAME }}/movie-app:latest .

      - name: Push Image to Docker Hub
        run: docker push ${{ secrets.DOCKERHUB_USERNAME }}/movie-app:latest
```

Kubernetes

```
kubernetes
    argo.yaml
    argo_commands.txt
    configmap.yaml
    db-service.yaml
    deployment.yaml
    ingress.yaml
    namespace.yaml
    secret.yaml
    service.yaml
    statefulset.yaml
```

namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: movieapp
```

configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: movieapp
data:
  SQLALCHEMY_DATABASE_URL: "postgresql://postgres:postgresql@db-service:5432/fastapi_db"
```

secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
  namespace: movieapp
type: Opaque
data:
  POSTGRESS_USER: cG9zdGdyZXM=
  POSTGRESS_PASSWORD: cG9zdGdyZXNxbA==
  POSTGRESS_DB: ZmFzdGFwaV9kYg==
```

deployment.yaml

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movie-app
  namespace: movieapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: movie-app
  template:
    metadata:
      labels:
        app: movie-app
    spec:
      containers:
        - name: movie-app
          image: brankogeorgiev/movie-app:latest
          ports:
            - containerPort: 8000
          envFrom:
            - configMapRef:
                name: app-config
            - secretRef:
                name: app-secret
```

service.yaml

service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: movieapp-service
  namespace: movieapp
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 31000
  type: NodePort
  selector:
    app: movie-app
```

db-service.yaml

db-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: db-service
  namespace: movieapp
spec:
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
  selector:
    app: postgres
  type: ClusterIP
```

statefulset.yaml

statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: movieapp
spec:
  serviceName: "postgres-service"
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:latest
          env:
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: POSTGRESS_PASSWORD
            - name: POSTGRES_USER
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: POSTGRESS_USER
            - name: POSTGRES_DB
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: POSTGRESS_DB
          ports:
            - containerPort: 5432
          volumeMounts:
            - name: database-storage
              mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
    - metadata:
        name: database-storage
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 1Gi
        storageClassName: standard
```

ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: movieapp-ingress
  namespace: movieapp
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: movieapp.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: movieapp-service
                port:
                  number: 80
```

argo.yaml

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: myapp-argo-application
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/brankogeorgiev/movie-app
    targetRevision: HEAD
    path: kubernetes
  destination:
    server: https://kubernetes.default.svc
    namespace: movieapp

  syncPolicy:
    syncOptions:
      - CreateNamespace=true
```

Links

GitHub repo: https://github.com/brankogeorgiev/movie-app
DockerHub repo: https://hub.docker.com/r/brankogeorgiev/movie-app