Movie Recommendation System: Software Engineering in AI

# 1. Project Overview

This document outlines the implementation of a containerized movie recommendation system using modern software engineering practices. The project demonstrates the application of DevOps principles to an AI-based recommendation engine, including containerization, orchestration, and cloud deployment strategies.

# 2. System Architecture

## Core Components

- \*\*Frontend\*\*: Streamlit web application  
- \*\*Backend\*\*: Python-based recommendation engine using similarity matrices  
- \*\*Data\*\*: Pre-computed movie similarity data and TMDB API integration

## Infrastructure

- \*\*Containerization\*\*: Docker  
- \*\*Orchestration\*\*: Kubernetes  
- \*\*CI/CD\*\*: GitHub Actions workflow  
- \*\*Cloud Strategy\*\*: AWS EKS (planned)

# 3. Implementation Steps

## 3.1 Application Setup

Created movie recommendation application using Streamlit  
Implemented TMDB API integration with error handling and retry mechanism  
Set up environment variable configuration for API keys

## 3.2 Containerization with Docker

Created Dockerfile for the application:  
```dockerfile  
FROM python:3.9-slim  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY . .  
EXPOSE 8501  
CMD ["streamlit", "run", "app.py", "--server.address=0.0.0.0"]  
```

Built and tested the Docker image locally:  
```bash  
docker build -t movie-recommender:latest .  
docker run -p 8501:8501 movie-recommender  
```

## 3.3 Kubernetes Deployment

Set up local Kubernetes environment using Docker Desktop  
Created deployment manifest (`k8s/deployment.yaml`):  
```yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: movie-recommender  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: movie-recommender  
 template:  
 metadata:  
 labels:  
 app: movie-recommender  
 spec:  
 containers:  
 - name: movie-recommender  
 image: movie-recommender:latest  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 8501  
 env:  
 - name: TMDB\_API\_KEY  
 valueFrom:  
 secretKeyRef:  
 name: movie-api-secrets  
 key: tmdb-api-key  
```

Created service manifest (`k8s/service.yaml`):  
```yaml  
apiVersion: v1  
kind: Service  
metadata:  
 name: movie-recommender  
spec:  
 selector:  
 app: movie-recommender  
 ports:  
 - port: 80  
 targetPort: 8501  
 type: NodePort  
```

Created Kubernetes secret for the TMDB API key:  
```bash  
kubectl create secret generic movie-api-secrets --from-literal=tmdb-api-key=d43f04baefd4a1295e4cc1224b256818  
```

Deployed the application to Kubernetes:  
```bash  
kubectl apply -f k8s/deployment.yaml  
kubectl apply -f k8s/service.yaml  
```

Verified deployment status and exposed the service

## 3.4 CI/CD Pipeline

Created GitHub Actions workflow (`.github/workflows/ci-cd.yml`):  
```yaml  
name: CI/CD Pipeline  
on:  
 push:  
 branches: [ main ]  
 pull\_request:  
 branches: [ main ]  
jobs:  
 build-and-deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v2  
 - name: Set up Python  
 uses: actions/setup-python@v2  
 with:  
 python-version: '3.9'  
 - name: Install dependencies  
 run: |  
 python -m pip install --upgrade pip  
 if [ -f requirements.txt ]; then pip install -r requirements.txt; fi  
 pip install pytest  
 - name: Test with pytest  
 run: pytest  
 - name: Set up Docker Buildx  
 uses: docker/setup-buildx-action@v1  
 - name: Login to DockerHub  
 uses: docker/login-action@v1  
 with:  
 username: ${{ secrets.DOCKER\_HUB\_USERNAME }}  
 password: ${{ secrets.DOCKER\_HUB\_TOKEN }}  
 - name: Build and push Docker image  
 uses: docker/build-push-action@v2  
 with:  
 push: true  
 tags: |  
 ${{ secrets.DOCKER\_HUB\_USERNAME }}/movie-recommender:latest  
 ${{ secrets.DOCKER\_HUB\_USERNAME }}/movie-recommender:${{ github.sha }}  
```

## 3.5 AWS Integration (Planned)

Created AWS EKS cluster configuration (`deploy/aws.py`):  
```python  
import boto3  
  
def create\_eks\_cluster():  
 eks = boto3.client('eks', region\_name='us-west-2')  
  
 response = eks.create\_cluster(  
 name='movie-recommendation-cluster',  
 version='1.26',  
 roleArn='arn:aws:iam::ACCOUNT\_ID:role/EKSClusterRole',  
 resourcesVpcConfig={  
 'subnetIds': ['subnet-XXXXXXXX', 'subnet-YYYYYYYY'],  
 'securityGroupIds': ['sg-ZZZZZZZZ'],  
 'endpointPublicAccess': True,  
 'endpointPrivateAccess': False  
 }  
 )  
  
 print("Cluster creation initiated:", response)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 create\_eks\_cluster()  
```

# 4. Challenges and Solutions

|  |  |
| --- | --- |
| Challenge | Solution |
| GitHub file size limit | Implemented Git LFS for large model files |
| Docker networking issues | Used proper port mapping and explicit host binding |
| Kubernetes secret management | Created secrets for API keys |
| Local Kubernetes setup | Leveraged Docker Desktop's built-in Kubernetes |
| AWS permissions | Documented required IAM permissions for production deployment |

# 5. Future Enhancements

- Implement database storage for movie data  
- Add user authentication and personalized recommendations  
- Set up monitoring and logging with Prometheus and Grafana  
- Implement A/B testing for recommendation algorithms  
- Create automated model retraining pipeline

# 6. Conclusion

This project successfully demonstrates the application of modern software engineering principles to an AI-based recommendation system. The containerized application can be deployed to any Kubernetes environment, providing scalability and maintainability.