Deployment Guide - Heart Disease Prediction MLOps Project

Table of Contents

Local Development Setup

Prerequisites

- Python 3.9 or higher
- Git
- Virtual environment tool (venv or conda)

Step 1: Clone and Setup

```bash
# Clone repository
git clone <your-repo-url>
cd heart-disease-mlops

# Make setup script executable
chmod +x setup.sh

# Run setup script
./setup.sh
```

Step 2: Manual Setup (if needed)

```bash
# Create virtual environment
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Download and prepare data
python src/download_data.py
```

```bash
# Run tests to verify installation
pytest tests/ -v
```

Running Locally

1. Train Models

```bash
# Activate virtual environment
source venv/bin/activate

# Train models with MLflow tracking
python src/train.py

# View experiments in MLflow UI
mlflow ui
# Access at http://localhost:5000
```

2. Start API Server

```bash
# Start FastAPI server
uvicorn src.app:app --reload --host 0.0.0.0 --port 8000

# Access API documentation
# Open browser: http://localhost:8000/docs
```

3. Test API

```bash
# Health check
curl http://localhost:8000/health

# Make prediction
curl -X POST http://localhost:8000/predict \
  -H "Content-Type: application/json" \
  -d @sample_input.json

# Or use the interactive docs at /docs
```

Docker Deployment

Build Docker Image

```
# Build the image
docker build -t heart-disease-api:latest .

# Verify image
docker images | grep heart-disease-api
```

Run Single Container

```
# Run container
docker run -d \
  --name heart-api \
  -p 8000:8000 \
  -v $(pwd)/models:/app/models \
  heart-disease-api:latest

# Check logs
docker logs heart-api

# Test container
curl http://localhost:8000/health
```

Run with Docker Compose (Full Stack)

```
# Start all services (API + Monitoring)
docker-compose up -d

# Check status
docker-compose ps

# View logs
docker-compose logs -f api

# Stop all services
docker-compose down
```

Services:

- API: http://localhost:8000
- Prometheus: http://localhost:9090
- Grafana: http://localhost:3000 (admin/admin)

Kubernetes Deployment

Prerequisites

- Kubernetes cluster (local: Docker Desktop, Minikube, or cloud: GKE, EKS, AKS)

- kubectl configured
- Docker image pushed to registry (optional for cloud)

Local Kubernetes (Docker Desktop / Minikube)

1. Enable Kubernetes in Docker Desktop

- Open Docker Desktop
- Go to Settings > Kubernetes
- Enable Kubernetes
- Apply & Restart

2. Load Docker Image

```
# For Docker Desktop (image already available locally)
docker build -t heart-disease-api:latest .

# For Minikube
minikube image load heart-disease-api:latest
```

3. Deploy Application

```
# Deploy API
kubectl apply -f deployment/kubernetes/deployment.yaml

# Verify deployment
kubectl get pods
kubectl get services

# Check pod logs
kubectl logs -l app=heart-disease-api

# Get service URL
kubectl get svc heart-disease-api-service
```

4. Access Application

```
# Port forward for local access
kubectl port-forward svc/heart-disease-api-service 8000:80

# Test
curl http://localhost:8000/health
```

Cloud Kubernetes Deployment

Google Kubernetes Engine (GKE)

```
# 1. Create GKE cluster
gcloud container clusters create heart-disease-cluster \
  --num-nodes=3 \
  --machine-type=n1-standard-2 \
  --zone=us-central1-a

# 2. Get credentials
gcloud container clusters get-credentials heart-disease-cluster \
  --zone=us-central1-a

# 3. Push image to Google Container Registry
docker tag heart-disease-api:latest gcr.io/PROJECT_ID/heart-disease-
api:latest
docker push gcr.io/PROJECT_ID/heart-disease-api:latest

# 4. Update deployment.yaml with GCR image path
# Edit: deployment/kubernetes/deployment.yaml
# Change: image: gcr.io/PROJECT_ID/heart-disease-api:latest

# 5. Deploy
kubectl apply -f deployment/kubernetes/deployment.yaml
kubectl apply -f deployment/kubernetes/ingress.yaml

# 6. Get external IP
kubectl get svc heart-disease-api-service
```

AWS Elastic Kubernetes Service (EKS)

```
# 1. Create EKS cluster
eksctl create cluster \
  --name heart-disease-cluster \
  --region us-west-2 \
  --nodegroup-name standard-workers \
  --node-type t3.medium \
  --nodes 3

# 2. Push image to ECR
aws ecr get-login-password --region us-west-2 | \
  docker login --username AWS --password-stdin ACCOUNT_ID.dkr.ecr.us-west-
2.amazonaws.com

docker tag heart-disease-api:latest \
  ACCOUNT_ID.dkr.ecr.us-west-2.amazonaws.com/heart-disease-api:latest

docker push ACCOUNT_ID.dkr.ecr.us-west-2.amazonaws.com/heart-disease-
api:latest
```

```
# 3. Deploy
kubectl apply -f deployment/kubernetes/deployment.yaml
```

Azure Kubernetes Service (AKS)

```
# 1. Create resource group
az group create --name heart-disease-rg --location eastus

# 2. Create AKS cluster
az aks create \
  --resource-group heart-disease-rg \
  --name heart-disease-cluster \
  --node-count 3 \
  --generate-ssh-keys

# 3. Get credentials
az aks get-credentials \
  --resource-group heart-disease-rg \
  --name heart-disease-cluster

# 4. Push to ACR
az acr create --resource-group heart-disease-rg \
  --name heartdiseaseacr --sku Basic

az acr login --name heartdiseaseacr

docker tag heart-disease-api:latest \
  heartdiseaseacr.azurecr.io/heart-disease-api:latest

docker push heartdiseaseacr.azurecr.io/heart-disease-api:latest

# 5. Deploy
kubectl apply -f deployment/kubernetes/deployment.yaml
```

Monitoring Setup

Deploy Monitoring Stack

```
# Deploy Prometheus and Grafana
kubectl apply -f deployment/kubernetes/monitoring.yaml

# Check status
kubectl get pods -l app=prometheus
kubectl get pods -l app=grafana

# Access Prometheus
kubectl port-forward svc/prometheus-service 9090:9090
```

```
# Access Grafana
kubectl port-forward svc/grafana-service 3000:3000
```

Configure Grafana Dashboard

1. Open Grafana: http://localhost:3000
2. Login: admin / admin
3. Add Prometheus data source:
   - URL: http://prometheus-service:9090
4. Import dashboard or create custom dashboard
5. Add panels for:
   - Request rate
   - Prediction latency
   - Predictions by class
   - Error rate

Key Metrics to Monitor

```
# Request rate
rate(prediction_requests_total[5m])

# Average latency
rate(prediction_latency_seconds_sum[5m]) /
rate(prediction_latency_seconds_count[5m])

# Predictions by class
prediction_by_class

# Error rate
rate(http_requests_total{status=~"5.."}[5m])
```

CI/CD Pipeline

GitHub Actions Setup

The CI/CD pipeline is automatically triggered on:

- Push to `main` or `develop` branches
- Pull requests to `main`

Pipeline stages:

1. **Lint & Test**: Code quality checks and unit tests
2. **Train Model**: Model training with MLflow tracking
3. **Build Docker**: Container image creation
4. **Deploy**: Deployment to production (main branch only)

Manual Pipeline Trigger

```
# Push to trigger
git add .
git commit -m "Your commit message"
git push origin main
```

View Pipeline Status

- GitHub: Actions tab in repository
- Check workflow runs and logs

Troubleshooting

Common Issues

1. Model not found error

```
# Ensure models are trained
python src/train.py

# Check models directory
ls -la models/

# Should contain:
# - best_model.pkl
# - preprocessor.pkl
```

2. Docker build fails

```
# Clear Docker cache
docker system prune -a

# Rebuild
docker build --no-cache -t heart-disease-api:latest .
```

3. Kubernetes pod fails

```
# Check pod status
kubectl describe pod <pod-name>

# Check logs
kubectl logs <pod-name>
```

```
# Common fixes:
# - Ensure image is available
# - Check resource limits
# - Verify environment variables
```

4. Port already in use

```
# Find process using port 8000
lsof -i :8000

# Kill process
kill -9 <PID>

# Or use different port
uvicorn src.app:app --port 8001
```

5. Import errors in tests

```
# Ensure src is in Python path
export PYTHONPATH="${PYTHONPATH}:$(pwd)/src"

# Or install as package
pip install -e .
```

Logs and Debugging

```
# Application logs
tail -f api_logs.log

# Docker container logs
docker logs -f heart-api

# Kubernetes pod logs
kubectl logs -f -l app=heart-disease-api

# MLflow logs
cat mlruns/*/meta.yaml
```

Health Checks

API Health

```
# Basic health
curl http://localhost:8000/health

# Expected response:
{
  "status": "healthy",
  "model_loaded": true,
  "preprocessor_loaded": true,
  "timestamp": "2024-01-01T12:00:00"
}
```

Kubernetes Health

```
# Check deployment
kubectl get deployments

# Check pods
kubectl get pods

# Check services
kubectl get services

# Detailed pod info
kubectl describe pod <pod-name>
```

Performance Optimization

1. API Performance

```
# Enable uvicorn workers
uvicorn src.app:app --workers 4 --host 0.0.0.0 --port 8000
```

2. Kubernetes Scaling

```
# Manual scaling
kubectl scale deployment heart-disease-api --replicas=5

# Auto-scaling is configured in deployment.yaml
# Based on CPU and memory utilization
```

3. Model Optimization

- Use model quantization

- Enable batch prediction
- Cache preprocessing results
- Use ONNX for faster inference

Security Considerations

1. **API Security**
   - Add authentication (JWT tokens)
   - Rate limiting
   - HTTPS/TLS certificates
2. **Kubernetes Security**
   - Use secrets for sensitive data
   - Network policies
   - RBAC permissions
   - Pod security policies
3. **Data Privacy**
   - Encrypt patient data
   - Anonymize logs
   - Secure data storage

Next Steps

1. ✅ Complete local development and testing
2. ✅ Build and test Docker container
3. ✅ Deploy to Kubernetes
4. ✅ Set up monitoring and alerts
5. ✅ Configure CI/CD pipeline
6. ⬜ Add authentication
7. ⬜ Implement A/B testing
8. ⬜ Set up model retraining pipeline
9. ⬜ Create Helm chart
10. ⬜ Production deployment

For additional support, refer to:

- README.md - Project overview
- API Documentation: http://localhost:8000/docs
- MLflow UI: http://localhost:5000
- Monitoring: Prometheus and Grafana dashboards