Docker Deployment Guide - CMU-Africa Campus Assistant

This guide provides comprehensive instructions for deploying the CMU-Africa Campus Assistant using Docker.

📋 Table of Contents

🔧 Prerequisites

Before deploying, ensure you have the following installed:

- **Docker**: Version 20.10 or higher
  `bash`
  `docker --version`
- **Docker Compose**: Version 2.0 or higher
  `bash`
  `docker-compose --version`
- **API Keys**:
- OpenAI API key from platform.openai.com (https://platform.openai.com/api-keys)
- Pinecone API key from app.pinecone.io (https://app.pinecone.io/)

🚀 Quick Start

1. Clone and Navigate

```
cd /path/to/cmu-africa-campus-assistant
```

2. Configure Environment

```
# Copy the example environment file
cp .env.example .env

# Edit with your API keys
nano .env
```

Add your credentials:

```
OPENAI_API_KEY=your_openai_api_key_here
PINECONE_API_KEY=your_pinecone_api_key_here
PINECONE_ENVIRONMENT=us-east-1
```

3. Build and Start

```
# Build all images
./docker-build.sh

# Start all containers
./docker-start.sh
```

4. Access the Application

- **Frontend**: http://localhost
- **Backend API**: http://localhost:8001
- **API Documentation**: http://localhost:8001/docs

🏗️ Architecture

Docker Services

The application consists of two main services:

1. Backend Service

- **Image**: Python 3.11 slim
- **Server**: Gunicorn with Uvicorn workers
- **Port**: 8001
- **Features**:
- FastAPI REST API
- RAG (Retrieval-Augmented Generation) pipeline
- OpenAI GPT-4 integration
- Pinecone vector database
- Health checks

2. Frontend Service

- **Image**: Multi-stage build (Node.js 18 → Nginx Alpine)
- **Server**: Nginx
- **Port**: 80
- **Features**:

- React 18 with TypeScript
- Tailwind CSS
- API proxy to backend
- Static file serving
- Gzip compression
- Health checks

Network Architecture

```
┌──────────────────┐
│   User Browser   │
└──────────┬───────┘
           │ HTTP
           ↓
┌──────────────────────┐
│  Frontend (Nginx)    │
│   Port 80            │
└──────────┬───────────┘
           │ /api/* → http://backend:8001/
           ↓
┌──────────────────────┐
│  Backend (FastAPI)   │
│   Port 8001          │
└──────────┬───────────┘
           │
           ├─→ OpenAI API
           └─→ Pinecone Vector DB
```

⚙️ Configuration

Environment Variables

Root .env File (Required)

```
OPENAI_API_KEY=your_openai_api_key_here
PINECONE_API_KEY=your_pinecone_api_key_here
PINECONE_ENVIRONMENT=us-east-1
```

Backend Environment Variables

The backend service uses these variables from the root `.env`:
- `OPENAI_API_KEY`: OpenAI API authentication
- `PINECONE_API_KEY`: Pinecone database authentication
- `PINECONE_ENVIRONMENT`: Pinecone environment region

Frontend Environment Variables

The frontend uses nginx proxy to route API requests:
- API requests to `/api/*` are proxied to `http://backend:8001/`
- No direct API keys needed in frontend

Docker Compose Configuration

The `docker-compose.yml` defines:

```
services:
  backend:
    - Builds from backend/Dockerfile
    - Exposes port 8001
    - Mounts data directory (read-only)
    - Health checks every 30s

  frontend:
    - Builds from frontend/Dockerfile
    - Exposes port 80
    - Depends on backend health
    - Proxies /api/* to backend
```

🔨 Building Images

Build All Images

```
./docker-build.sh
```

This script:
1. Checks for `.env` file
2. Builds both backend and frontend images
3. Uses `--no-cache` for clean builds
4. Shows build progress and errors

Build Individual Services

```
# Backend only
docker-compose build backend

# Frontend only
docker-compose build frontend
```

Build Options

```
# Parallel builds (faster)
docker-compose build --parallel

# No cache (clean build)
docker-compose build --no-cache

# Pull latest base images
docker-compose build --pull
```

▶️ Running the Application

Start All Services

```
./docker-start.sh
```

This script:
1. Validates `.env` file exists
2. Checks required environment variables
3. Starts containers in detached mode
4. Shows service URLs and status
5. Displays container health status

Manual Start

```
# Start in detached mode
docker-compose up -d

# Start with output (for debugging)
docker-compose up

# Start specific service
docker-compose up -d backend
```

Stop All Services

```
./docker-stop.sh
```

Or manually:

```
docker-compose down
```

Restart Services

```
docker-compose restart

# Restart specific service
docker-compose restart backend
```

📊 Monitoring and Logs

View All Logs

```
./docker-logs.sh
```

View Service-Specific Logs

```
# Backend logs only
./docker-logs.sh backend

# Frontend logs only
./docker-logs.sh frontend
```

Manual Log Commands

```
# Follow all logs
docker-compose logs -f

# Last 100 lines
docker-compose logs --tail=100

# Logs for specific service
docker-compose logs -f backend

# Logs since specific time
docker-compose logs --since 2h backend
```

Check Container Status

```
# List running containers
docker-compose ps

# Detailed container info
docker-compose ps -a

# Check health status
docker inspect cmu-africa-backend | grep -A 10 Health
```

Resource Usage

```
# Real-time stats
docker stats

# Container resource limits
docker-compose config
```

🔍 Troubleshooting

Common Issues

1. Port Already in Use

Error: `Bind for 0.0.0.0:80 failed: port is already allocated`

Solution:

```
# Check what's using the port
sudo lsof -i :80

# Kill the process or change port in docker-compose.yml
ports:
  - "8080:80"  # Use port 8080 instead
```

2. Environment Variables Not Loaded

Error: `Missing required environment variables`

Solution:

```
# Ensure .env file exists
ls -la .env

# Check .env format (no spaces around =)
cat .env

# Verify variables are loaded
docker-compose config | grep -A 3 environment
```

3. Backend Health Check Failing

Error: `Unhealthy` status for backend

Solution:

```
# Check backend logs
./docker-logs.sh backend

# Verify API is responding
curl http://localhost:8001/health

# Check Pinecone connection
docker-compose exec backend python -c "import pinecone; print('OK')"
```

4. Frontend Cannot Connect to Backend

Error: 502 Bad Gateway or API errors

Solution:

```
# Verify backend is healthy
docker-compose ps

# Check nginx proxy configuration
docker-compose exec frontend cat /etc/nginx/conf.d/default.conf

# Test backend from frontend container
docker-compose exec frontend wget -O- http://backend:8001/health
```

5. Build Failures

Error: Build fails during Docker image creation

Solution:

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

# Rebuild without cache
docker-compose build --no-cache

# Check disk space
df -h
```

Debug Commands

```
# Access backend container shell
docker-compose exec backend /bin/bash

# Access frontend container shell
docker-compose exec frontend /bin/sh

# View environment variables inside container
docker-compose exec backend env

# Test network connectivity
docker-compose exec frontend ping backend

# Check mounted volumes
docker-compose exec backend ls -la /app/data
```

Health Check URLs

```
# Backend health
curl http://localhost:8001/health

# Frontend health
curl http://localhost/health

# Backend API docs
curl http://localhost:8001/docs
```

🚀 Production Deployment

Pre-Deployment Checklist

- [ ] All environment variables configured
- [ ] SSL/TLS certificates obtained (for HTTPS)
- [ ] Firewall rules configured
- [ ] Domain name configured (if applicable)
- [ ] Backup strategy in place
- [ ] Monitoring tools set up
- [ ] Log rotation configured
- [ ] Resource limits defined

Production Configuration

1. Use Production .env

```
# Production .env
OPENAI_API_KEY=prod_key_here
PINECONE_API_KEY=prod_key_here
PINECONE_ENVIRONMENT=us-east-1

# Additional production variables
NODE_ENV=production
LOG_LEVEL=warning
```

2. Resource Limits

Update `docker-compose.yml`:

```yaml
services:
  backend:
    deploy:
      resources:
        limits:
          cpus: '2'
          memory: 2G
        reservations:
          cpus: '1'
          memory: 1G
    restart: always
```

3. SSL/HTTPS Setup

Add reverse proxy (Nginx or Traefik):

```yaml
services:
  reverse-proxy:
    image: traefik:v2.10
    command:
      - "--providers.docker=true"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--certificatesresolvers.myresolver.acme.tlschallenge=true"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
      - "./letsencrypt:/letsencrypt"
```

4. Logging Configuration

Set up centralized logging:

```yaml
services:
  backend:
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

5. Security Hardening

```
# Run containers as non-root
# Add to Dockerfile:
USER nonroot:nonroot

# Use Docker secrets for sensitive data
docker secret create openai_key ./openai_key.txt

# Enable Docker Content Trust
export DOCKER_CONTENT_TRUST=1
```

Deployment to Cloud Providers

AWS ECS

```
# Install AWS CLI
# Configure credentials
# Push images to ECR
# Create ECS task definitions
# Deploy service
```

Google Cloud Run

```
# Build and push to GCR
gcloud builds submit --tag gcr.io/PROJECT_ID/cmu-africa-backend
gcloud builds submit --tag gcr.io/PROJECT_ID/cmu-africa-frontend

# Deploy services
gcloud run deploy backend --image gcr.io/PROJECT_ID/cmu-africa-backend
gcloud run deploy frontend --image gcr.io/PROJECT_ID/cmu-africa-frontend
```

Azure Container Instances

```
# Push to Azure Container Registry
az acr build --registry myregistry --image cmu-africa-backend .
az acr build --registry myregistry --image cmu-africa-frontend .

# Deploy containers
az container create --resource-group mygroup --name backend \
  --image myregistry.azurecr.io/cmu-africa-backend
```

Production Monitoring

1. Health Monitoring

```
# Set up monitoring script
#!/bin/bash
while true; do
  curl -f http://localhost/health || echo "Frontend down!"
  curl -f http://localhost:8001/health || echo "Backend down!"
  sleep 60
done
```

2. Log Aggregation

Use tools like:

- **ELK Stack** (Elasticsearch, Logstash, Kibana)
- **Grafana Loki**
- **Cloud provider logging** (CloudWatch, Stackdriver)

3. Metrics Collection

```
# Add Prometheus metrics
services:
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
```

🔄 Maintenance

Updating the Application

```
# Pull latest code
git pull origin main

# Rebuild images
./docker-build.sh

# Restart with new images
docker-compose up -d --build
```

Database Maintenance

```
# Backup data
docker-compose exec backend python load_knowledge_base.py

# Update knowledge base
# Edit data/sample_knowledge_base.json
docker-compose restart backend
```

Cleanup

```
# Remove stopped containers
docker-compose rm

# Remove unused images
docker image prune -a

# Remove unused volumes
docker volume prune

# Full cleanup (be careful!)
docker system prune -a --volumes
```

Backup Strategy

```
# Backup script
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)

# Backup data
tar -czf backup_data_$DATE.tar.gz ./data

# Backup configuration
tar -czf backup_config_$DATE.tar.gz .env docker-compose.yml

# Store backups securely
aws s3 cp backup_*.tar.gz s3://my-backup-bucket/
```

Log Rotation

```
# Configure log rotation
cat > /etc/logrotate.d/docker-containers <<EOF
/var/lib/docker/containers/*/*.log {
  rotate 7
  daily
  compress
  size=10M
  missingok
  delaycompress
  copytruncate
}
EOF
```

📝 Additional Resources

- **Docker Documentation**: https://docs.docker.com/
- **Docker Compose**: https://docs.docker.com/compose/
- **FastAPI**: https://fastapi.tiangolo.com/
- **React**: https://react.dev/
- **Nginx**: https://nginx.org/en/docs/

🆘 Support

If you encounter issues:

1. Check the Troubleshooting section
2. Review logs with `./docker-logs.sh`
3. Verify environment configuration
4. Check container health: `docker-compose ps`
5. Consult the main README.md for application-specific help

📄 License

[Include your license information here]

Last Updated: October 2025
Version: 1.0.0