Docker Test Plan - CMU-Africa Campus Assistant

This document outlines the testing procedures for the Docker deployment.

✅ Pre-Test Checklist

- [ ] Docker installed (20.10+)
- [ ] Docker Compose installed (2.0+)
- [ ] `.env` file configured with valid API keys
- [ ] Ports 80 and 8001 available
- [ ] Sufficient disk space (minimum 5GB)

🧪 Test Categories

1. Build Tests

Test 1.1: Backend Image Build

```
cd backend
docker build -t cmu-africa-backend:test .
```

Expected Result:
- Build completes successfully
- Image size reasonable (~500MB)
- All Python dependencies installed

Verification:

```
docker images | grep cmu-africa-backend
docker run --rm cmu-africa-backend:test python --version
```

Test 1.2: Frontend Image Build

```
cd frontend
docker build -t cmu-africa-frontend:test .
```

Expected Result:
- Multi-stage build completes
- Production bundle created
- Nginx configured correctly
- Image size optimized (~50MB)

Verification:

```
docker images | grep cmu-africa-frontend
docker run --rm -p 8080:80 cmu-africa-frontend:test
```

Test 1.3: Full Stack Build

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

Expected Result:
- Both images build successfully
- No errors or warnings
- Script completes with success message

2. Configuration Tests

Test 2.1: Environment Variables

```
# Test .env loading
docker-compose config | grep -A 3 environment
```

Expected Result:
- All required variables present
- No exposed secrets in output
- Proper variable substitution

Test 2.2: Docker Compose Validation

```
docker-compose config
```

Expected Result:
- Valid YAML configuration
- All services defined correctly
- Networks and volumes configured

Test 2.3: Network Configuration

```
docker-compose up -d
docker network inspect cmu-africa-network
```

Expected Result:
- Bridge network created
- Both containers connected
- Proper DNS resolution between services

3. Runtime Tests

Test 3.1: Container Startup

```
./docker-start.sh
docker-compose ps
```

Expected Result:
- Both containers start successfully
- Status shows “Up” and “healthy”
- No restart loops

Test 3.2: Backend Health Check

```
# Wait 30 seconds for startup
sleep 30

# Test health endpoint
curl http://localhost:8001/health
curl http://localhost:8001/api/health
```

Expected Result:

```
{
  "status": "healthy"
}
```

Test 3.3: Frontend Health Check

```
curl http://localhost/health
curl -I http://localhost
```

Expected Result:
- HTTP 200 status
- Nginx serving correctly
- HTML content delivered

Test 3.4: API Proxy Test

```
curl http://localhost/api/
```

Expected Result:
- Request proxied to backend
- Response from FastAPI
- CORS headers present

4. Integration Tests

Test 4.1: Frontend-Backend Communication

```
# From host
curl -X POST http://localhost/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "Hello", "session_id": "test"}'
```

Expected Result:
- Request reaches backend
- RAG pipeline processes query
- Valid response returned

Test 4.2: Inter-Container Networking

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

Expected Result:
- Frontend can reach backend by service name
- DNS resolution works
- Response received

Test 4.3: Volume Mounts

```
docker-compose exec backend ls -la /app/data
```

Expected Result:
- Data directory mounted
- Knowledge base file accessible
- Read-only mount working

5. Performance Tests

Test 5.1: Resource Usage

```
docker stats --no-stream
```

Expected Result:
- Backend: < 500MB memory
- Frontend: < 50MB memory
- CPU usage reasonable

Test 5.2: Response Time

```
time curl http://localhost/api/
```

Expected Result:
- Response time < 100ms for simple requests
- No significant delays

Test 5.3: Concurrent Requests

```
ab -n 100 -c 10 http://localhost/api/
```

Expected Result:
- No failures
- Consistent response times
- No container crashes

6. Reliability Tests

Test 6.1: Container Restart

```
docker-compose restart backend
docker-compose ps
```

Expected Result:
- Service restarts cleanly

- Health check passes after restart
- No data loss

Test 6.2: Crash Recovery

```
docker kill cmu-africa-backend
sleep 5
docker-compose ps
```

Expected Result:
- Container auto-restarts
- Service recovers automatically
- Logs show restart

Test 6.3: Health Check Monitoring

```
# Wait for health checks to run
sleep 60
docker inspect cmu-africa-backend | grep -A 10 Health
```

Expected Result:
- Health checks running
- Status: healthy
- No failed checks

7. Logging Tests

Test 7.1: Backend Logs

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

Expected Result:
- Application logs visible
- Gunicorn worker logs present
- No error messages

Test 7.2: Frontend Logs

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

Expected Result:
- Nginx access logs visible
- No 502/503 errors
- Proper request routing

Test 7.3: Error Logging

```
# Trigger error
curl http://localhost/nonexistent
./docker-logs.sh frontend | tail -20
```

Expected Result:
- 404 error logged
- No application crashes
- Graceful error handling

8. Security Tests

Test 8.1: Environment Isolation

```
docker-compose exec backend env | grep -i key
```

Expected Result:
- Environment variables present
- API keys not exposed in logs
- Proper secret handling

Test 8.2: Network Isolation

```
docker network inspect cmu-africa-network
```

Expected Result:
- Containers on private network
- No unnecessary port exposure
- Proper firewall rules

Test 8.3: File Permissions

```
docker-compose exec backend ls -la /app
docker-compose exec frontend ls -la /usr/share/nginx/html
```

Expected Result:
- Files have appropriate permissions
- No world-writable files
- Proper ownership

9. Cleanup Tests

Test 9.1: Stop Services

```
./docker-stop.sh
docker-compose ps -a
```

Expected Result:
- All containers stopped
- No running processes
- Cleanup successful

Test 9.2: Remove Containers

```
docker-compose down
docker ps -a | grep cmu-africa
```

Expected Result:
- Containers removed
- Network removed
- Volumes retained (if configured)

Test 9.3: Image Cleanup

```
docker rmi cmu-africa-backend:latest cmu-africa-frontend:latest
docker images | grep cmu-africa
```

Expected Result:
- Images removed successfully
- No dangling images
- Disk space freed

10. Production Readiness Tests

Test 10.1: SSL/TLS Readiness

- [ ] HTTPS configuration tested
- [ ] Certificate validation
- [ ] Secure headers present

Test 10.2: Scalability Test

```
docker-compose up -d --scale backend=3
```

Expected Result:
- Multiple backend instances start
- Load balancing works
- No port conflicts

Test 10.3: Backup and Restore

```
# Backup
docker-compose exec backend tar -czf /tmp/backup.tar.gz /app/data
docker cp cmu-africa-backend:/tmp/backup.tar.gz ./backup.tar.gz

# Restore
docker cp backup.tar.gz cmu-africa-backend:/tmp/
docker-compose exec backend tar -xzf /tmp/backup.tar.gz
```

Expected Result:
- Data backed up successfully
- Restore completes without errors
- Application continues working

📊 Test Results Template

Create a test report using this template:

```
# Docker Test Report
Date: [Date]
Tester: [Name]
Environment: [Dev/Staging/Production]

## Summary
- Total Tests: X
- Passed: X
- Failed: X
- Skipped: X

## Failed Tests
1. [Test Name]
   - Error: [Description]
   - Resolution: [Action taken]

## Performance Metrics
- Build Time: X minutes
- Startup Time: X seconds
- Memory Usage: X MB
- Response Time: X ms

## Recommendations
[List any recommendations]

## Sign-off
- [ ] All critical tests passed
- [ ] Ready for production
```

🔄 Continuous Testing

Automated Test Script

Create `test-docker.sh`:

```
#!/bin/bash

echo "Running Docker tests..."

# Build test
./docker-build.sh || exit 1

# Start test
./docker-start.sh || exit 1

# Wait for startup
sleep 30

# Health checks
curl -f http://localhost:8001/health || exit 1
curl -f http://localhost/health || exit 1

# API test
response=$(curl -s -X POST http://localhost/api/chat \
  -H "Content-Type: application/json" \
  -d '{"message": "test", "session_id": "test"}')

echo "Response: $response"

# Cleanup
./docker-stop.sh

echo "All tests passed!"
```

CI/CD Integration

Add to `.github/workflows/docker-test.yml`:

```
name: Docker Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Build images
        run: docker-compose build
      - name: Run tests
        run: ./test-docker.sh
```

📝 Notes

- Run tests in a clean environment
- Document all failures and resolutions
- Update this plan based on findings
- Keep test scripts in version control

Last Updated: October 2025
Version: 1.0.0