AI Platform - Complete Setup and Deployment Guide

📋 Table of Contents

1. Prerequisites
2. Quick Start (5 Minutes)
3. Detailed Setup
4. Script Reference
5. Database Management
6. Backup & Restore
7. Testing & Verification
8. Troubleshooting
9. Production Deployment

Prerequisites

Required Software

- **Docker** 20.10+ (Install)
- **Docker Compose** 2.0+ (Install)
- **Git** (for cloning/version control)
- **curl** (for API testing)

System Requirements

- 2GB+ RAM
- 5GB+ free disk space
- Linux, macOS, or Windows with WSL2

Check Prerequisites

```bash
docker --version          # Should show 20.10+
docker-compose --version  # Should show 2.0+
docker info               # Should not show errors
```

Quick Start (5 Minutes)

For users who already have all code files:

```bash
# 1. Make all scripts executable
chmod +x *.sh

# 2. Verify file structure
./check.sh

# 3. Deploy everything (automated)
./deploy.sh

# That's it! System is running.
```

The `deploy.sh` script handles:

- ✅ Environment configuration
- ✅ Docker container setup
- ✅ Database initialization
- ✅ Service verification
- ✅ Health checks

Access your platform:

- API: http://localhost/api
- API Health: http://localhost/api/health
- Traefik Dashboard: http://localhost:8080

Detailed Setup

Step 1: Create Project Structure

```bash
# Option A: Use setup script (recommended)
./setup.sh

# Option B: Manual creation
mkdir -p ai-platform
cd ai-platform
mkdir -p backend/src/{config,middleware,utils,services/adapters,routes,jobs}
mkdir -p backend/init-db
```

Step 2: Copy All Code Files

Refer to **"Artifact to File Mapping Guide"** to copy all files:

Root Level:

- `docker-compose.yml`

Backend Root:

- `backend/Dockerfile`
- `backend/package.json`
- `backend/.env` (create from template below)

Database:

- `backend/init-db/init.sql`

Source Files: (20 JavaScript files total)

- See "Accurate Backend File Structure & Checklist" artifact for complete list

Scripts: (10 shell scripts)

- All `.sh` files in project root

.env Template

Create `backend/.env`:

```
env

NODE_ENV=production
PORT=3000

# Database Configuration
DB_HOST=postgres
DB_PORT=5432
DB_USER=aiplatform
DB_PASSWORD=aiplatform123
DB_NAME=aiplatform

# JWT Secret - Generate a random secret key!
# You can generate one with: openssl rand -base64 32
JWT_SECRET=CHANGE-THIS-TO-A-RANDOM-SECRET-KEY
```

Step 3: Verify File Structure

```
bash

# Make check script executable
chmod +x check.sh

# Run verification
./check.sh
```

Expected output:

```
✓ docker-compose.yml
✓ backend/Dockerfile
✓ backend/package.json
...
✅ All files and folders are present!
```

If any files are missing, refer to the mapping guide and copy them.

Step 4: Deploy the Platform

```
bash

# Make deploy script executable
chmod +x deploy.sh

# Run master deployment script
./deploy.sh
```

The deploy script will:

1. ✅ Check Docker is installed and running
2. ✅ Verify all files exist (runs check.sh)
3. ✅ Create .env if missing (with random JWT secret)
4. ✅ Ask if you want to remove existing containers/volumes
5. ✅ Build and start Docker containers
6. ✅ Wait for services to be ready
7. ✅ Initialize database (runs check-db.sh)
8. ✅ Offer manual DB init if needed
9. ✅ Test API health endpoint
10. ✅ Show next steps

Interactive prompts:

- "Create .env file?" - Usually answer **yes**
- "Stop existing containers?" - **Yes** for clean slate
- "Remove volumes?" - **Yes** for fresh start (DELETES DATA)
- "Initialize database manually?" - **Yes** if auto-init failed

Step 5: Verify Deployment

```bash
# Check all containers are running
docker-compose ps

# Should show:
# traefik    Up
# postgres   Up (healthy)
# backend    Up

# Verify database
./check-db.sh

# Test API
curl http://localhost/api/health
# Should return: {"status":"ok","timestamp":"..."}
```

Script Reference

Core Scripts

1. setup.sh (Optional)

Purpose: Create directory structure

```
bash

./setup.sh
```

Use when: First time, before copying files

---

2. check.sh (Required)

Purpose: Verify all files and folders exist

```
bash

./check.sh
```

Output: Lists all files with ✓ or ✗
Use when: Before deployment, troubleshooting

---

3. deploy.sh ⭐ (Master Script)

Purpose: Complete automated deployment

```
bash

./deploy.sh
```

What it does:

- Pre-flight checks
- File verification
- Environment setup
- Container management
- Database initialization
- Health checks

Use when: Initial setup, redeployment

---

4. check-db.sh (Required)

Purpose: Verify database initialization

```
bash

./check-db.sh
```

Output:

```
✓ Table 'users' exists
✓ Table 'ai_providers' exists
...
✅ Database is properly initialized!
```

Use when: After deployment, troubleshooting

---

5. init-db-manual.sh (As Needed)

Purpose: Manually initialize database

```
bash

./init-db-manual.sh
```

Use when: Automatic initialization failed

Backup & Restore Scripts

6. backup.sh (Essential)

Purpose: Create database backup

```
bash

./backup.sh
```

Creates: `backups/backup_YYYYMMDD_HHMMSS.sql.gz`

Features:

- Timestamped files
- Automatic compression (gzip)
- Auto-cleanup old backups
- Keeps last 10 backups

Use when:

- Before major changes
- Daily (automated with cron)
- Before updates/deployments

---

7. list-backups.sh (Useful)

Purpose: View all available backups

```
bash

./list-backups.sh
```

Shows:

- Total backup count
- Backup sizes
- Backup dates
- Regular vs safety backups

---

8. restore.sh (Essential)

Purpose: Restore database from backup

```
bash

# Interactive (choose from list)
./restore.sh

# Direct (specify file)
./restore.sh ./backups/backup_20241205_143022.sql.gz
```

Safety features:

- Creates safety backup first
- Warns about data loss
- Stops backend during restore
- Verifies after restore

Use when: Recovery, migration, rollback

---

9. rollback.sh (Essential)

Purpose: Emergency rollback to latest backup

```
bash

./rollback.sh
```

Use when: Quick recovery needed, something went wrong

---

10. cleanup-backups.sh (Useful)

Purpose: Remove old backups

```
bash

./cleanup-backups.sh
```

Features:

- Configurable retention
- Shows files before deletion
- Can clean safety backups

Use when: Disk space management, weekly maintenance

Database Management

Checking Database Status

```
bash

# Verify tables exist
./check-db.sh

# Connect to database
docker-compose exec postgres psql -U aiplatform -d aiplatform

# Inside psql:
\dt                 # List tables
\d users            # Describe users table
SELECT * FROM users;   # Query users
\q                  # Exit
```

Manual Database Operations

```bash
# View PostgreSQL logs
docker-compose logs postgres

# Restart database
docker-compose restart postgres

# Reset database (DELETES ALL DATA)
docker-compose down -v
docker-compose up -d
./check-db.sh
```

Adding AI Provider API Keys

```bash
# 1. Register a user first
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123","email":"admin@example.com"}'

# Save the token from response

# 2. Add API key for ChatGPT
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "provider_id": 1,
    "key_value": "sk-your-openai-api-key-here"
  }'

# 3. Add API key for Claude
curl -X POST http://localhost/api/providers/keys \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "provider_id": 2,
    "key_value": "sk-ant-your-anthropic-key-here"
  }'

# 4. List all providers
curl http://localhost/api/providers \
  -H "Authorization: Bearer YOUR_TOKEN"
```

Backup & Restore

Daily Backup Routine

```bash
# Create backup
./backup.sh

# View all backups
./list-backups.sh

# Weekly: Clean old backups
./cleanup-backups.sh
```

Automated Backups (Cron)

```bash
# Edit crontab
crontab -e

# Add daily backup at 2 AM
0 2 * * * cd /path/to/ai-platform && ./backup.sh >> /var/log/ai-backup.log 2>&1

# Add weekly cleanup on Sunday at 3 AM
0 3 * * 0 cd /path/to/ai-platform && echo "10" | ./cleanup-backups.sh >> /var/log/ai-cleanup.log 2>&1
```

Restore Workflow

```bash
# 1. List available backups
./list-backups.sh

# 2. Restore from backup (interactive)
./restore.sh

# OR specify backup directly
./restore.sh ./backups/backup_20241205_143022.sql.gz

# 3. Verify restore
./check-db.sh
curl http://localhost/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123"}'
```

Emergency Recovery

```bash
# Something went wrong - quick rollback!
./rollback.sh

# This automatically:
# - Finds latest backup
# - Creates safety backup
# - Restores database
# - Restarts services
```

Testing & Verification

API Testing Workflow

```bash
# 1. Health check
curl http://localhost/api/health

# 2. Register user
curl -X POST http://localhost/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"username":"testuser","password":"test123"}'

# Save the token from response as TOKEN

# 3. Create project
curl -X POST http://localhost/api/projects \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"name":"Test Project","description":"Testing"}'

# Save project_id from response

# 4. Create chat (assuming provider_id=1 for ChatGPT)
curl -X POST http://localhost/api/chats/project/1 \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"title":"Test Chat","provider_id":1}'

# Save chat_id from response

# 5. Send message
curl -X POST http://localhost/api/chats/1/messages \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"content":"Hello! Tell me a joke."}'

# Should receive AI response!
```

Service Health Checks

```bash
# Check container status
docker-compose ps

# Check backend logs
docker-compose logs backend | tail -50

# Check PostgreSQL logs
docker-compose logs postgres | tail -50

# Check Traefik dashboard
open http://localhost:8080
```

Database Verification

```bash
# Run full database check
./check-db.sh

# Check specific tables
docker-compose exec postgres psql -U aiplatform -d aiplatform -c "
  SELECT
    'users' as table, COUNT(*) as count FROM users
  UNION ALL
  SELECT 'projects', COUNT(*) FROM projects
  UNION ALL
  SELECT 'chat_sessions', COUNT(*) FROM chat_sessions
  UNION ALL
  SELECT 'messages', COUNT(*) FROM messages;
"
```

Troubleshooting

Common Issues

1. "Docker daemon not running"

Problem: Docker Desktop/daemon not started

Solution:

```bash
# macOS/Windows: Start Docker Desktop application
# Linux:
sudo systemctl start docker
sudo systemctl enable docker
```

---

2. "Port already in use"

Problem: Port 80, 5432, or 8080 already occupied

Solution:

```bash
# Check what's using the port
sudo lsof -i :80
sudo lsof -i :5432

# Stop conflicting service or change ports in docker-compose.yml
```

---

3. "Database not initialized"

Problem: Tables don't exist after deployment

Solution:

```bash
# Check if auto-init worked
./check-db.sh

# If failed, run manual init
./init-db-manual.sh

# If still fails, reset completely
docker-compose down -v
docker-compose up -d
sleep 10
./check-db.sh
```

---

4. "Backend won't start"

Problem: Backend container keeps restarting

Solution:

```bash
# Check logs for errors
docker-compose logs backend

# Common causes:
# - Database not ready: Wait 10 more seconds
# - Missing dependencies: Rebuild container
docker-compose up -d --build backend

# - Syntax error: Check all .js files exist
./check.sh
```

---

5. "Permission denied" errors

Problem: Can't run scripts

Solution:

```bash
# Make scripts executable
chmod +x *.sh

# If still issues, check ownership
ls -la *.sh
chown $USER:$USER *.sh
```

---

6. "Backup failed"

Problem: backup.sh shows errors

Solution:

```bash
# Check disk space
df -h

# Check PostgreSQL is running
docker-compose ps postgres

# Check permissions
ls -la backups/
mkdir -p backups
chmod 700 backups
```

---

7. "API returns 502 Bad Gateway"

Problem: Traefik can't reach backend

Solution:

```bash
# Check backend is running
docker-compose ps backend

# Check backend logs
docker-compose logs backend

# Restart backend
docker-compose restart backend

# Check Traefik dashboard
open http://localhost:8080
```

Debug Mode

```bash
# Start with verbose logging
docker-compose up

# Keep logs open in separate terminal
docker-compose logs -f

# Check network connectivity
docker-compose exec backend ping postgres
docker-compose exec backend curl http://localhost:3000/api/health
```

Production Deployment

Security Hardening

1. Change Default Passwords

```bash
# Edit docker-compose.yml
nano docker-compose.yml

# Change:
POSTGRES_PASSWORD: aiplatform123  # ← Change this!
JWT_SECRET: your-secret-key       # ← Change this!
```

Generate secure passwords:

```bash
# PostgreSQL password
openssl rand -base64 32

# JWT secret
openssl rand -base64 32
```

2. Disable Traefik Dashboard in Production

```bash
# Edit docker-compose.yml, remove:
- "8080:8080"  # Traefik dashboard

# Or restrict to localhost only
- "127.0.0.1:8080:8080"
```

3. Enable HTTPS with Let's Encrypt

Add to docker-compose.yml traefik service:

```yaml
command:
  - "--certificatesresolvers.letsencrypt.acme.email=your@email.com"
  - "--certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json"
  - "--entrypoints.websecure.address=:443"
volumes:
  - "./letsencrypt:/letsencrypt"
```

4. Restrict API Access

Add rate limiting middleware to backend or use Traefik rate limiting.

5. Secure Backup Files

```bash
# Encrypt backups
gpg --symmetric --cipher-algo AES256 backups/*.sql.gz

# Restrict backup directory
chmod 700 backups/
chmod 600 backups/*
```

Environment Configuration

Create separate .env files:

backend/.env.production:

```env
NODE_ENV=production
PORT=3000
DB_HOST=postgres
DB_PORT=5432
DB_USER=aiplatform
DB_PASSWORD=<strong-random-password>
DB_NAME=aiplatform
JWT_SECRET=<strong-random-secret>
```

Monitoring

Set up log rotation

```bash
# Create /etc/logrotate.d/ai-platform
cat > /etc/logrotate.d/ai-platform << EOF
/var/log/ai-platform/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    create 0644 root root
}
EOF
```

Health Check Monitoring

```bash
# Add to crontab for alerts
*/5 * * * * curl -f http://localhost/api/health || mail -s "AI Platform Down" admin@example.com
```

Backup Strategy for Production

```bash
# Daily local backups
0 2 * * * cd /opt/ai-platform && ./backup.sh

# Weekly offsite backups
0 3 * * 0 cd /opt/ai-platform && rsync -av ./backups/ backup-server:/backups/ai-platform/

# Monthly long-term backups
0 4 1 * * cd /opt/ai-platform && ./backup.sh && cp backups/backup_*.sql.gz /archive/monthly/
```

Quick Reference

Essential Commands

```bash
# Start everything
./deploy.sh

# Stop everything
docker-compose down

# Restart a service
docker-compose restart backend

# View logs
docker-compose logs -f backend

# Create backup
./backup.sh

# Emergency rollback
./rollback.sh

# Check health
curl http://localhost/api/health
```

File Locations

```
ai-platform/
├── docker-compose.yml      # Container orchestration
├── *.sh                    # All management scripts
├── backend/
│   ├── .env                # Environment config
│   ├── Dockerfile          # Backend container
│   ├── package.json        # Node dependencies
│   ├── init-db/init.sql    # Database schema
│   └── src/                # Application code
└── backups/                # Database backups
```

Port Reference

- **80** - API endpoints (via Traefik)
- **8080** - Traefik dashboard
- **5432** - PostgreSQL (container only)
- **3000** - Backend (container only)

Documentation Reference

In the artifacts, find:

- **Backend API Design** - Complete API documentation
- **Database Initialization Guide** - DB setup details
- **Backup & Restore System Guide** - Backup procedures
- **How to Add New AI Provider** - Extending the platform
- **Accurate Backend File Structure** - File organization

Next Steps

After successful deployment:

1. ✅ **Add API Keys** - Register and add keys for AI providers
2. ✅ **Test API** - Create project, chat, send messages
3. ✅ **Setup Backups** - Configure automated daily backups
4. ✅ **Create Frontend** - Build React UI (next phase)
5. ✅ **Configure Monitoring** - Set up health checks and alerts

---

Support & Resources

- Check logs: `docker-compose logs -f`
- Database shell: `docker-compose exec postgres psql -U aiplatform`
- Backend shell: `docker-compose exec backend sh`
- All scripts have `--help` or show usage when run

Happy deploying! 🚀