Docker项目迁移

📋 第一步：安装 Docker Desktop

1. 下载并安装

- 访问 Docker Desktop for Windows
- 下载安装包并运行
- 安装过程中选择 **WSL 2** 后端（推荐）

2. 启动 Docker Desktop

- 安装完成后启动 Docker Desktop
- 等待 Docker 引擎启动（右下角图标变绿）

3. 验证安装

```
复制docker --version
docker-compose --version
```

📦 第二步：创建 Docker 配置文件

1. 在项目根目录创建 `Dockerfile`

```
复制# ================================================================================
# AI Chat 项目 Dockerfile
# ================================================================================

# 第一阶段：依赖安装
FROM node:20-alpine AS deps
WORKDIR /app

# 复制 package 文件
COPY package.json package-lock.json ./

# 安装依赖（包括 devDependencies）
RUN npm ci

# ================================================================================
# 第二阶段：构建应用
FROM node:20-alpine AS builder
WORKDIR /app

# 复制依赖
COPY --from=deps /app/node_modules ./node_modules
COPY . .
```

```
# 生成 Prisma Client
RUN npx prisma generate

# 构建 Next.js 应用
ENV NEXT_TELEMETRY_DISABLED 1
RUN npm run build

# ============================================================================
# 第三阶段：生产运行
FROM node:20-alpine AS runner
WORKDIR /app

ENV NODE_ENV production
ENV NEXT_TELEMETRY_DISABLED 1

# 创建非 root 用户
RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

# 复制必要文件
COPY --from=builder /app/public ./public
COPY --from=builder /app/.next/standalone ./
COPY --from=builder /app/.next/static ./.next/static
COPY --from=builder /app/prisma ./prisma
COPY --from=builder /app/node_modules/.prisma ./node_modules/.prisma

# 设置文件权限
RUN chown -R nextjs:nodejs /app

USER nextjs

EXPOSE 3000

ENV PORT 3000
ENV HOSTNAME "0.0.0.0"

CMD ["node", "server.js"]
```

2. 创建 `docker-compose.yml`

```
复制# ============================================================================
# AI Chat 项目 Docker Compose 配置
# ============================================================================

version: '3.8'

services:
  # PostgreSQL 数据库
  postgres:
    image: postgres:16-alpine
    container_name: ai-chat-db
```

```
    restart: unless-stopped
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: your_secure_password  # ⚠️ 修改为强密码
      POSTGRES_DB: ai_chat
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - ai-chat-network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  # Next.js 应用
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: ai-chat-app
    restart: unless-stopped
    depends_on:
      postgres:
        condition: service_healthy
    environment:
      # 数据库连接
      DATABASE_URL: postgresql://postgres:your_secure_password@postgres:5432/ai_chat

      # NextAuth 配置
      NEXTAUTH_URL: http://localhost:3000
      NEXTAUTH_SECRET: your_nextauth_secret_key_here  # ⚠️ 修改为随机字符串

      # OpenRouter API
      OPENAI_API_KEY: ${OPENAI_API_KEY}  # 从 .env 读取

      # 其他环境变量
      NODE_ENV: production
    ports:
      - "3000:3000"
    volumes:
      # 挂载上传目录（持久化）
      - ./public/uploads:/app/public/uploads
      # 开发模式：挂载源代码（可选）
      # - .:/app
      # - /app/node_modules
      # - /app/.next
    networks:
      - ai-chat-network
    command: >
      sh -c "
```

```
        npx prisma migrate deploy &&
        node server.js
      "

volumes:
  postgres_data:
    driver: local

networks:
  ai-chat-network:
    driver: bridge
```

---

3. 创建 `.dockerignore`

```
复制# Node.js
node_modules
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Next.js
.next
out
build

# 环境变量
.env
.env.local
.env.*.local

# Git
.git
.gitignore

# IDE
.vscode
.idea

# 其他
*.md
.DS_Store
```

---

4. 修改 `next.config.mjs`

在文件末尾添加：

```
复制const nextConfig = {
  // ... 现有配置

  // ✅ 添加：启用 standalone 输出（Docker 优化）
  output: 'standalone',
}

export default nextConfig
```

🔗 第三步：配置环境变量

创建 `.env` 文件（Docker 专用）

```
复制# OpenRouter API Key
OPENAI_API_KEY=your_openrouter_api_key_here

# 数据库密码（与 docker-compose.yml 一致）
POSTGRES_PASSWORD=your_secure_password

# NextAuth Secret（生成方法见下方）
NEXTAUTH_SECRET=your_nextauth_secret_key_here
```

生成 NEXTAUTH_SECRET

```
复制# 在 PowerShell 中运行
openssl rand -base64 32

# 或者使用 Node.js
node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"
```

🚀 第四步：启动 Docker 容器

1. 构建并启动

```
复制# 在项目根目录运行
docker-compose up -d --build
```

参数说明：

- `-d`：后台运行
- `--build`：强制重新构建镜像

2. 查看日志

```
复制# 查看所有服务日志
docker-compose logs -f

# 只看应用日志
docker-compose logs -f app

# 只看数据库日志
docker-compose logs -f postgres
```

3. 验证运行状态

```
复制# 查看容器状态
docker-compose ps

# 预期输出：
# NAME             IMAGE               STATUS
# ai-chat-app      ai-chat-app:latest  Up 2 minutes
# ai-chat-db       postgres:16-alpine  Up 2 minutes (healthy)
```

4. 访问应用

打开浏览器访问：http://localhost:3000

💻 第五步：VSCode 开发配置

方案 A：直接修改本地代码（推荐）

1. 安装 VSCode 扩展

- **Docker**（Microsoft 官方）
- **Remote - Containers**

2. 修改 `docker-compose.yml`（开发模式）

创建 `docker-compose.dev.yml`：

```
复制version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile.dev  # 使用开发版 Dockerfile
    volumes:
      # ✅ 挂载源代码（实时同步）
      - .:/app
      - /app/node_modules
```

```
      - /app/.next
    environment:
      NODE_ENV: development
    command: npm run dev
```

3. 创建 `Dockerfile.dev`

```
复制FROM node:20-alpine

WORKDIR /app

# 安装依赖
COPY package*.json ./
RUN npm install

# 复制源代码
COPY . .

# 生成 Prisma Client
RUN npx prisma generate

EXPOSE 3000

CMD ["npm", "run", "dev"]
```

4. 启动开发环境

```
复制# 使用开发配置启动
docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d

# 查看日志
docker-compose logs -f app
```

5. 在 VSCode 中开发

- 直接编辑本地文件
- 修改会自动同步到容器
- 支持 Hot Reload（热重载）

方案 B：在容器内开发（完全隔离）

1. 在 VSCode 中打开项目

2. 按 `F1`，输入 `Remote-Containers: Attach to Running Container`

3. 选择 `ai-chat-app` 容器

4. VSCode 会在容器内打开，可以直接编辑代码

🛠️ 常用 Docker 命令

容器管理

```
复制# 启动服务
docker-compose up -d

# 停止服务
docker-compose down

# 重启服务
docker-compose restart

# 查看日志
docker-compose logs -f

# 进入容器 Shell
docker exec -it ai-chat-app sh
```

数据库操作

```
复制# 运行 Prisma 迁移
docker exec ai-chat-app npx prisma migrate dev

# 重置数据库
docker exec ai-chat-app npx prisma migrate reset

# 打开 Prisma Studio
docker exec -it ai-chat-app npx prisma studio
```

清理资源

```
复制# 停止并删除容器
docker-compose down

# 删除所有数据（包括数据库）
docker-compose down -v

# 清理未使用的镜像
docker system prune -a
```

🐛 常见问题排查

1. 端口被占用

```
复制# 查看端口占用
netstat -ano | findstr :3000

# 修改 docker-compose.yml 中的端口映射
ports:
  - "3001:3000"  # 改为 3001
```

2. 数据库连接失败

```
复制# 检查数据库健康状态
docker exec ai-chat-db pg_isready -U postgres

# 查看数据库日志
docker-compose logs postgres
```

3. Prisma 迁移失败

```
复制# 手动运行迁移
docker exec ai-chat-app npx prisma migrate deploy

# 重新生成 Prisma Client
docker exec ai-chat-app npx prisma generate
```

4. 文件权限问题（WSL 2）

```
复制# 在 WSL 2 中修复权限
docker exec -u root ai-chat-app chown -R nextjs:nodejs /app
```

📊 性能优化建议

1. 使用多阶段构建（已包含）

- 减小镜像体积
- 提高构建速度

2. 启用 Docker BuildKit

```
复制# 在 PowerShell 中设置环境变量
$env:DOCKER_BUILDKIT=1
docker-compose build
```

3. 配置 WSL 2 资源限制

创建 `%USERPROFILE%\.wslconfig`：

```
复制[wsl2]
memory=4GB
processors=2
swap=2GB
```

🔄 CI/CD 集成（可选）

GitHub Actions 示例

创建 `.github/workflows/docker.yml`：

```
复制name: Docker Build

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Docker Image
        run: docker build -t ai-chat-app .

      - name: Run Tests
        run: docker run ai-chat-app npm test
```

✅ 验证清单

- Docker Desktop 已安装并运行
- 所有配置文件已创建
- 环境变量已正确设置
- 容器成功启动（`docker-compose ps`）
- 应用可访问（http://localhost:3000）
- 数据库连接正常
- 文件上传功能正常
- VSCode 可以编辑代码并实时生效

📚 推荐学习资源

- Docker 官方文档
- Next.js Docker 部署指南
- Prisma Docker 最佳实践