AI Configuration

OpenAI API Key

••

✓ API Key configured

AI Model

gpt-4o

Selected Model: gpt-4o

Custom API Base URL (Optional)

http://localhost:11434/v1 (optional)

Advanced Settings

About Multi-Agent Framework

Multi-Agentic System powered by AutoGen with GPT-4o

This framework orchestrates 7 specialized AI agents that collaborate to transform natural language requirements into production-ready code with full documentation, tests, and deployment configuration.

Agent Pipeline

1. Requirement Analyst - Structure requirements
2. Senior Developer - Generate code
3. Code Reviewer - Review & iterate (AutoGen loop)
4. Tech Writer - Create documentation
5. QA Engineer - Generate tests
6. DevOps - Deployment config
7. UI Designer - Streamlit interface

© 2026 AutoGen Multi-Agent Code Generator • Powered by AutoGen with GPT-4o

AutoGen Multi-Agent Code Generator

Transform Ideas into Production-Ready Code with AI Agent Collaboration

AI Agents	Framework	Model	Version
7	Multi-Agent	GPT-4o	2026

Enter Your Requirements

Describe what you want to build:

Create a Fast API REST API for a Todo List Manager with the following features:

1. CRUD Operations:
 - Create new Todo items with title, description, priority (low/medium/high), and due date

Quick Start Examples

Generate Code with AI Agents

AutoGen Pipeline Results

Generated Artifacts from Multi-Agent Collaboration

Execution Metrics

	Review Iterations	Iteration Limit	Run ID
✓ SUCCESS	2	Within Limit	37e66994

Requirements Analysis | Python Code | Code Review | Documentation | Test Suite | Deployment

Deployment Configuration

Generated by DevOps Agent

Deployment Configuration

requirements.txt

```
fastapi==0.95.0
uvicorn==0.22.0
sqlalchemy==1.4.47
pydantic==1.10.2
fastapi-limiter==0.1.0
python-decouple==3.6
```

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.10-slim as base

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Create a non-root user
RUN addgroup --system appgroup && adduser --system --group appuser

# Set the working directory
WORKDIR /app

# Copy the requirements file into the image
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the current directory contents into the container at /app
COPY . .

# Change to the non-root user
USER appuser

# Expose the port the app runs on
EXPOSE 8000

# Run the application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

docker-compose.yml

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=${DATABASE_URL}
    volumes:
      - ./data:/app/data
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/todos"]
      interval: 30s
      timeout: 10s
      retries: 3
    networks:
      - app-network

networks:
  app-network:
    driver: bridge
```

deploy.sh

```
#!/bin/bash

# Load environment variables from .env file
export $(grep -v '^#' .env | xargs)

# Build the Docker image
docker-compose build

# Run tests (assuming tests are in a directory named 'tests')
docker-compose run --rm app pytest tests

# Start the services
docker-compose up -d
```

```
# Check the status of the services
docker-compose ps
```

.env.example

```
# Database URL for SQLAlchemy
DATABASE_URL=sqlite:///./data/test.db
```

.dockerignore

```
__pycache__
*.pyc
*.pyo
*.pyd
.Python
env
venv
ENV
*.env
*.env.*
*.git
*.gitignore
.DS_Store
*.sqlite3
*.db
data/
```

Deployment Instructions

1. **Environment Setup**:
   - Ensure Docker and Docker Compose are installed on your system.
   - Copy `.env.example` to `.env` and configure the `DATABASE_URL` as needed.
2. **Build and Deploy**:
   - Run `./deploy.sh` to build the Docker image, run tests, and start the application.
   - The application will be available at `http://localhost:8000`.
3. **Security Best Practices**:
   - The application runs as a non-root user for enhanced security.
   - Use environment variables for sensitive configurations.
   - Ensure the `.env` file is not included in version control.
4. **Health Checks and Logging**:
   - Health checks are configured in `docker-compose.yml` to ensure the service is running.
   - Logging is set up in the application to capture all operations.
5. **Port Mappings and Volume Mounts**:
   - The application is exposed on port 8000.
   - Data persistence is achieved by mounting the `./data` directory to `/app/data` in the container.
6. **Environment Variable Management**:
   - Use the `.env` file to manage environment variables.
   - Ensure sensitive information is not hardcoded in the application code.
7. **Testing**:
   - Tests are run as part of the deployment script to ensure the application is functioning correctly before starting the services.

By following these instructions, you can deploy the Todo List Manager API in a production-ready environment using Docker and Docker Compose.

Download Deployment Config

Download All Artifacts (ZIP)

Multi-Agentic Framework © 2026
Powered by AutoGen with GPT-4o • Version 2026.1.0