

DevOps × AI: Automated CI/CD Pipeline for a RAG API

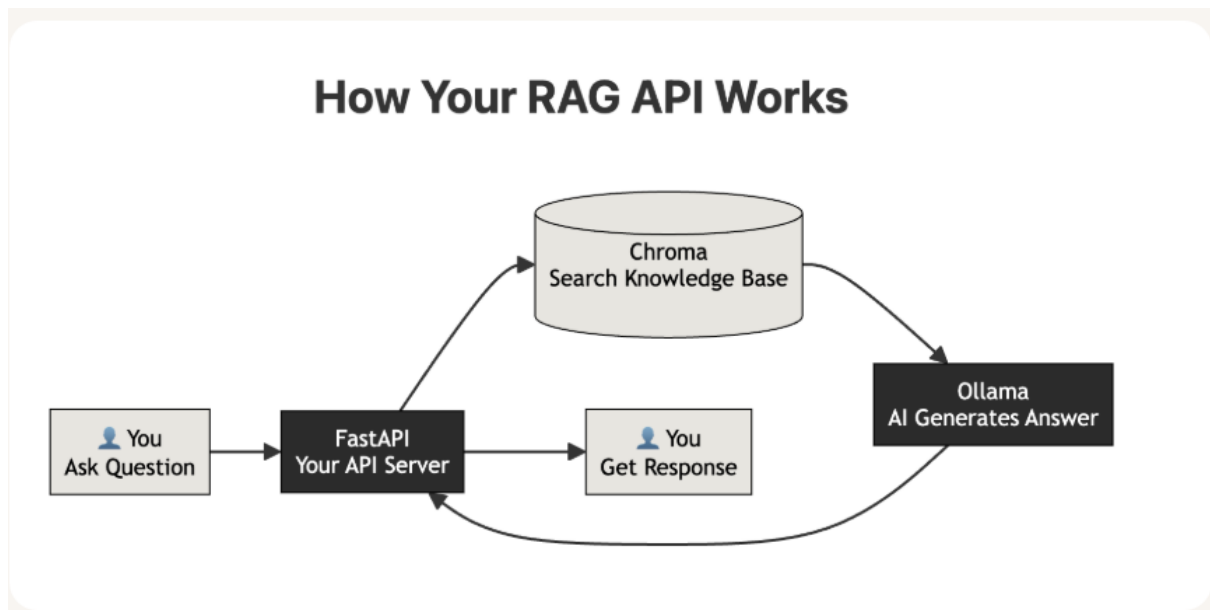
Project Overview

This project demonstrates how DevOps practices can be applied to AI systems by building and automating a Retrieval-Augmented Generation (RAG) API.

The project implements a complete workflow that includes:

- Document ingestion and embedding
- API-based retrieval and response generation
- Containerization for consistency
- Automated CI testing for AI data quality
- Scalable deployment using Kubernetes

The goal was to eliminate manual testing and environment-specific behavior and replace it with a **repeatable, automated, and reliable DevOps pipeline for AI workloads**.



Why This Project Matters

AI systems behave differently from traditional applications due to:

- Non-deterministic LLM outputs
- Data-driven behavior
- Frequent knowledge base updates

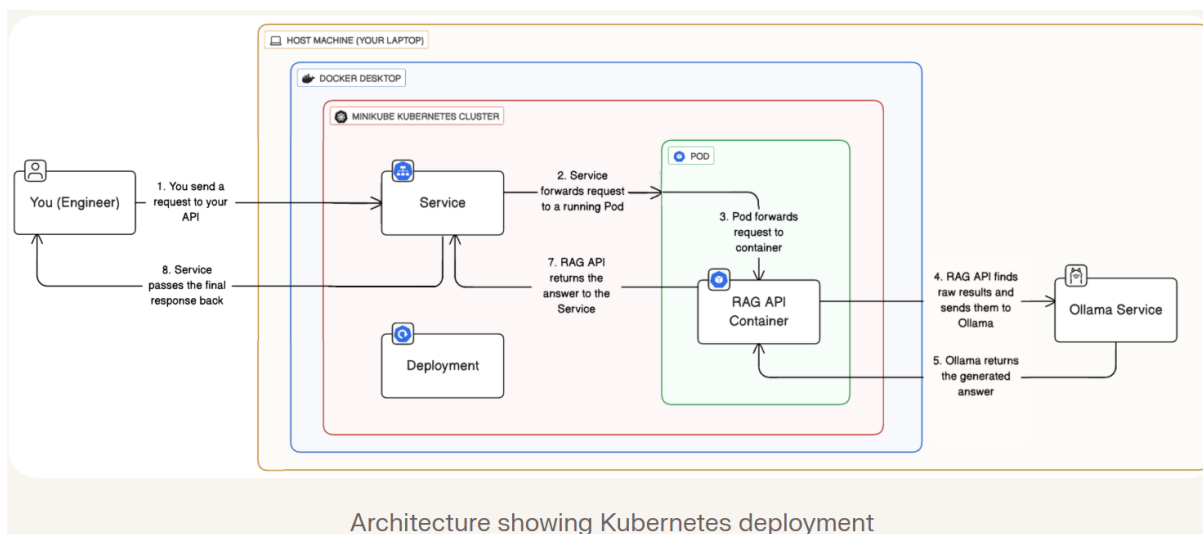
This project shows how DevOps principles can be extended into AI systems to:

- Ensure consistent behavior across environments
- Catch data quality issues early
- Automate validation of AI outputs
- Enable safe and scalable deployments
- Treat AI pipelines as production systems

Architecture Overview

End-to-End Flow

1. Developer updates code or knowledge documents
2. Documents are embedded into a vector database
3. API retrieves relevant context for user queries
4. Responses are generated (or mocked during CI)
5. CI pipeline runs automated semantic tests
6. Docker ensures consistent runtime environments
7. Kubernetes manages containerized deployment



Step-by-Step Implementation

Step 1: Project Initialization

- Create a Python-based project repository
 - Initialize virtual environment
 - Install required dependencies:
 - FastAPI
 - Uvicorn
 - ChromaDB
 - Requests
 - Set up project structure:
 - API code
 - Embedding scripts
 - Test scripts
 - Knowledge base documents
-

Step 2: Knowledge Base Setup

- Create a docs/ directory
 - Store domain-specific .txt files
 - These documents act as the single source of truth for AI responses
 - Changes to documents automatically affect API behavior
-

Step 3: Embedding Pipeline (Vector Store)

- Implement an embedding script to:
 - Read all documents in docs/
 - Convert text into vector embeddings
 - Store embeddings in ChromaDB
- Ensure embeddings can be rebuilt deterministically
- This step enables semantic search instead of keyword search

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/query?q=What%20is%20docker%3F' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://127.0.0.1:8000/query?q=What%20is%20docker%3F
```

Server response

Code Details

200

Response body

```
{
  "answer": "Docker is a popular application containerization tool that simplifies the process of deploying and managing applications in multiple environments. It enables developers to easily create, test, and deploy applications on different platforms, including virtual machines, containers, and physical servers. With Docker, you can build, ship, and run your code across various environments without worrying about compatibility issues or infrastructure complexity."
}
```

Response headers

```
content-length: 469
content-type: application/json
date: Tue, 27 Jan 2026 03:48:24 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

```
"string"
```

Step 4: RAG API Development

- Build a FastAPI application exposing a /query endpoint
- Query flow:
 - Receive user question
 - Retrieve relevant context from ChromaDB
 - Pass context to the language model
 - Return generated answer
- API designed to be stateless and container-friendly

Step 5: Mock LLM Mode for CI

LLMs are non-deterministic, which breaks automated testing.

To solve this:

- Introduce an environment variable-based **Mock LLM mode**
- During mock mode:
 - The API skips LLM generation
 - Returns retrieved context directly
- This enables deterministic and reliable CI testing

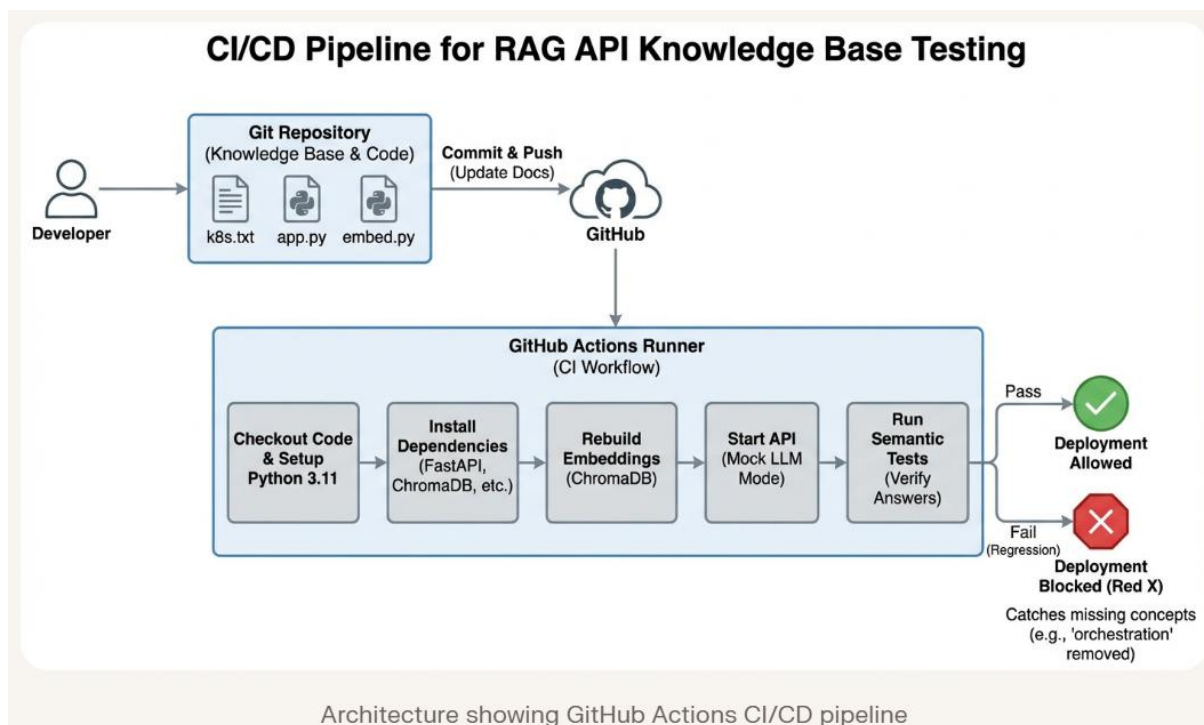
Step 6: Semantic Testing Strategy

Instead of exact string matching:

- Implement semantic tests that validate **meaning**
- Tests check:
 - Presence of expected concepts
 - Correct retrieval behavior
- This approach reflects real-world AI validation practices

Step 7: GitHub Actions CI Pipeline

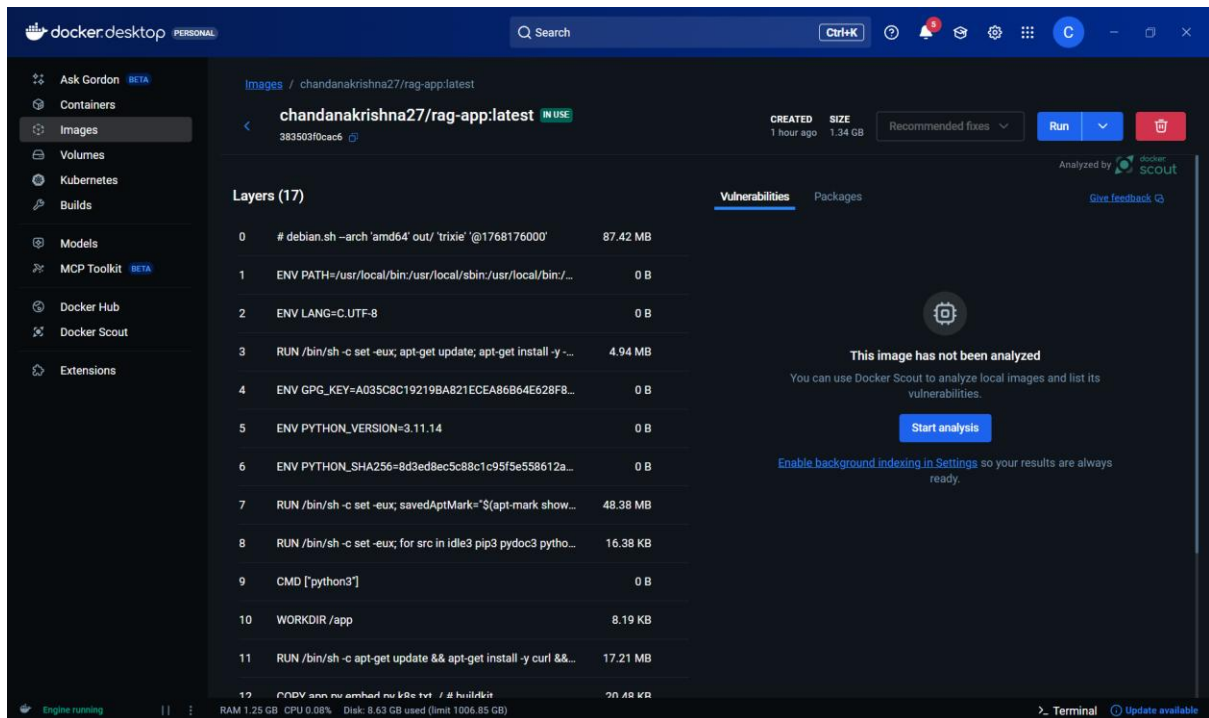
- Create a GitHub Actions workflow triggered on:
 - Knowledge base changes
 - API logic changes
 - Embedding script updates
- CI workflow performs:
 - Dependency installation
 - Embedding rebuild
 - API startup in mock mode
 - Automated semantic tests
- Failures block further deployment steps



Step 8: Containerization with Docker

- Package the API, dependencies, and embeddings into a Docker image
- Docker ensures:

- Identical runtime across machines
 - Isolation from local system dependencies
- Enables easy handoff between developers and environments



Step 9: Kubernetes Deployment

- Deploy containerized API to Kubernetes
- Kubernetes handles:
 - Pod management
 - Scaling
 - Restarting failed containers
- Mirrors production-grade deployment strategies

Step 10: Continuous Validation

- Any update to documents or logic:
 - Automatically triggers CI
 - Rebuilds embeddings
 - Runs semantic tests
- Prevents broken or misleading AI behavior from reaching production

Final Result

The project successfully delivers:

- A working RAG API using custom documents
- Automated testing for AI data quality
- Deterministic CI for non-deterministic models
- Consistent deployments via Docker
- Scalable container orchestration via Kubernetes

The system behaves predictably across environments and detects failures early.

Key Services & Tools Used

- FastAPI
- Uvicorn
- ChromaDB
- Ollama (tinyllama)
- Docker
- Kubernetes
- GitHub Actions
- GitHub
- Python

Key Concepts Learned

- Retrieval-Augmented Generation (RAG)
- Vector databases and embeddings
- Semantic testing for AI systems
- Handling LLM non-determinism in CI/CD
- Containerizing AI workloads
- Kubernetes-based deployment
- CI pipelines for data-driven applications
- Applying DevOps principles to AI