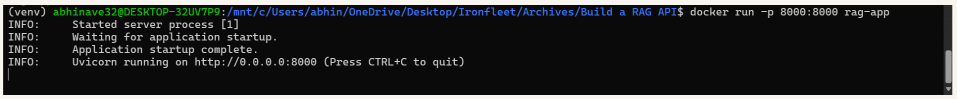
nextwork.org

Containerize a RAG API with Docker

Abhinave P.B



Abhinave P.B
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to design and build a Retrieval-Augmented Generation (RAG) system using FastAPI. The API allows users to submit natural language questions, retrieves the most relevant information from a custom document knowledge base stored in Chroma, and generates accurate, context-aware responses using Ollama. I’m doing this project to learn how modern AI systems combine information retrieval with large language models, and how to expose such systems through real-world APIs. Through this project, I gained hands-on experience with vector databases, embeddings, FastAPI backend development, and integrating local LLMs. This project helped me understand how AI-powered applications are built end-to-end and how APIs act as the bridge between users and intelligent systems.

Key services and concepts

Services I used were FastAPI, ChromaDB, Ollama, Uvicorn, Docker, and Docker Hub. Key concepts I learnt include API development, Retrieval-Augmented Generation (RAG), virtual environment and dependency management, containerization, building and running Docker images, cross-environment consistency, and versioning/distribution via Docker Hub.

Challenges and wins

This project took me approximately 2 hours. The most challenging part was docker build. It was most rewarding to see the final result

Why I did this project

I did this project because i want to learn how Docker works. And it was quiet helpful

Abhinave P.B
NextWork Student

Setting Up the RAG API

In this step, I'm setting up the complete development environment required to build and run the RAG API. This includes configuring the project structure, installing all necessary dependencies, setting up a virtual environment, and preparing the database used for document retrieval. The RAG API is a backend service that combines document retrieval and AI-based text generation. It stores document embeddings in a vector database (Chroma), retrieves the most relevant information based on a user's question, and uses Ollama to generate accurate responses grounded in the retrieved data. I created and activated a Python virtual environment to manage dependencies in an isolated workspace. I installed required libraries such as FastAPI, ChromaDB, and supporting AI tools. I ran Ollama locally to serve the language model that generates answers for the RAG system. This setup ensures that the RAG API runs reliably, remains modular, and can be easily extended or deployed in future

API setup and workspace

In this step, I create a virtual environment to isolate my RAG API's dependencies and ensure a reproducible setup, which will later be packaged into a Docker container for consistent deployment across systems.

Dependencies installed

The packages I installed are FastAPI, Chroma, Uvicorn, and Ollama. FastAPI is used for building fast and efficient backend APIs in Python. Chroma is used for storing vector embeddings and enabling semantic search for AI applications. Uvicorn is used for running and serving FastAPI applications as an ASGI server. Ollama is used for running large language models locally for AI-powered tasks without relying on cloud APIs.

Local API working

I tested the API locally, and it returned the expected response. This confirms that the endpoint is correctly configured and the backend logic is working as intended.

Abhinave P.B
NextWork Student


```
abhinave32@DESKTOP-32UV7P9:~$ ollama list
NAME                ID              SIZE      MODIFIED
tinyllama:latest    2644915ede35    637 MB    2 weeks ago
abhinave32@DESKTOP-32UV7P9:~$ curl http://localhost:11434
Ollama is runningabhinave32@DESKTOP-32UV7P9:~$
```

Abhinave P.B
NextWork Student

Installing Docker Desktop

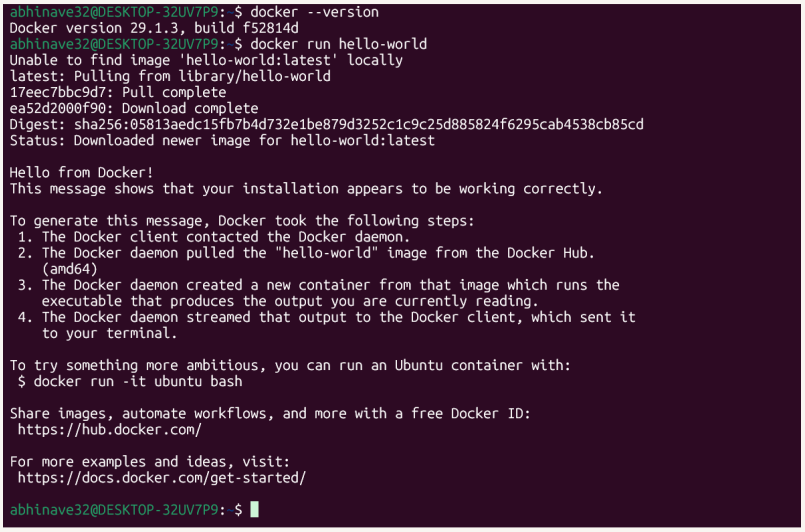
Docker Desktop setup

Docker Desktop is a tool that allows me to run and manage Docker containers on my system, providing a consistent and isolated development environment. I installed it because my project depends on specific runtimes, libraries, and services, and Docker helps avoid compatibility and setup issues across different machines. Containerization will help my project by ensuring consistency, simplifying dependency management, making the application easier to test, scale, and deploy, and eliminating “it works on my machine” problems.

Docker verification

I verified Docker is working by running the hello-world container. The hello-world container proves that Docker is correctly installed, the Docker daemon is running, images can be pulled from Docker Hub, and containers can be created and executed successfully.

Abhinave P.B
NextWork Student



Abhinave P.B
NextWork Student

Creating the Dockerfile

In this step, I'm building a RAG API. RAG stands for Retrieval-Augmented Generation, an approach that combines information retrieval with large language models to generate accurate, context-aware responses. I'm creating files such as main.py to define the API endpoints, rag.py to implement the retrieval and generation logic, requirements.txt to manage dependencies, and a Dockerfile to containerize the API for consistent deployment and testing.

How the Dockerfile works

A Dockerfile is a text file that contains a set of instructions used by Docker to build a Docker image. The key instructions in my Dockerfile are FROM, COPY, RUN, and CMD. FROM tells Docker to use a specific base image as the starting point for building the image. COPY is used for copying files and folders from the host machine into the Docker image. RUN executes commands during the image build process (for example, installing packages or dependencies). CMD defines the default command that runs when a container is started from the image.

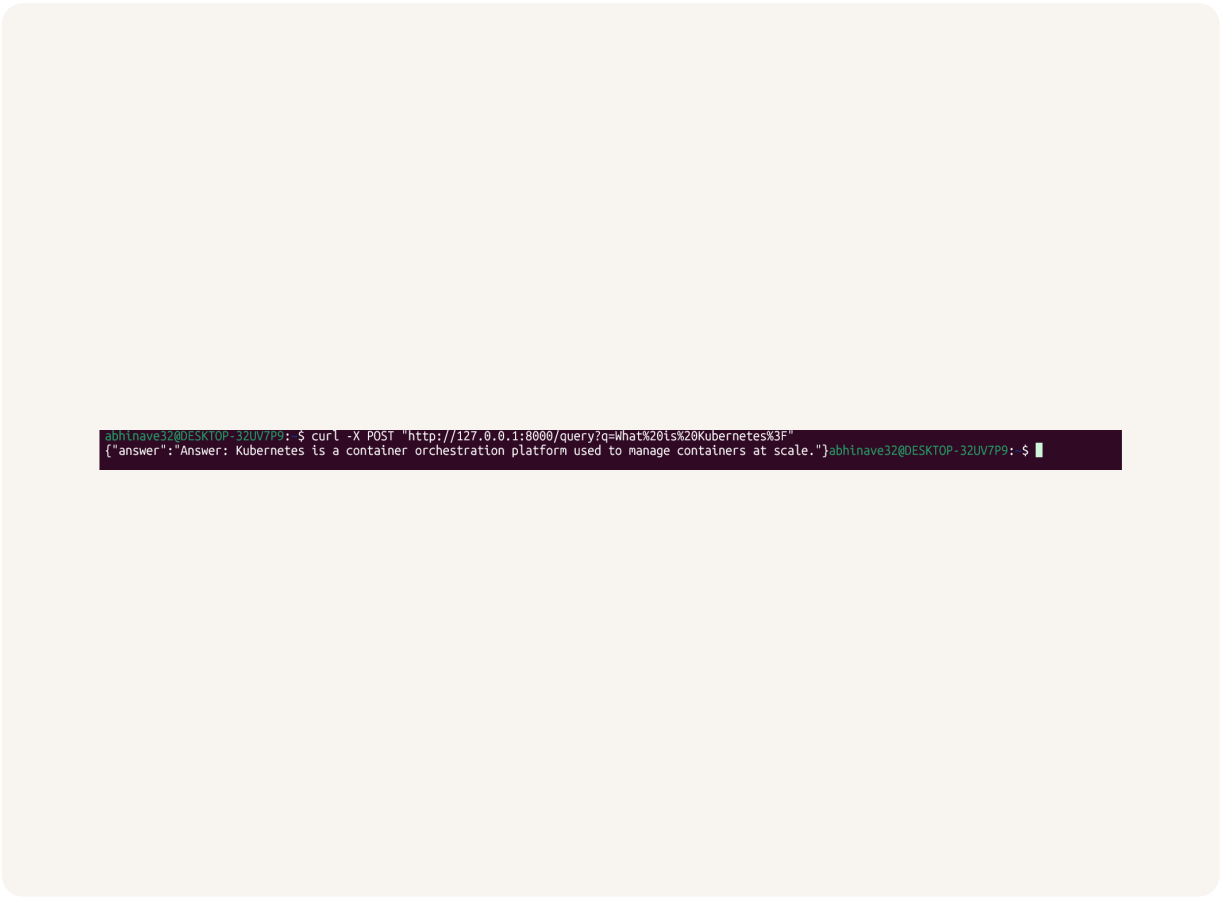
Containerized API test results

Testing the API after containerization proved that the FastAPI application runs consistently with all its dependencies packaged, but it exposed issues like host-specific services, e.g., Ollama, not being directly reachable from inside the container. The difference between running locally and in Docker is that locally the app can access services on localhost directly, whereas inside Docker, the container is isolated and requires host.docker.internal to reach host services.

Abhinave P.B
NextWork Student

Containerization helps because it ensures reproducibility across environments, eliminates dependency conflicts, simplifies deployment, and allows anyone to run the API without manually installing Python, libraries, or databases, making collaboration and scaling much easier.

Abhinave P.B
NextWork Student

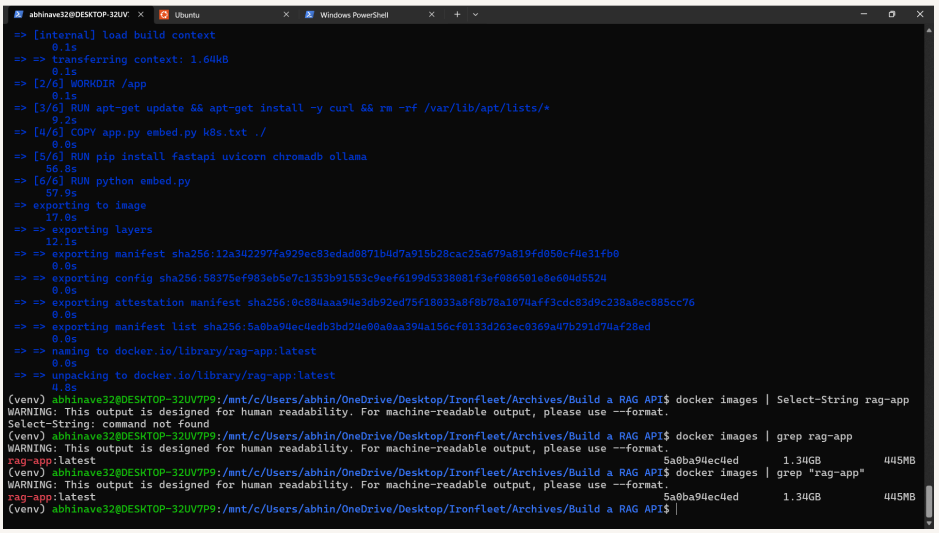


Abhinave P.B
NextWork Student

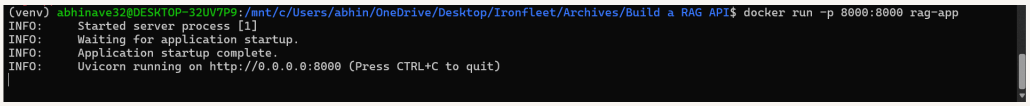
Building and Running the Container

Docker image build complete

Building a Docker image involves writing a Dockerfile that defines the base environment, installing all required dependencies, copying the application code, and executing necessary setup steps so the application can run consistently inside a container. I verified my Docker image was built successfully by listing the available Docker images and confirming that the rag-app:latest image appeared with a valid image ID and size. This confirms that my API is now containerized because the entire application, its dependencies, and the precomputed embeddings are packaged into a single Docker image that can be run consistently on any system using Docker.



Abhinave P.B
NextWork Student



Abhinave P.B
NextWork Student

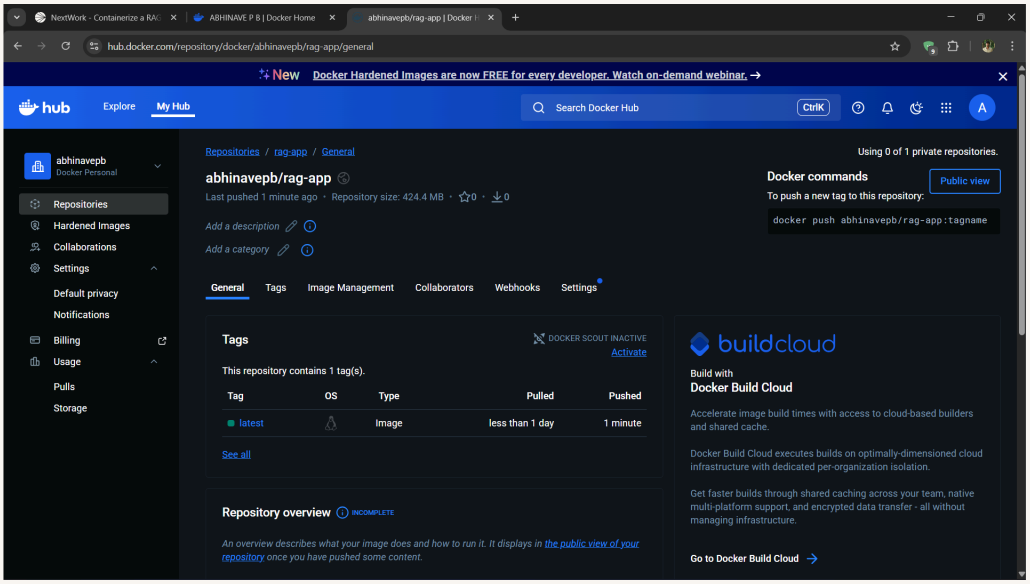
Pushing to Docker Hub

In this project extension, I'm pushing to Docker Hub. Docker Hub is Docker Hub is a cloud-based registry where you can store and share Docker images I'm doing this because i want to learn how to share docker images through docker hub

Docker Hub push complete

I pushed to Docker Hub by tagging my local Docker image with my Docker Hub repository name and then using docker push to upload it. Docker Hub is useful because it allows me to store, share, and manage Docker images in a central, cloud-based repository. The advantage of pushing to a registry is that it enables easy distribution of images, ensures consistency across environments, supports versioning, and allows teams or deployment pipelines to pull and run the exact same image anywhere.

Abhinave P.B
NextWork Student



Pulling from Docker Hub

Pulling an image from Docker Hub means downloading a pre-built Docker image from Docker's online repository to your local machine. When I ran docker pull, Docker fetched the image layers from Docker Hub and stored them locally so I can run containers from it. The difference between building locally and pulling from Docker Hub is that building locally creates a new image from your Dockerfile on your machine, while pulling from Docker Hub uses an already-built image created by someone else.

Abhinave P.B

NextWork Student


```
(venv) abhinave32@DESKTOP-32UV7P9:/mnt/c/Users/abhin/OneDrive/Desktop/Ironfleet/Archives/Build a RAG API$ docker pull abhinavepb/rag-app
Using default tag: latest
latest: Pulling from abhinavepb/rag-app
b4f293182552: Pull complete
5b04959817b2: Pull complete
0f55b00a1c54: Pull complete
f68ee924fd03: Download complete
Digest: sha256:2e281b6fd4ec7cc52c0ff1afd212b64054bb9f45ad99f8b31a21836843686217
Status: Downloaded newer image for abhinavepb/rag-app:latest
docker.io/abhinavepb/rag-app:latest
(venv) abhinave32@DESKTOP-32UV7P9:/mnt/c/Users/abhin/OneDrive/Desktop/Ironfleet/Archives/Build a RAG API$
```

The place to learn & showcase your skills

Check out nextwork.org for more projects