Docker for AI Apps: Website Course Description

Course Overview

AI applications that work on a developer's machine often fail to run anywhere else. Different Python versions, mismatched dependencies, missing environment variables, and undocumented system requirements make deployment unpredictable. Docker solves this problem by packaging an application together with everything it needs to run inside a container that behaves identically on any machine. This course teaches you how to build, configure, and deploy AI applications using Docker, from first principles through a complete production-style multi-service stack.

Rather than teaching Docker in isolation using generic examples, this course introduces every concept through a real AI application. Images and containers are taught through a Python CLI chatbot. Volumes are taught by persisting conversation history. Networking is taught by wiring a chatbot to Redis. Docker Compose is taught by managing a three-service AI stack. Every Docker concept is grounded in the context where AI developers actually use it.

The course closes with a capstone project: a Document Q&A system with four containerized services, including FastAPI, ChromaDB, Redis, and Nginx, managed by Docker Compose, following all production best practices from Chapter 10. By the end, you will have built and containerized five AI applications and have the skills to package, run, optimize, and ship any AI application using Docker.

Course Objectives

By the end of this course, you will be able to:

- Explain what Docker is, how it differs from virtual machines, and why it solves the environment reproducibility problem for AI applications.
- Write Dockerfiles that build fast, stay small, and run predictably using layer caching, .dockerignore, and appropriate base images.
- Pass API keys and secrets to containers securely at runtime without storing them in images or source control.

- Persist AI application data across container stops and recreations using named Docker volumes.
- Connect multiple containers using user-defined Docker networks with automatic DNS-based service discovery.
- Manage a multi-service AI stack with Docker Compose using a single YAML file and a single command.
- Build a production-ready AI REST API with FastAPI, Redis-backed session management, HEALTHCHECK, and SSE streaming.
- Reduce an AI image from over one gigabyte to under two hundred megabytes using multi-stage builds and a non-root user.
- Scan images for vulnerabilities, version them with semantic versioning, and push them to Docker Hub.

What You Will Learn

- Why Docker exists, what problem it solves for AI development, and how containers differ from virtual machines.
- How to install and configure Docker Desktop on Windows with WSL 2, macOS, and Linux.
- The essential Docker CLI skills: pulling images, running containers, using key flags, inspecting, debugging, and cleaning up.
- How to write a Dockerfile from scratch for a Python application, including base image selection, layer caching, and instruction ordering.
- How to containerize a CLI chatbot that calls the OpenAI API with correct secrets management using environment variables.
- How to use named Docker volumes and bind mounts to persist AI application data across container restarts.
- How Docker networking works, how user-defined networks provide DNS-based service discovery, and how to wire containers together.
- How to declare and manage a multi-container AI stack with Docker Compose, including services, networks, volumes, and environment configuration.
- How to build a FastAPI AI REST API with session management in Redis, HEALTHCHECK, and SSE streaming, all running in a Compose stack.
- How to apply Docker best practices: multi-stage builds, non-root user, .dockerignore, docker scout vulnerability scanning, and Docker Hub publishing.
- How to build a four-service Document Q&A capstone application using FastAPI, ChromaDB, Redis, and Nginx.

Prerequisites

- Python fundamentals: variables, functions, loops, dictionaries, and file input and output.
- Basic command line comfort: running scripts, installing packages, navigating directories.
- An OpenAI API key with billing configured (used from Chapter 5 onwards).
- No prior Docker experience is required. The course starts from zero.
- No experience with FastAPI, Redis, ChromaDB, or Nginx is required.

Who This Course Is For

- Python developers who want to deploy AI applications reliably across any machine, server, or environment.
- AI engineers who have hit the 'works on my machine' wall and want a permanent, professional solution.
- Developers building multi-service AI stacks who need to manage them with Docker Compose.
- Students and professionals learning production-ready AI deployment skills from the ground up.
- Anyone who wants to go from zero Docker knowledge to a complete, containerized, multi-service AI application.

Course Syllabus

Chapter	Topics Covered	Outcome
Chapter 1 Why Docker for AI?	The 'works on my machine' problem. Containers vs virtual machines. How Docker's packaging model solves dependency and reproducibility failures in AI development. Why Docker is the right tool for AI app deployment.	Understand exactly what Docker is, why it exists, and why it matters for AI.
Chapter 2 Installation & Setup	Docker Desktop installation on Windows with WSL 2, macOS Apple Silicon and Intel, and Docker Engine on Linux. Memory limits, verification, and supporting	Docker installed, configured, and verified. Able to run docker run hello-world.

	tools: VS Code with Docker extension, Python 3.11, curl, Postman. Common errors and fixes.	
Chapter 3 Docker Fundamentals	Images vs containers. Full container lifecycle: create, start, stop, remove. Key docker run flags: -d, -it, -p, -e, --name, --rm. Inspecting, debugging, and cleaning up containers and images. Every command practiced, not just shown.	Confident with the Docker CLI. Able to pull, run, exec into, stop, and remove containers.
Chapter 4 Writing Dockerfiles	FROM, WORKDIR, COPY, RUN, ENV, EXPOSE, CMD, ENTRYPOINT. Layer caching and .dockerignore. Choosing between python:3.11, python:3.11-slim, and python:3.11-alpine. Instruction order for fast builds. Project: containerized Flask web server.	Write a Dockerfile from scratch, build an image, run it as a web server.
Chapter 5 First Dockerized AI App	Multi-turn CLI chatbot using the OpenAI Chat Completions API. Secrets management: passing API keys with -e and --env-file without baking them into the image. PYTHONUNBUFFERED=1 for real-time output. Running containers in interactive mode with -it.	Containerized CLI chatbot with multi-turn history and correct secrets management.
Chapter 6 Volumes & Persistent Storage	Why containers are stateless by default. Docker volumes and bind mounts. When to use each. Persisting conversation history with a named volume. Upgrading the CLI chatbot to save and reload history across container restarts.	Conversation history survives container stops, removals, and recreations.
Chapter 7 Docker Networking	Bridge, host, and none network drivers. User-defined networks and DNS-based service	Two-container AI app communicating by name on a private network.

	discovery. Port publishing. Project: two-container AI app with the chatbot talking to Redis over a private Docker network.	
Chapter 8 Docker Compose	Why docker run commands do not scale. Compose YAML: services, image vs build, depends_on, restart policies, env_file, named volumes. Project: three-service stack with FastAPI chatbot, Redis, and Nginx managed with docker compose up -d.	Three-service AI stack running from a single Compose file.
Chapter 9 Dockerizing an AI REST API	FastAPI with a /chat endpoint and async OpenAI calls. Redis-backed session management keyed by session_id. HEALTHCHECK instruction and Docker health states. Server-Sent Events SSE streaming. Testing with curl and a browser frontend.	Containerized AI REST API with session history, streaming, and health monitoring.
Chapter 10 Best Practices & Optimization	Multi-stage builds to shrink image size. Layer caching optimization. Running as a non-root user. .dockerignore hygiene. Vulnerability scanning with docker scout. Semantic versioning and pushing images to Docker Hub.	FastAPI AI app rebuilt from 1+ GB to under 200 MB, non-root, versioned, and on Docker Hub.
Chapter 11 Capstone: Document Q&A	Four-container Document Q&A system: FastAPI, ChromaDB for vector storage, Redis for session history, and Nginx as reverse proxy. Wired with Docker Compose. All best practices from Chapter 10 applied. Versioned image pushed to Docker Hub. README that lets anyone run the full stack with one command.	Complete, working Document Q&A AI system running in Docker Compose.

Skills You Will Gain

- Dockerfile authoring: writing reproducible, optimized build specifications for Python AI applications.
- Secrets management: passing API keys and environment variables to containers at runtime without storing them in images.
- Volume management: persisting AI application data across container restarts using named volumes.
- Docker networking: connecting containers on user-defined networks with DNS-based service discovery.
- Docker Compose: declaring and managing multi-service AI stacks in a single YAML file.
- Image optimization: applying multi-stage builds and best practices to reduce image size by 80 percent or more.
- Production readiness: implementing HEALTHCHECK, non-root user, .dockerignore, and vulnerability scanning.
- Docker Hub publishing: versioning images with semantic versioning and pushing to a public registry.
- End-to-end deployment: packaging, running, and shipping a complete multi-service AI application using Docker.