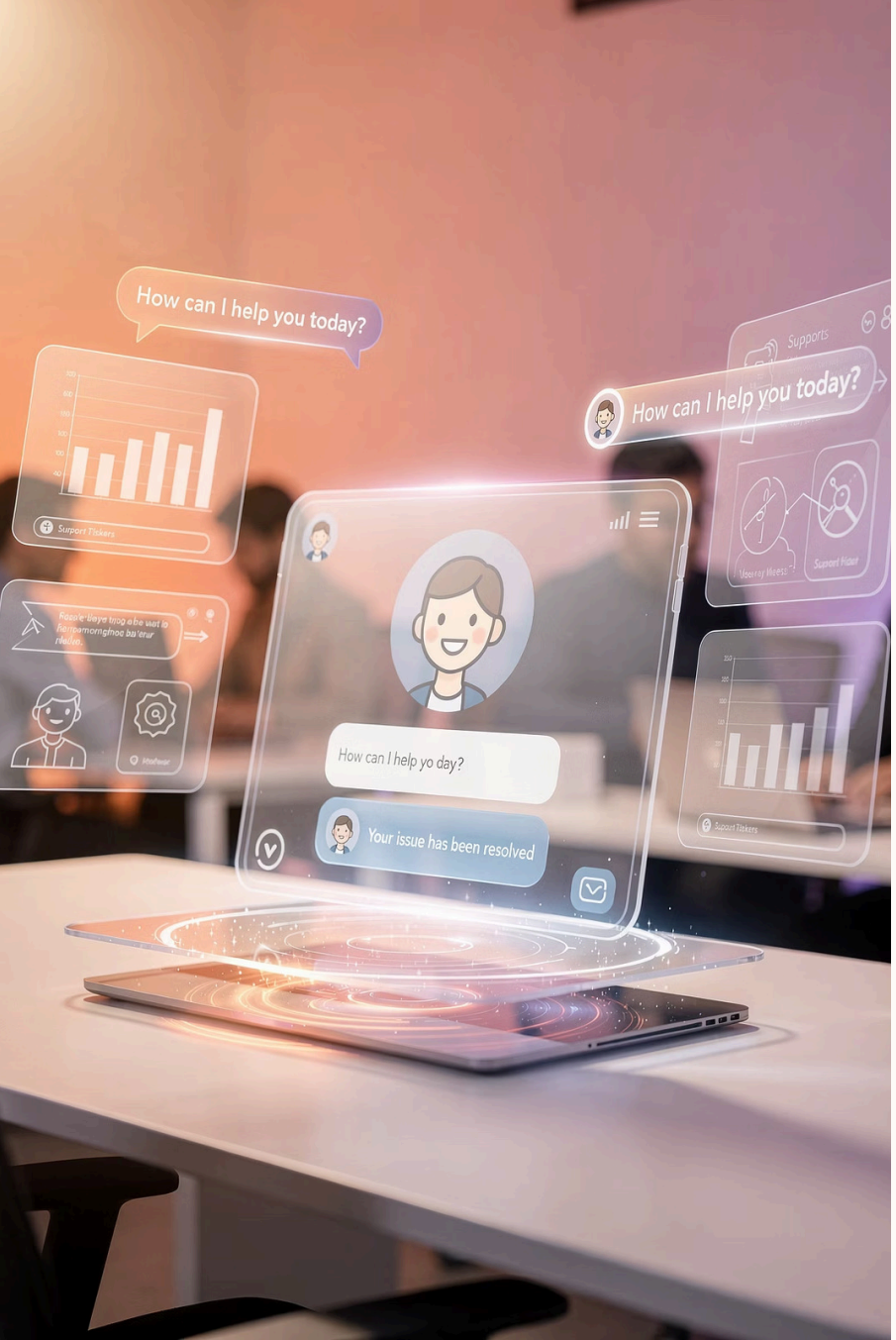


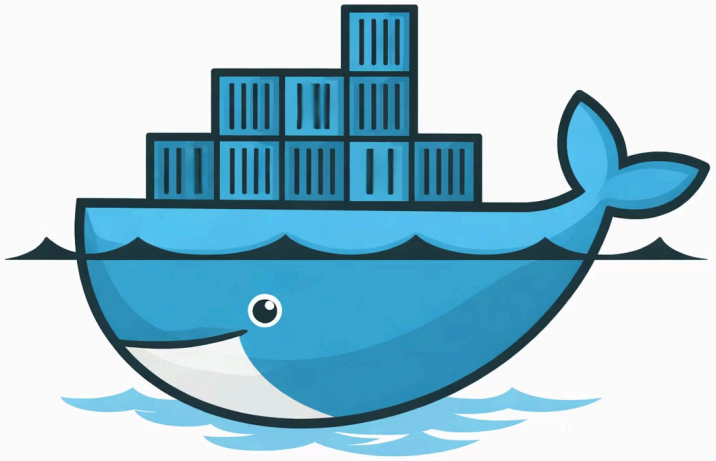
使用 Docker 建立 AI 大型軟體

以 RAG AI 客服系統為例

Docker + LLM + Vector DB 架構設計實戰指南

智慧科技產研中心

陳啟彰　教授



甚麼是Docker?

Docker 是一個開源專案，出現於 2013 年初，最初是 Dotcloud 公司內部的 Side-Project。（ Dotcloud 公司後來改名為 Docker ）

Docker = 把程式 + 所有依賴環境打包成一個「可攜式盒子」

這個盒子叫做 **Container（容器）**。

容器化（Containerization）核心概念

作業系統層級虛擬化

封裝程式碼、函式庫與環境設定，形成可獨立運行的單元

共用主機核心

秒級啟動速度，資源使用效率遠超傳統虛擬化方案

解決環境差異

終結「在我電腦跑得好，為甚麼在你的電腦或伺服器上卻不行?」的經典問題

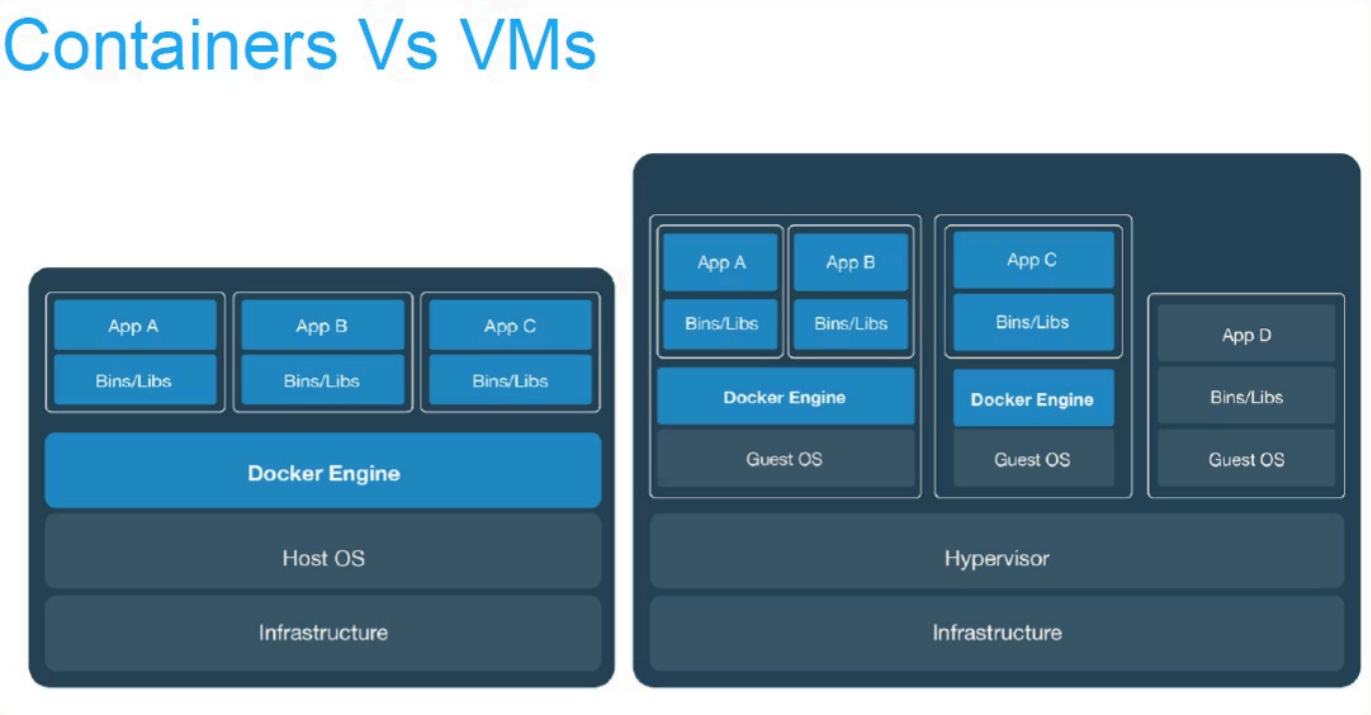
容器 vs 虛擬機（VM）大對決

🖥️ 虛擬機（VM）

包含完整作業系統，啟動需數分鐘，每台 VM 獨立佔用大量 CPU 與記憶體資源，適合需要強隔離的傳統企業系統

📦 容器（Container）

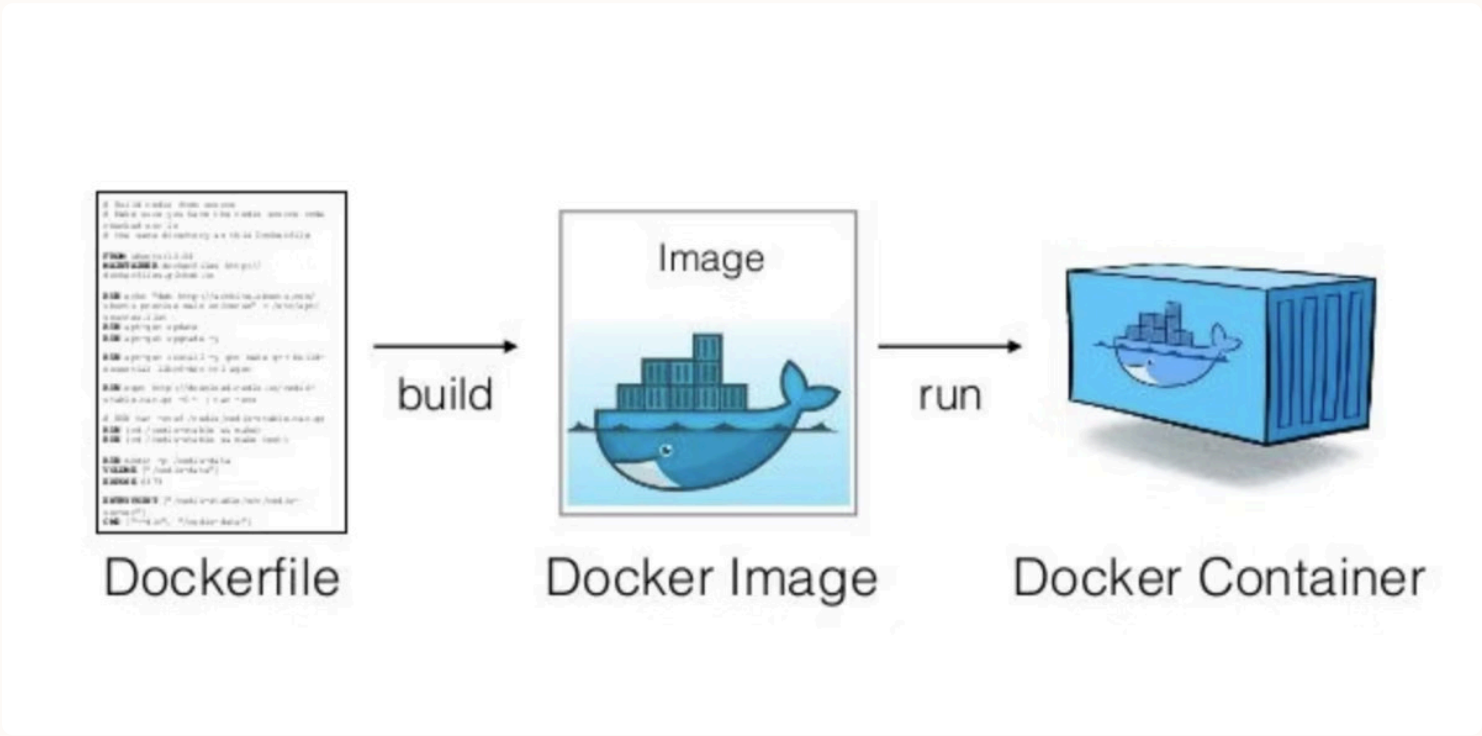
共用主機 OS 核心，**秒級啟動**，輕量高效，非常適合微服務架構與雲端原生部署



- Hypervisor: 管理VM的軟體, eg. VMware, VirturalBox
- Docker Engine: 管理Container的軟體

架構解析

Docker 架構與運作流程



- Dockerfile 主要做三件事：

1. 指定基礎環境
2. 安裝需要的套件
3. 設定程式如何啟動

Client-Server 架構

Docker Client 發送指令，Docker Daemon 在背景負責管理所有容器的生命週期與資源分配

Docker Hub

全球最大的雲端Image倉庫，可拉取官方映像或推送自製映像，實現版本管控與團隊協作

Docker 的核心優勢

環境一致性

開發、測試、生產環境無縫銜接，消除環境差異引發的問題

輕量快速

秒級啟動，大幅節省硬體資源，提升團隊交付速度

高度隔離

容器間互不干擾，安全穩定，故障不會蔓延至其他服務

高可攜性

同一容器跨平台執行，無需修改，真正實現「一次建構，到處執行」

AI 客服系統的挑戰：為什麼需要 RAG？

傳統聊天機器人的限制

- 回答不準確，缺乏上下文理解
- 無法有效運用企業知識庫資源
- 內容更新困難，維護成本高
- 對專業問題的處理能力不足

RAG 解決方案

Retrieval Augmented Generation（檢索增強生成）技術透過兩階段處理提升回答品質：

01

檢索階段

從知識庫中搜尋相關文件與資料

02

生成階段

將檢索結果交給 LLM 生成精確回答

RAG AI 客服系統整體架構

本系統採用模組化設計，整合多項核心技術元件，建構完整的企業級 AI 客服解決方案。

1	**ai-bot（前台介面）** 提供使用者互動的 Web 介面，接收問題並呈現回答結果
2	**ai-manager（後台 API）** 核心服務層，負責 RAG Pipeline 執行與知識庫管理
3	**Ollama + gpt-oss:20b (另建一個volumn)** 本地部署的大型語言模型，提供自然語言生成能力 (Note: volumn是一種用來保存資料的儲存空間, 讓資料不會因containser被刪除而消失)
4	**Postgres 資料庫** 儲存使用者資料、對話記錄與系統設定
5	**Milvus 向量資料庫** 執行高效能語意搜尋，實現知識庫檢索功能

為什麼 RAG 系統需要 Docker？

AI 系統的複雜特性

多服務架構

前台、後台、資料庫、LLM 多個服務協作

GPU 與 DB 混合

需要同時管理運算資源與資料存取

環境依賴複雜

各服務有不同的程式庫與設定需求

Docker 提供的解決方案

容器化技術為 RAG 系統帶來關鍵優勢：

- **模組化部署**：每個服務獨立打包，易於管理與更新
- **快速移植**：在開發、測試、正式環境間無縫切換
- **可重現執行環境**：確保所有環境的一致性，減少「在我電腦上可以跑」的問題
- **資源隔離**：避免服務間的資源競爭與衝突

Docker 在 RAG 系統中的角色

隔離 LLM 推論環境

確保模型運行的穩定性與獨立性

管理 Vector DB

簡化向量資料庫的部署與維護

統一 API 服務

提供一致的服務介面與通訊機制

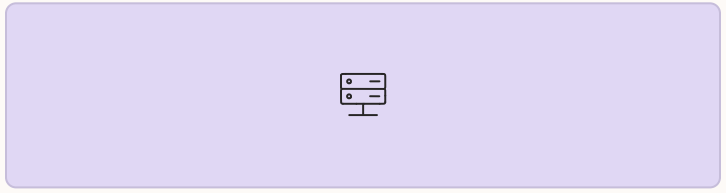
> **核心概念**：Docker 是 RAG 系統的基礎運行平台，提供完整的容器化解決方案，讓複雜的 AI 系統變得可管理、可擴展、可重現。

RAG 系統 Container 分層設計

採用三層架構模式，清楚劃分職責，提升系統的可維護性與擴展性。

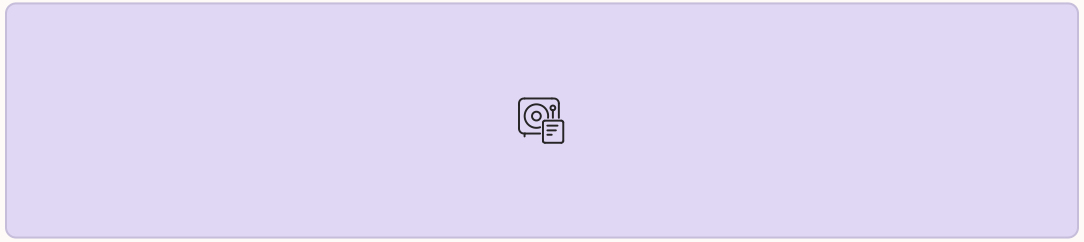
Front Layer

ai-bot 前台介面層，處理使用者互動



Service Layer

ai-manager 服務層，執行核心業務邏輯與 RAG Pipeline



Infrastructure Layer

Ollama、Postgres、Milvus 基礎設施層，提供 AI 推論、資料儲存與向量搜尋能力

這種分層設計遵循關注點分離原則，讓每一層專注於特定職責，降低耦合度，便於獨立擴展與升級各個元件。

Docker Network 設計：雙網路安全架構

採用雙網路隔離設計，強化系統安全性與資料保護。

前端網路（front_net）

連接 ai-bot 與 ai-manager，處理使用者請求與前後台通訊。

後端網路（back_net）

連接 ai-manager 與 LLM、資料庫，處理 AI 推論與資料存取。

安全原則：前台不能直接存取資料庫與 LLM，所有 AI 操作與資料查詢都必須透過 ai-manager 集中管理，確保資料安全與存取控制。

ai-bot Container 設計

核心功能

Web 前台 UI

提供直覺的使用者介面

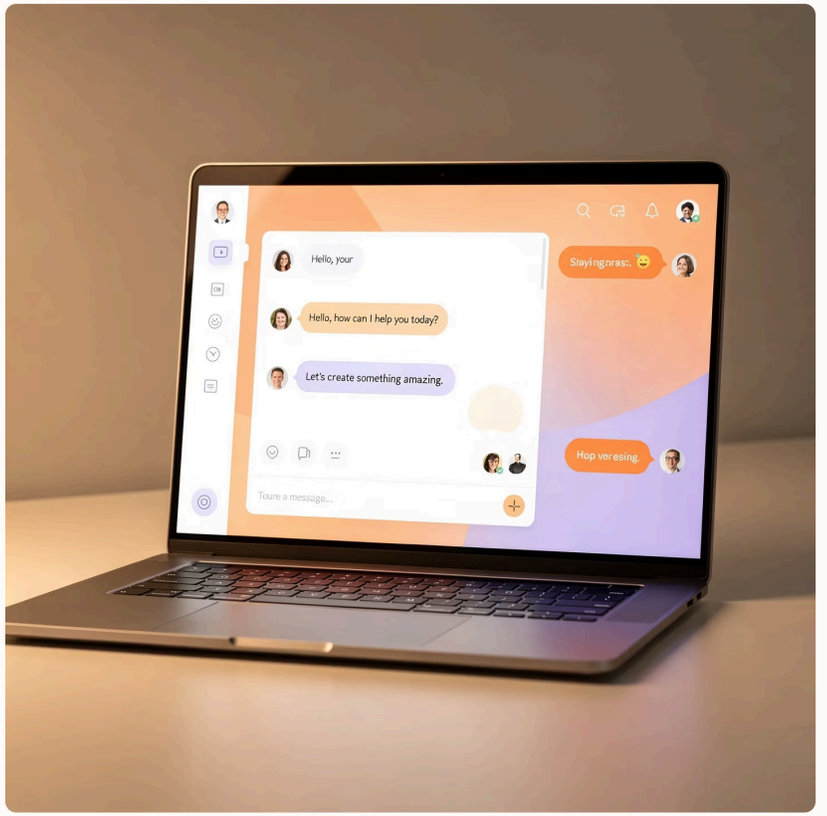
接收使用者問題

處理輸入與對話管理

呼叫後台 API

轉發請求至 ai-manager

ai-bot 作為輕量級前端容器，專注於使用者體驗與介面互動，不處理任何 AI 推論或資料庫操作，確保前後端職責清晰分離。



設計原則

- 不包含 LLM 模型
- 不直接存取 Vector DB
- 不連接資料庫
- 僅透過 API 與後台溝通

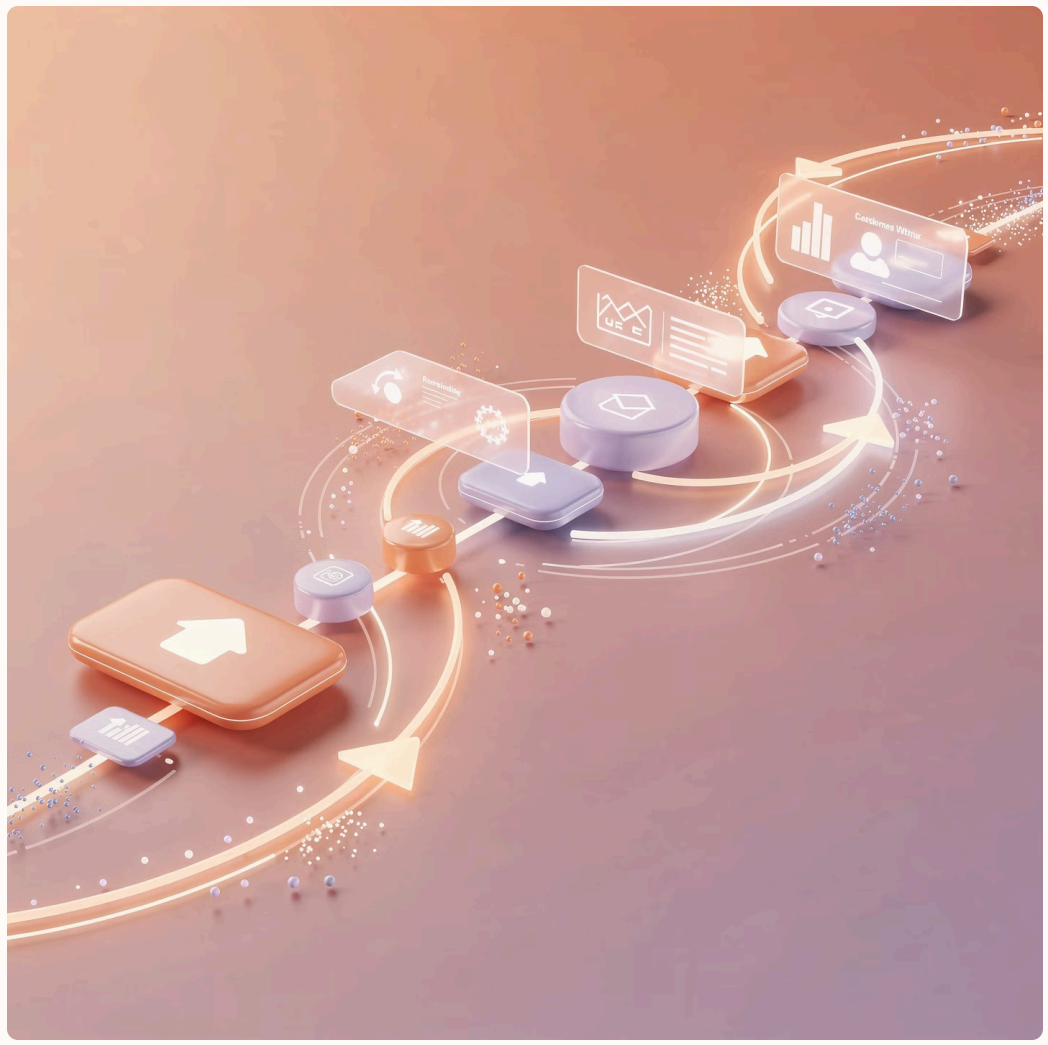
ai-manager Container：RAG Pipeline 核心

ai-manager 是整個 RAG 系統的大腦，負責協調所有 AI 相關操作。

1. **Query 接收**
   從 ai-bot 接收使用者問題
2. **Embedding 轉換**
   將問題轉換為向量表示
3. **Milvus 檢索**
   在向量資料庫中搜尋相關知識
4. **Context 組裝**
   整合檢索結果建立上下文
5. **LLM 生成**
   呼叫 Ollama 產生最終回答

核心職責

- 實作完整 RAG Pipeline 流程
- 管理與 Milvus 的向量搜尋互動
- 協調 Ollama LLM 的推論請求
- 知識庫的匯入、更新與管理



Ollama + gpt-oss:20b Container

設計原則

使用官方 Image

採用 Ollama 官方容器映像檔，確保穩定性與相容性

模型儲存在 Volume

將 20B 參數的 gpt-oss 模型持久化保存，避免重複下載

GPU 資源可選配置

支援 GPU 加速推論，也可在 CPU 模式下運行

架構考量：將 LLM 與 Web 服務分離部署，避免資源競爭與衝突。LLM 推論需要大量記憶體與運算資源，獨立容器設計讓資源管理更靈活，也便於未來升級或替換不同的模型。

Milvus Vector Database：RAG 的核心基礎

Milvus 是專為 AI 應用設計的開源向量資料庫，提供高效能的語意搜尋能力。

核心用途

儲存知識庫 Embedding

將企業文件、FAQ、產品資訊轉換為向量並儲存

支援語意搜尋

透過向量相似度計算，找出最相關的知識片段

高效能檢索

毫秒級回應時間，支援大規模向量搜尋

子服務架構

Milvus 依賴兩個關鍵子服務：

- **etcd**：分散式元資料儲存，管理 Milvus 的配置與協調
- **MinIO**：物件儲存服務，持久化向量資料與索引檔案

這三個容器共同構成完整的向量資料庫解決方案，是 RAG 檢索功能的技術基石。

Postgres 資料庫：結構化資料管理

使用者資料

儲存使用者帳號、權限、偏好設定等結構化資訊

對話記錄

保存完整的問答歷史，支援追蹤與分析

系統設定

管理 API 金鑰、模型參數、系統配置等

與 Milvus 的職責分工

Postgres：負責結構化關聯式資料，適合儲存需要複雜查詢與關聯的業務資料。

Milvus：專注於向量資料與語意搜尋，提供高維度向量的高效檢索能力，是 RAG 知識庫的專用儲存。

docker-compose.yml 架構設計

透過 Docker Compose 統一管理所有服務，實現一鍵部署與環境一致性。

Services 定義

宣告所有容器服務：ai-bot、ai-manager、ollama、postgres、milvus 及其依賴

Networks 配置

建立 front_net 與 back_net 雙網路隔離架構

Volumes 管理

定義持久化儲存：模型檔案、資料庫資料、向量索引

關鍵優勢

- **一鍵啟動**：單一指令啟動整個 RAG 系統的所有服務
- **環境一致**：確保本地開發、測試、正式環境的配置完全相同
- **依賴管理**：自動處理服務間的啟動順序與依賴關係
- **易於維護**：集中管理所有配置，降低維運複雜度

RAG AI 系統部署流程

從零開始建構完整的 Docker 化 RAG 系統，遵循以下標準化流程。

1

建立 Dockerfile

為 ai-bot 與 ai-manager 撰寫容器建構檔案，定義執行環境與依賴

2

設計 docker-compose.yml

整合所有服務配置，包含網路、儲存卷與環境變數

3

執行 docker compose up

啟動所有容器服務，建立網路與儲存卷

4

下載 gpt-oss:20b 模型

透過 Ollama CLI 拉取 20B 參數的語言模型

5

匯入知識庫

將企業文件轉換為 embedding 並儲存至 Milvus

6

測試 RAG 功能

驗證完整的檢索增強生成流程，確保系統正常運作

整個部署流程標準化且可重現，大幅降低環境建置的時間與錯誤率。

Docker 化 RAG 系統的核心優勢

技術優勢

快速部署企業 AI 客服

從零到上線僅需數小時，大幅縮短開發週期

模型與服務解耦

可獨立升級 LLM 或更換其他模型，不影響其他服務

易於水平擴展

透過容器編排輕鬆擴展服務實例，應對流量成長

架構優勢

API Gateway 模式

統一入口管理，簡化服務間通訊與安全控制

微服務設計

每個功能模組獨立運作，提升系統彈性與可維護性

安全隔離機制

網路分層設計確保資料安全與存取控制

> Docker 為 RAG AI 系統提供了完整的容器化解決方案，讓複雜的 AI 架構變得可管理、可擴展、可重現，是建構現代企業級 AI 應用的最佳實踐。

End.

Thank You!!