Ollama + ChromaDB Local Ingestion Pipeline Explained

This document explains, step by step, how a local DevOps knowledge base is built using Ollama for embeddings and ChromaDB as a vector database. The goal is to prepare high-quality retrieval data for an autonomous DevOps agent.

Step 1 – Installing Dependencies

Python libraries are installed to handle vector storage (ChromaDB), PDF text extraction (PyPDF), HTTP communication with Ollama (requests), and progress visibility (tqdm). These components together form the ingestion pipeline.

Step 2 – Running Ollama Locally

Ollama runs as a local service exposing HTTP APIs. It acts as an embedding and inference server, allowing the system to generate vectors without sending data to the cloud.

Step 3 – Pulling an Embedding Model

An embedding model such as nomic-embed-text is downloaded. This model converts text into numerical vectors that capture semantic meaning, which is essential for similarity search.

Step 4 – Project Directory Structure

A clean directory layout separates source PDFs, persisted vector data, and ingestion code. This mirrors infrastructure best practices and makes the system easy to maintain.

Step 5 – Configuration Settings

Paths, model names, and service URLs are defined at the top of the script. This centralizes configuration and allows easy tuning without changing core logic.

Step 6 – Chunking Strategy

Documents are split into overlapping chunks of fixed size. Chunking ensures that embeddings stay within model limits while preserving enough context for accurate retrieval.

Step 7 – Collection Mapping

Each PDF is mapped to its own Chroma collection. This allows domain-specific searches and enables the agent to query only the most relevant knowledge base.

Step 8 – Initializing ChromaDB

A local Chroma client is created with persistence enabled. This ensures embeddings and metadata are stored on disk and reused across restarts.

Step 9 – Reading PDF Content

PDFs are processed page by page, extracting readable text while skipping empty pages. Page numbers are retained as metadata for traceability.

Step 10 – Creating Chunks

Each page’s text is divided into overlapping chunks using a sliding window approach. This avoids losing information that falls at chunk boundaries.

Step 11 – Stable ID Generation

Each chunk receives a deterministic hash-based ID. This prevents duplication and allows safe re-ingestion when documents are updated.

Step 12 – Generating Embeddings

Each chunk is sent to Ollama’s embedding API. The model returns a vector representation that captures the semantic meaning of the text.

Step 13 – Storing Data in ChromaDB

Chunks, embeddings, and metadata are stored together. This enables semantic similarity search combined with metadata filtering.

Step 14 – Persisting the Database

The vector index and metadata are flushed to disk. This step finalizes ingestion and ensures durability.

Step 15 – Resulting System

The final result is a fully local, cloud-free vector database powering a DevOps agent capable of accurate, domain-aware semantic search.