The Ultimate Guide: Building & Deploying a Private RAG AI Assistant on a VPS

Introduction

This guide details the complete process for building and deploying a private AI application. The system uses a self-hosted language model to analyze and answer questions about your own documents, ensuring all data remains secure on a server you control.

We will cover two major phases:

1. **Local Setup & Testing:** Building all components and verifying they work together on a local macOS machine.
2. **Cloud Deployment:** Migrating the entire application to a cloud server (VPS) and making it securely accessible on the internet.

Final Architecture Overview

The system is composed of four main containerized services managed by Docker Compose, ensuring stability and separation of concerns.

Code snippet

```
graph TD
    subgraph "User's Computer"
        User[<fa:fa-user> User in Browser]
    end

    subgraph "Cloud Server / VPS"
        Firewall{{<fa:fa-fire> ufw Firewall}}

        subgraph "Docker Environment"
            direction TB

            subgraph "Gateway / Edge Layer"
                Caddy(Caddy Reverse Proxy)
            end

            subgraph "Internal Application Services"
                Streamlit(Streamlit App Container)
                N8N(n8n App Container)
                LLM(LLM Server Container)
            end

            subgraph "Persistent Data (Linked via Bind Mounts)"
                RAGScripts[<fa:fa-code> RAG Scripts]
                Docs[<fa:fa-file-pdf> Docs]
                Models[<fa:fa-microchip> LLM File]
                ChromaDB[<fa:fa-database> Vector DB]
                LIStorage[<fa:fa-database> Index Storage]
            end

            %% Internal Connections
            Streamlit --> N8N
            N8N --> RAGScripts
            RAGScripts --> ChromaDB
            RAGScripts --> LIStorage
            RAGScripts --> LLM
            LLM -.-> Models
            RAGScripts -- Reads --> Docs
        end

        %% External Flow
        User -- "HTTPS Request" --> Firewall
        Firewall -- "Allows Port 443" --> Caddy
        Caddy -- "Routes to Streamlit" --> Streamlit
    end
```

Part 1: Local Machine Preparation (macOS)

This section covers setting up your project folder and all necessary files on your local computer.

Step 1.1: Install Prerequisites

If you don't already have them, install Apple's Xcode Command Line Tools and the Homebrew package manager.

1. **Xcode Tools:** Open your terminal and run `xcode-select --install`.
2. **Homebrew:** Follow the installation instructions on the official Homebrew website.
3. **CMake:** Use Homebrew to install CMake, a necessary build tool.
4. Bash

```
brew install cmake
```

5.
6.

Step 1.2: Create the Project Directory

A clean folder structure is essential. On your Desktop or another preferred location, create your main project folder `AI_LawFirmProject` and the required sub-folders.

```
/AI_LawFirmProject/
|-- docs/
|-- models/
`-- rag_scripts/
```

- **docs/**: Place the .pdf or .txt files you want the AI to learn from here.
- **models/**: Download your language model file (e.g., Phi-4-mini-instruct.Q8_0.gguf) and place it here.
- **rag_scripts/**: Your core Python scripts will go here.

Step 1.3: Create All Code & Config Files

Create all the text-based files in your AI_LawFirmProject folder. You can use any code editor (like VS Code) or the nano command in the terminal. The complete, final code for each file is listed in the **Appendix** at the end of this guide.

- Dockerfile (for the n8n service)
- llm-server.Dockerfile (for building the LLM server)
- streamlit.Dockerfile (for the Streamlit UI)
- docker-compose.yml (the master orchestration file)
- Caddyfile (for the reverse proxy)
- app.py (the Streamlit UI script)
- requirements.txt (Python libraries)
- rag_scripts/rag_setup.py
- rag_scripts/query_rag.py

Step 1.4: Set Up Python Virtual Environment

This creates an isolated environment for your Python packages, preventing conflicts with other projects.

1. In your terminal, navigate into your AI_LawFirmProject directory.
2. Create the environment folder (named venv):
3. Bash

```bash
python3 -m venv venv
```

4.
5.
6. Activate it. Your terminal prompt will now be prefixed with (venv).
7. Bash

```bash
source venv/bin/activate
```

8.

9.
10. Install all required libraries into this environment:
11. Bash

```
pip install -r requirements.txt
pip install streamlit
```

12.
13.

At this point, your local machine is fully prepared. All code is written, and all local dependencies are installed.

Part 2: Cloud Server (VPS) Provisioning & Setup

Now, we will rent and configure a clean cloud server to host our application.

Step 2.1: Get a VPS

1. **Sign up** for a cloud provider like **DigitalOcean**.
2. **Create a Droplet** (their name for a VPS) with these specifications:
   - **Region:** A data center near you (e.g., Toronto, New York).
   - **OS: Ubuntu 22.04 (LTS) x64**.
   - **Plan:** Choose a **Regular / Shared CPU** plan with at least **8 GB of RAM**.
   - **Authentication:** Select **SSH Key**. Follow the on-screen instructions to add your public SSH key from your Mac (located at `~/.ssh/id_rsa.pub`). This is far more secure than a password.
3. Click **Create Droplet**. After about a minute, your server will be live. Copy its public **IP Address**.

Step 2.2: Initial Server Security

1. **Log in as `root`:** From your Mac's terminal, connect to the server.
2. Bash

```
ssh root@YOUR_VPS_IP_ADDRESS
```

3.
4. Type `yes` to accept the host authenticity fingerprint on first connection.
5. **Create a new, non-root user** (replace `daniel` with your desired username):
6. Bash

```
adduser daniel
```

7.
8. Enter a strong password and press Enter through the other prompts.
9. **Give your new user sudo (administrator) privileges:**
10. Bash

```
usermod -aG sudo daniel
```

11.
12.
13. **Copy your SSH key to the new user** so you can log in directly:
14. Bash

```
rsync --archive --chown=daniel:daniel ~/.ssh /home/daniel
```

15.
16.
17. **Log out and log back in as your new user.** This is crucial for all subsequent steps.
18. Bash

```
exit
ssh daniel@YOUR_VPS_IP_ADDRESS
```

19.
20.

Step 2.3: Configure the Firewall

Set up the ufw (Uncomplicated Firewall) to only allow essential traffic.

Bash

```
# Allow SSH connections (so you can log in)
sudo ufw allow OpenSSH
# Allow web traffic for Caddy
sudo ufw allow http
sudo ufw allow https
# Enable the firewall
sudo ufw enable
```

Press y to confirm. Your server is now firewalled.

Part 3: Deploying the Application

Now we will install Docker, transfer our project, and launch the application.

Step 3.1: Install Docker and Docker Compose

1. Run Docker's official installation script on your VPS:
2. Bash

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

3.
4.
5. Add your user to the `docker` group to run Docker commands without `sudo`.
6. Bash

```
sudo usermod -aG docker daniel
```

7.
8.
9. **CRITICAL:** Log out (`exit`) and log back in (`ssh daniel@...`) for this group change to take effect.

Step 3.2: Transfer Project Files & Set Up DNS

1. **On your Mac's terminal**, navigate to the folder *containing* `AI_LawFirmProject` and use `scp` to upload the entire project to your server's home directory.
2. Bash

```
scp -r ./AI_LawFirmProject daniel@YOUR_VPS_IP:~/
```

3.
4.
5. **In your domain registrar's dashboard** (e.g., Namecheap), create an **A Record** for your domain.
    - **Host:** `@`
    - **Value:** Your VPS IP Address

Step 3.3: Final Server-Side Configuration

1. **Log in to the GitHub Container Registry.** This is required to download some of the base images. Generate a **Personal Access Token** on GitHub with `read:packages` scope, then run this command on your VPS:
2. Bash

```
docker login ghcr.io -u YOUR_GITHUB_USERNAME
```

3.
4. When prompted for a password, paste your Personal Access Token.
5. **Fix Data Folder Permissions.** This is a crucial step to allow the containers to write data to the folders we will link from the server. Navigate into your project folder (`cd AI_LawFirmProject`) and run:
6. Bash

```
sudo chmod -R 777 ./n8n-data ./chroma_db ./storage
```

7.
8.
9. **Update Configuration Files.** Make sure your `Caddyfile` and `app.py` on the server are updated with your real domain name and n8n webhook URL.

Step 3.4: Launch the Application!

This is the final command. From inside your `AI_LawFirmProject` directory on the VPS, run:

Bash

```
docker compose up --build -d
```

The first time you run this, it will take several minutes to download base images, compile `llama.cpp`, and build your custom containers. Once it's done, your entire application is live.

Part 4: Post-Deployment Configuration & Use

Step 4.1: Configure n8n Workflows

1. **Access n8n:** To make your n8n UI accessible, add a new `A` record for a subdomain (e.g., `n8n`) at your registrar, then add a block for `n8n.your-domain.com` to your `Caddyfile` that reverse-proxies to `n8n-app:5678`. Restart Caddy with `docker compose restart caddy`. You can then access n8n securely at `https://n8n.your-domain.com`.
2. **Build the Ingestion Workflow:** Create a workflow with a **Manual Trigger** connected to an **Execute Command** node. The command should be `python3 /app/rag_scripts/rag_setup.py`. Run this once to index your documents.
3. **Build the Query Workflow:** Create a workflow with a **Webhook** node. Connect it to an **Execute Command** node. The command should be `python3 /app/rag_scripts/query.py "{{ $json.body.question }}"`. The quotes are essential.
4. **Save and Activate** both workflows.

Step 4.2: Use Your AI Assistant

1. Navigate to your main domain: **https://your-domain.com**.
2. The Streamlit UI will load. Ask a question about your documents.
3. The request will go through the entire architecture, and the answer from your private LLM will appear on the screen.

Congratulations! You have successfully built and deployed a complete, private, full-stack AI application.

Appendix: Final Configuration Files

Of course. Here is the complete, final source code for all 9 text files required to build and deploy your project.

This reflects all the corrections and improvements we made during our debugging sessions.

---

1. requirements.txt

Purpose: *Lists all the Python libraries required by the n8n-app and streamlit-app containers.*

```
requests

pypdf

chromadb

sentence-transformers

llama-index

llama-index-vector-stores-chroma

llama-index-embeddings-huggingface

llama-index-llms-openai
```

2. rag_scripts/rag_setup.py

Purpose: *This "smart" ingestion script runs inside the n8n container. It scans the docs folder, compares it against the database, and processes only the new, un-indexed files.*

Python

```
import os
import sys
from llama_index.core import (
    VectorStoreIndex,
    SimpleDirectoryReader,
    StorageContext,
    load_index_from_storage,
    Settings
)
from llama_index.vector_stores.chroma import ChromaVectorStore
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
import chromadb

# Define absolute paths for use inside the Docker container
DB_PATH = "/app/chroma_db"
DOCS_PATH = "/app/docs"
STORAGE_PATH = "/app/storage"

print("--- Smart Ingestion Script Started ---")


try:
    db = chromadb.PersistentClient(path=DB_PATH)
    chroma_collection = db.get_or_create_collection("my_collection")
    vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
    embed_model = HuggingFaceEmbedding(model_name="sentence-transformers/all-MiniLM-L6-v2")
    Settings.embed_model = embed_model
```

```
except Exception as e:
    print(f"Error initializing database or models: {e}")
    sys.exit(1)

print("Checking for already indexed documents...")
try:
    existing_items = chroma_collection.get(include=["metadatas"])
    indexed_files = set(meta['file_path'] for meta in existing_items['metadatas'])
    print(f"Found {len(indexed_files)} source files already in the index.")
except Exception:
    indexed_files = set()

all_files_on_disk = set()
if os.path.exists(DOCS_PATH):
    for filename in os.listdir(DOCS_PATH):
        all_files_on_disk.add(os.path.join(DOCS_PATH, filename))

new_files_to_process = all_files_on_disk - indexed_files

if not new_files_to_process:
    print("No new documents found to process. Exiting.")
    sys.exit(0)

print(f"\nFound {len(new_files_to_process)} new document(s) to ingest:")
for file in new_files_to_process:
    print(f"  - {os.path.basename(file)}")

try:
    storage_context = StorageContext.from_defaults(vector_store=vector_store, persist_dir=STORAGE_PATH)
    try:
        index = load_index_from_storage(storage_context)
        print("Loaded existing index from storage.")
    except:
        print("No existing index found. Creating a new one.")
        index = VectorStoreIndex.from_documents([], storage_context=storage_context)

    for filepath in new_files_to_process:
        print(f"\nProcessing '{os.path.basename(filepath)}'...")
        new_document = SimpleDirectoryReader(input_files=[filepath]).load_data()
        index.insert_nodes(new_document)
        print(f"Successfully inserted '{os.path.basename(filepath)}' into the index.")

    print("\nPersisting updated index...")
    index.storage_context.persist(persist_dir=STORAGE_PATH)
    print("--- Smart Ingestion Script Finished Successfully ---")

except Exception as e:
    print(f"\nAn error occurred during indexing: {e}")
    sys.exit(1)
```

3. rag_scripts/query_rag.py

Purpose: *This script receives a question from a command-line argument, queries the RAG pipeline, and prints the final answer to the console for n8n to capture.*

Python

```python
import os
import sys
import argparse
import logging
from llama_index.core import VectorStoreIndex, StorageContext, load_index_from_storage, Settings
from llama_index.vector_stores.chroma import ChromaVectorStore
from llama_index.embeddings.huggingface import HuggingFaceEmbedding
from llama_index.llms.openai import OpenAI
import chromadb

# This configuration silences the noisy logs from underlying libraries
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logging.getLogger("llama_index").setLevel(logging.WARNING)
logging.getLogger("sentence_transformers").setLevel(logging.WARNING)
logging.getLogger("chromadb.telemetry.product.posthog").setLevel(logging.WARNING)
logging.getLogger("httpx").setLevel(logging.WARNING)

# 1. SETUP COMMAND-LINE ARGUMENT PARSER
parser = argparse.ArgumentParser(description="Query the RAG pipeline with a specific question.")
parser.add_argument("question", type=str, help="The question you want to ask.")
args = parser.parse_args()
```

```
question = args.question

# 2. GET LLM SERVER ADDRESS FROM ENVIRONMENT VARIABLE
LLM_API_BASE = os.getenv("LLM_API_BASE")
if not LLM_API_BASE:
    print("Error: The LLM_API_BASE environment variable was not set.")
    sys.exit(1)

# 3. INITIALIZE MODELS AND SETTINGS
try:
    llm = OpenAI(model="local-model", api_base=LLM_API_BASE, api_key="dummy")
    embed_model = HuggingFaceEmbedding(model_name="sentence-transformers/all-MiniLM-L6-v2")
    Settings.llm = llm
    Settings.embed_model = embed_model
except Exception as e:
    print(f"Error initializing models: {e}")
    sys.exit(1)

# 4. LOAD VECTOR DATABASE AND INDEX
try:
    db_path = "/app/chroma_db"
    storage_path = "/app/storage"
    db = chromadb.PersistentClient(path=db_path)
    chroma_collection = db.get_or_create_collection("my_collection")
    vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
    storage_context = StorageContext.from_defaults(persist_dir=storage_path, vector_store=vector_store)
    index = load_index_from_storage(storage_context)
except Exception as e:
    print(f"Error loading vector database or index: {e}")
    sys.exit(1)

# 5. QUERY THE PIPELINE AND PRINT THE ANSWER
try:
    query_engine = index.as_query_engine()
    response = query_engine.query(question)
    print(str(response))
except Exception as e:
    print(f"Error during querying: {e}")
    sys.exit(1)
```

4. app.py

Purpose: *The Streamlit script that creates the user-friendly web interface.*

Python

```
import streamlit as st
import requests
import json

# IMPORTANT: This URL uses the n8n service name from docker-compose.
# The final path segment must match the production URL from your n8n Webhook node.
N8N_WEBHOOK_URL = "http://n8n-app:5678/webhook/22398436-911c-4798-a801-789a7411d5e8"
```

```
st.title("Private AI Assistant")
st.info("Ask a question about your documents.")

with st.form(key='query_form'):
    question = st.text_input("Your Question:")
    submit_button = st.form_submit_button(label='Get Answer')

if submit_button and question:
    with st.spinner("Searching documents and generating an answer..."):
        try:
            payload = {"question": question}
            response = requests.post(N8N_WEBHOOK_URL, json=payload)
            response.raise_for_status()
            result = response.json()

            if result and isinstance(result, list) and len(result) > 0 and 'stdout' in result[0]:
                answer = result[0]['stdout']
                st.success("Answer:")
                st.write(answer)
            else:
                st.error("The workflow returned an unexpected response format.")
                st.json(result)

        except Exception as e:
```

```
        st.error(f"An unexpected error occurred: {e}")
```

5. Dockerfile (for the n8n service)

Purpose: *The blueprint for building your custom n8n container, which includes Python and all your required libraries.*

Dockerfile

```
# Start from a Python 3.10 Debian-based image
FROM python:3.10-slim-bookworm

# Set user to root for system package and n8n installation
USER root

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends tini npm curl && curl -fsSL https://deb.nodesource.com/setup_20.x | bash - && apt-get install -y nodejs && rm -rf /var/lib/apt/lists/*

# Install n8n globally using npm
RUN npm install -g n8n

# Create a directory for our app and scripts
WORKDIR /app

# Copy requirements file and install Python packages
COPY requirements.txt .
```

```
RUN pip install -r requirements.txt


# Create a non-root user and directory for n8n's own data
RUN adduser --system --group --home /home/node node


# Expose n8n port
EXPOSE 5678


# Switch to the n8n user
USER node
WORKDIR /home/node


# Define the entrypoint to run n8n
ENTRYPOINT ["tini", "--", "n8n"]
```

6. llm-server.Dockerfile

Purpose: *Builds the Llama.cpp server from source inside a container, ensuring it is perfectly compatible with your server's CPU architecture. This was the final fix for the `manifest unknown` and `lib...so` errors.*

Dockerfile

```
# llm-server.Dockerfile - Final Single-Stage Build


FROM debian:bullseye


# 1. Install all build and runtime dependencies in one go
RUN apt-get update && apt-get install -y --no-install-recommends \
    git \
    cmake \
    build-essential \
    ca-certificates \
    libgomp1

# 2. Clone the llama.cpp repository
RUN git clone https://github.com/ggerganov/llama.cpp.git

WORKDIR /llama.cpp

# 3. Build the code. The BUILD_SHARED_LIBS=OFF flag is crucial.
RUN mkdir build && cd build && \
    cmake .. -DLLAMA_CURL=OFF -DBUILD_SHARED_LIBS=OFF && \
    cmake --build .

# 4. Set up the runtime environment
WORKDIR /
RUN mkdir /models
EXPOSE 8080

# 5. Define the program to run when the container starts
ENTRYPOINT [ "/llama.cpp/build/bin/llama-server" ]
```

7. streamlit.Dockerfile

Purpose: *Packages your* `app.py` *script into its own container.*

Dockerfile

```
# streamlit.Dockerfile
FROM python:3.10-slim-bookworm


WORKDIR /app


COPY requirements.txt .
RUN pip install -r requirements.txt && pip install streamlit


COPY app.py .


EXPOSE 8501


CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

8. `Caddyfile`

Purpose: *Configures the Caddy reverse proxy. It tells Caddy how to direct traffic from your public domain to the correct internal container.* ***Remember to replace `your-domain.com` with your actual domain.***

```
your-domain.com {
    reverse_proxy streamlit-app:8501
}
```

9. `docker-compose.yml`

Purpose: *The master orchestration file. It defines all your services, links them on a private network, and manages all the volumes and environment variables.* ***Remember to replace your-model-file.gguf with your actual model filename.***

YAML

```yaml
services:
  caddy:
    image: caddy:latest
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
      - caddy_data:/data
    networks:
      - private-ai-net

  streamlit-app:
    build:
      context: .
      dockerfile: streamlit.Dockerfile
    restart: unless-stopped
    networks:
      - private-ai-net

  n8n-app:
    build:
      context: .
      dockerfile: Dockerfile
    restart: unless-stopped
    environment:
      - LLM_API_BASE=http://llm-server:8080/v1
      - N8N_SECURE_COOKIE=false
    volumes:
      - ./n8n-data:/home/node/.n8n
      - ./chroma_db:/app/chroma_db
      - ./storage:/app/storage
      - ./docs:/app/docs
      - ./rag_scripts:/app/rag_scripts
    networks:
      - private-ai-net

  llm-server:
    build:
      context: .
      dockerfile: llm-server.Dockerfile
    restart: unless-stopped
    volumes:
      - ./models:/models
    command:
      - "-m"
      - "/models/your-model-file.gguf"
      - "--host"
      - "0.0.0.0"
      - "--port"
      - "8080"
    networks:
      - private-ai-net

networks:
  private-ai-net:

volumes:
  caddy_data:
```