CDW Documentation

RAG Application Documentation

RAG Application Documentation

Overview

This Retrieval-Augmented Generation (RAG) application integrates Azure OpenAI, Azure AI Search (formerly Azure Cognitive Search), and Azure Blob Storage to enable document ingestion, vector embedding, indexing, and querying of content using natural language.

What This App Is

This is a **Retrieval-Augmented Generation (RAG)** system using:

- **Azure OpenAI** for embeddings and completions
- **Azure Cognitive Search** for vector-based document retrieval
- **Azure Blob Storage** to store source documents
- **FastAPI** backend for HTTP-based querying

It allows you to ask questions about your documents and receive GPT-based answers grounded in your content — not just pretraining data.

Why It’s Useful

Contextual Accuracy

- Augments GPT with your **domain-specific** documents
- Greatly reduces hallucinations

Enterprise Search Capability

- Uses **semantic vector search** to match concepts, not just keywords
- Fast, scalable, and supports ranked retrieval

Private and Secure

- Documents remain in your **own Azure Blob Storage**
- Hosted in **Azure**, supports enterprise security and compliance

Pluggable and Extensible

- Easily swap GPT models (`gpt-4`, `gpt-35-turbo`, etc.)

- Add metadata, analytics, monitoring
- Extend with frontend UI or APIs

Use Cases

- Internal knowledge base Q&A
- Compliance audit support
- Customer support assistants
- Product/engineering documentation lookup
- Competitive research summarization

File Structure

- app.py - FastAPI backend for query handling.
- ingest.py - Extracts documents from Blob Storage, creates embeddings, and indexes them into AI Search.
- requirements.txt - Python dependencies.
- .env - Environment variable configuration (not included here for security).
- README.md - Basic readme file.
- request.json - Sample input format for queries.

Prerequisites

- Python 3.10 or higher
- Azure OpenAI deployment
- Azure AI Search index (with vector search enabled)
- Azure Blob Storage container with documents
- Required environment variables:
    1. `AZURE_OPENAI_ENDPOINT`
    2. `AZURE_OPENAI_API_KEY`
    3. `AZURE_OPENAI_EMBEDDING_MODEL`
    4. `AZURE_SEARCH_ENDPOINT`
    5. `AZURE_SEARCH_KEY`
    6. `AZURE_SEARCH_INDEX_NAME`
    7. `AZURE_STORAGE_ACCOUNT_URL`
    8. `AZURE_STORAGE_CONTAINER_NAME`
    9. `AZURE_STORAGE_KEY`

Setup Instructions

- Install dependencies:

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

- Set environment variables in a `.env` file or export them manually.
- Run the ingest process to index documents:

```
python ingest.py
```

- Launch the app locally:

```
uvicorn app:app --reload
```

How It Works

Document Ingestion (ingest.py)

1. Connects to Azure Blob Storage.
2. Reads files, extracts content.
3. Generates vector embeddings using the OpenAI embedding model.
4. Uploads documents + vectors to Azure AI Search index.

Query Handling (app.py)

1. Accepts a query via POST `/query`.
2. Embeds the query, runs vector search on the index.
3. Retrieves the most relevant documents.
4. Builds a prompt with context and sends to OpenAI chat completion model.
5. Returns the answer and source document names.

Sample Request

```
POST /query
{
  "question": "What are the limitations of AFFIRM?"
}
```

Sample Response

```
{
  "answer": "AFFIRM lacks automated compliance checks...",
  "sources": ["AFFIRM_Notes_-_Max.docx", "Affirm_Bullet_Points.docx"]
}
```

Additional Notes

- This app assumes `.docx` files are uploaded to the Blob container.

- Token and prompt limits apply when sending context to the OpenAI model.
- There are prerequisite Azure resources required for this project:
    1. Azure AI Search with a Search Index and Vector Search Configuration
    2. Azure OpenAI resources:
        1. Text Embedding model (text-embedding-ada-002 used here)
        2. Azure OpenAI gpt model (gpt-4o used here)

1. Azure Blob Storage with container for document storage
2. Azure Key Vault (Optional for dev but recommended)
3. Azure App Service (if not running locally)