PROJECT OVERVIEW – AZURE-POWERED RAG AGENT API (BUSINESS + DEVOPS)

Elevator pitch  
 This project is a small, production-style Knowledge API you can put in front of business users or internal products. It lets people ask questions against your own documents and get grounded, accurate answers (Retrieval-Augmented Generation). It’s built with FastAPI and LangChain, uses Azure AI Search for vector retrieval, and runs as a container on Azure Container Apps. It’s secure, observable, cost-controlled, and easy to extend.

Who this helps  
 • Operations and Support teams – instant answers from policies, SOPs, FAQs.  
 • Sales/CS – fast access to product sheets, proposals, pricing notes.  
 • Engineering – searchable runbooks and architecture docs.  
 • IT/Compliance – a controllable, auditable alternative to “copy/paste into public AI”.

Business outcomes  
 • Faster resolution: users self-serve from approved content.  
 • Trustworthy answers: responses cite your documents.  
 • Measured cost: each request and token usage is visible; the stack scales to zero when idle.  
 • Enterprise posture: authentication, secrets, logging, and a clean upgrade path to Key Vault, API Management, and CI/CD.

What the system does (plain language)

1. You upload your files. The system splits them into small chunks and stores semantic vectors in Azure AI Search.
2. A user asks a question via a simple, secure API (or Swagger UI).
3. The API finds the most relevant chunks and asks the LLM to answer only using that context.
4. The API returns a clear answer plus sources (file names), and logs the interaction for observability.

High-level architecture  
 • Client: Swagger UI (OpenAPI) for quick testing; any front-end or system can call the REST endpoints.  
 • API service: FastAPI, with OAuth2/JWT for a simple bearer-token flow; documented via OpenAPI.  
 • RAG logic: LangChain orchestrates retrieval from Azure AI Search and prompts the LLM.  
 • Vector store: Azure AI Search (vector + keyword) holds chunked text and embeddings.  
 • Embeddings/LLM: OpenAI API for embeddings and model calls (can switch to Azure OpenAI later).  
 • Containerization: Docker image published to Azure Container Registry (ACR).  
 • Runtime: Azure Container Apps with external ingress for a public URL; secrets stored as Container Apps secrets (Key Vault ready).  
 • Observability: Container logs in Azure Monitor/Log Analytics; health endpoint; scaling controls.  
 • Security: JWT on the API, secrets out of code, optional future fronting with API Management.

Key features  
 • Retrieval-Augmented Generation (RAG) with Azure AI Search (HNSW vector indexing).  
 • Secure REST API with /token and /query; OpenAPI docs built in.  
 • Document ingestion scripts for PDFs/Word/TXT (extensible); safe document IDs; batch upload.  
 • Delete and re-ingest without breaking the index schema.  
 • Cloud-native deployment on Azure Container Apps; ACR for image hosting.  
 • Cost controls: Basic SKUs, scale to zero, tiny footprint.

What’s live and testable  
 • Navigate to the Base URL (live):<https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs> to see Swagger.  
 • Request a token via /token, click Authorize, then call /query with a question about your docs.  
 • You’ll get an answer grounded in your uploaded files with source names included.

Folder and file map (what each piece does)  
 app/  
 • api.py – FastAPI routes: /health, /token, /query (auth + OpenAPI docs).  
 • auth.py – Lightweight OAuth2/JWT generation and verification for the API.  
 • rag.py – The RAG brain: builds the Azure AI Search retriever and asks the LLM with retrieved context.  
 • settings.py – Central configuration (reads .env for secrets and endpoints).  
 • ingest/  
 – index\_schema.json – Azure AI Search index definition (fields and vector profile).  
 – readers.py – Text loaders (TXT/Word/PDF). Easy to extend with more formats.  
 – chunkers.py – Splits long documents into chunks that embed well.  
 – load\_docs.py – Creates the index if missing, embeds chunks, uploads to Azure AI Search.  
 – delete\_docs.py – Safely deletes documents (by filename pattern or exact IDs) from the index.  
 data/ – Example documents to ingest (TXT, DOCX, PDF).  
 tests/ – Small scripts (e.g., to validate your OpenAI key).  
 Dockerfile – Production image build for the API service.  
 requirements.txt – Python dependencies.  
 .env – Local development secrets (never commit to public repos).

How data flows end-to-end

1. Ingest: You run load\_docs.py. It reads files from data/, extracts text, chunks it, creates embeddings, and uploads chunks + vectors to Azure AI Search.
2. Query: A client gets a JWT from /token (e.g. [https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.i](https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs)o/token), then calls /query?q=… . The API retrieves top-k chunks from the index, constructs a grounded prompt, calls the LLM, and returns a concise answer with sources.
3. Observe: Logs go to Azure Monitor; you can view request counts, latency, and errors.

Public API surface (short)

Base URL (live):<https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io>

Docs (Swagger):<https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs>

• GET /health – Liveness probe for your monitoring.  
 • POST /token – Returns a short-lived JWT (demo credentials).  
 • POST /query?q=… – Returns a JSON answer based on your documents; requires Authorization: Bearer <token>.  
 Swagger/OpenAPI – Built in at /docs for exploration and quick testing.

Security posture (right-sized for a demo, upgrade-ready)  
 • JWT protects the query endpoint; add real identity later (Azure AD, APIM, etc.).  
 • Secrets are consumed as environment variables; on Azure they are stored as Container Apps secrets (recommended next step: Key Vault + Managed Identity).  
 • No secrets in code or in the image; rotate keys by updating secrets on the app.  
 • Least-privilege: only your Search admin/query keys and OpenAI key are needed.

SRE/DevOps details that matter  
 • Container boundary: Single service image; deterministic builds via Dockerfile.  
 • Image registry: ACR holds the image; optionally enable admin temporarily for manual pulls (use ACR roles for production).  
 • Runtime platform: Azure Container Apps provides external ingress, autoscaling, revisions, and rollbacks.  
 • Scaling: Min replicas 0, max 1 by default; add rules for HTTP or CPU if needed.  
 • Observability:  
 – Logs: az containerapp logs show or Azure Portal → Container Apps → Log stream.  
 – Health: /health endpoint for probes.  
 – Metrics: requests, errors, replica state, CPU/memory via ACA and Log Analytics.  
 • Release flow options:  
 – Manual: docker build/push to ACR, then az containerapp update to roll out.  
 – CI/CD (recommended): GitHub Actions to build and push on main, then update ACA image.  
 • Rollback: Container Apps supports revisions; you can pin or swap to the previous revision if a deploy misbehaves.  
 • Runbooks:  
 – Scaling outage: scale replicas to 1, check logs, verify secrets, call /health.  
 – Search issues: verify index exists, keys valid, and service status is running.  
 – OpenAI issues: check key, rate limits/quota, and retry with backoff.

Cost controls and sizing  
 • Azure AI Search: Basic SKU with 1 replica and 1 partition is enough for small corpora and demos.  
 • Container Apps: minimal vCPU/memory and scale-to-zero keep runtime costs low.  
 • OpenAI: embedding and chat costs depend on tokens; you can cap spend on the OpenAI dashboard.  
 • Stop the resource group when not demoing to avoid residual costs.

How to add new documents

1. Place new files in the data/ folder (txt, docx, pdf supported now).
2. From the project root, activate your virtual environment and run: python -m app.ingest.load\_docs
3. Re-test /query in Swagger with a question about the new material.

How to remove or replace documents  
 • Remove a single file’s chunks: python -m app.ingest.delete\_docs --filename about.txt  
 • Remove a set by pattern: python -m app.ingest.delete\_docs --pattern \*.pdf  
 • Re-ingest after updates by running load\_docs.py again.

Deployment summary (what’s in Azure)  
 • Resource group: rg-rag  
 • Azure AI Search service: holds the index and vectors  
 • Azure Container Registry (ACR): stores your Docker image  
 • Azure Container Apps: hosts the API; gives you a public FQDN https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs  
 • Optional: Log Analytics workspace for richer logs/queries

Live demo script (2 minutes)

1. Open the public URL Base URL (live):<https://aca-rag.wittysand-46da5683.ukwest.azurecontainerapps.io/docs> (Swagger).
2. Click POST /token, enter any non-empty username/password, Execute, and copy the access\_token.
3. Click Authorize (top-right), paste Bearer <token>, then try POST /query with a question about your uploaded docs.
4. Point out the answer plus sources.
5. Show GET /health (green) and mention logs in Azure Portal.
6. Optional: upload a new file and re-run the query to show the updated knowledge.

Design choices recruiters care about  
 • RAG instead of raw LLM: makes answers grounded and auditable.  
 • Azure AI Search over pure vector DB: first-class Azure service, HNSW performance, easy integration, enterprise controls.  
 • Container Apps: fast, public URL, autoscaling, revisions—great for demos and small production use cases.  
 • FastAPI + OpenAPI: clear contract, simple auth, widely adopted.  
 • Secrets and IAM: environment-based secrets now, designed to move to Key Vault + Managed Identity.  
 • Extensibility: new file types, guardrails, APIM front-door, and CI/CD can be added without re-architecting.

Risks and mitigations  
 • Hallucination risk: reduced via retrieval and prompt constraints; can add citation enforcement and refusal rules.  
 • Data sensitivity: put private files in a private storage account and index from there; enforce network/IP restrictions as needed.  
 • Key leakage: store only in secrets providers; rotate regularly; avoid printing to logs.  
 • Costs drifting: keep Search on Basic; scale ACA to zero; monitor OpenAI usage/budgets.

Roadmap (quick wins)  
 • Key Vault + Managed Identity for secrets.  
 • Azure API Management front-door for enterprise auth, quotas, and versioning.  
 • CI/CD with GitHub Actions and OIDC-based Azure login.  
 • Guardrails: prompt safety checks, citation requirement, and unit test prompts (Promptfoo/Ragas).  
 • UI: a minimal web front-end that calls the API and shows sources inline.  
 • Analytics: store anonymized Q&A for insights and content gaps.

FAQ (non-technical stakeholders)  
 Is this safe to use with company documents?  
 Yes—your files live in your Azure subscription, the API is authenticated, and you can restrict the endpoint and storage further (private networking, APIM, RBAC, Key Vault).

What happens when we add more documents?  
 Re-run the ingestion script; the index is updated incrementally. No downtime is required.

Can this connect to SharePoint, Confluence, etc.?  
 Yes—add a loader to readers.py or introduce a lightweight ingestion microservice; the API doesn’t change.

Can we use Azure OpenAI only?  
 Yes—swap the embedding/model provider in settings and rag.py; the abstractions are already in place.

Interview talking points (to show senior ownership)  
 • I designed this as a small, composable RAG service with secure APIs and cloud-native deployment on Azure.  
 • I focused on enterprise signals: auth, secrets, observability, and cost control from day one.  
 • I chose Azure AI Search for managed vector search and Container Apps for a frictionless, scalable runtime.  
 • I separated ingestion from serving so content updates don’t require redeploying the API.  
 • I left clear upgrade paths: Key Vault, APIM, CI/CD, and agent workflows.