Azure RAG Architecture for SharePoint

Custom RAG Solution for Enterprise Environments

Date: February 2026 **Purpose:** Design document for building a custom RAG (Retrieval-Augmented Generation) system using Azure services to query SharePoint documents.

Table of Contents

Overview

This document outlines a custom RAG solution for organizations that:

- Use Azure as their enterprise cloud provider
- Store documents in SharePoint Online
- Cannot use external AI services (e.g., Claude/Anthropic)
- Want to avoid M365 Copilot licensing costs ($30/user/month)

What is RAG?

Retrieval-**A**ugmented **G**eneration combines:

1. **Retrieval**: Find relevant documents using semantic search
2. **Augmented**: Add retrieved content to the AI prompt
3. **Generation**: AI generates answers based on your documents

---

Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Azure Cloud                              │
│                                                                       │
│   ┌──────────────┐       ┌──────────────────┐       ┌────────────────┐   │
│  │ SharePoint  │───────▶│ Azure Function  │───────▶│ Azure AI Search│  │
│  │ (Documents) │        │ (Sync & Process)│        │ (Vector Store) │  │
│  └─────────────┘        └─────────────────┘        └────────────────┘  │
│         │                                                  │           │
│         │ Webhook                                          │ Search    │
│         ▼                                                  ▼           │
│  ┌─────────────┐        ┌─────────────────┐        ┌────────────────┐  │
│  │ Event Grid  │        │ Azure OpenAI    │◀───────│ Azure Function │  │
│  │ (Triggers)  │───────▶│ (LLM)           │        │ (RAG API)      │  │
│  └─────────────┘        └─────────────────┘        └────────────────┘  │
│                                                            │           │
│                                                            ▼           │
│                              ┌─────────────────────────────────┐       │
│                              │ Power App / Teams Bot / Web UI  │       │
│                              │ (User Interface)                │       │
│                              └─────────────────────────────────┘       │
│                                                                        │
└────────────────────────────────────────────────────────────────────────┘
```

Components

Component	Azure Service	Purpose
Documents	SharePoint Online	Source of truth for all documents
Change Detection	Event Grid + Webhooks	Detect when files are added/modified
Processing	Azure Functions	Parse documents, chunk text, generate embeddings
Vector Database	Azure AI Search	Store embeddings and perform semantic search
LLM	Azure OpenAI (GPT-4)	Generate natural language answers
User Interface	Power Apps / Teams / Web App	End-user interaction
Authentication	Azure AD / Entra ID	Enterprise SSO and access control

Document Ingestion Pipeline

When a document is uploaded or modified in SharePoint:

```
SharePoint File Added/Updated
        │
        ▼
Event Grid (detects change)
        │
        ▼
Azure Function: "ProcessDocument"
```

```
                 │
                 ├── 1. Download file from SharePoint (via Graph API)
                 │
                 ├── 2. Parse content (PDF, Word, Excel, PowerPoint)
                 │
                 ├── 3. Chunk text into segments (500-800 characters)
                 │
                 ├── 4. Generate embeddings (Azure OpenAI text-embedding-ada-002)
                 │
                 └── 5. Store chunks + embeddings in Azure AI Search
```

Supported File Formats

Format	Extension	Parser
PDF	.pdf	PyMuPDF / Azure Document Intelligence
Word	.docx	python-docx
Excel	.xlsx	openpyxl
PowerPoint	.pptx	python-pptx
Text	.txt, .md	Direct read

Query Pipeline

When a user asks a question:

```
User asks question
                 │
                 ▼
Azure Function: "QueryRAG"
                 │
                 ├── 1. Convert question to embedding vector
                 │
                 ├── 2. Search Azure AI Search (vector similarity)
                 │
                 ├── 3. Retrieve top 5 relevant document chunks
                 │
                 ├── 4. Build prompt with context + question
                 │
                 ├── 5. Send to Azure OpenAI (GPT-4)
                 │
                 └── 6. Return generated answer to user
```

Azure Services Setup

1. Azure AI Search Index Schema

Create an index to store document chunks and their embeddings:

```
{
  "name": "sharepoint-docs",
  "fields": [
    {
      "name": "id",
      "type": "Edm.String",
      "key": true
    },
    {
      "name": "content",
      "type": "Edm.String",
      "searchable": true
    },
    {
      "name": "content_vector",
      "type": "Collection(Edm.Single)",
      "dimensions": 1536,
      "vectorSearchProfile": "default"
    },
    {
      "name": "source_file",
      "type": "Edm.String",
      "filterable": true,
      "facetable": true
    },
    {
      "name": "site_name",
      "type": "Edm.String",
      "filterable": true
    },
    {
      "name": "file_path",
      "type": "Edm.String"
    },
    {
      "name": "modified_date",
      "type": "Edm.DateTimeOffset",
      "filterable": true,
      "sortable": true
    },
    {
      "name": "chunk_index",
      "type": "Edm.Int32"
    }
  ],
  "vectorSearch": {
    "profiles": [
      {
        "name": "default",
        "algorithm": "hnsw-algorithm"
      }
    ],
    "algorithms": [
      {
        "name": "hnsw-algorithm",
        "kind": "hnsw",
        "hnswParameters": {
          "metric": "cosine",
          "m": 4,
          "efConstruction": 400,
          "efSearch": 500
        }
      }
    ]
  }
}
```

2. Azure OpenAI Deployment

Required deployments in Azure OpenAI:

Model	Deployment Name	Purpose
text-embedding-ada-002	embeddings	Convert text to vectors
gpt-4	gpt4	Generate answers

3. Azure Function App

- **Runtime**: Python 3.11
- **Plan**: Consumption (pay-per-use) or Premium (for VNet integration)
- **Functions**:
  - `ProcessDocument` - Triggered by Event Grid
  - `QueryRAG` - HTTP trigger for user queries
  - `SyncSharePoint` - Timer trigger for periodic full sync

Code Examples

Document Processing Function

```python
import azure.functions as func
from azure.search.documents import SearchClient
from azure.identity import DefaultAzureCredential
from openai import AzureOpenAI
from msgraph import GraphServiceClient
import os

# Initialize clients
credential = DefaultAzureCredential()
```

```
openai_client = AzureOpenAI(
    azure_endpoint=os.getenv("AZURE_OPENAI_ENDPOINT"),
    api_key=os.getenv("AZURE_OPENAI_KEY"),
    api_version="2024-02-15-preview"
)

search_client = SearchClient(
    endpoint=os.getenv("AZURE_SEARCH_ENDPOINT"),
    index_name="sharepoint-docs",
    credential=credential
)

def process_document(file_id: str, site_id: str, filename: str):
    """Process a SharePoint document and add to search index."""

    # 1. Download from SharePoint
    graph_client = GraphServiceClient(credential)
    content =
graph_client.sites[site_id].drive.items[file_id].content.get()

    # 2. Parse document based on file type
    text = parse_document(content, filename)

    # 3. Chunk the text
    chunks = chunk_text(text, chunk_size=800, overlap=100)

    # 4. Generate embeddings and prepare documents
    documents = []
    for i, chunk in enumerate(chunks):
        # Get embedding from Azure OpenAI
        embedding_response = openai_client.embeddings.create(
            input=chunk,
            model="text-embedding-ada-002"
        )
        embedding = embedding_response.data[0].embedding

        documents.append({
            "id": f"{file_id}_{i}",
            "content": chunk,
            "content_vector": embedding,
            "source_file": filename,
            "site_name": site_id,
            "chunk_index": i
        })

    # 5. Upload to Azure AI Search
    search_client.upload_documents(documents)

    return len(documents)


def chunk_text(text: str, chunk_size: int = 800, overlap: int = 100) ->
list:
    """Split text into overlapping chunks."""
    chunks = []
    start = 0

    while start < len(text):
        end = start + chunk_size
        chunk = text[start:end]

        # Try to break at sentence boundary
        if end < len(text):
            last_period = chunk.rfind('.')
            if last_period > chunk_size * 0.5:
                chunk = chunk[:last_period + 1]
                end = start + last_period + 1

        chunks.append(chunk.strip())
        start = end - overlap

    return chunks


def parse_document(content: bytes, filename: str) -> str:
    """Parse document content based on file type."""
    ext = filename.lower().split('.')[-1]

    if ext == 'pdf':
        import fitz
        doc = fitz.open(stream=content, filetype="pdf")
        return "\n".join([page.get_text() for page in doc])

    elif ext == 'docx':
        from docx import Document
        import io
        doc = Document(io.BytesIO(content))
        return "\n".join([p.text for p in doc.paragraphs])

    elif ext == 'xlsx':
        from openpyxl import load_workbook
        import io
        wb = load_workbook(io.BytesIO(content))
        text_parts = []
        for sheet in wb:
            for row in sheet.iter_rows(values_only=True):
                row_text = " | ".join([str(c) for c in row if c])
                if row_text:
                    text_parts.append(row_text)
        return "\n".join(text_parts)

    elif ext in ['txt', 'md']:
        return content.decode('utf-8', errors='ignore')

    return ""
```

RAG Query Function

```
import azure.functions as func
import json

def query_rag(question: str, site_filter: str = None) -> dict:
    """
    Process a user question using RAG.

    Args:
        question: User's natural language question
        site_filter: Optional SharePoint site to filter results

    Returns:
        Dictionary with answer and sources
    """

    # 1. Generate embedding for the question
    question_embedding = openai_client.embeddings.create(
        input=question,
        model="text-embedding-ada-002"
    ).data[0].embedding

    # 2. Search Azure AI Search with vector query
    search_results = search_client.search(
        search_text=question,
        vector_queries=[{
            "vector": question_embedding,
            "k_nearest_neighbors": 5,
            "fields": "content_vector"
        }],
        filter=f"site_name eq '{site_filter}'" if site_filter else None,
        select=["content", "source_file", "file_path"]
    )

    # 3. Build context from search results
    results_list = list(search_results)

    if not results_list:
        return {
            "answer": "I couldn't find any relevant information in the
documents.",
            "sources": []
        }

    context_parts = []
    sources = []

    for result in results_list:
        context_parts.append(
            f"From '{result['source_file']}':\n{result['content']}"
        )
        if result['source_file'] not in sources:
            sources.append(result['source_file'])

    context = "\n\n---\n\n".join(context_parts)

    # 4. Generate answer using Azure OpenAI
    system_prompt = """You are a helpful assistant that answers questions
based on
the provided document context.

Rules:
- Only answer based on the provided context
- If the context doesn't contain the answer, say so
- Be concise and direct
- Cite which document the information came from"""

    response = openai_client.chat.completions.create(
        model="gpt-4",
        temperature=0,
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": f"Context:\n{context}\n\nQuestion:
{question}"}
        ]
    )

    return {
        "answer": response.choices[0].message.content,
        "sources": sources
    }


# HTTP Trigger for Azure Function
def main(req: func.HttpRequest) -> func.HttpResponse:
    try:
        body = req.get_json()
        question = body.get('question')
        site_filter = body.get('site')

        if not question:
            return func.HttpResponse(
                json.dumps({"error": "Question is required"}),
                status_code=400
            )

        result = query_rag(question, site_filter)

        return func.HttpResponse(
            json.dumps(result),
            mimetype="application/json"
        )

    except Exception as e:
        return func.HttpResponse(
            json.dumps({"error": str(e)}),
            status_code=500
        )
```

Cost Estimates

Monthly Cost Breakdown

Service	Tier	Estimated Cost
Azure AI Search	Basic (15GB, 3 replicas)	$75/month
Azure OpenAI - Embeddings	text-embedding-ada-002	$0.0001/1K tokens
Azure OpenAI - GPT-4	gpt-4	$0.03/1K input, $0.06/1K output
Azure Functions	Consumption plan	$0-20/month
Azure Storage	Standard	$5/month
Event Grid	Per operation	~$1/month

Cost Scenarios

Usage Level	Queries/Month	Documents	Est. Monthly Cost
Light	1,000	500	~$100
Moderate	10,000	2,000	~$150
Heavy	50,000	10,000	~$300

Comparison with M365 Copilot

Aspect	M365 Copilot	Custom Azure RAG
Cost (100 users)	$3,000/month	~$150/month
Cost (500 users)	$15,000/month	~$200/month
Setup Time	Immediate	2-4 weeks
Maintenance	Microsoft managed	Self-managed
Customization	Limited	Full control
Prompt Engineering	Not possible	Fully customizable
Data Location	Microsoft cloud	Your Azure tenant
Supported Sources	All M365 apps	SharePoint (extensible)

Annual Savings (100 users)

```
M365 Copilot:     $3,000 × 12 = $36,000/year
Custom Azure RAG: $150 × 12   = $1,800/year
                               ─────────────
Savings:                        $34,200/year
```

UI Options

Option 1: Power Apps (Recommended for Quick Start)

- **Effort**: Low (1-2 days)
- **Skills**: No coding required
- **Best for**: Internal business users
- **Features**: Forms, basic chat interface, SharePoint integration

Option 2: Teams Bot

- **Effort**: Medium (1 week)
- **Skills**: Bot Framework, Node.js/C#
- **Best for**: Teams-centric organizations
- **Features**: Conversational UI, embedded in Teams

Option 3: SharePoint Web Part (SPFx)

- **Effort**: Medium (1-2 weeks)
- **Skills**: React, TypeScript, SPFx
- **Best for**: Embedded experience in SharePoint
- **Features**: Native SharePoint look and feel

Option 4: Custom Web Application

- **Effort**: High (2-4 weeks)
- **Skills**: Full-stack development
- **Best for**: Public-facing or highly custom needs
- **Features**: Complete flexibility

Implementation Roadmap

Phase 1: Foundation (Week 1)

- [ ] Set up Azure resource group
- [ ] Deploy Azure AI Search
- [ ] Deploy Azure OpenAI
- [ ] Configure Azure AD app registration
- [ ] Set up SharePoint API permissions

Phase 2: Ingestion Pipeline (Week 2)

- [ ] Create Azure Function App
- [ ] Implement document processing function
- [ ] Set up Event Grid subscription for SharePoint
- [ ] Test with sample documents
- [ ] Initial document sync

Phase 3: Query API (Week 3)

- [ ] Implement RAG query function
- [ ] Add authentication/authorization
- [ ] Implement rate limiting
- [ ] Test query accuracy
- [ ] Tune search parameters

Phase 4: User Interface (Week 4)

- [ ] Build Power App / Teams Bot / Web UI
- [ ] Connect to RAG API
- [ ] Add source citations
- [ ] User acceptance testing
- [ ] Documentation and training

Phase 5: Production (Ongoing)

- [ ] Monitor performance and costs
- [ ] Tune prompts based on feedback
- [ ] Add additional SharePoint sites
- [ ] Implement feedback loop for improvement

Security Considerations

1. **Authentication**: Use Azure AD for all service-to-service auth
2. **Authorization**: Respect SharePoint permissions in search results
3. **Data Encryption**: Enable encryption at rest and in transit
4. **Network**: Consider Private Endpoints for sensitive data
5. **Logging**: Enable Azure Monitor for audit trails
6. **API Security**: Use API Management for rate limiting and key management

---

Support and Resources

- Azure AI Search Documentation
- Azure OpenAI Documentation
- Microsoft Graph API for SharePoint
- Azure Functions Documentation

Document generated for enterprise RAG implementation planning.