**Runwei Technical Documentation**

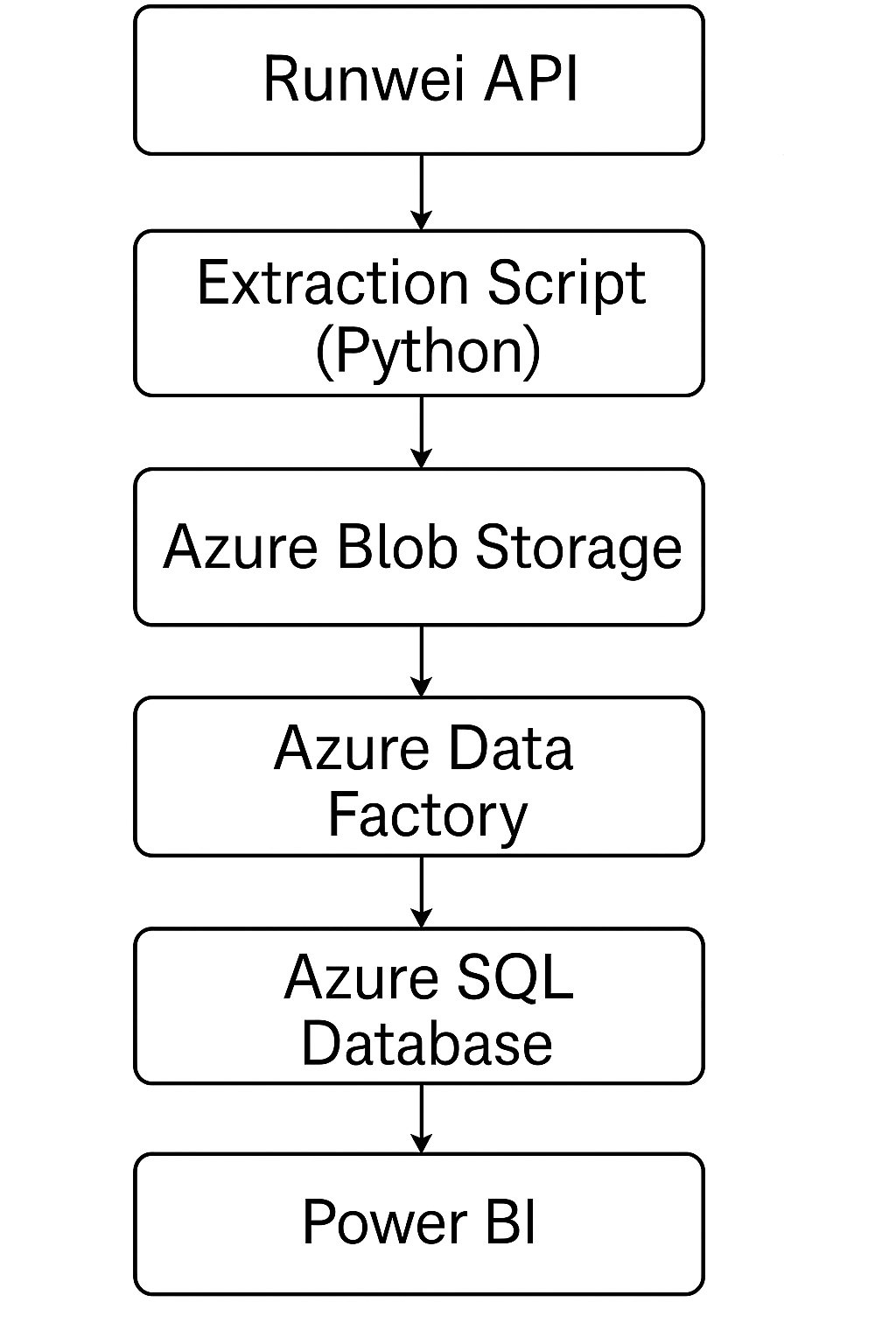
**Project 1 : Opportunities ETL Pipeline with Azure Blob Storage & Power BI Integration**

To establish a robust and scalable data pipeline that extracts raw opportunity data, transforms it into clean, queryable formats, and loads it into an Azure SQL Database. The pipeline supports dynamic business reporting through Power BI.

**1. Azure Services in Use**

|  |  |  |
| --- | --- | --- |
| **Service** | **Role** | **Notes** |
| Azure Data Factory | ETL orchestration | Pipelines for extraction, transformation, and load; scheduled daily. |
| Azure Blob Storage | Raw data storage | JSON files from API saved with timestamps; intermediate storage optional for RAG inputs. |
| Azure SQL Database | Data warehouse | Stores transformed opportunity records; primary storage for analytics and AI systems. |
| Power BI | Visualization | Connects to Azure SQL; dashboards updated daily. |
| FAISS (On VM or local server) | Vector database | Local FAISS flat index built from SQL exports. |
| DeepSeek API | Language generation | Used for RAG response generation; API key secured via Azure Key Vault (future plan). |

**2. ETL Pipeline Details**

****

**3.1 Extraction**

* Source: [Runwei API](https://api.beta.runwei.io/docs#/Opportunity/get_opportunities_using_api_key_api_v1_opportunity_via_api_key_get).
* Extraction script: Python (extract\_and\_upload.py).
* Storage: Files named runwei\_opportunities\_YYYYMMDD.json saved to Azure Blob Storage.
* Trigger: Task Scheduler on VM; plan to move to Azure Functions.

**3.2 Transformation**

* Tool: Azure Data Factory (Mapping Data Flow).
* Tasks:
  + Blank strings ("", N/A, n/a) converted to NULL.
  + Date formats standardized (YYYY-MM-DD).
  + Schema enforced for 40+ fields.

**3.3 Loading**

* Destination: Azure SQL Database (table: [dbo].[Opportunities]).
* Constraints:
  + Primary Key: ID
  + Insert/Update strategy considered for future (currently truncate-load).

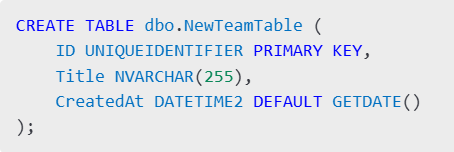
**3.4 Editing and Creating SQL Tables**

To edit and create tables in Azure SQL Database using the web interface:

* **Login to Azure Portal** and navigate to the RunweiOpportunities SQL database.
* In the left menu, click **Query editor**.
* Authenticate using either:
  + SQL Server Authentication (sql-admin + password),

\* Login: sql-admin ; Password: Runwei2025 (Need add addressIP first time)

* After logging in, you'll see the database structure (tables, views, etc.).
* To create a new table:
  + Click New Query.
  + Enter SQL like below:



1. Click **Run** to execute.
2. Refresh the Tables list to confirm creation.

⚠️ Note: Use dbo. prefix to follow convention.

**4. Visualization Layer (Power BI)**

* Connected to Azure SQL Database.(\* Contact me to acquire permission)
* Main databases:
  + [Opportunities]
  + [Opportunities\_cleaned]
* Data refreshed daily automatically.

**5. RAG (Retrieval-Augmented Generation) Integration**

**5.1 Data Preparation**

* Export fields: Title, ShortDescription, Eligibility, Tags, Industry, Description from Azure SQL.
* Text concatenated and embedded using sentence-transformer models (all-MiniLM-L6-v2).

**5.2 Vector Store**

* FAISS Index:
  + Flat index (IndexFlatIP) for maximum retrieval speed.
  + Stored locally; plan to deploy in VM blob storage if scaling is needed.

**5.3 Query + LLM**

* Top-K (e.g., K=5) nearest neighbors retrieved based on user query embedding.
* Context documents passed to DeepSeek API with a structured prompt.
* DeepSeek returns a summarized or direct answer.

**6. Automation and Monitoring**

* ADF Pipeline Monitoring:
  + Email alerts for failures (to be integrated with Azure Monitor).
* Blob Storage Cleanup:
  + Older JSON files archived or deleted periodically.
* Manual FAISS Refresh:
  + Currently triggered after daily pipeline; automate with cron job or Azure Function.

**7. Best Practices and Future Recommendations**

* Move Task Scheduler to Azure Function App for scalable extraction.
* Secure API keys and sensitive parameters using Azure Key Vault.
* Implement delta loading instead of full loads if API supports change tracking.
* Move FAISS deployment onto Azure VM with persistent volume support.
* Explore Azure OpenAI Service as alternative or supplement to DeepSeek for improved cost/performance.

**8. Repository and Access**

* Extraction Script: /runwei-etl/scripts/extract\_and\_upload.py
* ADF Pipelines: Azure Data Factory instance Runwei-ADF
* SQL Server: runwei-sql.database.windows.net - Database: OpportunitiesDB
* FAISS RAG Service: (internal use only, repo pending migration)

\*Access requires appropriate Azure AD permissions.

**Project 2: Runwei Retrieval-Augmented Generation (RAG) Pipeline with FAISS & DeepSeek**

**Overview:**

Implemented a Retrieval-Augmented Generation (RAG) system that enhances user interaction by providing context-aware answers to user queries. Leveraged FAISS for efficient semantic search and DeepSeek API for generating natural-language responses.

**GitHub Repository:**

* [Runwei-Retrieval-Augmented-Generation-RAG-Pipeline-with-FAISS-DeepSeek](https://github.com/Cloudpeng121/Runwei-Retrieval-Augmented-Generation-RAG-Pipeline-with-FAISS-DeepSeek)

**Technology Stack:**

* **FAISS:** Fast semantic similarity search.
* **Sentence Transformers (BAAI/bge-base-en):** Embedding generation.
* **DeepSeek API:** Large Language Model (LLM) for conversational AI.
* **Python:** Backend scripting and integration.
* **Azure SQL Database:** Data source for opportunities metadata.

**System Workflow:**

1. Data ingestion from Azure SQL Database.
2. Embeddings generation using Sentence Transformers.
3. FAISS vector index creation for rapid similarity retrieval.
4. Integration of DeepSeek API to formulate contextually accurate responses.

**Achievements:**

* Improved response accuracy and reduced latency for natural language queries.
* Enabled scalable handling of thousands of daily user queries, significantly enhancing user experience.

**Project 3: Superset Dataset Documentation and Optimization**

**Business Need**

Internal analysts and stakeholders rely on Apache Superset for reporting. However, dataset structures lacked consistency, documentation, and performance tuning.

**Objectives**

* Document Superset datasets and underlying SQL tables
* Add business definitions to fields and metrics
* Propose optimizations for dashboard speed and usability

**Deliverables**

* **Dataset Inventory Spreadsheet**:
  + Columns: Dataset Name, Table Name, Primary Keys, Foreign Keys, Description, Owner
* **Field Documentation Examples**:
  + AwardValue: *“Total monetary value offered for this opportunity in USD”*
  + IsOpen: *“Boolean field showing if the opportunity is still accepting applications”*
* **BI Readiness Enhancements**:
  + Recommended materialized views for complex joins
  + Indexed Deadline, Industry, and AwardValue columns
  + Proposed calculated fields like:
    - DaysUntilDeadline = DATEDIFF(DAY, GETDATE(), Deadline)
    - AwardCategory = CASE WHEN AwardValue > 100000 THEN 'High Value'...
* **Collaboration**:
  + Worked with product and analytics team to ensure datasets align with KPIs and user expectations