Retrieval Augmented Generation Industry Accelerator

Introduction

Retrieval Augmented Generation (RAG) is an AI framework for retrieving facts from an external knowledge base to ground large language models (LLMs) on the most accurate, up-to-date information and to give users insight into LLMs' generative process. A RAG pattern improves the quality of LLM-generated responses by grounding the model on external sources of knowledge to supplement the LLM's internal information. Implementing RAG in an LLM-based question answering system has two main benefits: it ensures that the model has access to the most current, reliable facts, and it ensures that users have visibility to the model's sources, so that its claims can be checked for accuracy and ultimately trusted.

This accelerator includes these main tasks:

- Connect to a vector database (either Elasticsearch or Milvus) and ingest some data, which can be PDF files, HTML pages, Word documents, markdown documents, text documents or PPTX documents.
- Split and index the documents for search and retrieval into the vector database.
- Deploy an AI Service Python function that performs the RAG steps for an input prompt and generates a response.
- Set up a Question & Answer interaction with the deployed function to demonstrate the process end-to-end.

Additionally, the project includes artifacts for these optional tasks:

- Test queries against vector databases by using different retrieval techniques.
- Create of an index for storing user feedback information based on the answers generated by the LLMs.
- Retrieve log data for analysis and run topic modelling to gain insights based on user feedback.
- Ingest expert profile data based on the sample JSON file provided in the project or your own expert data.
- Search and retrieve expert profiles with the RAG AI Service Python function for each question.
- Demonstrate how a Question & Answer UI-based interaction can be set up with the watsonx Assistant service.
- Index large sets of PDF documents from TAR archives only, stored in Cloud Object Storage (IBM Cloud Object Storage or AWS S3), for search and retrieval into Watsonx.data Milvus and Watsonx Discovery vector databases.
- Demonstrate how to setup Indexing of large datasets with Watsonx Discovery S3 Connector.
- Demonstrate how to setup & deploy Streamlit UI based chat bot App for Question & Answer with RAG on IBM Code Engine.

Tip: Download the PDF of these instructions from the Data assets section on the **Assets** page so you can keep these instructions open while you work.

Prerequisites

The accelerator requires the following prerequisites:

- A vector database: either Milvus or Elasticsearch.
    - If you use Elasticsearch, the machine learning capability must be enabled, and the ELSER2 sparse encoder model or E5 Multilingual dense encoder model must be deployed.
    - If you use Milvus with watsonx.data, follow these instructions: Adding Milvus service.
- An IBM watsonx.ai account on IBM Cloud where you can run prompts on LLMs. Prompt templates are provided with the accelerator for several variations of the Granite and Llama models (see the Prompt Templates section for more details).
- Optional: Additional Cloud Object Service (COS) connection for bulk ingestion.
    1. *COS Input Data Connection*: IBM/AWS Cloud Object Storage connection to the bucket containing your input data for ingestion. Ensure that your input data is already uploaded to the relevant account. This connection will be used for bulk ingestion in both Elasticsearch and Milvus scenarios.
    2. *COS Milvus Connection*: IBM/AWS Cloud Object Storage connection to the Milvus-associated bucket for storing intermediary output. Ensure that the necessary permissions are granted to allow watsonx.ai to access the specified IBM COS/AWS S3 buckets. This connection will be used only for bulk ingestion into Milvus.
- Optional: a watsonx Orchestrate service provisioned to set up the chat user interface for Question and Answer with RAG.

Setting up the accelerator

Before you can run the accelerator, you must create the sample project, create a connection to your vector database, create a deployment space, and set parameter values.

To set up the accelerator:

1. Log in to watsonx.ai and go to the Q&A with RAG Accelerator sample project in the Resource hub and click **Create project**.
2. In your project, create an Elasticsearch or Milvus connection asset:
    1. Go to the **Assets** tab and click **New asset**.
    2. Select **Connect to a data source** and then select Elasticsearch or Milvus and click **Next**.
    3. Enter `elasticsearch_connect` or `milvus_connect` as the name of the connection.
    4. Provide the connection URL, either username and password or apikey, and SSL certificate. For information on adding connections to a project, see Adding Elasticsearch Connection or Adding Milvus Connection.
    5. Test the connection, then click **Create** to save to it in the project.
3. Optional: For Bulk Indexing with Elasticsearch or Milvus, create COS connection assets in your project:
    1. Go to the **Assets** tab and click **New asset**.
    2. Select **Connect to a data source** and then select `IBM Cloud Object Storage` or `Amazon S3` and click **Next**.
    3. Provide the name of the connection for **COS Input Data Connection** as COS connection to the input data bucket for both Elasticsearch and Milvus indexing.

    4. When bulk indexing into Milvus, also provide the name of the connection for **COS Milvus Connection** as COS connection to the Milvus-associated bucket.
    5. Two Cloud Object Storage (COS) Connections are supported. Below are the instructions to connect:
        - For IBM Cloud Object Storage, Provide the Bucket name, Endpoint URL, [Resource instance ID, API Key, Access Key and Secret Key] as Authentication method. For information on adding connections to a project, see Adding IBM Cloud Object Storage Connection.
        - For AWS S3, Provide the Bucket name, Endpoint URL, region, [Access Key and Secret Key] as Basic Credentials. For information on adding connections to a project, see Adding Amazon S3 Connection.
    6. Test the connection, then click **Create** to save to it in the project.
4. Make sure that the watsonx.ai Runtime service is associated with the project:
    1. Go to the **Manage** tab and select **Services and Integrations**.
    2. If the watsonx.ai Runtime services is not listed, click **Associate Service**, select your watsonx.ai Runtime service, and then click **Associate**.
5. Create a deployment space:
    1. From the main menu, select **Deployments** and then click **New Deployment Space**.
    2. Enter a name for the deployment space.
    3. Select a storage service and a watsonx.ai Runtime service.
    4. Click **Create**, and then **View New Space**.
    5. Go to the **Manage** tab and copy the **Space GUID** value. You need this value to set the `watsonx_ai_space_id` parameter in the next step.
6. Set parameter values:
    1. Return to your project. From the main menu, select **Projects > View all projects** and then select the project name to open the project.
    2. Go to the **Assets** tab in the project, select the **Configurations** asset type, and then select the **RAG_parameter_set** parameter set.
    3. Update the following parameter values:
        - `connection_asset`: Enter the name of the connection asset that you just created.
        - `log_connection_asset`: Optional. If you want to set up feedback logging, enter the same connection asset name.
        - `watsonx_ai_space_id`: Enter the deployment space GUID value that you just copied.
        - `watsonx_ai_api_key` - You can enter an existing key or create an API key in the Keys section of the IBM Cloud console.
        - `deployment_serving_name`: Enter a unique serving name required for deploying a scoring pipeline function.
        - `expert_profiles_index`: Optional. If you want to set up expert profile, enter a valid index name.
        - `cos_data_input_connection_asset`: Optional. If you want to set up bulk indexing for either Elasticsearch or Milvus, enter the name of the `COS Input Data Connection` asset that you just created.
        - `cos_milvus_connection_asset`: Optional. If you want to set up bulk indexing for Milvus, enter the name of the `COS Milvus Connection` asset that you just created.
7. Set other parameters in the **RAG Parameter Set** and **RAG Advanced Parameter Set** parameter sets or retain the default values.

Parameter Sets

Instead of entering variable factors as part of the code design, parameters are used to represent processing variables. This project contains the following parameter sets:

RAG Parameter Set

1. `connection_asset`: The name of the connection asset in the project that connects to the vector database.
2. `watsonx_ai_api_key`: An API key for accessing IBM watsonx.ai.
3. `ingestion_data_file`: The input file which needs to be extracted, processed into documents and indexed into a vector database. This file type could be PDF/DOCX/PPTX/HTML or a ZIP archive with multiple documents.
4. `watsonx_ai_space_id`: The ID of the deployment space where you deploy the AI service Python function.
5. `vector_store_index_name`: The name of the Elasticsearch index or Milvus collection. Use only characters with letters, numbers, and underscores. The accelerator creates this index or collection if it does not exist.
6. `log_connection_asset`: Optional. The name of the Elasticsearch or Milvus connection asset for the feedback logging feature. Set if you want to set up feedback logging.
7. `log_index_name`: Optional. The name of the Elasticsearch index or Milvus collection feedback log index. Use only characters with letters, numbers, and underscores. The accelerator creates this index if it does not exist.
8. `expert_profiles_index`: Optional. The name of the expert profiles index or collection in the vector database. The accelerator creates this index or collection if it does not exist. Use only characters with letters and underscores.
9. `expert_profiles_document`: Optional. The document that contains expert profile data for ingestion into the vector database. You can use the default file that is provided in the data asset section or use your own document.
10. `deployment_serving_name`: A unique serving name required for deploying a scoring pipeline function. On IBM Cloud, this serving name must be unique within each region.
11. `cos_data_input_connection_asset`: (Bulk Ingestion only) The Cloud Object Storage (COS) connection asset that points to the bucket containing your input dataset for bulk indexing. This bucket is used to retrieve data, which will be extracted, processed into documents, and indexed into the vector database. The connection asset should reference either IBM Cloud Object Storage or Amazon S3. (Optional)
12. `cos_milvus_connection_asset`: (Bulk Ingestion only) The COS connection asset associated with the Milvus COS bucket for bulk ingestion into the Milvus vector database. This bucket is used to store processed data files for bulk ingestion into Milvus. The connection asset should reference either IBM Cloud Object Storage or Amazon S3. (Optional)

RAG Advanced Parameter Set

1. `vectorsearch_top_n_results`: The number of top results to retrieve from the vector query.
2. `elastic_search_model_id`: An identifier for a specific model or configuration that is used within Elasticsearch, for search optimization or data processing.
3. `elastic_search_vector_type`: Type of vector index created for the specified elastic_search_model_id (sparse or dense)

4. `es_number_of_shards`: The number of primary shards into which an Elasticsearch index is divided, determining how data is partitioned across the cluster.
5. `rag_es_min_score`: The minimum score for results from the vector query. Results below this threshold are not passed to the LLM.
6. `elastic_search_template_file`: A template file to perform Elasticsearch search. Here are the list of templates that are provided in the project are `elastic_search_ELSER_template.json`, `elastic_search_ELSER_BM25_hybrid_template.json`, `elastic_search_hybrid_multisearch_template.json` , `elastic_search_hybrid_rrf_template.json`, `elastic_search_ELSER_BM25_hybrid_template_rrf.json`, `elastic_search_multilingual_template.json` and `sample_elastic_search_NESTED_template.json`.
7. `ingestion_chunk_size`: The size of data chunks in tokens to ingest data into the vector index.
8. `ingestion_chunk_overlap`: The amount of overlap between consecutive data chunks.
9. `index_chunk_size`: The size of data chunks ingested into the vector index. This value influences indexing efficiency by balancing speed and resource consumption based on factors like data volume and system capabilities.
10. `milvus_hybrid_search`: Flag that determines whether to use hybrid search (combining dense and sparse embeddings) or only dense search in Milvus.
11. `embedding_model_id`: The name of the embedding model to store the documents in the vector database.
12. `llm_prompt_template_file`: The name of a prompt file in the project that contains an LLM prompt template.
13. `default_hallucination_technique`: The hallucination technique to use in the project. The supported values are `word_overlap` and `sentence_transformer`, `none`.
14. `hallucination_threshold_max_text_overlap`: The word overlap threshold for hallucination detection that uses the word overlap technique. Responses with scores below either threshold are considered to be hallucinations.
15. `hallucination_threshold_concatenated_text_overlap`: The concatenated text overlap threshold for hallucination detection that uses the word overlap technique. Responses with scores below either threshold are considered to be hallucinations.
16. `similarity_threshold_for_sentence_transformer_hallucination`: The threshold for hallucination detection that uses the sentence transformer technique. Responses with scores below this threshold are considered to be hallucinations.
17. `log_pii_removal`: A Boolean value that indicates whether personally identifiable information (PII) is removed in log records. The supported values are `true` and `false`.
18. `topic_modeling_method`: The method that is used for topic modeling in the **Analyze Log and Feedback** notebook. Supported values: `watson_nlp`, `top2vec`, `bertopic`. They are different NLP (Natural Language Processing) tools/libraries used for topic modeling and text analysis. Default is `watson_nlp` if the Watson NLP library is present in the Jupyter notebook environment, otherwise `top2vec`.
19. `wml_rag_deployment_id`: An identifier for the RAG deployed function that is set by the notebook when the RAG AI service Python function is deployed.
20. `watsonx_ai_url`: The URL for accessing the IBM watsonx.ai service on IBM Cloud from different environment. (optional)

21. `wx_ai_inference_space_id`: An identifier for a watsonx.ai deployment space that will run the LLM inferencing from a different environment. (optional)
22. `top_k_experts`: The maximum number of top recommended expert profiles to retrieve from the vector database query. Only required if ingestion and retrieval of expert profiles is required. (optional)
23. `expert_profiles_es_min_score`: The minimum score for results from the Elasticsearch query. Results below this threshold are not passed. Only required if ingestion and retrieval of expert profiles is required. (optional)
24. `prompt_deployment_id`: Deployment id of a prompt template in a deployment space. (optional)
25. `reuse_existing_space_assets`: Reuse the data assets already promoted to the deployment space by setting this `True` (optional)
26. `ai_guardrails`: Enable or disable AI guardrails for Harmful, Abusive, or Profane (HAP) content during LLM inference to ensure safe and compliant outputs.
27. `enable_pii_detection`: Flag that enables or disable the automatic detection of Personally Identifiable Information (PII) in input data.
28. `guardrails_hap_threshold`: Threshold for identifying and preventing Harmful, Abusive, or Profane (HAP) content in generated text.
29. `document_source_field`: Optional. The document field in the vector database that identifies the original source document. Used for re-merging of subsequent chunks in search result. Nested keys are separated by a '.' (dot). If left blank, not set or the specified field does not exist, chunk merging is deactivated.
30. `cos_data_path`: Path to the input dataset within the COS bucket. This is the relative path to the directory containing the input data files. If left empty, all TAR datasets in the bucket are used for processing and bulk indexing. (optional)
31. `bulk_writer_remote_data_path`: Path to the directory within the Milvus-associated COS bucket for storing processed data files for bulk ingestion. If left empty, the files are written to the root directory by default. (optional)
32. `num_workers`: Number of parallel processes to use for document processing into vector databases. Only required during bulk ingestion. (optional)

After you complete the steps in the **Prerequisites** and **Setting up the accelerator** sections, you run the accelerator by running each notebook in sequence or by running only the mandatory notebooks in a pipeline.

To run each notebook in sequence:

1. Go to the **Assets** tab and select the **Notebooks** asset type to see the notebooks that are provided with the project.
2. Edit each of the following notebooks by clicking the vertical ellipsis next to the notebook, clicking **Edit**, and then following the instructions in the notebook documentation to run the notebook cell by cell:
    - **Process and Ingest data into vector DB**
    - Optional. **Process and Ingest data from COS into vector DB**
    - Optional. **Test Queries for Vector Database**
    - Optional. **Ingest Expert Profile data to vector DB**
    - **Create and Deploy QnA AI Service**
    - **QnA with RAG**
    - Optional. **Analyze Log and Feedback notebook**

To run the accelerator pipeline:

1. Go to the **Assets** tab and select the **Flows** asset type.
2. Click the **QnA with RAG pipeline**.
3. Optional. Enable the deployment of the RAG function:
    1. Click the **Global objects** icon and click the **Edit** icon next to the **Deploy WML function** pipeline parameter.
    2. Select **True** as the default value, click **Update**, and close the window.
4. Click **Run pipeline > Trial run** to run the following notebook jobs in sequence:
    - **Process and Ingest data into vector DB**
    - **Create and Deploy QnA AI Service**
5. Edit the **QnA with RAG** notebook by clicking the vertical ellipsis next to the notebook, clicking **Edit**, and then following the instructions in the notebook documentation to run the notebook cell by cell.

Sample data assets

This project contains the following data assets:

- **elastic_search_ELSER_template.json**: The query template for Elastic Learned Sparse EncodeR search.
- **elastic_search_ELSER_BM25_hybrid_template.json**: The hybrid query template for Elastic Learned Sparse EncodeR search.
- **elastic_search_ELSER_BM25_hybrid_template_rrf.json**: A hybrid query template using Elastic Learned Sparse EncodeR (ELSER) and BM25 with Reciprocal Rank Fusion (RRF) for improved ranking by blending multiple retrieval scores.
- **elastic_search_hybrid_multisearch_template.json:** A multi-search query template for hybrid retrieval using multiple techniques, such as BM25 and ELSER, allowing simultaneous execution of different retrieval strategies.
- **elastic_search_hybrid_rrf_template.json:** A hybrid retrieval query template using Reciprocal Rank Fusion (RRF) to merge results from multiple retrieval approaches, such as BM25 and learned sparse embeddings.
- **elastic_search_multilingual_template.json**: The query template for E5 multilingual dense embedding search.
- **sample_elastic_search_NESTED_template.json**: The sample nested query template for Elastic search.
- **RAG-OpenAPI.json**: An Open API specification of the RAG deployed function to integrate the function into conversational AI using watsonx Assistant.
- **ibm-docs-SSYOK8.zip**: A ZIP file comprising 1710 HTML pages from the watsonx.ai documentation. You can use this file as sample data to run the accelerator.
  Important: This data asset contains a snapshot of the IBM watsonx as a Service documentation set taken on 19 Jun 2025.
- **expert_profiles_data.json**: A sample JSON file comprising 500 expert profiles across various domains. The profile information resembles professional resumes and is generated by an LLM. Alternatively, you can build your own expert profiles in a similar format and ingest them into the vector database.
- **IBM-WA-RAG-AI-ASSISTANT.zip**: A ZIP file consisting of sample files that are related to WatsonX AI Assistant actions which will be used to upload during creation of watsonx Orchestrate AI Assistant

extension for this RAG usecase.

Prompt Templates

This project contains the following prompt templates that specify an LLM and model parameters:

- **RAG template - llama-4-maverick-17b-128e-instruct**: Prompt template for the llama 4 maverick 17b model.
- **RAG template - granite-3-3-8b-instruct**: Prompt template for the granite 3.3 8b model.
- **RAG template - llama3-3_70b_instruct**: Prompt template for the llama 3.3 70b model.
- **RAG template - granite-3-2_8b_instruct**: Prompt template for the granite 3.2 8b model.
- **RAG template - granite-3_2b_instruct**: Prompt template for the granite 3.1 2b model.
- **Prompt_Topic_Labeling**: Prompt template used for generating labels for Topic detection of type Top2Vec using granite 3.3 8b model.

Note For latest updates on foundation models on SaaS. please check official documentation here

Notebooks

This project contains the following notebooks. Follow the instructions in the notebooks to run them.

Process and Ingest data into vector DB:

This notebook processes various types of data and convert data for subsequent ingestion into the vector index. The different data types processed are:

- PDF documents
- HTML files contained within a ZIP archive
- Any HTML content
- DOCX files
- PPTX files
- Markdown (MD) files
- TXT files

You must set several important parameters in the **RAG_parameter_set** parameter set before you run the notebook. See the **Setting up the accelerator** section.

The next steps in the notebook are:

1. Connect to the vector database.
2. Split documents into multiple segments with optimal overlap.
3. Load document segments into the vector index using the LangChain vector store.

Process and Ingest data from COS into vector DB:

This notebook facilitates the processing of PDF documents from TAR archives and prepares them for ingestion into Elasticsearch or Milvus vector indexes. It only supports PDF document processing. Before running the notebook, ensure you have configured all required parameters in the **RAG_parameter_set** and **RAG_advanced_parameter_set** sections. Refer to the **Setting up the accelerator** section for guidance.

The notebook workflow includes the following steps:

1. Connect to the vector database.
2. Connect to the AWS or IBM Cloud Object Storage.
3. Load documents from the specified AWS S3 or IBM Cloud Object Storage bucket.
4. For bulk indexing into the Elasticsearch vector database:
    1. Split documents into multiple segments with optimal overlap.
    2. Organize the segmented data for efficient bulk indexing.
    3. Perform bulk loading of the processed data into the Elasticsearch vector database.
5. For bulk indexing into the Milvus vector database:
    1. Split documents into multiple segments with optimal overlap.
    2. Store the segmented data in a Milvus-associated Cloud Object Storage bucket.
    3. Perform bulk loading of the processed data from the Milvus-associated bucket into the Watsonx Data Milvus vector database.

Note: It is recommended to execute this notebook as a job for bulk ingestion, as processing time may vary from several minutes to hours depending on the size of the input data.
Important: To ingest data directly from AWS S3 into a Watsonx Discovery index using the S3 connector, refer to the **Readme - Watsonx Discovery S3 Connector.pdf** document.

Test Queries for Vector Database:

This notebook demonstrates sample search on a vector database using different techniques:

- For Elasticsearch: use an Elastic Learned Sparse Encoder (ELSER) or a E5 Multilingual embedding model and LangChain to search and retrieve relevant documents based on specific queries.
- For Milvus: employ an embedding model and LangChain to search and retrieve relevant documents based on specific queries.

Ingestion of Expert profile data to vector db and retrieval:

This notebook processes expert profiles data from the JSON document format to ingest into a vector index. You can set a few parameters in the `RAG_Parameter_set` before you run the notebook. Otherwise, the default values of the parameters are used. See the **Setting up the accelerator** section above.
The next steps in the notebook are:

1. Connect to a vector database.
2. Load the documents profile wise into the vector index using the LangChain vector store.
3. Demonstrate a sample search on the vector database. The search and retrieval, like the **Test Queries for Vector Database** notebook, supports both Elasticsearch and Milvus.

Create and Deploy QnA AI Service:

This notebook creates and deploys an AI service Python function that takes a user-defined question as input and generates an answer using the RAG process. The main steps in the notebook are:

1. Vector Search: Connects to Elasticsearch or Milvus vector databases to retrieve the top N relevant documents from the vector index, filtering results based on a configurable score threshold.
2. Prompt Construction: Combines the user's question and retrieved documents into an optimized prompt template for the language model.

3. LLM Inference: Executes inference on IBM watsonx.ai to generate a response, supporting both streaming and non-streaming modes.
4. Hallucination Detection: Validates the generated response for accuracy using either word overlap or embedding-based cosine similarity techniques, applying configurable threshold values.
5. Feedback Logging: Updates log records with user feedback on the generated response, stored in Elasticsearch or Milvus, with PII suppression for compliance.
6. Expert Recommendation: Retrieves the top K expert profiles from a vector index in Elasticsearch or Milvus, matching the question based on a relevance threshold.
7. Expert Profile Logging: Updates log records with recommended expert profiles for future reference.
8. Autocomplete Functionality: Provides a function to retrieve answer suggestions by auto-completing the question based on a provided prefix, leveraging logged interactions in Elasticsearch or Milvus.
9. Guardrail Checks: Detects Personally Identifiable Information (PII) and Harmful, Abusive, or Profane (HAP) content in responses to ensure compliance.
10. Performance Monitoring: Tracks execution times for key pipeline stages, enabling optimization and debugging. **QnA with RAG**:

This notebook tests the deployed function by asking some questions, either by running a notebook, or in a more interactive mode using an ipywidget.

Alternatively, you can test the deployed function in your deployment space:

1. From the main menu, under **Deployments** select the **View all deployment spaces** option.
2. Navigate to **Spaces** tab and select your space.
3. Go to the **Deployments** tab and select **rag_ai_service** to see the endpoint for the function. (Depending on elasticsearch / milvus)
4. Go to the **Test** tab, paste the following code into the json input box, and then click **Predict**.

```
{"question": "how to run a project in watsonx.ai"}
```

Analyze Log and Feedback:

This notebook provides insights into user feedback.
This notebook implements log data retrieval, data preparation, topic modeling, and correlation analysis. It forms a basis for feedback analytics and is intended to be enhanced with more functions. The main steps in the notebook are:

1. Import log records.
2. Map feedback to percentage.
3. Calculate the rating distribution.
4. Run topic modeling.
5. Calculate scores by topic.
6. Calculate scores by response length.
7. Document search scores by topic.
8. List feedback on answers by topic.

Note: Default tool for topic modeling is `watson_nlp` if the **Watson NLP** library is present in the Jupyter notebook environment, otherwise `top2vec`. To use the watson_nlp library for topic modeling, you must run

the notebook in the `NLP + DO Runtime 24.1 on Python 3.11 XS` environment. See the Watson NLP documentation. The notebook requires a minimum amount of data in the log index to run the analytics. Therefore, before you run this notebook, populate the log index with data by running the deployed function with various questions.

Scripts

The project contains the following script:

- `rag_helper_functions.py`: Python script that provides many helper functions that are used throughout the notebooks.

Jobs

The project contains the following jobs:

- **Process and Ingest data into vector DB Job**: Runs the `Process and Ingest data into vector DB` notebook.
- **Create and Deploy QnA AI Service Job**: Runs the `Create and Deploy QnA AI Service` notebook.
- **QnA with RAG Pipeline Job**: Runs the `QnA with RAG watson pipeline` job.

Creating a watsonx Orchestrate assistant chatbot by using the RAG function

To create a watsonx Orchestrate assistant chatbot that calls the deployed RAG function:

1. Associate the watsonx Orchestrate service with the project:
    1. From the **Manage** tab, select **Services and Integrations**.
    2. Click **Associate Service**, select your watsonx Orchestrate service, and then click **Associate**.
2. To set up the chatbot, follow the instructions in the **Readme - RAG as watsonx Orchestrate AI Assistant extension.pdf** document. The PDF document is in the Data assets section on the **Assets** tab of the project.
3. Optional. If you enabled logging with the RAG function, you can add a feature to your watsonx Assistant skill that adds user feedback to the corresponding log records. Follow the instructions in the **Readme - RAG as watsonx Orchestrate AI Assistant extension.pdf** document. The prerequisite for this feature is that you must set the `log_connection_asset` parameters in the **RAG_parameter_set** parameter set.

Indexing Large Datasets with Watsonx Discovery S3 Connector

To setup the S3 connector with Wastonx Discovery, follow the instructions in the **Readme - Watsonx Discovery S3 Connector.pdf** document. The process involves several key tasks:

1. **S3 Connector Setup**: Connect an S3 bucket to Watsonx Discovery for automatic document ingestion.
2. **Elasticsearch Pipeline Configuration**: Set up a pipeline for chunking and indexing large documents to optimize performance.
3. **Handling Large Files**: For files exceeding 100MB, a Content Extraction Service is used, which can be deployed via Docker and integrated with the Wastonx Discovery S3 connector.

This process allows for scalable indexing, ensuring that documents in formats like PDFs are efficiently processed and stored in Elasticsearch, ready for querying and analysis.

Streamlit App for Question & Answer with RAG

In addition to the interactive Q&A with RAG notebook and the Watsonx Orchestrate assistant chatbot setup, a sample Q&A Streamlit application is also provided. This application demonstrates how to utilize the Q&A with RAG accelerator and offers a slick user interface that enables the execution of the aforementioned tasks with ease and efficiency. You can clone or download Q&A Streamlit application from the IBM Industry-Accelerators public GitHub repository. Go to `Industry-Accelerators/watsonx.ai/QnA_chatbot_app` folder, where you will find the setup and execution instructions in a README file included with the app.

The app supports the following features:

- Question and Answer based on your ingested documents via LLM inference.
- Retrieval of document links to support the responses.
- Retrieval of relevant expert profile data based on the question.
- Answer rating and feedback logging.

Terms and Conditions