GENERATIVE AI

Enterprise Data with LLMs

Enterprise Data with LLMs

Large Language Models (LLMs) are great but they are not aware of enterprise data. There are atleast three basic techiques to utilize LLMs with enterprise data

""GenAI : Unleashing the power of innovation, intelligence and imagination - a testament to the limitless possibilities when technology and human ingenuity converge"" *(ChatGPT)*

Build

Enterprises can build their own LLM by using the dataset which can consist only of enterprise data. This can be multi-purpose or domain specific

Finetune

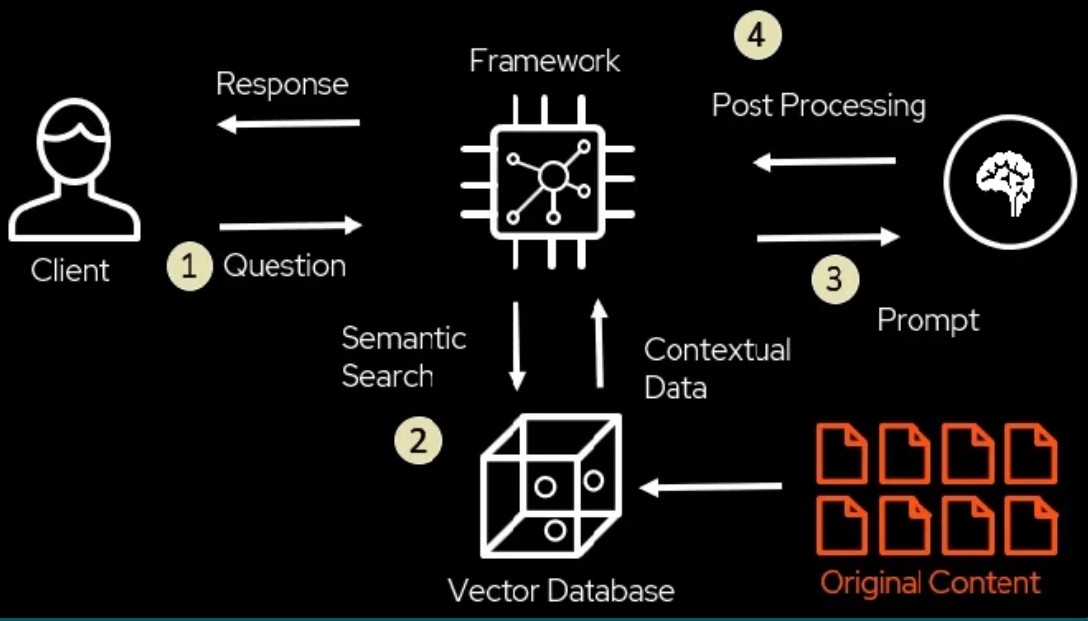
Fine-tuning is a supervised learning process where you use a dataset of labeled examples to update or add the weights of LLM and make the model improve its ability for specific task

RAG

Retrieval Augmented Generation *a.k.a* RAG is a simple technique to inject enterprise data based on a user query to the Prompt and then use existing LLM to generate response.

This document is focused on RAG

RAG - Basic Flow



1 Question

The user submits a natural language question or query. RAG systems are usually directed to solve limited set of problems. Domain specific questions which can be answered through specific knowledge base helps to reduce "noise" and "interference" which may be present in multi-domain knowledge base.

2 Search

The RAG framework performs a semantic search of a vector database containing embeddings from the specified data sources (which can be updated with new content). this can also be combined with "Keyword" where sources are certain and are required to provide better results. Before the search can happen, data needs to be chunked, converted into embeddings and stored in vector database.

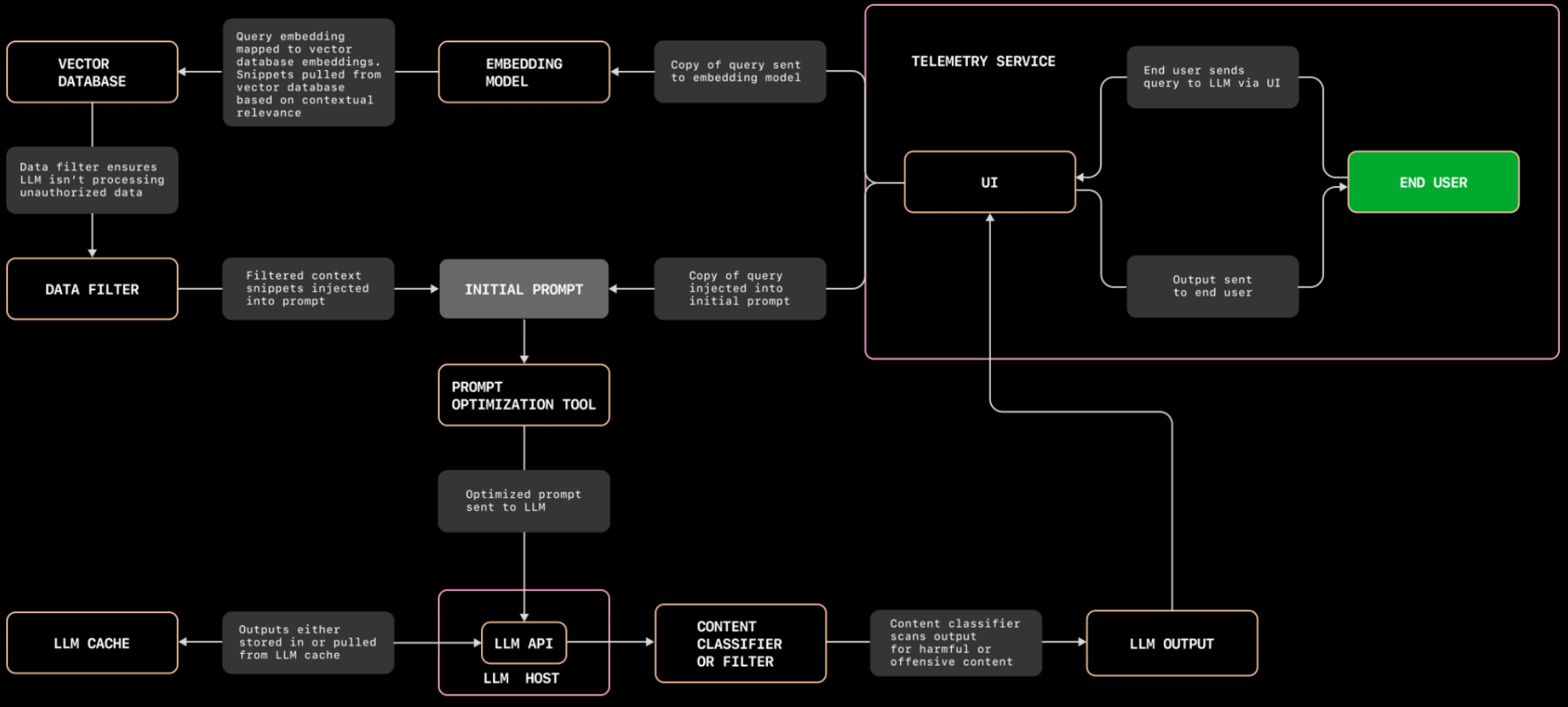
3 Prompt

The search results are submitted as a prompt to the LLM with the original question and based on provided prompt, LLMs generates a response. Prompts can be customized and enhanced based on domain specific needs. In some cases, chat history can also be added to the prompt so that LLM can have better context.

4 Post Processing

The LLM's response is then processed and sent to the framework, which formulates a response to the user. In the RAG model, the response is also likely to provide a link to some or all of the related source documents for the answer, similar to what keyword search applications provides.

RAG-Advance Flow



RAG Components

Prompt

Prompt is the way to ask questions to LLMs. This component serves as "User Interface" for users to interact with RAG systems. Users can type their queries or question regarding domain specific knowledge base/information which is already loaded into the system. They can also have an option to choose or customize system prompt from here to have better results. Other options like, selection of certain knowledge base, optionality have response from LLM with the referenced documents and enabling or disabling chat history can also be incorporated with this component.

Ingestion

RAG systems are highly dependent on data corpses which are unstructured data of any enterprise, can be in the form of files, webpages, source code, videos, and images. All the required data needs to be converted into chunks and then stored in vector database in the form of embeddings or vectors so that Retrieval can happen. In some cases, a direct access to actual documents can also be provisioned. Choice of domain specific data and their formats are critical to get better results. Chunking strategy and structure of indexes within vector databases also plays major role for quality results. For simple or non-prod scenarios, in-memory vector databases and search engines like FAISS can also be used.

Retrieval

This component serves as "*Search Engine*" for RAG systems. Certain searching algorithm can help us to identify the most similar or relevant chunks of information based on user query. Both "*semantic*", "*keyword*" or a combination of these two searching mechanism can be consumed. The retrieved documents (or chunks of documents) will then become a part of the Prompt which eventually serves as the input for the LLMs. This component can also utilize *Agents (specialized smart assistants)* which can drastically increase the productivity of LLMs.

Generation

Large Language Models (LLM) and Embedding Models (EM) provides the AI features to RAG based systems. EMs helps in converting documents into vectors or embedding for the Ingestion process while LLMs generates the actual response for the users. LLMs can also be consumed during Retrieval process to increase the relevance of search results if required. Parameters such as *temperature* and *topK* along with others can also alter the response from LLM.

Orchestration

This component is usually the "*business layer*" of any solution. A glueing layer which can help navigate among the other components of RAG. This can be designed in the form of microservices, set of APIs or as a monolithic API system. Different patterns which already exist in the industry to define this layer can be utilized. This layer can also provide a better mechanism test all the other components and then help in integration testing. Application frameworks like *Langchain* and *LlamaIndex* can be utilized in this component to perform rapid development.

AWS Services Matrix

Components	Services	Environments				Primary Purpose
		CFS 2.0	CFS 1.0	Launchpad - AWS	Launchpad - GovCloud	
Prompt	EC2	✓	✓	✓	✓	Hosting and data storage services. A web interface through which user can interact with the RAG system. Two factor authentication, prompt/response, sessions and chat history are some of the common
	ECS	✓	✓	✓	✓	
	EKS	?	?	✓	✓	
	DynamoDB	✓	✓	✓	✓	
	AWS Lex	✗	✗	✓	✓	
	RDS	✓	✓	✓	✓	
Ingestion & Retrieval	S3	✓	✓	✓	✓	Primarily the data storage for RAG systems with robust searching capabiltiies. Various services to store raw and vectorize data can be utilized. Semantic, keyword and similarity search algorithms become useful to provide search engine type capabilities.
	AWS OpenSearch	✓	✓	✓	✓	
	AWS OpenSearch Serverless	✗	✓	✓	✓	
	AWS Kendra	✗	✗	✓	✓	
	DocumentDB	✓	✓	✓	✓	
	Aurora PostgreSQL (pgvector)	✓	✓	✓	✓	
	AWS MemoryDB	✗	✗	✓	✗	
	AWS Bedrock	✗	✗	✓	✓	
	AWS SageMaker	?	?	✓	✓	
	AWS SageMaker JumpStart	✗	✗	✓	✗	
	Open Source Embedding Models	✗	✗	✓	✓	
Generation	EC2	✓	✓	✓	✓	The artificial intelligence centric layer for RAG systems. We may host open source LLMs and Embedding models as IaaS or we can leverage SaaS based services like BedRock.
	ECS	✓	✓	✓	✓	
	EKS	?	?	✓	✓	
	AWS Bedrock	?	?	✓	✓	
	AWS SageMaker	?	?	✓	✓	
	AWS SageMaker JumpStart	✗	✗	✓	✗	
	Open Source LLMs	✗	✗	✓	✓	
Orchestration	Lambda	✓	✓	✓	✓	Services which can help us build and test other components of RAG system in an scalable and secure manner. Data ingestion triggers, request routing and interactivity b/w services are some common features of this component.
	AWS SQS	✓	✓	✓	✓	
	AWS EventBridge	✓	✓	✓	✓	
	AWS API Gateway	✓	✓	✓	✓	
	EC2	✓	✓	✓	✓	
	ECS	✓	✓	✓	✓	
	EKS	?	?	✓	✓	
	NIS Integration Patterns	✓	✓	✗	✗	

AWS Service Matrix

✓ ***Available***
✗ ***Not available***
? ***Limited avaialbility***

Source : https://confluence.frb.org/pages/viewpage.action?spaceKey=Foundation&title=CFS+AWS+Services+List

Application Frameworks

LangChain

LlamaIndex

Introduction

LangChain is a framework that enables the development of data-aware and agentic applications. It provides a set of components and off-the-shelf chains that make it easy to work with LLMs. Whether you are a beginner or an advanced user, LangChain is suitable for simple prototyping and production apps.

LlamaIndex, previously known as the GPT Index, is an innovative data framework specially designed to support LLM-based application development. It offers an advanced framework that empowers developers to integrate diverse data sources with large language models

Key Features

- Building a wide range of Gen AI applications

- LangChain includes off-the-shelf chains, these provided pre-built chains can help accomplish specific tasks. These chains can be customized or used as a base for building new apps

- Suitable for applications that require complex interactions like chatbots, GQA, summarization

- Best for applications that require quick data lookup and retrieval

- LlamaIndex structures the ingested data into intermediate representations that are optimized for LLM consumption. This ensures efficient and performant access to the data

- Offers a straightforward interface for connecting custom data sources to large language models.

Helpful Info

Maintained at -
https://www.langchain.com/

Frontend Library -
LangChain.js

Base Framework -
Python

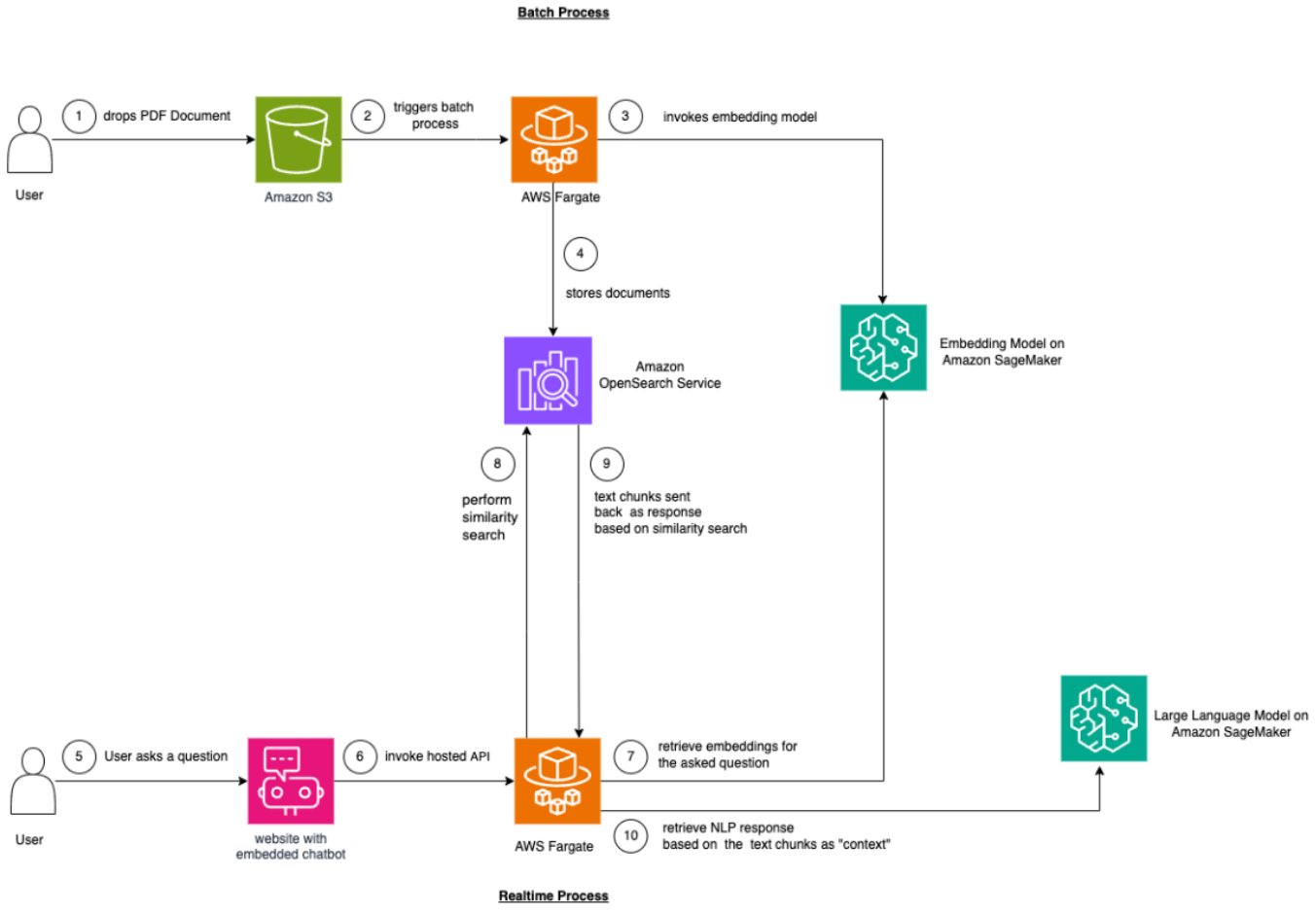
Maintained at -
https://www.llamaindex.ai/

Frontend Library -
LlamaIndex.ts

Base Framework -
Python

Both LangChain and LlamaIndex are powerful tools for building search and retrieval applications, leveraging the capabilities of large language models to extract insights from data. By understanding their unique features and differences, developers can choose the right tool for their specific needs and create powerful, efficient, and accurate search and retrieval applications. By following best practices for optimizing indexing performance and fine-tuning components, you can unlock the full potential of LlamaIndex and LangChain and create applications that truly stand out in the world of search and retrieval

Open Source Ref Architecture - Example One



Description

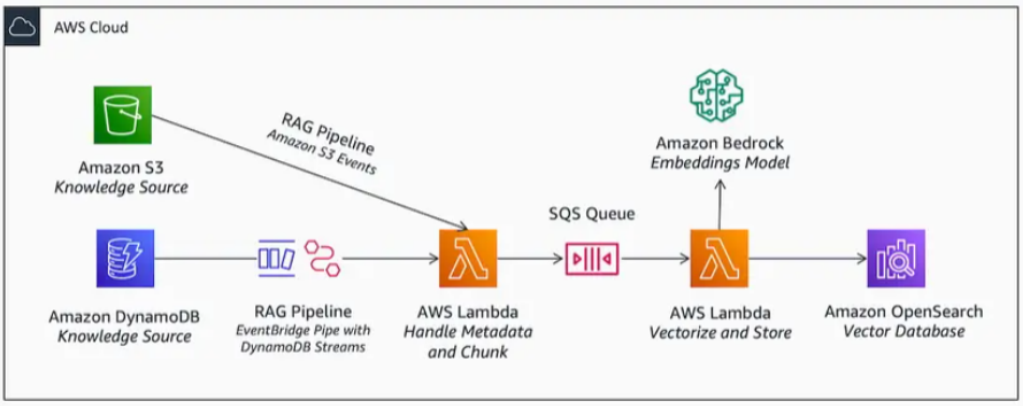
More Details : https://community.aws/tutorials/fullstack-llm-langchain-chatbot-on-aws

Services : API Gateway, S3,Lambda, Fargate, OpenSearch, SageMaker

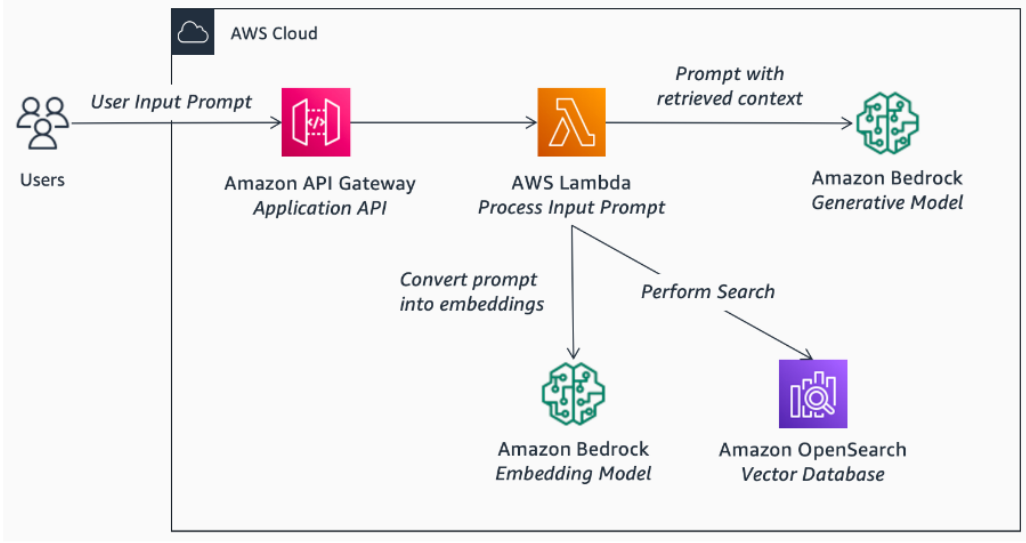
Overview : It's a simple architecture to host RAG application. Supporting batch ingestion of documents (conversion of documents into vectors). AWS OpenSearch is being used as vector database. Both Large Language Model (LLM) and Embedding Models (EM) are from open-source community and being hosted through AWS SageMaker and is being exposed through SageMaker's endpoints. For batch ingestion, when a document is going to placed inside S3, a Lambda function (container based) is being triggered. All the logic of reading and chunking of document(s) is written in python by using Langchain. The Lambda based container is hosted by using AWS ECS (Fargate). A simple node-based website for user to interaction and a backend API is also hosted on AWS ECS (Fargate). The API is communicating with AWS OpenSearch and AWS SageMaker endpoints to work as an orchestration layer. Within this architecture, we can easily replace AWS SageMaker with AWS Bedrock (SaaS) to leverage its LLM and EM capabilities or we can also host open source LLMs on EC2s as IaaS.

Open Source Ref Architecture - Example Two

Rag Pipeline



Rag Runtime



Description

More Details : https://community.aws/posts/build-generative-ai-applications-with-amazon-bedrock

Services : API Gateway, S3, Lambda, EventBridge, DynamoDB, SQS, OpenSearch, Bedrock

Overview : This scalable architecture consist of AWS serverless services like S3, API Gateway, Lambda, Bedrock and OpenSearch Serverless.Ingestion and Runtime components are separately defined. To orchestrate, open-source tool, such as LangChain is being used. LangChain contains pre-built libraries for integrating with various data sources and Bedrock. First, data converts from knowledge sources (such as Amazon S3 or Amazon DynamoDB) to an appropriate vector format for later retrieval. At runtime, application will need to process the user's original input prompt and augment it with the retrieved context.

Within this architecture, we can easily replace with AWS Bedrock (SaaS) with open source LLMs and EMs hosted on EC2s as IaaS or AWS SageMaker (PaaS).

SIO

Incubator Team : sys.innovation.office@frit.frb.org