LANGCHAIN — Structuring Docs with Indexes.

① Document loaders: RAG approach, diff sources to the LLMs — embeddings.

Component → for loading documents from various sources — DB, online store, etc.

Documents — HTML, PDF, code.

```
from langchain.document_loaders import S3FileLoader

loader = S3FileLoader("my_bucket", "sample2.docx")
data = loader.load()
```

② Retrievers ⇒ RAG applications on AWS, you can use Amazon Kendra to index & query various Data Sources.

Amazon Kendra = pre-built connectors
- S3, SP, Confluence, websites
- HTML, Word, PPT, Excel, PDF, etc

AmazonKendraRetriever method.

```
from langchain_aws.retrievers import AmazonKendraRetriever
from langchain.chains import ConversationalRetrievalChain
from langchain.prompts import PromptTemplate
from langchain_aws import ChatBedrock

llm = ChatBedrock(
    model_kwargs={"max_tokens_to_sample":300,"temperature":1,"top_k":250,"top_p":0.999,"anthropic_version":"bedrock-2023-05-31"},
    model_id="anthropic.claude-3-sonnet-20240229-v1:0"
)

retriever = AmazonKendraRetriever(index_id=kendra_index_id,top_k=5,region_name=region)

prompt_template = """Human: This is a friendly conversation between a human and an AI.
The AI is talkative and provides specific details from its context but limits it to 240 tokens.
If the AI does not know the answer to a question, it truthfully says it
does not know.

Assistant: OK, got it, I'll be a talkative truthful AI assistant.

Human: Here are a few documents in <documents> tags:
<documents>
{context}
</documents>
Based on the above documents, provide a detailed answer for, {question}
Answer "do not know" if not present in the document.

Assistant:
"""

PROMPT = PromptTemplate(
    template=prompt_template, input_variables=["context", "question"]
)

response = ConversationalRetrievalChain.from_llm(
    llm=llm,
    retriever=retriever,
    return_source_documents=True,
    combine_docs_chain_kwargs={"prompt":PROMPT},
    verbose=True)
```

③ Vector Stores

RAG Approach, steps

1. You convert data into embeddings (Text embedding model)
2. You can store the embeddings in Vector DB.
3. You can extract the relevant Docs based on the user request from vector DB.
4. LLM as context for the response.

Langchain — open source — provider specific vector stores.

```
import os
from langchain_community.embeddings import BedrockEmbeddings
from langchain_community.vectorstores import OpenSearchVectorSearch

index_name = os.environ["AOSS_INDEX_NAME"]
endpoint = os.environ["AOSS_COLLECTION_ENDPOINT"]

embeddings = BedrockEmbeddings(client=bedrock_client)

vector_store = OpenSearchVectorSearch(
    index_name=index_name,
    embedding_function=embeddings,
    opensearch_url=endpoint,
    use_ssl=True,
    verify_certs=True,
)
retriever = vector_store.as_retriever()
```

Langchain agents → External sources (search engine, calculator, APIs, DBs)

LLMchain = Basic chain.

① RAG Application — LLM chain → Response ← user query, context (or set of docs), vector store.

② Router chain = user input → Admin, Fin, HR

③ Langchain agents = Reasoning engine LLM, Action can be tool → search engine, math calculator.