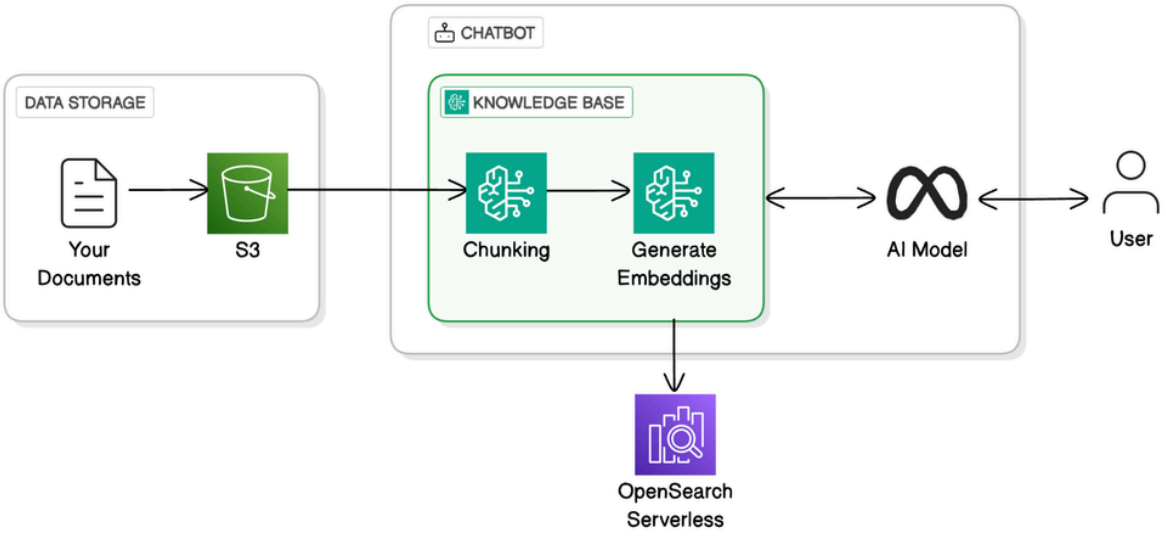
Kareshma Rajaananthapadmanaban

Set Up a RAG Chatbot in Bedrock

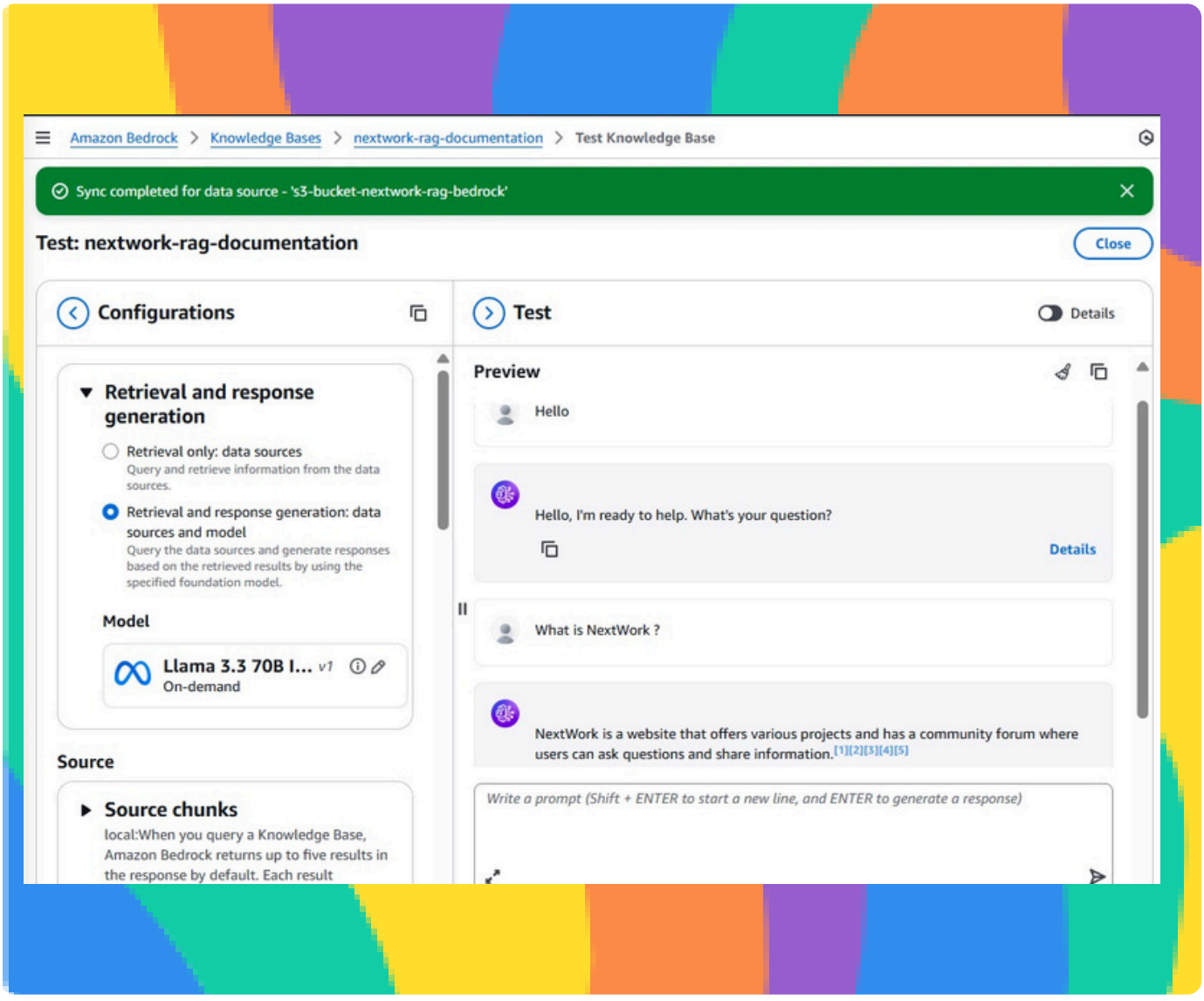
Build an AI chatbot that learns from your data with RAG and Amazon Bedrock



nextwork.org

Set Up a RAG Chatbot in Bedrock

Kareshma Rajaananthapadmanaban



Introducing Today's Project!

RAG (Retrieval Augmented Generation) is an AI technique that enhances a models ability to provide **context-specific** answers by integrating retrieval mechanisms with generation. In this project, I will demonstrate RAG by training an AI chatbot using documents stored in Amazon S3. The chatbot will **retrieve relevant data** from a **Knowledge Base** indexed in **Amazon OpenSearch** and generate human-like responses based on the documents.

Tools and concepts

Services I used were **Amazon Bedrock, S3, and OpenSearch.** Key concepts I learnt include creating and managing a Knowledge Base, setting up vector stores for retrieval, and integrating S3 as a data source. I also explored guardrails, query modification, and best practices like cleaning up unused resources to avoid extra costs.

Project reflection

This project took me approximately 1 hour. The most challenging part was understanding how to **fine-tune the knowledge base** settings for better responses. It was most rewarding to see the chatbot provide more detailed and contextual answers after the improvements.

I did this project today to strengthen my hands-on skills with RAG in Amazon Bedrock and to better understand how to improve **chatbot responses with advanced configurations.** Yes, this project met my goals as I was able to practice key steps like tuning source chunks, custom prompts, and cleaning up resources, which boosted both my learning and confidence.


Kareshma
Rajaananthapadmanaban

Understanding Amazon Bedrock

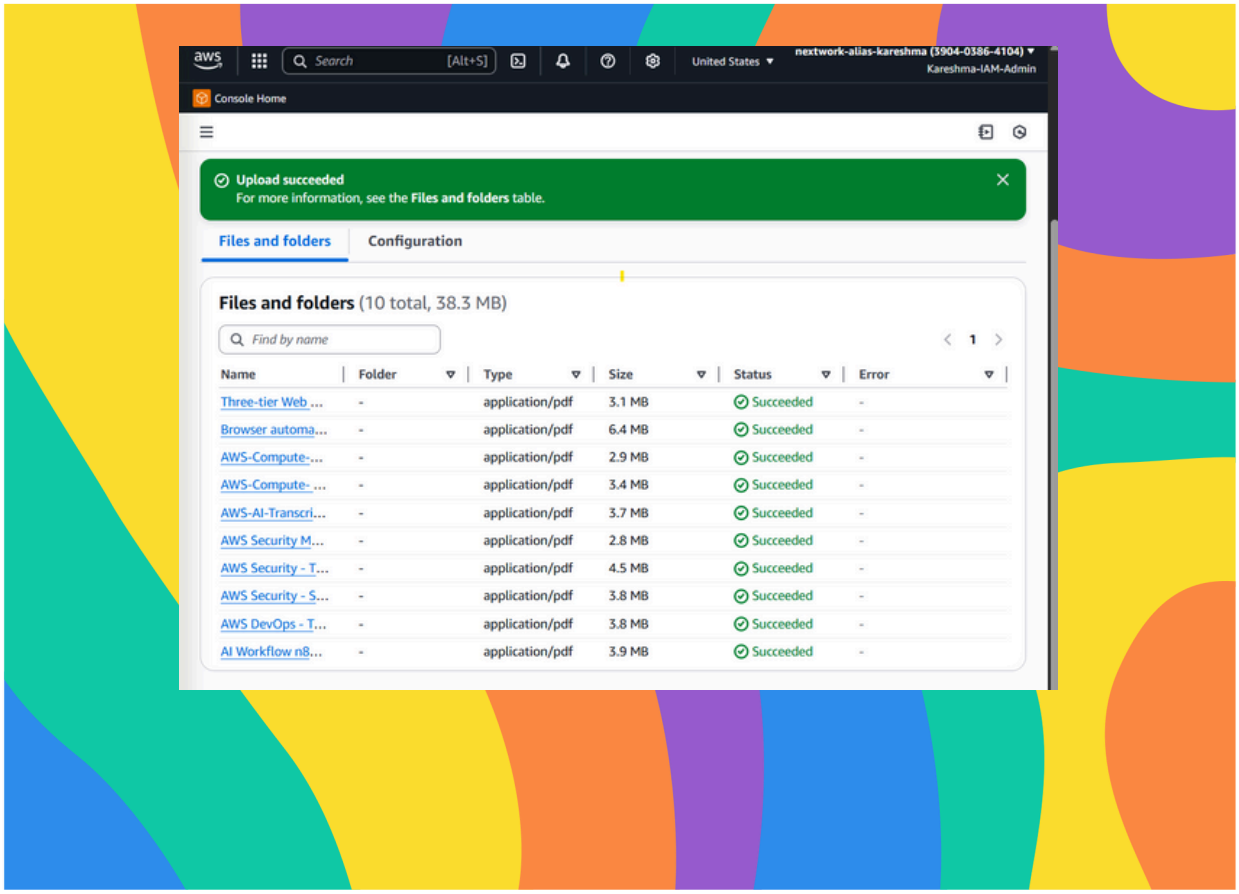
In this step, I will create a Knowledge Base (KBase) in Amazon Bedrock. This will serve as the **chatbot's personal library**, where all the information it needs to answer queries will be stored.

Additionally, I will set up an S3 bucket to hold the documents that will populate the Knowledge Base. The **KBase** will help the chatbot retrieve relevant information and provide accurate responses.

Amazon Bedrock is a managed service that provides access to AI models from top providers like OpenAI, Anthropic, and Meta, all within a single platform. It simplifies integrating AI models into applications by eliminating the need for separate sign-ups and API management. I'm using Bedrock in this project to **create a chatbot** that can interact with my **personal data** through a Knowledge Base and respond to queries using Retrieval Augmented Generation (**RAG**).

My Knowledge Base is connected to S3 because S3 is a scalable and secure storage service for my documents. By selecting **Amazon S3** as the **data source** for my Knowledge Base, I can store and manage my files separately from the chatbot itself. This allows me to easily organize, update, and access my documents, while Bedrock uses them to train the chatbot and generate context-specific responses.

In this step, I will create an S3 bucket to store the documents that will **populate** my **chatbot's Knowledge Base**. After setting up the bucket, I will upload the necessary files, such as documentation or personal data, that the chatbot will use to generate responses. These documents will be linked to the Knowledge Base in Amazon Bedrock, enabling the chatbot to retrieve relevant information during interactions.



Files getting uploaded into your S3 bucket.

In an S3 bucket, I uploaded my project documentation related to AI, AWS Security, and DevOps. My S3 bucket is in the same region as my Knowledge Base because keeping resources in the **same region** ensures **lower latency** and **optimized performance.**

This alignment ensures that the documents are readily accessible by Bedrock without any delays, allowing the chatbot to retrieve and use the content effectively for responses.

My Knowledge Base Setup

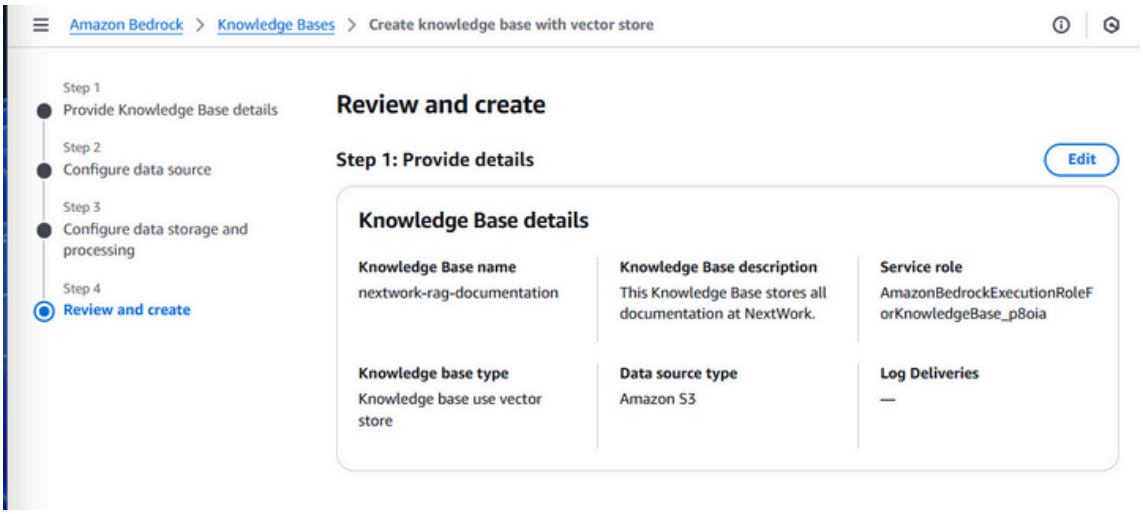
In this step, I will finish setting up the Knowledge Base in Bedrock by selecting the S3 bucket as the data source. This will connect the documents I uploaded to the Knowledge Base, making them available for the chatbot to use when generating responses. I'll ensure that the **S3** bucket is **properly linked,** so the chatbot can retrieve the relevant information from the documents during interactions.

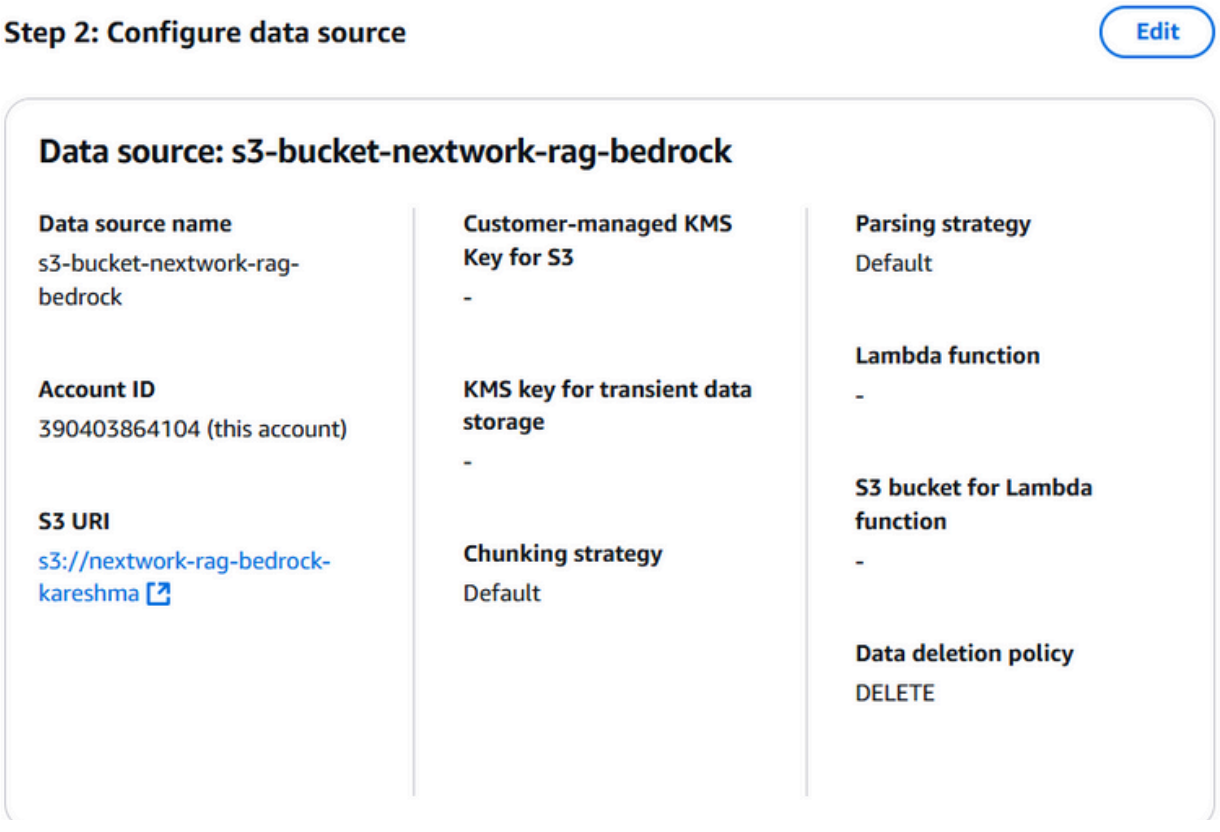
My Knowledge Base uses a **vector store**, which means it **stores the embeddings** of my documents, allowing the AI to understand the meaning behind the text. When I query my Knowledge Base, OpenSearch will efficiently search through the stored embeddings to find the most **relevant text chunks** based on the meaning of my question, not just matching words. This makes the search process faster and more accurate, ensuring the chatbot provides **contextually relevant responses**.

Embeddings are **numerical representations of text** that capture the meaning, themes, and context of the content. The embeddings model I'm using is Titan Text Embeddings v2 because it's fast, accurate, and designed to work seamlessly with AWS services. It helps the chatbot match questions with the most relevant documents in my Knowledge Base, even when the exact wording doesn't match, by **comparing** the **numerical patterns** of the question and documents.

Chunking is the process of **breaking large texts** into **smaller**, more manageable sections to help AI models process information efficiently. In my Knowledge Base, chunks are set to **300 tokens**, which is approximately 300 words or punctuation marks. This ensures the chatbot can handle the text without hitting processing limits, enabling more accurate and faster responses from the Knowledge Base.

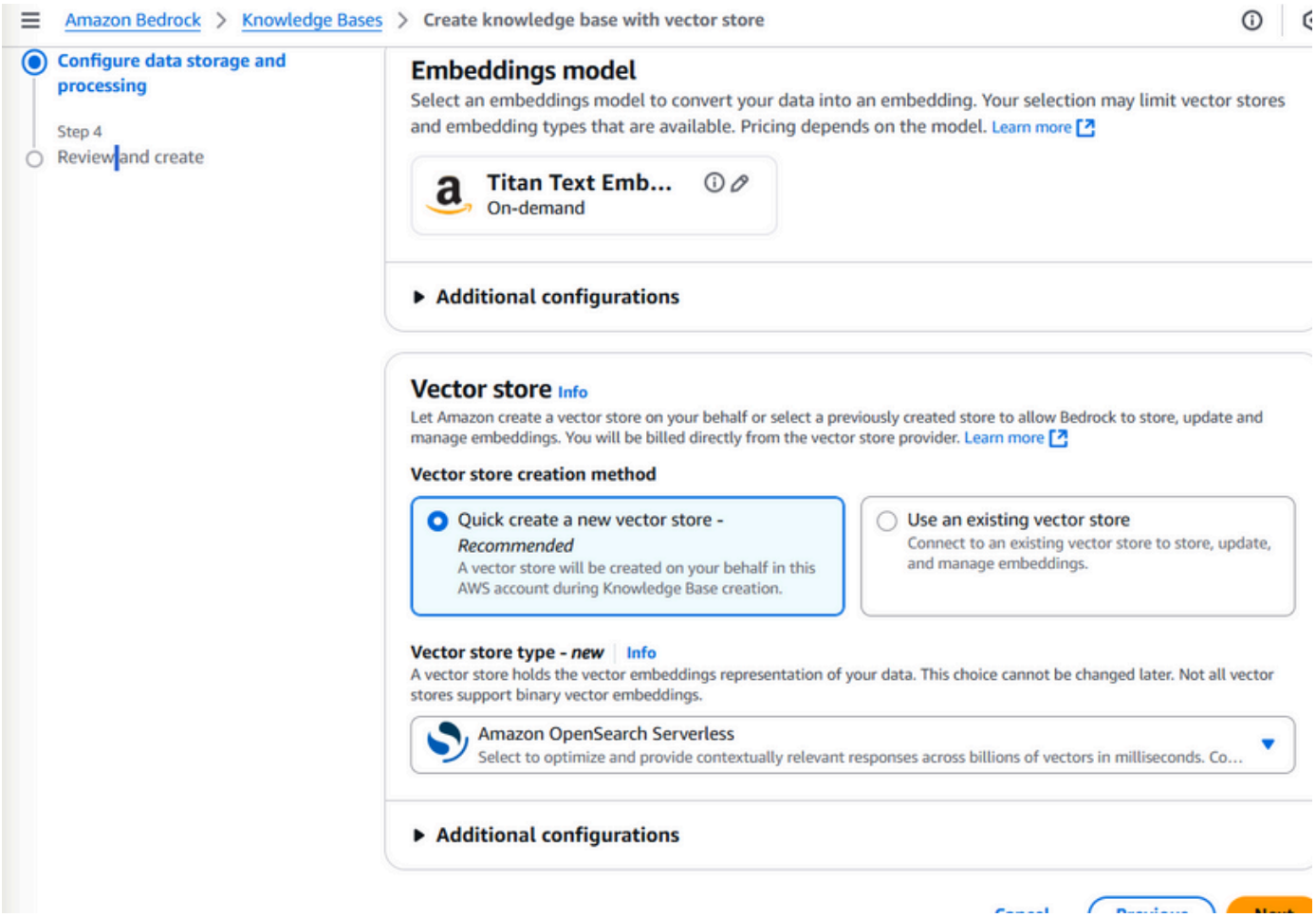
Configuration of Knowledge Base

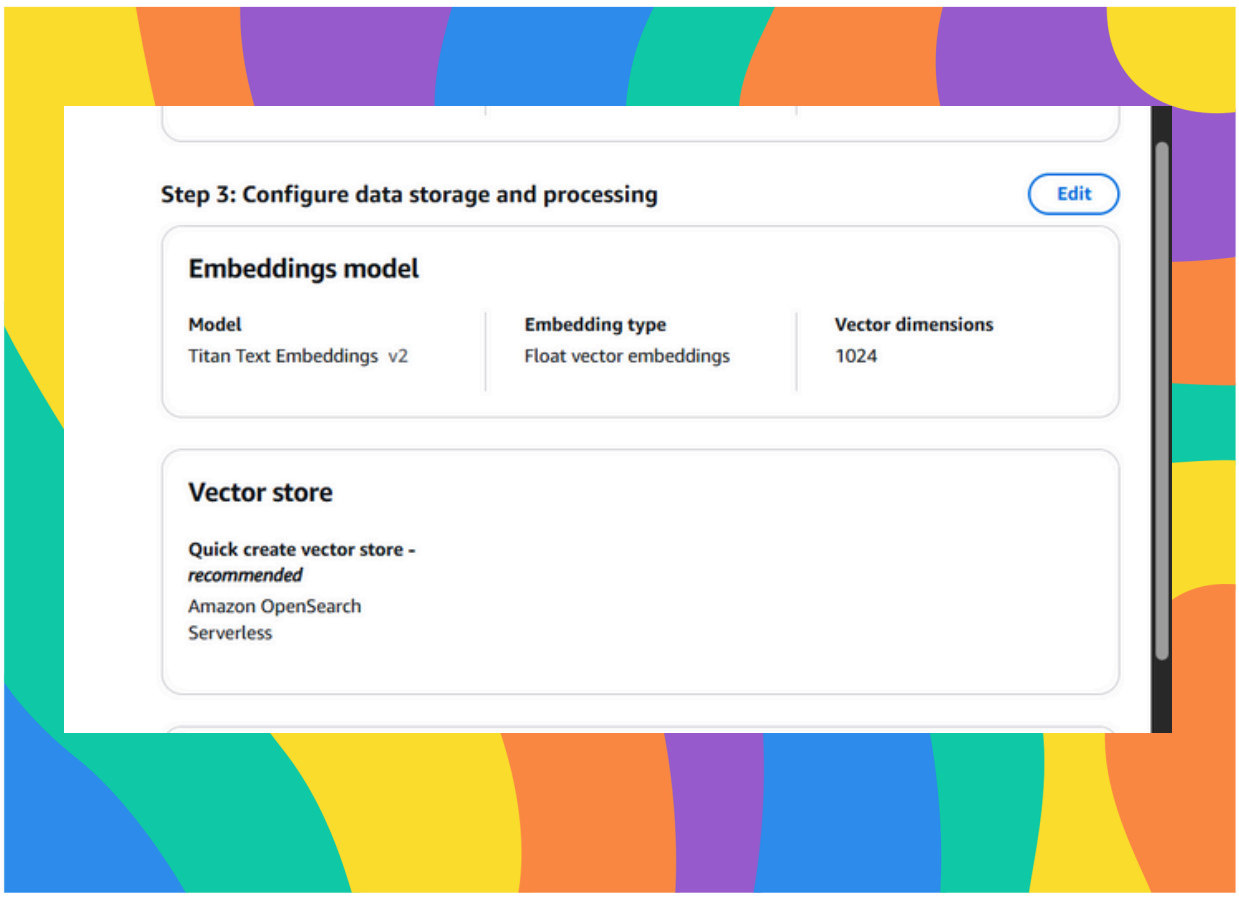




Kareshma
Rajaananthapadmanaban

Configuration of Knowledge Base





Knowledge Base review

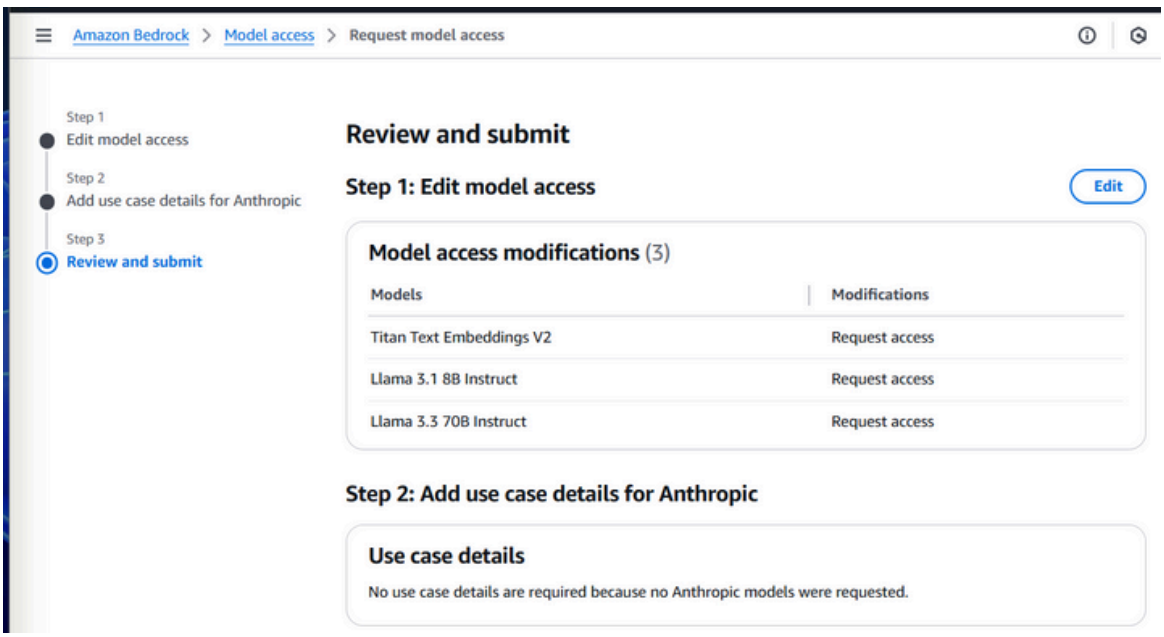
AI Models

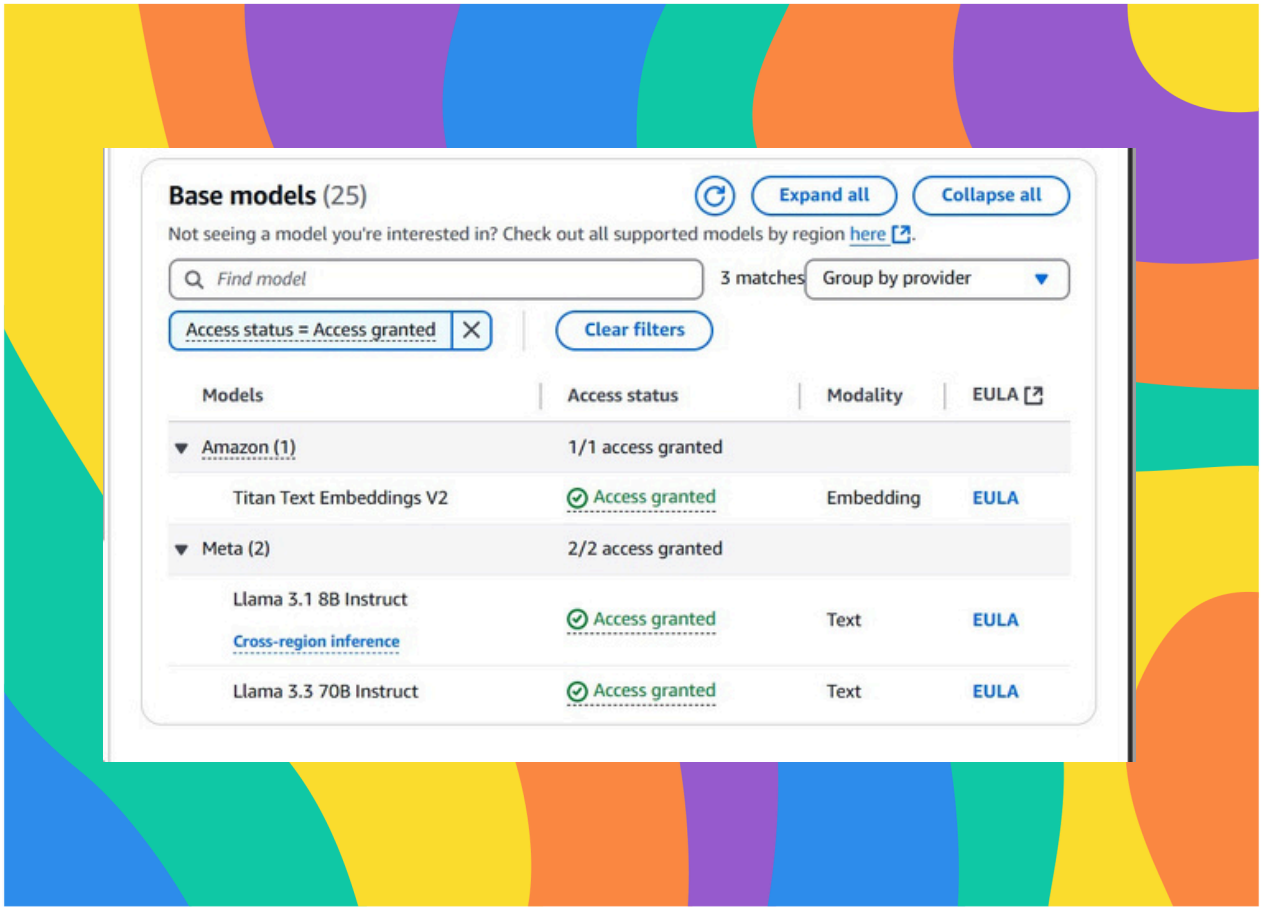
AI models are important for my chatbot because they transform the **raw data retrieved** from the Knowledge Base into human-like, contextually relevant responses.

Without AI models, my chatbot would **only provide chunks of text** from the documents, making the conversation feel mechanical and disjointed. These models allow the chatbot to understand and generate smooth, natural language, enhancing the user experience by making it interactive and intelligent.

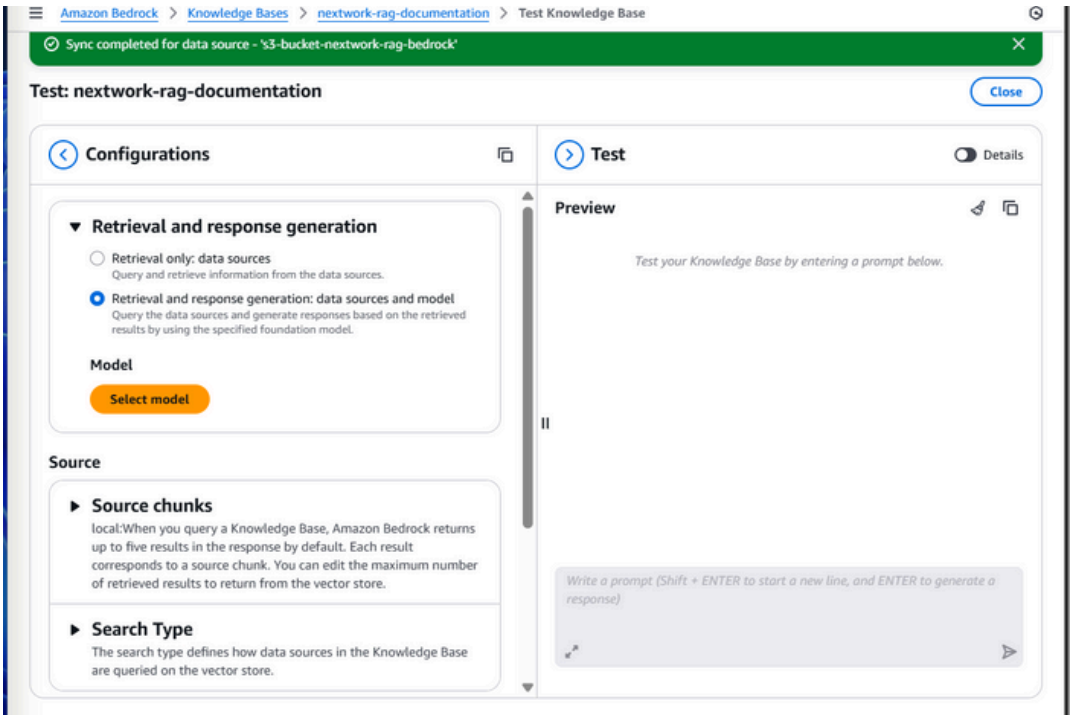
To get **access to AI models** in Bedrock, I had to **enable** specific models by **requesting explicit access** through the Bedrock console. AWS needs explicit access because some models are costly, require additional server capacity in my region, and come with specific usage conditions.

This process ensures that I am aware of the model's requirements and costs before integrating them into my project. I selected the Titan Text Embeddings V2 and the Llama models for my chatbot.





Model access granted



Kbase in grey have to select model

Syncing the Knowledge Base

In this step, I will **synchronize** the **data** from my **S3 bucket** to the **Knowledge Base.** This step ensures that the documents stored in S3 are loaded into the Knowledge Base, making them **available for the AI models** to access and use when generating responses.

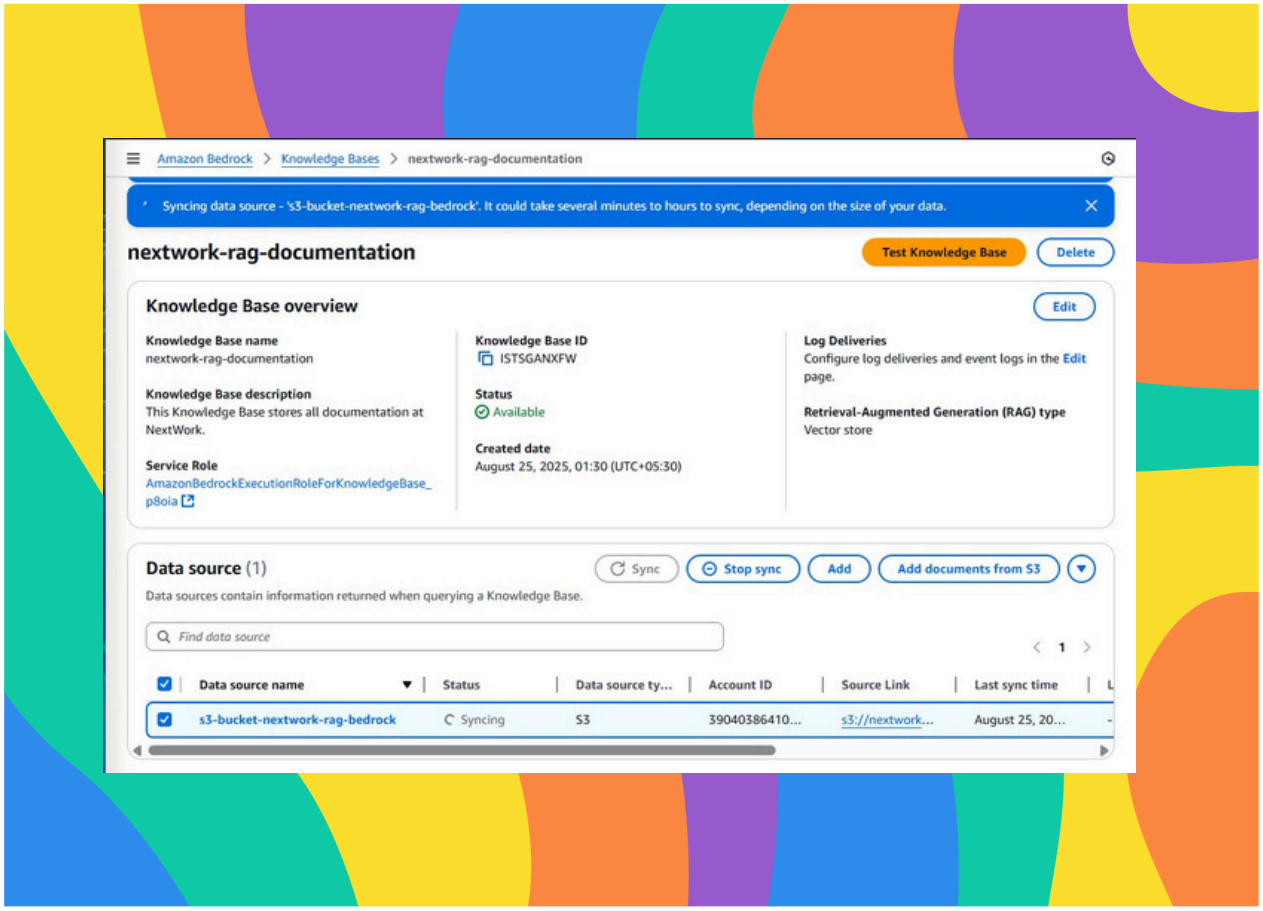
Without this synchronization, the Knowledge Base would not have the data it needs to answer queries accurately.

Even though I already connected my S3 bucket when creating the Knowledge Base, I still need to **sync** because the data has not yet been transferred from S3 to the Knowledge Base.

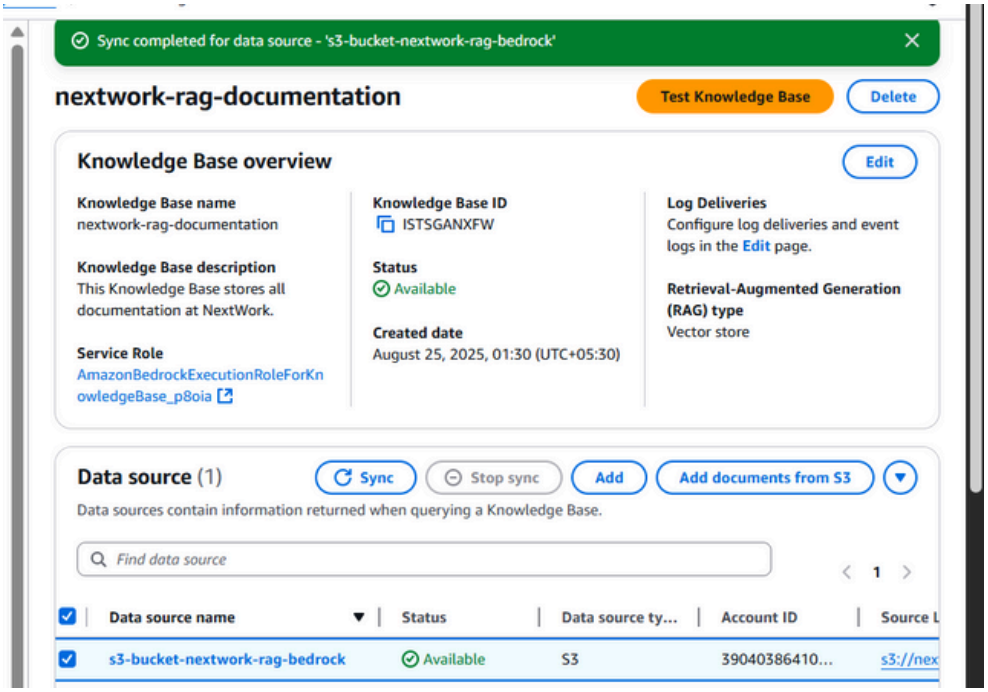
Syncing ensures that the documents in the S3 bucket are **properly loaded** into the **Knowledge Base**, making them available for the AI models to retrieve and use when generating responses. Without this step, the chatbot wouldn't have any data to work with.

The sync process involves **three steps**: first, **ingesting**, where Bedrock retrieves the documents from the S3 bucket. Next is **processing**, where the documents are broken into smaller chunks and transformed into embeddings, which are numerical representations the model can work with.

Finally, **storing** takes place, where these embeddings are saved in the vector store (OpenSearch Serverless) so the chatbot can quickly search and retrieve relevant information during queries.



Knowledge Base sync in progress



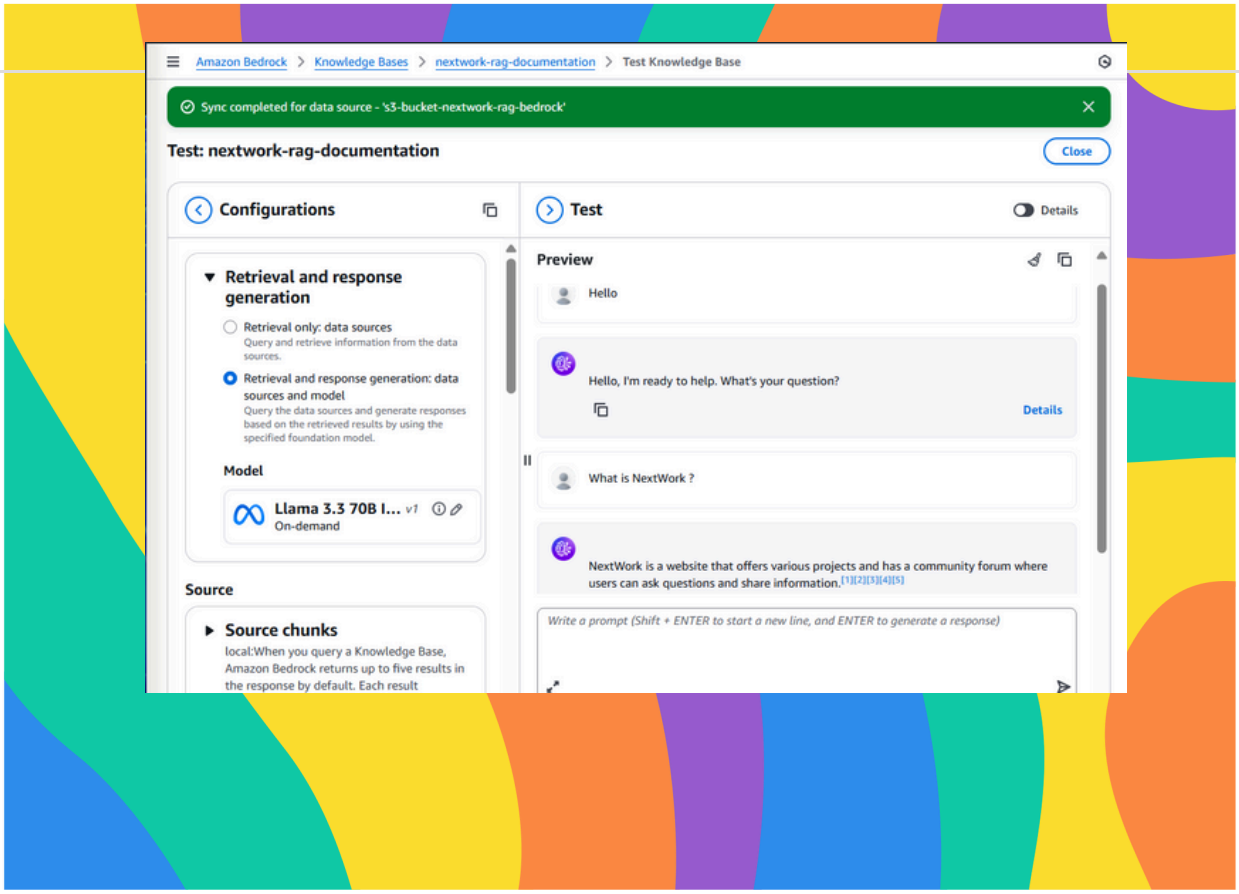
Testing My Chatbot

In this step, I will test the chatbot to see if it can **generate personalized responses** based on the data synced from my S3 bucket. I'll also troubleshoot any issues that come up with the AI model, such as errors in retrieving information or problems with the responses, to make sure the chatbot is working as expected.

I initially tried to test my chatbot using Llama 3.1 8B as the AI model, but it required **pre- provisioned inference**, meaning it could only run efficiently when dedicated resources were always kept running. Since it **doesn't** support **on-demand inference**, I faced errors while testing. I had to switch to Llama 3.3 70B because it supports on-demand inference, allowing the model to **start up only** when needed, making it both faster to test and more cost-effective for my use case.

When I asked about **topics unrelated** to my data, my chatbot either responded with an error or stated that it couldn't find relevant information. This proves that the chatbot is **limited** to the **knowledge** I uploaded and doesn't pull from outside sources. It only answers based on my data, which is exactly how a secure and reliable knowledge- based chatbot should work.

You can also turn off the Generate Responses setting to see only the raw chunks of information retrieved directly from your Knowledge Base. In this mode, the chatbot doesn't create a summarized or conversational answer it just shows the text blocks that matched your query. This is useful if you want to analyze the raw data yourself without model interpretation, letting you confirm exactly what content the Knowledge Base is returning for your queries.

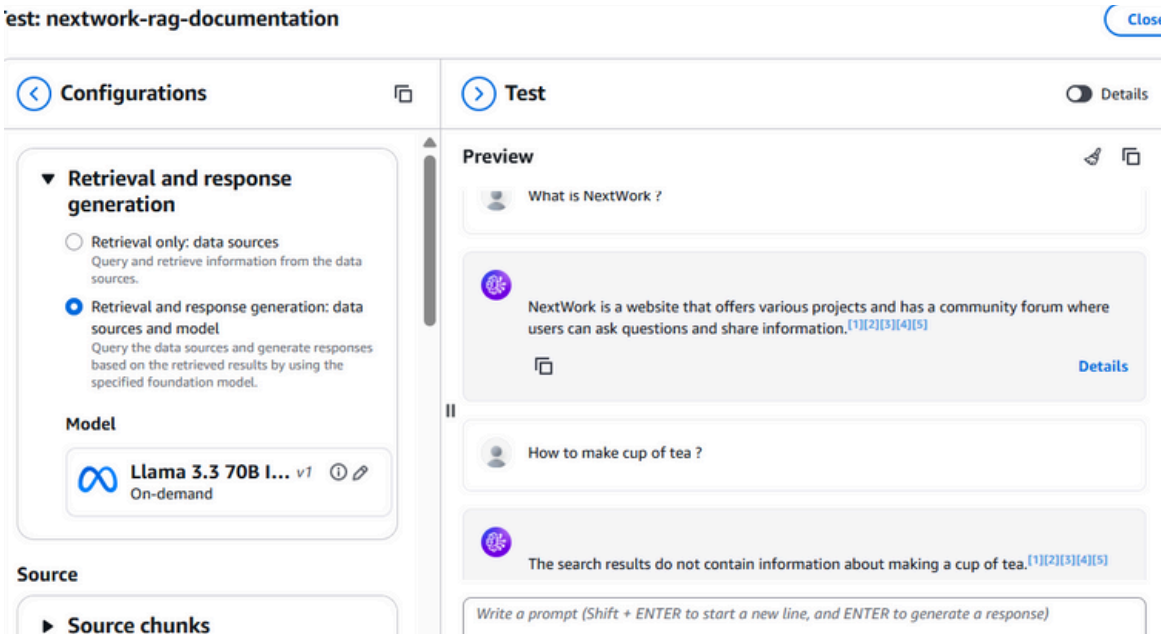


Chatbot's response

When you switch Generate Responses back on, the AI model processes those same chunks and delivers a fluent, summarized response. However, I noticed during my own setup that the **UI has changed** and the toggle option isn't visible anymore. Instead, it works by **switching** between "retrieve only data source" and "retrieve plus generate response" modes. The effect is the same you can either get plain data or a polished chatbot answer depending on your selection.

Chat with Chatbot

When I try asking a **question** that's **not related** to the **data** uploaded.

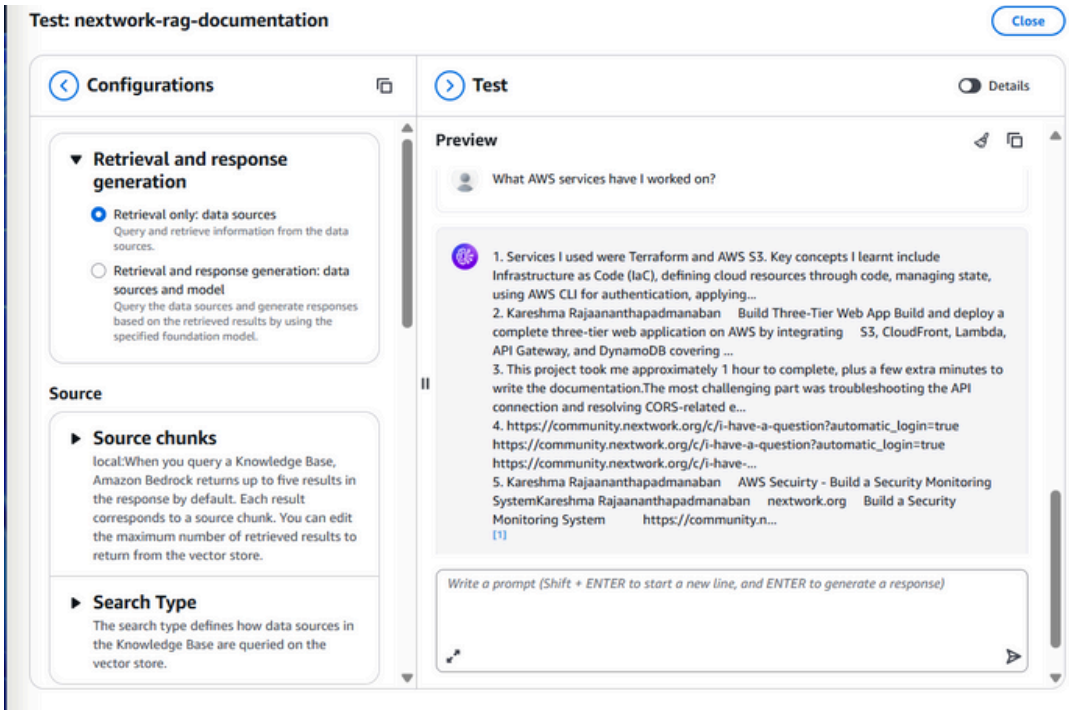


How do you make a cup of tea?

This proves to you that your chatbot has **no knowledge** outside of your data. Now that's tea! 🍵

Chatbot response difference

When I try asking a same question by switching the **Retrieval and response generation**

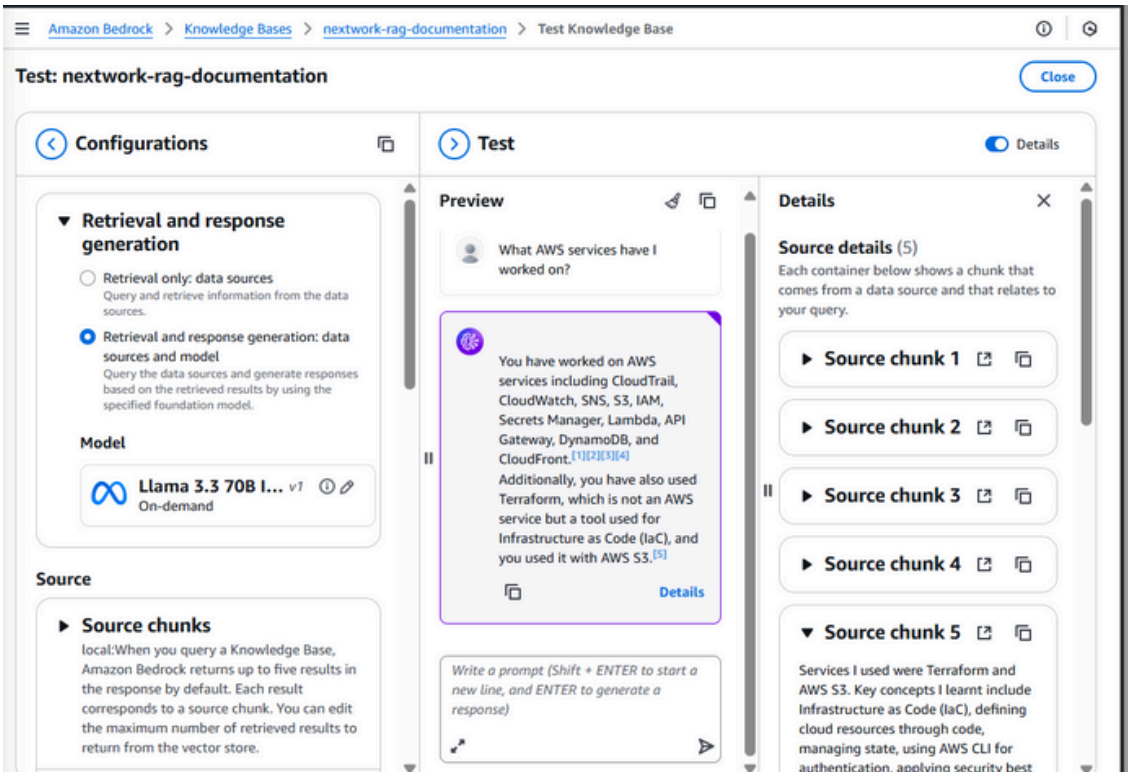


What AWS services have I worked on ?

Oops, the chatbot responds with chunks of text from the data source. When the ***Retrieval only: data sources*** setting is ON , the AI model will no longer translating the Knowledge Base's search results into a chat response.

Chatbot response difference

When I try asking a same question by switching the **Retrieval and response generation to Data sources and model**



What AWS services have I worked on ?

Now, chatbot goes back to responding with a **detailed summary** of the AWS services that I have worked on.

This is a great option if you just want to retrieve the processed data directly from your Knowledge Base, to analyze it further yourself.

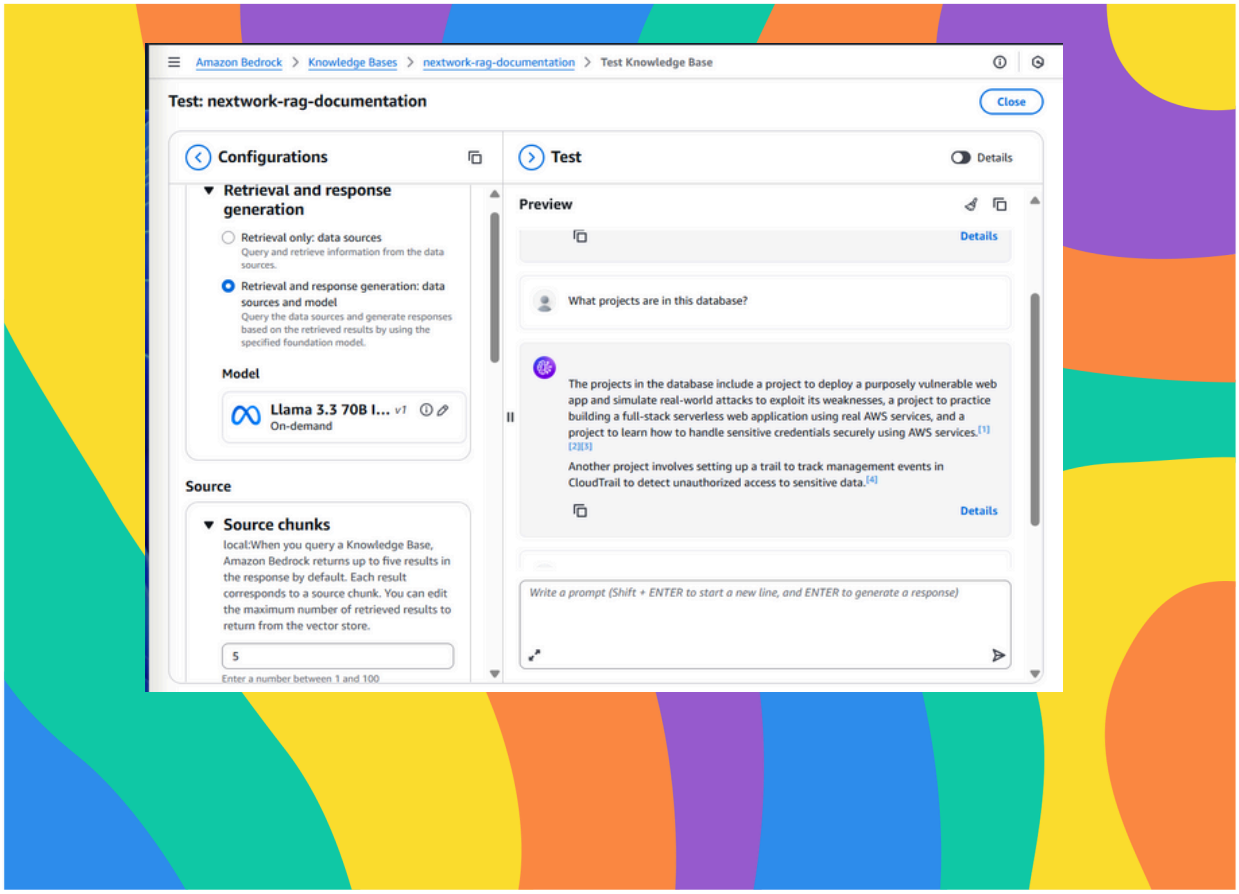
Upgrading My Chatbot

In this secret mission, I will enhance my chatbot by **tuning its RAG configuration** to provide richer, more contextual answers. This involves increasing the number of knowledge chunks it can pull from, so it has a broader base to generate accurate responses.

I'll also design **custom prompts** to guide the chatbot's tone and detail, then test and compare the output quality. The goal is to showcase advanced RAG skills that make my chatbot smarter and more reliable.

In a project extension, I looked into **increasing the number** of source chunks, which are the **pieces of text** my chatbot retrieves from the database to answer a question. By raising this limit, the chatbot can access a broader range of context at once. This will improve my chatbot's responses by making them more complete, accurate, and detailed, since it won't miss important parts of the data.

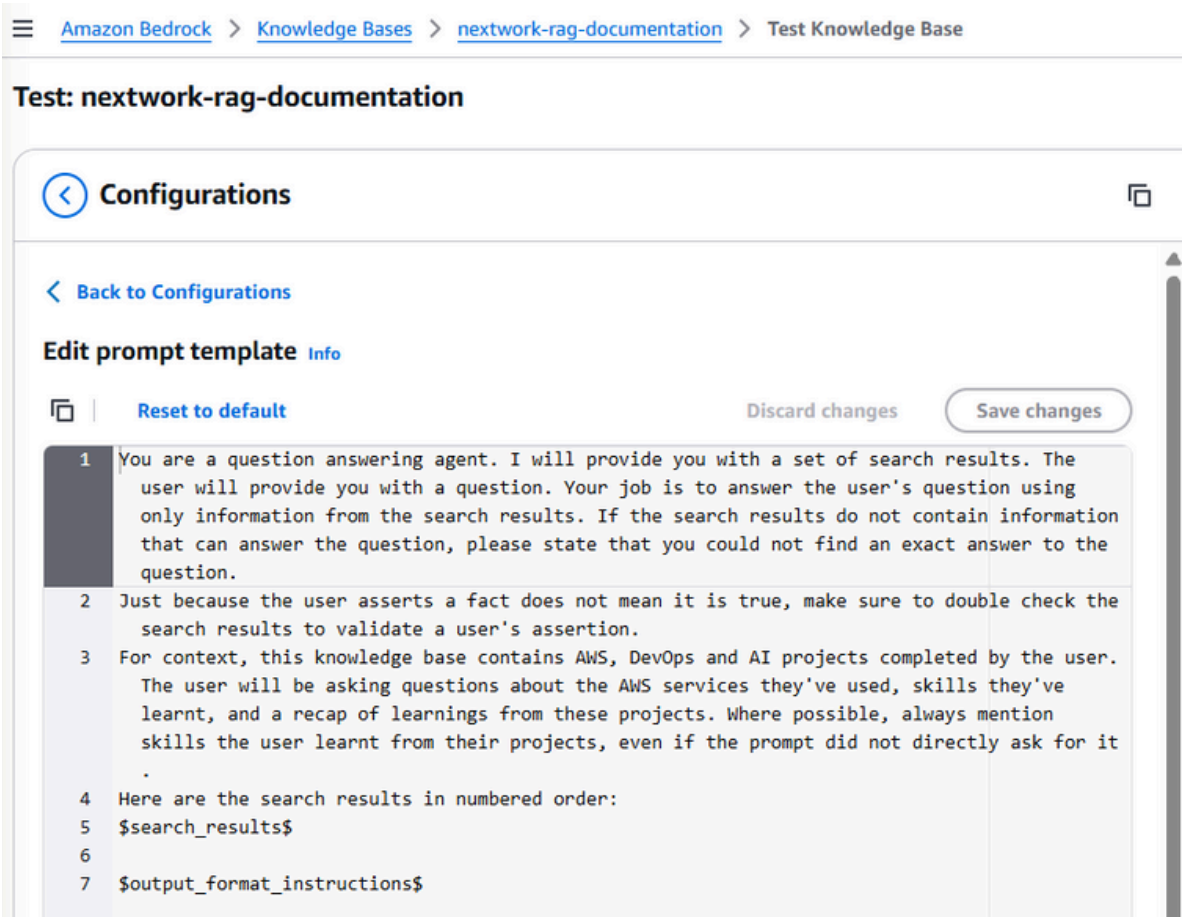
I also added a **custom prompt that tells my chatbot** to treat the knowledge base as a collection of AWS, DevOps, and AI projects completed by me. It guides the chatbot to mention the AWS services used, highlight the skills I learned, and provide a recap of my project learnings. This way, even if the question doesn't directly ask about skills, the chatbot will still include them for more **context-rich answers.**



Before Improved response with edit prompt

Before adding the **edit prompt template** and increasing **source chunks** to **10**, the chatbot gave shorter, **less detailed answers**. It often returned only a limited number of projects from the database, missing important context or additional entries, which made the responses feel incomplete and less informative.

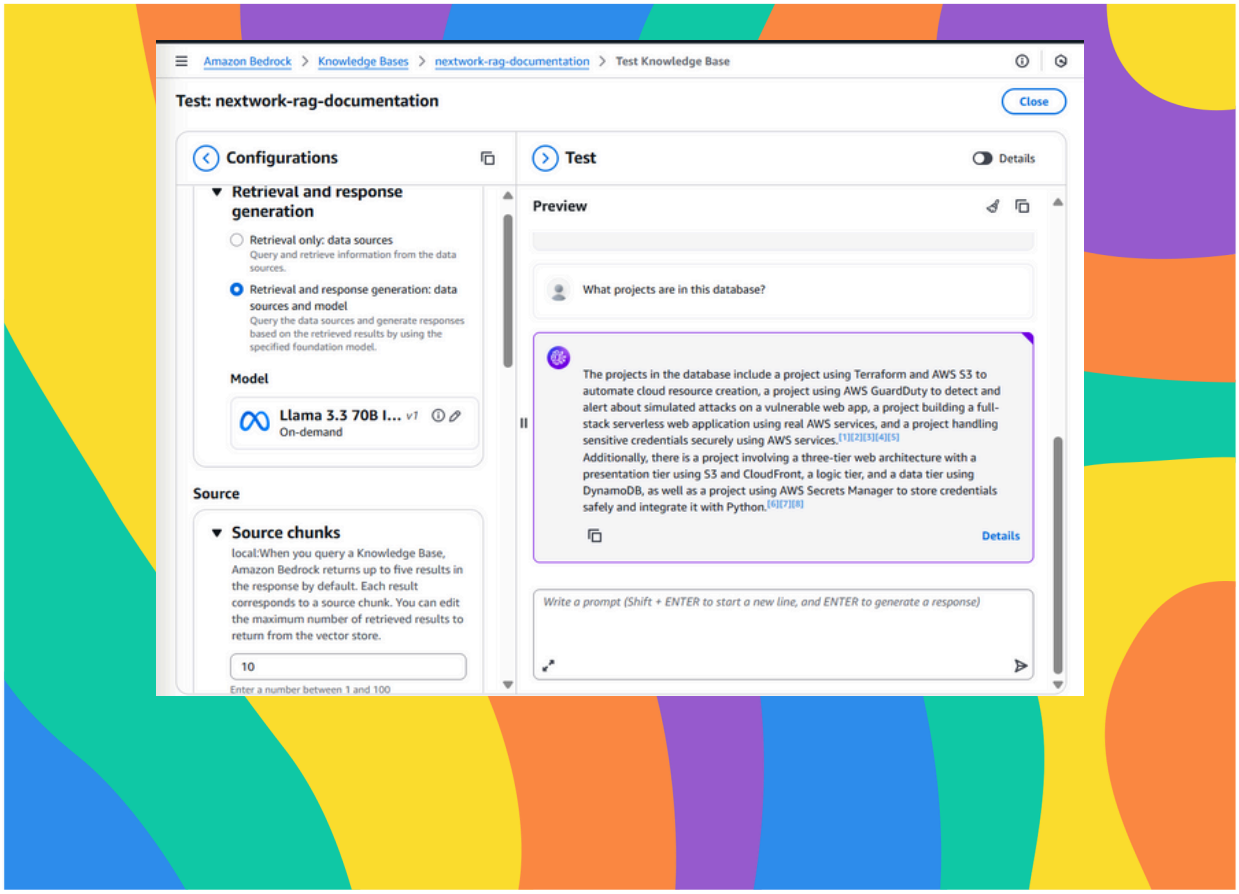
Adding Custom Prompt



Edit prompt template

Custom prompt added

For context, this knowledge base contains AWS, DevOps and AI projects completed by the user. The user will be asking questions about the AWS services they've used, skills they've learnt, and a recap of learnings from these projects. Where possible, always mention skills the user learnt from their projects, even if the prompt did not directly ask for it.



Improved response with edit prompt

After adjusting the **source chunks** and the **generation prompt**, my chatbot's **response** became much **more detailed**. Instead of listing only a few projects or giving a vague answer, it pulled in more projects from the database and highlighted the AWS services and skills I gained from each. Compared to the earlier response, the improved version felt richer, **more accurate**, and better aligned with my project learnings.

Delete the Resources

To avoid unnecessary costs, I deleted all resources created during this project.

1. **Delete Knowledge Base (Amazon Bedrock)**
   - Go to Knowledge Bases in Bedrock console.
   - Select the Knowledge Base.
   - Click Delete.
   - Type delete in the confirmation field.
   - Confirm deletion.

2. **Delete S3 Bucket**
   - Open the S3 console → General purpose buckets.
   - Select bucket: nextwork-bedrock-rag.
   - Click Empty → type permanently delete → confirm.
   - After emptying, select the bucket again → Delete.
   - Type nextwork-bedrock-rag → confirm deletion.

3. **Delete Vector Store (OpenSearch)**
   - Open the OpenSearch console → Collections.
   - Select the vector store.
   - Click Delete.
   - Type confirm in the confirmation field.
   - Confirm deletion.

Kareshma
Rajaananthapadmanaban

Follow up for more interesting projects !!!

Check out nextwork.org / my profile for more projects