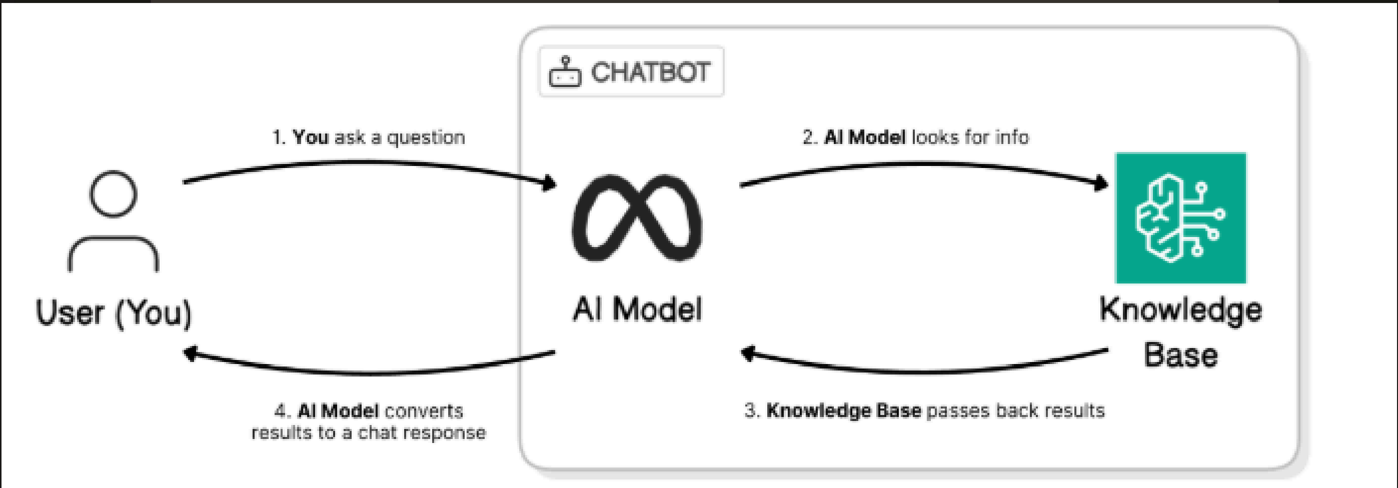
Linkedin

Set Up a RAG Chatbot in Bedrock

DI Dineshraj Dhanapathy


CHATBOT
1. You ask a question
2. AI Model looks for info
User (You)
AI Model
Knowledge Base
4. AI Model converts results to a chat response
3. Knowledge Base passes back results

A simple diagram of what we're building

DI Dineshraj Dhanapathy

Linkedin

Introducing Today's Project!

RAG (Retrieval Augmented Generation) is a method that combines retrieved data with AI to produce informed answers. In this project, I will demonstrate RAG by merging live search results with AI output.

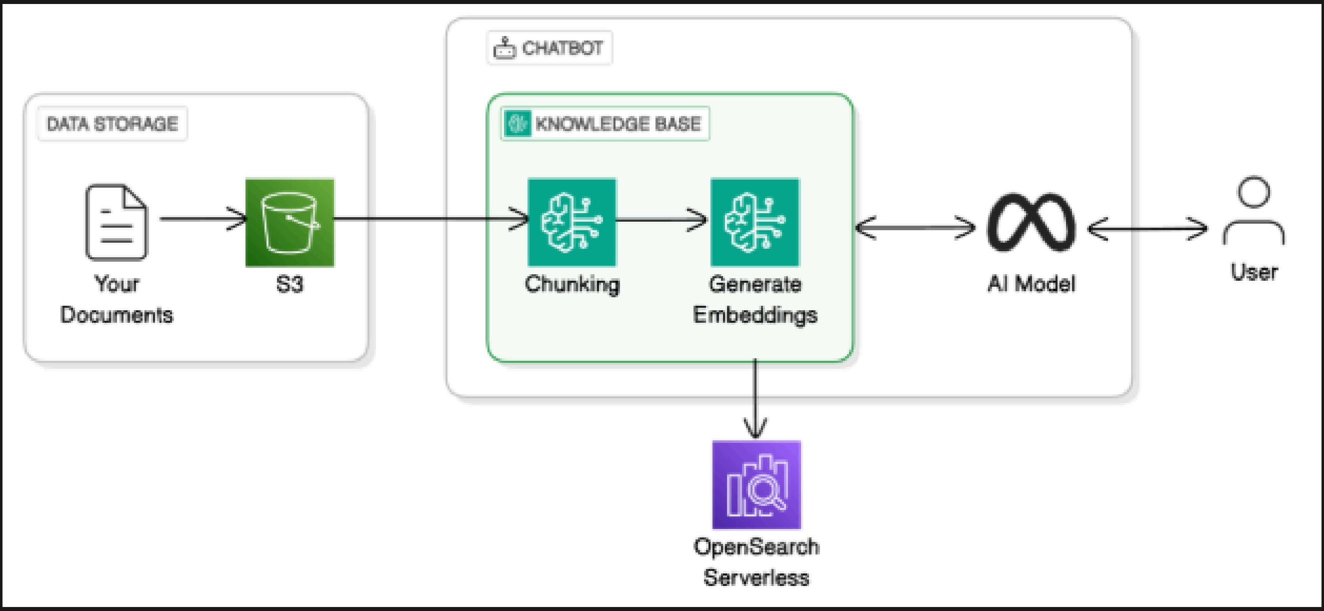
Tools and concepts

Services I used were Amazon Bedrock, S3, and OpenSearch Serverless. Key concepts I learnt include chunking, embeddings, vector stores, and the importance of syncing data for effective AI-powered

Project reflection

This project took me approximately 2 hours. The most challenging part was setting up the Knowledge Base and troubleshooting model responses. It was most rewarding to see the chatbot provide accurate, real-time answers.

I did this project today to deepen my understanding of AI models and enhance my skills in integrating knowledge bases. The project met my goals by successfully building a functional, responsive AI chatbot.



DI

Dineshraj Dhanapathy

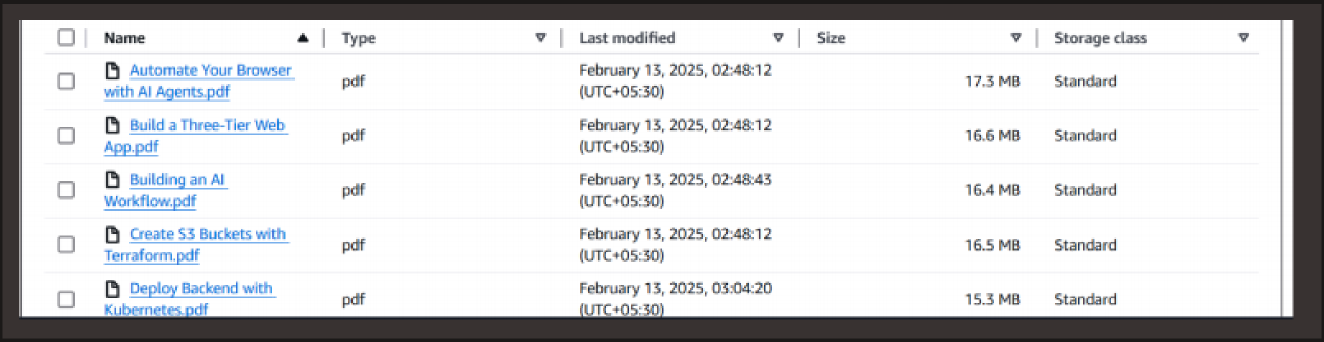
Linkedin

Understanding Amazon Bedrock

Amazon Bedrock is a managed service offering scalable foundation models and tools to build generative AI applications. I'm using Bedrock in this project to integrate and deploy AI-powered solutions.

My Knowledge Base is connected to S3 because S3 is a secure, scalable, and durable storage solution that hosts our data and enables efficient retrieval and processing for AI models for fast access.

In an S3 bucket, I uploaded documents including FAQs, manuals, and guides for our chatbot's Knowledge Base. My S3 bucket is in the same region as my KB because I'm using Ohio East 2 for low latency



Name	Type	Last modified	Size	Storage class
Automate Your Browser with AI Agents.pdf	pdf	February 13, 2025, 02:48:12 (UTC+05:30)	17.3 MB	Standard
Build a Three-Tier Web App.pdf	pdf	February 13, 2025, 02:48:12 (UTC+05:30)	16.6 MB	Standard
Building an AI Workflow.pdf	pdf	February 13, 2025, 02:48:43 (UTC+05:30)	16.4 MB	Standard
Create S3 Buckets with Terraform.pdf	pdf	February 13, 2025, 02:48:12 (UTC+05:30)	16.5 MB	Standard
Deploy Backend with Kubernetes.pdf	pdf	February 13, 2025, 03:04:20 (UTC+05:30)	15.3 MB	Standard

DI Dineshraj Dhanapathy

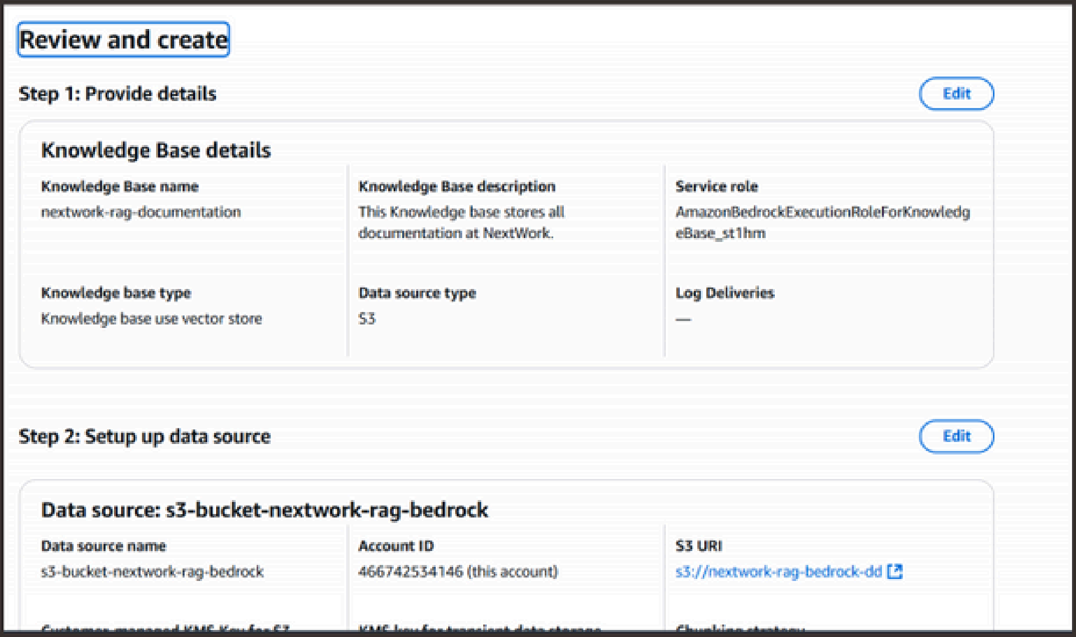
Linkedin

My Knowledge Base Setup

My Knowledge Base uses a vector store, which means it efficiently stores and retrieves embeddings for fast AI responses. When I query my Knowledge Base, OpenSearch will quickly find relevant document chunks

Embeddings are numerical representations of text that help AI understand context and meaning. The embedding model I'm using is Titan Text Embeddings v2 because it provides accurate, efficient semantic search.

Chunking is the process of breaking large text into smaller parts for efficient AI processing. In my Knowledge Base, chunks are set to 300 tokens, ensuring the chatbot retrieves precise and relevant info.



Review and create

Step 1: Provide details

Knowledge Base details

Knowledge Base name	Knowledge Base description	Service role
nextwork-rag-documentation	This Knowledge base stores all documentation at NextWork.	AmazonBedrockExecutionRoleForKnowledgeBase_st1hm
Knowledge base type	Data source type	Log Deliveries
Knowledge base use vector store	S3	—

Step 2: Setup up data source

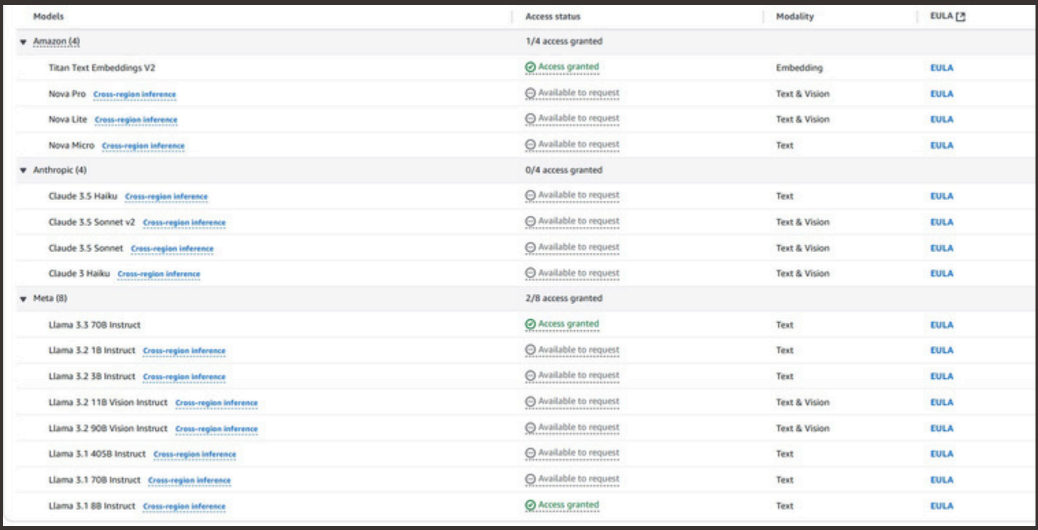
Data source: s3-bucket-nextwork-rag-bedrock

Data source name	Account ID	S3 URI
s3-bucket-nextwork-rag-bedrock	466742534146 (this account)	s3://nextwork-rag-bedrock-dd

AI Models

AI models are important for my chatbot because they generate intelligent, context-aware responses. Without AI models, my chatbot would only retrieve static text without understanding user intent.

To get access to AI models in Bedrock, I had to request model access in the AWS console. AWS needs explicit access because it ensures compliance, security, and proper usage of foundation models.



Models	Access status	Modality	EULA
▼ Amazon (4)	1/4 access granted		
Titan Text Embeddings V2	✓ Access granted	Embedding	EULA
Nova Pro Cross-region inference	⊖ Available to request	Text & Vision	EULA
Nova Lite Cross-region inference	⊖ Available to request	Text & Vision	EULA
Nova Micro Cross-region inference	⊖ Available to request	Text	EULA
▼ Anthropic (4)	0/4 access granted		
Claude 3.5 Haiku Cross-region inference	⊖ Available to request	Text	EULA
Claude 3.5 Sonnet v2 Cross-region inference	⊖ Available to request	Text & Vision	EULA
Claude 3.5 Sonnet Cross-region inference	⊖ Available to request	Text & Vision	EULA
Claude 3 Haiku Cross-region inference	⊖ Available to request	Text & Vision	EULA
▼ Meta (8)	2/8 access granted		
Llama 3.3 70B Instruct	✓ Access granted	Text	EULA
Llama 3.2 1B Instruct Cross-region inference	⊖ Available to request	Text	EULA
Llama 3.2 3B Instruct Cross-region inference	⊖ Available to request	Text	EULA
Llama 3.2 11B Vision Instruct Cross-region inference	⊖ Available to request	Text & Vision	EULA
Llama 3.2 90B Vision Instruct Cross-region inference	⊖ Available to request	Text & Vision	EULA
Llama 3.1 405B Instruct Cross-region inference	⊖ Available to request	Text	EULA
Llama 3.1 70B Instruct Cross-region inference	⊖ Available to request	Text	EULA
Llama 3.1 8B Instruct Cross-region inference	✓ Access granted	Text	EULA

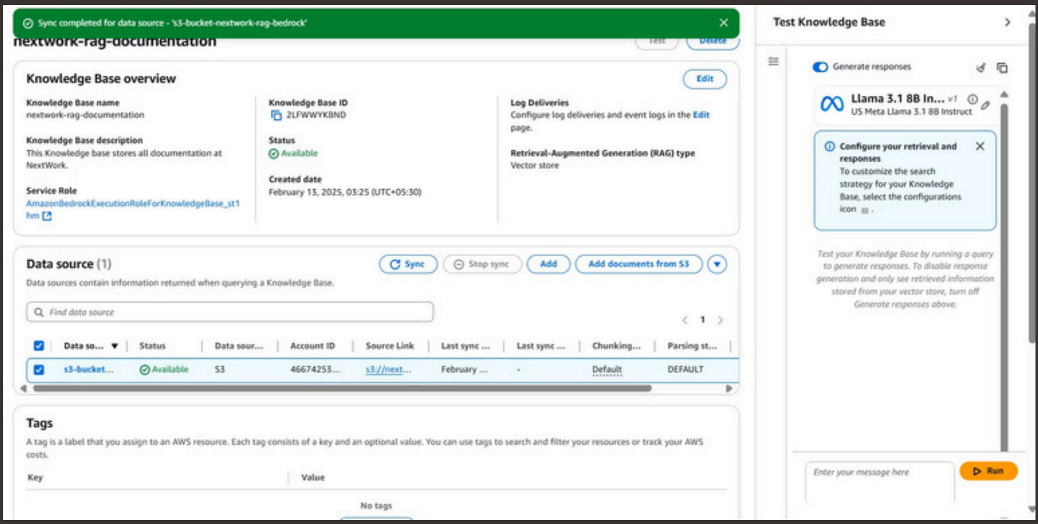
DI Dineshraj Dhanapathy

Linkedin

Syncing the Knowledge Base

Even though I already connected my S3 bucket when creating the Knowledge Base, I still need to sync because it ensures the latest data is indexed, updated, and ready for accurate retrieval by the AI.

The sync process involves three steps: Ingesting, where Bedrock retrieves data from S3; Processing, where it chunks and embeds the data; and Storing, where Bedrock stores the processed data in OpenSearch Serverless.


Sync completed for data source - 's3-bucket-nextwork-rag-bedrock'

nextwork-rag-documentation

Knowledge Base overview

Knowledge Base name
nextwork-rag-documentation

Knowledge Base description
This Knowledge base stores all documentation at NextWork.

Service Role
AmazonBedrockExecutionRoleForKnowledgeBase_st1hm

Knowledge Base ID
2LFWWYKBND

Status
Available

Created date
February 13, 2025, 03:25 (UTC+05:30)

Log Deliveries
Configure log deliveries and event logs in the Edit page.

Retrieval-Augmented Generation (RAG) type
Vector store

Data source (1)

Data sources contain information returned when querying a Knowledge Base.

Data so...	Status	Data sour...	Account ID	Source Link	Last sync ...	Last sync ...	Chunking...	Parsing st...
s3-bucket...	Available	S3	46674253...	s3://next...	February ...	-	Default	DEFAULT

Tags

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key	Value

No tags

Test Knowledge Base

Generate responses

Llama 3.1 8B In... v1
US Meta Llama 3.1 8B Instruct

Configure your retrieval and responses
To customize the search strategy for your Knowledge Base, select the configurations icon.

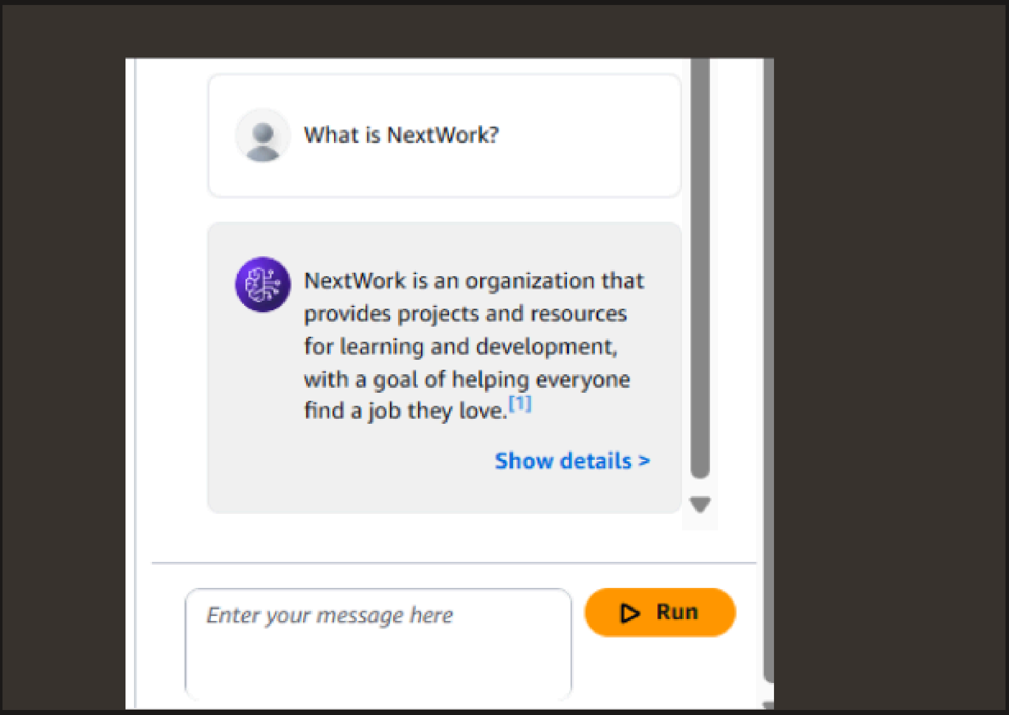
Test your Knowledge Base by running a query to generate responses. To disable response generation and only see retrieved information stored from your vector store, turn off Generate responses above.

Testing My Chatbot

I initially tried to test my chatbot using Llama 3.1 8B as the AI model, but it lacked the depth needed for complex queries. I had to switch to Llama 3.3 70B because it provides more accurate, nuanced responses.

When I asked about topics unrelated to my data, my chatbot struggled to provide relevant answers. This proves that the chatbot relies on the knowledge base, and without related data, its accuracy drops.

You can also turn off the Generate Responses setting to prevent the AI from automatically generating answers. This allows you to manually adjust or review the responses before they are delivered, giving more control over the interaction.

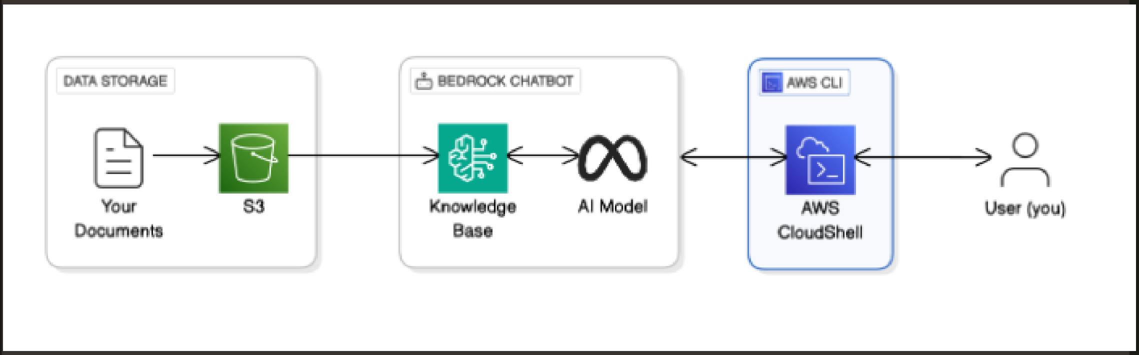

What is NextWork?

NextWork is an organization that provides projects and resources for learning and development, with a goal of helping everyone find a job they love.[1]

Linkedin

Interacting With My RAG Chatbot in the Terminal

DI Dineshraj Dhanapathy



Introducing Today's Project!

In this project, I will build a chatbot using AWS tools. I'm doing this project to learn how to store data in Amazon S3, create a knowledge base using Amazon Bedrock, and use AI models to enable chatbot conversations. I'll also use AWS CloudShell to interact with the chatbot through the terminal. This hands-on project helps me understand the process of deploying and testing AI applications using real-world AWS services.

Tools and concepts

The services I used were Amazon S3, Amazon Bedrock, and AWS CloudShell. The key concepts I learnt included storing and organizing data in S3, creating a Knowledge Base in Bedrock to connect documents to an AI model, and using the AWS CLI in CloudShell to run commands and test chatbot responses. I also understood how synchronization prepares documents for retrieval and how APIs and AI models work together to power intelligent chat interactions.

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was troubleshooting CLI errors and ensuring the correct Knowledge Base ID and Model ARN were used. It was most rewarding to see the chatbot successfully retrieve information from my documents and respond accurately, proving that all services were connected and working together as expected.

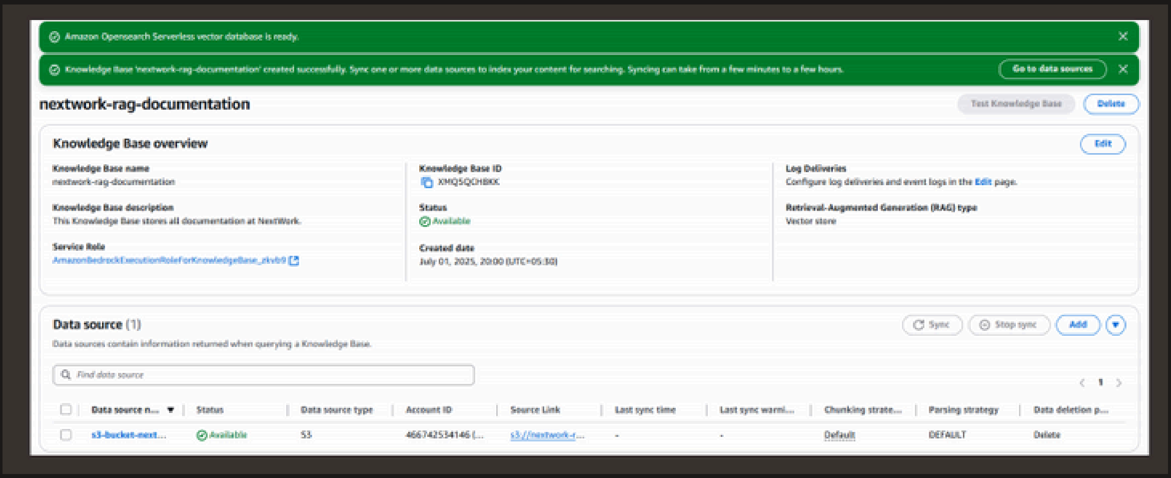
I did this project today to deepen my understanding of how to build and connect AI chatbots using AWS services. I wanted hands-on experience with S3, Bedrock, and the AWS CLI. This project met my goals by helping me learn how to store data, create a Knowledge Base, and interact with AI models in a real-world workflow. It gave me practical skills I can apply to future AI or cloud-based projects and improved my confidence in using AWS tools effectively.

Setting Up The Knowledge Base

To set up my Knowledge Base, I used S3 to store and organize the documents that my chatbot will learn from. The documents I uploaded contain information about a student's portfolio of NextWork project documentation, including their experiences, skills, and achievements. Using S3 allows me to securely store this data in a structured way, making it easy to connect with Amazon Bedrock for further processing. This ensures that the chatbot has accurate and relevant reference material to provide helpful, informed responses.

I also created a Knowledge Base in Bedrock to help my chatbot understand and retrieve information from the documents stored in S3. This allows the chatbot to answer questions using real data instead of just general knowledge. By linking the documents to Bedrock, I make it possible for the AI to search, find, and respond with accurate, context-based answers, making the chatbot more intelligent and useful.

My chatbot also needs access to two AI models, which were selected to generate accurate and helpful responses. I then synchronized the Knowledge Base to ensure it could read and understand the documents stored in S3. This synchronisation processes the content, converts it into a format that the AI models can use, and stores it in the OpenSearch vector database. This step is essential so the chatbot can retrieve relevant information and provide meaningful answers based on the uploaded documents.


Amazon Opensearch Serverless vector database is ready.

Knowledge Base 'nextwork-rag-documentation' created successfully. Sync one or more data sources to index your content for searching. Syncing can take from a few minutes to a few hours.

nextwork-rag-documentation

Knowledge Base overview

Knowledge Base name
nextwork-rag-documentation

Knowledge Base description
This Knowledge Base stores all documentation at NextWork.

Service Role
AmazonBedrockExecutionRoleForKnowledgeBase_zkvb9

Knowledge Base ID
XMQ5QCHBKK

Status
Available

Created date
July 01, 2025, 20:00 (UTC+05:30)

Log Deliveries
Configure log deliveries and event logs in the Edit page.

Retrieval-Augmented Generation (RAG) type
Vector store

Data source (1)

Data sources contain information returned when querying a Knowledge Base.

Find data source

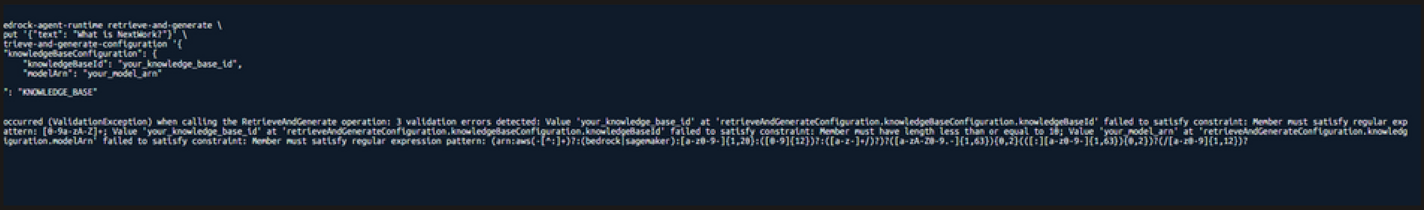
Data source n...	Status	Data source type	Account ID	Source Link	Last sync time	Last sync warni...	Chunking strate...	Parsing strategy	Data deletion p...
s3-bucket-next...	Available	S3	466742534146 (...	s3://nextwork-r...	-	-	Default	DEFAULT	Delete

Running CLI Commands in CloudShell

AWS CLI is a command line tool that lets me create, manage, and interact with AWS services using simple text commands instead of clicking through the AWS Management Console. To start testing CLI commands, I first opened CloudShell, which is a browser-based shell environment built into the AWS Console. CloudShell comes with AWS CLI already installed, so I don't need to set anything up on my computer. It makes it easy and fast to run commands, test my chatbot, and manage AWS resources directly.

When I first ran a Bedrock command, I ran into an error because I needed to provide values for the knowledgeBaseId and the modelArn. The error message showed that placeholders like 'your_knowledge_base_id' and 'your_model_arn' were not replaced with real values. AWS CLI requires these values to match specific patterns and length constraints. To fix this, I looked up the actual Knowledge Base ID and the correct model ARN from the AWS Console and replaced the placeholders in the command. Once corrected, the command ran successfully.

While finding the parameters takes extra time, the advantage of using the CLI is that it allows you to manage AWS resources more quickly and efficiently through simple commands. Itś' especially useful for automation, scripting, and handling repetitive tasks. The CLI also gives you more control and flexibility compared to the AWS Console, making it ideal for experienced users who want to work faster and customize their workflows with precision.

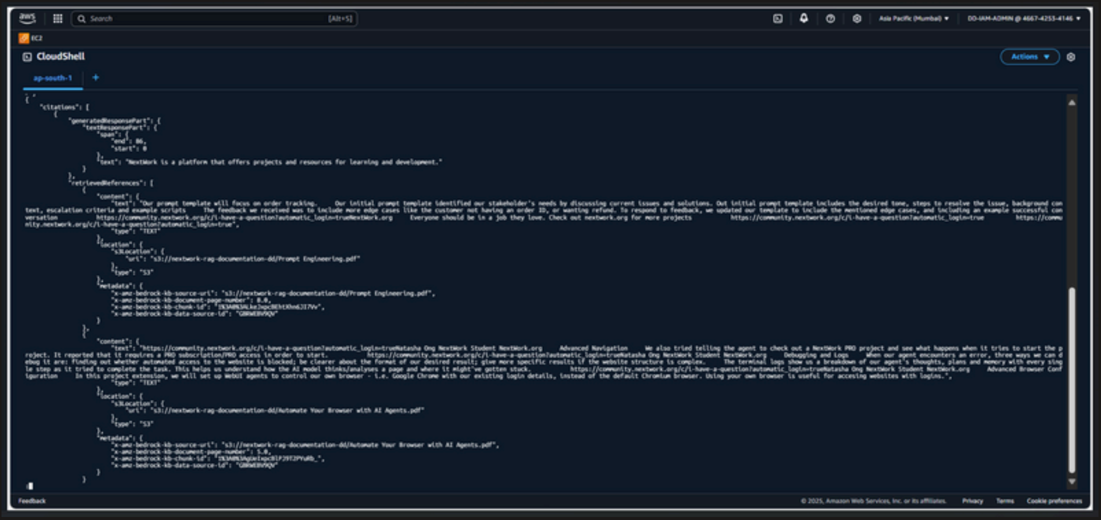


```
edrock-agent-runtime retrieve-and-generate \
put '{"text": "What is NextWork?"}' \
trieve-and-generate-configuration '{
"knowledgeBaseConfiguration": {
    "knowledgeBaseId": "your_knowledge_base_id",
    "modelArn": "your_model_arn"

": "KNOWLEDGE_BASE"


occurred (ValidationException) when calling the RetrieveAndGenerate operation: 3 validation errors detected: Value 'your_knowledge_base_id' at 'retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId' failed to satisfy constraint: Member must satisfy regular exp
attern: [0-9a-zA-Z]+; Value 'your_knowledge_base_id' at 'retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId' failed to satisfy constraint: Member must have length less than or equal to 10; Value 'your_model_arn' at 'retrieveAndGenerateConfiguration.knowledg
iguration.modelArn' failed to satisfy constraint: Member must satisfy regular expression pattern: (arn:aws(-[^:]+)?:(bedrock|sagemaker):[a-z0-9-]{1,20}:([0-9]{12})?:([a-z-]+/)?)?([a-zA-Z0-9.-]{1,63}){0,2}(([:][a-z0-9-]{1,63}){0,2})?(/[a-z0-9]{1,12})?
```

Running Bedrock Commands

To find the required values, I had to go to the Amazon Bedrock console, open my Knowledge Base, and copy the Knowledge Base ID. Then, I selected the AI model I wanted to use (like Llama 3 70B Instruct or Titan Text Embeddings V2) and copied its Model ARN. I updated my CLI command with these values. The Bedrock command ran successfully and showed me a response with relevant information retrieved from my S3 documents, including details from files like Prompt Engineering.pdf and Automate Your Browser with AI Agents.pdf, confirming my chatbot is now working correctly.

The retrieve-and-generate command typically also outputs raw metadata, references, and document source details along with the generated response. To tidy up the terminal response, I added the --query 'output.text' parameter. This filtered the output to show only the chatbot's answer, making it cleaner and easier to read. Instead of seeing all the technical details, the terminal now displayed just the final response, like: “NextWork is a platform that offers projects and resources for learning and development.”



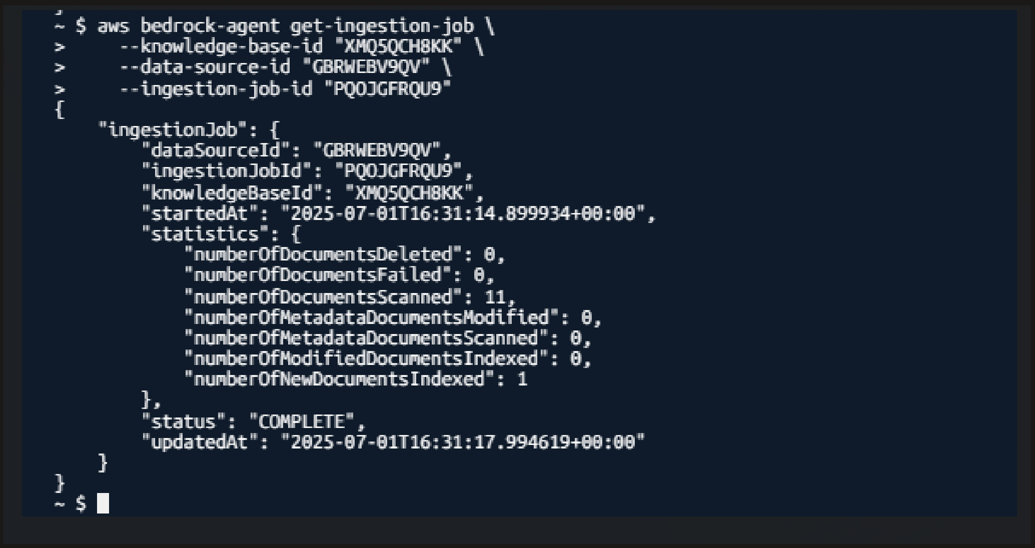
```
{
    "citations": [
        {
            "generatedResponsePart": {
                "textResponsePart": {
                    "span": {
                        "end": 86,
                        "start": 0
                    },
                    "text": "NextWork is a platform that offers projects and resources for learning and development."
                }
            },
            "retrievedReferences": [
                {
                    "content": {
                        "text": "Our prompt template will focus on order tracking.     Our initial prompt template identified our stakeholder's needs by discussing current issues and solutions. Out initial prompt template includes the desired tone, steps to resolve the issue, background con
text, escalation criteria and example scripts     The feedback we received was to include more edge cases like the customer not having an order ID, or wanting refund. To respond to feedback, we updated our template to include the mentioned edge cases, and including an example successful con
versation          https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNextWork.org     Everyone should be in a job they love. Check out nextwork.org for more projects          https://community.nextwork.org/c/i-have-a-question?automatic_login=true        https://commu
nity.nextwork.org/c/i-have-a-question?automatic_login=true",
                        "type": "TEXT"
                    },
                    "location": {
                        "s3Location": {
                            "uri": "s3://nextwork-rag-documentation-dd/Prompt Engineering.pdf"
                        },
                        "type": "S3"
                    },
                    "metadata": {
                        "x-amz-bedrock-kb-source-uri": "s3://nextwork-rag-documentation-dd/Prompt Engineering.pdf",
                        "x-amz-bedrock-kb-document-page-number": 8.0,
                        "x-amz-bedrock-kb-chunk-id": "[illegible]",
                        "x-amz-bedrock-kb-data-source-id": "G8RWEBV9QV"
                    }
                },
                {
                    "content": {
                        "text": "https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Advanced Navigation     We also tried telling the agent to check out a NextWork PRO project and see what happens when it tries to start the p
roject. It reported that it requires a PRO subscription/PRO access in order to start.          https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Debugging and Logs     When our agent encounters an error, three ways we can d
ebug it are: finding out whether automated access to the website is blocked; be clearer about the format of our desired result; give more specific results if the website structure is complex.     The terminal logs show us a breakdown of our agent's thoughts, plans and memory with every sing
le step as it tried to complete the task. This helps us understand how the AI model thinks/analyses a page and where it might've gotten stuck.          https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Advanced Browser Conf
iguration     In this project extension, we will set up WebUI agents to control our own browser - i.e. Google Chrome with our existing login details, instead of the default Chromium browser. Using your own browser is useful for accessing websites with logins.",
                        "type": "TEXT"
                    },
                    "location": {
                        "s3Location": {
                            "uri": "s3://nextwork-rag-documentation-dd/Automate Your Browser with AI Agents.pdf"
                        },
                        "type": "S3"
                    },
                    "metadata": {
                        "x-amz-bedrock-kb-source-uri": "s3://nextwork-rag-documentation-dd/Automate Your Browser with AI Agents.pdf",
                        "x-amz-bedrock-kb-document-page-number": 5.0,
                        "x-amz-bedrock-kb-chunk-id": "[illegible]",
                        "x-amz-bedrock-kb-data-source-id": "G8RWEBV9QV"
                    }
                }
```

Extending Your Knowledge Base

In a project extension, I asked my Knowledge Base about a text—"Why did the coffee go to the police?" I noticed that the response was: “Sorry, I am unable to assist you with this request.” This showed that while the Knowledge Base is working technically, it may not respond to queries outside the context of the uploaded documents. It reminded me that the chatbot is designed to answer based on specific content and may not handle unrelated questions.

To add new information to my Knowledge Base, I ran commands to upload new documents to my S3 bucket and then triggered a sync by running the appropriate CLI commands with my Knowledge Base ID, data source ID, and ingestion job ID. This ensured the new content was processed and indexed for retrieval. Compared to using the console, this process was quicker and more efficient, especially when managing updates frequently, as it allowed me to automate and monitor everything directly from the terminal.

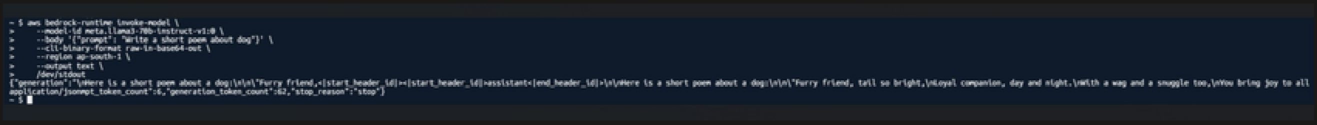
To validate the update worked, I ran a retrieve-and-generate command using AWS CLI with a new query. The Knowledge Base successfully responded with the updated information, showing that it could access and retrieve data from the newly uploaded documents. For example, when I asked, “Why did the coffee go to the police?” it correctly responded with “Because it got mugged!”—confirming that the synchronization included recent content and the chatbot was functioning as expected.



```
~ $ aws bedrock-agent get-ingestion-job \
>     --knowledge-base-id "XMQ5QCH8KK" \
>     --data-source-id "GBRWEBV9QV" \
>     --ingestion-job-id "PQOJGFRQU9"
{
    "ingestionJob": {
        "dataSourceId": "GBRWEBV9QV",
        "ingestionJobId": "PQOJGFRQU9",
        "knowledgeBaseId": "XMQ5QCH8KK",
        "startedAt": "2025-07-01T16:31:14.899934+00:00",
        "statistics": {
            "numberOfDocumentsDeleted": 0,
            "numberOfDocumentsFailed": 0,
            "numberOfDocumentsScanned": 11,
            "numberOfMetadataDocumentsModified": 0,
            "numberOfMetadataDocumentsScanned": 0,
            "numberOfModifiedDocumentsIndexed": 0,
            "numberOfNewDocumentsIndexed": 1
        },
        "status": "COMPLETE",
        "updatedAt": "2025-07-01T16:31:17.994619+00:00"
    }
}
~ $
```

Interacting with AI Models Directly

On top of chatting with my chatbot, I also interacted directly with an AI model via the terminal by running a retrieve-and-generate command using the AWS CLI. This allowed me to send a prompt directly to the model and receive a generated response, like a short poem about a dog. It demonstrated that I could bypass the Knowledge Base if needed and engage with the model for creative or general-purpose tasks. This direct interaction was fast, flexible, and showed the modelš’ standalone capabilities.

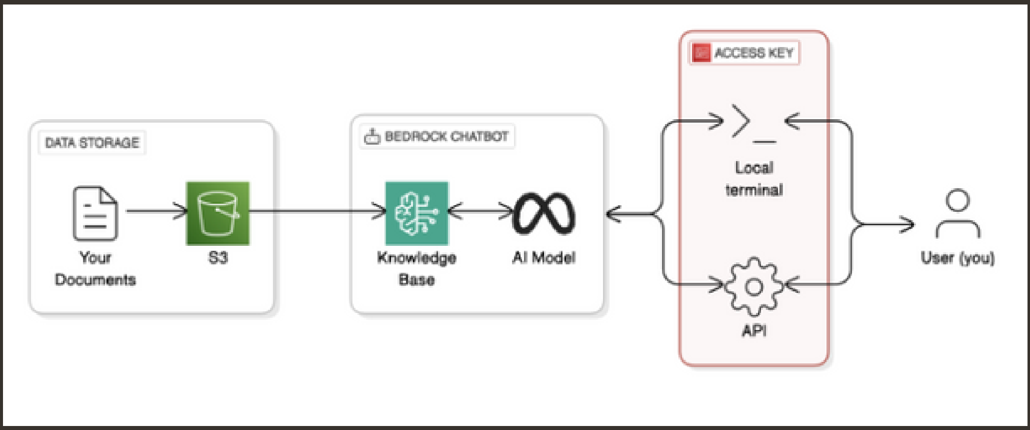


```
~ $ aws bedrock-runtime invoke-model \
>     --model-id meta.llama3-70b-instruct-v1:0 \
>     --body '{"prompt": "Write a short poem about dog"}' \
>     --cli-binary-format raw-in-base64-out \
>     --region ap-south-1 \
>     --output text \
>     /dev/stdout
{"generation":"\nHere is a short poem about a dog:\n\n\"Furry friend,<|start_header_id|>=<|start_header_id|>assistant<|end_header_id|>\n\nHere is a short poem about a dog:\n\n\"Furry friend, tall so bright,\nLoyal companion, day and night.\nWith a wag and a snuggle too,\nYou bring joy to all
application/jsonmpt_token_count":6,"generation_token_count":62,"stop_reason":"stop"}
~ $
```

Linkedin

How I Built an API for My RAG Chatbot

DI Dineshraj Dhanapathy



Introducing Today's Project!

In this project, I will demonstrate how a chatbot can interact with users through a website using an API. I'm doing this project to learn how APIs connect front-end interfaces with backend AI services hosted in AWS. By understanding how a user's question travels from the website to the chatbot and back, I'll gain hands-on experience in API communication, chatbot response handling, and real-time user interaction—all essential skills for building intelligent web applications.

Tools and concepts

Services I used were Amazon Bedrock, and FastAPI. Key concepts I learnt include Retrieval-Augmented Generation (RAG), how to use retrieve_and_generate for knowledge base queries, and invoke_model for direct model access. I also learned how to manage environment variables securely with .env, configure Boto3 clients, and handle errors and logging effectively in an AI-powered API setup.

Project reflection

This project took me approximately 2 hours to complete. The most challenging part was debugging the missing MODEL_ID and ensuring all environment variables were correctly set for both knowledge base and direct model access. It was most rewarding to see both endpoints working successfully—one powered by specific documents and the other by general AI knowledge—demonstrating the flexibility and power of building intelligent APIs with AWS Bedrock.

I did this project today to deepen my understanding of Amazon Bedrock and how to build intelligent, flexible chatbot APIs using both knowledge base retrieval and direct model access. This project met my goals by helping me learn key AWS services, improve my API development skills, and successfully implement a dual-endpoint chatbot. It gave me hands-on experience with real-world AI integration, which aligns with my learning and career goals in cloud and AI.

Setting Up The Knowledge Base

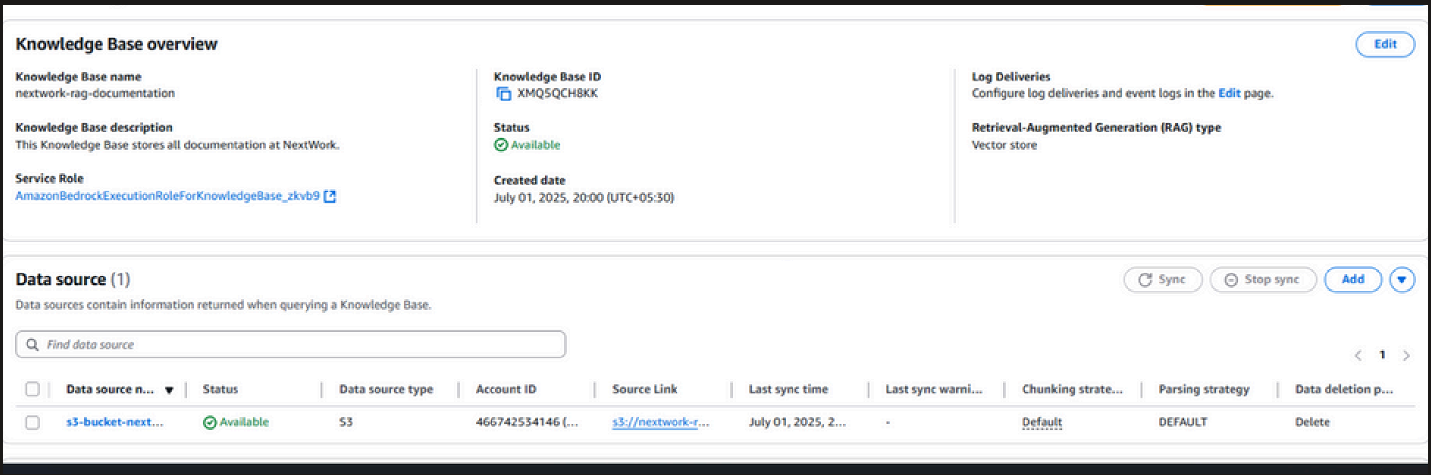
To set up my Knowledge Base, I used S3 to securely store and organize the documents my chatbot will learn from. The documents I uploaded contain information about a student's NextWork project experience, including skills developed and feedback received. S3 makes it easy to manage and access this data, ensuring that the Knowledge Base can retrieve and process the content accurately for generating relevant responses through the chatbot.

Amazon Bedrock is a fully managed service that allows you to build and scale generative AI applications using foundation models. I created a Knowledge Base in Bedrock to connect my uploaded documents in S3 with an AI model, enabling the chatbot to retrieve and understand specific information. This setup allows the chatbot to provide accurate, context-aware answers based on real content, turning static documents into interactive, searchable knowledge that powers intelligent conversations.

My chatbot also needs access to two AI models to understand user questions and generate accurate responses based on the documents I uploaded. I then synchronized the Knowledge Base with the data in S3 because this process allows the AI models to read, process, and convert the documents into a searchable format. Without synchronization, the AI wouldn’t be able to access or understand the content, making it impossible for the chatbot to retrieve the right information and provide meaningful answers.

Dineshraj Dhanapathy

Linkedin


Knowledge Base overview

Edit

Knowledge Base name
nextwork-rag-documentation

Knowledge Base description
This Knowledge Base stores all documentation at NextWork.

Service Role
AmazonBedrockExecutionRoleForKnowledgeBase_zkvb9

Knowledge Base ID
XMQ5QCH8KK

Status
Available

Created date
July 01, 2025, 20:00 (UTC+05:30)

Log Deliveries
Configure log deliveries and event logs in the Edit page.

Retrieval-Augmented Generation (RAG) type
Vector store

Data source (1)

Data sources contain information returned when querying a Knowledge Base.

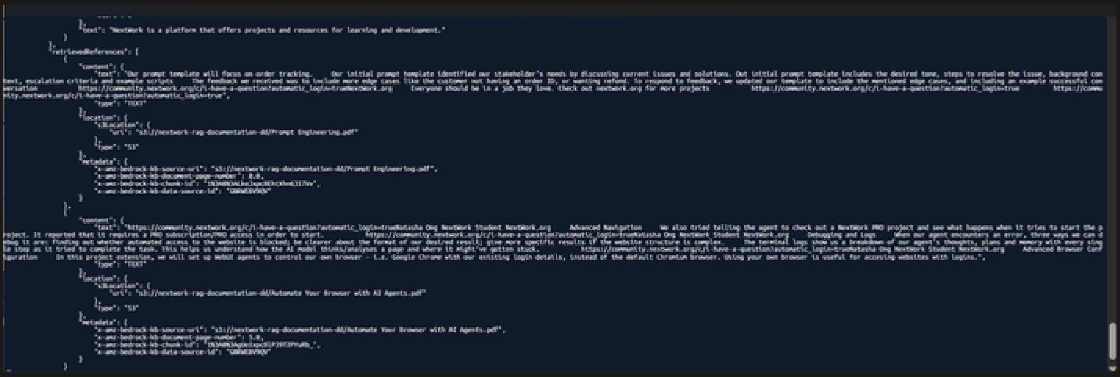
Sync | Stop sync | Add

Find data source

Data source n...	Status	Data source type	Account ID	Source Link	Last sync time	Last sync warni...	Chunking strate...	Parsing strategy	Data deletion p...
s3-bucket-next...	Available	S3	466742534146 (...	s3://nextwork-r...	July 01, 2025, 2...	-	Default	DEFAULT	Delete

Running CLI Commands Locally

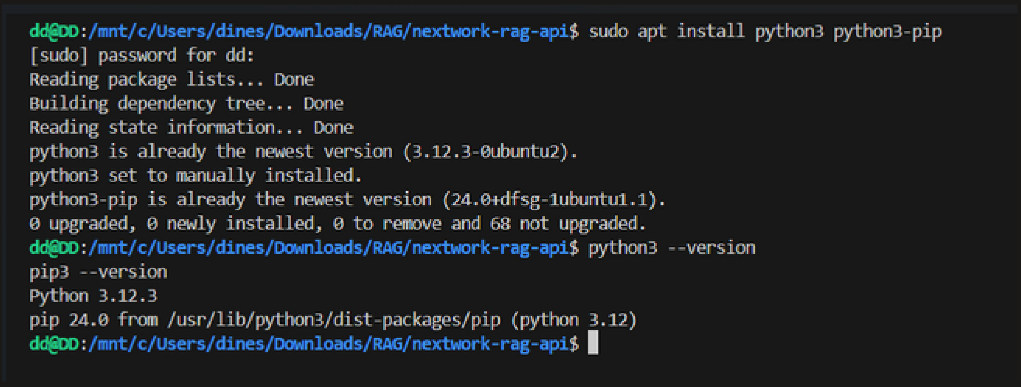
When I ran a Bedrock command in my terminal, I initially got an error because the AWS CLI wasn’t configured with my credentials. To resolve this, I installed the AWS CLI, then ran aws configure to enter my AWS Access Key ID, Secret Access Key, region, and output format. This setup allowed my terminal to securely connect with my AWS account and run Bedrock commands successfully, enabling me to interact with my Knowledge Base from my local environment.


```
            },
            "text": "NextWork is a platform that offers projects and resources for learning and development."
        }
    ],
    "retrievedReferences": [
        {
            "content": {
                "text": "Our prompt template will focus on order tracking.     Our initial prompt template identified our stakeholder's needs by discussing current issues and solutions. Out initial prompt template includes the desired tone, steps to resolve the issue, background con
text, escalation criteria and example scripts     The feedback we received was to include more edge cases like the customer not having an order ID, or wanting refund. to respond to feedback, we updated our template to include the mentioned edge cases, and including an example successful con
versation          https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNextWork.org     Everyone should be in a job they love. Check out nextwork.org for more projects          https://community.nextwork.org/c/i-have-a-question?automatic_login=true          https://commu
ity.nextwork.org/c/i-have-a-question?automatic_login=true",
                "type": "TEXT"
            },
            "location": {
                "s3Location": {
                    "uri": "s3://nextwork-rag-documentation-dd/Prompt Engineering.pdf"
                },
                "type": "S3"
            },
            "metadata": {
                "x-amz-bedrock-kb-source-uri": "s3://nextwork-rag-documentation-dd/Prompt Engineering.pdf",
                "x-amz-bedrock-kb-document-page-number": 0.0,
                "x-amz-bedrock-kb-chunk-id": "1%3A0%3AlkeJxpc8EhtXhn6JI7Vv",
                "x-amz-bedrock-kb-data-source-id": "Q8RWEBVRQV"
            }
        },
        {
            "content": {
                "text": "https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Advanced Navigation     We also tried telling the agent to check out a NextWork PRO project and see what happens when it tries to start the p
roject. It reported that it requires a PRO subscription/PRO access in order to start.        https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Debugging and Logs     When our agent encounters an error, three ways we can d
ebug it are: finding out whether automated access to the website is blocked; be clearer about the format of our desired result; give more specific results if the website structure is complex.     The terminal logs show us a breakdown of our agent's thoughts, plans and memory with every sing
le step as it tried to complete the task. This helps us understand how the AI model thinks/analyses a page and where it might've gotten stuck.        https://community.nextwork.org/c/i-have-a-question?automatic_login=trueNatasha Ong NextWork Student NextWork.org     Advanced Browser Conf
iguration     In this project extension, we will set up WebUI agents to control our own browser - i.e. Google Chrome with our existing login details, instead of the default Chromium browser. Using your own browser is useful for accessing websites with logins.",
                "type": "TEXT"
            },
            "location": {
                "s3Location": {
                    "uri": "s3://nextwork-rag-documentation-dd/Automate Your Browser with AI Agents.pdf"
                },
                "type": "S3"
            },
            "metadata": {
                "x-amz-bedrock-kb-source-uri": "s3://nextwork-rag-documentation-dd/Automate Your Browser with AI Agents.pdf",
                "x-amz-bedrock-kb-document-page-number": 1.0,
                "x-amz-bedrock-kb-chunk-id": "1%3A0%3AgUelxpcBlP2YE2PYnRb_",
                "x-amz-bedrock-kb-data-source-id": "Q8RWEBVRQV"
            }
        }
```

Running Bedrock Commands

When I first ran a Bedrock command, I ran into an error because I needed to provide values for the knowledgeBaseId and modelArn. I had left the placeholders 'your_knowledge_base_id' and 'your_model_arn' in the command, which caused a validation error. Bedrock requires valid, correctly formatted values for these fields to connect to the right resources. Once I retrieved the actual values from the AWS Console and updated the command, it ran successfully and returned a valid response from the Knowledge Base.

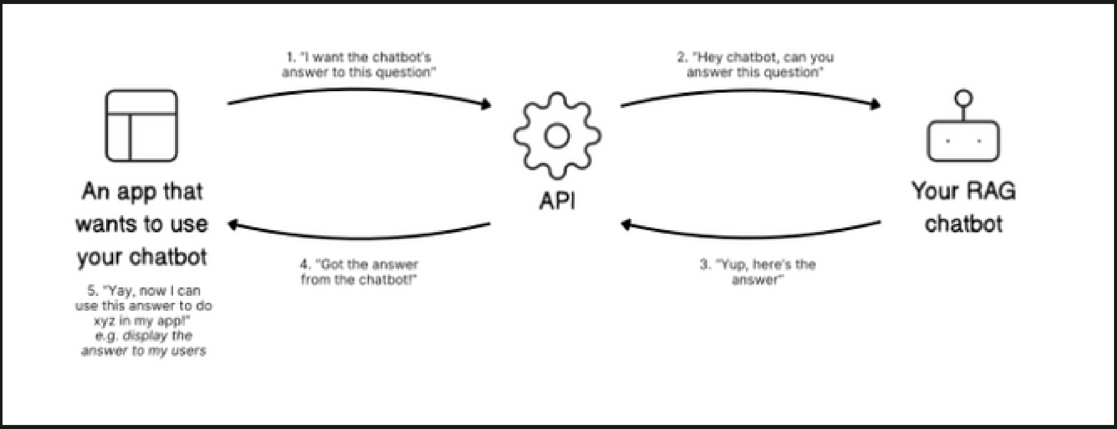
To find the required values, I had to go to the Bedrock console and copy my Knowledge Base ID and Model ARN. The Bedrock command ran successfully and showed me a clear response: “NextWork is a platform that offers projects and resources for learning and development.” This confirmed that the chatbot could access and retrieve information from the synchronized documents, proving that the setup was working correctly.



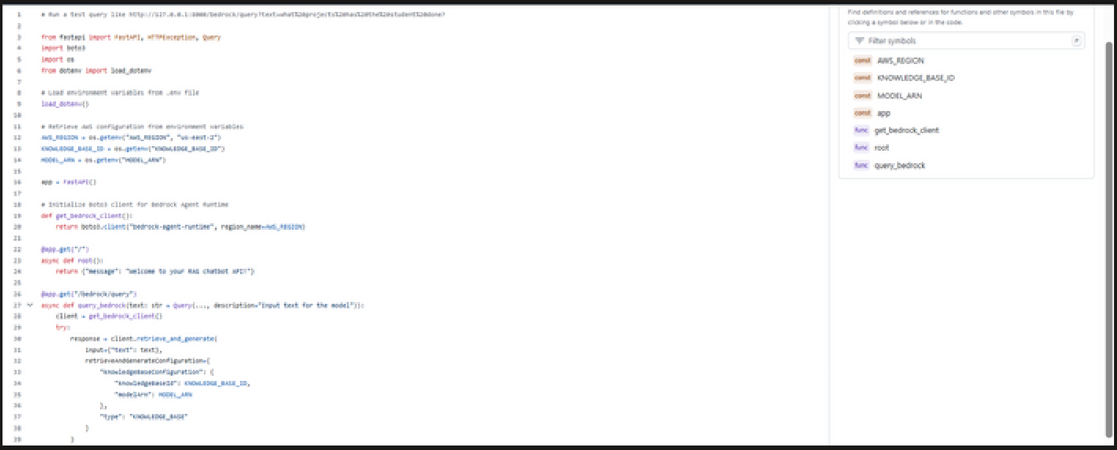
```
dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-api$ sudo apt install python3 python3-pip
[sudo] password for dd:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.12.3-0ubuntu2).
python3 set to manually installed.
python3-pip is already the newest version (24.0+dfsg-1ubuntu1.1).
0 upgraded, 0 newly installed, 0 to remove and 68 not upgraded.
dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-api$ python3 --version
pip3 --version
Python 3.12.3
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-api$
```

Creating an API

An API is an interface that allows different software systems to communicate with each other. The API I'm creating will act as a bridge between a web application and my chatbot hosted in Amazon Bedrock. It will receive user questions, send them to the chatbot, and return the chatbot's responses back to the web app. This will be helpful for integrating the chatbot into websites or applications, enabling real-time interaction with users through simple web requests.


1. "I want the chatbot's answer to this question"
2. "Hey chatbot, can you answer this question"
An app that wants to use your chatbot
API
Your RAG chatbot
4. "Got the answer from the chatbot!"
3. "Yup, here's the answer"
5. "Yay, now I can use this answer to do xyz in my app!" e.g. display the answer to my users

The API code has three main sections: configuration, route handling, and Bedrock interaction. First, it retrieves AWS settings like AWS_REGION, KNOWLEDGE_BASE_ID, and MODEL_ARN from environment variables to securely connect to the correct AWS resources. Next, it defines routes (like a /chat endpoint) to handle incoming requests. Finally, it sends user input to Amazon Bedrock and returns the chatbotś' response, allowing smooth communication between the user and the AI.



```
# Run a test query like http://127.0.0.1:8000/bedrock/query?text=what%20projects%20has%20the%20student%20done?

from fastapi import FastAPI, HTTPException, Query
import boto3
import os
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Retrieve AWS configuration from environment variables
AWS_REGION = os.getenv("AWS_REGION", "us-east-2")
KNOWLEDGE_BASE_ID = os.getenv("KNOWLEDGE_BASE_ID")
MODEL_ARN = os.getenv("MODEL_ARN")

app = FastAPI()

# Initialize boto3 client for Bedrock Agent Runtime
def get_bedrock_client():
    return boto3.client("bedrock-agent-runtime", region_name=AWS_REGION)

@app.get("/")
async def root():
    return {"message": "Welcome to your RAG chatbot API!"}

@app.get("/bedrock/query")
async def query_bedrock(text: str = Query(..., description="Input text for the model")):
    client = get_bedrock_client()
    try:
        response = client.retrieve_and_generate(
            input={"text": text},
            retrieveAndGenerateConfiguration={
                "knowledgeBaseConfiguration": {
                    "knowledgeBaseId": KNOWLEDGE_BASE_ID,
                    "modelArn": MODEL_ARN
                },
                "type": "KNOWLEDGE_BASE"
            }
        )
```

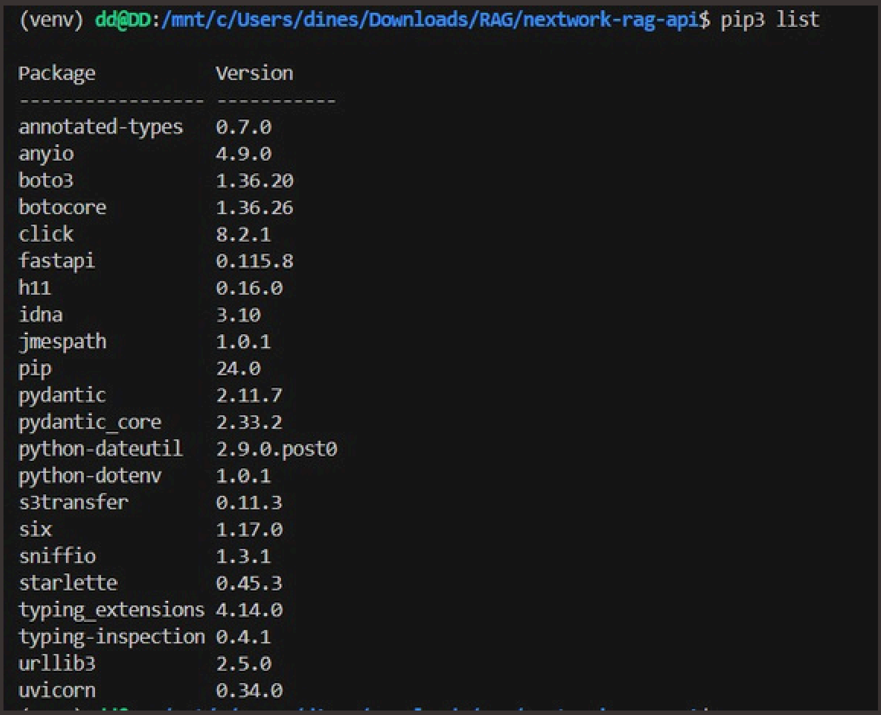
Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

const	AWS_REGION
const	KNOWLEDGE_BASE_ID
const	MODEL_ARN
const	app
func	get_bedrock_client
func	root
func	query_bedrock

Installing Packages

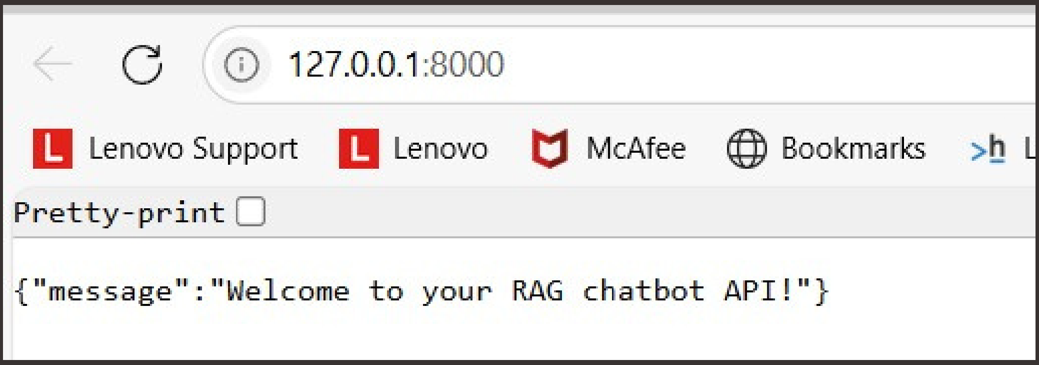
To run the API, I had to install requirements like fastapi, uvicorn, boto3, and python- dotenv. These packages are important because fastapi is used to build the web API, uvicorn runs the FastAPI app as an ASGI server, and boto3 allows the API to interact with AWS services like Amazon Bedrock. python-dotenv helps load environment variables securely, such as MODEL_ARN and KNOWLEDGE_BASE_ID. Supporting libraries like pydantic, anyio, and starlette ensure fast and reliable request handling, validation, and asynchronous execution.



```
(venv) dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-api$ pip3 list

Package           Version
----------------- -----------
annotated-types   0.7.0
anyio             4.9.0
boto3             1.36.20
botocore          1.36.26
click             8.2.1
fastapi           0.115.8
h11               0.16.0
idna              3.10
jmespath          1.0.1
pip               24.0
pydantic          2.11.7
pydantic_core     2.33.2
python-dateutil   2.9.0.post0
python-dotenv     1.0.1
s3transfer        0.11.3
six               1.17.0
sniffio           1.3.1
starlette         0.45.3
typing_extensions 4.14.0
typing-inspection 0.4.1
urllib3           2.5.0
uvicorn           0.34.0
```

Testing the API

When I visited the root endpoint, I saw the message: "message": "Welcome to your RAG chatbot API!" This confirms that the FastAPI server is running correctly and the API is live. It also means that the routing is set up properly, and the application is ready to handle requests, such as sending user input to the chatbot and returning responses from Amazon Bedrock.

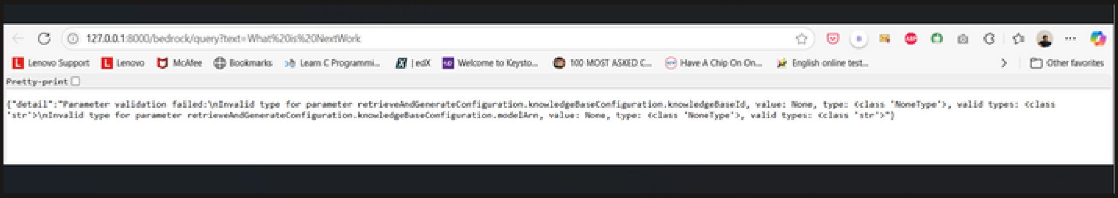


```
{"message":"Welcome to your RAG chatbot API!"}
```

The Query Endpoint

The query endpoint connects an app with the RAG chatbot to retrieve and generate answers from a knowledge base. I tested the query endpoint by sending a request with sample input data to evaluate if the chatbot could respond correctly. The response was: {"detail":"Parameter validation failed:\nInvalid type for parameter retr ieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId, value: None...\n} indicating missing or null values that should be valid strings.

Looking at the API code, I can see that the API calls for environment variables because it relies on values like knowledgeBaseId and modelArn to be passed correctly for processing requests. To resolve the error in my query endpoint, I need to ensure these environment variables are defined and not None, and that they are correctly passed as strings in the request payload to meet the expected parameter types.



```
{"detail":"Parameter validation failed:\nInvalid type for parameter retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId, value: None, type: <class 'NoneType'>, valid types: <class
'str'>\nInvalid type for parameter retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.modelArn, value: None, type: <class 'NoneType'>, valid types: <class 'str'>"}
```

DI Dineshraj Dhanapathy

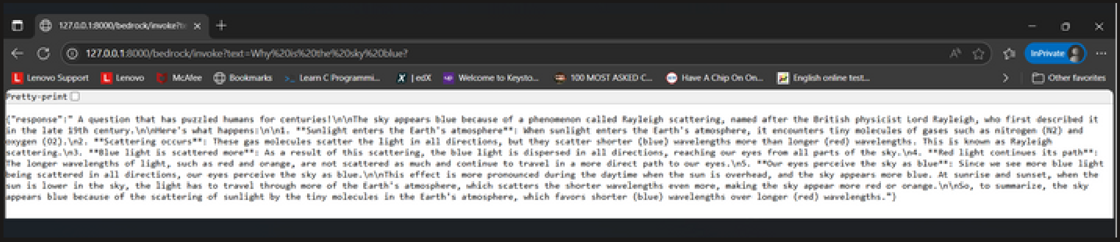
Linkedin

Extending the API

In a project extension, I decided to extend my API by enabling both knowledge base retrieval and direct model invocation. Changes to the API code include adding a new /bedrock/invoke endpoint that talks directly to the AI model using invoke_model, bypassing the knowledge base. I added structured logging, error handling, and validation for missing environment variables. This version also separates AWS clients for bedrock-runtime and bedrock-agent-runtime, showcasing advanced control, flexibility, and AI integration skills.

I initially ran into an error with the new endpoint because I didn't provide the MODEL_ID in the .env file, which is required for the /bedrock/invoke endpoint to communicate with the AI model. Once I fixed it by adding MODEL_ID=<your-model-id> to the .env file, I validated the new endpoint by running a test query, and it successfully returned a response from the model. This confirmed that the direct invocation setup was working correctly with the configured environment variables.

When I use /bedrock/query, the chatbot searches the Knowledge Base first and gives answers based on specific, curated documents—great for domain-specific accuracy. But when I use /bedrock/invoke, the chatbot relies on the AI models' general knowledge to answer, which is broader and more flexible but may lack context from my custom data. This dual approach lets my chatbot respond with either precision from my content or versatility from the models' training.

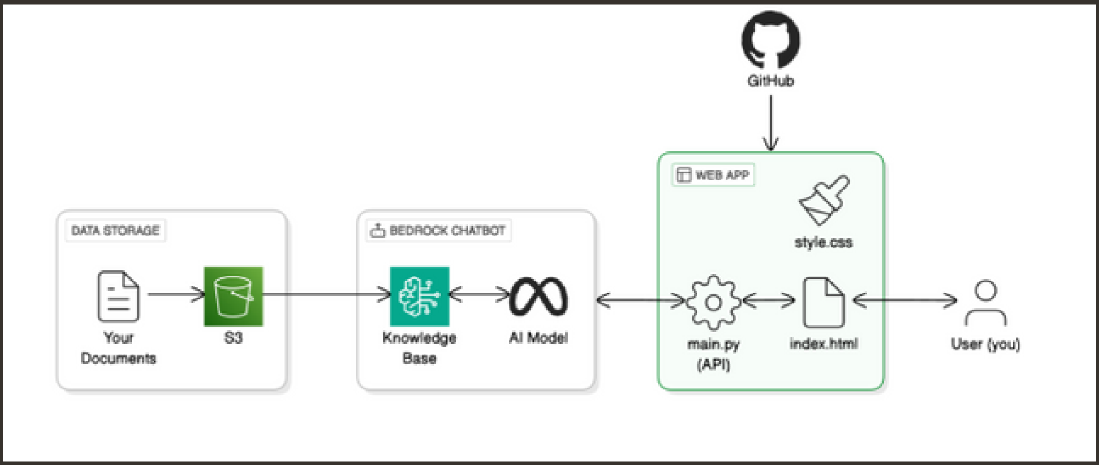


```
{"response":" A question that has puzzled humans for centuries!\n\nThe sky appears blue because of a phenomenon called Rayleigh scattering, named after the British physicist Lord Rayleigh, who first described it
in the late 19th century.\n\nHere's what happens:\n\n1. **Sunlight enters the Earth's atmosphere**: When sunlight enters the Earth's atmosphere, it encounters tiny molecules of gases such as nitrogen (N2) and
oxygen (O2).\n2. **Scattering occurs**: These gas molecules scatter the light in all directions, but they scatter shorter (blue) wavelengths more than longer (red) wavelengths. This is known as Rayleigh
scattering.\n3. **Blue light is scattered more**: As a result of this scattering, the blue light is dispersed in all directions, reaching our eyes from all parts of the sky.\n4. **Red light continues its path**:
The longer wavelengths of light, such as red and orange, are not scattered as much and continue to travel in a more direct path to our eyes.\n5. **Our eyes perceive the sky as blue**: Since we see more blue light
being scattered in all directions, our eyes perceive the sky as blue.\n\nThis effect is more pronounced during the daytime when the sun is overhead, and the sky appears more blue. At sunrise and sunset, when the
sun is lower in the sky, the light has to travel through more of the Earth's atmosphere, which scatters the shorter wavelengths even more, making the sky appear more red or orange.\n\nSo, to summarize, the sky
appears blue because of the scattering of sunlight by the tiny molecules in the Earth's atmosphere, which favors shorter (blue) wavelengths over longer (red) wavelengths."}
```

Linkedin

Building a RAG Chatbot with a Web Interface

DI Dineshraj Dhanapathy



Introducing Today's Project!

In this project, I will demonstrate how to build a full-stack AI chatbot application using Amazon Bedrock and S3 for knowledge storage, a pre-built web app for the frontend and backend, and FastAPI for API customization. I'm doing this project to learn how to integrate a Knowledge Base with a chatbot, connect it to a web app, and test both backend and frontend interactions. This hands-on setup will help me gain practical skills in deploying intelligent apps using AWS tools.

Tools and concepts

Services I used were Amazon Bedrock, S3, and FastAPI. Key concepts I learnt include how to create and use a Knowledge Base for Retrieval-Augmented Generation (RAG), how to connect frontend and backend components using API routes, and how to securely manage configuration with environment variables. I also learned how to build and customize a complete chat interface, integrate it with AWS services, and handle real-time interactions between a user and an AI model.

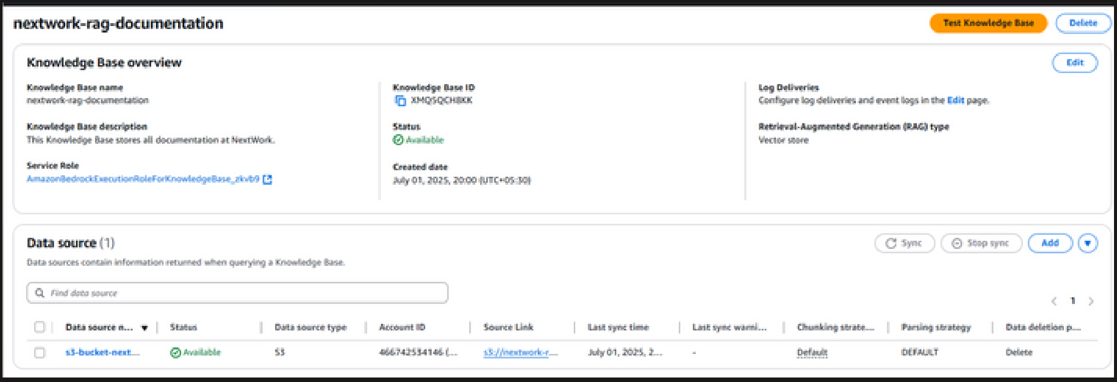
Project reflection

This project took me approximately 2 hours to complete. The most challenging part was debugging the API integration and ensuring the environment variables were correctly set for both Knowledge Base and direct model access. It was most rewarding to see the customized frontend working seamlessly with the backend, successfully switching between RAG and direct AI responses in real-time.

I did this project today to deepen my practical skills in building AI-powered web apps using Amazon Bedrock and FastAPI. It helped me connect backend APIs with a custom frontend and understand how to work with both general AI models and knowledge-based responses. This project met my goals by giving me hands-on experience in API integration, frontend customization, and deploying intelligent chat interfaces.

Chatbot setup

I created a Knowledge Base to link my uploaded documents in S3 with Amazon Bedrock, enabling the chatbot to retrieve relevant information from them. This is important because it allows the AI to generate accurate, context-specific responses based on my own content, rather than relying only on general knowledge. It enhances the chatbotś' usefulness for personalized or domain-specific queries.



nextwork-rag-documentation

Knowledge Base overview

Knowledge Base name
nextwork-rag-documentation

Knowledge Base description
This Knowledge Base stores all documentation at NextWork.

Service Role
AmazonBedrockExecutionRoleForKnowledgeBase_zkvb9

Knowledge Base ID
XMQSQCH8KK

Status
Available

Created date
July 01, 2025, 20:00 (UTC+05:30)

Log Deliveries
Configure log deliveries and event logs in the Edit page.

Retrieval-Augmented Generation (RAG) type
Vector store

Data source (1)

Data sources contain information returned when querying a Knowledge Base.

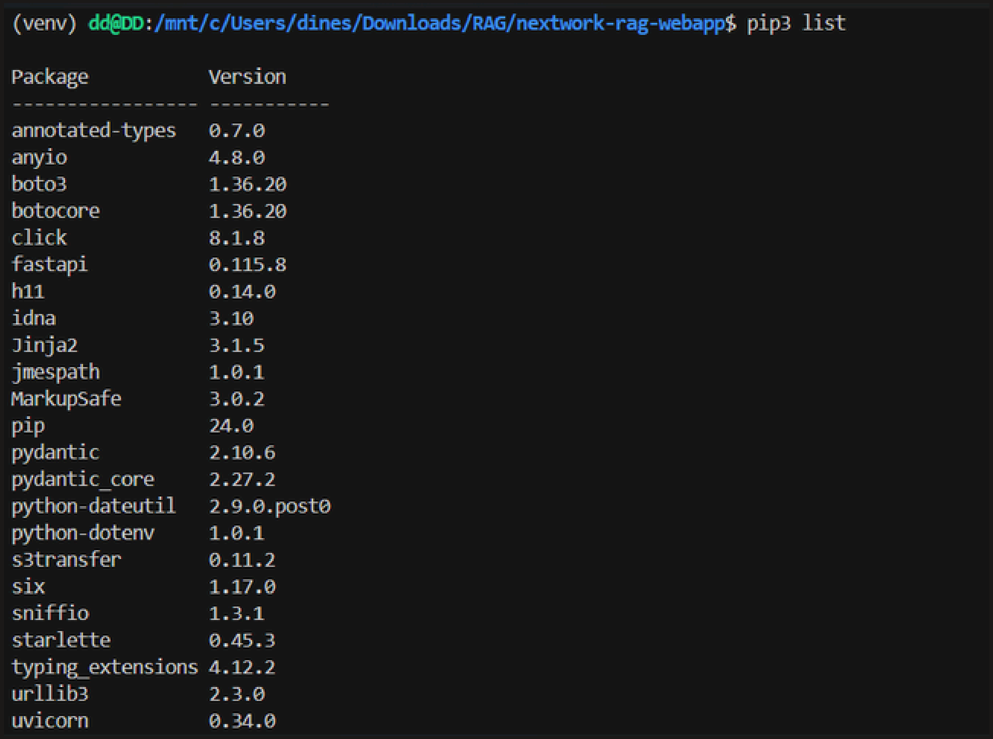
Find data source

Data source n...	Status	Data source type	Account ID	Source Link	Last sync time	Last sync warni...	Chunking strate...	Parsing strategy	Data deletion p...
s3-bucket-next...	Available	S3	466742534146 (...	s3://nextwork-r...	July 01, 2025, 2...	-	Default	DEFAULT	Delete

Setting Up the API

To run the API, I had to install requirements like fastapi, uvicorn, boto3, and python-dotenv. These packages are important because fastapi helps build the API quickly, uvicorn runs the server, boto3 lets the app connect to AWS services like Bedrock and S3, and python-dotenv loads environment variables securely. Other packages like pydantic and starlette support data validation and request handling, making the app stable and production-ready.

A virtual environment helps me by creating an isolated space for this projectś' dependencies, so the Python packages I install won't affect other projects on my computer. This is especially useful when working with specific versions of libraries or frameworks. We need it for this project to ensure that the chatbot app runs consistently, without conflicts from global packages or other environments, making development and troubleshooting much easier.



```
(venv) dd@DD:/mnt/c/Users/dines/Downloads/RAG/nextwork-rag-webapp$ pip3 list

Package           Version
----------------- -----------
annotated-types   0.7.0
anyio             4.8.0
boto3             1.36.20
botocore          1.36.20
click             8.1.8
fastapi           0.115.8
h11               0.14.0
idna              3.10
Jinja2            3.1.5
jmespath          1.0.1
MarkupSafe        3.0.2
pip               24.0
pydantic          2.10.6
pydantic_core     2.27.2
python-dateutil   2.9.0.post0
python-dotenv     1.0.1
s3transfer        0.11.2
six               1.17.0
sniffio           1.3.1
starlette         0.45.3
typing_extensions 4.12.2
urllib3           2.3.0
uvicorn           0.34.0
```

Breaking down the API

In our web app, we need an API because it acts as a bridge between the frontend and the Amazon Bedrock chatbot. The API takes the user's question from the web interface, sends it to the chatbot, and then returns the chatbot's response back to the user. Without the API, the frontend wouldn’t be able to communicate with the AI model or the Knowledge Base. It ensures smooth, secure, and structured interaction between the user and the chatbot.

Digging deeper into the API code, the main tools we need to import are FastAPI, boto3, os, and load_dotenv. These tools help us by simplifying backend development and securely connecting our web app to Amazon Bedrock. FastAPI builds and runs the API efficiently, boto3 lets us access AWS services like Bedrock and S3, os retrieves environment variables securely, and load_dotenv loads those variables from a .env file—making our app safer, more organized, and easier to manage.

The environment variables we need are AWS_REGION, KNOWLEDGE_BASE_ID, and MODEL_ARN. We store them separately because hardcoding them into our script is a security risk—especially if the code is pushed to GitHub. Using environment variables keeps sensitive information like AWS credentials and model IDs secure and makes it easier to manage and update settings without changing the code itself.

placeholder

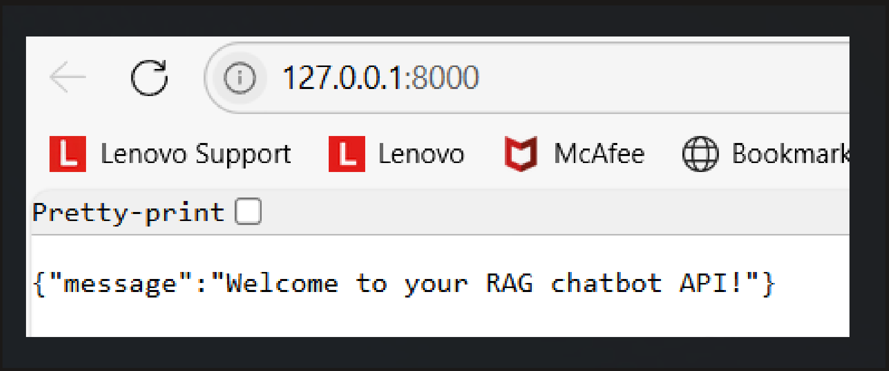
My API uses the Python AWS SDK (boto3) to connect to Amazon Bedrock and handle chatbot interactions programmatically.

The difference between the AWS CLI and an SDK is that the CLI is a command-line tool used for quick, manual tasks in the terminal, while an SDK is a library you import into your code to let applications interact with AWS services automatically. The SDK is ideal for building apps, while the CLI is great for testing or running one-off commands.

Our API has two main routes: /, which is the root route that simply confirms the API is running, and /bedrock/query, which sends user questions to the chatbot via the Knowledge Base. In general, routes help us organize different functions in our API and define how the app should respond to specific requests. They allow us to handle multiple tasks, like sending queries, switching models, or managing different chatbot configurations, all through clear and structured endpoints.

Testing the API

When I visited the root endpoint, I saw {"message":"Welcome to your RAG chatbot API!"}. This confirms that the FastAPI server is running successfully and the backend is correctly set up. It also shows that the application is responding to requests and ready to handle further API calls for both the Knowledge Base and direct model interactions.

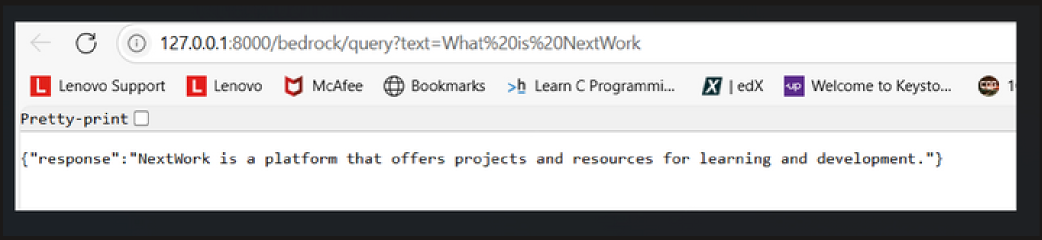


```
{"message":"Welcome to your RAG chatbot API!"}
```

Troubleshooting the API

My second endpoint had errors!

To fix the Parameter Failed error, I had to set the missing environment variables KNOWLEDGE_BASE_ID and MODEL_ARN in the .env file. These values were initially None, causing the API to fail validation. Doing this helps the API pass the correct parameters to Amazon Bedrock, allowing it to connect with the Knowledge Base and generate accurate chatbot responses.

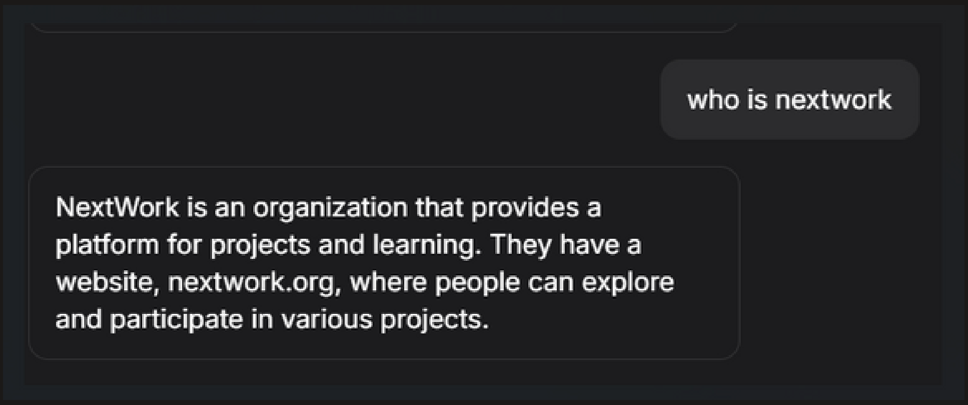


```
{"response":"NextWork is a platform that offers projects and resources for learning and development."}
```

Running the Web App

The difference between the API and the web app for the user is that the API is a backend service that processes requests and returns responses in JSON, usually accessed by developers or other applications. The web app, on the other hand, provides a user-friendly interface where users can type messages and see responses visually. While the API handles the logic, the web app delivers the full chat experience by connecting the frontend to the backend.

The web app has the ability to switch between using the Knowledge Base (RAG) and talking to the AI directly via the model. When I asked the same question through both modes, the RAG-enabled response pulled specific details from my uploaded documents, while the direct AI response relied on general knowledge. This functionality lets users choose between personalized, document-based answers and broader, general-purpose responses—all within a simple interface.



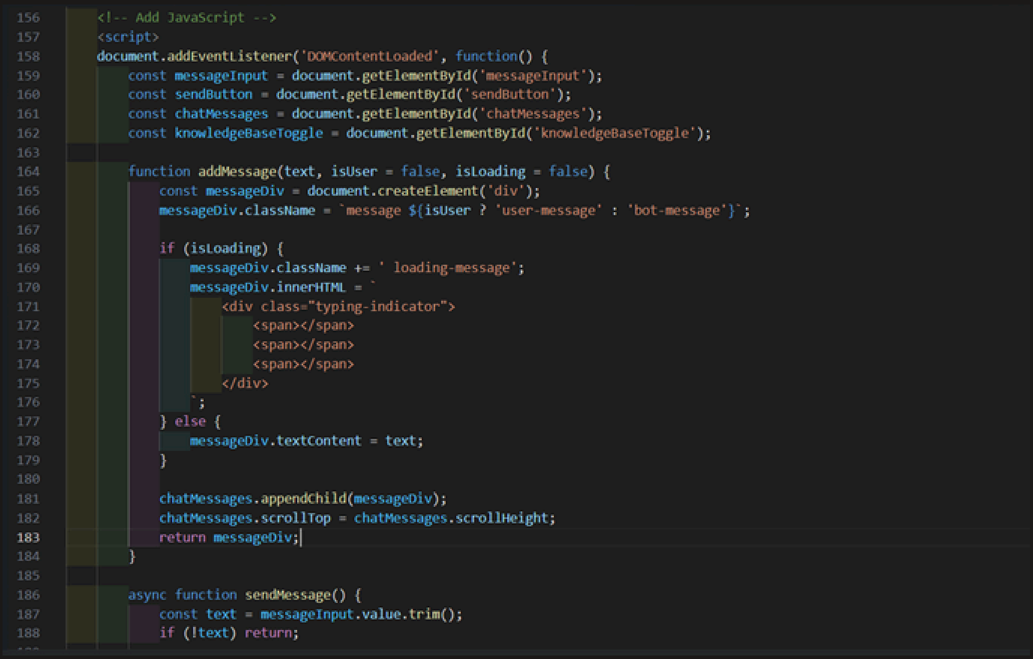
who is nextwork

NextWork is an organization that provides a platform for projects and learning. They have a website, nextwork.org, where people can explore and participate in various projects.

Breaking down the Web App

When a user sends a chat message through the web app, the frontend captures the input and sends it to the backend API using a request. The backend receives the message, either sends it to Amazon Bedrock directly or through the Knowledge Base, and gets a response from the AI model. It then returns that response to the frontend, which displays it to the user. This flow connects the UI, API routes, and AWS services to create a smooth chat experience.

web_app.py extends the API by adding a frontend layer using HTML templates and static files, allowing users to interact with the chatbot through a web interface. Unlike main.py, which only handles API logic, web_app.py serves both the API and the UI. It includes template rendering with Jinja2, static file handling, and a home route (/) that loads index.html. This makes web_app.py a full-stack app, combining backend logic with a user-friendly frontend experience.

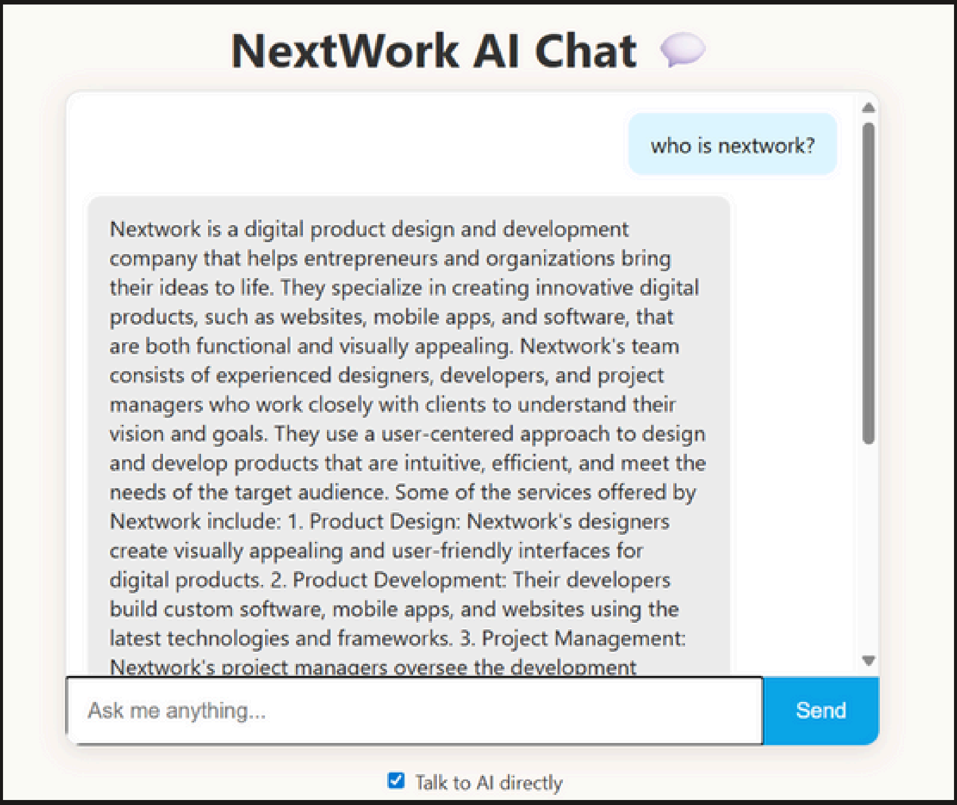


```
<!-- Add JavaScript -->
<script>
document.addEventListener('DOMContentLoaded', function() {
    const messageInput = document.getElementById('messageInput');
    const sendButton = document.getElementById('sendButton');
    const chatMessages = document.getElementById('chatMessages');
    const knowledgeBaseToggle = document.getElementById('knowledgeBaseToggle');

    function addMessage(text, isUser = false, isLoading = false) {
        const messageDiv = document.createElement('div');
        messageDiv.className = `message ${isUser ? 'user-message' : 'bot-message'}`;

        if (isLoading) {
            messageDiv.className += ' loading-message';
            messageDiv.innerHTML = `
                <div class="typing-indicator">
                    <span></span>
                    <span></span>
                    <span></span>
                </div>
            `;
        } else {
            messageDiv.textContent = text;
        }

        chatMessages.appendChild(messageDiv);
        chatMessages.scrollTop = chatMessages.scrollHeight;
        return messageDiv;
    }

    async function sendMessage() {
        const text = messageInput.value.trim();
        if (!text) return;
```

Customizing the Frontend

I want to experiment with customizing the frontend because it allows me to improve the user experience, match the design to my project's branding, and add features like input hints or styling. This is possible because as long as the API endpoints stay the same, the frontend can be updated freely without breaking the connection to the backend. It gives me flexibility to enhance usability while keeping the chatbot functionality intact.

My customized interface now features a clean, pastel-colored design with a card-style chatbox centered on the screen. It replaces the original dark-themed layout with a light, modern look. The chat messages are displayed in soft bubbles, with user messages on the right and bot responses on the left. A simple toggle lets users switch between talking to the Knowledge Base or the AI model directly. The layout is minimalist, responsive, and designed to enhance usability without changing the API functionality.



NextWork AI Chat

who is nextwork?

Nextwork is a digital product design and development company that helps entrepreneurs and organizations bring their ideas to life. They specialize in creating innovative digital products, such as websites, mobile apps, and software, that are both functional and visually appealing. Nextwork's team consists of experienced designers, developers, and project managers who work closely with clients to understand their vision and goals. They use a user-centered approach to design and develop products that are intuitive, efficient, and meet the needs of the target audience. Some of the services offered by Nextwork include: 1. Product Design: Nextwork's designers create visually appealing and user-friendly interfaces for digital products. 2. Product Development: Their developers build custom software, mobile apps, and websites using the latest technologies and frameworks. 3. Project Management: Nextwork's project managers oversee the development