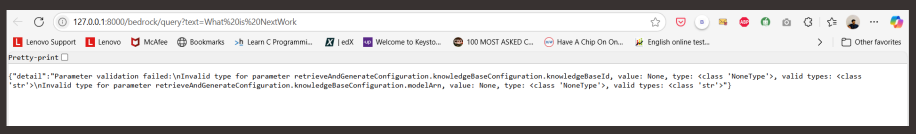
nextwork.org

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

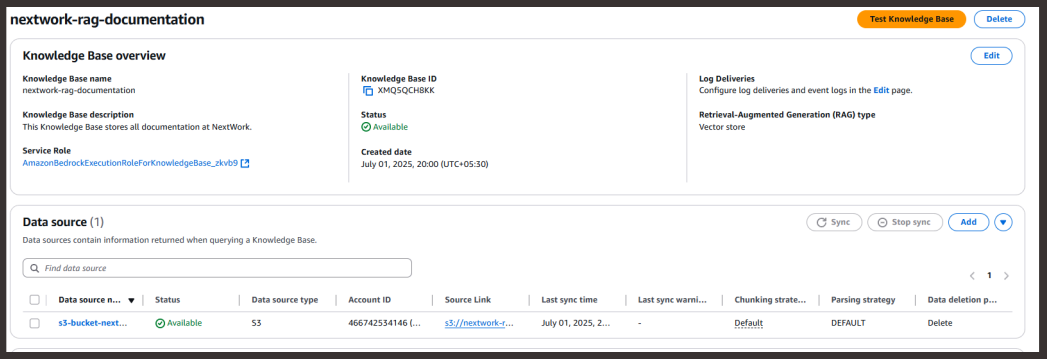
To set up my Knowledge Base, I used S3 to securely store and organize the documents my chatbot will learn from. The documents I uploaded contain information about a student's NextWork project experience, including skills developed and feedback received. S3 makes it easy to manage and access this data, 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.

DI

Dineshraj Dhanapathy
NextWork Student



DI
Dineshraj Dhanapathy
NextWork Student

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.

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.

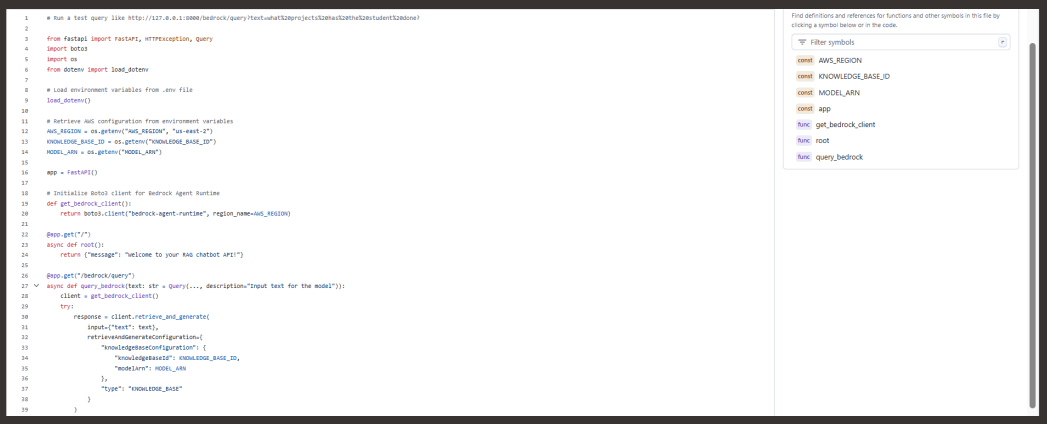
Creating an API

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.

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 chatbot's response, allowing smooth communication between the user and the AI.

DI

Dineshraj Dhanapathy
NextWork Student



```python
# 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"
            }
        )
```

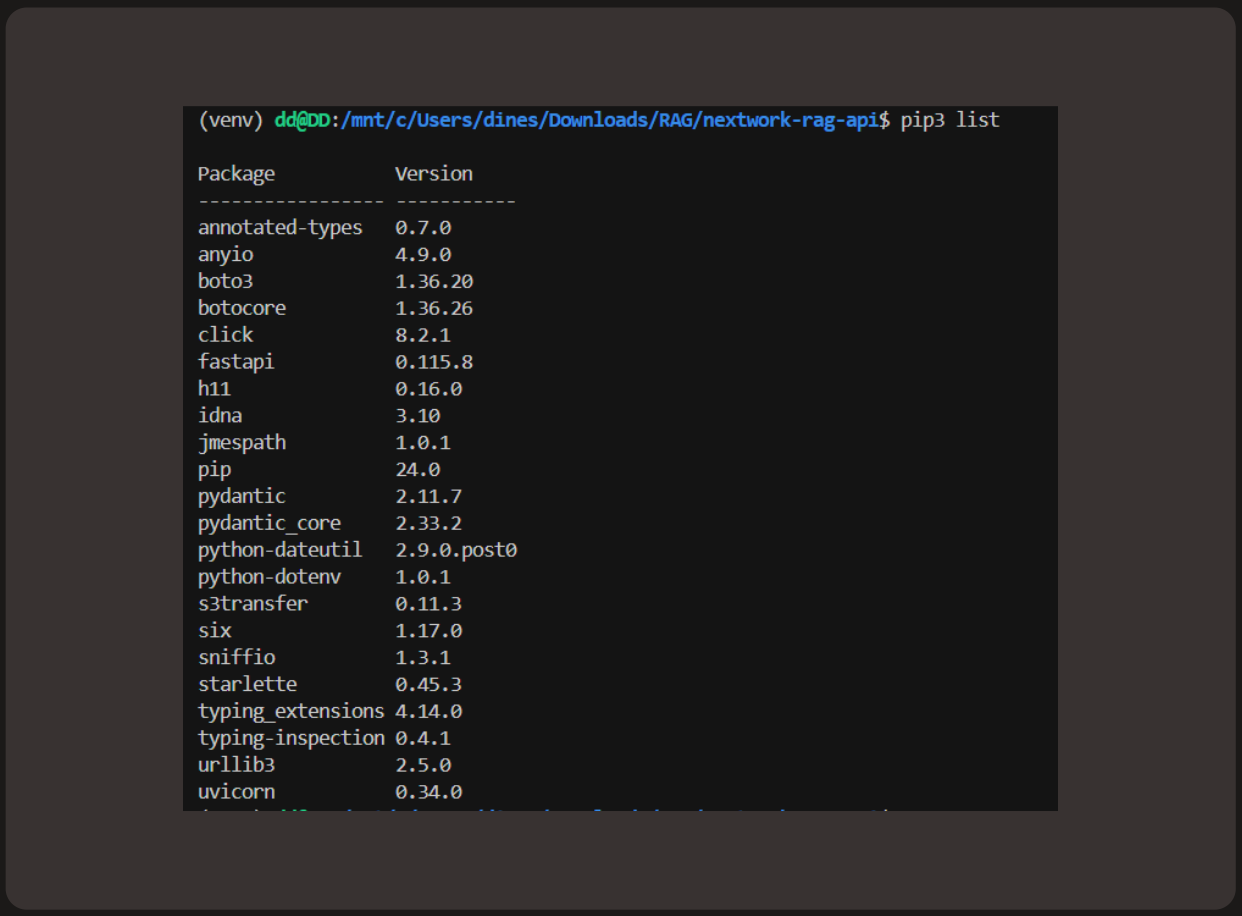
Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

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

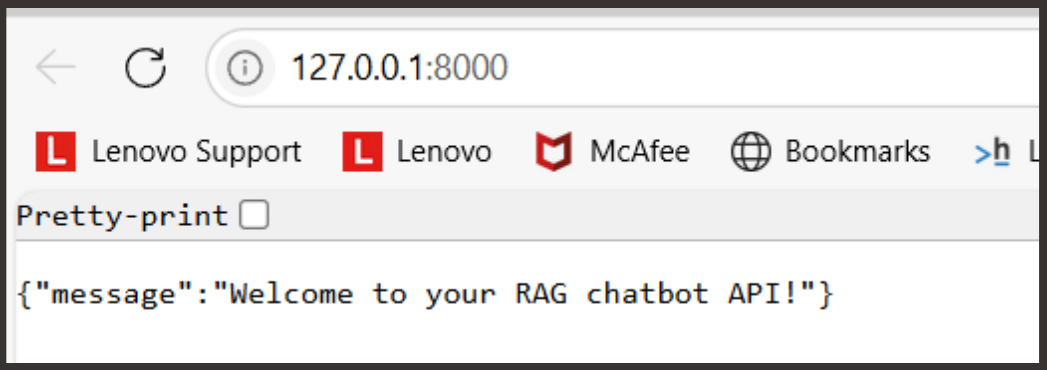
Installing Packages

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.



Testing the API

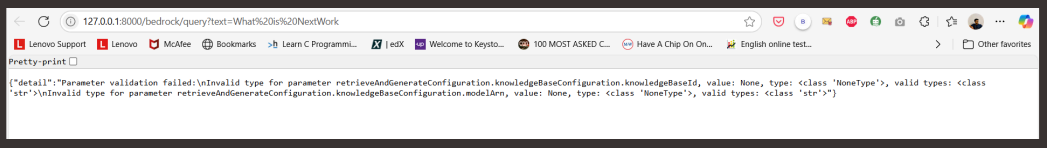
When I visited the root endpoint, I saw the message: "message": "Welcome to your RAG chatbot API!" This confirms that the FastAPI server is running correctly and the API is live. It also means that the routing is set up properly, and the application is ready to handle requests, such as sending user input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



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 retrieveAndGenerateConfiguration.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.

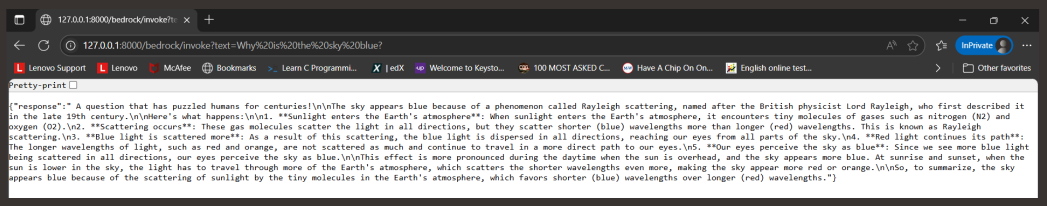

127.0.0.1:8000/bedrock/query?text=What%20is%20NextWork
Lenovo Support
Lenovo
McAfee
Bookmarks
Learn C Programmi...
| edX
Welcome to Keysto...
100 MOST ASKED C...
Have A Chip On On...
English online test...
Other favorites
Pretty-print
{"detail":"Parameter validation failed:\nInvalid type for parameter 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'>"}

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 model's 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 model's training.


127.0.0.1:8000/bedrock/invoke?text=Why%20is%20the%20sky%20blue?
Pretty-print
{"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."}