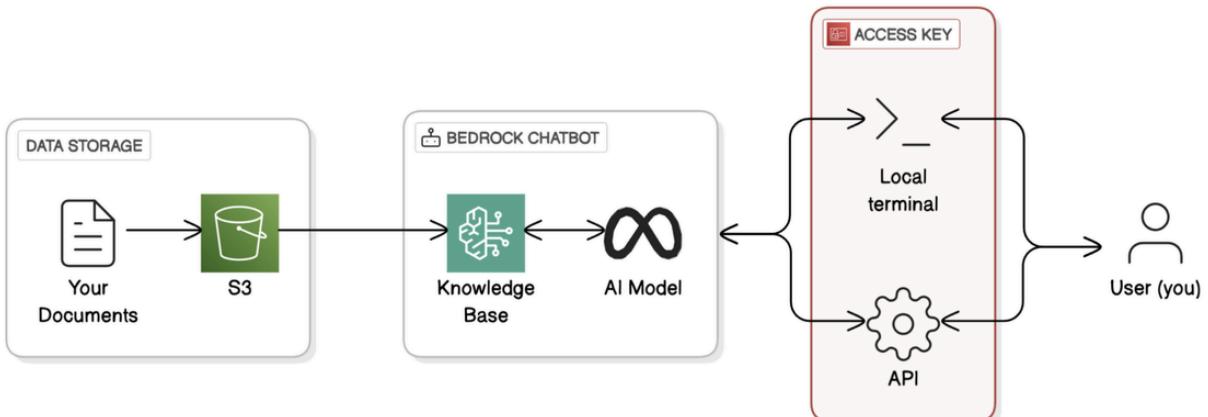




Kareshma  
Rajaananthapadmanaban

# Create an API for Your RAG Chatbot

Build a RAG chatbot using Amazon Bedrock, and create an API that makes your chatbot accessible to any application!

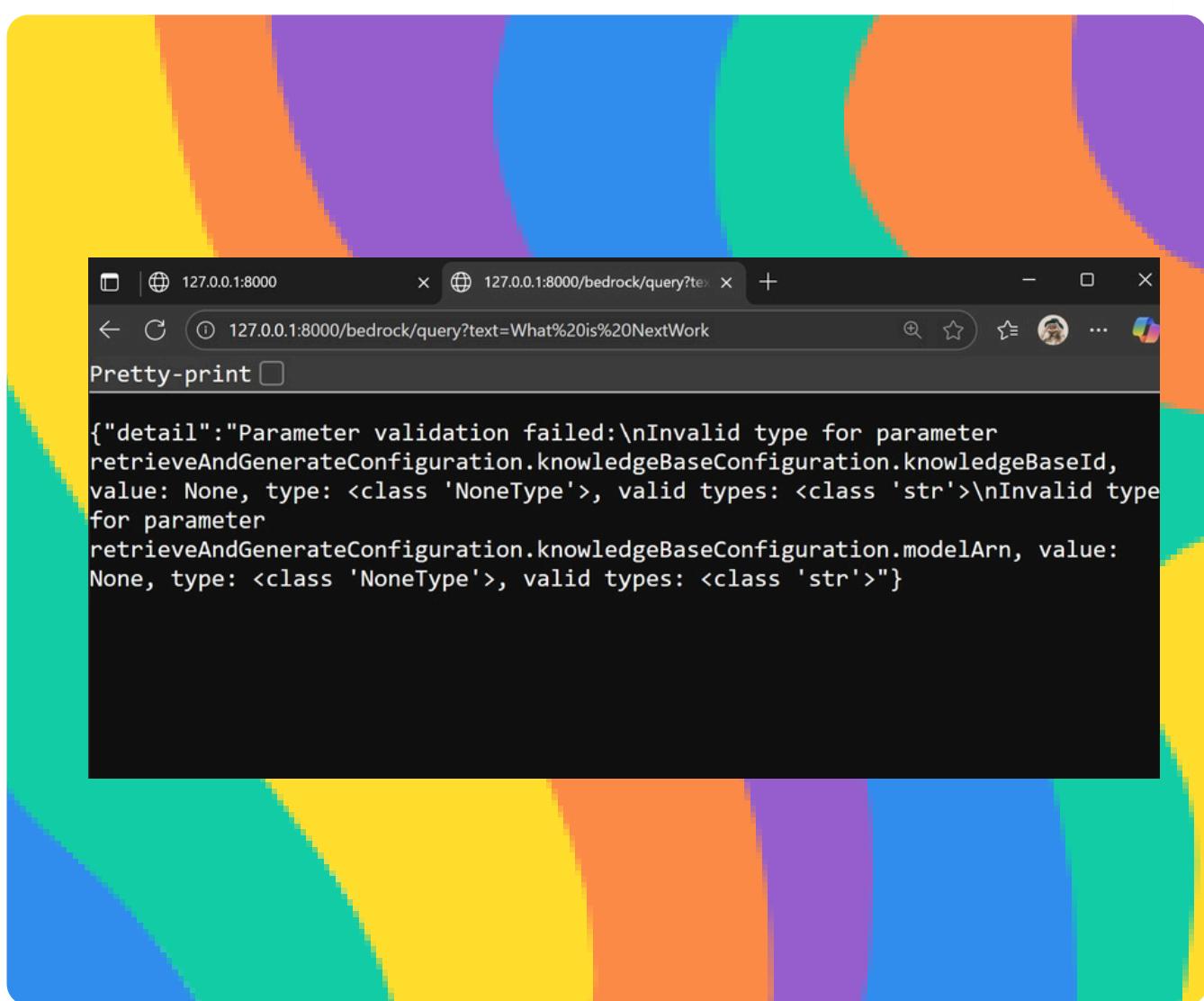




# How I Built an API for My RAG Chatbot



Kareshma Rajaananthapadmanaban





# Introducing Today's Project!

In this project, I will demonstrate how to turn my **RAG chatbot** into an **API** that any application can access. I'm doing this project to learn how to connect Amazon Bedrock with **FastAPI** so my chatbot isn't limited to the console or terminal. This way, I can make it **production-ready** and integrate it into websites, apps, or even other tools.

## Tools and concepts

Services I used were **Amazon Bedrock** for calling foundation models, **FastAPI** for building **API endpoints**, and `dotenv` for managing environment variables. Key concepts I learnt include the difference between **MODEL\_ID** and **MODEL\_ARN**, how to extend an API with new endpoints, handling environment variables, and the contrast between direct model invocation and Knowledge Base queries.

## Project reflection

This project took me approximately 2 hours to complete. The most challenging part was troubleshooting the missing `MODEL_ID` environment variable for the new endpoint. It was most rewarding to see the chatbot handle both Knowledge **Base queries** and **direct AI model** questions, giving me flexibility to use specialized or general knowledge.

I did this project today to deepen my hands-on understanding of how **Retrieval-Augmented Generation** works in **AWS Bedrock** and to practice extending APIs for both **document-based** and **general AI queries**. This project met my goals because I now feel more confident in configuring endpoints, managing environment variables, and explaining the difference between Knowledge Base and direct model calls.



# Setting Up The Knowledge Base

In this step, I will create an **S3 bucket** and **upload documents** that my chatbot will learn from. I'm doing this to give my RAG chatbot a knowledge source, since S3 will act as the home for all the information it needs to reference when generating responses.

To set up my Knowledge Base, I used **S3** to store the reference documents my chatbot will learn from. The documents I uploaded contain information about the NextWork project documentation, including project steps, outcomes, and skills. S3 acts as the **central storage location**, making it easy for Bedrock to retrieve and process this data securely for my chatbot.

Amazon Bedrock is an AWS service for building **generative AI applications**. I created a Knowledge Base in Bedrock to store and organize my chatbots' information, enabling it to quickly find and answer questions from my S3 documents. This setup gives my **chatbot** a structured “**brain**” that understands the meaning of the text, not just exact key words .

I will select an **AI model** in **Amazon Bedrock** because the model will act as the brain of my chatbot, turning raw Knowledge Base results into coherent, **human-like responses** that users can easily understand and interact with.

My chatbot also needs access to **two AI models** (Llama 70B, Titan text embedding V2) to process text and generate human- like responses. I then **synchronized my Knowledge Base** with my **S3 data** because this ensures the chatbot can read, understand, and store the documents in the vector database, allowing it to answer questions accurately.

Kareshma  
Rajaananthapadmanaban

[next work.org](#)

The screenshot shows the 'Knowledge Base overview' section of the Amazon Bedrock interface. The knowledge base is named 'nextwork-rag-documentation'. Key details include:

- Knowledge Base ID:** 2V55BXLKNM
- Status:** Available
- Created date:** September 22, 2025, 20:54 (UTC+05:30)
- Log Deliveries:** Configure log deliveries and event logs in the [Edit](#) page.
- Retrieval-Augmented Generation (RAG) type:** Vector store

The 'Data source (1)' section lists an S3 bucket as a data source. The table shows:

Data source name	Status	Data sour...	Account ID	Source Link
s3-bucket-nextwork-rag-document...	Available	S3	39040386...	<a href="s3://next...">s3://next...</a>

## Knowledge Base creation



# Running CLI Commands Locally

In this step, I will set up my **local terminal** to run AWS CLI commands because this lets me **interact** with my **Knowledge Base** and **test** my **chatbot directly from my computer**, making it faster and easier to verify responses before building the API.

When I ran a Bedrock command in my terminal, I initially got an error because AWS CLI wasn't installed. To resolve this, I downloaded and installed the AWS CLI for my operating system, then verified the installation with **aws --version**. Once I saw the version number, I knew the CLI was set up correctly and ready to connect with my Knowledge Base.

I set up **AWS access keys** to let my local terminal securely connect to AWS and run Bedrock CLI commands. Since the CLI doesn't share authentication with the AWS Management Console, the keys act like a username and password pair. This way, I could test my Knowledge Base **queries locally**, validate responses faster, and prepare my environment before building the API.

To pass my access key credentials to the AWS CLI, I ran the command `aws configure`. I had to provide my **Access Key ID**, **Secret Access Key**, and set the **region** to **us-east-2** where my Knowledge Base is hosted. For the default output format, I left it empty. This allowed my terminal to securely connect to AWS services like Bedrock.

# Running CLI Commands Locally

```
PS C:\Users\kares> aws bedrock-agent-runtime retrieve-and-generate \
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:
aws help
aws <command> help
aws <command> <subcommand> help

aws.exe: error: the following arguments are required: --input

PS C:\Users\kares> --input '{"text": "What is NextWork?"} \
ParserError:
Line |
  1 |   --input '{"text": "What is NextWork?"}' \
     |   ^
     | Missing expression after unary operator '--'.
PS C:\Users\kares> --retrieve-and-generate-configuration '{
>>   "knowledgeBaseConfiguration": {
>>     "knowledgeBaseId": "your_knowledge_base_id",
>>     "modelArn": "your_model_arn"
>>   },
>>   "type": "KNOWLEDGE_BASE"
>> }'
ParserError:
Line |
  1 |   --retrieve-and-generate-configuration '{
     |   ^
     | Missing expression after unary operator '--'.
PS C:\Users\kares>
```

PowerShell parsing Error

In command using \ at the end of lines. That's bash syntax, **not PowerShell**.  
PowerShell doesn't understand line continuation with \, so it throws “**Missing expression after unary operator '--'**.

To fix the error I use **backtick `** for line continuation.

```
aws bedrock-agent-runtime retrieve-and-generate `-
--input '{"text": "What is NextWork?"}`-
--retrieve-and-generate-configuration '{
  "knowledgeBaseConfiguration": {
    "knowledgeBaseId": "your_knowledge_base_id",
    "modelArn": "your_model_arn"
  },
  "type": "KNOWLEDGE_BASE"
}'
```

Fixed command

```
PS C:\Users\kares> aws configure
AWS Access Key ID [None]: AKIAVZPCGYUBZLRGEWQ
AWS Secret Access Key [None]: +am42HwcJfloZcYnrEHh02d3DmnIn4CBcYgEaH43
Default region name [us-east-2]:
Default output format [None]: json
PS C:\Users\kares> aws bedrock get-foundation-model --model-identifier meta.llama3-3-70b-instruct-v1:0
{
  "modelDetails": {
    "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0",
    "modelId": "meta.llama3-3-70b-instruct-v1:0",
    "modelName": "Llama 3.3 70B Instruct",
    "providerName": "Meta",
    "inputModalities": [
      "TEXT"
    ],
    "outputModalities": [
      "TEXT"
    ],
    "responseStreamingSupported": true,
    "customizationsSupported": [],
    "inferenceTypesSupported": [
      "ON_DEMAND",
      "INFERENCE_PROFILE"
    ],
    "modelLifecycle": {
      "status": "ACTIVE"
    }
  }
}

PS C:\Users\kares> |
```

## AWS Configure and getting Model ARN

When I first ran Bedrock CLI commands, I kept getting **Unable to locate credentials** and **You must specify a region**. To fix it, I used `aws configure` and entered the **Access Key ID** and **Secret Access Key** I created in IAM under Security Credentials. I set the region to us-east-2 and output to json.

## Retrieve access keys Info

### Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key

Secret access key



AKIAW3MEFRAF2IPLWWFV



\*\*\*\*\* Show

# Running Bedrock Commands

In this step, I will locate my **Knowledge Base ID** and **Model ARN** and update my Bedrock CLI command because this allows me to successfully **send queries** to my chatbot from the **terminal** and verify that it can respond accurately using the data in my Knowledge Base.

When I first ran a Bedrock command, I ran into an error because I needed to **provide values** for the Knowledge Base ID and the Model ARN. Without these identifiers, Bedrock **didn't know** which **Knowledge Base to query** or which AI model to use, so the command couldn't process my request and returned a validation error.

To find the required values, I had to first locate my Knowledge Base ID in the Bedrock console and then **retrieve my model ARN** using the CLI with the model identifier `meta.llama3-3-70b-instruct-v1:0`. The Bedrock command ran successfully and showed me a context-rich answer from my chatbot, confirming that my Knowledge Base was correctly linked to the AI model and ready to respond to queries from the terminal.

```
aws bedrock-agent-runtime retrieve-and-generate `--input '{"text": "What is NextWork?"}'` --retrieve-and-generate-configuration '{ "knowledgeBaseConfiguration": { "knowledgeBaseId": "2VS5BXLKNM", "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0" }, "type": "KNOWLEDGE_BASE" }'
```

Kareshma  
Rajaananthapadmanaban

[nextwork.org](https://nextwork.org)

# Running Bedrock Commands

```
PS C:\Users\kares> aws bedrock-agent-runtime retrieve-and-generate `>> --input '{"text": "What is NextWork?"}' `>> --retrieve-and-generate-configuration '{`>>     "knowledgeBaseConfiguration": {`>>         "knowledgeBaseId": "2VS5BXLKNM",`>>         "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0" `>>     },`>>     "type": "KNOWLEDGE_BASE" `>> }`> {`>     "citations": [`>         {`>             "generatedResponsePart": {`>                 "textResponsePart": {`>                     "span": {`>                         "end": 182,`>                         "start": 0`>                     },`>                     "text": "NextWork is a website that offers various projects, including those related`> to AWS DevOps and Terraform, and has a community forum where users can ask questions and share information."`>                 },`>             },`>             "retrievedReferences": [`>                 {`>                     "content": {`>                         "text": "https://community.nextwork.org/c/i-have-a-question?automatic_login=true`> https://community.nextwork.org/c/i-have-a-question?automatic_login=true https://communi`> ty.nextwork.org/c/i-have-a-question?automatic_login=true https://community.nextwork.org/c/i-have`> -a-question?automatic_login=trueKareshma Rajaananthapadmanaban nex t wor k.org Website loaded throug`> h CloudFront. To improve performance and security, I enabled caching and HTTPS in CloudFront. This e`> nsures that static assets are served quickly and securely, even during high traffic. Updating the site i`> s simple just re-upload files to the S3 bucket, and CloudFront handles the global delivery with minimal`> "}}`>     },`> 
```

Local Terminal getting Bedrock command's results

# Creating an API

An API is an Application Programming Interface that lets **different software communicate with each other**. The API I'm creating will let **any application send questions** to my **RAG chatbot** and **receive responses**, without needing to access the AWS Console or CLI. This will be helpful for integrating the chatbot into websites, mobile apps, or other tools, making it accessible and interactive for users everywhere.

The API code has three main sections that work together to make the chatbot accessible. **First, Imports** bring in necessary libraries like **FastAPI** for the API framework, **boto3** for AWS interactions, and **dotenv/os** for environment variables. **Second, Environment Variables** load your **AWS region**, **Knowledge Base ID**, and **model ARN** to connect the API to Bedrock. **Third, Functions and API Endpoints** define routes like the **root page** and **/bedrock/query** to handle requests and send them to the chatbot.

I set up the API by first cloning the GitHub repository using **git clone [HTTPS-URL]**, which downloaded all the **project files locally**. Next, I created a virtual environment with **python -m venv venv** to keep dependencies isolated from my system Python. Finally, since I was on Windows, I activated the virtual environment using **.\venv\Scripts\activate**

When I first tried running the API with **python -m unicorn main:app --reload**, I got a **ModuleNotFoundError: No module named unicorn**. This happened because the required dependencies were not installed in my virtual environment. To fix this, I ran **pip3 install -r requirements.txt**, which installed all the necessary Python packages like **FastAPI** and **Uvicorn**. After that, the API started without errors, and I could continue testing it.

Kareshma  
Rajaananthapadmanaban

[nextwor\\_k.org](https://nextwor_k.org)

The screenshot shows a GitHub code editor interface with a multi-colored background. The main window displays the Python file `main.py` from the repository `github.com/NatNextWork1/nextwork-rag-api/blob/main/main.py`. The code implements a FastAPI application for a RAG chatbot API. It uses Boto3's Bedrock Agent Runtime to handle queries. The GitHub UI includes a sidebar for symbols and a search bar.

```
13 KNOWLEDGE_BASE_ID = os.getenv("KNOWLEDGE_BASE_ID")
14 MODEL_ARN = os.getenv("MODEL_ARN")
15
16 app = FastAPI()
17
18 # Initialize Boto3 client for Bedrock Agent Runtime
19 def get_bedrock_client():
20     return boto3.client("bedrock-agent-runtime", region_name=AWS_REGION)
21
22 @app.get("/")
23 async def root():
24     return {"message": "Welcome to your RAG chatbot API!"}
25
26 @app.get("/bedrock/query")
27 async def query_bedrock(text: str = Query(..., description="Input text for the mo
28     client = get_bedrock_client()
29     try:
30         response = client.retrieve_and_generate(
31             input={"text": text},
32             retrieveAndGenerateConfiguration={
33                 "knowledgeBaseConfiguration": {
34                     "knowledgeBaseId": KNOWLEDGE_BASE_ID,
35                     "modelArn": MODEL_ARN
36                 },
37                 "type": "KNOWLEDGE_BASE"
38             }
39         )
40         return {"response": response["output"]["text"]}
41     except Exception as e:
42         raise HTTPException(status_code=500, detail=str(e))
```

main.py file in GotHub showing API code

The screenshot shows a PowerShell session with two tabs. The current tab is running in `PowerShell 7 (x64)`. The user is navigating to the project directory, activating the virtual environment, and running `uvicorn` to start the application. They then install dependencies using `pip3`. The output shows the installation of various packages including `boto3`, `fastapi`, `python-dotenv`, and `uvicorn`.

```
PS D:\Projects\Nxtwrkcode> cd nextwork-rag-api
PS D:\Projects\Nxtwrkcode\nextwork-rag-api> python -m venv venv
PS D:\Projects\Nxtwrkcode\nextwork-rag-api> .\venv\Scripts\activate
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> echo $VIRTUAL_ENV
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> python -m uvicorn main:app --reload
D:\Projects\Nxtwrkcode\nextwork-rag-api\venv\Scripts\python.exe: No module named uvicorn
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> cat requirements.txt
boto3==1.36.20
fastapi==0.115.8
python-dotenv==1.0.1
uvicorn==0.34.0
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> pip3 install -r requirements.txt
Collecting boto3==1.36.20 (from -r requirements.txt (line 1))
  Downloading boto3-1.36.20-py3-none-any.whl.metadata (6.7 kB)
Collecting fastapi==0.115.8 (from -r requirements.txt (line 2))
  Downloading fastapi-0.115.8-py3-none-any.whl.metadata (27 kB)
Collecting python-dotenv==1.0.1 (from -r requirements.txt (line 3))
  Downloading python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Collecting uvicorn==0.34.0 (from -r requirements.txt (line 4))
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
Collecting botocore<1.37.0,>=1.36.20 (from boto3==1.36.20->-r requirements.txt (line 1))
  Downloading botocore-1.36.26-py3-none-any.whl.metadata (5.7 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3==1.36.20->-r requirements.txt (line 1))
  Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
Collecting s3transfer<0.12.0,>=0.11.0 (from boto3==1.36.20->-r requirements.txt (line 1))
  Downloading s3transfer-0.11.5-py3-none-any.whl.metadata (1.7 kB)
Collecting starlette<0.46.0,>=0.40.0 (from fastapi==0.115.8->-r requirements.txt (line 2))
  Downloading starlette-0.45.3-py3-none-any.whl.metadata (6.3 kB)
Collecting pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4 (from fastapi==0.115.8->-r requirements.txt (line 2))
  Downloading pydantic-2.11.9-py3-none-any.whl.metadata (68 kB)
Collecting typing-extensions>=4.8.0 (from fastapi==0.115.8->-r requirements.txt (line 2))
  Downloading typing_extensions-4.15.0-py3-none-any.whl.metadata (3.3 kB)
```

# Installing Packages

To run the API, I had to install requirements like `fastapi`, `unicorn`, `boto3`, and `python-dotenv`. These packages are important because **FastAPI** provides the **framework** for building the API, **Uvicorn** runs the **API server and handles requests**, **Boto3** connects the **API to AWS Bedrock**, and `python-dotenv` loads environment variables so the API can access the Knowledge Base ID, model ARN, and region.

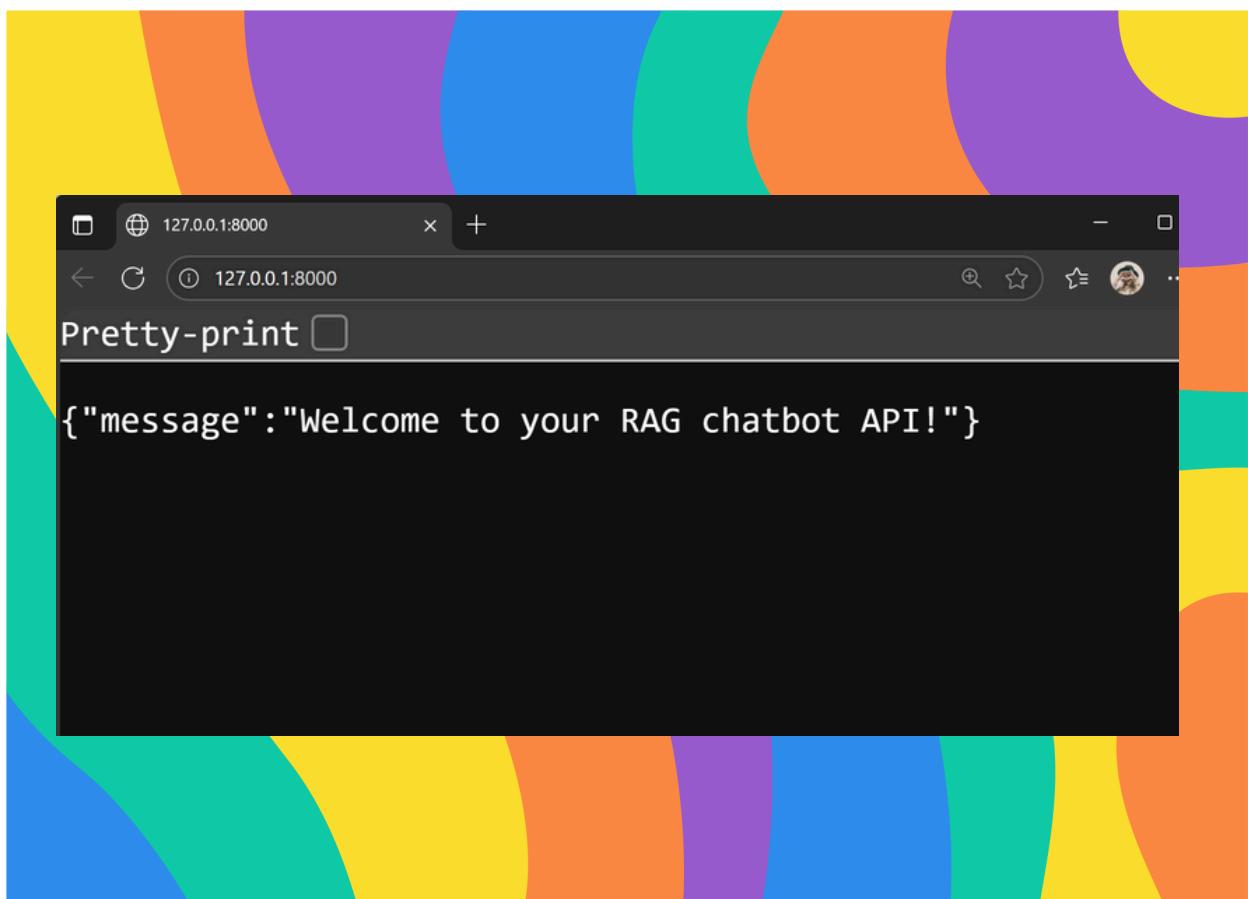
```
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> pip3 list
Package           Version
-----
annotated-types   0.7.0
anyio              4.10.0
boto3              1.36.20
botocore           1.36.26
click               8.3.0
colorama            0.4.6
fastapi             0.115.8
h11                 0.16.0
idna                3.10
jmespath            1.0.1
pip                  25.2
pydantic             2.11.9
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    4.15.0
typing-inspection    0.4.1
urllib3              2.5.0
unicorn              0.34.0
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> python -m unicorn main:app --reload
INFO: Will watch for changes in these directories: ['D:\\Projects\\Nxtwrkcode\\nextwork-rag-api']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [20720] using StatReload
INFO: Started server process [22700]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

pip list with unicorn running

Running API by `python -m unicorn main:app --reload`

# Testing the API

When I visited the **root endpoint**, I saw `{"message": "Welcome to your RAG chatbot API!"}`. This confirms that the API server is running correctly, the FastAPI framework is handling requests, and the **root endpoint** is working as expected, meaning my local setup is ready to process further **chatbot queries** through the **API**.



Welcome message from API's root endpoint

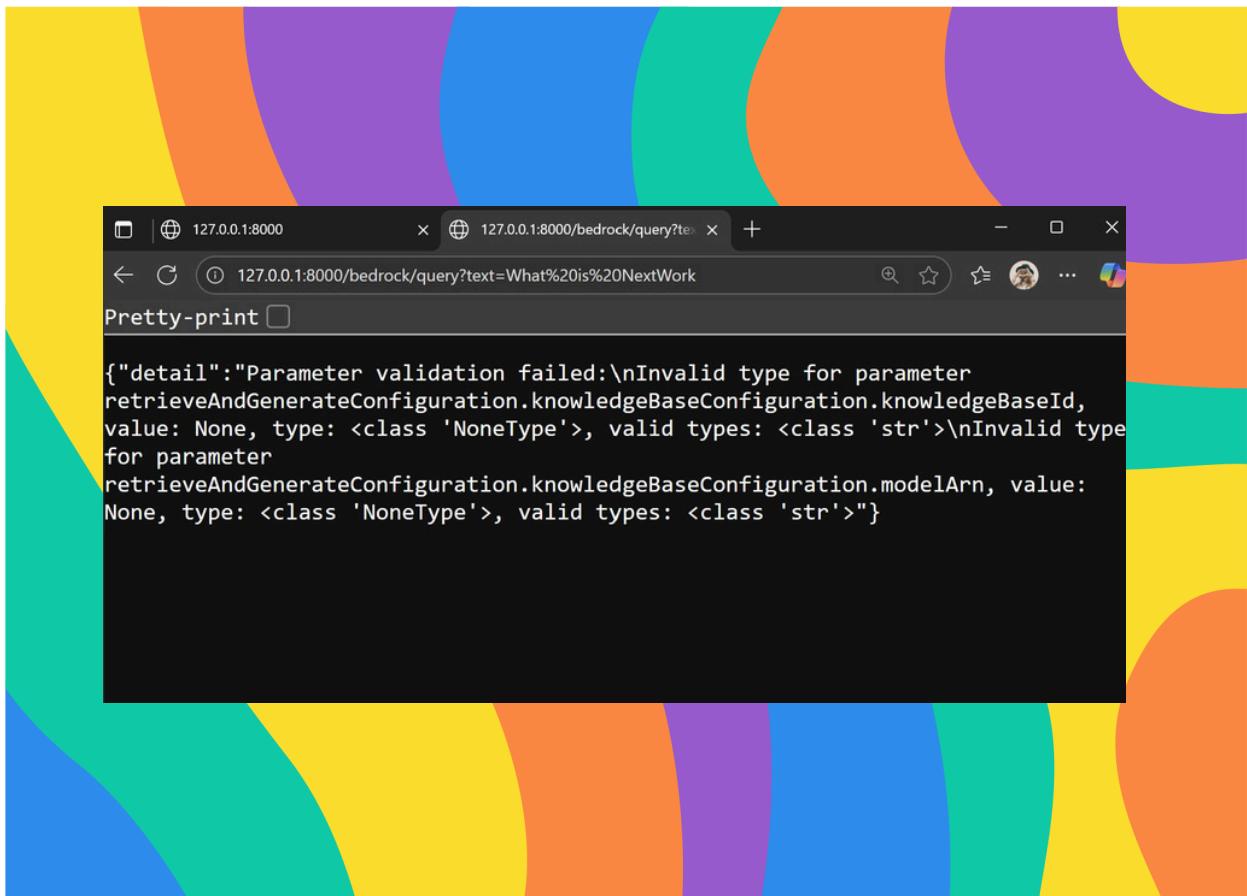
# The Query Endpoint

In this step, I will test the **query endpoint** by sending a question to my chatbot through the API using a URL like `/bedrock/query?text=What%20is%20NextWork?`. This is important because it verifies that the API correctly passes user queries to the Bedrock Knowledge Base and returns accurate, human-like responses, showing that my chatbot is fully functional and ready for integration.

The query endpoint **connects** an **app with my Bedrock Knowledge Base** and AI model. I tested the query endpoint by visiting `http://127.0.0.1:8000/bedrock/query?text=What%20is%20NextWork` in my browser. The response was a clear, human-like answer generated from my Knowledge Base documents, confirming that the API correctly processes queries and communicates with the chatbot.

Looking at the API code, I can see that the API calls for environment variables because it uses `os` to load `KNOWLEDGE_BASE_ID` and `MODEL_ARN` **instead of hardcoding them**. To resolve the error in my query endpoint, I need to **set these values as environment variables** on my computer or in a `.env` file so the API can read them securely when sending requests to Bedrock.

I created an **API using FastAPI** that connects to my Amazon Bedrock RAG chatbot. The API reads environment variables for `AWS_REGION`, `KNOWLEDGE_BASE_ID`, and `MODEL_ARN` from a `.env` file. The API endpoint `/bedrock/query` responded with a **human-like answer** from my chatbot, successfully returning information from my Knowledge Base when I asked, "What is NextWork?"



## Error while testing query endpoint

At this step, I ran into **errors** while testing the query endpoint. To fix it, I added my Knowledge Base ID and Model ARN as environment variables. I created a `.env` file using `New-Item .env` and opened it with `notepad .env`. Inside, I set `AWS_REGION=us-east-2`, `KNOWLEDGE_BASE_ID=<my_id>`, and `MODEL_ARN=<my_arn>`. After saving the file, I ran the API again. Visiting `http://127.0.0.1:8000/bedrock/query?text=What%20is%20NextWork` now gave me successful results.

# The Query Endpoint

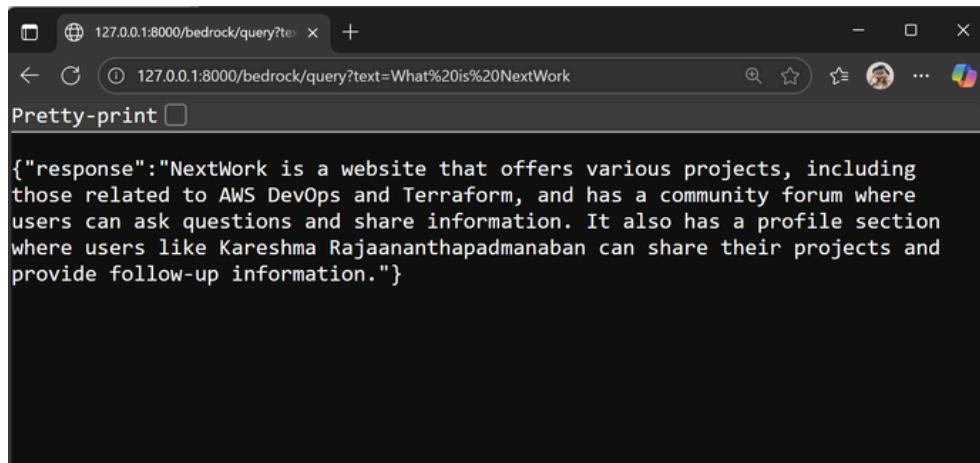
```
Error
INFO: 127.0.0.1:63945 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:59630 - "GET /bedrock/query?text=What%20is%20NextWork HTTP/1.1" 500 Internal Server
Error
INFO: 127.0.0.1:62142 - "GET /bedrock/query?text=What%20is%20NextWork HTTP/1.1" 500 Internal Server
Error
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [22700]
INFO: Stopping reloader process [20720]
(venv) PS D:\Projects\Nxtwrkcode\network-rag-api> New-Item .env

Directory: D:\Projects\Nxtwrkcode\network-rag-api

Mode                LastWriteTime         Length Name
----                <-----              ----- 
-a---        9/22/2025 10:43 PM            0 .env

(venv) PS D:\Projects\Nxtwrkcode\network-rag-api> notepad .env
(venv) PS D:\Projects\Nxtwrkcode\network-rag-api> cat .env
AWS_REGION=us-east-2
KNOWLEDGE_BASE_ID= 2VS5BXLKMN
MODEL_ARN= arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0
(venv) PS D:\Projects\Nxtwrkcode\network-rag-api> python -m uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['D:\\Projects\\Nxtwrkcode\\network-rag-api']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [26404] using StatReload
INFO: Started server process [28520]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:63081 - "GET /bedrock/query?text=What%20is%20NextWork HTTP/1.1" 200 OK
```

Add item to env and again running



Browser showing the response from your query endpoint.



# Extending the API

In this secret mission, I will add a **new endpoint** to my API that **communicates directly with the AI model**, bypassing the Knowledge Base. This allows my chatbot to **answer general questions** using the **model's built-in knowledge**, giving my API more versatility and letting other applications get instant AI responses without needing document data.

In a project extension, I decided to extend my API by adding a new endpoint that calls the AI model directly, bypassing the Knowledge Base. Changes to the API code include **two new imports**—logging and json for better error tracking and handling JSON data, an additional environment variable MODEL\_ID for direct model calls, and a new endpoint `/bedrock/invoke` that lets the API **respond** to general questions **without document context**.

I initially ran into an error with the new endpoint because the MODEL\_ID environment variable wasn't set. Once I fixed it, I validated the new endpoint by adding MODEL\_ID to my .env file using the part after model/ from the ARN (for example, meta.llama3-3- 70b-instruct-v1:0). After saving the file and restarting the API, I confirmed it worked by hitting `/bedrock/invoke` with a test query like "Why is the sky blue?".

When I use `/bedrock/query`, the chatbot **first searches** my Knowledge Base for context and then generates an answer, which makes it accurate for questions about my documents or projects. But when I use `/bedrock/invoke`, the chatbot **skips the search** and answers directly from its general knowledge, which works for broad questions but **lacks my specific context**.

# Call an AI Model Directly

To query the **AI model directly** I enter a **new URL** that sends a request to the **/bedrock/invoke endpoint**.

This URL will use the **/bedrock/invoke** endpoint to ask your AI model why the sky is blue:

`http://127.0.0.1:8000/bedrock/invoketext=Why%20is%20the%20sky%20blue?`

```
AWS_REGION= us-east-2
KNOWLEDGE_BASE_ID= 2VS5BXLKNM
MODEL_ARN= arn:aws:bedrock:us-east-2::foundation-
model/meta.llama3-3-70b-instruct-v1:0
MODEL_ID= meta.llama3-3-70b-instruct-v1:0
```

When I tested the secret mission, I saw an error `{"detail":"Model ID is missing"}`. This happened because I hadn't set the MODEL\_ID in my .env file. To fix it, I opened .env, added **MODEL\_ID=<my\_model\_id>**, and saved the file.

# Call an AI Model Directly

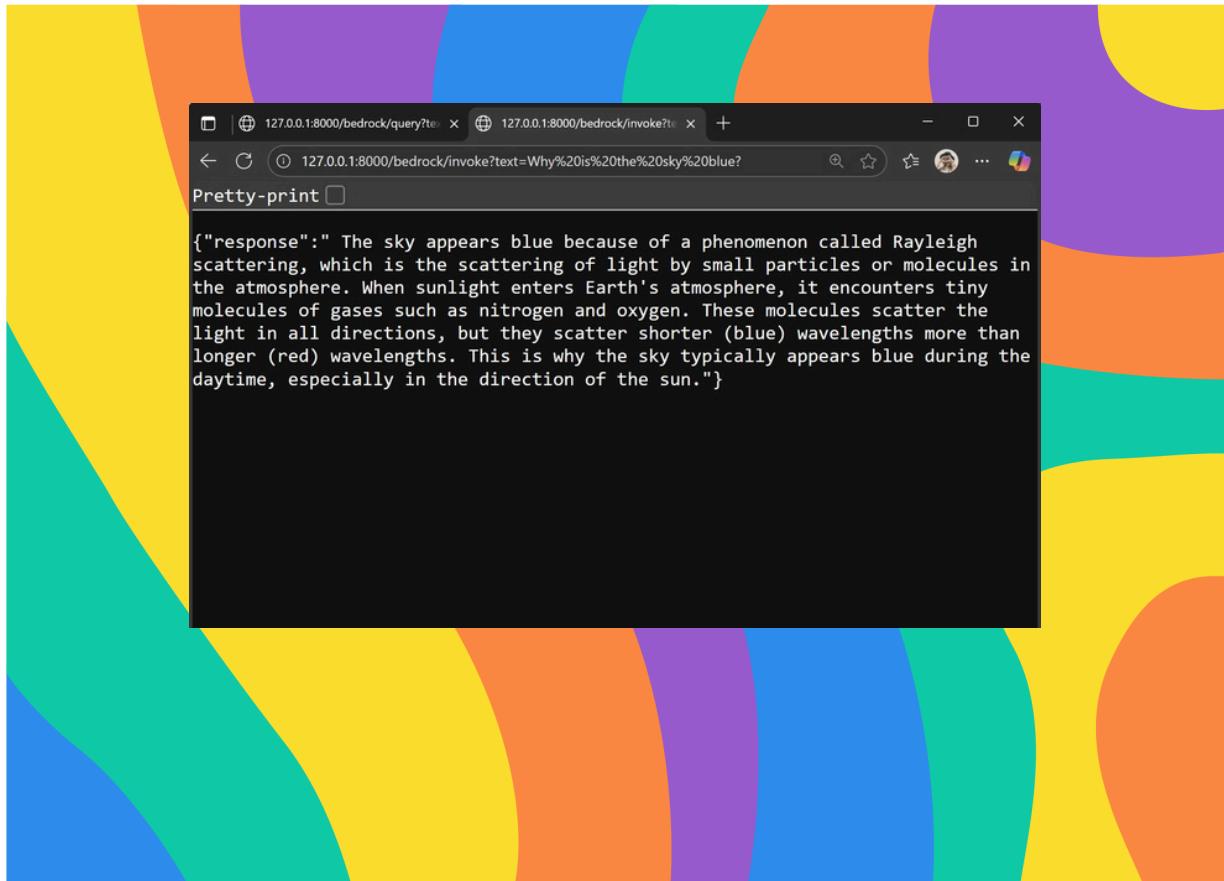
```
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> cat .env
AWS_REGION=us-east-2
KNOWLEDGE_BASE_ID= 2VS5BXLKNM
MODEL_ARN= arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> cat .env
AWS_REGION= us-east-2
KNOWLEDGE_BASE_ID= 2VS5BXLKNM
MODEL_ARN= arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0
MODEL_ID= meta.llama3-3-70b-instruct-v1:0
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> python -m uvicorn secret_mission:app --reload
INFO: Will watch for changes in these directories: ['D:\\\\Projects\\\\Nxtwrkcode\\\\nextwork-rag-api']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [25236] using StatReload
INFO:botocore.credentials:Found credentials in shared credentials file: ~/.aws/credentials
INFO: Started server process [28208]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:62276 - "GET /bedrock/invoke?text=Why%20is%20the%20sky%20blue? HTTP/1.1" 200 OK
INFO: 127.0.0.1:64667 - "GET /bedrock/query?text=Why%20is%20the%20sky%20blue? HTTP/1.1" 200 OK
INFO: 127.0.0.1:64668 - "GET /bedrock/invoke?text=what%20projects%20has%20this%20student%20done? HT
TP/1.1" 200 OK
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [28208]
INFO: Stopping reloader process [25236]
(venv) PS D:\Projects\Nxtwrkcode\nextwork-rag-api> |
```

Running the API server again

Then I restarted the API using `python -m uvicorn secret_mission:app --reload`. This time, when I hit the invoke endpoint, I got a JSON response from the Llama model explaining why the sky is blue.

Kareshma  
Rajaananthapadmanaban

[nextwork.org](https://nextwork.org)



API response for invoke query 'Why is the sky blue"

# Comparing with Knowledge Base query

So I change the URL to use `/bedrock/query` with the **same question** about why the sky is blue:

`http://127.0.0.1:8000/bedrock/query?text=Why%20is%20the%20sky%20blue?`

A screenshot of a browser window showing three tabs. The active tab has the URL `127.0.0.1:8000/bedrock/query?text=Why%20is%20the%20sky%20blue?`. The response is displayed in a JSON-like format:

```
{  
  "response": "I couldn't find an exact answer to the question."  
}
```

Running the API server with bedrock query

Now , I try directly asked the AI model about knowledge that's in my Knowledge Base! For example, if we asked it about projects the I have done:

`http://127.0.0.1:8000/bedrock/invoke?  
text=what%20projects%20has%20this%20student%20done?`

As imagine, the AI model **wouldn't have enough context** to answer this question - it's not using your Knowledge Base!

A screenshot of a browser window showing three tabs. The active tab has the URL `127.0.0.1:8000/bedrock/invoke?text=what%20projects%20has%20this%20student...`. The response is displayed in a JSON-like format:

```
{  
  "response": "This conversation has just begun. I'm happy to chat with you,  
  but I don't have any information about a specific student or their projects. If  
  you'd like to share more context or details, I'd be happy to try and help!"  
}
```

## Delete Your Resources

Now that the chatbot is built and tested, it's time to clean up. Deleting unused resources prevents accidental charges and follows AWS security best practices.

### 1) Knowledge Base

- Go to Bedrock console → Knowledge Bases
- Select your Knowledge Base → Delete
- Type delete in the confirmation field → Confirm

### 2) S3 Bucket

- Open S3 console → Select your bucket bedrock-kb-data-<your\_initials>
- Select Empty → Type permanently delete → Confirm
- Then select the bucket again → Delete
- Type nextwork-rag-documentation-<your\_initials> → Confirm

### 3) Access Key

- Open IAM console → Users → Select your IAM user
- Go to Security credentials tab → Access keys
- Find your access key → Actions → Delete → Deactivate
- Type the access key ID → Confirm deletion

Why? Access keys allow programmatic AWS access. Deleting them removes any chance of unauthorized access.

### 4) Vector Store

- Go to OpenSearch console → Collections
- Select your vector store → Delete
- Type confirm → Confirm deletion



Kareshma  
Rajaananthapadmanaban

*Follow up for more  
interesting projects !!!*



Check out [nextwork.org](https://nextwork.org) / my profile for more projects