

UTD GenAI TA – AI-Powered Learning Assistant for Students

Introduction

"Welcome to UTD GenAI TA – an AI-powered learning assistant designed to help UTD students with their coursework. Instead of relying solely on Teaching Assistants, students can now get instant, accurate responses to their queries based on class materials, lecture slides, and professor-provided documents."

How It Works

1. Uploading Course Materials

"The first step in building this system is creating a knowledge base from course materials. Professors or TAs upload lecture slides, class notes, research papers, and other relevant documents to an S3 bucket in AWS."

2. Creating the Knowledge Base

"Once the materials are uploaded, we configure a **Knowledge Base** in AWS Bedrock. This knowledge base automatically processes documents, converting them into **vector embeddings** for efficient retrieval. The system also breaks the text into chunks, making it easier to locate relevant information."

3. AI-Driven Response Generation

"The Lambda function interacts with the Knowledge Base, retrieving the most relevant information from the uploaded course materials. This retrieved data is then passed to an **AWS Bedrock Foundation Model**, which generates an intelligent, contextual response. The AI doesn't just guess—it bases its answers directly on course content."

Example Use Cases

❖ Scenario 1: Understanding Lecture Concepts

💡 **Student Question:** "Can you explain the difference between normalization and denormalization from my database class?"

❖ **Response:** The AI retrieves relevant sections from the professor's slides and class notes, providing a precise answer along with references to the original material.

❖ Scenario 2: Homework Help

💡 **Student Question:** "How do I solve this problem from last week's assignment?"

❖ **Response:** The system finds the related example from lecture slides and presents a structured explanation.

💡 Scenario 3: Exam Preparation

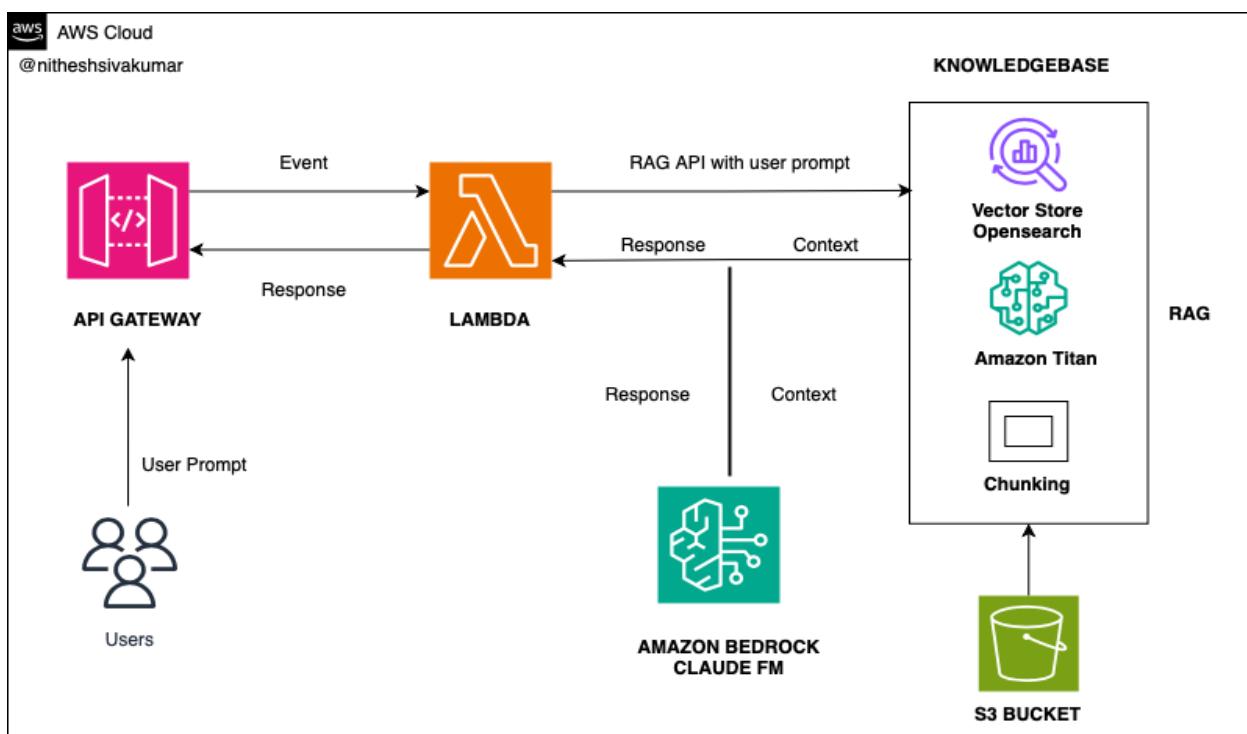
💡 *Student Question: "What are the key topics covered in last semester's cybersecurity midterm?"*

❖ *Response: The AI summarizes key points based on past exam materials.*

Testing the System

*"To ensure accuracy, I tested the system using **Postman** by making API requests. Queries related to different topics successfully returned detailed answers, including references to the original documents. The AI also provides **source citations**, so students know exactly where the answer is coming from."*

Architecture:



Architecture Overview

1. **S3 Bucket as a Data Source**
 - A dedicated Amazon S3 bucket is created to store PDF documents related to AWS services, including FAQs on EC2, EBS, and ECS.
2. **Knowledge Base for Retrieval-Augmented Generation (RAG)**
 - A Knowledge Base in AWS Bedrock is configured to process and retrieve relevant information from stored documents.
 - It handles document chunking, vector embedding, and storage in a vector database.
3. **API Gateway for Client Requests**
 - Amazon API Gateway is deployed as the front-facing endpoint for the application, allowing users to send queries.
4. **AWS Lambda for Processing Requests**
 - A Lambda function is implemented to receive user queries from API Gateway.
 - The function invokes the **Retrieve and Generate API** of the Knowledge Base to fetch contextual information.
 - The retrieved context is sent to a Bedrock Foundation Model, which generates a relevant response.
5. **Response Generation & Delivery**
 - The generated answer is returned to AWS Lambda.
 - The Lambda function sends the final response to the user through API Gateway.
 - The system also provides **citations**, showing the source of the retrieved answer.

Workflow Summary

1. A **user** asks a question (e.g., “Which EBS volume is best for high throughput?”).
2. The query is sent to **API Gateway**, which triggers the **Lambda function**.
3. **Lambda** invokes the **Retrieve and Generate API** from the **Knowledge Base**, retrieving relevant data from the S3-stored documents.
4. The **Bedrock Foundation Model** generates an AI-powered response using the retrieved context.
5. The response, along with **source citations**, is returned to the user via **API Gateway**.

Step-by-Step Implementation Guide

You cannot create **AWS Bedrock** resources if you are logged in as the **root user**. To proceed, sign in with an **IAM user** that has the necessary permissions.

Step 1: Create an IAM User

1. Navigate to the AWS Management Console and go to the IAM (Identity and Access Management) service.
2. Click on “Users” in the left sidebar.
3. Select “Create User” and enter a username of your choice.
4. Configure the required permissions:
 - o For admin access, attach the AdministratorAccess policy.
5. Enable AWS Management Console access and set a password for the user.
6. Click “Create User” to finalize the setup.

The screenshot shows the 'Create user' wizard in the AWS IAM console. The title bar says 'Specify user details'. On the left, a sidebar shows steps: Step 1 (selected), Step 2 (Set permissions), and Step 3 (Review and create). The main area has a 'User details' section with a 'User name' field containing 'nithesh-sivakumar'. Below it is a note: 'The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = . @ _ - (hyphen)' and a checkbox for 'Provide user access to the AWS Management Console - optional'. A callout box at the bottom left says: 'If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keypairs, you can generate them after you create this IAM user.' A 'Next' button is at the bottom right.

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

- Add user to group Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1240)

Choose one or more policies to attach to your new user.

Policy name	Type	Attached entities
<input type="checkbox"/> AccessAnalyzerServiceRolePolicy	AWS managed	0
<input checked="" type="checkbox"/> AdministratorAccess	AWS managed - job function	4
<input type="checkbox"/> AdministratorAccess-Amplify	AWS managed	0
<input type="checkbox"/> AdministratorAccess-AWSElasti...	AWS managed	0

Step 2: Log in as an IAM User

1. Open the AWS Management Console login page.
2. Instead of using the root credentials, select Sign in as an IAM user.
3. Enter your AWS Account ID or alias and click Next.
4. Provide your IAM username and password created in the previous step.
5. Click "Sign In" to access the AWS console with IAM privileges.
6. Select us-west-2 region.

Console Home

Recently visited

- Elastic Kubernetes Service
- EC2
- CloudFormation
- RDS
- EFS
- Support
- Billing and Cost Management
- IAM

View all services

Welcome to AWS

Getting started with AWS

Learn the fundamentals and find valuable information to get the most out of AWS.

Application

Region: US West

Name	Region
US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

Cost and usage

Current month costs: \$0.00

Forecasted monthly costs: \$0.00

AWS Health

Open issues: 0

Past 7 days

Scheduled changes: 0

CloudShell **Feedback**

Creating an S3 Bucket

To store the PDFs that will serve as the knowledge base for the application, follow these steps:

1. In the AWS Management Console, navigate to the S3 service.
2. Click on "Create bucket."
3. Enter a bucket name. The name must be globally unique across all AWS accounts.
4. Choose the AWS region where you want to create the bucket.
5. Configure additional settings such as object ownership and public access settings based on your requirements.
6. Click on "Create bucket" to complete the process.

This bucket will serve as the **source storage** for all PDFs used in the knowledge base. Once created, you can upload the required documents to this bucket.

The screenshot shows the 'Create bucket' page in the AWS Management Console. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, a search bar, and user information ('Oregon' and 'nithesh-sivakumar @ 7541-7327-2413'). Below the navigation is a breadcrumb trail: 'Amazon S3 > Buckets > Create bucket'. The main content area has a title 'Create bucket' with a 'Info' link. A sub-section titled 'General configuration' contains fields for 'Bucket name' (set to 'nitheshskmr-bedrock-bucket') and 'AWS Region' (set to 'US West (Oregon) us-west-2'). It also includes a 'Bucket type' section with two options: 'General purpose' (selected) and 'Directory'. The 'General purpose' option is described as recommended for most use cases and access patterns, noting it uses the original S3 bucket type with redundant storage across multiple Availability Zones. The 'Directory' option is described as recommended for low-latency use cases, using the S3 Express One Zone storage class for faster processing within a single Availability Zone. Below this is a 'Copy settings from existing bucket - optional' section with a 'Choose bucket' button and a note about copied settings. At the bottom of the configuration section is an 'Object Ownership' link with a note about controlling object access from other accounts via ACLs. The footer of the page includes links for 'CloudShell', 'Feedback', and copyright information ('© 2024, Amazon Web Services, Inc. or its affiliates.'), along with 'Privacy', 'Terms', and 'Cookie preferences' links.

Uploading PDFs to the S3 Bucket

To populate the knowledge base, upload the relevant PDFs and documents to the S3 bucket. Since professor materials are not available, general PDFs have been used for this implementation.

The screenshot shows the AWS S3 'Upload' interface. At the top, it displays the navigation path: Amazon S3 > Buckets > nithehskmr-bedrock-bucket > Upload. Below this, there's a large central area for uploading files, with a placeholder text 'Drag and drop files and folders you want to upload here, or choose Add files or Add folder.' A 'Files and folders' table lists 7 items totaling 5.1 MB. The table includes columns for Name, Folder, and Type, with all entries being application/pdf files. Below the file list is a 'Destination' section. The bottom of the screen shows standard AWS footer links: CloudShell, Feedback, © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

Creating a Knowledge Base in AWS Bedrock

To enable the retrieval of information from the uploaded PDFs, a knowledge base needs to be created in AWS Bedrock. Follow these steps:

1. In the AWS Management Console, navigate to **Amazon Bedrock**.
2. Click on "**Knowledge Bases**" from the left menu.
3. Select "**Create Knowledge Base**" to start the setup.

The screenshot shows the AWS Bedrock 'Knowledge bases' interface. On the left, a sidebar contains sections like 'Getting started', 'Foundation models', 'Playgrounds', 'Builder tools', 'Safeguards', and 'Inference'. The main area is titled 'Knowledge bases' and includes sections for 'How it works' (with 'Upload and chat', 'Create a knowledge base', 'Test the knowledge base', and 'Use the knowledge base' options), and a 'Knowledge bases' table. The table has columns for Name, Status, Description, Source files, Creation time, Last sync w..., and Last sync. It shows a single entry: 'No knowledge base'. At the bottom, there's a 'Create knowledge base' button. The bottom of the screen shows standard AWS footer links: CloudShell, Feedback, © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

4. Enter a name for the knowledge base.

Configuring IAM Permissions for the Knowledge Base

When setting up the Knowledge Base in Amazon Bedrock, you need to grant it access to your S3 bucket and other AWS services.

1. In the **IAM Permissions** section, select "**Create new service role**".
2. This role allows Bedrock to read documents from S3, process them, and interact with other AWS services securely.
3. Choosing the auto-generated role simplifies setup and ensures the correct permissions are applied.

This ensures Bedrock can retrieve and process knowledge base data securely.

The screenshot shows the 'Create knowledge base' wizard in the Amazon Bedrock console. The left sidebar contains navigation links for Getting started, Foundation models, Playgrounds, Builder tools, Safeguards, and Inference. The main area shows the 'Provide knowledge base details' step, which includes fields for 'Knowledge base name' (set to 'nitheshskmr-knowledge-base') and 'Knowledge base description - optional'. Below this is the 'IAM permissions' section, which notes that certain permissions are necessary to access other services or perform actions in order to create this resource. It offers two options: 'Create and use a new service role' (selected) and 'Use an existing service role'. A service role name 'AmazonBedrockExecutionRoleForKnowledgeBase_maiq8' is specified. At the bottom, there is a 'Choose data source' section and a footer with links to CloudShell, Feedback, and various AWS terms and conditions.

Under Data Source, select Amazon S3.

The screenshot shows the AWS Amazon Bedrock console. On the left, there's a sidebar with various sections like 'Getting started', 'Foundation models', 'Playgrounds', 'Builder tools', 'Safeguards', 'Inference', and 'Provisioned Throughput'. The 'Amazon S3' option under 'Choose data source' is selected, highlighted with a blue border. Other options shown include 'Web Crawler - Preview', 'Confluence - Preview', 'Salesforce - Preview', and 'Sharepoint - Preview'. Below the data sources, there's a 'Tags' section with an 'Add new tag' button. At the bottom, there are links for CloudShell, Feedback, and navigation icons.

Specify the S3 bucket where the PDFs are stored.

The screenshot shows the 'Data source: knowledge-base-quick-start-ghcdd-data-source' configuration page. It includes fields for 'Data source name' (set to 'knowledge-base-quick-start-ghcdd-data-source'), 'Data source location' (set to 'This AWS account'), 'S3 URI' (set to 's3://nitheshskmr-bedrock-bucket'), 'Add customer-managed KMS key for S3 data - optional' (unchecked), 'Chunking and parsing configurations' (set to 'Default'), and an 'Advanced settings - optional' section. There's also an 'Add data source' button and a note about adding up to 4 more data sources. Navigation buttons for 'Cancel', 'Previous', and 'Next' are at the bottom.

Click next, then select **Titan Embeddings** as the embedding model.

An embedding model converts text into numerical representations, allowing efficient searching and retrieval of relevant information. It helps the knowledge base understand and match user queries with the most relevant content from the uploaded documents.

The screenshot shows the 'Amazon Bedrock' service in the AWS console. The left sidebar has a navigation menu with sections like 'Getting started', 'Foundation models', 'Playgrounds', 'Builder tools', 'Safeguards', 'Inference', and 'CloudShell'. The main content area is titled 'Select embeddings model and configure vector store'. It shows four options for 'Embeddings model': 'Titan Text Embeddings v2' (By Amazon), 'Titan Embeddings G1 - Text v1.2' (By Amazon), 'Embed English v3' (By Cohere), and 'Embed Multilingual v3' (By Cohere). Below this is a 'Vector dimensions' section with a dropdown set to '1536'. The bottom section is titled 'Vector database' with a note about letting Amazon create a vector store or selecting a previously created store. At the bottom right are links for 'CloudShell', 'Feedback', and copyright information.

For the vector database, click **Create new vector database** to store and index the processed document embeddings for efficient retrieval. An AWS OpenSearch DB will be created.

The screenshot shows the 'Amazon Bedrock' service in the AWS console. The left sidebar has a navigation menu with sections like 'Getting started', 'Foundation models', 'Playgrounds', 'Builder tools', 'Safeguards', 'Inference', and 'CloudShell'. The main content area is titled 'Configure vector store settings'. It shows two options for 'Vector dimensions': 'Embed English v3' (By Cohere) and 'Embed Multilingual v3' (By Cohere). Below this is a 'Vector database' section with a note about letting Amazon create a vector store or selecting a previously created store. At the bottom, there are three configuration options: 'Quick create a new vector store - Recommended' (selected), 'Choose a vector store you have created' (with a note about selecting Amazon OpenSearch Serverless, Amazon Aurora, MongoDB Atlas, Pinecone or Redis Enterprise Cloud and providing field mappings), and 'Enable redundancy (active replicas) - optional' (with a note about enabling this for development workloads). At the very bottom are 'Cancel', 'Previous', and 'Next' buttons, along with copyright information.

Click next, review all the settings, and then click **Create Knowledge Base**. The creation process will take approximately 4-5 minutes to complete.

You need to sync your data to make it available for retrieval. Click on **Go to Data Source**, then select **Sync** to process the uploaded documents.

You can now see the **data source name** listed under the **Data Source** section, which is linked to the S3 bucket where the documents are stored.

The screenshot shows the Amazon Bedrock service in the AWS Management Console. A green success message at the top states: "Knowledge base 'nitheshskmr-knowledge-base' is 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." Below this, the "Data source (1)" section shows a single entry: "knowledge-base-quick-start-ghcdd-data-source" (Status: Available). To the right, the "Test knowledge base" panel shows a "Select model" button and a note that "One or more data sources have not been synced." A "Configure your retrieval and responses" section is also present. At the bottom, there's a "Please select a model" dropdown and a "Run" button.

This screenshot is identical to the one above, showing the same successful knowledge base creation message and the "knowledge-base-quick-start-ghcdd-data-source" entry in the Data source list. The "Test knowledge base" panel also displays the same status and configuration options. The bottom section remains the same with the "Please select a model" dropdown and "Run" button.

Meanwhile, you need access to **Amazon Titan Embeddings G1 - Text**. If you don't have access, request it through the **Service Quotas** or **Bedrock Access Request** page in the AWS Management Console.

Amazon Titan Embeddings G1 - Text is used for efficient text retrieval by converting documents into searchable vectors. It ensures accurate, scalable, and AWS-optimized search results.

S | Services | Search | [Option+S] | Oregon | nithesh-sivakumar @ 7541-7327-2413 |

Amazon Bedrock > Model access > Request model access

Step 1
Edit model access
Step 2
Review and submit

Edit model access

Base models (1/36)

Not seeing a model you're interested in? Check out all supported models by region [here](#).

Find model | Group by provider

Models	Access status	Modality	EULA
Titan Embeddings G1 - Text	Available to request	Embedding	EULA
Titan Text G1 - Lite	Available to request	Text	EULA
Titan Text G1 - Express	Available to request	Text	EULA
Titan Image Generator G1 v2	Available to request	Image	EULA
Titan Image Generator G1	Available to request	Image	EULA
Titan Multimodal Embeddings G1	Available to request	Embedding	EULA
Titan Text Embeddings V2	Available to request	Embedding	EULA
Titan Text G1 - Text	Available to request	Text	EULA
Claude 3.5 Sonnet	Available to request	Text & Vision	EULA
Claude 3 Opus	Available to request	Text & Vision	EULA

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Click Submit to finalize the request.

S | Services | Search | [Option+S] | Oregon | nithesh-sivakumar @ 7541-7327-2413 |

Amazon Bedrock > Model access > Request model access

Step 1
Edit model access
Step 2
Review and submit

Review and submit

Step 1: Edit model access

Edit

Model access modifications (1)

Models	Modifications
Titan Embeddings G1 - Text	Request access

Terms
By selecting Submit, you are requesting access to the selected third party models through the AWS Marketplace. By doing so, you agree to the seller's pricing terms and End User License Agreements (EULA), and the [Bedrock Service Terms](#). You also agree and acknowledge that AWS may share information about this transaction with the respective sellers, in accordance with the [AWS Privacy Notice](#).

AWS will issue invoices and collect payments from you on behalf of the seller through your AWS account. Your use of AWS services is subject to the [AWS Customer Agreement](#) or other agreements with AWS governing your use of such services.

Cancel Previous Submit

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

You can see that access has been granted.

The screenshot shows the Amazon Bedrock Model access page. A blue banner at the top states "Model access updates submitted" and "1/7 access granted". Below this, there's a section titled "What is Model access?" with a note about IAM permissions. A "Modify model access" button is present. The main area is titled "Base models (36)" and shows a table of models. One row for "Titan Embeddings G1 - Text" is highlighted with "Access granted" status. Other rows include "Titan Text G1 - Lite", "Titan Text G1 - Express", and "Titan Image Generator G1 v2", all marked as "Available to request". The table includes columns for "Models", "Access status", "Modality", and "EULA".

Next, you need to select the foundation model. Click **Select Model** to proceed.

A foundation model processes user queries and generates responses based on retrieved knowledge. It enhances accuracy by leveraging AI to understand and summarize relevant information.

The screenshot shows the "Test knowledge base" configuration page. A green success message says "Sync completed for data source - 'knowledge-base-quick-start-ghcdd-data-source'". There are sections for "Log Deliveries" (Configure log deliveries and event logs), "Tags" (A tag is a label assigned to an AWS resource), and "Data source (1)". The data source table shows one entry: "knowledge... Available S3 75417327... s3://n". On the right, there's a "Select model" button and a "Configure your retrieval and responses" section with a note about customizing search strategy. At the bottom, there's a "Please select a model" field and a "Run" button.

Currently, only **Anthropic** and **Meta models** are available for access via the GUI. For testing purposes, we can use one of these models to check if it retrieves information from our S3 bucket.

I will select **Claude**, then click **Apply**.

The screenshot shows the 'Select model' dialog in the AWS Bedrock interface. On the left, a sidebar lists various tools and services. The main area is divided into three sections: 'Category' (Model providers: Anthropic, Meta), 'Model' (Models with access: Claude Instant v1.2, Claude v2.1, Claude 3 Sonnet v1, Claude 3 Haiku v1, Claude 3.5 Sonnet v1), and 'Throughput' (On-demand). A note at the bottom says 'Not seeing a model you are interested in? Check out all supported models here.' At the bottom right are 'Cancel' and 'Apply' buttons.

I am asking a question related to **EBS**, which was included in the uploaded PDFs, to test if the model retrieves the correct information from the knowledge base.

The screenshot shows the 'Test knowledge base' dialog for the 'nitheshskmr-knowledge-base'. It displays the 'Knowledge base overview' (Knowledge base name: nitheshskmr-knowledge-base, Knowledge base ID: OBDOQYLCYL, Status: Ready, Service Role: AmazonBedrockExecutionRoleForKnowledgeBase_maiq8), 'Log Deliveries' (Configure log deliveries and event logs in the Edit page), and 'Tags' (A brief description of tags). On the right, there's a section titled 'Configure your retrieval and responses' with a note about customizing search strategy. At the bottom, a question is asked: 'What types of EBS volumes are available?' with a 'Run' button.

Screenshot of the AWS Cloud interface showing the Amazon Bedrock Knowledge Base configuration page for 'nitheshskmr-knowledge-base'.

The left sidebar contains navigation links for Text, Image, Builder tools (Prompt management, Knowledge bases, Agents, Prompt flows), Safeguards (Guardrails, Watermark detection), Inference (Provisioned Throughput, Batch inference, Cross-region inference), Assessment (Model Evaluation), and Bedrock configurations (Model access, Bedrock Studio, Settings).

The main content area displays the 'Knowledge base overview' for the specified knowledge base. It includes:

- Knowledge base name: nitheshskmr-knowledge-base
- Knowledge base ID: OBDOQYLCYL
- Status: Ready
- Created date: September 19, 2024, 19:01 (UTC-05:00)
- Service Role: AmazonBedrockExecutionRoleForKnowledgeBase_maiq8
- Log Deliveries: Configure log deliveries and event logs in the Edit page.

A 'Tags' section explains what tags are and provides a table for managing them. A 'Data sources (1)' section is also present.

The right side features a 'Claude Instant' AI interface for generating responses. It shows a configuration panel for retrieval and responses, a question about EBS volumes, and a detailed answer about EBS volume types. An input field and a 'Run' button are at the bottom.

Screenshot of the AWS Cloud interface showing the Amazon Bedrock Knowledge Base configuration page for 'nitheshskmr-knowledge-base'.

The left sidebar contains navigation links for Text, Image, Builder tools (Prompt management, Knowledge bases, Agents, Prompt flows), Safeguards (Guardrails, Watermark detection), Inference (Provisioned Throughput, Batch inference, Cross-region inference), Assessment (Model Evaluation), and Bedrock configurations (Model access, Bedrock Studio, Settings).

The main content area displays the 'Knowledge base overview' for the specified knowledge base. It includes:

- Knowledge base name: nitheshskmr-knowledge-base
- Knowledge base ID: OBDOQYLCYL
- Status: Ready
- Created date: September 19, 2024, 19:01 (UTC-05:00)
- Service Role: AmazonBedrockExecutionRoleForKnowledgeBase_maiq8
- Log Deliveries: Configure log deliveries and event logs in the Edit page.

A 'Tags' section explains what tags are and provides a table for managing them. A 'Data sources (1)' section is also present.

The right side features a 'Claude Instant' AI interface for generating responses. It shows a configuration panel for retrieval and responses, a question about EBS volumes, and a detailed answer about EBS volume types. An input field and a 'Run' button are at the bottom.

You can also verify the source of the retrieved information. Under **Source Details**, it shows the specific PDF from the S3 bucket where the answer was extracted.

Key	Value
x-amz-bedrock-kb-source-uri	s3://nitheshskmr-bedrock-buc
x-amz-bedrock-kb-chunk-id	1%3A0%3Ap0vKDJIBUSE2Lwk
x-amz-bedrock-kb-data-source-id	NOFLZASJP2

Setting Up the Retrieve and Generate API

Next, you need to create a Lambda function to handle API requests.

1. Go to **AWS Lambda** in the console.
2. Click **Create Function**.
3. Enter a **function name** (e.g., `RetrieveGenerateFunction`).
4. Select **Runtime** as **Python**.
5. Click **Create** to proceed.

Edit the Lambda function by going to **Configuration**. The default timeout is set to 3 seconds, but for this use case, increase it to 1 minute to prevent timeouts. Click **Save** to apply the changes.

The screenshot shows the 'Edit basic settings' page for a Lambda function. The 'Basic settings' tab is selected. Key configuration options visible include:

- Description - optional:** A text input field containing no text.
- Memory:** Set to 128 MB, with a range from 128 MB to 10240 MB.
- Ephemeral storage:** Set to 512 MB, with a range from 512 MB to 10240 MB.
- SnapStart:** Set to None.
- Timeout:** Set to 1 min 3 sec.
- Execution role:** Set to 'Use an existing role' (radio button selected).

At the bottom, there are links for CloudShell, Feedback, and various AWS footer links like Privacy, Terms, and Cookie preferences.

Next, grant the Lambda function permission to access Bedrock.

1. Go to Configuration → Permissions.
2. Click on the Execution Role, which will open the IAM dashboard.
3. Click on Add Permissions and search for BedrockFullAccess.
4. Select it and apply the changes.

The screenshot shows the AWS Identity and Access Management (IAM) service in the AWS Management Console. The left sidebar is collapsed, and the main area displays the 'nitheshskmr-bedrock-function-role-9jjhq9tw' role. A green banner at the top indicates that the 'Policy was successfully attached to role.' Below the banner, the 'Summary' section provides basic information about the role, including its creation date (September 19, 2024, 19:54 UTC-05:00), ARN (arn:aws:iam::754173272413:role/service-role/nitheshskmr-bedrock-function-role-9jjhq9tw), and last activity (none). The 'Permissions' tab is selected, showing two managed policies attached: 'AmazonBedrockFullAccess' (AWS managed, 3 entities) and 'AWSLambdaBasicExecutionRole-369...' (Customer managed, 1 entity). The 'Permissions policies' section lists these two policies with options to simulate, remove, or add more permissions.

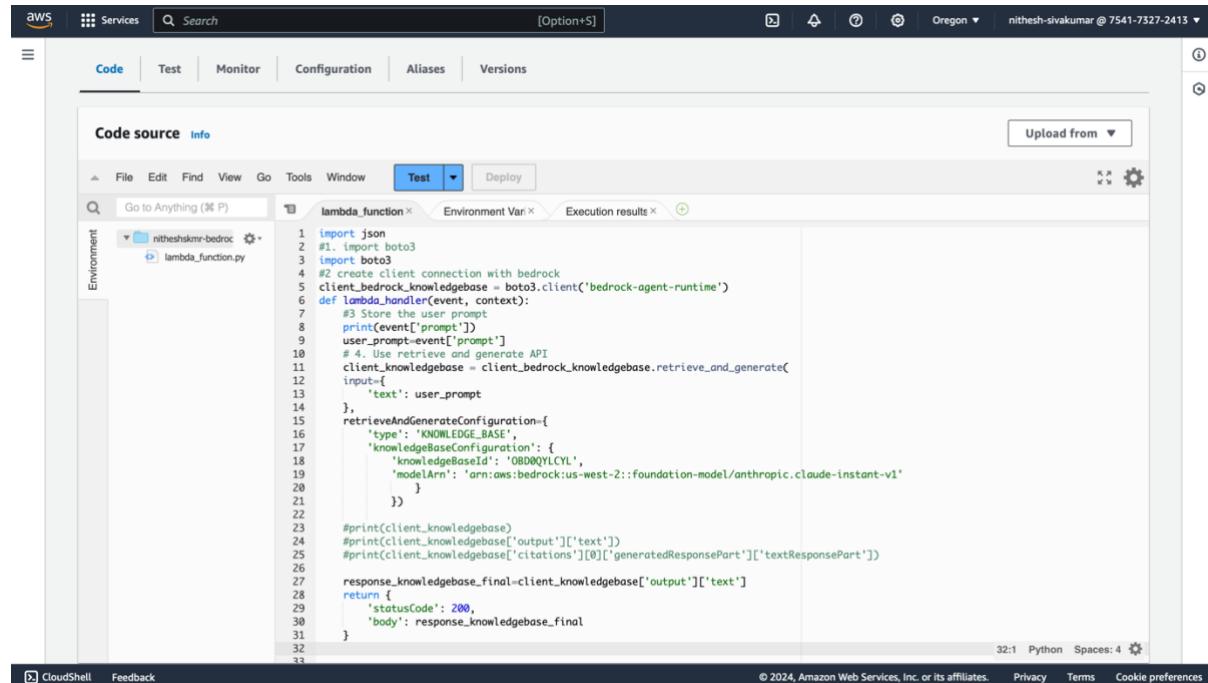
Go back to the **Lambda function**, then under the **Code** section, copy and paste the provided code.

```
import json
#1. import boto3
import boto3
#2 create client connection with bedrock
client_bedrock_knowledgebase = boto3.client('bedrock-agent-runtime')
def lambda_handler(event, context):
    #3 Store the user prompt
    print(event['prompt'])
    user_prompt=event['prompt']
    # 4. Use retrieve and generate API
    client_knowledgebase =
client_bedrock_knowledgebase.retrieve_and_generate(
    input={
        'text': user_prompt
    },
    retrieveAndGenerateConfiguration={
        'type': 'KNOWLEDGE_BASE',
        'knowledgeBaseConfiguration': {
            'knowledgeBaseId': 'O41RCIQ46A',
            'modelArn': 'arn:aws:bedrock:us-west-2::foundation-
model/anthropic.claude-instant-v1'
        }
    })
    # print(client_knowledgebase)
    #print(client_knowledgebase['output']['text'])

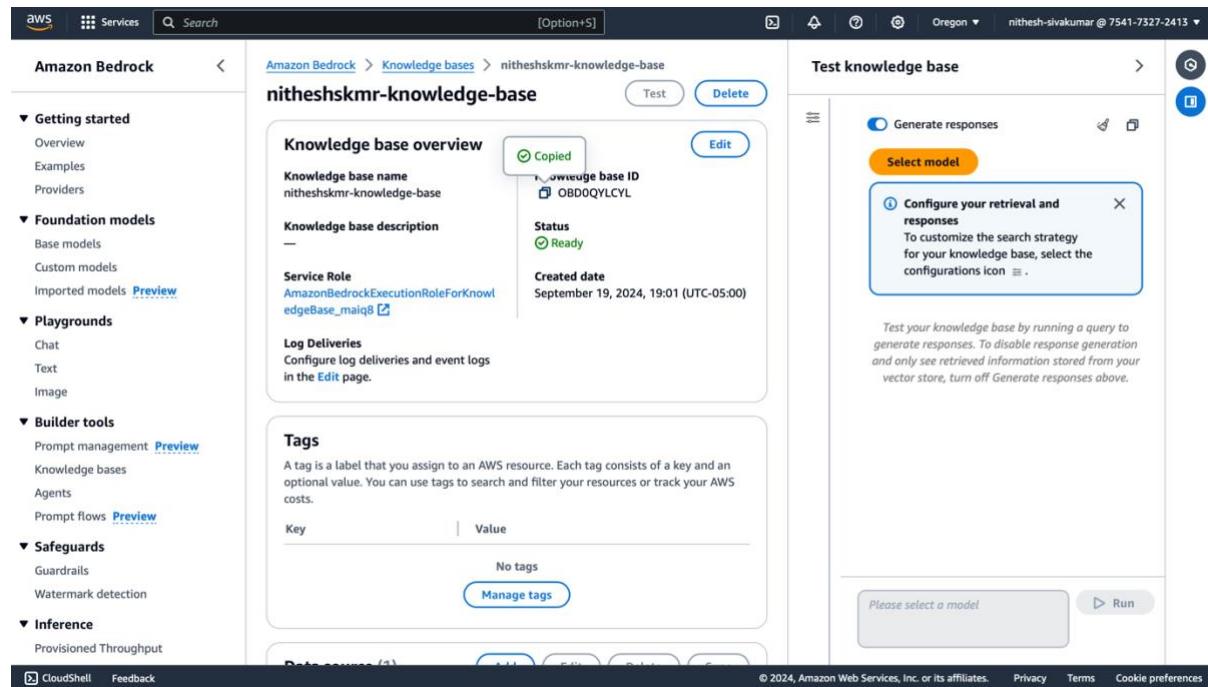
#print(client_knowledgebase['citations'][0]['generatedResponsePart']['te
xtResponsePart'])
    response_kbbase_final=client_knowledgebase['output']['text']
    return {
        'statusCode': 200,
        'body': response_kbbase_final
    }
```

In the code, you need to update the **Knowledge Base ID** to ensure it connects to the correct data source.

1. Go to **Amazon Bedrock**.
2. Navigate to your **Knowledge Base**.
3. Copy the **Knowledge Base ID**.
4. Paste it in the line 18 of your Lambda function code.
5. Deploy the updated code.



```
1 import json
2 #1. import boto3
3 import boto3
4 #2. create client connection with bedrock
5 client_bedrock_knowledgebase = boto3.client('bedrock-agent-runtime')
6 def lambda_handler(event, context):
7     #3. Store the user prompt
8     print(event['prompt'])
9     user_prompt=event['prompt']
10    # 4. Use retrieve and generate API
11    client_knowledgebase = client_bedrock_knowledgebase.retrieve_and_generate(
12        input={
13            'text': user_prompt
14        },
15        retrieveAndGenerateConfiguration={
16            'type': 'KNOWLEDGE_BASE',
17            'knowledgeBaseConfiguration': {
18                'knowledgeBaseId': 'OBDOQQLCYL',
19                'modelArn': 'arn:aws:bedrock:us-west-2:foundation-model/anthropic.claude-instant-v1'
20            }
21        }
22    )
23    #print(client_knowledgebase)
24    #print(client_knowledgebase['output']['text'])
25    #print(client_knowledgebase['citations'][0]['generatedResponsePart']['textResponsePart'])
26
27    response_knowledgebase_final=client_knowledgebase['output']['text']
28    return {
29        'statusCode': 200,
30        'body': response_knowledgebase_final
31    }
32 }
```



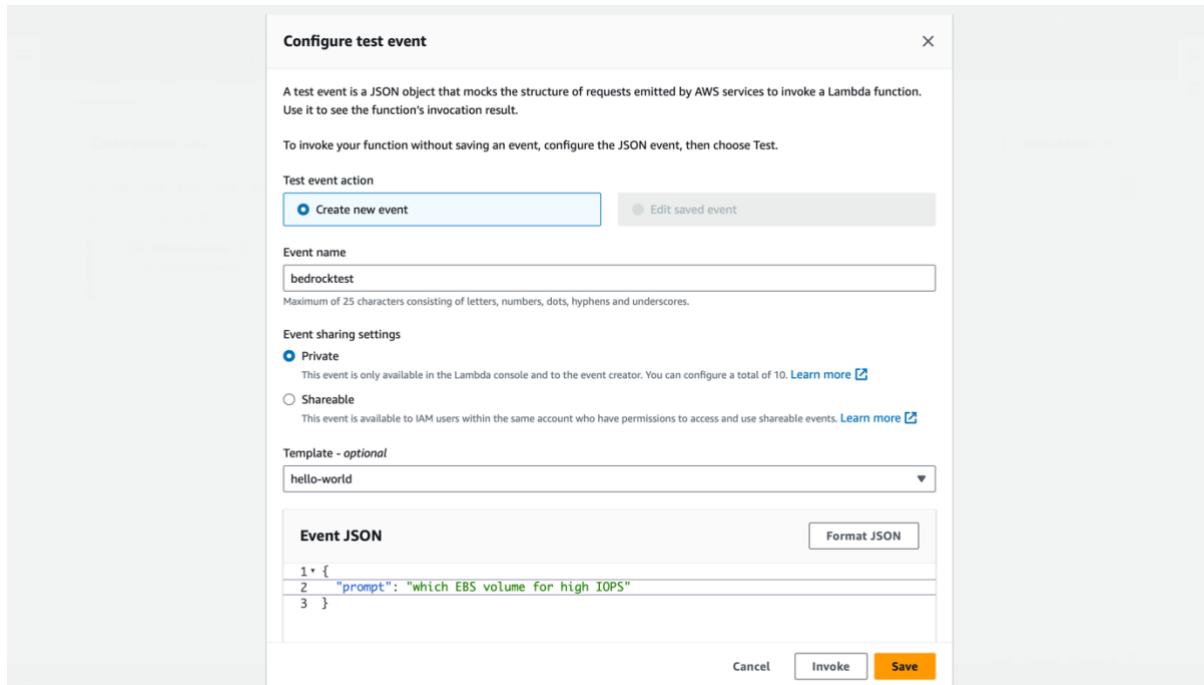
The screenshot shows the Amazon Bedrock console interface. On the left, there's a navigation sidebar with sections like 'Getting started', 'Foundation models', 'Playgrounds', 'Builder tools', 'Safeguards', and 'Inference'. The main area displays a 'Knowledge base overview' for 'nitheshskmr-knowledge-base'. It includes fields for 'Knowledge base name' (copied), 'Knowledge base description' (empty), 'Service Role' (AmazonBedrockExecutionRoleForKnowledgeBase_maiq8), and 'Created date' (September 19, 2024, 19:01 UTC-05:00). Below this is a 'Log Deliveries' section. At the bottom, there's a 'Tags' section with a note about tags and a 'Manage tags' button. To the right, a 'Test knowledge base' panel has a 'Generate responses' toggle turned on, a 'Select model' button, and a tooltip for 'Configure your retrieval and responses'. A message at the bottom says '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.' At the very bottom, there's a 'Please select a model' dropdown and a 'Run' button.

Click **Deploy**, then create and run a **Test Event** to verify the function's execution.

Create a test event with the following details:

- **Test Name:** bedrocktest
- **Prompt:** *Which EBS volume is best for high IOPS?*

Run the test to check if the Lambda function retrieves the correct response from the knowledge base.



You can also view the log events in Amazon CloudWatch to monitor the execution details and debug any issues.

The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar has sections for Favorites and recent items, Alarms, Logs (selected), Log groups, Metrics, X-Ray traces, Events, Application Signals, Network monitoring, Insights, Settings, Getting Started, and What's new. The main area shows a list of log events for the path /aws/lambda/nitheshskmr-bedrock-function. The first event is timestamped 2024-09-20T01:27:01.653Z and contains the message "INIT_START Runtime Version: python:3.12.v30 Runtime Version ARN: arn:aws:lambda:us-west-2::runtime:ac6...". Subsequent events show the Lambda starting, provisioning IOPS SSD volumes, and reporting the request ID and duration. A message at the bottom indicates "No newer events at this moment. Auto retry paused. Resume".

We can see a **200 status code**, indicating a successful request, along with the response from the knowledge base.

The screenshot shows the AWS Lambda Test feature. The top navigation bar includes services, search, and options. Below it, tabs for Code, Test, Monitor, Configuration, Aliases, and Versions are present, with Code selected. The main area displays the "Code source" tab for a function named lambda_function. It shows the Python file lambda_function.py with the following code:

```
def lambda_handler(event, context):
    response = {
        "statusCode": 200,
        "body": "Provisioned IOPS SSD (io1 and io2) volumes are designed for transactional, IOPS-intensive database workloads and require applications that need high random I/O performance. These volume types provision a set level of random IOPS that remains consistent."
    }
    return response
```

The "Test" tab is active, showing the "Execution results" section with a status of "Succeeded", maximum memory used of 74 MB, and a duration of 3001.09 ms. The "Function Logs" section shows the Lambda starting, provisioning IOPS SSD volumes, and reporting the request ID and duration. The "Request ID" is listed as 723f5c45-4cf4-4a05-8347-09643e5e7935.

Creating an API Gateway

To expose the Lambda function as an endpoint, create an API Gateway.

1. Search for **API Gateway** in the AWS console.
2. Click **Create API** and select **REST API**.

Why use API Gateway?

- Acts as an **entry point** for client requests.
- Handles **authentication, authorization, and rate limiting**.
- Allows secure and scalable communication between the frontend and backend services.
- Integrates seamlessly with **AWS Lambda**, enabling a **serverless API**.

The screenshot shows the AWS Services Catalog interface. At the top, there's a search bar and a 'Build' button. Below the search bar, there are three main service cards:

- WebSocket API**: Described as "Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards." It works with Lambda, HTTP, and AWS Services. It includes a "Build" button.
- REST API**: Described as "Develop a REST API where you gain complete control over the request and response along with API management capabilities." It works with Lambda, HTTP, and AWS Services. It includes "Import" and "Build" buttons.
- REST API Private**: Described as "Create a REST API that is only accessible from within a VPC." It works with Lambda, HTTP, and AWS Services. It includes "Import" and "Build" buttons.

At the bottom of the screen, there are links for CloudShell, Feedback, and various AWS terms like Privacy, Terms, and Cookie preferences.

Give the API a name, such as **nitheshskmr-knowledgebase**, and click **Create** to proceed.

The screenshot shows the 'Create REST API' wizard in the AWS Management Console. The 'API details' step is selected, showing four options: 'New API' (selected), 'Clone existing API', 'Import API', and 'Example API'. The 'API name' field contains 'nitheshskmr-knowledgebase'. The 'Description - optional' field is empty. Under 'API endpoint type', the dropdown is set to 'Regional'. At the bottom right are 'Cancel' and 'Create API' buttons. The status bar at the bottom includes links for CloudShell, Feedback, and copyright information.

Creating a method in API Gateway defines how the API handles incoming requests. It specifies the **method** (GET, POST, etc.) and determines how requests are processed, including integration with AWS Lambda to retrieve responses from the knowledge base.

Go to the API you created. Now, create a method by clicking on **Create Method**.

The screenshot shows the 'Resources' page for the previously created API. The left sidebar shows the API structure: 'APIs', 'Custom domain names', 'VPC links', 'API: nitheshskmr-knowledgebase' (expanded), 'Resources' (selected), 'Stages', 'Authorizers', 'Gateway responses', 'Models', 'Resource policy', 'Documentation', 'Dashboard', and 'API settings'. The main area displays a single resource path '/'. On the right, the 'Resource details' section shows the path '/' and resource ID 'glt5qjfr3e'. The 'Methods (0)' section has a 'Create method' button. The status bar at the bottom includes links for CloudShell, Feedback, and copyright information.

Select **GET** as the method type and choose **Lambda Function** as the integration type. This allows API Gateway to forward GET requests to the Lambda function, which will process queries and return responses from the knowledge base.

The screenshot shows the 'Create method' page in the AWS API Gateway console. In the 'Method details' section, the 'Method type' is set to 'GET'. Under 'Integration type', the 'Lambda function' option is selected, indicated by a blue background and a checked radio button. Other options like 'HTTP', 'Mock', 'AWS service', and 'VPC link' are shown with their respective icons and descriptions. Below the integration type section, there is a note about Lambda proxy integration and a field for providing a Lambda function name or ARN. At the bottom right of the page, there are links for CloudShell, Feedback, and a copyright notice for 2024 Amazon Web Services, Inc.

Select the **Lambda function** you created earlier, then click **Create Method** to complete the setup.

The screenshot shows the 'Create method' configuration page. In the 'Lambda proxy integration' section, the note 'Send the request to your Lambda function as a structured event.' is present. In the 'Lambda function' section, the 'us-west-2' region is selected, and the ARN '754173272413:function:nitheshskmr-bedrock-function' is entered into the input field. A tooltip provides information about granting API Gateway permission to invoke the Lambda function. Below these sections are expandable sections for 'Method request settings', 'URL query string parameters', 'HTTP request headers', and 'Request body'. At the bottom right, there are 'Cancel' and 'Create method' buttons, with the 'Create method' button being orange.

Configuring the Method Request

1. Select **Method Request** under the API method.
2. Click **Edit** to modify the settings.

The screenshot shows the AWS API Gateway interface. On the left, the navigation pane is open, showing sections like APIs, Stages, and Resources. Under Resources, a GET method for the root path '/' is selected. The main panel displays the details for this method. At the top right, there are 'API actions' and a 'Deploy API' button. Below that, the ARN is listed as arn:aws:execute-api:us-west-2:754173272413:c42pnovvw9/*/GET/. To the right, the Resource ID is glt5qjfr3e. A flow diagram illustrates the request process: Client → Method request → Integration request → Lambda integration. Below the flow diagram, tabs for Method request, Integration request, Integration response, Method response, and Test are shown, with Method request selected. Under Method request settings, it shows Authorization set to NONE and API key required set to False. At the bottom of the screen, standard AWS footer links are visible.

Steps to update settings:

- Enable **Request Validator** to ensure input validation.
- Click on **Request Validator** and select **Validate body, query string parameters, and headers**.
- Under **URL Query String Parameters**, click **Add Parameter** and enter the name as **prompt**. Click **Save**.

These changes ensure that API Gateway validates incoming requests and requires a query parameter for the prompt.

S | Services | Search | [Option+S] | Oregon | nithesh-sivakumar @ 7541-7327-2415 | ☰

API Gateway > APIs > Resources - nitreshskmr-knowledgebase (c42pnovvw9) > Edit method request

Edit method request

Method request settings

Authorization: None

Request validator: Validate body, query string parameters, and headers

API key required

Operation name - optional: GetPets

URL query string parameters

Name	Required	Caching	Remove
prompt	<input type="checkbox"/>	<input type="checkbox"/>	<button>Remove</button>

Add query string

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Next, select **Integration Request** and click **Edit** to modify its settings.

S | Services | Search | [Option+S] | Oregon | nithesh-sivakumar @ 7541-7327-2415 | ☰

API Gateway

APIs
Custom domain names
VPC links

▼ API: nitreshskmr-knowledgebase
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings

Usage plans
API keys
Client certificates
Settings

Create resource

/ GET

Method request | **Integration request** | Integration response | Method response | Test

Integration request settings

Integration type: Lambda
Region: us-west-2

Lambda proxy integration: False
Lambda function: nitreshskmr-bedrock-function

Input passthrough
When no template matches the request content-type header

Timeout: Default (29 seconds)

URL path parameters (0)

No URL path parameters defined

URL query string parameters (0)

No URL query string parameters defined

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Scroll down to **Mapping Templates**, then click **Add**.

- Set **Content Type** to application/json.
- Under **Template Body**, copy and paste the provided JSON template.

```
{  
  "prompt": "$input.params('prompt')"  
}
```

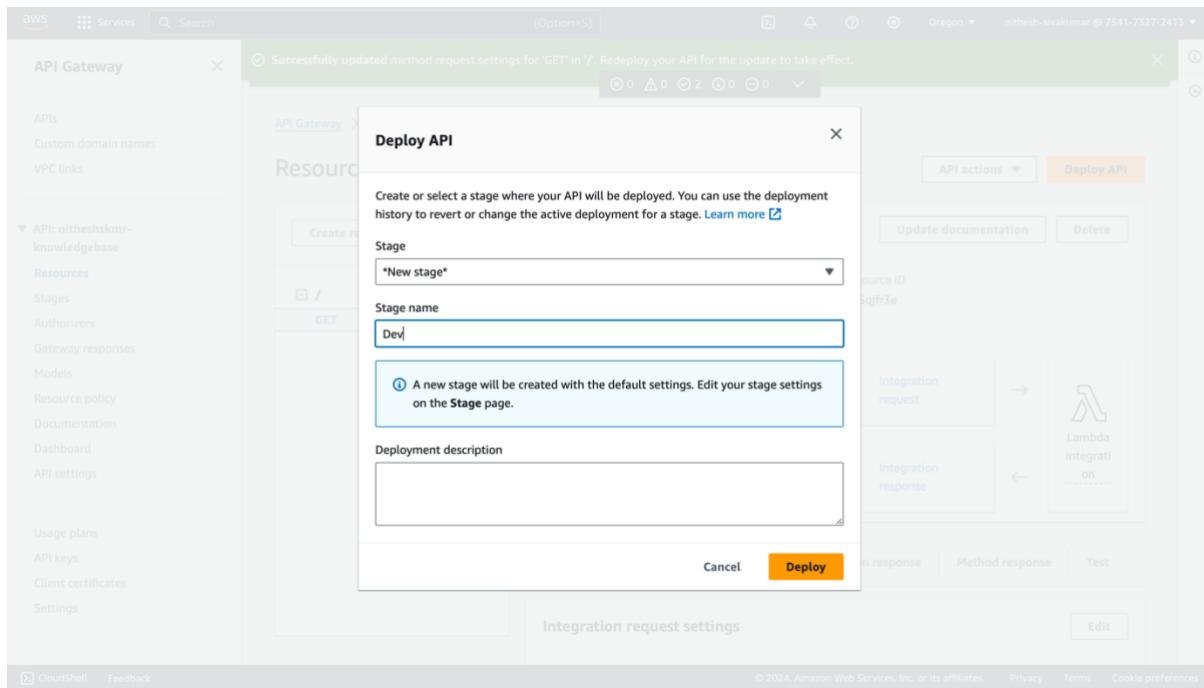
This ensures that incoming API requests are properly formatted before being sent to the Lambda function.

The screenshot shows the AWS Lambda function configuration interface. In the 'Mapping templates' section, the 'Content type' is set to 'application/json'. The 'Template body' field contains the following JSON template:

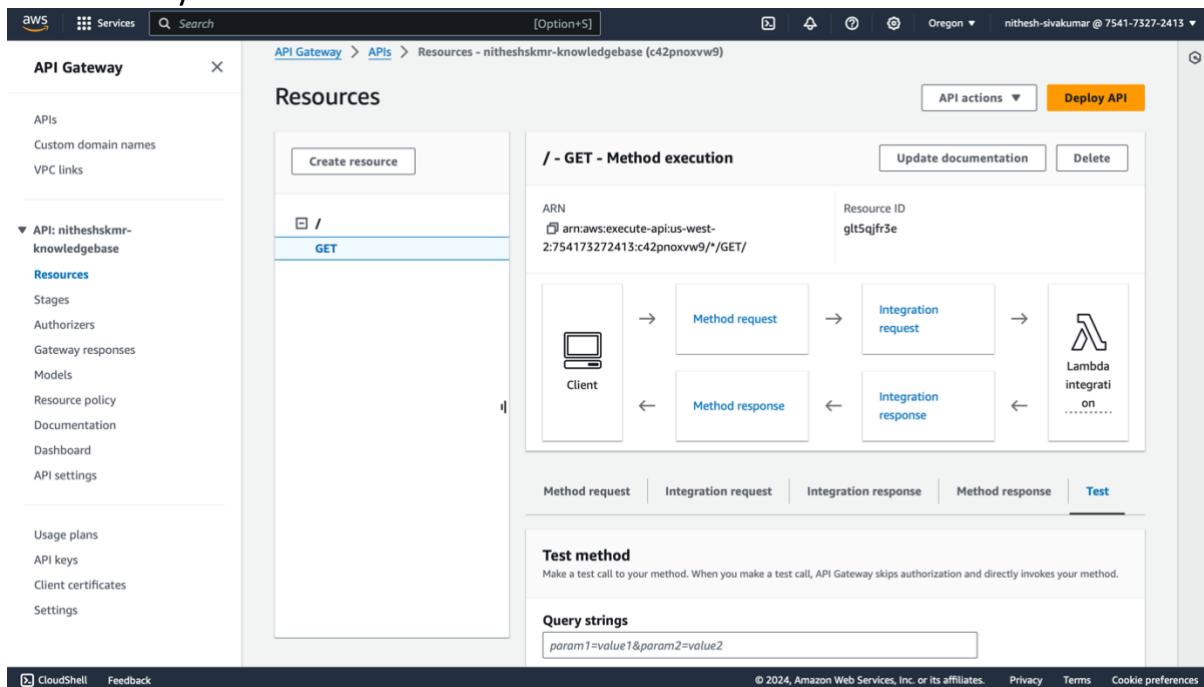
```
1 [ {  
2   "prompt": "$input.params('prompt')"  
3 } ]
```

Next, deploy the API:

1. Click **Deploy API**.
2. Select **New Stage**.
3. Enter **Stage Name** as dev.
4. Click **Deploy** to make the API live.



Next, select **Resources** from the left pane, then choose **Test** to verify the API's functionality.



Under **Query Strings**, enter the required parameters (e.g., prompt=Which is the best compute resource?), then click **Test** to execute the request and verify the response

The screenshot shows the AWS API Gateway Test interface. On the left, the navigation sidebar is visible with sections like APIs, Stages, Resources, and Usage plans. The main area shows a GET request configuration with a query string parameter 'prompt=Which is the best compute resource'. Below this, the 'Test method' section displays the results of the test. The results panel shows a status code of 200, a latency of 3638 ms, and a response body containing a JSON object describing GPU-based compute instances. The response body is as follows:

```
{"statusCode": 200, "body": "GPU-based compute instances like P3 and P4 instances are generally considered the best compute resource for applications that can benefit from highly parallel processing on GPUs, such as deep learning, scientific computing, and graphics workloads."}
```

This screenshot is similar to the one above, but it includes a detailed view of the logs for the executed request. The logs show the execution log for the request, including the start of execution, the HTTP method, the resource path, the method request path, the method request query string, and the method request headers. The logs are as follows:

```
Execution log for request 4d7135b6-375d-4d94-a557-2982f4c8975d
Fri Sep 20 01:43:35 UTC 2024 : Starting execution for request: 4d7135b6-375d-4d94-a557-2982f4c8975d
Fri Sep 20 01:43:35 UTC 2024 : HTTP Method: GET, Resource Path: /
Fri Sep 20 01:43:35 UTC 2024 : Method request path: {}
Fri Sep 20 01:43:35 UTC 2024 : Method request query string: {prompt=Which is the best compute resource}
Fri Sep 20 01:43:35 UTC 2024 : Method request headers: {}
```

Success!

You can also test the API using **Postman**:

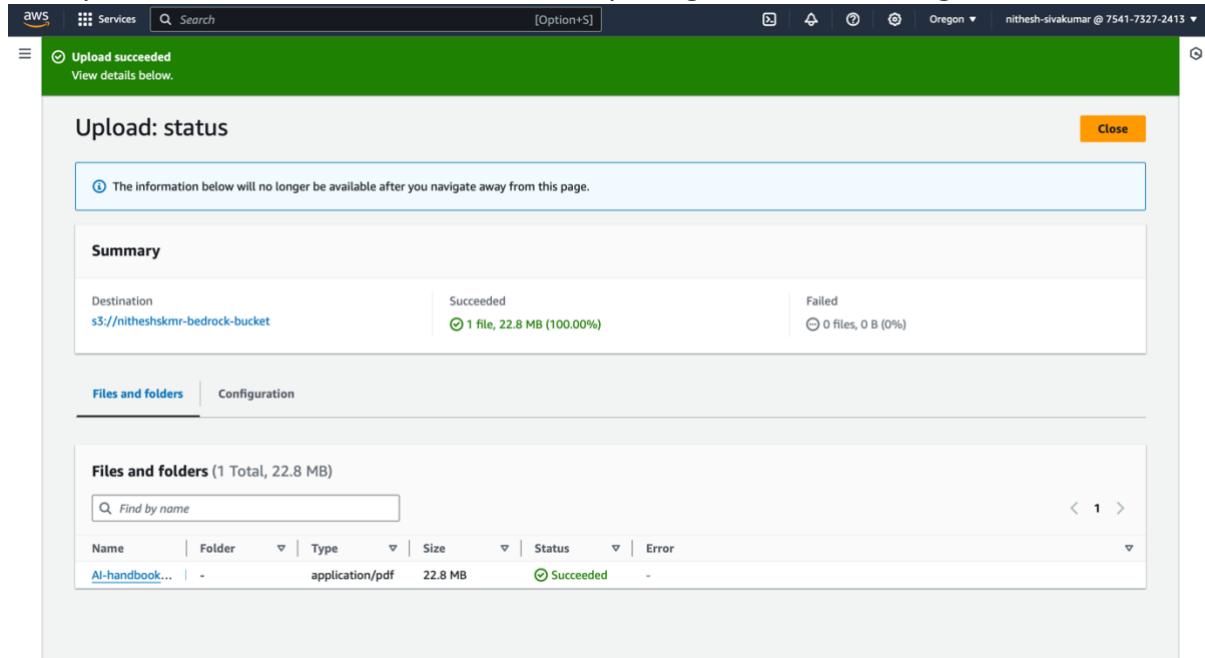
1. Copy the **API Invoke URL** from API Gateway.
2. Open **Postman**.
3. Paste the URL into the request field.
4. Set the **method** to **GET**.
5. Add the query parameter:
 - o Key: prompt
 - o Value: *Which EBS volume is best for high IOPS?*
6. Click **Send** to test the API response.

The screenshot shows the AWS API Gateway console. On the left, the navigation pane is open with the 'Stages' section selected. In the main area, under the 'Dev' stage, there is a 'Method overrides' section. A tooltip indicates that this method inherits its settings from the 'Dev' stage. Below it, a copy operation is shown with the URL: <https://c42pnoxvw9.execute-api.us-west-2.amazonaws.com/Dev/>. The status bar at the bottom right shows the URL: <https://c42pnoxvw9.execute-api.us-west-2.amazonaws.com/Dev/>.

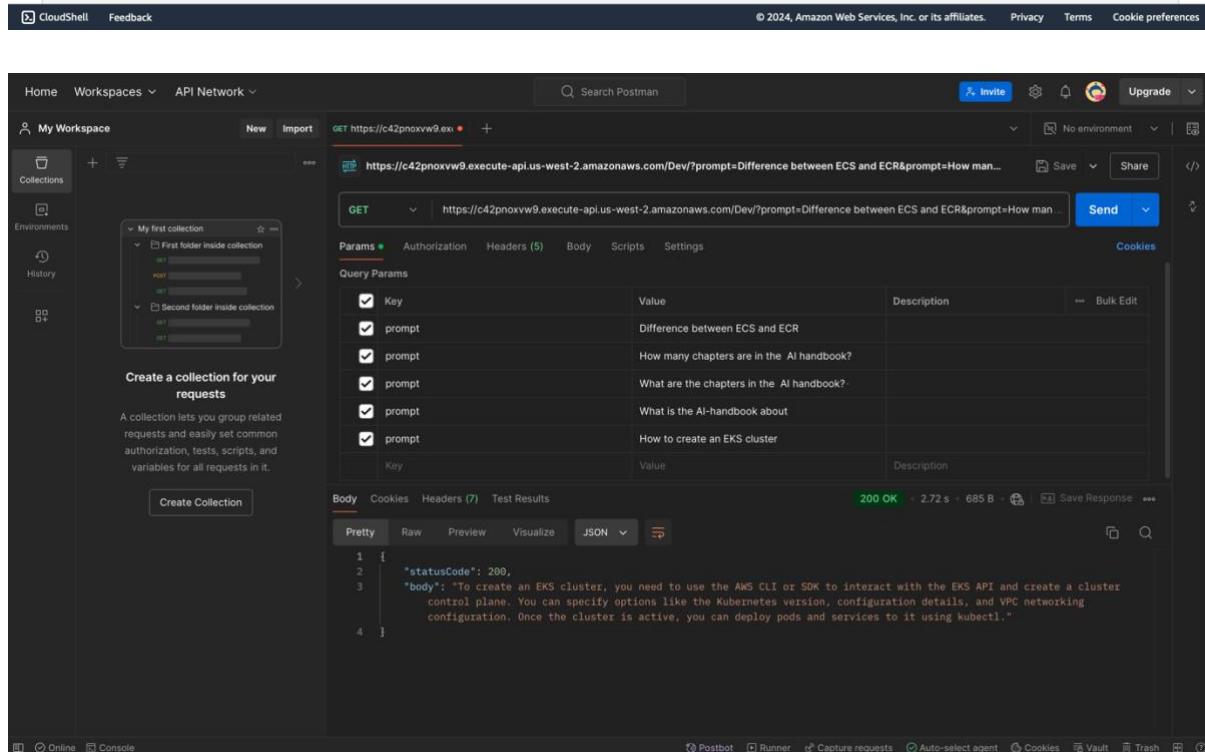
The screenshot shows the Postman application interface. The top bar includes 'CloudShell', 'Feedback', and links to 'Privacy', 'Terms', and 'Cookie preferences'. The main workspace shows a collection named 'My first collection' with two folders: 'First folder inside collection' and 'Second folder inside collection'. A 'Create a collection for your requests' button is visible. On the right, a 'GET' request is defined with the URL: <https://c42pnoxvw9.execute-api.us-west-2.amazonaws.com/Dev/?prompt=Difference between ECS and ECR>. The 'Params' tab is selected, showing two parameters: 'Key' and 'prompt', both with the value 'Difference between ECS and ECR'. The 'Body' tab shows a JSON response with the following content:

```
1 {  
2     "statusCode": 200,  
3     "body": "Amazon Elastic Container Service (ECS) is a container orchestration service that supports Docker containers and allows you to run and scale containerized applications. Amazon Elastic Container Registry (ECR) is a container image registry that is used to store, manage, share and deploy Docker container images. ECS is used to run containerized applications using the images stored in ECR."  
4 }
```

Uploaded a new PDF, **AI Handbook**, to the S3 bucket and tested the API again using **Postman**, and it successfully retrieved responses based on the updated knowledge base. This confirms that the system dynamically processes and incorporates new documents without requiring additional configuration.



The screenshot shows the AWS CloudShell interface. At the top, a green banner displays the message "Upload succeeded" with a link to "View details below.". Below this, a modal window titled "Upload: status" contains a summary table and a detailed file list. The summary table shows "Destination: s3://nitheshskmr-bedrock-bucket" with "Succeeded: 1 file, 22.8 MB (100.00%)" and "Failed: 0 files, 0 B (0%)". The "Files and folders" tab is selected, showing a table with one item: "AI-handbook..." (application/pdf, 22.8 MB, Succeeded). A note at the top of the modal says, "The information below will no longer be available after you navigate away from this page."



The screenshot shows the Postman interface. On the left, the "My Workspace" sidebar shows a collection named "My first collection" with two folders: "First folder inside collection" and "Second folder inside collection". A tooltip for "Create a collection for your requests" explains that it groups related requests and sets common authorization, tests, scripts, and variables. The main workspace shows a GET request to "https://c42pnoxvw9.execute-api.us-west-2.amazonaws.com/Dev/?prompt=Difference between ECS and ECR&prompt=How many chapters are in the AI handbook?". The "Params" tab lists several "prompt" parameters with their values. The "Body" tab shows a JSON response with a statusCode of 200 and a body containing instructions for creating an EKS cluster using the AWS CLI or SDK. The bottom navigation bar includes links for "Postbot", "Runner", "Capture requests", "Auto-select agent", "Cookies", "Vault", "Trash", and "Help".

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing 'Collections', 'Environments', and 'History'. Below it, a 'Create a collection for your requests' section is visible. The main area displays a collection named 'My first collection' with two folders: 'First folder inside collection' and 'Second folder inside collection'. Under 'First folder inside collection', there are four items: 'Difference between ECS and ECR', 'How many chapters are in the AI handbook?', 'What are the chapters in the AI handbook?', and 'What is the AI-handbook about'. Under 'Second folder inside collection', there are three items: 'How to create an EKS cluster', 'Who is the author of this book?', and 'What does Multi-layer perceptron (MLP) network...'. A modal window is open for the first item, showing a GET request to 'https://c42pnovw9.execute-api.us-west-2.amazonaws.com/Dev/?prompt=Difference between ECS and ECR&prompt=How man...'. The 'Params' tab is selected, listing 'prompt' seven times with corresponding descriptions. The 'Body' tab shows a JSON response with a status code of 200 OK, a duration of 3.71 s, and a size of 640 B. The response body is a multi-layer perceptron (MLP) definition.

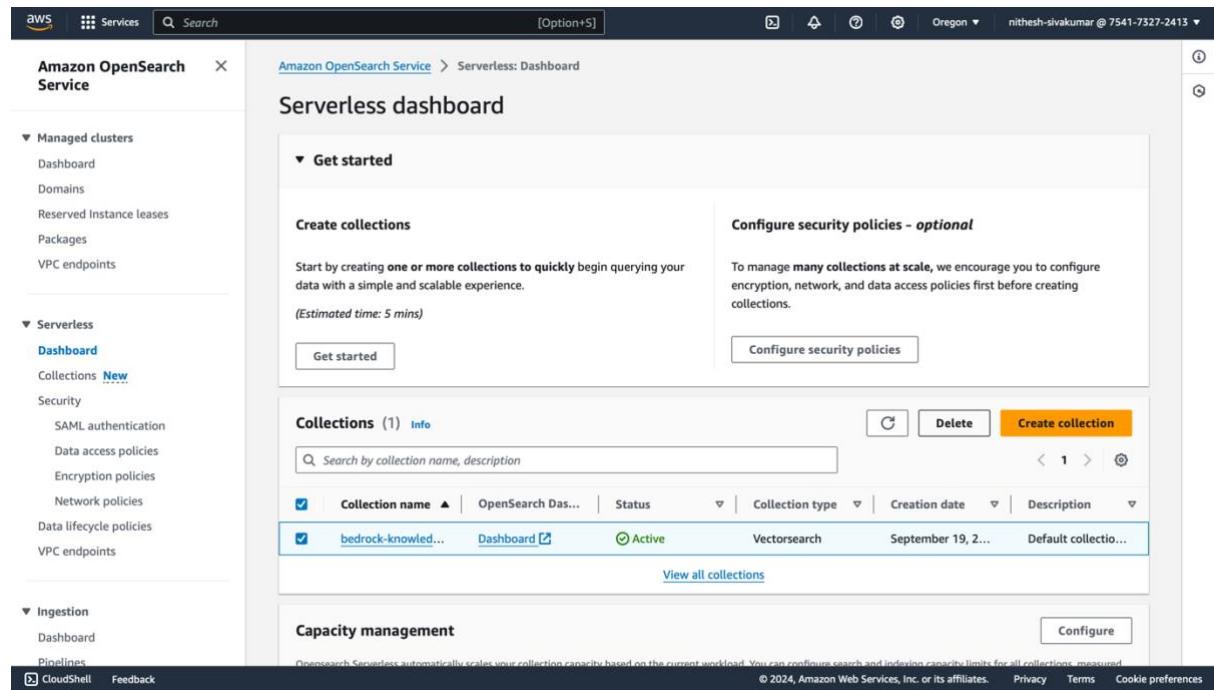
Go to **Amazon Bedrock**, navigate to the **Knowledge Base** section, select the knowledge base you created, and click **Delete** to remove it.

The screenshot shows the Amazon Bedrock console. The left sidebar includes sections like 'Getting started', 'Foundation models' (with 'Base models', 'Custom models', and 'Imported models' listed), 'Playgrounds', 'Builder tools' (with 'Prompt management', 'Knowledge bases', 'Agents', and 'Prompt flows'), 'Safeguards', 'Inference', and 'CloudShell Feedback'. The main content area is titled 'Knowledge bases' and contains a 'How it works' section with four cards: 'Upload and chat', 'Create a knowledge base', 'Test the knowledge base', and 'Use the knowledge base'. Below this is a table titled 'Knowledge bases (1)' showing one entry: 'nitheshskmr-knowledge-base' (Status: Ready, Description: -, Source files: 1, Creation time: September 19...). There are buttons for 'Edit', 'Delete', 'Test knowledge base', and 'Create knowledge base'.

Go to **Amazon OpenSearch**, locate the **vector database** created by the knowledge base, select it, and click **Delete** to remove it.

How the Vector Database Was Created

The vector database was not manually created. When setting up the **Knowledge Base** in Amazon Bedrock, selecting "Create New Vector Database" triggered AWS to automatically provision an **OpenSearch-managed vector database**. This database stores the vector embeddings of the documents uploaded to S3, enabling efficient retrieval of relevant information for user queries.



The screenshot shows the Amazon OpenSearch Service Serverless dashboard. The left sidebar navigation includes 'Managed clusters', 'Serverless' (with 'Dashboard' selected), 'Ingestion', and 'CloudShell'. The main content area displays the 'Serverless dashboard' for the 'Amazon OpenSearch Service > Serverless: Dashboard'. It features a 'Get started' section with 'Create collections' and 'Configure security policies - optional'. Below this is a 'Collections' table with one entry:

Collection name	Status	Collection type	Creation date	Description
bedrock-known...	Active	Vectorsearch	September 19, 2...	Default collectio...

At the bottom, there is a 'Capacity management' section with a 'Configure' button.