

## Set up Pinecone and AWS Bedrock for a smart QnA system:

### 1. Log into Pinecone and Define Your Project

- First, you need to sign in to your Pinecone account (create one if you don't have it already). Once you're in, you'll create a **project** (for example, "QnA Chat") that will hold all your QnA data and settings.

### 2. Create an Index in Pinecone

- An **index** in Pinecone is where your information is stored as "vectors" (basically, numbers that represent the meaning of your data).
- Go to **Database > Indexes > Configuration** in Pinecone. You'll set up an index to hold your vectors. Here's what you'll define:
  - **Vector size (1536)**: This is how detailed your vectors are. Think of it as how much information the system can store for each item.
  - **Vector type (Dense)**: This just means you're storing continuous, complex data. It's how the information is represented.
  - **Similarity metric (Cosine)**: This determines how Pinecone will compare vectors to find similar ones. Cosine similarity is a common way to do this.
- Then, you'll select the compute settings:
  - **Capacity**: You can choose **serverless**, meaning it will automatically adjust as needed based on how much traffic you're getting.
  - **Compute Provider (AWS)**: This simply means you're using AWS for computing power.

### 3. Create a Knowledge Base

- Your **knowledge base** is where all the information your bot will use to answer questions is stored. You'll gather this data from different places (like **S3 buckets** in AWS, where documents or FAQs might be stored).
- You'll need to decide how to **parse** and **embed** that data. This means turning all the text into a form that Pinecone can understand (vectors). Not to forget, chunk strategies.

### 4. Configure Pinecone in AWS Bedrock

- In **AWS Bedrock**, you'll set Pinecone as your **vector database**. You'll need to give Bedrock access to Pinecone by providing the **API Key** and **Endpoint URL** that Pinecone gives you.

### 5. Store the API Key in AWS Secrets Manager

- To keep your **Pinecone API Key** safe and secure, you'll store it in **AWS Secrets Manager**. This way, you don't have to hard-code it into your app, which is more secure.

- Once you've saved it in Secrets Manager, you'll get a **Secret ARN** (a unique ID) that you'll use later in Bedrock to access that key.

## 6. Link the Secret ARN to Bedrock

- Now, go into your **Bedrock** settings and enter the **Secret ARN** from AWS Secrets Manager. This lets Bedrock pull the Pinecone API key securely when it needs to.

## 7. Sync Knowledge Base

- Finally, you'll set up the sync between **AWS Bedrock** and **Pinecone**. You'll configure how Bedrock should query Pinecone to find relevant answers based on the data in your knowledge base.
- This step ensures your knowledge base is available to the QnA system and the system knows how to query it effectively for the right answers.

By following these steps, you'll be creating a powerful QnA system that can understand and answer questions based on your knowledge base. **Pinecone** helps store the answers in a way that makes it easy to find the right one quickly, and **AWS Bedrock** helps connect all the pieces together. The **Secrets Manager** keeps things secure, and everything runs smoothly on **AWS's** infrastructure.

This setup allows for a smart, scalable, and secure QnA chatbot or application that can efficiently search your knowledge base and provide accurate answers to users.

✓ Sync completed for data source - 'knowledge-base-quick-start-ldils-data-source'

[Amazon Bedrock](#) > [Knowledge Bases](#) > knowledge-base-quick-start-790it

### knowledge-base-quick-start-790it

Test
Delete

#### Knowledge Base overview

**Knowledge Base name**  
knowledge-base-quick-start-790it

**Knowledge Base description**  
—

**Service Role**  
[AmazonBedrockExecutionRoleForKnowledgeBase\\_790it](#)

**Knowledge Base ID**  
JOLJOBXVX

**Status**  
✓ Available

**Created date**  
March 06, 2025, 04:56 (UTC+05:30)

**Log Deliveries**  
Configure log deliveries and event logs in the [Edit](#) page.

**Retrieval-Augmented Generation (RAG) type**  
Vector store

**Data source (1)** Sync Stop sync Add Add documents from S3

Data sources contain information returned when querying a Knowledge Base.

✓	Data so...	Status	Data sour...	Account ID	Source Link	Last sync ...	Last
✓	knowledg...	✓ Available	S3	29606256...	<a href="#">s3://aws-r...</a>	March 06,...	-

### Test Knowledge Base

☒ Generate responses

**Llama 3 8B Inst... v1**  
On-demand

Define feature engineering

Feature engineering is the process of creating new features as a combination of existing ones, adding domain knowledge to the dataset, and transforming raw features into more meaningful and relevant ones for learning models.<sup>[1]</sup>

[\[1\] A Short Guide for Feature Engineering and Feature Selection.pdf](#)

Enter your message here

Common dimensions used for crossing include: time region business types Still take call log for example, we can have crossed features like: number of calls during night times/day times, number...

## Embeddings model

### Model

Titan Embeddings G1 - Text v1.2

### Embeddings type

Float vector embeddings

### Vector dimensions

1536

## Vector database


### Vector database

Pinecone

### Connection string

https://rag-01-y94kwmu.svc.aped-4627-b74a.pinecone.io

### Credentials secret ARN

 arn:aws:secretsmanager:us-east-1:296062562306:secret:prod/rag-JfEiR1

### Text field name

text

### Metadata field name

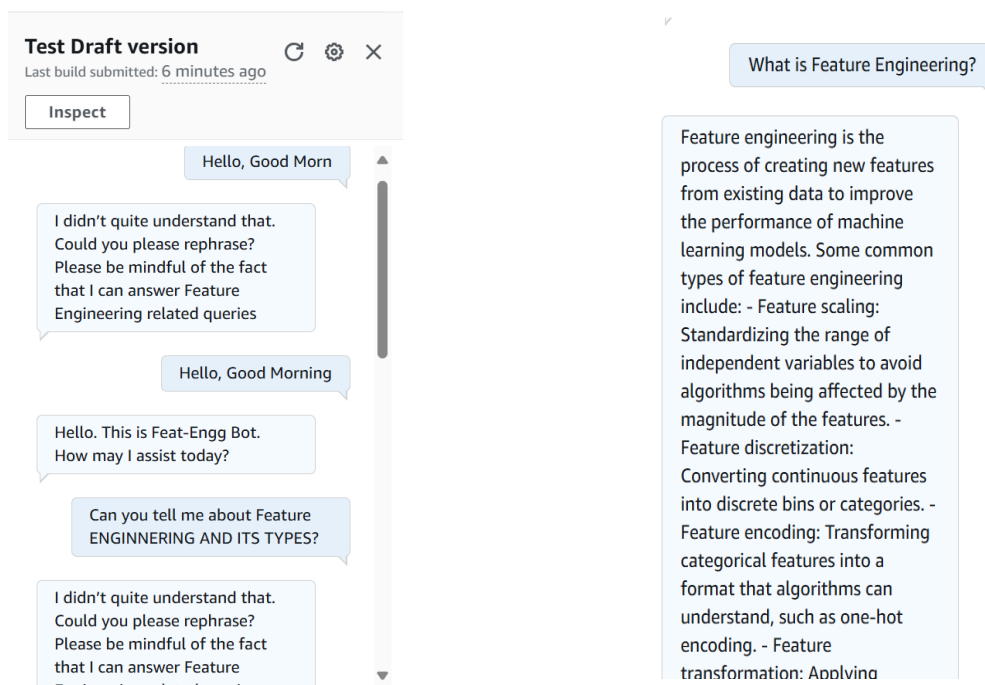
metadata

## Creating Bot with Lex:

>> Starting with a Blank (Traditional) Bot. This is where we define intents manually. Firstly an intent is defined ("Welcome Intent") where we define some sample utterances.

**Note:** If we ask something other than those in Sample Utterances. Then initial response is not obtained and **fallback intent** is invoked. A fallback intent is used when the bot cannot find a suitable match for the user's input in any of the defined intents. It's crucial to provide a good fallback mechanism to ensure that the bot can gracefully handle **unrecognized** or **out-of-scope** queries.

We can also choose a **built-in qna intent** for our purpose. Then select the respective Knowledge Base ID (Bedrock). Save Intent and Build. Default Model: Anthropic Claude Haiku.



The screenshot displays the AWS Lex console's 'Test Draft version' interface. At the top, it shows 'Last build submitted: 6 minutes ago' and an 'Inspect' button. The chat history on the left includes:

- User: 'Hello, Good Morn'
- Bot: 'I didn't quite understand that. Could you please rephrase? Please be mindful of the fact that I can answer Feature Engineering related queries'
- User: 'Hello, Good Morning'
- Bot: 'Hello. This is Feat-Engg Bot. How may I assist today?'
- User: 'Can you tell me about Feature ENGINNERING AND ITS TYPES?'
- Bot: 'I didn't quite understand that. Could you please rephrase? Please be mindful of the fact that I can answer Feature Engineering related queries'

On the right, a sample response for the query 'What is Feature Engineering?' is shown:

Feature engineering is the process of creating new features from existing data to improve the performance of machine learning models. Some common types of feature engineering include:

- Feature scaling: Standardizing the range of independent variables to avoid algorithms being affected by the magnitude of the features.
- Feature discretization: Converting continuous features into discrete bins or categories.
- Feature encoding: Transforming categorical features into a format that algorithms can understand, such as one-hot encoding.
- Feature transformation: Applying

Roles and Policies stated in IAM:

Roles (5)

Info

Delete

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Q Search

<input type="checkbox"/>	Role name	▲ Trusted entities	Last activity
<input type="checkbox"/>	<a href="#">AmazonBedrockExecutionRoleForKnowledgeBase_790it</a>	AWS Service: bedrock	10 minutes ago
<input type="checkbox"/>	<a href="#">AWSServiceRoleForCostOptimizationHub</a>	AWS Service: cost-optimization-hub	7 hours ago
<input type="checkbox"/>	<a href="#">AWSServiceRoleForLexV2Bots_BUNWWPBL2A</a>	AWS Service: lexv2 (Service-Linked R	10 minutes ago
<input type="checkbox"/>	<a href="#">AWSServiceRoleForSupport</a>	AWS Service: support (Service-Linker	-
<input type="checkbox"/>	<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS Service: trustedadvisor (Service	-

Permissions policies (4)

Remove

Permissions are defined by policies attached to the user directly or through groups.

Q Search

Filter by Type

All types

<input type="checkbox"/>	Policy name <div></div>	▲ Type	▼ Attached via <div></div>
<input type="checkbox"/>	<div></div> <a href="#">AdministratorAccess</a>	AWS managed - job function	Group <a href="#">rag-01</a>
<input type="checkbox"/>	<div></div> <a href="#">AmazonBedrockFullAccess</a>	AWS managed	Directly
<input type="checkbox"/>	<div></div> <a href="#">AmazonS3FullAccess</a>	AWS managed	Directly, Group <a href="#">rag-01</a>
<input type="checkbox"/>	<div></div> <a href="#">IAMUserChangePassword</a>	AWS managed	Directly

Set up AWS bedrock at backend

Go to IAM >> User >> Security Credentials >> Create Access Key >> Use Case >> CLI..

In the CLI (VSCode Terminal), type **aws configure**. Provide **access keys, secret access keys, region** etc and have it configured.

Type **aws s3 ls** to know about the buckets that have been created.

Launch **vscode** from Anaconda Navigator.

### Chatbot Backend

Establish connections with Bedrock Model. Define **Model id and kwargs** (Max tokens, temperature, top\_p value etc). In the code section, we define **demo\_chatbot** function.

In the next part, we make use of **ConversationBufferMemory** (where a Buffer Memory creates summary of all the previous questions and takes the next question, with the help of a foundation model, here **Bedrock Claude Haiku**). Primary Take in parameters are: llm\_data = "**demo\_chatbot**" function in our case, and max\_token\_limit = **300**.

Next, connect Prompt template + Bedrock KB & Model + Buffer Memory and bring them under one function.

Finally we use the **invoke** method to send the Input and Memory to the Foundation Model.

Save as **chatbot\_backend.py**

### Chatbot Front end

Import Streamlit and the backend function. Then, provide details of the memory function which in our case is **demo\_memory()**.

Add UI chat History to the session cache.

Run Locally!

