gen-AI for SIM

Architecture & Approach note

Table of Contents

­­

Scope of Gen AI for SIM ..............................................................................................................................3

Gen AI Architecture ..……………………………………………………………………………………………………………………………..3

Detailed Overview ………………………………………………………………………………………………………………………………….3

1. Our system relies on three main models: ..………………………………………………………………………………..3
2. How We Build Our Knowledge Base: ..……………………………………………………………………………………….4
3. The Process in Action: .……………………………………….……………………………………………………………………..4
4. Message Formatting and Response Prediction: ..……………………………………………………………………….4
5. Querying from Azure SQL: .…………………………….………………………………………………………………………….6
6. SageMaker Studio endpoint configuration ....................................................................................6

API Overview ………………………………………………………....……………………………………………………………………………10

Maintenance/Enhancement Activity Catalog: ………...……………………………………………………………………………12

1. Updates to functionality of application – addition or modifications: ……………………………………….12
2. Update to OLAP schema for application: …………………………………………………………………………………13
3. Switching Models: ..………………………………………………………………………………………………………………...16
4. Audio: …………………………………………………………………………………………….....……………………………………16

Gen AI Code Overview …………………………………………………………………………………..…………………………………….17

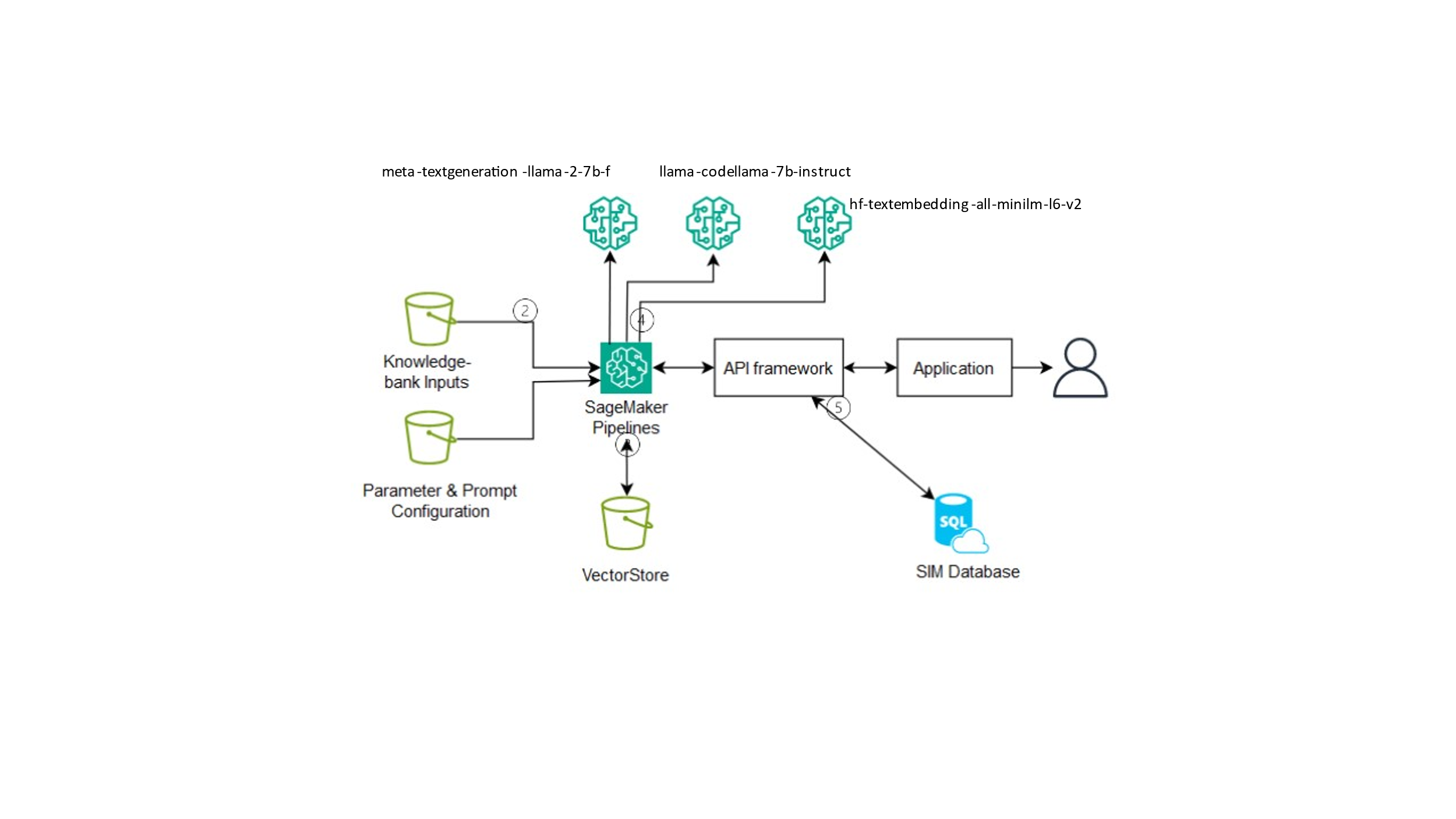
Infrastructure Overview …………………………………………………………………………………...……………………………….. 22

Security Overview .................................................................................................................................... 31

# Scope of Gen AI for SIM

To develop an agnostic platform API for Summarization as well as Navigation functionality consisting of Input-text and Input-voice use case that serves web, android and ios applications. This document captures the functional and technical details of the Gen AI APIs.

# Gen AI Architecture



# Detailed Overview

### Our system relies on three main models:

* **Text Embedding Model:** We employ a sophisticated Hugging Face MiniLM model with 6 layers (L6). This model functions as the backbone of our system, transforming text inputs into embeddings (384 dimensions). This is deployed on a ml.g4dn.xlarge instance and is made available through a SageMaker endpoint.
* **Chat Model:** Additionally, we utilize another model from Meta named meta-textgeneration-llama-2-7b-f which aids in understanding user queries and generating appropriate responses. The 7B version of the same is deployed on a ml.g5.xlarge SageMaker instance and is accessible through a SageMaker endpoint.
* **Code Generation Model:** A pretrained version of Llama models from meta for code generation, called -llama-codellama-7b-instruct. The 7B-Instruct version of the model is deployed on a ml.g5.xlarge SageMaker instance and is accessible through a SageMaker endpoint.

### How We Build Our Knowledge Base:

* We have two primary knowledge bases created:
  + Knowledge base for Schema definition context:
    - We have created a CSV file catalogue containing detailed table descriptions stored in our cloud bucket. In addition, schema extracts for each table is stored on S3.
    - These descriptions are loaded, and vector stores are created to efficiently store and manage our data, facilitating navigation and schema comprehension.
  + Knowledge base for navigation/sitemap context:
    - A navigation catalogue is created with details on the modules, pages, items within sections and the description of the activities which a user undertakes on them.
    - This is then processed (chunked and converted into embeddings) and stored into a vector store.

### The Process in Action:

* Loading CSVs and creating vector stores tailored for various purposes.
* Leveraging FAISS (Facebook AI Similarity Search) for similarity searches and clustering, enhancing our ability to locate relevant documents promptly.
* Our numerical data representations, or vectors, are stored in FAISS index files. Additionally, we ensure accessibility by storing these vectors in our cloud storage (S3).

### Message Formatting and Response Prediction:

* Functions are in place to format messages for conversations, alternating between system and user roles and generating appropriate prompts. SageMaker predictors route user queries to the relevant model based on whether they pertain to navigation or schema-related information.
* Essentially, a multi-step process is configured to chain the inference flow:
  + The first step is identification of user intent to navigate or to summarize.
  + Depending on the response the flow is split to perform a similarity search on corresponding vector database and return the best match on document chunks.
  + The chunks are then fed into respective LLM models to generate a response as a query or a summarization.
  + For schema-related queries, SQL queries are dynamically generated based on user inputs and relevant schema documents.
  + The query execution is done on Azure SQL database and the results summarized through llama chat models again.

### Querying from Azure SQL

* The LLM model is generating SQL query, this SQL query is being queried on Azure SQL database by making use of API call to Azure SQL database and getting the response of the query in JSON format and storing as a vector database.
* This is executed through the API framework with data transfer required only for sharing the output of the SQL query with the API.

### SageMaker Studio endpoint configuration

* + The configurations below can be changed based on the updated models.
  + Config.py has 3 endpoint named as:

ENDPOINT\_CHAT = "JS-Meta-llama2"

ENDPOI NT\_CODE= "JS-Meta-code-llama"

ENDPOINT\_EMBEDDING = "JS-HF-MiniLM-L6-v2"

**Starting a SageMaker Endpoint (Repeat the steps for all the three endpoints)**

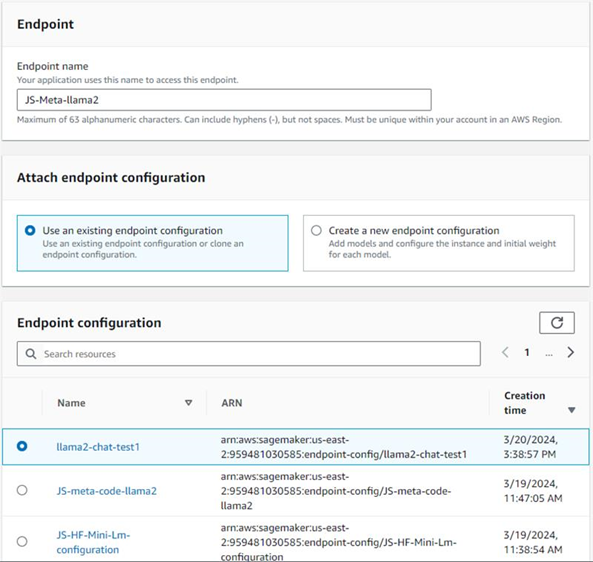
1. Access AWS Management Console: Log in to the AWS Management Console.

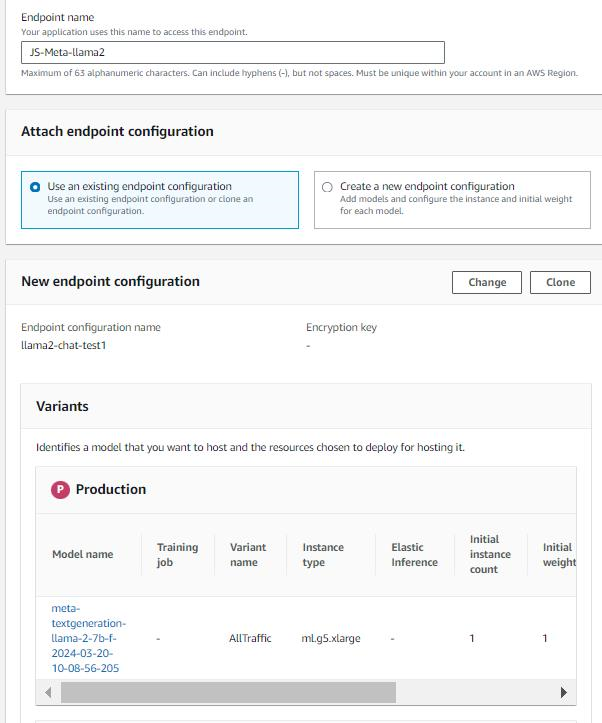
2. Navigate to SageMaker: Go to the Amazon SageMaker console by searching for "SageMaker".

3. Select Endpoint: From the left-hand menu, under inference, select "Endpoints" then select "create endpoints".

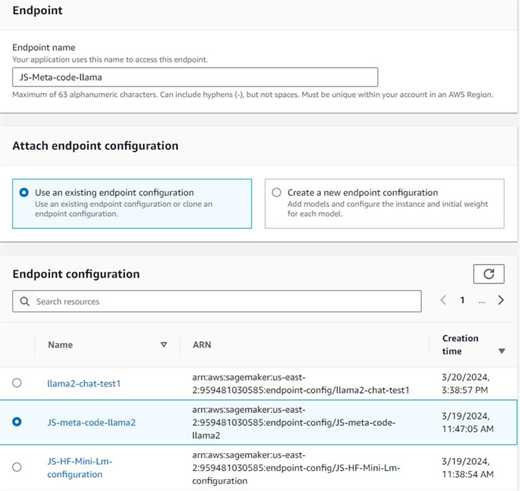
4. Choose Endpoint: Enter name as mentioned in the config.py file for "Endpoint name". Then choose "use an existing endpoint configuration" and select the endpoint corresponding to the name of the endpoint from the list of endpoints displayed.

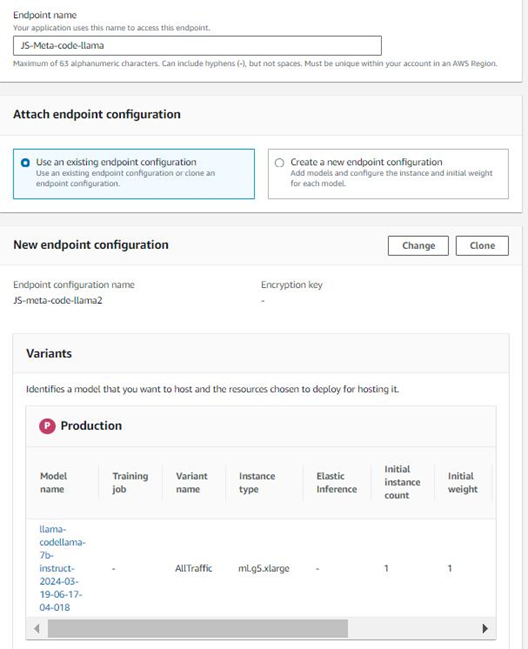
For ENDPOINT\_CHAT = "JS-Meta-llama2"



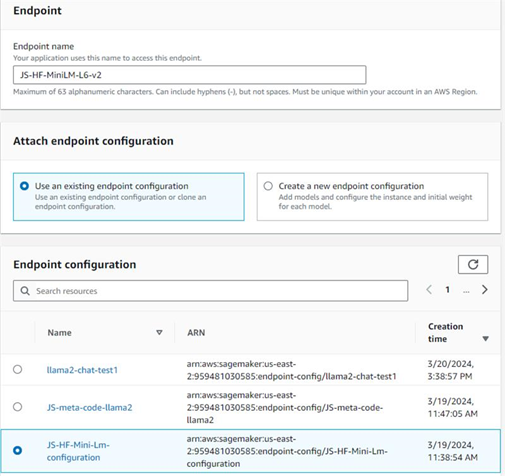


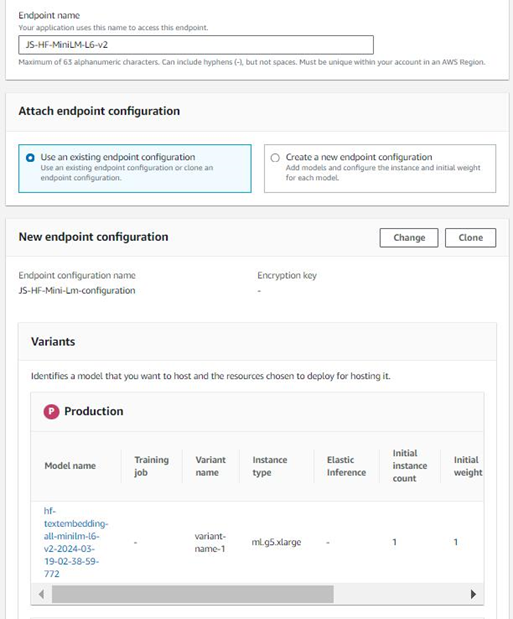
ENDPOINT\_CODE= "JS-Meta-code-llama"





ENDPOINT\_EMBEDDING = "JS-HF-MiniLM-L6-v2"

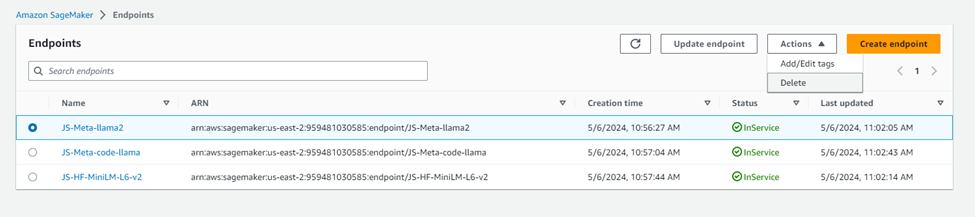




5. Start Endpoint: Click on "select endpoint configuration" and then "create endpoint". The endpoint will now start, and its status will change to "InService" once it is fully operational.

**Delete a SageMaker Endpoint**

1. Access AWS Management Console: Log in to the AWS Management Console.
2. Navigate to SageMaker: Go to the Amazon SageMaker console.
3. Select Endpoint: From the left-hand menu, under inference, select "Endpoints" to view a list of existing endpoints.
4. Choose Endpoint: Choose the endpoint you wish to stop from the list of endpoints displayed.
5. Stop Endpoint: Click on the "Actions" drop-down menu and select "Delete".



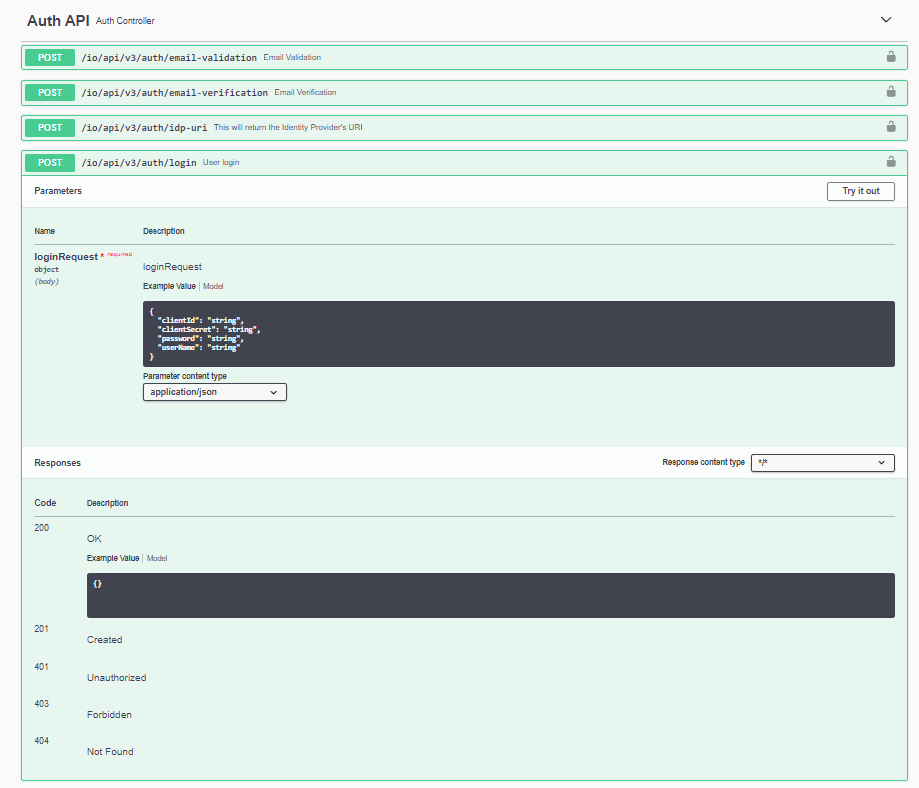
# API Overview

**Authentication process:**

* The SIM application will get an input from the user (this will the existing user login authentication followed both for SIM as well as 2.0)
* Based on the user input once an API gets called, the API will hit the 2.0 platform and for this, the SIM team needs to authenticate in the 2.0 platform from the API service user profile standpoint which will have the Gen-AI APIs in place. The service user credentials are created by 2.0 team.
* The service user credentials will provide the access token to authenticate and validate along with the username.
* Once the authentication is successful, the SIM team will be copying the files to the Gen-AI S3 bucket [dev-data-and-gen-ai - S3 bucket | S3 | Global (amazon.com)](https://s3.console.aws.amazon.com/s3/buckets/dev-data-and-gen-ai?region=us-east-2&bucketType=general&tab=objects)
* And as last step, the SIM application will trigger the Gen AI APIs (i.e., Conversation API and Vector Refresh API). These APIs will generate the responses based on which the application will take action whether to show text on the screen or to navigate to the link.

To trigger the Gen AI APIs, the first step is to retrieve the access token from the **Auth API** for validating the token which is present in the dev 2.0 server <https://iiop-dev.aatmunn.net/io/swagger-ui/#/> and as part of validation, below steps are to be followed:

1. Under Auth API go to - POST - /io/api/v3/auth/login



1. Click on try it out.
2. Use the below request body to hit the API mentioned above:

**{**

**"clientId": "2.0-api",**

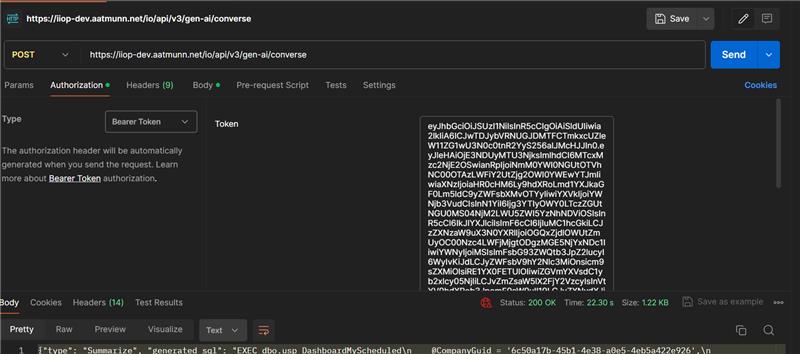
**"clientSecret": "wltzLsW89vdPsltQjfJBjLJDnOpUBU11",**

**"password": "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",**

**"userName": "ai-api-user"**

**}**

1. After executing, copy the access token.
2. On Postman, paste the token in the token textbox under Authorization tab (bearer token).



1. Execute the query. With the Authentication token selected we will get the desired output. But when we keep it as ‘No Auth’ we will get access denied error.

There are 2 API’s which are deployed in 2.0 server for Gen AI functionality:

1. **The Converse API**that is using a POST method is a simple API that is exposed in a Web or Mobile interface where the user provides their question based on their requirements to navigate through the application or get a summarized text based on their query as the response returned by the API.

**The URL for Converse API is "/api/v3/gen-ai/converse"**which takes input payload as below:

*{*

*userQuery: “string”*

*sourcePlatform: “ios/android/web”,*

*preferredFormat: “html/json”,*

*userName: “string”,*

*companyGuid: “string“*

*}*

and returns the response as below:

*{*

*“type”: “summarize/navigate”,*

*“generated\_sql”: “string”,*

*“generated text”: “string”,*

*“generated data”: “string”*

*}*

1. **The Vector-Refresh**API using the POST Method call is a component of the system responsible for refreshing the vector store containing embeddings of schema files. This API provides a convenient means of refreshing the vector store programmatically, eliminating the need for manual intervention.

**The URL for Vector-Refresh API is "/api/v3/gen-ai/vector-refresh".**

It does not require any input payload.

Success Response: If the execution is completed successfully, the API will respond with an HTTP status code of 200 (OK) and a message indicating successful execution:

*"Execution completed successfully."*

Failure Response: If the execution fails for any reason, the API will respond with an appropriate HTTP status code and a message indicating the failure:

*"Vector refresh failed."*

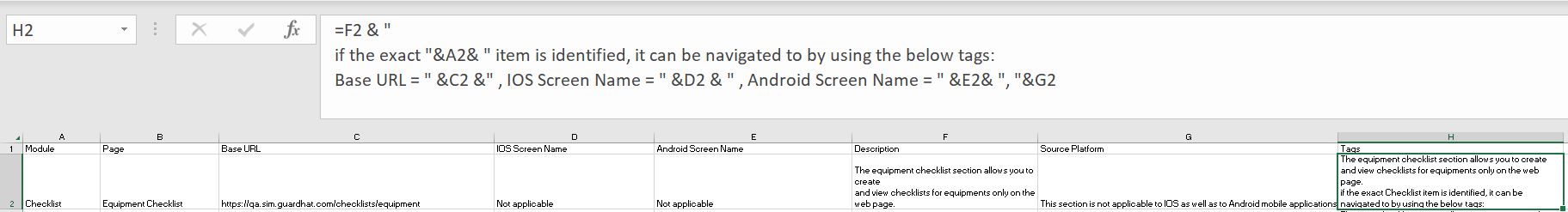
# Maintenance/Enhancement Activity Catalog:

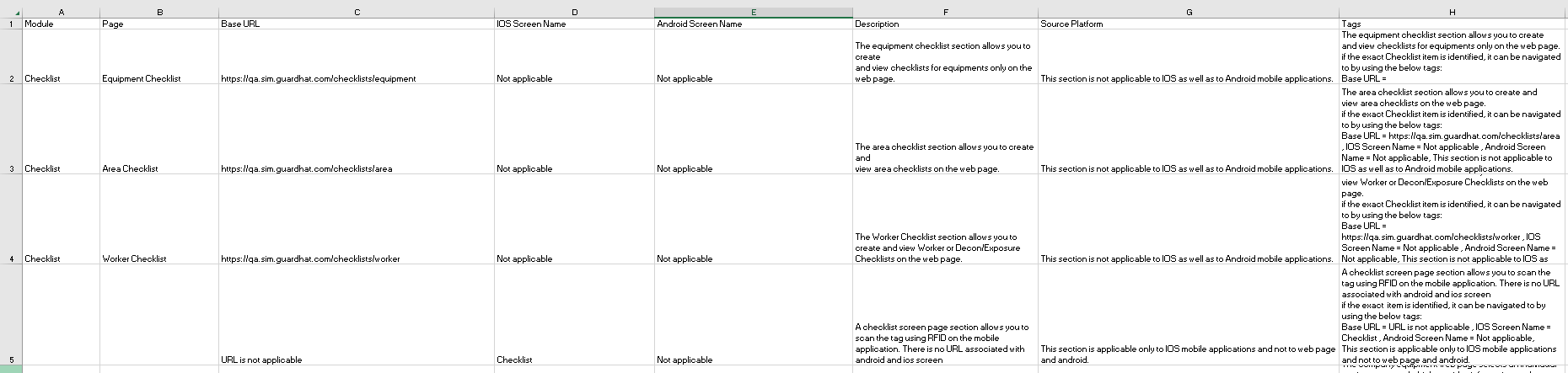
### Updates to functionality of application – addition or modifications:

As and when the functionality on the front applications is updated, the only change needed will be to update the navigation knowledge bank and re-create the navigation vector store and the files are uploaded on S3 bucket as below:

1. We need to update the navigation csv file.

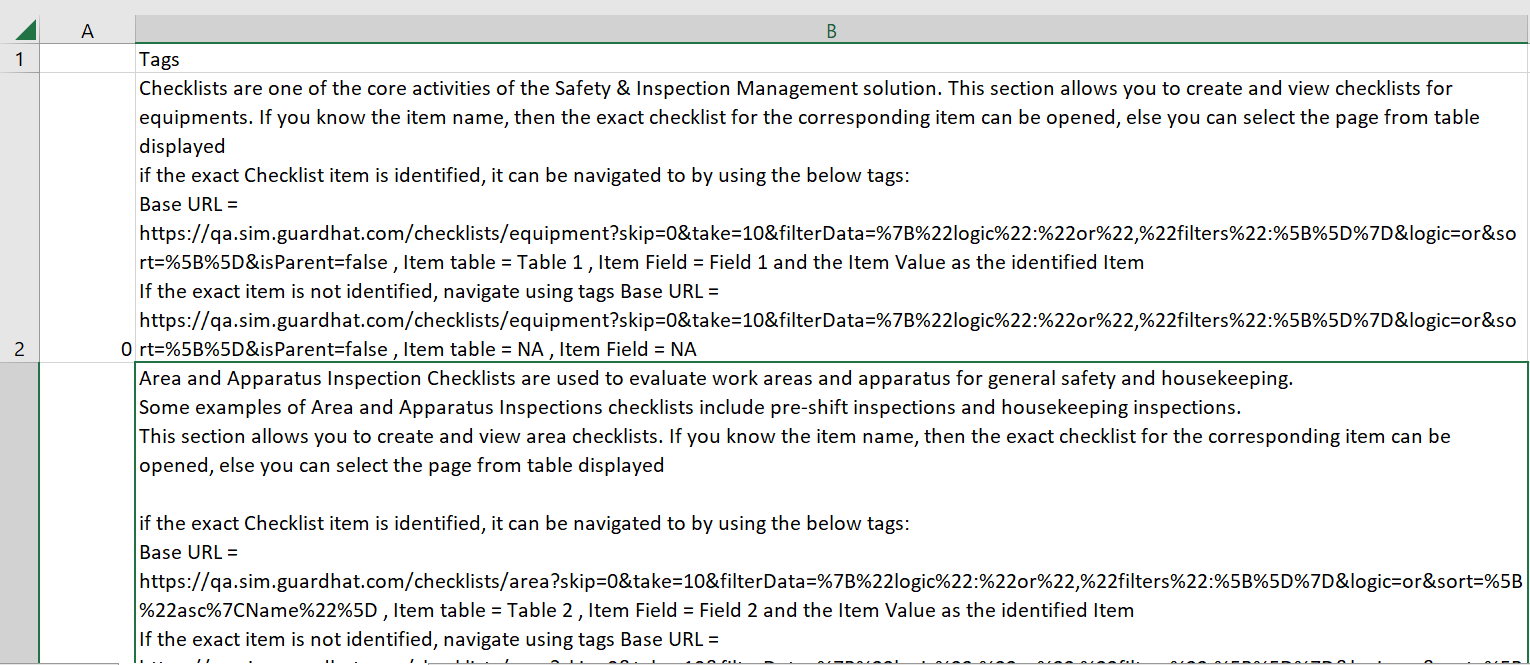
**Navigation\_links.csv** - This file comprises information about all modules within the Web, Android and IOS platforms. It includes base url for web pages and screen names for IOS and Android and their descriptions. The H column in excel file can be updated using the provided formula and then converted to CSV before uploading it to the S3 bucket.





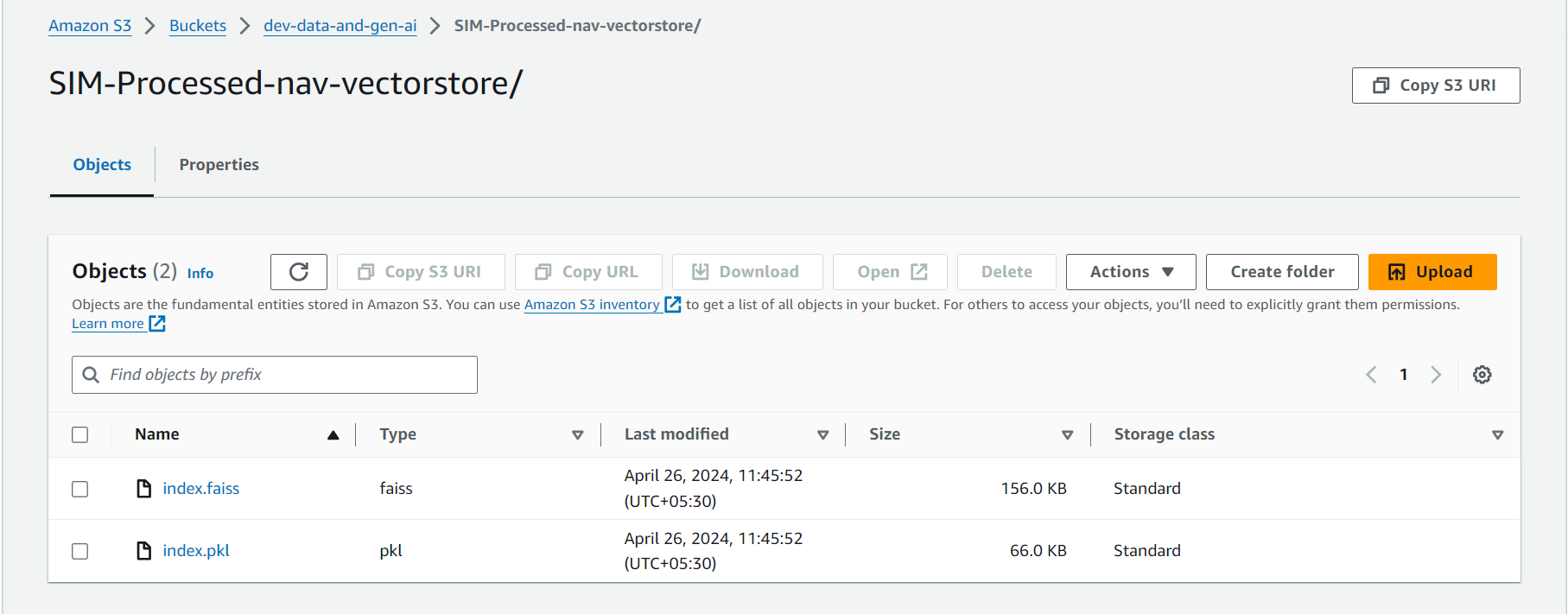
1. Process the updated navigation list to create a new vector store using existing pipeline.

**Processed\_navigation\_links.csv** - After processing data from "navigation\_links.csv", the model extracts essential information from the file and saved it in processed\_navigation\_links.csv.



1. Ensure that new page/functionality is included into the vector store by executing the vector store creation pipelines.

**SIM\_processed\_navigation\_vectorstore** - After undergoing processing, navigation link vectors are stored as FAISS indices. This indexing method facilitates efficient similarity search and retrieval operations based on the processed navigation link vectors.

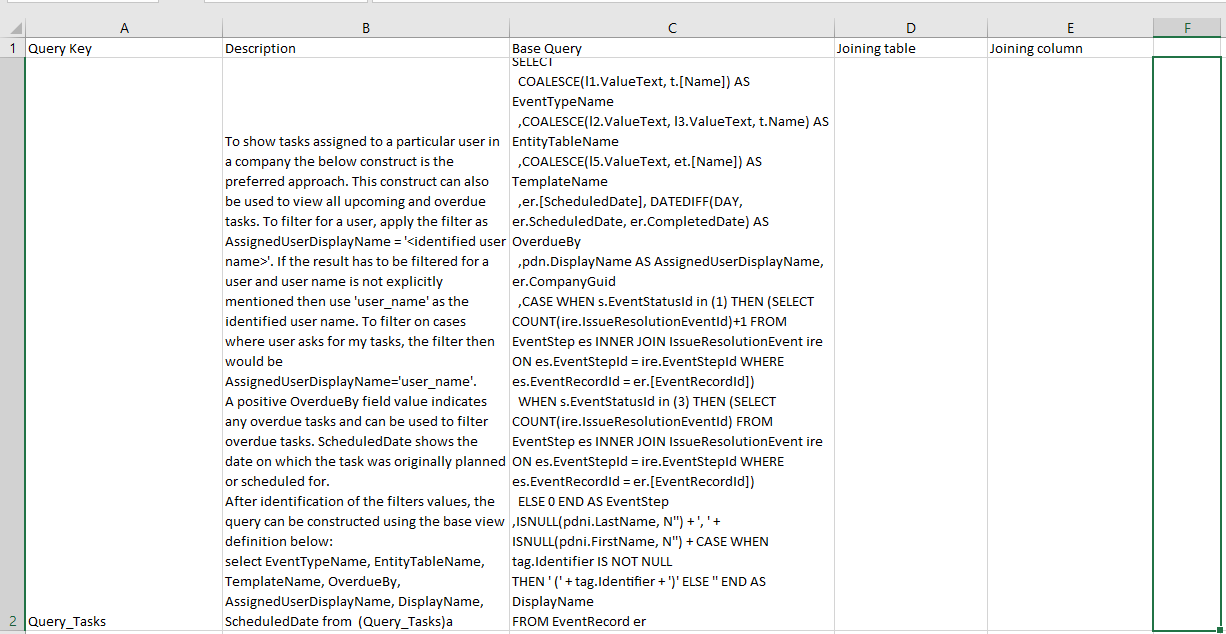


### Update to OLAP schema for application:

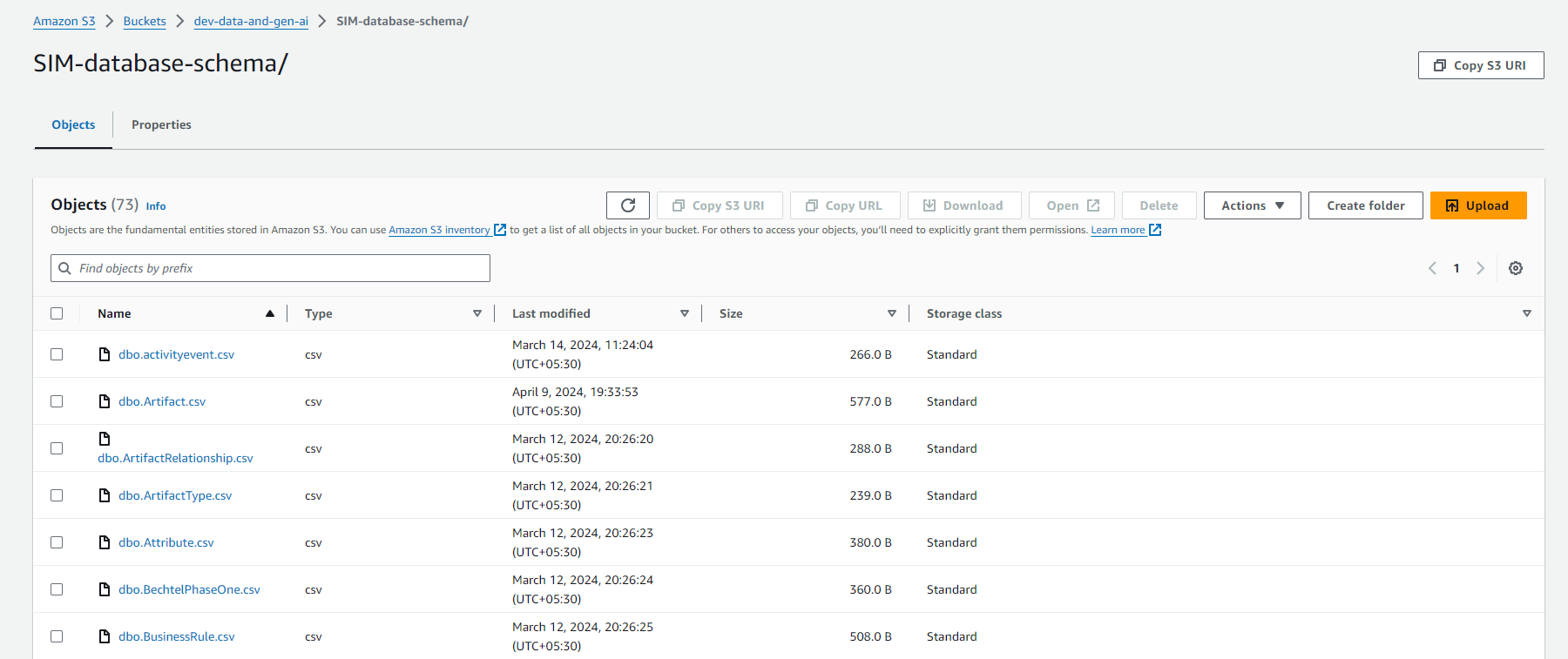
Whenever there is an enhancement of the models, essentially the knowledge base will need to be updated with addition information – change in entities and descriptions and the vector subsequently updated and the files are uploaded on S3 bucket as below:

1. We need to update the database schema list csv file.

**SIM-Tables-descriptions\_performance.csv -** This file provides thorough information on the modules (Query Key) like tasks, issues, equipment checklist, area inspection, area checklist and equipment inspection. The description of each of these query keys contains details about the module itself along with the filters like username or in case of checklist it is missed, pending etc. that is required for that module. The base query column contains the entire SQL query required to populate the expected output.

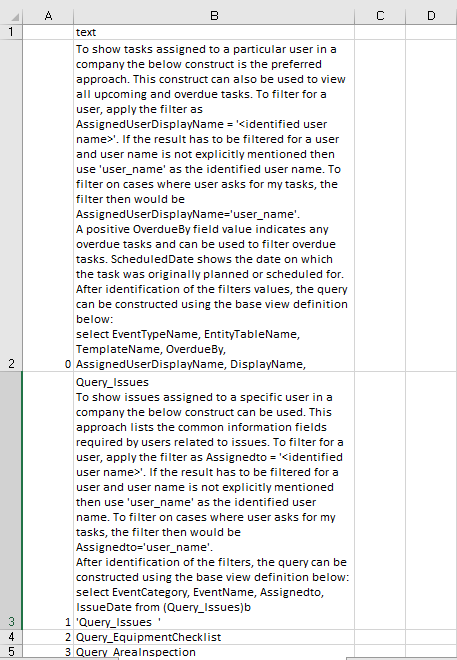


**SIM-database-schema** - Each schema file is uploaded to the s3 bucket, accompanied by its respective table and store procedure ensuring systematic storage within the SIM-database



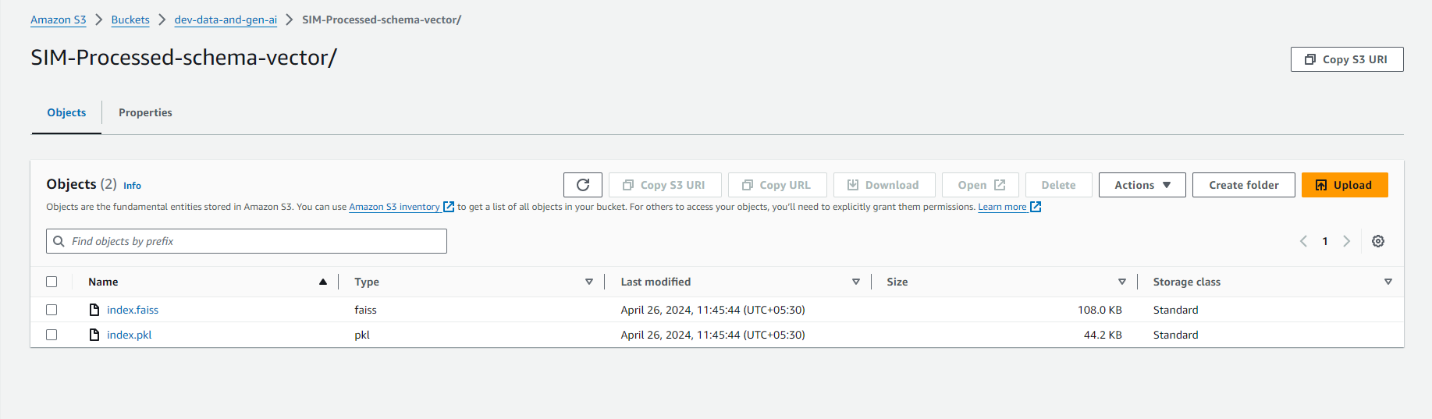
1. We can process the updated schema list to create a new schema vector store using existing pipeline.

**Processed\_schema.csv** - The initial step involves uploading individual schema files containing details of tables and stored procedures, which the model reads and processes. It extracts descriptions from the "SIM-Tables-descriptions.csv" file and merges them with the schema of each table. The resulting merged schema is then stored in the "Processed\_schema.csv" file.



1. Incorporate the new table/KPI into the schema vector by executing the vector store creation pipelines.

**SIM\_processed\_schema\_vector/** - The schema data undergoes vectorization before being stored. The resulting vectors are indexed using FAISS for efficient similarity search and the processed schema data is saved in PKL files.



### Switching Models:

While inference pipeline is modularized, each model requires specific formatting of inputs and responses, when changing the model family, the below changes will be required:

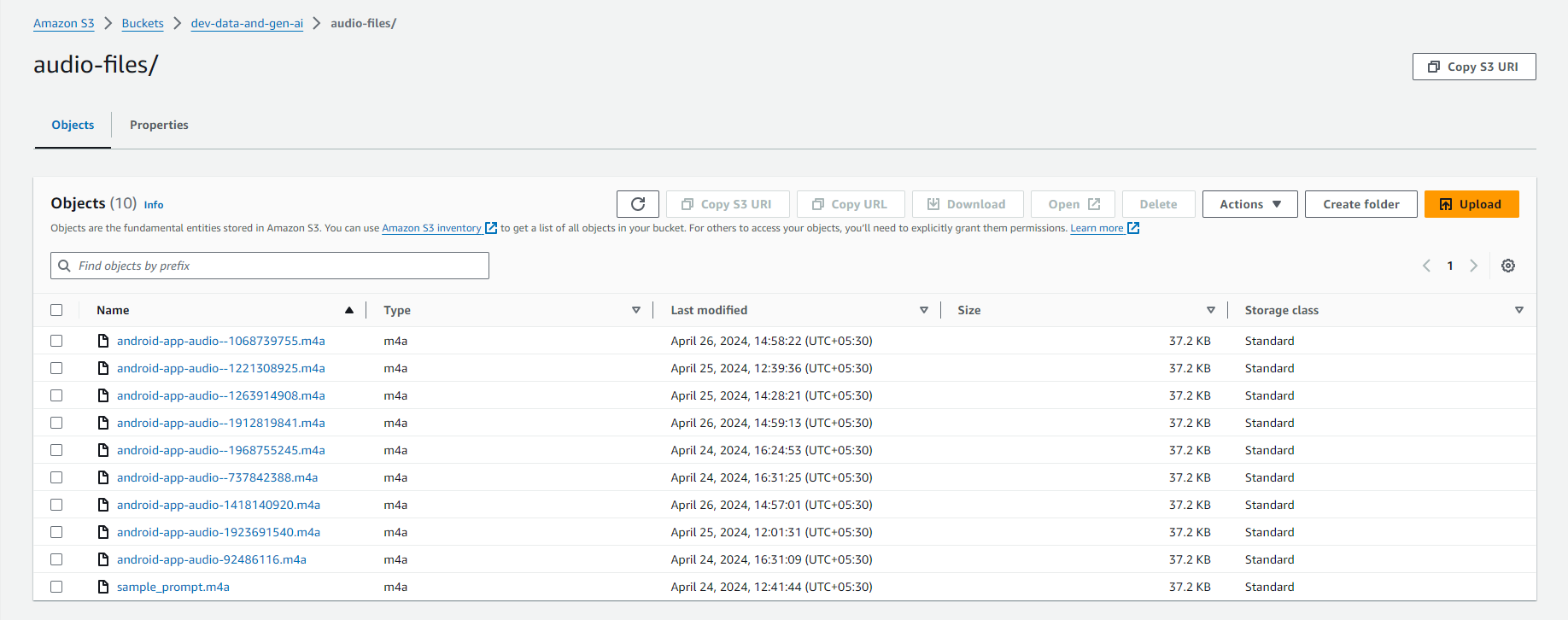
1. After identifying the new model which suits the requirement, need to switch the model configuration accordingly.
2. We need to make the changes in the code to utilize the new model for embeddings.
3. Ensure the compatibility and adaptability of the existing pipeline with the new model.

For models in the same family, the formatting remains the same and the update can be done, just by executing the model deployment change with no impact on code.

### Processing Audio Files:

**Audio files** - Audio files are used as input for AWS (Amazon Web Services) Transcribe, which transcribes speech from audio files into text. Below are the steps/procedure used in this job.

1. First the user will upload the audio file in the below S3 location
   1. s3://dev-data-and-gen-ai/audio-files/



1. Then by giving s3Url in the prompt, the code for AWS transcribe starts
2. Initially it checks if the files exist or not and if file exists in s3 bucket, the transcribe job starts
3. We are using AWS transcribe to do the transcription from audio to text
4. Once the transcribe job is completed, the transcript file is stored in S3 bucket as json file
5. From the json file, we are extracting the translated text from the audio file and feeding that to user query for further execution.

# Gen AI Code Overview

The following artefacts are incorporated for the Gen AI code flow which has been deployed in ‘*gh-2-api’* GitHub repository in develop branch:

1. **config.py**

The document outlines key variables for S3 storage, system prompts, SageMaker endpoints, and model parameters. It manages data processing, navigation, and language model interactions, making text generation and schema processing for database tables smooth and efficient.

TABLE\_LIST\_FILE\_URI = Stores the URI for a CSV file on S3 containing descriptions of database tables.

BUCKET = Stores the S3 bucket name

BASE\_S3\_URI = Stores the URI for the S3 bucket

INPUT\_NAVIGATION\_KEY = It points to a specific S3 file, that contains navigation links.

PROCESSED\_NAVIGATION\_KEY = Specifies the location on S3 where the processed navigation links data is stored after initial processing.

PROCESSED\_NAVIGATION\_VECTOR = Specifies the S3 location where navigation link vectors are stored after being processed into FAISS indices.

TABLE\_LIST\_KEY = Used to read the DB table description

DETAILS\_SCHEMA\_KEY = Here detailed schema files for various database tables are stored.

PROCESSED\_SCHEMA\_KEY = where processed schema information is stored.

PROCESSED\_SCHEMA\_VECTOR = S3 location where vectorized schema data is stored.

EMBED\_LLM\_ENDPOINT\_NAME = Specifies the name of the SageMaker endpoint used for language model embeddings

ENDPOINT\_CHAT = [meta-textgeneration-llama-2-7b-f-2024-03-20-10-08-56-205](https://us-east-2.console.aws.amazon.com/sagemaker/home?region=us-east-2#/models/meta-textgeneration-llama-2-7b-f-2024-03-20-10-08-56-205)

ENDPOINT\_CODE = llama-codellama-7b-instruct-2024-03-19-06-17-04-018

ENDPOINT\_EMBEDDING = [hf-textembedding-all-minilm-l6-v2-2024-03-19-02-38-59-772](https://us-east-2.console.aws.amazon.com/sagemaker/home?region=us-east-2#/models/hf-textembedding-all-minilm-l6-v2-2024-03-19-02-38-59-772)

CHAT\_INPUT\_SYSTEM\_PROMPT / SUMMARIZATION\_SYSTEM\_PROMPT / SUMMARIZATION\_INPUT\_SYSTEM\_PROMPT / NAVIGATION\_INPUT\_SYSTEM\_PROMPT =

System prompts are in place to provide prompts or cues to the Large Language Model (LLM) system to guide its responses during an interaction. It offers contextual hints or suggestions to steer the model's understanding and generation of text, helping it produce more relevant and coherent responses based on the given prompt

CLASSIFICATION\_PARAMETERS = {

"max\_new\_tokens": 20,

"top\_p": 0.9,

"temperature": 0.1,

}

NAVIGATION\_PARAMETERS = {

"max\_new\_tokens": 100,

"top\_p": 0.9,

"temperature": 0.1,

}

QUERY\_GENERATION\_PARAMETERS = {

"max\_new\_tokens": 100,

"top\_p": 0.9,

"temperature": 0.1,

}

SUMMARIZATION\_PARAMETERS = {

"max\_new\_tokens": 100,

"top\_p": 0.9,

"temperature": 0.1,

}

max\_new\_tokens:

• Description: This parameter specifies the maximum number of new tokens (or words/pieces of words) that the model can generate in a single operation.

• Effect: Setting this value to 1024 means the model will generate up to 1024 tokens in response to a prompt. This limits the length of the output, which can help manage the computational resources needed for generation and ensure that responses are of a manageable size

Top\_p:

Description: This sampling strategy involves selecting the smallest set of words or tokens whose cumulative probability exceeds the threshold p (in this case, 0.9 or 90%).

temperature:

• Description: This parameter controls the randomness of the predictions by scaling the logits before applying softmax to convert them into probabilities.

• Effect: A lower temperature (like 0.1) makes the model outputs more deterministic, favoring more likely outcomes and resulting in less varied text. This is useful when you want the model to be more predictable and stick closer to high-probability text completions.

The chunk size defines how many input texts will be grouped together as request.

CHUNK\_SIZE\_TABLE\_LIST = 2000

CHUNK\_SIZE\_NAV\_VECTOR\_STORE = 1000

CHUNK\_OVERLAP\_TABLE\_LIST = 0

CHUNK\_OVERLAP\_NAV\_VECTOR\_STORE = 0

CHUNK\_SIZE = 100

The code defines a dictionary named "errors" where each key represents a specific error type, accompanied by a status code and a corresponding error message. This structure provides a centralized way to manage and handle various types of errors that may occur during program execution.

*errors = {*

*"NoError":{"status\_code": 200, "message":"Execution Complete"},*

*"FileNotFoundError":{"status\_code": 501, "message":"File not found"},*

*"UnexpectedError":{"status\_code": 503, "message":"Unexpected Error"},*

*"EmptyDataFrame":{"status\_code": 504, "message":"Empty Data Frame"},*

*"NoSuchBucket": {"status\_code": 504, "message":"Bucket not found"},*

*"EmptyDataError": {"status\_code": 504, "message":"Empty Data"},*

*"EndpointConnectionError": {"status\_code": 504, "message":"Endpoint connection cannot be established"},*

*"RuntimeError": {"status\_code": 504, "message":"Runtime error"},*

*"NoSuchKey": {"status\_code": 504, "message":"No such key exists"},*

*"KeyError": {"status\_code": 504, "message":"Invalid file structure"},*

*"TypeError": {"status\_code": 504, "message":"Data fetch error"},*

*"InterfaceError":{"status\_code": 503, "message":"Database Connection Error"},*

*"DatabaseError":{"status\_code": 503, "message":"Database Error"},*

*"AudioFileNotFound":{"status\_code": 512, "message":"Audio file not found"},*

*"TranscribeFailed":{"status\_code": 513, "message":"Transcribe Job Failed"},*

*"NavVectorFailed": {"status\_code": 513, "message":"Nav Vector Failed"},*

*"SchemaVectorFailed": {"status\_code": 513, "message":"Schema Vector Failed"},*

*"ValueError": {"status\_code": 504, "message":"Value Error"}*

*}*

1. **Constant.py**

The provided functionality enhances text embedding capabilities through SageMaker, facilitating batch processing and direct embedding retrieval. BaseEmbeddingsContentHandler ensures proper JSON formatting for communication with the SageMaker endpoint, guaranteeing compatibility and accurate data handling.

**class BaseSagemakerEndpointEmbeddings**

Extends functionality for computing document embeddings by sending text data in managed chunks to a SageMaker inference endpoint and collecting the results.

Methods: Includes embedded document for batching and processing text data, and an overridden \_\_call\_\_method to allow the class to function like a callable that directly returns embeddings for given texts.

**def embed\_documents**

Compute doc embeddings using a SageMaker Inference Endpoint.

Args:

texts: The list of texts to embed.

chunk\_size: The chunk size defines how many input texts will

be grouped together as request. If None, will use the

chunk size specified by the class.

Returns:

List of embeddings, one for each text.

**class BaseEmbeddingsContentHandler**

Manages the formatting and parsing of JSON data for requests and responses to and from the SageMaker endpoint, ensuring compatibility and correct data handling.

**def transform\_input and def transform\_output**

**Method :** Implements transformer\_input to convert text data into a JSON byte-stream for the endpoint, and transformer\_output to decode and extract embeddings from the endpoint’s response.

1. **Lambda\_converse\_function.py**

SageMakerEndpointEmbeddingsJumpStart facilitates communication with SageMaker for embedding transformation. ContentHandlerEmbedder converts text to JSON for model processing. Other functions handle AWS Secrets Manager, file operations, response formatting, SQL extraction, user input processing, and audio transcription via AWS Transcribe.

**class SageMakerEndpointEmbeddingsJumpStart**

It communicates with the SageMaker endpoint using embedding transformer for handling the content transformation

**class ContentHandlerEmbedder**

This class converts text input into JSON encoded bytes for model processing.

**def get\_secret\_value**

This function gets the secret values for the database credentials stored in AWS Secrets Manager.

**def format\_messages**

This function format the messages using Llama-2 chat models. The model only supports 'system', 'user' and 'assistant' roles, starting with 'system', then 'user' and alternating (u/a/u/a/u...). The last message must be from 'user’

**def copy\_from\_S3**

This function downloads files from a S3 bucket to a local directory and checks if the directory exists or not.

**def format\_response\_as\_html**

Format pandas DataFrame as HTML (not used).

**def format\_response\_as\_csv**

Format pandas DataFrame as CSV.

**def format\_response\_as\_xml**

Format pandas DataFrame as XML

**def format\_response\_as\_json**

Format pandas DataFrame as JSON

**def format\_response\_as\_text**

Format pandas DataFrame as plain text.

**def get\_stored\_procedure\_name**

Extracts the stored procedure name from generated SQL.

**def main or def lambda\_handler**

* The main function takes input variables from the config file along with the user inputs to be provided at the time of execution.
* It also checks for the file existance in S3 for audio transcription and converts the voice into text format.
* It also checks for navigation/summarization functionality using vector embedding and similarity search and uses llm models to generate appropriate responses or SQL queries.
* The function also takes input from the user to work on various platforms (iOS, Android, or web) and generates the final output according to the user’s preference (HTML, CSV, XML, text, or JSON).

1. **Lambda\_converse\_vector.py**

DocumentEmbedderSageMaker facilitates SageMaker endpoint communication for content transformation. JSONEmbeddingTransformer customizes data to JSON for embedding. Functions handle S3 operations like parsing URIs, file fetching, uploading, processing, and indexing for efficient navigation and schema management, driven by main function orchestrating these tasks.

**class DocumentEmbedderSageMaker**

It communicates with the SageMaker endpoint using embedding transformer for handling the content transformation

**class JSONEmbeddingTransformer**

This is a custom class for transforming data into a JSON format suitable for embedding purposes.

**def read\_csv\_file\_from\_s3**

This function fetches a CSV file from an S3 bucket using the specified bucket and key.

**def upload\_file\_to\_s3**

This function uploads a csv file to a s3 bucket using the specified bucket and key

**def copy\_from\_S3**

This function copies processed\_navigation\_links.csv, processed\_schema.csv files from an specified S3 bucket to a local directory.

**def create\_nav\_vector\_store**

Processes navigation related data by reading a CSV from S3, cleaning the DataFrame and reuploading it. Additionally, it handles text splitting and indexing using FAISS for efficient similarity search, and uploads the index to S3

**def process\_table\_list**

Processes database table schemas by reading from S3, compiling schema information, and saving the structured data back to S3.

**def main**

This function will take the input variables from the config file, reads csv file from s3, processes table list and create vector store for navigation.

# Infrastructure Overview

**List of resources created:**

|  |  |
| --- | --- |
| Resource | Resource Name/ID |
| VPC | vpc-04e234fbf3942c02b |
| Public Subnet | subnet-00c56a14d25f73ed2 |
| Public Subnet | subnet-02a1ee4cb8dd6542d |
| Private Subnet | subnet-06365f802852cb414 |
| Private Subnet | subnet-054e48fc2c3f7d418 |
| Main Route table | rtb-0122d09b5b41f658a |
| Private Route table | rtb-08f98a9c78d89a99c |
| Internet Gateway | igw-09996f548ae38a053 |
| Security Group | sg-0e8e9ae70616dfd8d |
| NAT Gateway | nat-020a3df6936c8f6c1 |
| Elastic IP | 3.20.193.194 |
| S3 Gateway Endpoint | vpce-034acf1c5c81067c9 |
| Sagemaker Runtime Interface Endpoint | vpce-09be1f22884f1e146 |
| Sagemaker API Interface Endpoint | vpce-051374a8470b50657 |
| Lambda Function | lambda\_function\_converse |
| Lambda Function | lambda\_function\_vector |
| ECR | lambda |
| IAM Role For Lambda Functions | LambdaExec |
| Policies Attached to IAM Role | AmazonEC2FullAccess  AWSLambdaBasicExecutionRole  AWSLambdaVPCAccessExecutionRole  S3PolicyForLambda  SagemakerPolicyForLambda  SecretManageraccess  AWSTranscribeFullAccess |
| CloudWatch LogGroups | /aws/lambda/lambda\_function\_vector  /aws/lambda/lambda\_function\_converse |
| Github Repository | <https://github.com/guardhat/gen-ai> |

**Detailed Overview:**

* We have deployed the Gen AI application on a serverless AWS service Lambda inside private subnet within a VPC
* We have created two lambda functions lambda\_function\_converse and lambda\_function\_vector,where lambda\_function\_converse is deployed in the private subnet of VPC (vpc-04e234fbf3942c02b)
* A NAT Gateway has been created for outbound internet access from Lambda Function
* A S3 Gateway Endpoint, Sagemaker Runtime and API interface endpoints have been created for Lambda to access S3 bucket and Sagemaker endpoints outside the VPC
* A Docker file is created with all the dependencies and libraries for the application, and it also has the odbc driver installation instructions embedded in it. Both the functions are being deployed using the same container image created from the docker file
* ECR Repository is created to push the docker image, so that the lambda functions can be created with the container image option
* lambda\_function\_converse has CMD override "lambda\_function\_converse.lambda\_handler"
* lambda\_function\_vector has CMD override “lambda\_function\_vector.lambda\_handler” for their execution
* We have attached an Elastic IP to the NAT Gateway for the purpose of having a static public IP and which is whitelisted in the Azure MSSQL database
* We have attached an IAM role to the Lambda functions for capturing logs through AWS CloudWatch
* The Gen AI python code is being pushed to the Github Repository <https://github.com/guardhat/gen-ai> to deploy the code via CI/CD pipeline
* Once the code is pushed to the GitHub Repository, which contains the Docker File and Jenkins file, the Docker image will be created and pushed to the ECR Repository. As soon as the ECR Repository receives the image, it gets deployed to both the Lambda functions lambda\_function\_converse and lambda\_function\_vector

**Steps to implement the infrastructure deployment**

* Navigate to VPC service -> Create VPC -> VPC and more -> Enter the VPC name(iiop-dev-vpc-analytics) -> Select the default IPV4 CIDR -> Number of AZs (2) if high availability feature is required -> No. of Public and Private subnets (2) -> NAT Gateway and VPC endpoints(None) -> Tags(project : analytics) -> create VPC
* By default, 4 route tables will be created, so delete one public (not main route table) and one private route table
* Associate the private subnets to private route table and public subnets to main public route table
* Add entry of the Internet Gateway to the main route table routes with Destination (0.0.0.0/0)
* Create S3 gateway endpoint , sagemaker runtime and API interface endpoints choosing private route table
* Create NAT gateway(nat-020a3df6936c8f6c1) in public subnet(subnet-00c56a14d25f73ed2) with allocating Elastic IP option
* Add NAT Gateway entry in the private route table(rtb-08f98a9c78d89a99c)
* Get the NAT gateway public IP (3.20.193.194) whitelisted in the Azure SQL server firewall
* Add VPC CIDR (10.0.0.0/16 and 172.31.0.0/16) as source in Security Group

(sg-0e8e9ae70616dfd8d) inbound rule open for all traffic

* Log into EC2 instance “[i-062cb191aa021ef3f](https://us-east-2.console.aws.amazon.com/ec2/home?region=us-east-2#InstanceDetails:instanceId=i-062cb191aa021ef3f)” using mobaxterm
* Create a directory for analytics: ‘mkdir analytics’
* Move to analytics directory: ‘cd analytics’
* Create a file with name “Dockerfile” and include below lines using vi editor:

FROM public.ecr.aws/lambda/python:3.10

RUN pip3 install pandas

RUN pip3 install sagemaker

RUN pip3 install langchain

RUN pip3 install faiss-cpu

RUN pip3 install boto3

RUN pip3 install langchain-community

RUN pip3 install s3fs

RUN pip install pyodbc

ENV PYTHONUNBUFFERED=TRUE

COPY lambda\_function\_converse.py ${LAMBDA\_TASK\_ROOT}

COPY constant.py ${LAMBDA\_TASK\_ROOT}

COPY config.py ${LAMBDA\_TASK\_ROOT}

COPY lambda\_function\_vector.py ${LAMBDA\_TASK\_ROOT}

CMD [ "lambda\_function\_converse.lambda\_handler" ]

RUN yum update -y && \

yum install -y wget && \

wget -q https://packages.microsoft.com/keys/microsoft.asc -O microsoft.asc && \

rpm --import microsoft.asc && \

wget https://packages.microsoft.com/config/rhel/7/prod.repo -O /etc/yum.repos.d/mssql-release.repo && \

yum update -y && \

ACCEPT\_EULA=Y yum install -y msodbcsql17 && \

yum clean all

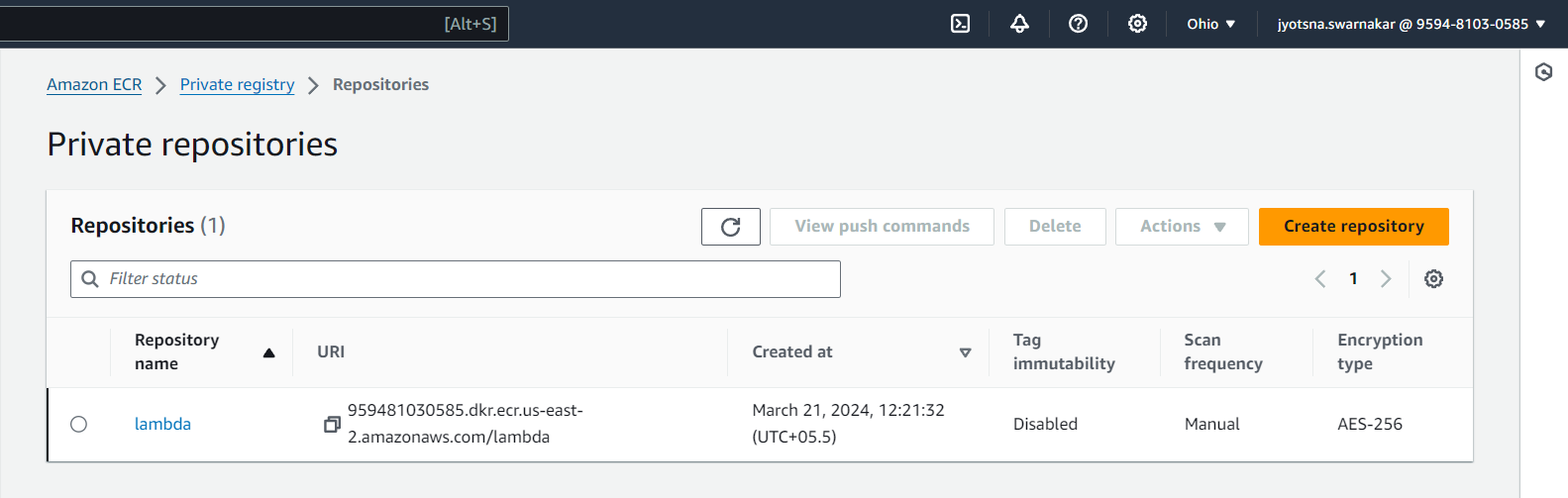
* Copy lambda\_function\_converse.py, lambda\_function\_vector.py, constant.py and config.py code file to the same path
* Go to AWS account “959481030585” -> Search “ECR” service and navigate to it -> In the left navigation pane expand “Private registry” -> Click on “Repositories” -> Click on “Create Repository” -> Select Private for visibility settings, Name : lambda and click on “Create repository”
* After the repository is created , copy its URI :
  + 959481030585.dkr.ecr.us-east 2.amazonaws.com/lambda
* Go the EC2 instance terminal and run the below command to create the image from the above created Dockerfile and tag it with the ECR repository URI

docker build -t 959481030585.dkr.ecr.us-east-2.amazonaws.com/lambda:latest -f Dockerfile .

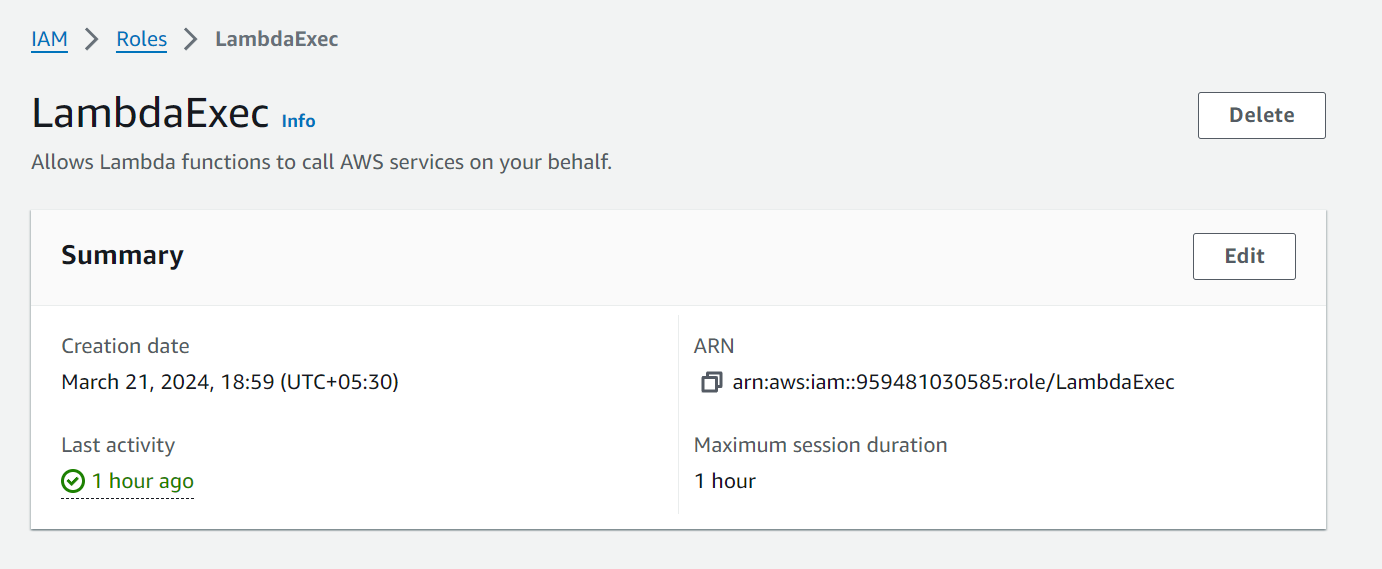
* Run the below commands one by one to push the image to ECR repository that we have created

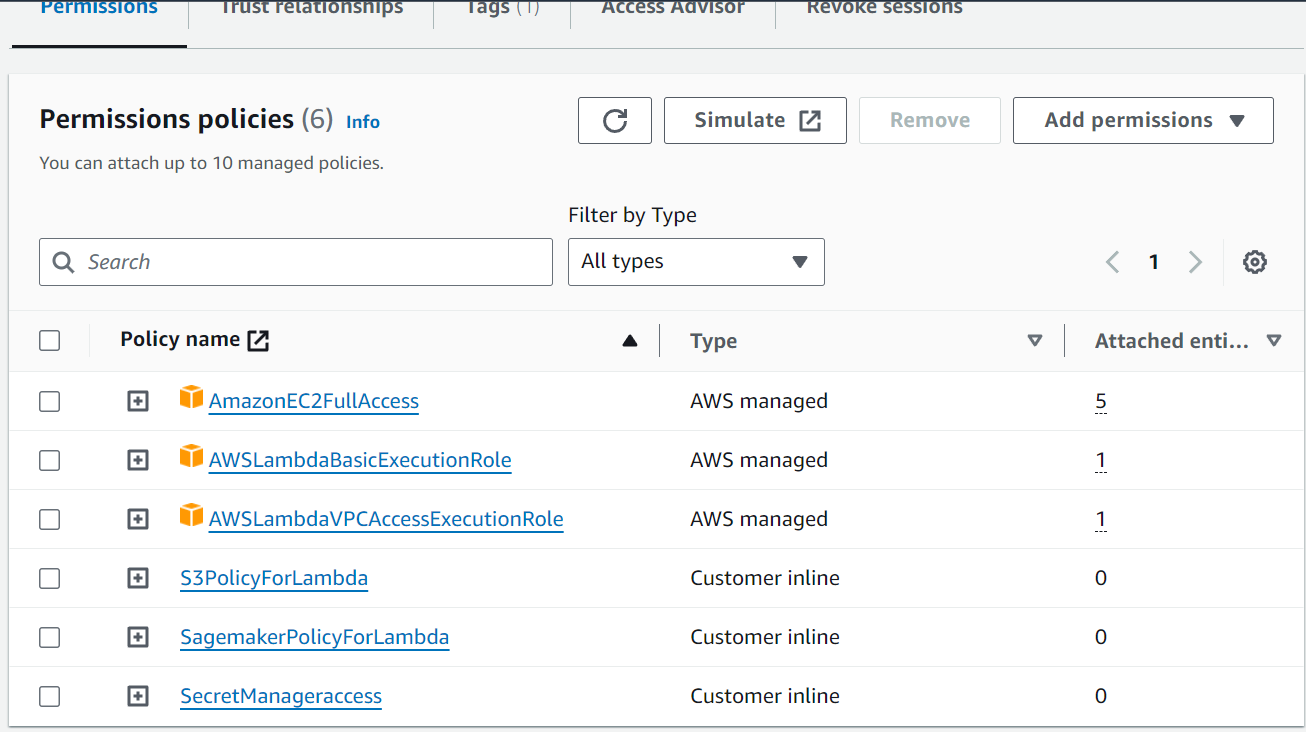
above

* Aws configure
* Enter your access key , secret access key and region
* aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 959481030585.dkr.ecr.us-east-2.amazonaws.com
* docker push 959481030585.dkr.ecr.us-east-2.amazonaws.com/lambda
* Go to ECR and check if image is visible inside the repository



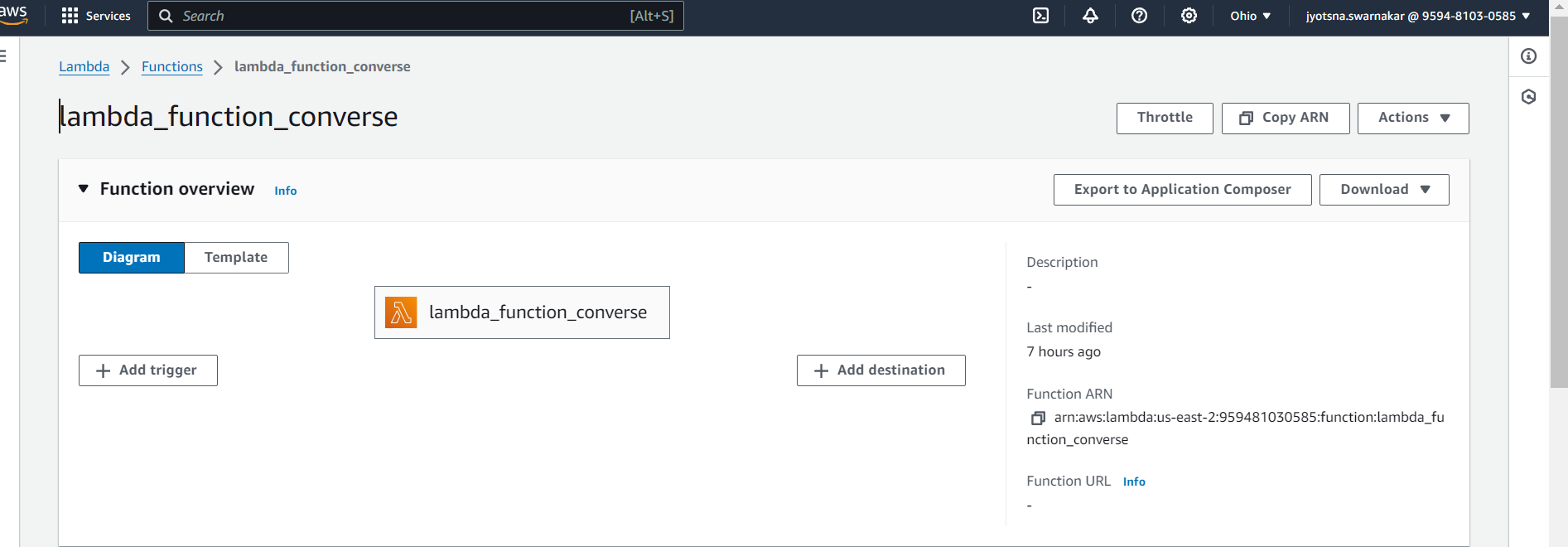
* Go to IAM and navigate to Roles in the left navigation pane -> Click “create role” -> Select AWS service-> Use case: Lambda -> Add permissions (Read/list/write to S3 bucket, Read/List sagemaker endpoints,basic lambda executions permissions, [AmazonEC2FullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-2#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEC2FullAccess), [AWSLambdaVPCAccessExecutionRole](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-2#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2Fservice-role%2FAWSLambdaVPCAccessExecutionRole), [SecretManageraccess](https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-2#/roles/details/LambdaExec/editPolicy/SecretManageraccess?step=addPermissions), AWSTranscribeFullAccess) -> Click Next -> Role name : LambdaExec -> Tags (project : analytics) -> Click create role



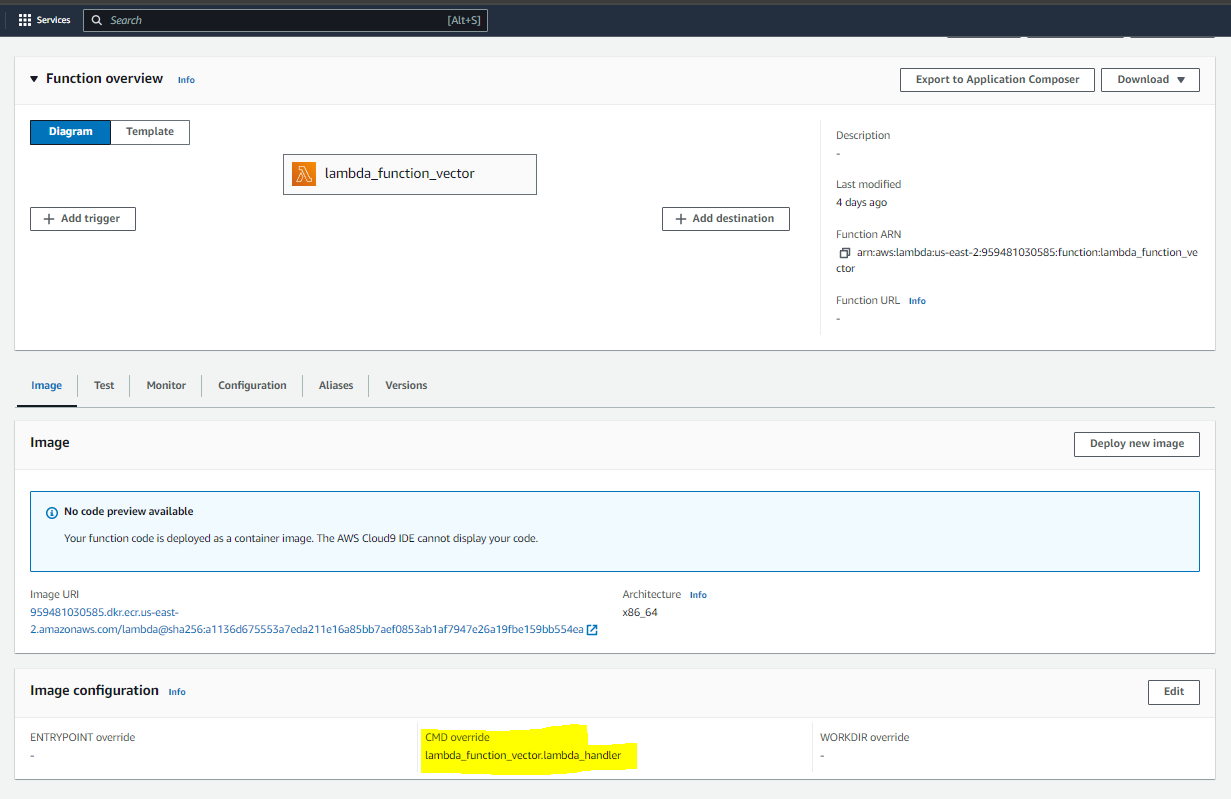


For Deploying Lambda Functions for the first time perform the below steps

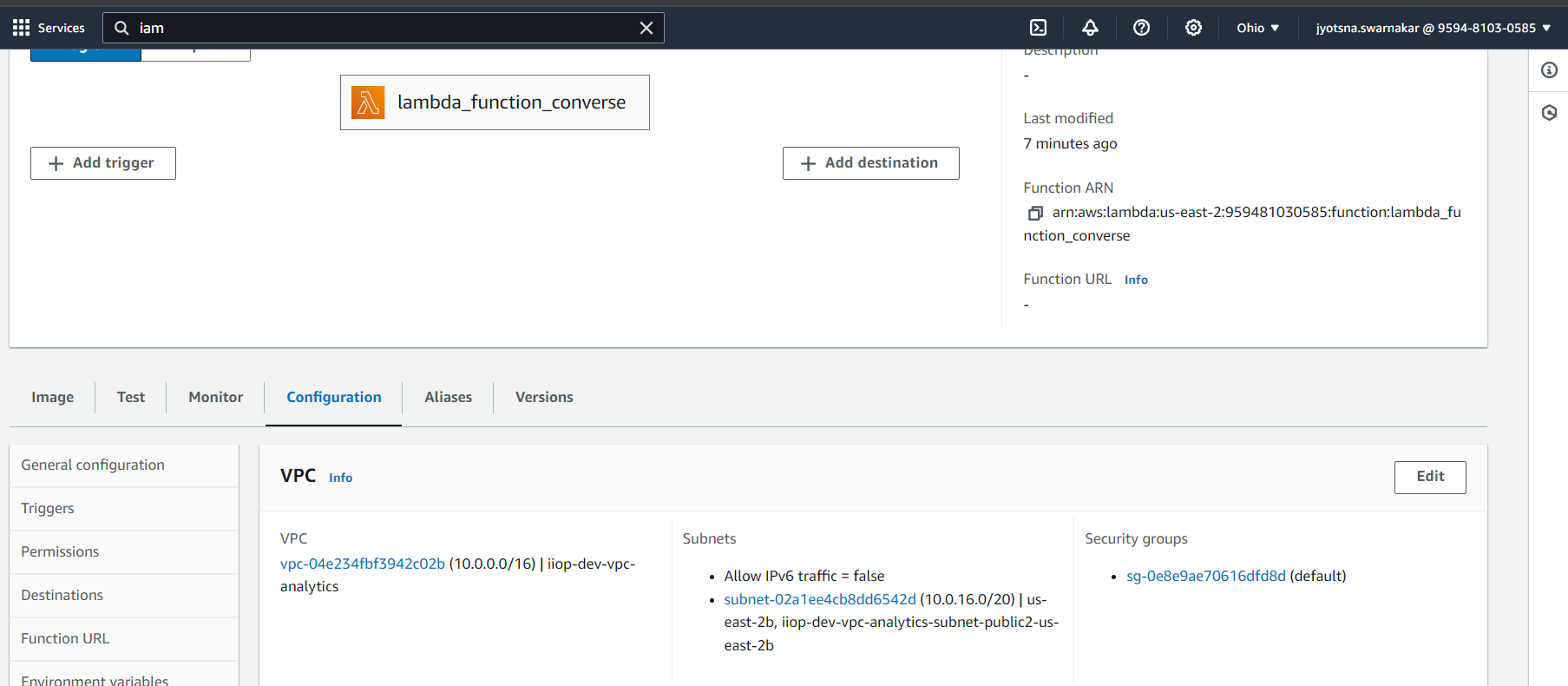
* Go to lambda service -> In the left navigation pane click on “Functions” -> Click on create function -> Choose “container image” , function name : “**lambda\_function\_converse**”, - > Click Browse images, select “lambda” repository from the drop down, select the image and click on select image -> Expand “Change default execution role” , select use an existing role and select “LambdaExec” role from the drop down -> Expand “Advanced settings” and create tag (project : analytics) -> Click create function

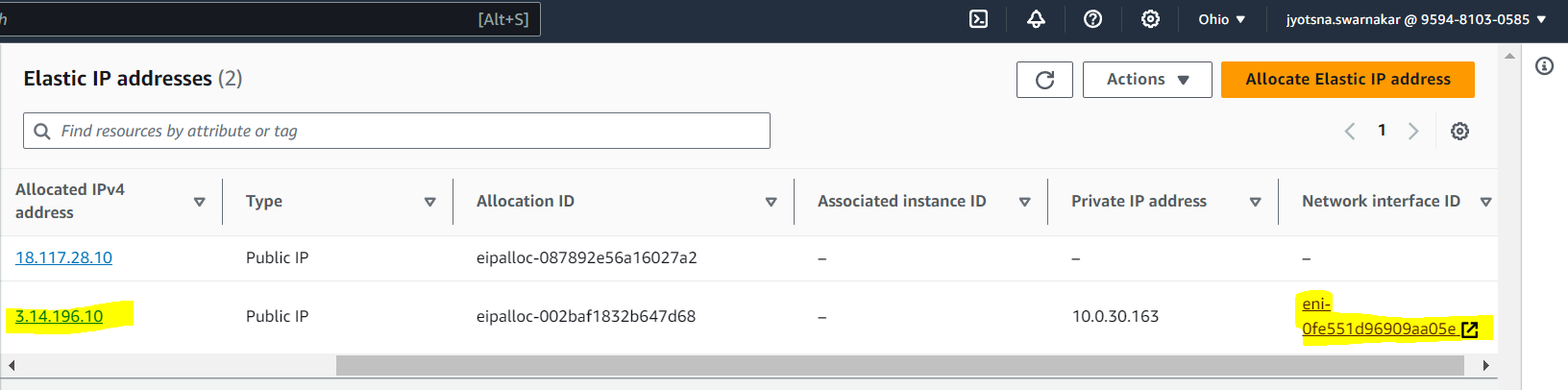


* Repeat the above step for creating “lambda\_function\_vector” function but in the CMD override mention “**lambda\_function\_vector.lambda\_handler”**



* After successful creation of “lambda\_function\_converse” function, Navigate to its Configuration tab, Click General Configuration, click on edit and add Memory(4096 MB), Ephemeral storage(4096 MB) & Timeout(1 min 0 sec). Click VPC , Click Edit, Choose created VPC and private subnet(subnet-06365f802852cb414) from the drop down, choose the default Security Group created while creating the VPC and click save.





* Now create a Test Event for the Lambda Function(lambda\_function\_converse) with the below sample user prompt in JSON format to test if it is successfully executing as expected

{

  "userQuery": "",

  "sourcePlatform": "web",

  "preferredFormat": "html",

  "userName": "Gaurangs",

  "companyGuid": "6c50a17b-45b1-4e38-a0e5-4eb5a422e926",

  "s3Url": "s3://dev-data-and-gen-ai/audio-files/android-app-audio-1923691540.m4a"

}

* Also, check the logs by navigating to the Cloudwatch log group([/aws/**lambda**/lambda\_function\_converse](https://us-east-2.console.aws.amazon.com/cloudwatch/home?region=us-east-2#logsV2:log-groups/log-group/$252Faws$252Flambda$252Flambda_function_converse)) for lambda\_function\_converse
* After successful creation of “lambda\_function\_vector” function, Navigate to its Configuration tab , Click General Configuration, click on edit and add Memory(4096 MB), Ephemeral storage(4096 MB) & Timeout(1 min 0 sec)

**For Deploying the Docker images to already running Lambda functions via CI/CD pipeline , follow the below steps**

* Create a multibranch pipeline job in Jenkins as below:

1. In the Jenkins dashboard, click on "New Item" to create a new Jenkins job
2. Give job a name, such as "githubpush"
3. Choose "Multibranch Pipeline" as the type of project
4. Scroll down to the configuration section for the multibranch pipeline job.
5. Choose your version control system as Git and provide the repository URL(<https://github.com/guardhat/gen-ai>)
6. Configure how Jenkins should scan for branches and pull requests. We can choose to scan for changes periodically or trigger scans manually.
7. Configure the branch source(gen-ai) from which Jenkins will create Pipeline jobs
8. Define the Jenkinsfile location
9. Once configured all the settings for your multibranch pipeline job, click on "Save" to create the job.
10. Jenkins will automatically scan the repository for branches and pull requests and create Pipeline jobs for each branch it finds. We can also manually trigger a scan if needed.

* Create a Jenkins file with the below contents and push it to the Github Repository
* Push the Dockerfile and the python files config.py, constant.py, lambda\_converse\_function.py and lambda\_vector\_fuction.py to the Github Repository
* As soon as the python files are pushed to the github repository, it will trigger the Jenkins job which will first execute the Dockerfile to create the Docker image and push it to the ECR Repository. Once the ECR Repository will receive the image, it will automatically deploy the images to both the lambda functions

# Security Overview

* The ODBC connection details of Azure SQL database are stored in AWS Secrets Manager and the secrets keys are used in the code
* Used Encrypt=yes and TrustServerCertificate=yes for encryption of data between Azure SQL database and Gen AI application in the code. The TrustServerCertificate parameter enables TLS to secure data in transit.
* Deployed the lambda function *lambda\_converse\_function* in private subnet ensuring the code is not available publicly
* NAT gateway is deployed in public subnet with static public IP for Lambda function for secure inbound and outbound access
* Attached Security Group to the Lambda Function has inbound rule added for only the specific VPC CIDR’s “10.0.0.0/16, 172.31.0.0/16”
* We have whitelisted the NAT Gateway public IP in the Azure SQL server firewall, so that the ODBC connection between lambda function and Azure SQL database is made through NAT Gateway only
* Enabled CloudWatch logs for logging all the access logs for the Lambda functions.
* The API uses dedicated service credentials for authentication. These credentials must be securely stored and managed to prevent unauthorized access. Each service credential pair includes a username and password required for accessing the API
* The API integrates with Key cloak for managing authentication and authorization that supports Single Sign-On (SSO)
* Authentication is performed using OAuth tokens issued by the Key Cloak server. Clients must obtain an OAuth token by providing their credentials and use this token in the Authorization header of their API requests. The token ensures that only authorized users can access the API resources
* The API is deployed in a secure environment where version 2.0 of the application is hosted. This environment is configured with appropriate security measures, including firewalls and to protect against vulnerabilities and unauthorized access