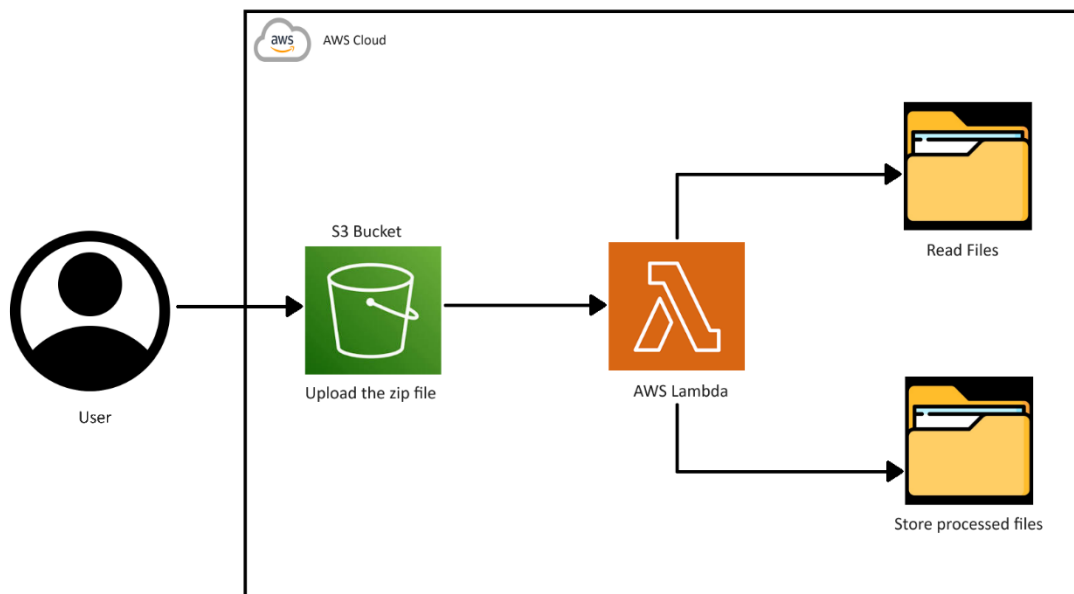


Task 2 : Python and Lambda

Work with RXNORM file,

1. Scrap the latest RXNORM file from NLM webpage
2. Download the latest RXNORM file with api_key
3. Create a log file for the downloaded file
4. While uploading the file to S3, create CloudWatch log file
5. Add header into each rff from RXNORM.xlsx
6. Add CODE_SET & VERSION_MONTH column with default values RxNorm and version month from downloaded filename
7. Convert dates into YYYY-MM-DD
8. Save files as txt delimited by comma(,)c
9. Validate row_count between original and converted files

Cloud Architecture of this Task



First we create a Lambda Function with following configurations:

The screenshot shows the 'Create function' page in the AWS Lambda console. The 'Author from scratch' option is selected. The 'Basic information' section is expanded, showing the following configurations: Function name: 'vrt_lambda', Runtime: 'Python 3.12', Architecture: 'x86_64', and Permissions: 'Change default execution role'. The 'Execution role' section shows 'Use an existing role' selected, with 'Lambda' chosen from the dropdown. The 'Advanced settings' section is collapsed. At the bottom right, there are 'Cancel' and 'Create function' buttons.

Secondly, we need to add Layers to our Lambda.

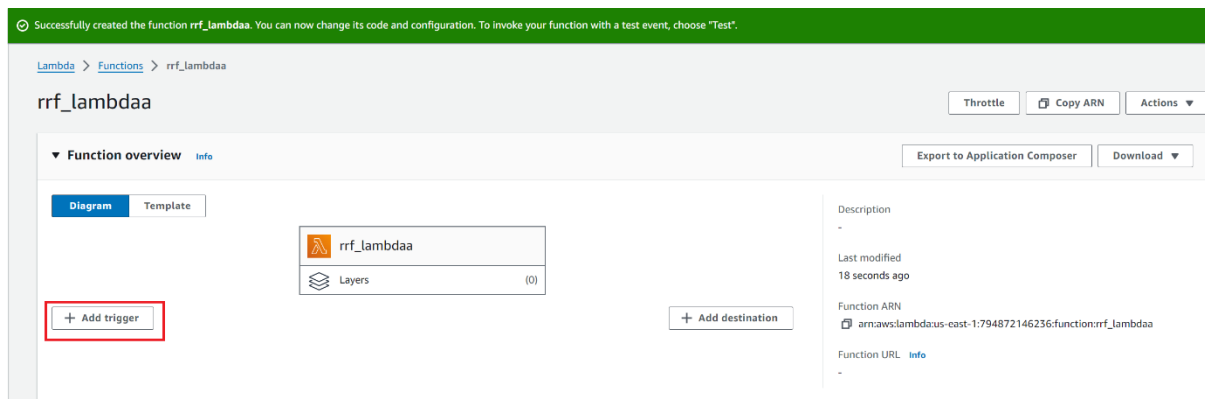
Open our newly created Lambda function, and scroll down. In the bottom of the page, we can see Layers tab. Here, we need to Add a Layer.

The screenshot shows the 'Layers' tab in the AWS Lambda console. It has a table with columns: 'Merge order', 'Name', 'Layer version', 'Compatible runtimes', 'Compatible architectures', and 'Version ARN'. The table is currently empty, with the text 'There is no data to display.' below it. At the top right, there are 'Edit' and 'Add a layer' buttons. The 'Add a layer' button is highlighted with a red rectangle.

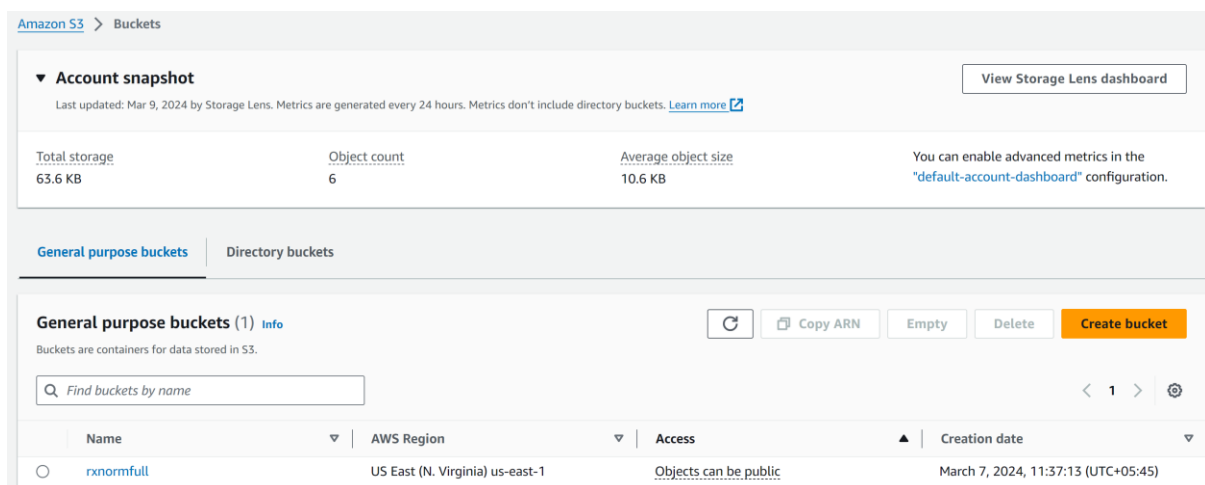
Here, select AWSSDKPandas-Python312 and click Add. Also choose Version as 4.

The screenshot shows the 'Add layer' dialog box in the AWS Lambda console. The 'Function runtime settings' section shows 'Runtime: Python 3.12' and 'Architecture: x86_64'. The 'AWS provided' section lists several layers, with 'AWSSDKPandas-Python312' highlighted by a red rectangle. To the right, there is a section for 'Specify the Amazon Resource Name (ARN) of a layer' with a radio button for 'Specify an ARN'. Below the list of layers, there is a search bar with the text 'Choose' and an upward arrow. At the bottom right, there are 'Cancel' and 'Add' buttons. The 'Add' button is highlighted with a red rectangle.

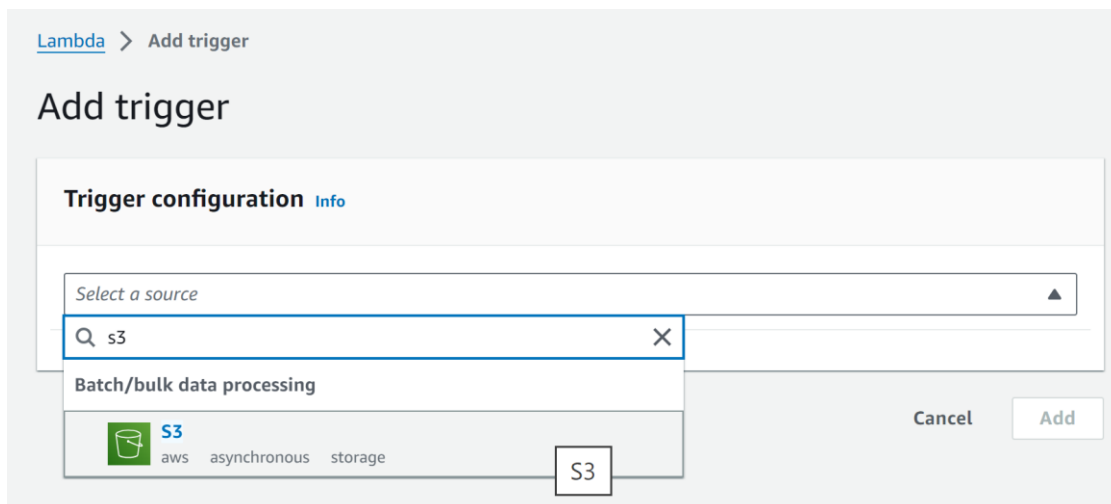
Next step, is to add a Trigger to our Lambda



We need to create S3 bucket for this. I have created a bucket.




Now go back to Lambda and Select S3 in Trigger Configuration



Lambda > Add trigger

Add trigger

Trigger configuration [Info](#)

 **S3**
aws asynchronous storage

Bucket
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.
 × ↺
Bucket region: us-east-1

Event types
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events ×

Prefix - optional
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

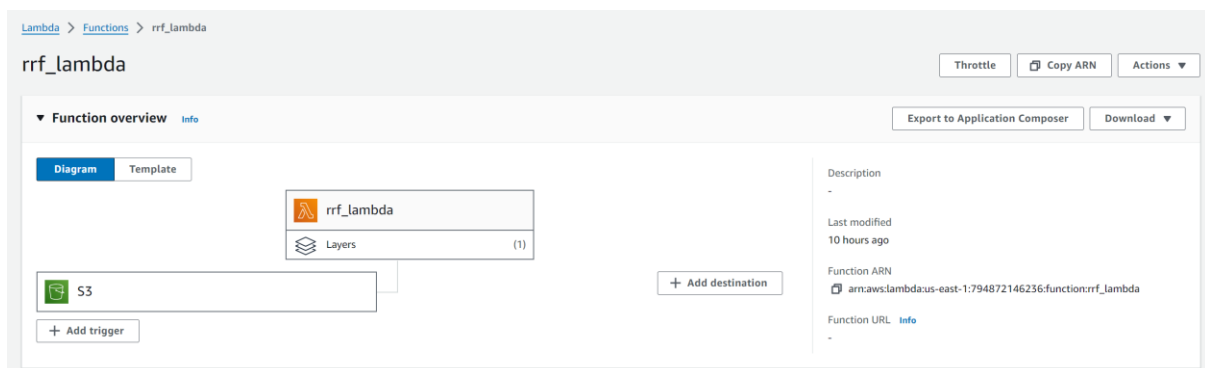
Suffix - optional
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)
☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

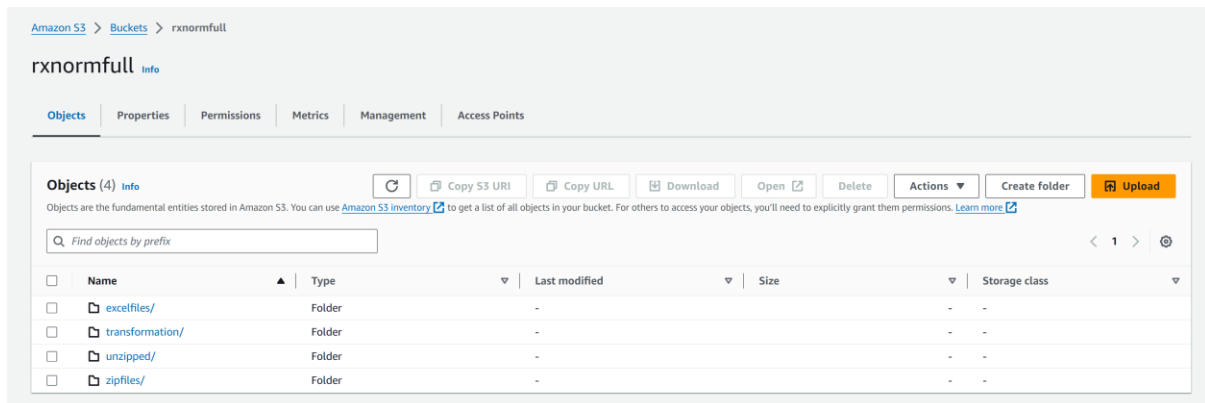
Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

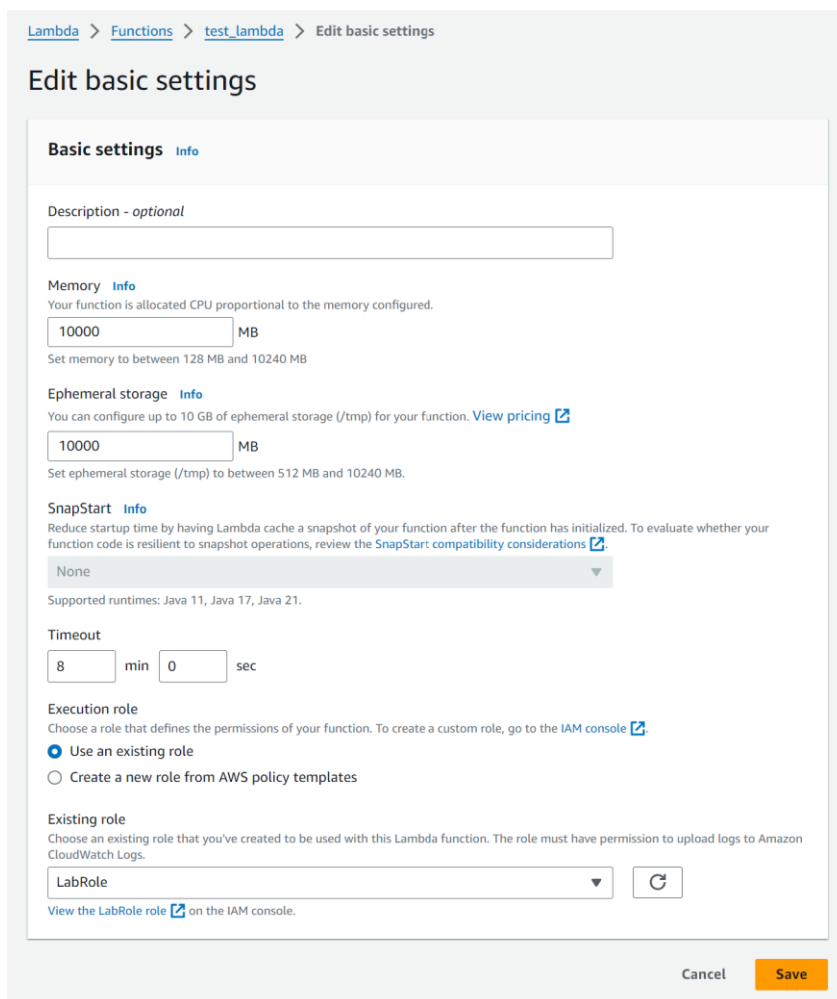
Now this is how our Lambda looks



In our bucket, two folders i.e, excelfiles/ and zipfiles/ are pre-created whereas transformations/ and unzippedfiles/ are created after the lambda is trigger and files are read



We have to



We have to import the following:

```
lambda_function x Environment Var x +
1 import boto3
2 import zipfile
3 import io
4 import os
5 import pandas as pd
6 import json
7 import openpyxl
8 from io import BytesIO
9
```

excel_headers holds the headers to be assigned to the rrf files

The following function (read_excel_from_s3) reads the excel files and updates the header

```
lambda_function x Environment Var x +
15 def read_excel_from_s3(bucket):
16     try:
17         folder_path = 'excelfiles/'
18         excel_file_name = 'RxNorm_Header.xlsx'
19         key = folder_path + excel_file_name
20
21         # Download the Excel file to the /tmp directory
22         local_excel_file = '/tmp/RxNorm_Header.xlsx'
23         s3.download_file(bucket, key, local_excel_file)
24
25         # Check if the Excel file exists
26         if os.path.exists(local_excel_file):
27             print(f"Excel file downloaded to: {local_excel_file}")
28
29
30
31
32         # Read the Excel file into an ExcelFile object
33         excel_file = pd.ExcelFile(local_excel_file)
34
35         # Get the sheet names
36         sheet_names = excel_file.sheet_names
37         print("Sheet names:", sheet_names)
38
39         for sheets in sheet_names:
40             # Read the data from the sheet into a DataFrame
41             sheets_data = excel_file.parse(sheets, header=None)
42             headers_data = sheets_data.iloc[:, 0].tolist()
43             excel_headers[sheets] = headers_data
44             print(f"excel_headers dictionary for sheet {sheet_names[0]}: {excel_headers[sheet_names[0]]}")
52.9 Python Spaces: 4
```

The following function incorporate the Code Set and Version Month: The Code Set has been designated as RXNORM, and the Version Month has been derived from the zip file name, specifically RxNorm_full_02052024.zip. As a result, the Version Month is identified as 2024-05-02.

Secondly, apply_header_to_rrf adds header to rrf files.

```
lambda_function x Environment Var x +
51 def code_set_and_version_month(zip_filename, rrf_df):
52     try:
53         # Convert the zip data to a string to extract information
54         version_month = os.path.splitext(zip_filename)[0].split('_')[-1]
55
56         # Convert version month to a more readable format
57         version_month = pd.to_datetime(version_month, format='%m%d%Y').strftime('%Y-%m-%d')
58         print(f"Version month: {version_month}")
59
60         # Add 'Code Set' and 'Version Month' columns to the DataFrame
61         rrf_df['Code Set'] = 'RxNorm'
62         rrf_df['Version Month'] = version_month
63     except Exception as e:
64         print(f"Error occurred while extracting version month: {e}")
65
66     return rrf_df
67
68
69 def apply_header_to_rrf(file_name, rrf_df):
70     # Check if the corresponding Excel sheet exists
71     if file_name in excel_headers:
72         # Get the headers from the Excel sheet
73         excel_headers_list = excel_headers[file_name]
74         excel_headers_list = [header for header in excel_headers_list if header != 'SVER']
75         # Take names from the excel header list up to the length of the split DataFrame
76         excel_headers_list = excel_headers_list[:len(rrf_df.columns)]
77         # Set the correct header for the DataFrame
78         rrf_df.columns = excel_headers_list
79     return rrf_df
80
45.5 Python Spaces: 4
```

The following function is employed to transform dates into the specified format. In the case of RXNSAB, the SVER column obtains its value by extracting the year component from the VSTART column, as illustrated below:

```

lambda_function Environment Var
80
81 def convert_date_format(value):
82
83     try:
84         # Try to parse the value into datetime format
85         parsed_date = pd.to_datetime(value, format='%Y_%m_%d').date()
86         # Extract only the date part
87         return parsed_date.strftime('%Y-%m-%d')
88     except ValueError:
89         if value == '2020':
90             return '2020-01-01'
91         elif value == '5.0_2024_01_04':
92             # Remove the float value and parse the remaining string
93             return convert_date_format('2024_01_04')
94         elif value == '2020AA':
95             return '2024-01-02'
96         elif value == '20AA_240205F':
97             return '2024-02-05'
98         else:
99             return value
100
101

```

The following function handles different date formats of RXNATOMARCHIVE as below:

```

lambda_function Environment Var
103 def update_nato_date(value):
104     try:
105         # Attempt to parse the value using the first date format
106         parsed_date = pd.to_datetime(value, format='%m/%d/%Y %I:%M:%S %p').date()
107     except ValueError:
108         try:
109             # If the first format fails, attempt to parse using the second date format
110             parsed_date = pd.to_datetime(value, format='%d-%b-%y').date()
111         except ValueError:
112             # If both formats fail, return None or handle the error appropriately
113             return None # Or handle the error appropriately
114         # Check if the parsed date is NaT
115         if pd.isnull(parsed_date):
116             return '0000-00-00' # Replace NaT with '0000-00-00'
117         else:
118             # Extract only the date part and return it in the desired format
119             return parsed_date.strftime('%Y-%m-%d')
120

```

The following function processes date columns

```

lambda_function Environment Var
121
122 def process_date_columns(file_name, rrf_df):
123     date_columns = ['VSTART', 'VEND', 'CREATED_TIMESTAMP', 'UPDATED_TIMESTAMP', 'LAST_RELEASED']
124     for column in date_columns:
125         if column in rrf_df.columns:
126             if file_name == 'RXNSAB':
127                 # Apply the conversion function to each value in the column
128
129                 rrf_df[column] = rrf_df[column].apply(convert_date_format)
130                 # Extract year from the VSTART column after date conversion and save it directly as a string
131                 rrf_df['SVER'] = pd.to_datetime(rrf_df['VSTART'], format='%Y-%m-%d').dt.year.astype(str)
132                 # Reorder the columns to place 'SVER' before 'VSTART'
133                 # Reorder columns
134                 sver_index = rrf_df.columns.get_loc('SVER')
135                 vstart_index = rrf_df.columns.get_loc('VSTART')
136                 sf_index = rrf_df.columns.get_loc('SF')
137
138                 # Remove 'SVER' from its original position
139                 column_sver = rrf_df.pop('SVER')
140
141                 # Insert 'SVER' after 'SF', before 'VSTART'
142                 if sver_index < vstart_index:
143                     rrf_df.insert(vstart_index - 1, 'SVER', column_sver)
144                 elif sver_index > vstart_index:
145                     rrf_df.insert(vstart_index, 'SVER', column_sver)
146                 if file_name == 'RXNATOMARCHIVE':
147                     rrf_df[column] = rrf_df[column].apply(update_nato_date)
148
149     return rrf_df
150

```

The following code saves the converted file to .txt

```

153 def save_as_txt_file(rrf_df, file_name, bucket_name):
154     # Construct the filename for the output text file
155     transformation_folder = 'transformation/'
156
157     # Convert DataFrame to CSV format in memory
158     csv_buffer = io.StringIO()
159     rrf_df.to_csv(csv_buffer, sep='|', index=False)
160
161     # Upload the CSV buffer to S3
162     s3_key = transformation_folder + file_name + '.txt'
163     s3.put_object(Bucket=bucket_name, Key=s3_key, Body=csv_buffer.getvalue())
164
165     print(f"Transformed data saved to: s3://{bucket_name}/{s3_key}") #
166
167
168

```

The primary function where the preceding functions are invoked: Here, zip archives are retrieved from the correct directory, and the final pipe delimiter is disregarded by segmentation.

```

168
169
170 def read_and_relocate_rrf_files(s3, bucket, key):
171     try:
172         zip_response = s3.get_object(Bucket=bucket, Key=key)
173         zip_data = zip_response['Body'].read()
174         zip_filename = os.path.basename(key)
175
176         # Wrap the zip data in a BytesIO object
177         zip_file = BytesIO(zip_data)
178
179         file_path = 'test_full_02052024/rrf'
180
181         with zipfile.ZipFile(zip_file, 'r') as zip_ref:
182             for file_info in zip_ref.infolist():
183                 if file_info.filename.startswith(file_path) and not file_info.filename.endswith('/'):
184                     filename = os.path.basename(file_info.filename)
185                     print(f"The {filename} is read from zip file.")
186
187                     with zip_ref.open(file_info) as source_file:
188                         file_content = source_file.read().decode('utf-8')
189                         if file_content.endswith('|'):
190                             file_content = file_content[:-1]
191                             file_content_io = io.StringIO(file_content)
192                             rrf_df = pd.read_csv(file_content_io, delimiter='|', header=None)
193                             rrf_df = rrf_df.iloc[:, :-1]
194                             print(f"Row count before transformation: {rrf_df.shape[0]}")
195
196                             file_name = os.path.splitext(filename)[0]
197                             apply_header_to_rrf(file_name, rrf_df)
198                             rrf_df = process_date_columns(file_name, rrf_df)
199                             code_set_and_version_month(zip_filename, rrf_df)
200
201                             print(f"Row count of {file_name} after transformation: {rrf_df.shape[0]}")
202
203                             pd.set_option('display.max_columns', None)
204                             print(rrf_df.head(5))
205
206                             # Save the transformed DataFrame to a text file
207                             save_as_txt_file(rrf_df, file_name, bucket)
208
209                             # Upload the unzipped file to the 'unzipped' folder
210                             unzipped_key = 'unzipped/' + filename
211                             s3.put_object(Bucket=bucket, Key=unzipped_key, Body=file_content)
212                             print(f"Unzipped file saved to: s3://{bucket}/{unzipped_key}")
213     except Exception as e:
214         print(f"Error occurred: {e}")
215

```

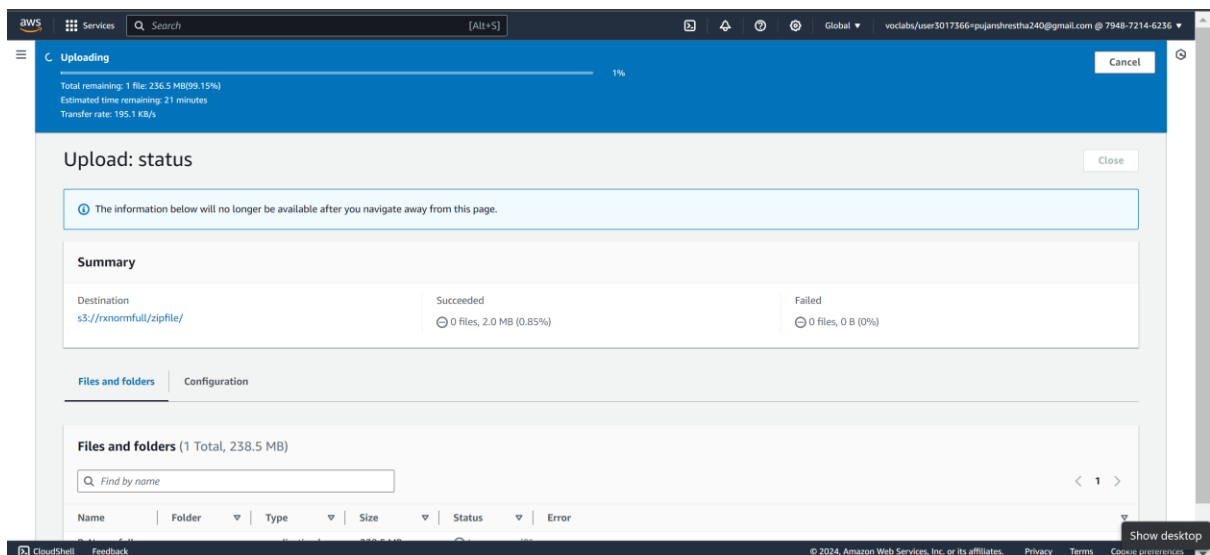
The following is the lambda_handler

```

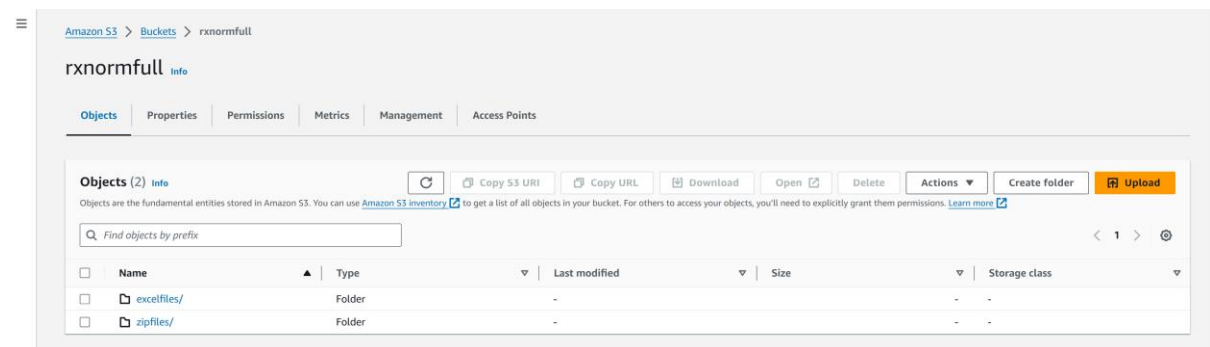
217 def lambda_handler(event, context):
218     bucket = event['Records'][0]['s3']['bucket']['name']
219     key = event['Records'][0]['s3']['object']['key']
220
221
222     # This is the function that relocate the rrf files from zip file
223     read_excel_from_s3(bucket)
224     read_and_relocate_rrf_files(s3,bucket,key)
225

```

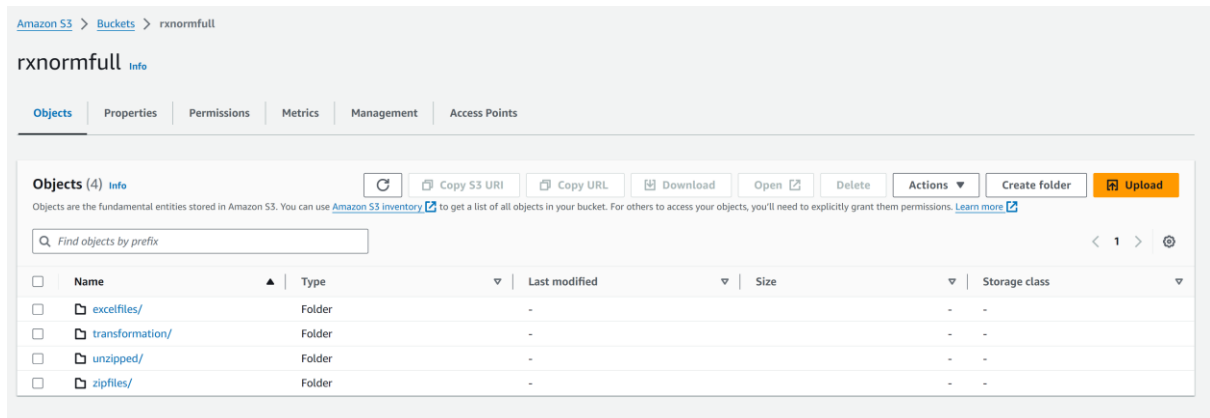
Uploading zip file



Initial rxnorm



After lambda is triggered:



excelfiles/

Amazon S3

>

Buckets

>

rxnormfull

>

excelfiles/

excelfiles/

Copy S3 URI

Objects

Properties

Objects (1)

Info

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

< 1 > ⚙

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	RxNorm_Header.xlsx	xlsx	March 10, 2024, 22:53:55 (UTC+05:45)	27.9 KB	Standard

zippedfiles/

Amazon S3

>

Buckets

>

rxnormfull

>

zipfiles/

zipfiles/

Copy S3 URI

Objects

Properties

Objects (1)

Info

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

< 1 > ⚙

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	RxNorm_full_02062024.zip	zip	March 11, 2024, 11:25:48 (UTC+05:45)	14.1 MB	Standard

Unzippedfiles/

Amazon S3

>

Buckets

>

rxnormfull2

>

unzipped/

unzipped/

Copy S3 URI

Objects

Properties

Objects (9)

Info

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

< 1 > ⚙

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	RxNATOMARCHIVE.RRF	RRF	March 11, 2024, 15:15:28 (UTC+05:45)	71.4 MB	Standard
<input type="checkbox"/>	RXNCONSO.RRF	RRF	March 11, 2024, 15:15:44 (UTC+05:45)	118.6 MB	Standard
<input type="checkbox"/>	RXNCUI.RRF	RRF	March 11, 2024, 15:15:45 (UTC+05:45)	1.7 MB	Standard
<input type="checkbox"/>	RXNCUICHANGES.RRF	RRF	March 11, 2024, 15:15:45 (UTC+05:45)	14.9 KB	Standard
<input type="checkbox"/>	RXNDOC.RRF	RRF	March 11, 2024, 15:15:45 (UTC+05:45)	214.2 KB	Standard
<input type="checkbox"/>	RXNREL.RRF	RRF	March 11, 2024, 15:15:09 (UTC+05:45)	484.4 MB	Standard
<input type="checkbox"/>	RXNSAB.RRF	RRF	March 11, 2024, 15:15:15 (UTC+05:45)	9.8 KB	Standard
<input type="checkbox"/>	RXNSAT.RRF	RRF	March 11, 2024, 15:16:06 (UTC+05:45)	498.7 MB	Standard
<input type="checkbox"/>	RXNSTY.RRF	RRF	March 11, 2024, 15:16:13 (UTC+05:45)	18.4 MB	Standard

Transformations/

transformation/

Copy S3 URI

Objects Properties

Objects (9) info

Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

< 1 > ⚙

<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class ▼
<input type="checkbox"/>	RXNATOMARCHIVE.txt	txt	March 11, 2024, 15:15:27 (UTC+05:45)	69.5 MB	Standard
<input type="checkbox"/>	RXNCONSO.txt	txt	March 11, 2024, 15:15:43 (UTC+05:45)	138.1 MB	Standard
<input type="checkbox"/>	RXNCUI.txt	txt	March 11, 2024, 15:15:45 (UTC+05:45)	2.1 MB	Standard
<input type="checkbox"/>	RXNCUICHANGES.txt	txt	March 11, 2024, 15:15:45 (UTC+05:45)	17.8 KB	Standard
<input type="checkbox"/>	RXNDOC.txt	txt	March 11, 2024, 15:15:45 (UTC+05:45)	271.6 KB	Standard
<input type="checkbox"/>	RXNREL.txt	txt	March 11, 2024, 15:16:55 (UTC+05:45)	645.2 MB	Standard
<input type="checkbox"/>	RXNSAB.txt	txt	March 11, 2024, 15:17:08 (UTC+05:45)	10.3 KB	Standard
<input type="checkbox"/>	RXNSAT.txt	txt	March 11, 2024, 15:15:58 (UTC+05:45)	621.9 MB	Standard
<input type="checkbox"/>	RXNSTY.txt	txt	March 11, 2024, 15:16:13 (UTC+05:45)	26.0 MB	Standard