Workflow Steps:

1. `files_in_blob_list_creation`

- **Purpose:** To list all files in the Azure Blob Container for a specific directory (Eg: jazz/)
- **Operation:** Generates a CSV file named "files_in_blob.csv".
- **Output Columns:** "Blob Name", "Blob URI".
- **Details:** The function accesses the Azure Blob Container, retrieves the names and URIs of all files, and stores this information in a CSV.

2. `files_indexed_creation`

- **Purpose:** To list all files currently indexed in Azure Cognitive Search.
- **Operation:** Generates a CSV file named "files_indexed.csv".
- **Output Columns:** "id", "Blob URI".
- **Details:** The function communicates with Azure Cognitive Search, fetching the IDs and URIs of indexed files, and then writes this data into a CSV.

3. `load_csv_into_set (for files_in_blob.csv)`

- **Purpose:** To read "files_in_blob.csv" and load it into a set.
- **Operation:** Reads URIs from "files_in_blob.csv".
- **Output:** A set named `blob_files_set` containing unique Blob URIs.

4. `load_csv_into_set (for files_indexed.csv)`

- **Purpose:** To read "files_indexed.csv" and load it into a set.
- **Operation:** Reads URIs from "files_indexed.csv".
- **Output:** A set named `indexed_files_set` with unique Blob URIs that are already indexed.

5. Calculate Set Differences:

- **Purpose:** Identify new files to index and old files to delete.
- **Operation:**
  - `uri_of_files_to_index = blob_files_set - indexed_files_set` : Finds new files in Blob.
  - `uri_of_files_to_delete = indexed_files_set - blob_files_set` : Identifies files that have been removed from Blob.

6. For Files in Index to be Deleted

- **Condition:** Execute if `uri_of_files_to_delete` is not empty.
- **Sub-steps:**
  - `get_ids_for_uris_to_delete(uri_of_files_to_delete)`
    - Retrieves IDs for URIs listed in `uri_of_files_to_delete`.
  - `delete_docs_in_index_by_uri(uri_of_files_to_delete)`
    - Retrieves IDs using `get_ids_for_uris_to_delete(uri)`.
    - Initiates document deletion from Azure index using `doc_deletion_in_index_by_id(document_id)` for each ID.
    - Provides deletion status feedback.

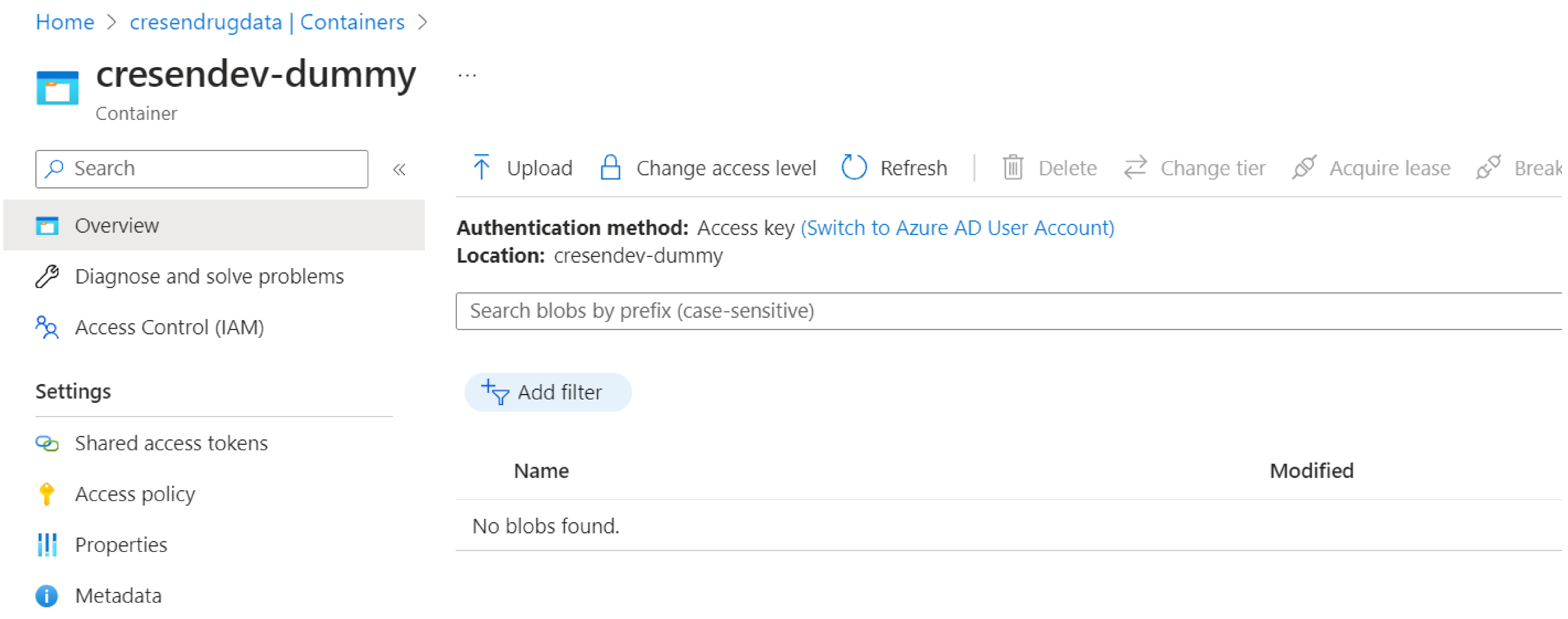
7. For Files to be Indexed:

- **Condition:** Execute if `uri_of_files_to_index' not empty
- **Sub-steps:**
  - **a.** **`blob_to_text_processing_with_uri_filter(connect_str, container_name, uri_of_files_to_index)`**
    - Processes text from the new Blob files intended for indexing, filtering by URI.
    - It will print statistics including the number of files, characters, tokens, and the cost of embedding for the loaded documents.
  - **b.** **`splitting_function(documents)`**
    - Splits the processed documents into smaller chunks (based on token or character count) with specified overlap.
    - This step is mandatory to handle large documents and facilitate more efficient indexing.
  - **c.** **`docs_to_index_ingestion(split_documents)`**
    - This function will take the split documents, initialize and configure OpenAI and Azure Search settings, then proceed to index the split documents into Azure Search.

In [8]:
```python
# 1. files_in_blob_list_creation
#enter credentials
account_name = 'cresendrugdata'
account_key = 'Hy4tA8wAO+6pOHFGIPN9YPUyEiBq+6qk7UY0Ew+vebWSCcmKelWkopuQVaonEL4258E5EljnFlvp+AStRNdAoA=='

#create a client to interact with blob storage
connect_str = 'DefaultEndpointsProtocol=https;AccountName=' + account_name + ';AccountKey=' + account_key + ';EndpointSuffix=core.windows.net'

from azure.storage.blob import BlobServiceClient
import os
import csv

def files_in_blob_list_creation(container_name, directory_name, account_name, connection_string):
    """
    List all blobs in a specified directory within a container and write their URI and name to a CSV file.

    Parameters:
    container_name (str): Azure Storage Container Name
    directory_name (str): Directory path within the blob container
    """
    try:
        # Get the Blob Service Client
        blob_service_client = BlobServiceClient.from_connection_string(connection_string)

        # Get the Container Client
        container_client = blob_service_client.get_container_client(container_name)

        # Create or open CSV file for writing
        with open('files_in_blob.csv', mode='w', newline='') as file:
            writer = csv.writer(file)
            # Write header row
            writer.writerow(["Blob Name", "Blob URI"])

            # List blobs in the specified directory
            blob_list = container_client.list_blobs(name_starts_with=directory_name)
            for blob in blob_list:
                # Construct blob URI
                blob_uri = f"https://{account_name}.blob.core.windows.net/{container_name}/{blob.name}"
                # Write blob name and URI to the CSV file
                writer.writerow([blob.name, blob_uri])

    except Exception as e:
        print(f"An exception occurred: {e}")

# Usage example:
```

```python
#files_in_blob_list_creation('cresendev-dummy', 'jazz', account_name, connect_str)

# 2. files_indexed_creation
# Configure Vector Store Settings
import os
from dotenv import load_dotenv
load_dotenv()
vector_store_address: str = os.getenv("AZURE_SEARCH_SERVICE_ENDPOINT")
vector_store_password: str = os.getenv("AZURE_SEARCH_ADMIN_KEY")
service_name = 'cresencognitivesearch'
index_name: str = "cresendev-dummy"

import pandas as pd
import requests
import json

# Set up the global variables
service_name = 'cresencognitivesearch'
api_version = '2020-06-30'
admin_key = vector_store_password  # make sure vector_store_password is defined

def files_indexed_creation(index_name: str):
    """
    Function to create a CSV with indexed files information.

    Parameters:
    index_name (str): Name of the index.

    Returns:
    None: Writes a CSV file named files_indexed.csv.
    """

    # Create the URL for your search index
    url = f"https://{service_name}.search.windows.net/indexes/{index_name}/docs?api-version={api_version}&$count=true&$select=id,metadata"

    # Set up the headers with the API key
    headers = {
        "Content-Type": "application/json",
        "api-key": admin_key
    }

    # Perform the GET request to retrieve the search results
    response = requests.get(url, headers=headers)

    if response.status_code != 200:
        print(f"Error in API request, status code: {response.status_code}")
        return

    # Assuming 'data' is the parsed JSON response
    data = response.json()

    # Extracting the 'value' field which contains the list of documents
    documents = data.get('value', [])

    # Creating lists for 'id' and 'url' to be used for creating the dataframe
    ids = []
    urls = []

    # Iterating over the documents and extracting 'id' and 'url' from the 'metadata'
    for doc in documents:
        ids.append(doc['id'])
        # Parsing the 'metadata' as JSON (since it's a string in the given data) and extracting 'url'
        metadata = json.loads(doc['metadata'])
        urls.append(metadata['url'])

    # Creating a dataframe with 'id' and 'url' as columns
    df = pd.DataFrame(list(zip(ids, urls)), columns=['id', 'Blob URI'])

    # Write the dataframe to a CSV file
    df.to_csv("files_indexed.csv", index=False)

# usage
# files_indexed_creation("cresendev-dummy")

# 3. load_csv_into_set (for files_in_blob.csv)
import pandas as pd

def load_csv_into_set(file_name: str, column_name: str) -> set:
    """
    Load specific column from a CSV file into a set.

    Parameters:
    file_name (str): Name of the CSV file to read.
    column_name (str): Name of the column whose values need to be loaded into a set.

    Returns:
    set: A set of unique values from the specified column in the CSV file.
    """
    try:
        # Load the CSV file into a DataFrame
        df = pd.read_csv(file_name)

        # Check if the column exists in the DataFrame
        if column_name not in df.columns:
            print(f"Column '{column_name}' not found in {file_name}")
            return set()

        # Convert the specified column into a set and return
        return set(df[column_name])

    except FileNotFoundError:
        print(f"File {file_name} not found.")
        return set()

    except Exception as e:
        print(f"An error occurred: {e}")
        return set()

# blob_files_set = load_csv_into_set("files_in_blob.csv", "Blob URI")

# 4. load_csv_into_set (for files_indexed.csv)
#indexed_files_set = load_csv_into_set("files_indexed.csv", "Blob URI")

# 5. Calculate Set Differences
#uri_of_files_to_index = blob_files_set - indexed_files_set # Finds new files in Blob.
#uri_of_files_to_delete = indexed_files_set - blob_files_set # Identifies files that have been removed from Blob.

# 6. For Files in Index to be Deleted
import pandas as pd

def get_ids_for_uris_to_delete(uri_of_files_to_delete: set) -> set:
    """
    Function to retrieve IDs corresponding to the given set of URIs from files_indexed.csv.

    Parameters:
    uri_of_files_to_delete (set): A set of URIs of files to be deleted.

    Returns:
    set: A set of IDs corresponding to the URIs to be deleted.
    """
    # Read the CSV file into a pandas DataFrame
    try:
        df = pd.read_csv('files_indexed.csv')
        # Filter the DataFrame to keep rows where 'Blob URI' is in uri_of_files_to_delete
        filtered_df = df[df['Blob URI'].isin(uri_of_files_to_delete)]

        # Return the set of IDs from the filtered DataFrame
        return set(filtered_df['id'])
    except FileNotFoundError:
        print("Error: 'files_indexed.csv' not found.")
        return set()
    except Exception as e:
        print(f"An error occurred: {e}")
        return set()

import requests
import json
import os
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Configure Vector Store Settings
vector_store_address: str = os.getenv("AZURE_SEARCH_SERVICE_ENDPOINT")
vector_store_password: str = os.getenv("AZURE_SEARCH_ADMIN_KEY")
service_name = 'cresencognitivesearch'
index_name: str = "cresendev-dummy"

def doc_deletion_in_index_by_id(document_id, service_name=service_name,
                                index_name=index_name, api_key=vector_store_password,
                                api_version="2020-06-30"):
    """
    Delete a document from Azure Cognitive Search by document ID.

    Parameters:
    - document_id (str): The ID of the document to delete.
    - service_name (str): Name of your search service. Default is pre-defined service_name.
    - index_name (str): Name of your index. Default is pre-defined index_name.
    - api_key (str): Your admin API key. Default is pre-defined vector_store_password.
    - api_version (str): Azure API version. Default is "2020-06-30".

    Returns:
    - Response object.
    """

    headers = {
        'Content-Type': 'application/json',
        'api-key': api_key
    }

    url = f'https://{service_name}.search.windows.net/indexes/{index_name}/docs/index?api-version={api_version}'

    # Preparing the data payload for delete action
    data_payload = {
        "value": [
            {
                "@search.action": "delete",
                "id": document_id  # Replace 'id' with the name of the key field in your index if it's different
            }
        ]
    }

    response = requests.post(url, headers=headers, data=json.dumps(data_payload))
    return response

import requests
import json

def delete_docs_in_index_by_uri(uri_of_files_to_delete: set):
    """
    Function to delete documents from Azure Cognitive Search index based on a set of URIs.

    Parameters:
    uri_of_files_to_delete (set): A set of URIs of files to be deleted.
    """
    # Get the set of IDs corresponding to the URIs to be deleted
    ids_to_delete = get_ids_for_uris_to_delete(uri_of_files_to_delete)

    # Delete each document from the index by ID
    for doc_id in ids_to_delete:
        response = doc_deletion_in_index_by_id(document_id=doc_id)
        if response.status_code == 200:
            print(f"Successfully deleted document with ID: {doc_id}")
        else:
            print(f"Failed to delete document with ID: {doc_id}. Response code: {response.status_code}")


# 7. For Files to be Indexed
#enter credentials
account_name = 'cresendrugdata'
account_key = 'Hy4tA8wAO+6pOHFGIPN9YPUyEiBq+6qk7UY0Ew+vebWSCcmKelWkopuQVaonEL4258E5EljnFlvp+AStRNdAoA=='

#create a client to interact with blob storage
connect_str = 'DefaultEndpointsProtocol=https;AccountName=' + account_name + ';AccountKey=' + account_key + ';EndpointSuffix=core.windows.net'

from typing import List
import os
import tempfile
from azure.storage.blob import BlobClient, ContainerClient

# Document class as provided
from langchain.pydantic_v1 import Field
from langchain.load.serializable import Serializable

class Document(Serializable):
    page_content: str
    metadata: dict = Field(default_factory=dict)

    @property
    def lc_serializable(self) -> bool:
        return True

# AzureBlobStorageFileLoader class as provided
import os
import tempfile
from typing import List
#from document import Document  # Assuming document.py is in the same directory or in the Python path
from langchain.document_loaders.base import BaseLoader
from langchain.document_loaders.unstructured import UnstructuredFileLoader

class AzureBlobStorageFileLoader(BaseLoader):
    def __init__(self, conn_str: str, container: str, blob_name: str):
        self.conn_str = conn_str
        self.container = container
        self.blob = blob_name

    def load(self) -> List[Document]:
        from azure.storage.blob import BlobClient
        client = BlobClient.from_connection_string(
            conn_str=self.conn_str, container_name=self.container, blob_name=self.blob
        )
        blob_url = client.url

        with tempfile.TemporaryDirectory() as temp_dir:
            # Sanitize self.blob by replacing or removing invalid characters
            safe_blob_name = self.blob.replace('/', '_')  # Replace slashes with underscores
            file_path = os.path.join(temp_dir, safe_blob_name)

            with open(file_path, "wb") as file:
                blob_data = client.download_blob()
                blob_data.readinto(file)

            loader = UnstructuredFileLoader(file_path)
            documents = loader.load()

            for doc in documents:
                doc.metadata['url'] = blob_url

            return documents


# AzureBlobStorageContainerLoader class as provided
from typing import List
#from document import Document  # Adjust this import to your project structure
from langchain.document_loaders.base import BaseLoader
#from azure_blob_storage_file_loader import AzureBlobStorageFileLoader  # Adjust this import to your project structure

class AzureBlobStorageContainerLoader(BaseLoader):
    def __init__(self, conn_str: str, container: str, prefix: str = ""):
        self.conn_str = conn_str
        self.container = container
        self.prefix = prefix

    def load(self) -> List[Document]:
        from azure.storage.blob import ContainerClient
        container = ContainerClient.from_connection_string(
            conn_str=self.conn_str, container_name=self.container
        )
        docs = []
        blob_list = container.list_blobs(name_starts_with=self.prefix) if self.prefix else container.list_blobs()

        # Filter blob_list based on uri_of_files_to_index
        filtered_blob_list = [blob for blob in blob_list if f"https://{account_name}.blob.core.windows.net/{self.container}/{blob.name}" in uri_of_files_to_index]

        for blob in filtered_blob_list:
            loader = AzureBlobStorageFileLoader(self.conn_str, self.container, blob.name)
            docs.extend(loader.load())
        return docs

def blob_to_text_processing_with_uri_filter(connect_str: str, container_name: str, prefix: str = ""):
    """
    Function to process text from blobs in Azure Blob Storage.

    Parameters:
    connect_str (str): The connection string.
    container_name (str): The name of the blob container.
    prefix (str): Optional prefix to filter blobs.
    """
    # Initialize AzureBlobStorageContainerLoader with connection string, container name, and optional prefix
    loader = AzureBlobStorageContainerLoader(connect_str, container_name, prefix)

    # Load documents
    documents = loader.load()

    # Calculate the total number of characters
    total_characters = sum([len(doc.page_content) for doc in documents])

    print(f'You have {len(documents)} file(s) in your data')
    print(f'There are a total of {total_characters} characters in your documents')

    # Calculate total number of tokens
    total_tokens = total_characters / 4

    # Calculate total price for embedding
    total_price = (total_tokens * 0.0004) / 1000
    print(f'The total price for embedding is $ {total_price}')
    return documents

from langchain.text_splitter import TokenTextSplitter
# from langchain.text_splitter import CharacterTextSplitter

def splitting_function(documents, chunk_size=1000, chunk_overlap=5, mode='token'):
    """
    Function to split documents using either TokenTextSplitter or CharacterTextSplitter.

    Parameters:
    documents (list): List of documents to be split.
    chunk_size (int): Size of each chunk after splitting. Default is 1000.
    chunk_overlap (int): Overlap size between chunks. Default is 5.
    mode (str): Splitting mode, either 'token' or 'character'. Default is 'token'.

    Returns:
    list: List of split documents.
    """
    if mode == 'token':
        text_splitter = TokenTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    else:
        # Uncomment the import for CharacterTextSplitter at the top if you choose to use this option
        # text_splitter = CharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
        raise ValueError("Unsupported mode. Use 'token' or implement 'character' mode with appropriate import and instantiation.")

    split_documents = text_splitter.split_documents(documents)
    print(f'After splitting, there are {len(split_documents)} documents')
    return split_documents

#split_docs = splitting_function(documents)

import os
import time
import openai
from dotenv import load_dotenv
from langchain.chat_models import AzureChatOpenAI
from langchain.embeddings import OpenAIEmbeddings
from azure.search.documents.indexes.models import (
    SemanticSettings,
    SemanticConfiguration,
    PrioritizedFields,
    SemanticField
)
from langchain.vectorstores.azuresearch import AzureSearch

def docs_to_index_ingestion(documents):
    """
    Ingests documents into the index.

    Parameters:
    documents (list): List of documents to be indexed.
    """
    # Load environment variables
    load_dotenv()

    # Set OpenAI configuration
    openai.api_type = "azure"
    openai.api_version = "2023-03-15-preview"
    openai.api_base =  os.getenv('OPENAI_API_BASE')
    openai.api_key = os.getenv("AZ_OPENAI_API_KEY")
    # Initialize AzureChatOpenAI and OpenAIEmbeddings
    llm = AzureChatOpenAI(deployment_name="gpt-35-turbo-16k", openai_api_version="2023-03-15-preview")
    embeddings = OpenAIEmbeddings(model = 'text-embedding-ada-002', deployment='text-embedding-ada-002',chunk_size=1)

    # More OpenAI configuration
    openai.api_key = os.getenv("AZURE_OPENAI_API_KEY")
    openai.api_base = os.getenv("AZURE_OPENAI_ENDPOINT")
    openai.api_version = os.getenv("AZURE_OPENAI_API_VERSION")
    model = "text-embedding-ada-002"

    # Set Vector Store Settings
    vector_store_address = os.getenv("AZURE_SEARCH_SERVICE_ENDPOINT")
    vector_store_password = os.getenv("AZURE_SEARCH_ADMIN_KEY")
    index_name = "cresendev-dummy"

    # Initialize OpenAIEmbeddings and AzureSearch
    embeddings = OpenAIEmbeddings(deployment=model, model=model, chunk_size=1, openai_api_base=os.getenv("AZURE_OPENAI_ENDPOINT"), openai_api_type="azure")
    vector_store = AzureSearch(
        azure_search_endpoint=vector_store_address,
        azure_search_key=vector_store_password,
        index_name=index_name,
        embedding_function=embeddings.embed_query,
        semantic_configuration_name='config',
        semantic_settings=SemanticSettings(
            default_configuration='config',
            configurations=[
                SemanticConfiguration(
                    name='config',
                    prioritized_fields=PrioritizedFields(
                        title_field=SemanticField(field_name='content'),
                        prioritized_content_fields=[SemanticField(field_name='content')],
                        prioritized_keywords_fields=[SemanticField(field_name='metadata')]
                    ))
            ])
    )

    # Start the timer
    start_time = time.time()

    # Add documents to vector store
    vector_store.add_documents(documents=documents)

    # Calculate and print the elapsed time
    elapsed_time = time.time() - start_time
    hours = int(elapsed_time // 3600)
    minutes = int((elapsed_time % 3600) // 60)
    seconds = int(elapsed_time % 60)
    print(f"Time taken for Embedding: {hours} hours, {minutes} minutes, {seconds} seconds")
    print("Successfully processed and added into index")

# Usage
#docs_to_index_ingestion(split_docs)
```

With deletion


Home > cresendrugdata | Containers >

cresendev-dummy ...
Container

Authentication method: Access key (Switch to Azure AD User Account)
Location: cresendev-dummy

Name   Modified

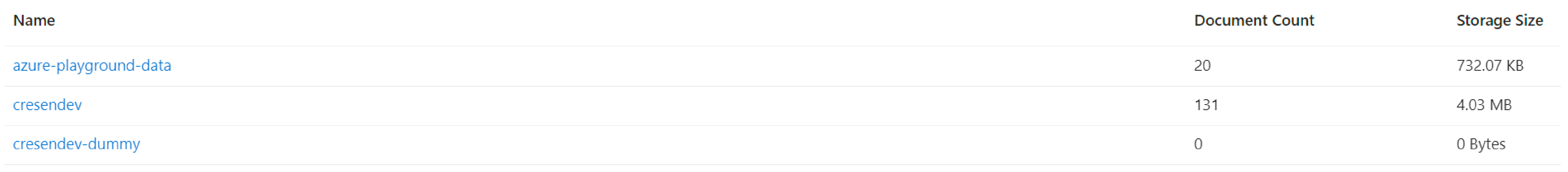
No blobs found.

```
In [2]: # 1. files_in_blob_list_creation
files_in_blob_list_creation('cresendev-dummy', 'jazz', account_name, connect_str)
# 2. files_indexed_creation
files_indexed_creation("cresendev-dummy")
# 3. load_csv_into_set (for files_in_blob.csv)
blob_files_set = load_csv_into_set("files_in_blob.csv", "Blob URI")
# 4. load_csv_into_set (for files_indexed.csv)
indexed_files_set = load_csv_into_set("files_indexed.csv", "Blob URI")
# 5. Calculate Set Differences
uri_of_files_to_index = blob_files_set - indexed_files_set # Finds new files in Blob.
uri_of_files_to_delete = indexed_files_set - blob_files_set # Identifies files that have been removed from Blob.
# 6. For Files in Index to be Deleted
if uri_of_files_to_delete:  # checks if the set is not empty
    uri_set_to_delete = uri_of_files_to_delete
    delete_docs_in_index_by_uri(uri_of_files_to_delete=uri_set_to_delete)
else:
    print("No URIs to delete.")  # message when there's nothing to delete
# 7. For Files to be Indexed
if uri_of_files_to_index: # checks if the set is not empty
    documents = blob_to_text_processing_with_uri_filter(connect_str, 'cresendev-dummy',prefix=None)
    split_docs = splitting_function(documents)
    docs_to_index_ingestion(split_docs)
```

```
Successfully deleted document with ID: ZDZjNDJlZTAtNDU3MC00YWRjLTkxZTUtOGZkZmM3NDIzMTY5
Successfully deleted document with ID: ZTk1ODZkYTAtMDQ0Ni00MzgwLWI4MGEtMDkwZjlhZTExY2Zh
Successfully deleted document with ID: OTlmYzAxOGUtZjgxOC00MWU0LWIxZWUtNzQ1NmExODQ4YTcy
Successfully deleted document with ID: ZjNiZTc2NDMtMzI3Zi00MTA5LTljMWItMzY4YjU2NDg4MmNl
Successfully deleted document with ID: NWE0NTIwNmQtNThhMy00ZTkzLTk1Y2QtMzQ2NDk4MGI1NTYw
Successfully deleted document with ID: YTBmZjRlMGQtNTJlMy00YjlhLTlmOWYtN2UxZTJjZjQyMTEz
Successfully deleted document with ID: Yzc2MGU3MmMtM2E2Ny00MGFmLTgzNzEtZjM3OGI5NTVmMmYw
Successfully deleted document with ID: NzYxZWJiZWUtOGYzMC00ZTlkLWFmN2UtNzVjOWUxNjNkOTIx
Successfully deleted document with ID: MTFlYjgyODAtOTBkZS00YzAzLTg0OTItZWIzOThkYTliMTRh
Successfully deleted document with ID: MjA5Y2QyZWYtZjcxMy00Y2U3LTk5ZGYtYzRiNGMzOGYzMWZh
Successfully deleted document with ID: MThhZTZiYTktNmQ4MS00NzA5LTliNGQtZjZlMDljMDhjMTc2
Successfully deleted document with ID: OWMzYTczMjYtZDQ4YS00OWUxLWFjN2MtODg4NTUzNmFmNzk0
Successfully deleted document with ID: OGY2MDE3NzgtYzIxYy00ZjI5LWFhZTgtNTE4OTZlYWZkYjZh
Successfully deleted document with ID: Y2FjMTEyMzAtMDJhYS00OGI1LWE2ODAtYjQzZTRmNWVmYmE4
Successfully deleted document with ID: MzIxNDNjZjEtN2JmOC00YzQ3LWFiMjgtMDI0N2MyMDZiNzNl
Successfully deleted document with ID: MWE1Y2FlOTktOWE4OS00ZTU2LTk2MTUtOGI0OWVjODNmZGU3
Successfully deleted document with ID: MzE5Yjc3OGEtZmRiYi00N2QwLTgyZDQtZDg3ZjQ1NzBhYTg1
Successfully deleted document with ID: MzlmNjhhOGEtMGQ1OC00M2RjLTg0YzUtYWM3NzBjMGEzOTlh
Successfully deleted document with ID: ZWRjZTg0M2MtNTdlOC00ODRhLTg5ZTItNTI0MDlhMTAyYmE1
Successfully deleted document with ID: YzY2ZjM5ZjYtODZlOS00MTg5LThkYjktNWQ1NmM4OGQwZWRk
Successfully deleted document with ID: NzE5NDZjYmMtYWFiYy00NjBkLThkNWMtY2JlMTFmNTJmNjk0
Successfully deleted document with ID: ZDgyNGY4Y2UtMzc2YS00NjBlLWI3NjItM2NkYzEyZDgwNGJm
Successfully deleted document with ID: ZjU4MTg4YTQtMGQ1Ny00ZmY5LWFlY2UtMmEyNGQxNjE5ODAy
Successfully deleted document with ID: MDZiZDdiNjMtNmZjYy00NWNmLTkxNTItY2JkZjU1OTRlN2Vk
Successfully deleted document with ID: MDc3NjdiNjMtNWQxZi00NDcyLTk0NDMtZWJiOWI2ODlhZmEw
Successfully deleted document with ID: MGFhYTMyZWQtMDYxYS00YmFiLWE0YTEtMGU2YWI3ODU3ODFh
Successfully deleted document with ID: MjM5Njc0MGQtNzIzNS00MDZlLWFkNzAtYmJmYmI3NmYxZDgz
Successfully deleted document with ID: Y2FhZTMxNTMtMGQzYi00OTMxLTllYWUtMDZkYjBiOTZlZjRj
Successfully deleted document with ID: YTc1ZDAxYmItNDJlMi00MWNiLWJjYmUtMTIzMWQ0MDMwNWU2
Successfully deleted document with ID: MmJhMGYzMWYtNmYyOC00ZmQ4LTk0MDgtYmQyZjY1MWJlOTRk
Successfully deleted document with ID: OTQ4MzQxM2ItYmE4OC00ZTA1LWJkNTAtMGQwYTc0ZjdmZTVl
Successfully deleted document with ID: ZDAyYTFhNjctY2UyMi00NzNjLTg4NzItMmFmYWI1NzBkOTkw
Successfully deleted document with ID: MTYyNGZmMzUtOWI3OC00NTYzLWFhMWMtZDBmMDkyOTU0NTBm
Successfully deleted document with ID: ODE1YzA0OWEtNDJjMy00YjFiLWJkZjItYjI4MjUwYTIwNzQz
Successfully deleted document with ID: OWU5OTE3ZWUtMTZhMS00MWYyLTg3YjYtZDMxZjM2MTFiM2E0
Successfully deleted document with ID: Yjk1MzA3OTUtYzIyYS00YTRlLTg1ZTItN2I4ZTc0OWE5MzY0
Successfully deleted document with ID: ZmU5MGRmMTMtYWM0OS00MGU2LTk4MzEtMTkwZDE4NzRmYjI1
You have 0 file(s) in your data
There are a total of 0 characters in your documents
The total price for embedding is $ 0.0
After splitting, there are 0 documents
Time taken for Embedding: 0 hours, 0 minutes, 0 seconds
```

Updation

Before:

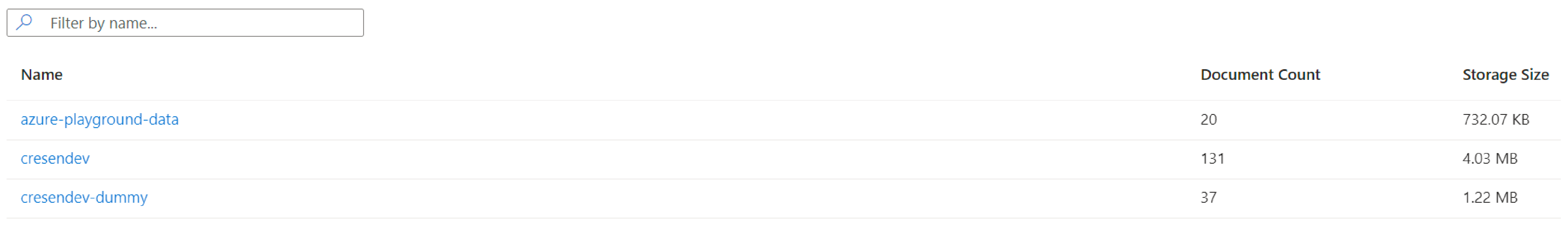


Name	Document Count	Storage Size
azure-playground-data	20	732.07 KB
cresendev	131	4.03 MB
cresendev-dummy	0	0 Bytes

In [9]:
```python
# 1. files_in_blob_list_creation
files_in_blob_list_creation('cresendev-dummy', 'jazz', account_name, connect_str)
# 2. files_indexed_creation
files_indexed_creation("cresendev-dummy")
# 3. load_csv_into_set (for files_in_blob.csv)
blob_files_set = load_csv_into_set("files_in_blob.csv", "Blob URI")
# 4. load_csv_into_set (for files_indexed.csv)
indexed_files_set = load_csv_into_set("files_indexed.csv", "Blob URI")
# 5. Calculate Set Differences
uri_of_files_to_index = blob_files_set - indexed_files_set # Finds new files in Blob.
uri_of_files_to_delete = indexed_files_set - blob_files_set # Identifies files that have been removed from Blob.
# 6. For Files in Index to be Deleted
if uri_of_files_to_delete:  # checks if the set is not empty
    uri_set_to_delete = uri_of_files_to_delete
    delete_docs_in_index_by_uri(uri_of_files_to_delete=uri_set_to_delete)
else:
    print("No URIs to delete.")  # message when there's nothing to delete
# 7. For Files to be Indexed
if uri_of_files_to_index: # checks if the set is not empty
    documents = blob_to_text_processing_with_uri_filter(connect_str, 'cresendev-dummy',prefix=None)
    split_docs = splitting_function(documents)
    docs_to_index_ingestion(split_docs)
```

```
No URIs to delete.
You have 1 file(s) in your data
There are a total of 170833 characters in your documents
The total price for embedding is $ 0.017083300000000003
After splitting, there are 37 documents
Time taken for Embedding: 0 hours, 0 minutes, 18 seconds
Successfully processed and added into index
```

After:



Filter by name...

Name	Document Count	Storage Size
azure-playground-data	20	732.07 KB
cresendev	131	4.03 MB
cresendev-dummy	37	1.22 MB

In [ ]: