Implement Retrieval Augmented Generation (RAG) with Azure OpenAI Service

The Azure OpenAI Service enables you to use your own data with the intelligence of the underlying LLM. You can limit the model to only use your data for pertinent topics, or blend it with results from the pre-trained model.

In the scenario for this exercise, you will perform the role of a software developer working for Margie's Travel Agency. You will explore how use Azure AI Search to index your own data and use it with Azure OpenAI to augment prompts.

This exercise will take approximately **30** minutes.

Provision Azure resources

To complete this exercise, you'll need:

- An Azure OpenAI resource.
- An Azure AI Search resource.
- An Azure Storage Account resource.

1. Sign into the **Azure portal** at `https://portal.azure.com`.
2. Create an **Azure OpenAI** resource with the following settings:
    - **Subscription**: *Select an Azure subscription that has been approved for access to the Azure OpenAI service*
    - **Resource group**: *Choose or create a resource group*
    - **Region**: *Make a **random** choice from any of the following regions**
        - East US
        - East US 2
        - North Central US
        - South Central US
        - Sweden Central
        - West US
        - West US 3
    - **Name**: *A unique name of your choice*
    - **Pricing tier**: Standard S0

    > ! * Azure OpenAI resources are constrained by regional quotas. The listed regions include default quota for the model type(s) used in this exercise. Randomly choosing a region reduces the risk of a single region reaching its quota limit in scenarios where you are sharing a subscription with other users. In the event of a quota limit being reached later in the exercise, there's a possibility you may need to create another resource in a different region.

3. While the Azure OpenAI resource is being provisioned, create an **Azure AI Search** resource with the following settings:
    - **Subscription**: *The subscription in which you provisioned your Azure OpenAI resource*
    - **Resource group**: *The resource group in which you provisioned your Azure OpenAI resource*
    - **Service name**: *A unique name of your choice*
    - **Location**: *The region in which you provisioned your Azure OpenAI resource*
    - **Pricing tier**: Basic
4. While the Azure AI Search resource is being provisioned, create a **Storage account** resource with the following settings:
    - **Subscription**: *The subscription in which you provisioned your Azure OpenAI resource*
    - **Resource group**: *The resource group in which you provisioned your Azure OpenAI resource*
    - **Storage account name**: *A unique name of your choice*

   - **Region**: *The region in which you provisioned your Azure OpenAI resource*
   - **Primary service**: Azure Blob Storage or Azure Data Lake Storage Gen 2
   - **Performance**: Standard
   - **Redundancy**: Locally redundant storage (LRS)
5. After all three of the resources have been successfully deployed in your Azure subscription, review them in the Azure portal and gather the following information (which you'll need later in the exercise):

   - The **endpoint** and a **key** from the Azure OpenAI resource you created (available on the **Keys and Endpoint** page for your Azure OpenAI resource in the Azure portal)
   - The endpoint for your Azure AI Search service (the **Url** value on the overview page for your Azure AI Search resource in the Azure portal).
   - A **primary admin key** for your Azure AI Search resource (available in the **Keys** page for your Azure AI Search resource in the Azure portal).

Upload your data

You're going to ground the prompts you use with a generative AI model by using your own data. In this exercise, the data consists of a collection of travel brochures from the fictional *Margies Travel* company.

1. In a new browser tab, download an archive of brochure data from `https://aka.ms/own-data-brochures`. Extract the brochures to a folder on your PC.
2. In the Azure portal, navigate to your storage account and view the **Storage browser** page.
3. Select **Blob containers** and then add a new container named `margies-travel`.
4. Select the **margies-travel** container, and then upload the .pdf brochures you extracted previously to the root folder of the blob container.

Deploy AI models

You're going to use two AI models in this exercise:

- A text embedding model to *vectorize* the text in the brochures so it can be indexed efficiently for use in grounding prompts.
- A GPT model that you application can use to generate responses to prompts that are grounded in your data.

Deploy a model

Next, you will deploy Azure OpenAI models from Cloud Shell.

1. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***Bash*** environment. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

> **Note**: If you have previously created a cloud shell that uses a *PowerShell* environment, switch it to ***Bash***.

Code Copy

```
az cognitiveservices account deployment create \
   -g <your_resource_group> \
   -n <your_OpenAI_resource> \
   --deployment-name text-embedding-ada-002 \
   --model-name text-embedding-ada-002 \
   --model-version "2"  \
   --model-format OpenAI \
   --sku-name "Standard" \
   --sku-capacity 5
```

> **Note**: Sku-capacity is measured in thousands of tokens per minute. A rate limit of 5,000 tokens per minute is more than adequate to complete this exercise while leaving capacity for other people using the same subscription.

After the text embedding model has been deployed, create a new deployment of the **gpt-4o** model with the following settings:

Code Copy

```
az cognitiveservices account deployment create \
   -g <your_resource_group> \
   -n <your_OpenAI_resource> \
   --deployment-name gpt-4o \
   --model-name gpt-4o \
   --model-version "2024-05-13" \
   --model-format OpenAI \
   --sku-name "Standard" \
   --sku-capacity 5
```

Create an index

To make it easy to use your own data in a prompt, you'll index it using Azure AI Search. You'll use the text embedding model to *vectorize* the text data (which results in each text token in the index being represented by numeric vectors - making it compatible with the way a generative AI model represents text)

1. In the Azure portal, navigate to your Azure AI Search resource.
2. On the **Overview** page, select **Import and vectorize data**.
3. In the **Setup your data connection** page, select **Azure Blob Storage** and configure the data source with the following settings:
    - **Subscription**: The Azure subscription in which you provisioned your storage account.
    - **Blob storage account**: The storage account you created previously.
    - **Blob container**: margies-travel
    - **Blob folder**: *Leave blank*
    - **Enable deletion tracking**: Unselected
    - **Authenticate using managed identity**: Unselected
4. On the **Vectorize your text** page, select the following settings:
    - **Kind**: Azure OpenAI
    - **Subscription**: The Azure subscription in which you provisioned your Azure OpenAI service.
    - **Azure OpenAI Service**: Your Azure OpenAI Service resource
    - **Model deployment**: text-embedding-ada-002
    - **Authentication type**: API key
    - **I acknowledge that connecting to an Azure OpenAI service will incur additional costs to my account**: Selected
5. On the next page, do **not** select the option to vectorize images or extract data with AI skills.
6. On the next page, enable semantic ranking and schedule the indexer to run once.
7. On the final page, set the **Objects name prefix** to `margies-index` and then create the index.

Prepare to develop an app in Visual Studio Code

Now let's explore the use of your own data in an app that uses the Azure OpenAI service SDK. You'll develop your app using Visual Studio Code. The code files for your app have been provided in a GitHub repo.

> **Tip**: If you have already cloned the **mslearn-openai** repo, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.

2. Open the palette (SHIFT+CTRL+P or **View** > **Command Palette...**) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/mslearn-openai` repository to a local folder (it doesn't matter which folder).

3. When the repository has been cloned, open the folder in Visual Studio Code.

> **Note**: If Visual Studio Code shows you a pop-up message to prompt you to trust the code you are opening, click on **Yes, I trust the authors** option in the pop-up.

4. Wait while additional files are installed to support the C# code projects in the repo.

> **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

Configure your application

Applications for both C# and Python have been provided, and both apps feature the same functionality. First, you'll complete some key parts of the application to enable using your Azure OpenAI resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/02-use-own-data** folder and expand the **CSharp** or **Python** folder depending on your language preference. Each folder contains the language-specific files for an app into which you're going to integrate Azure OpenAI functionality.

2. Right-click the **CSharp** or **Python** folder containing your code files and open an integrated terminal. Then install the Azure OpenAI SDK package by running the appropriate command for your language preference:

   C#:

   Code Copy

   ```
   dotnet add package Azure.AI.OpenAI --version 2.1.0
   dotnet add package Azure.Search.Documents --version 11.6.0
   ```

   Python:

   Code Copy

   ```
   pip install openai==1.65.2
   ```

3. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the configuration file for your preferred language

   - **C#**: appsettings.json
   - **Python**: .env

4. Update the configuration values to include:

   - The **endpoint** and a **key** from the Azure OpenAI resource you created (available on the **Keys and Endpoint** page for your Azure OpenAI resource in the Azure portal)
   - The **deployment name** you specified for your gpt-4o model deployment (should be `gpt-4o`).
   - The endpoint for your search service (the **Url** value on the overview page for your search resource in the Azure portal).
   - A **key** for your search resource (available in the **Keys** page for your search resource in the Azure portal - you can use either of the admin keys)
   - The name of the search index (which should be `margies-index`).

5. Save the configuration file.

Add code to use the Azure OpenAI service

Now you're ready to use the Azure OpenAI SDK to consume your deployed model.

1. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the code file for your preferred language, and replace the comment ***Configure your data source*** with code to your index as a data source for chat completion:

C#: ownData.cs

C# Copy

```
// Configure your data source
// Extension methods to use data sources with options are subject to SDK surface changes.
Suppress the warning to acknowledge this and use the subject-to-change AddDataSource method.
#pragma warning disable AOAI001

ChatCompletionOptions chatCompletionsOptions = new ChatCompletionOptions()
{
    MaxOutputTokenCount = 600,
    Temperature = 0.9f,
};

chatCompletionsOptions.AddDataSource(new AzureSearchChatDataSource()
{
    Endpoint = new Uri(azureSearchEndpoint),
    IndexName = azureSearchIndex,
    Authentication = DataSourceAuthentication.FromApiKey(azureSearchKey),
});
```

Python: ownData.py

Code Copy

```
# Configure your data source
text = input('\nEnter a question:\n')

completion = client.chat.completions.create(
    model=deployment,
    messages=[
        {
            "role": "user",
            "content": text,
        },
    ],
    extra_body={
        "data_sources":[
            {
                "type": "azure_search",
                "parameters": {
                    "endpoint": os.environ["AZURE_SEARCH_ENDPOINT"],
                    "index_name": os.environ["AZURE_SEARCH_INDEX"],
                    "authentication": {
                        "type": "api_key",
                        "key": os.environ["AZURE_SEARCH_KEY"],
                    }
                }
            }
        ],
    }
)
```

Clean up

2. Save the changes to the code file.

Run your application

Now that your app has been configured, run it to send your request to your model and observe the response. You'll notice the only difference between the different options is the content of the prompt, all other parameters (such as token count and temperature) remain the same for each request.

1. In the interactive terminal pane, ensure the folder context is the folder for your preferred language. Then enter the following command to run the application.
   - **C#**: `dotnet run`
   - **Python**: `python ownData.py`

   > **Tip**: You can use the **Maximize panel size** (^) icon in the terminal toolbar to see more of the console text.

2. Review the response to the prompt `Tell me about London`, which should include an answer as well as some details of the data used to ground the prompt, which was obtained from your search service.

   > **Tip**: If you want to see the citations from your search index, set the variable ***show citations*** near the top of the code file to **true**.

Clean up

When you're done with your Azure OpenAI resource, remember to delete the resources in the **Azure portal** at `https://portal.azure.com`. Be sure to also include the storage account and search resource, as those can incur a relatively large cost.