

**Lab Manual- Create an Azure AI Search solution**

**Prepared for**:

**Date:** 9th May 2024

**Prepared by:**

Document Name: Lab Manual **Document Number** AZLabn369

**Contributor:**

Contents

[1. Objective 3](#_Toc167717026)

[2. Provision an Azure AI Search Services 3](#_Toc167717027)

[3. Provision an Azure AI Services 8](#_Toc167717028)

[4. Provision an Azure Blob Storage 10](#_Toc167717029)

[5. Clone the repository in Visual Studio Code 14](#_Toc167717030)

[6. Upload the Documents to Blob 15](#_Toc167717031)

[7. Index the document 18](#_Toc167717032)

[8. Search the index 26](#_Toc167717033)

[9. Create a search client application 31](#_Toc167717034)

[10. Get the endpoint and keys for your search resource 32](#_Toc167717035)

[11. Prepare to use the Azure AI Search SDK 32](#_Toc167717036)

[1. Explore code to search an index 32](#_Toc167717037)

[12. Explore code to render search results 33](#_Toc167717038)

[13. Run the web app 33](#_Toc167717039)

# Objective

All organizations rely on information to make decisions, answer questions, and function efficiently. The problem for most organizations is not a lack of information, but the challenge of finding and extracting the information from the massive set of documents, databases, and other sources in which the information is stored.

For example, suppose Margie's Travel is a travel agency that specializes in organizing trips to cities around the world. Over time, the company has amassed a huge amount of information in documents such as brochures, as well as reviews of hotels submitted by customers. This data is a valuable source of insights for travel agents and customers as they plan trips, but the sheer volume of data can make it difficult to find relevant information to answer a specific customer question.

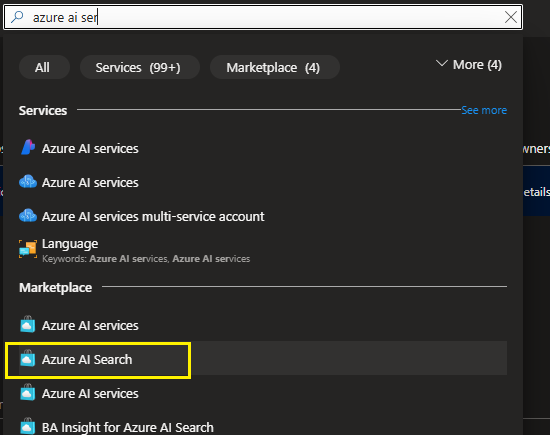
To address this challenge, Margie's Travel can use Azure AI Search to implement a solution in which the documents are indexed and enriched by using AI skills to make them easier to search.

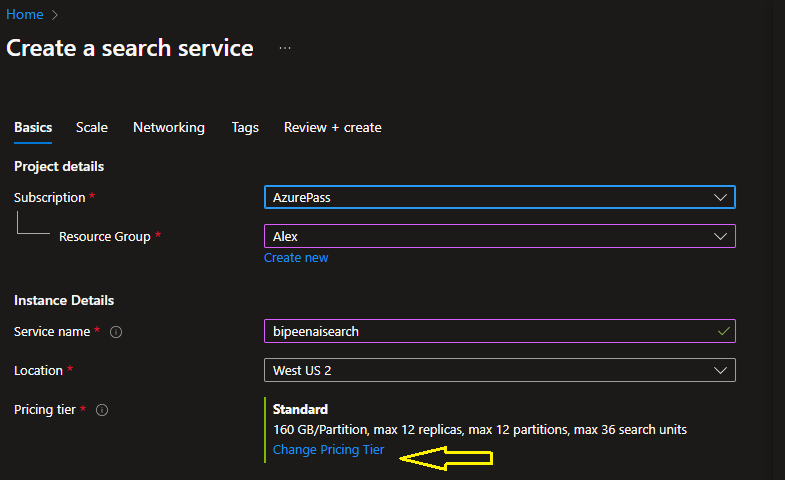
The solution you will create for Margie's Travel requires the following resources in your Azure subscription:

* An **Azure AI Search** resource, which will manage indexing and querying.
* An **Azure AI Services** resource, which provides AI services for skills that your search solution can use to enrich the data in the data source with AI-generated insights.
* A **Storage account** with a blob container in which the documents to be searched are stored

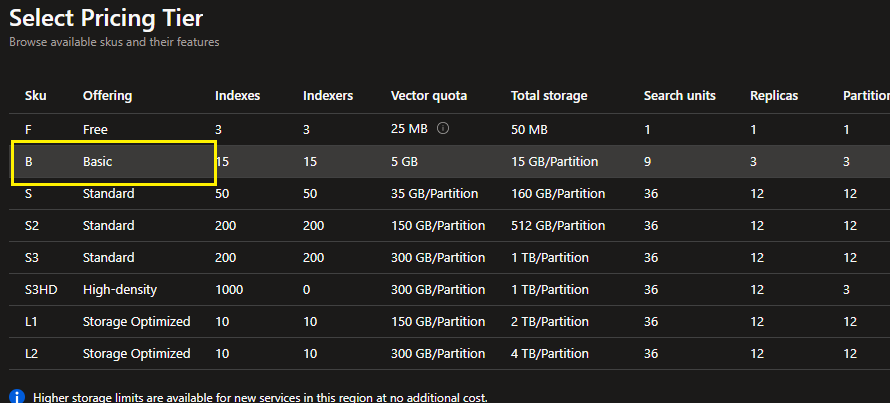
# Provision an Azure AI Search Services

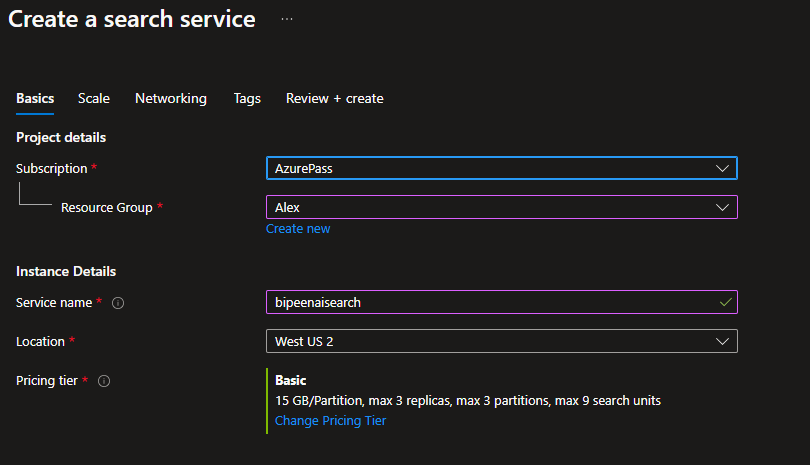
1. In a web browser, open the Azure portal at https://portal.azure.com, and sign in using the Microsoft account associated with your Azure subscription.
2. Select the **＋Create a resource** button, search for *search*, and create an **Azure AI Search** resource with the following settings:
   * **Subscription**: *Your Azure subscription*
   * **Resource group**: *Create a new resource group (if you are using a restricted subscription, you may not have permission to create a new resource group - use the one provided)*
   * **Service name**: *Enter a unique name*
   * **Location**: *Select a location - note that your Azure AI Search and Azure AI Services resources must be in the same location*
   * **Pricing tier**: Basic



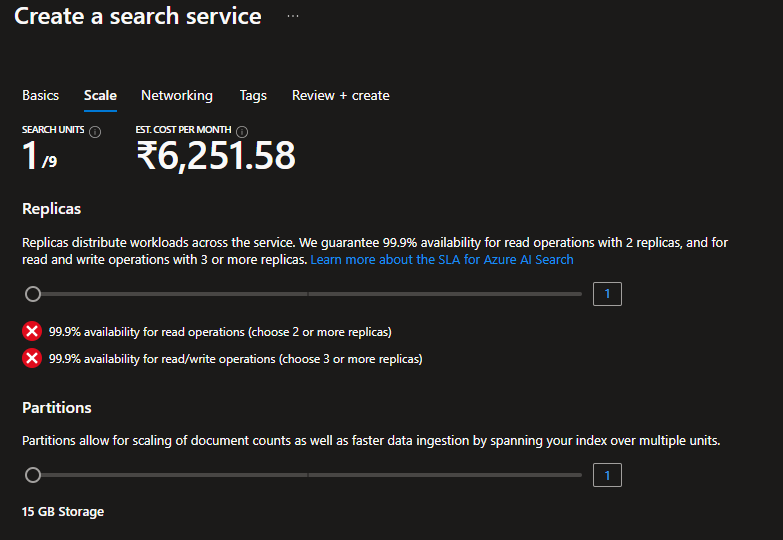


Select Basic Pricing Tier

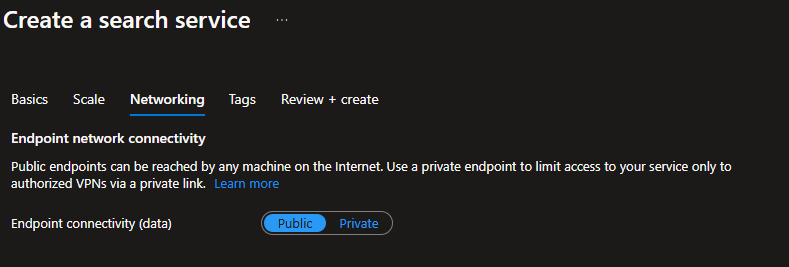




Click Next

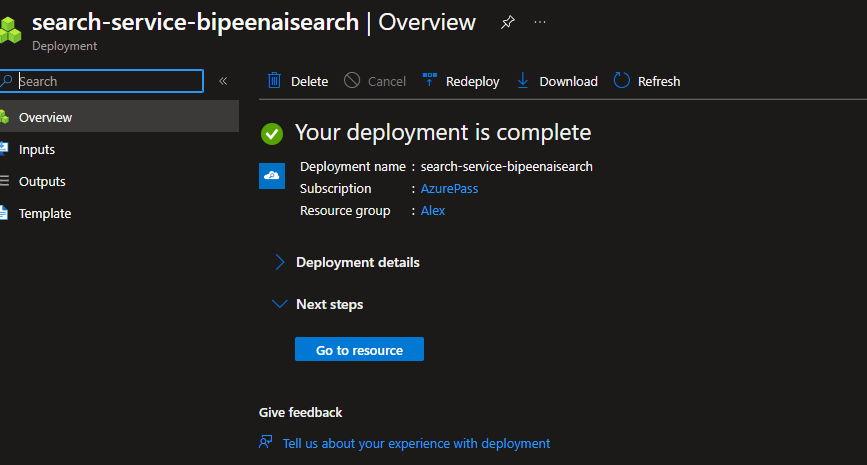


Click Next

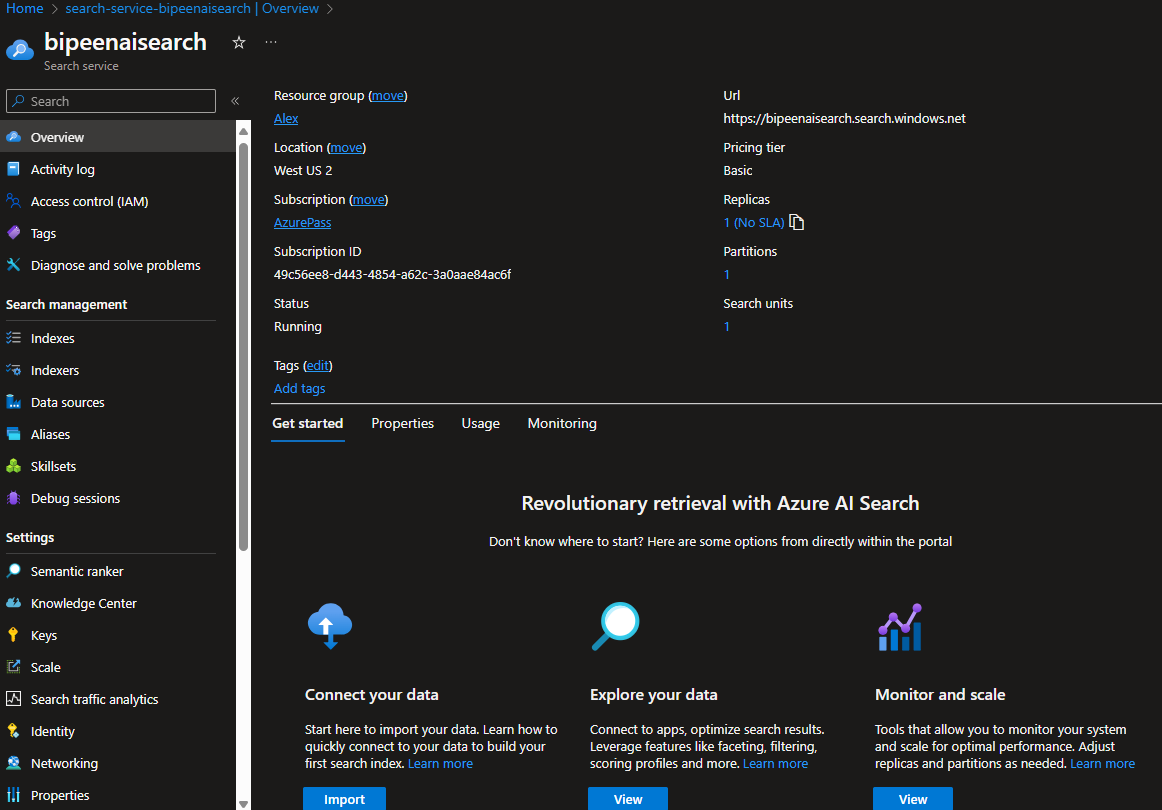


Review and Create

Wait for deployment to complete, and then go to the deployed resource.



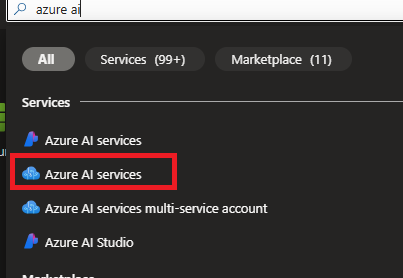
Review the **Overview** page on the blade for your Azure AI Search resource in the Azure portal. Here, you can use a visual interface to create, test, manage, and monitor the various components of a search solution; including data sources, indexes, indexers, and skillsets.



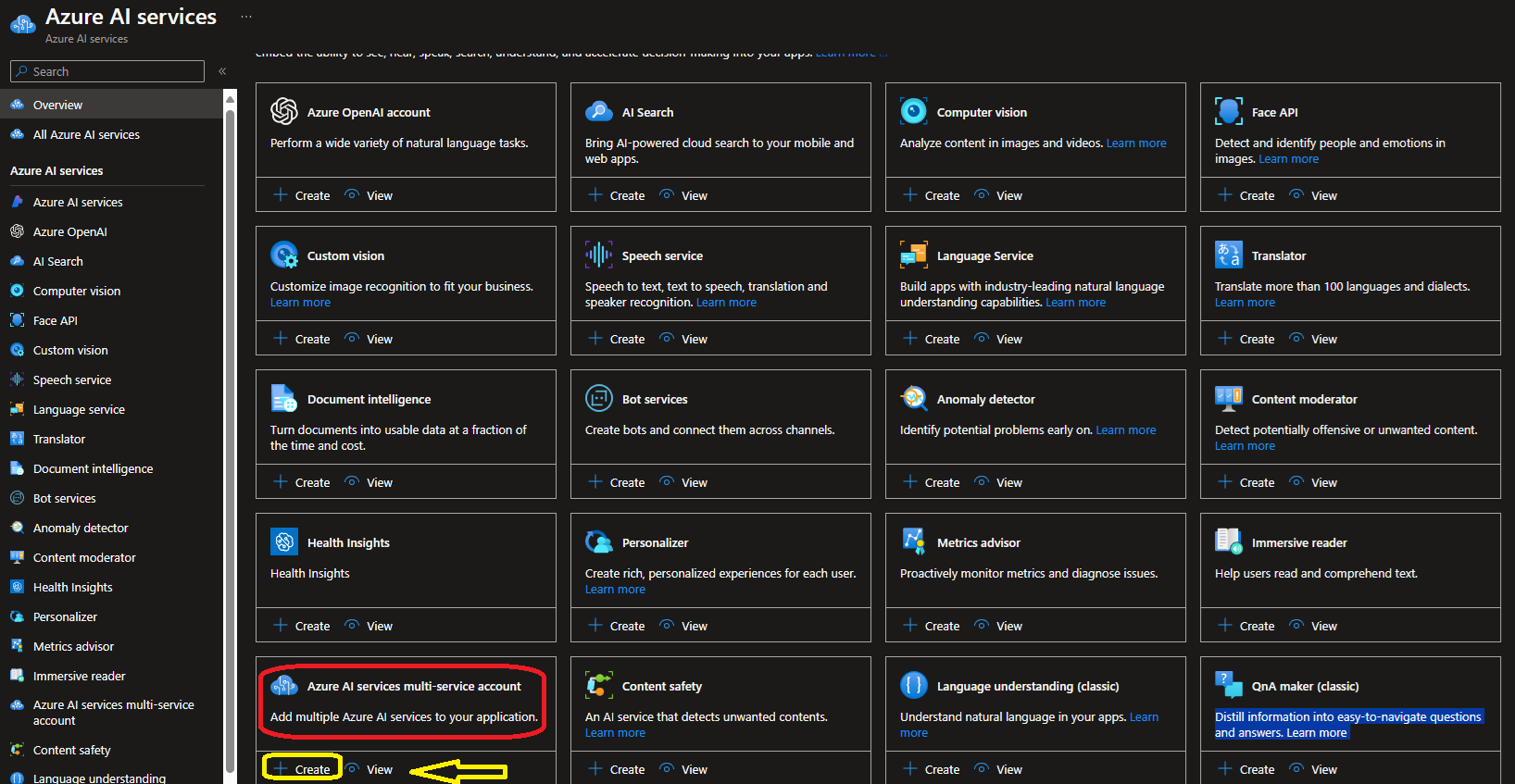
# Provision an Azure AI Services

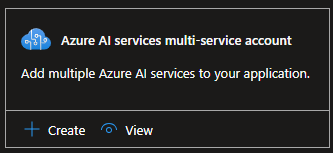
1. Return to the home page of the Azure portal, and then select the **＋Create a resource** button, search for *Azure AI Services*, and create an **Azure AI Services** resource with the following settings:
   * **Subscription**: *Your Azure subscription*
   * **Resource group**: *The same resource group as your Azure AI Search resource*
   * **Region**: *The same location as your Azure AI Search resource*
   * **Name**: *Enter a unique name*
   * **Pricing tier**: Standard S0

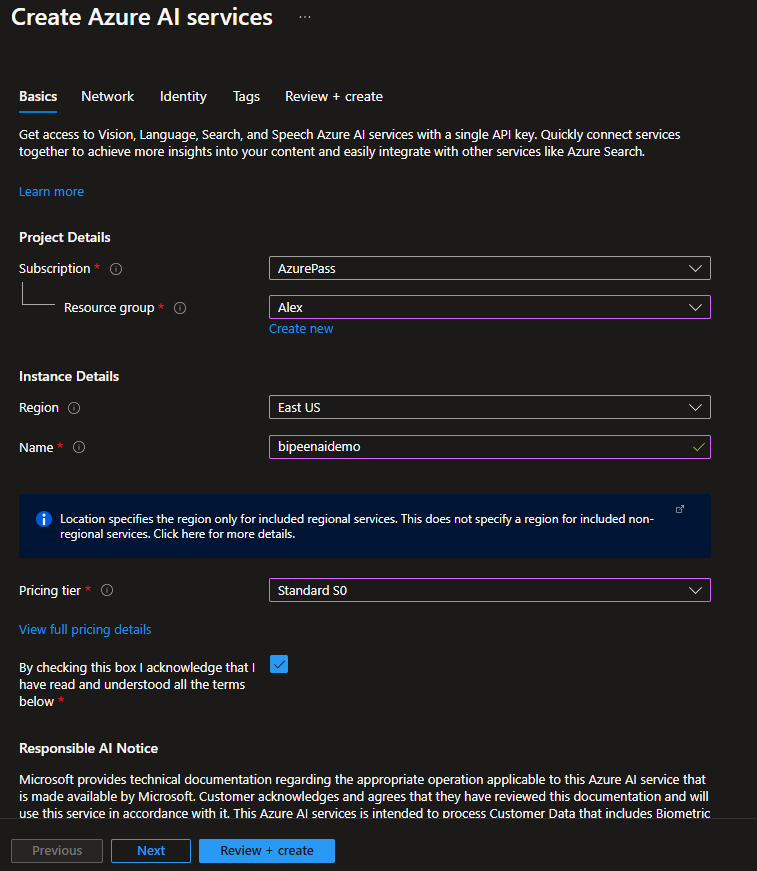
* Serach AI Service and select **Azure AI Services**



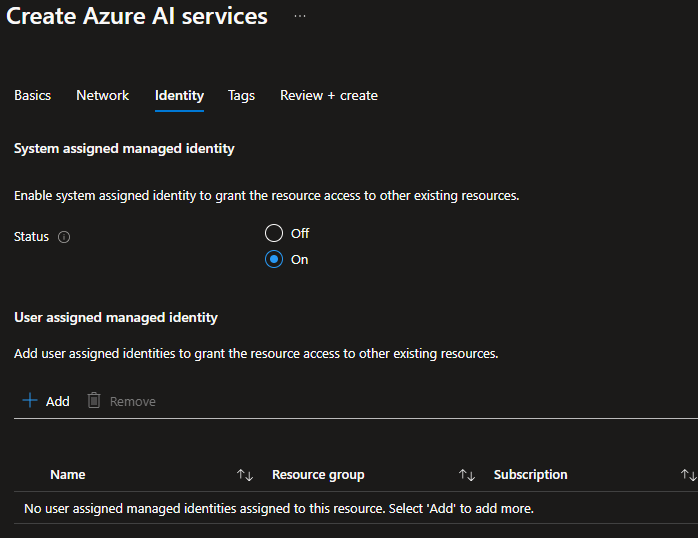
* Select **Azure Ai Service Muti-Service Account** and Click **create**



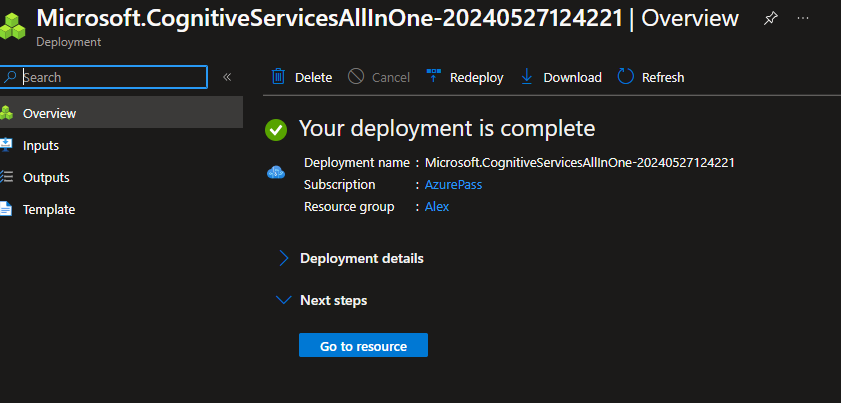




1. Click Next Select the required checkboxes and create the resource.



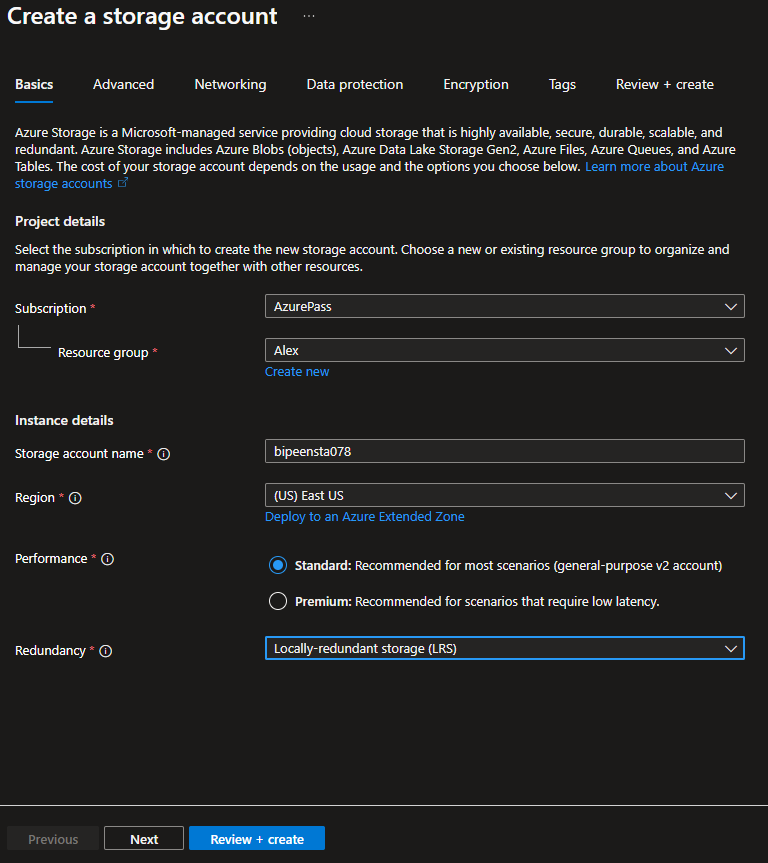
Click Review and Create



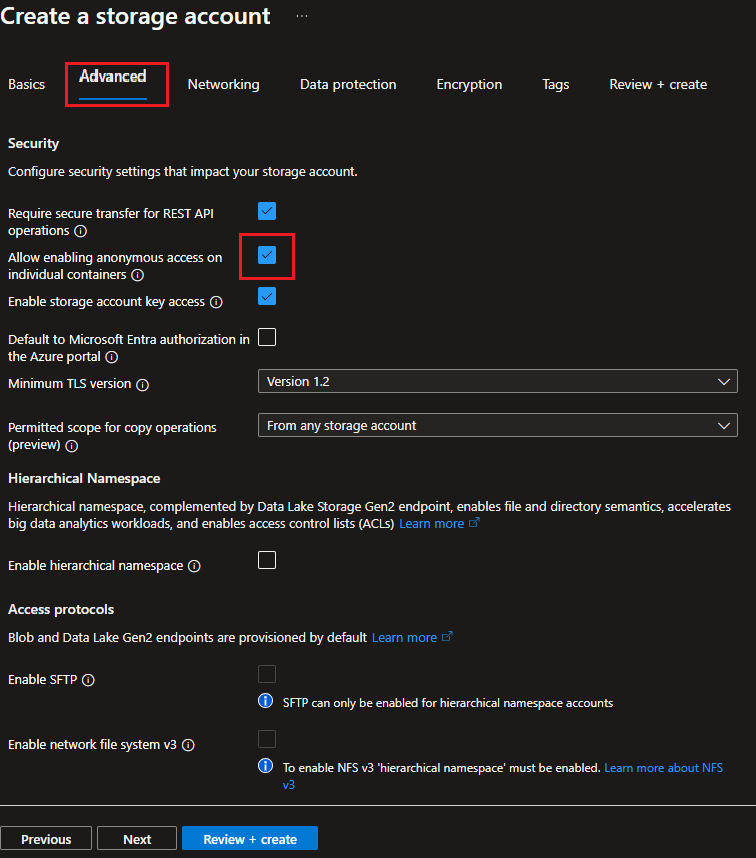
# Provision an Azure Blob Storage

Return to the home page of the Azure portal, and then select the **＋Create a resource** button, search for *storage account*, and create a **Storage account** resource with the following settings:

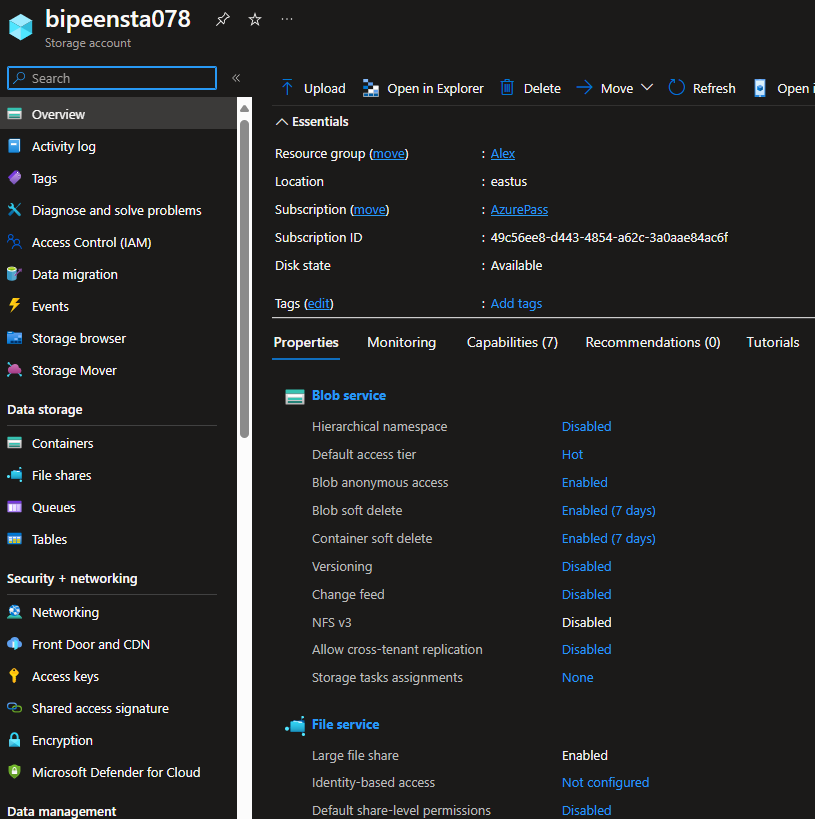
* **Subscription**: *Your Azure subscription*
* **Resource group**: *\*The same resource group as your Azure AI Search and Azure AI Services resources*
* **Storage account name**: *Enter a unique name*
* **Region**: *Choose any available region*
* **Performance**: Standard
* **Replication**: Locally-redundant storage (LRS)



* Click Next and On the **Advanced** tab, check the box next to *Allow enabling anonymous access on individual containers*

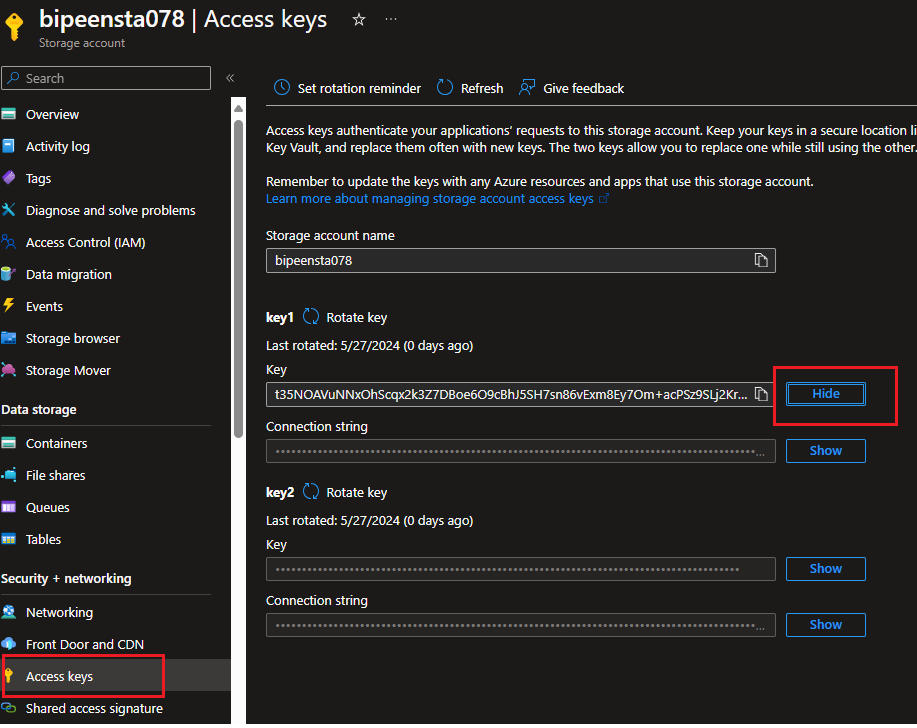


1. Click **Review + Create** and Wait for deployment to complete, and then go to the deployed resource.



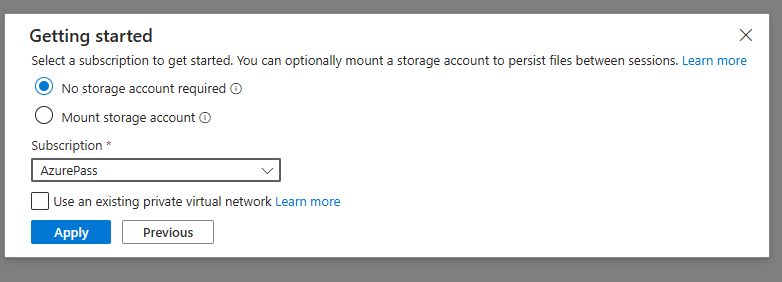
1. On the **Overview** page, note the **Subscription ID** -this identifies the subscription in which the storage account is provisioned.
2. On the **Access keys** page, note that two keys have been generated for your storage account. Then select **Show keys** to view the keys.

**Tip**: Keep the **Storage Account** blade open - you will need the subscription ID and one of the keys in the next procedure.

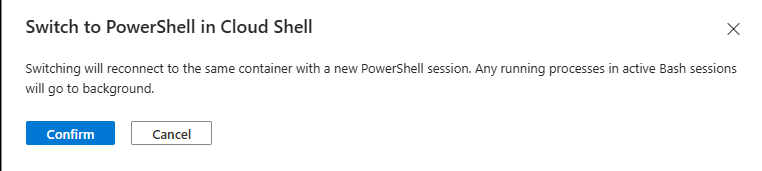


# Clone the repository in Visual Studio Code

1. Open Cloud shell and select your subscription and click apply

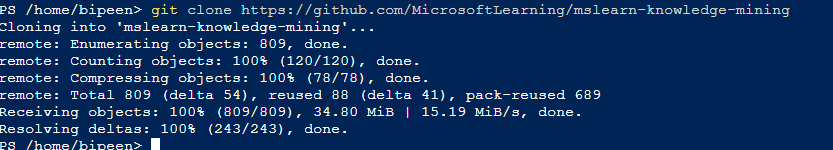


1. Once you login switch to PowerShell



1. Run a **Git: Clone** command to clone the https://github.com/MicrosoftLearning/mslearn-knowledge-mining repository to a local folder (it doesn't matter which folder).

**git clone https://github.com/MicrosoftLearning/mslearn-knowledge-mining**



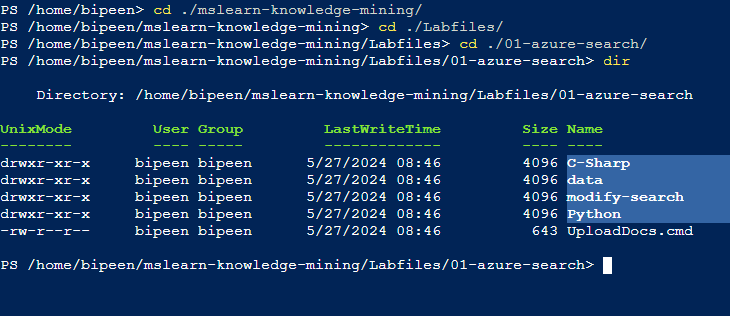
# Upload the Documents to Blob

1. Go Inside **mslearn-ai-services/Labfiles** directory

**cd mslearn-knowledge-mining**

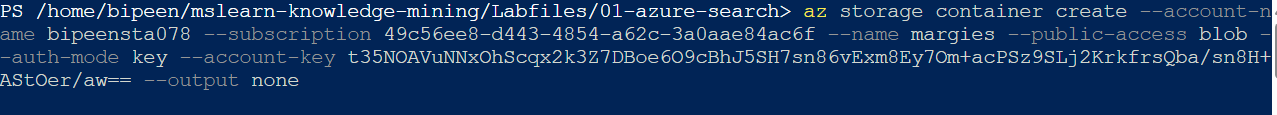
**Cd Labfiles**

**Dir**

****

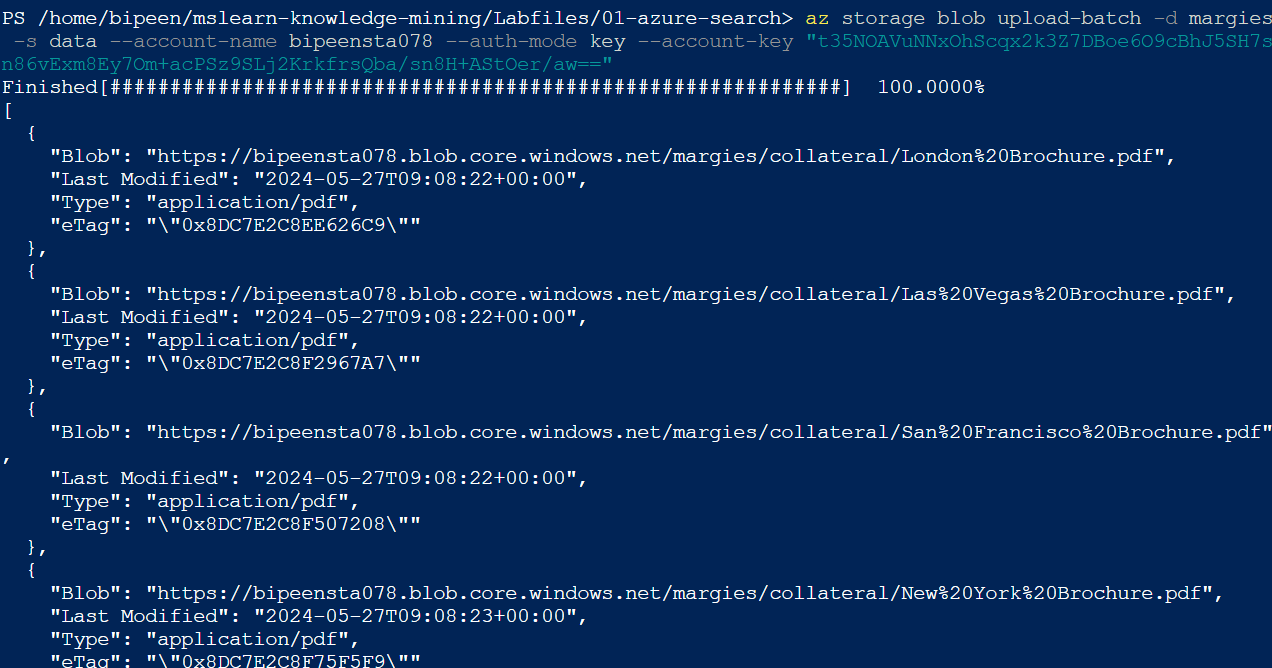
1. Create Container with CLI

**az storage container create --account-name bipeensta078 --subscription 49c56ee8-d443-4854-a62c-3a0aae84ac6f --name margies --public-access blob --auth-mode key --account-key "t35NOAVuNNxOhScqx2k3Z7DBoe6O9cBhJ5SH7sn86vExm8Ey7Om+acPSz9SLj2KrkfrsQba/sn8H+AStOer/aw=="**

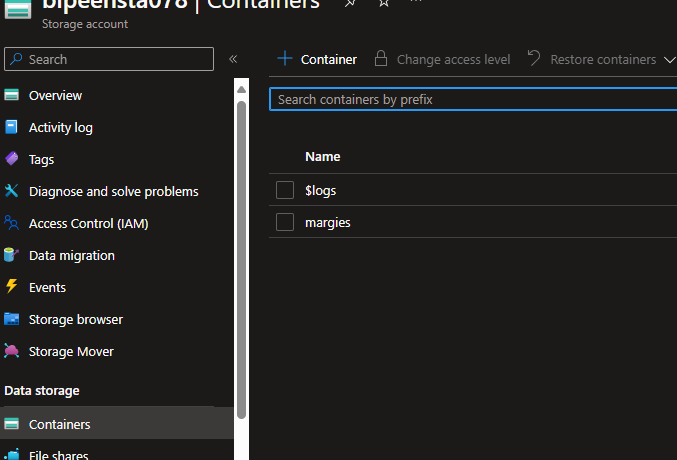
****

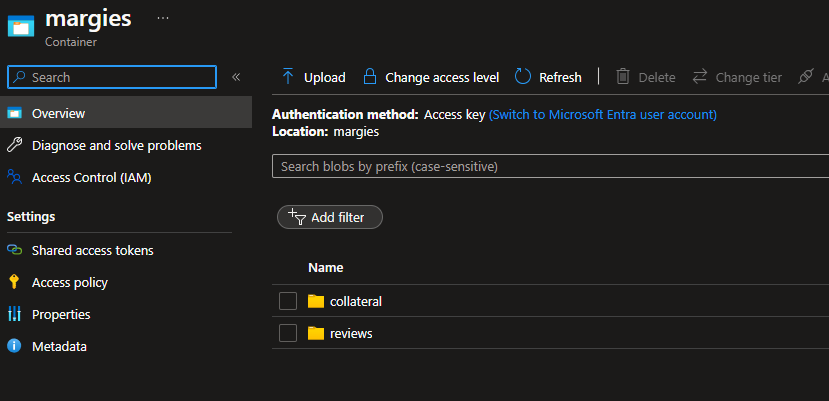
1. Upload all files from DATA folder to blob

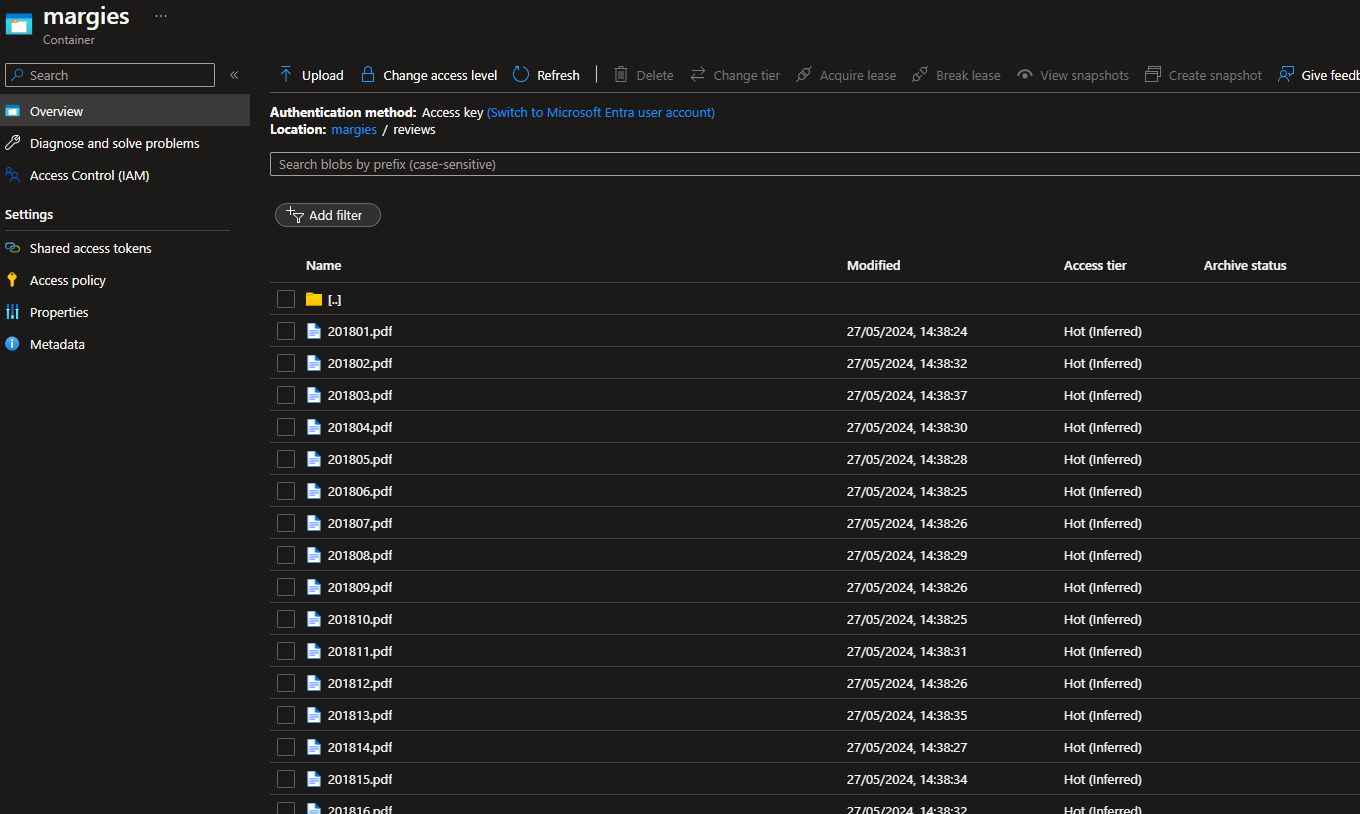
**az storage blob upload-batch -d margies -s data --account-name bipeensta078 --auth-mode key --account-key "t35NOAVuNNxOhScqx2k3Z7DBoe6O9cBhJ5SH7sn86vExm8Ey7Om+acPSz9SLj2KrkfrsQba/sn8H+AStOer/aw=="**

****

Go to Portal and Verify it

****

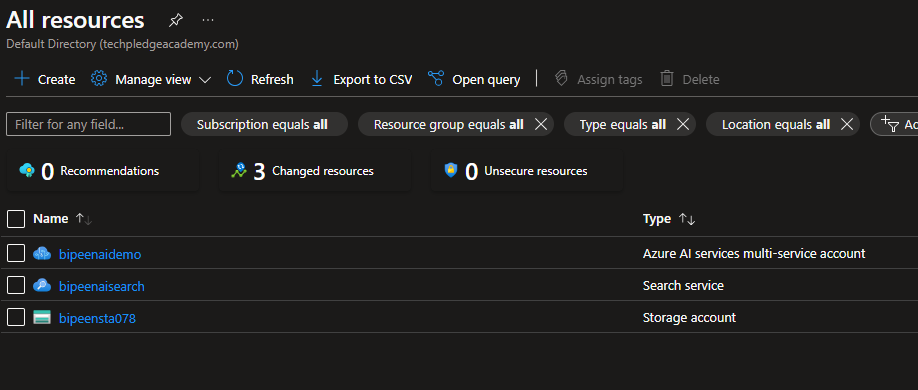
****

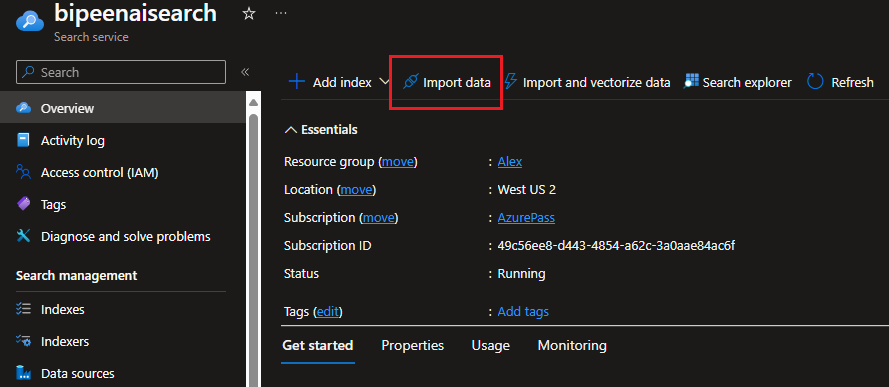
****

# Index the document

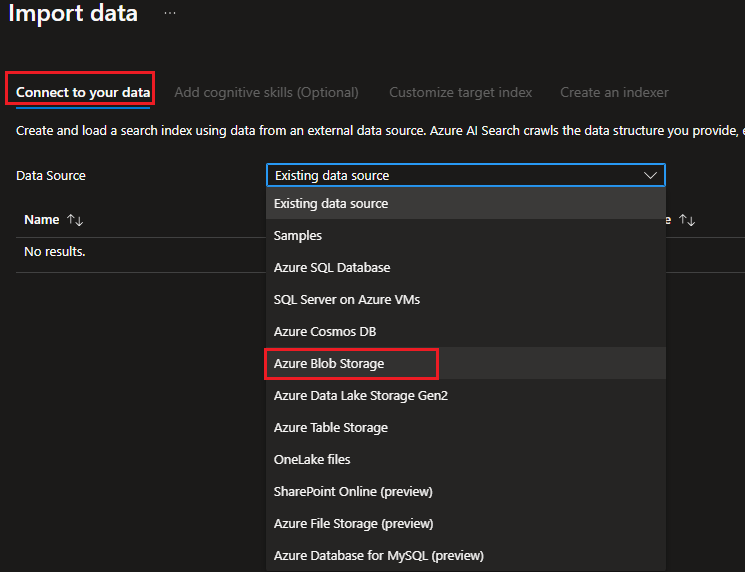
Now that you have the documents in place, you can create a search solution by indexing them.

1. In the Azure portal, browse to your Azure AI Search resource. Then, on its **Overview** page, select **Import data**.

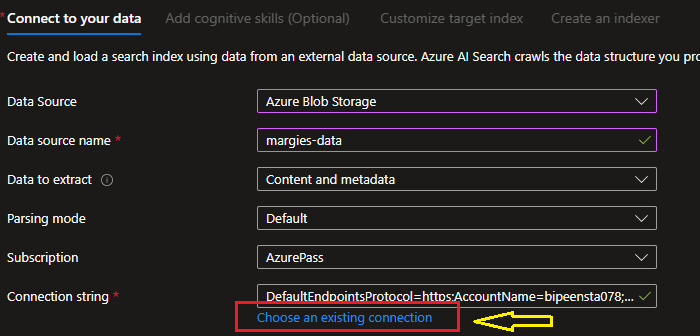


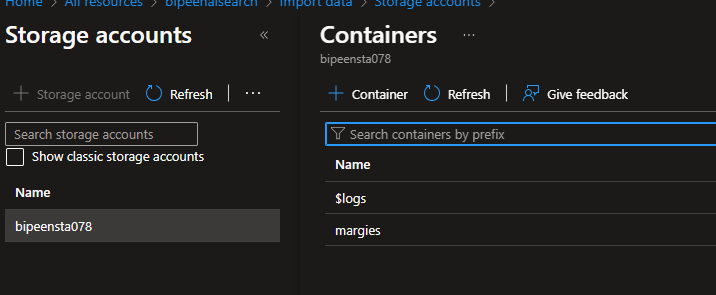


1. On the **Connect to your data** page, in the **Data Source** list, select **Azure Blob Storage**.

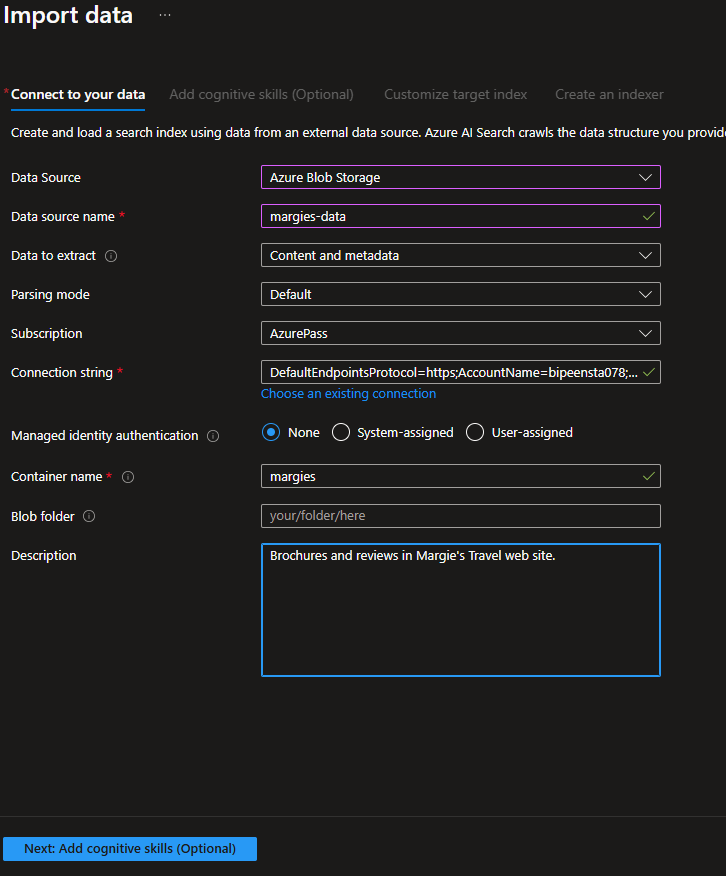


1. Then complete the data store details with the following values:
   * **Data Source**: Azure Blob Storage
   * **Data source name**: margies-data
   * **Data to extract**: Content and metadata
   * **Parsing mode**: Default
   * **Connection string**: *Select****Choose an existing connection****. Then select your storage account, and finally select the****margies****container that was created by the UploadDocs.cmd script.*

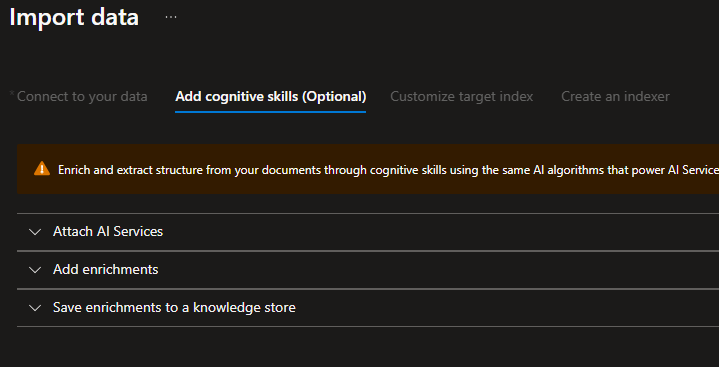
****

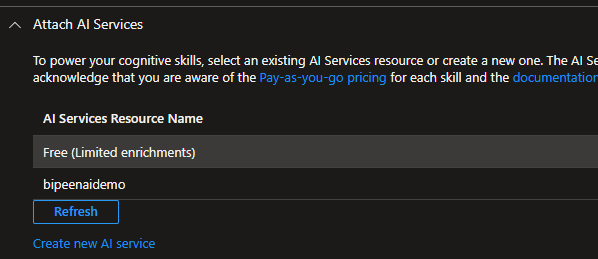


* + **Managed identity authentication**: None
  + **Container name**: margies
  + **Blob folder**: *Leave this blank*
  + **Description**: Brochures and reviews in Margie's Travel web site.



1. Proceed to the next step (*Add cognitive skills*).
2. in the **Attach Azure AI Services** section, select your Azure AI Services resource.

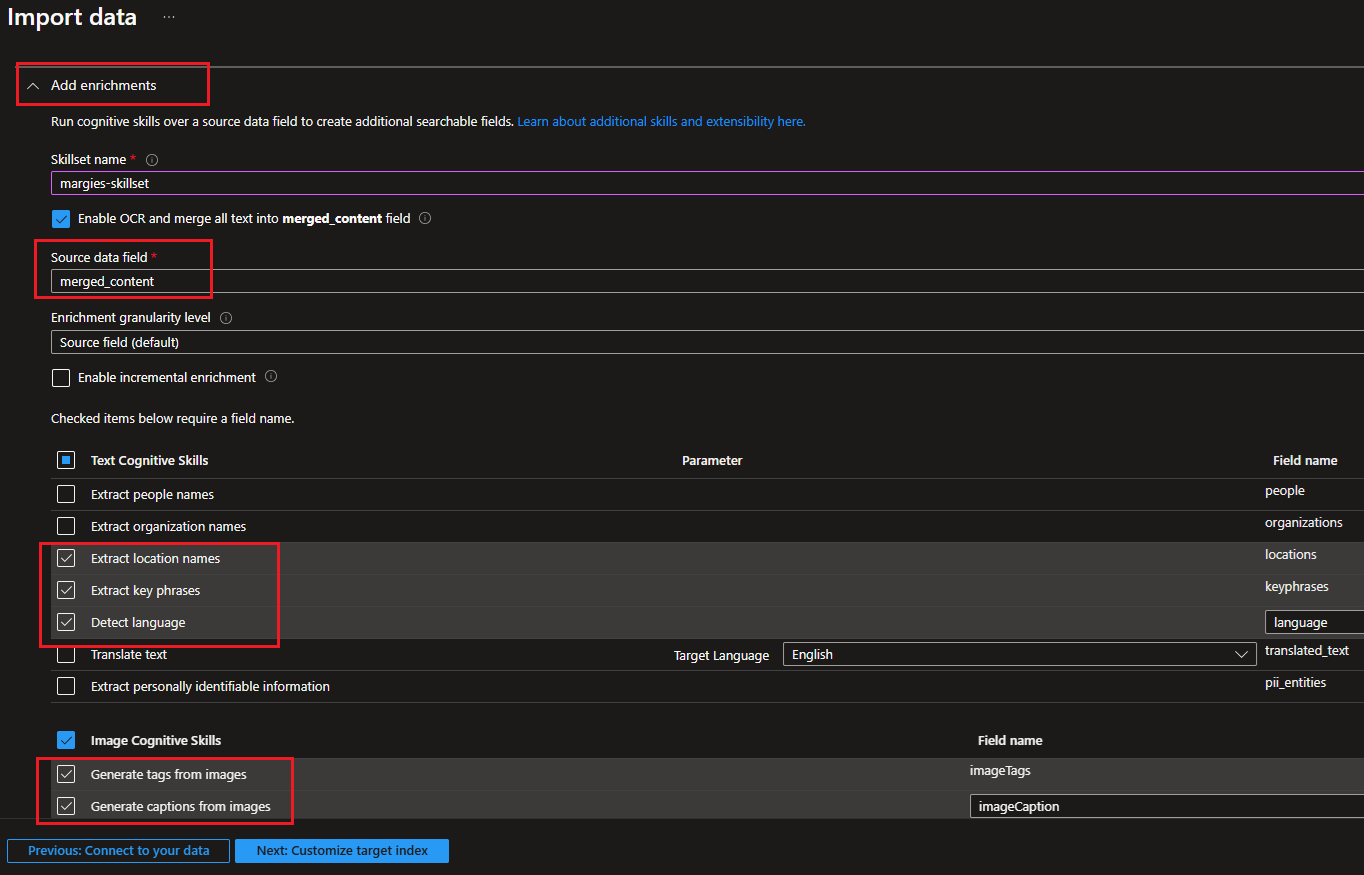




1. In the **Add enrichments** section:
   * Change the **Skillset name** to **margies-skillset**.
   * Select the option **Enable OCR and merge all text into merged\_content field**.
   * Ensure that the **Source data field** is set to **merged\_content**.
   * Leave the **Enrichment granularity level** as **Source field**, which is set the entire contents of the document being indexed; but note that you can change this to extract information at more granular levels, like pages or sentences.
   * Select the following enriched fields:

| **Cognitive Skill** | **Parameter** | **Field name** |
| --- | --- | --- |
| Extract location names |  | locations |
| Extract key phrases |  | keyphrases |
| Detect language |  | language |
| Generate tags from images |  | imageTags |
| Generate captions from images |  | imageCaption |

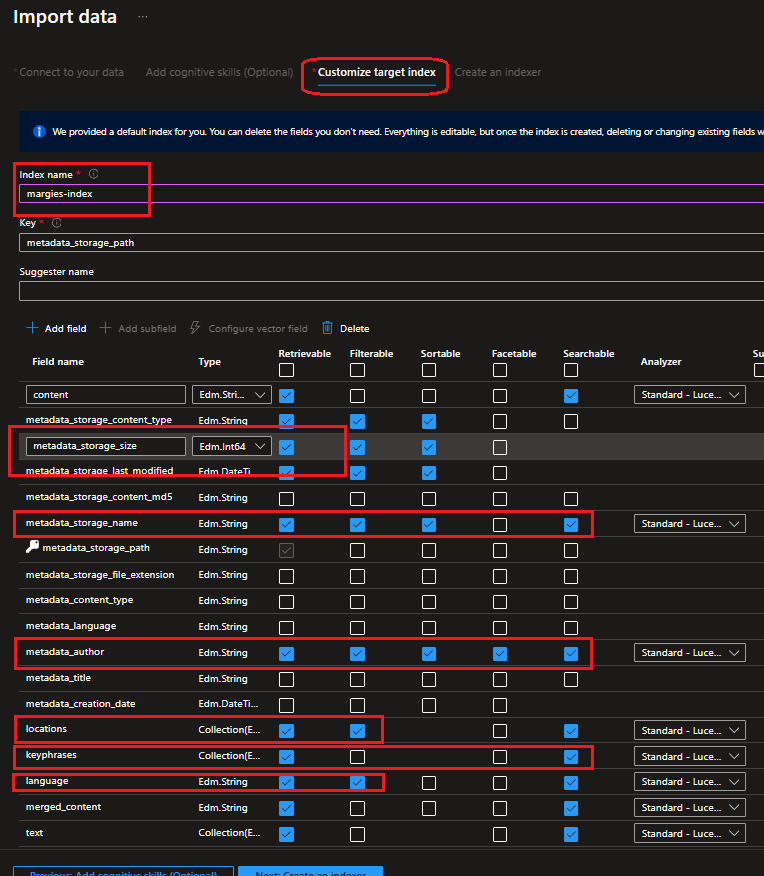
1. Double-check your selections (it can be difficult to change them later). Then proceed to the next step (*Customize target index*).



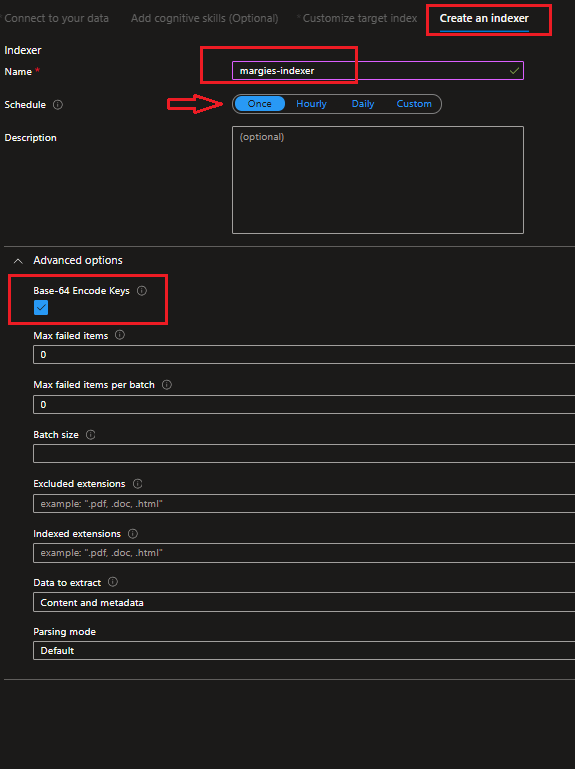
1. Change the **Index name** to **margies-index**.
2. Ensure that the **Key** is set to **metadata\_storage\_path** and leave the **Suggester name** blank and **Search mode** at its default.
3. Make the following changes to the index fields, leaving all other fields with their default settings (**IMPORTANT**: you may need to scroll to the right to see the entire table):

| **Field name** | **Retrievable** | **Filterable** | **Sortable** | **Facetable** | **Searchable** |
| --- | --- | --- | --- | --- | --- |
| metadata\_storage\_size | ✔ | ✔ | ✔ |  |  |
| metadata\_storage\_last\_modified | ✔ | ✔ | ✔ |  |  |
| metadata\_storage\_name | ✔ | ✔ | ✔ |  | ✔ |
| metadata\_author | ✔ | ✔ | ✔ | ✔ | ✔ |
| locations | ✔ | ✔ |  |  | ✔ |
| keyphrases | ✔ | ✔ |  |  | ✔ |
| language | ✔ | ✔ |  |  |  |

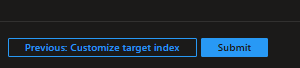
1. Double-check your selections, paying particular attention to ensure that the correct **Retrievable**, **Filterable**, **Sortable**, **Facetable**, and **Searchable** options are selected for each field (it can be difficult to change them later). Then proceed to the next step (*Create an indexer*).



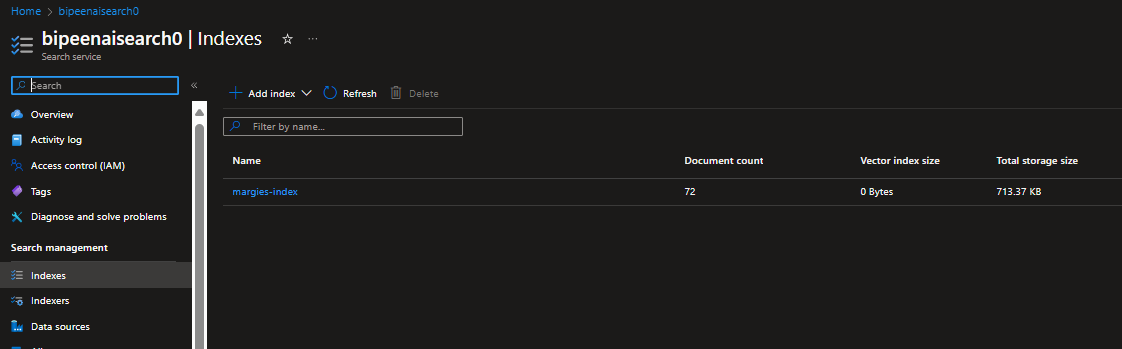
1. Change the **Indexer name** to **margies-indexer**.
2. Leave the **Schedule** set to **Once**.
3. Expand the **Advanced** options, and ensure that the **Base-64 encode keys** option is selected (generally encoding keys make the index more efficient).



1. Select **Submit** to create the data source, skillset, index, and indexer. The indexer is run automatically and runs the indexing pipeline, which:
   * Extracts the document metadata fields and content from the data source
   * Runs the skillset of cognitive skills to generate additional enriched fields
   * Maps the extracted fields to the index.



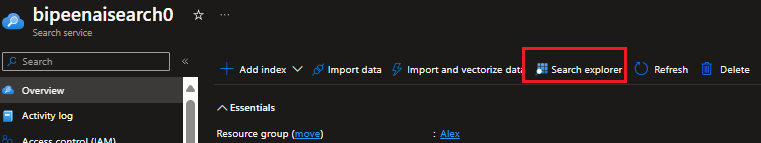
1. In the bottom half of the **Overview** page for your Azure AI Search resource, view the **Indexers** tab, which should show the newly created **margies-indexer**. Wait a few minutes, and click **↻ Refresh** until the **Status** indicates success

****

# Search the index

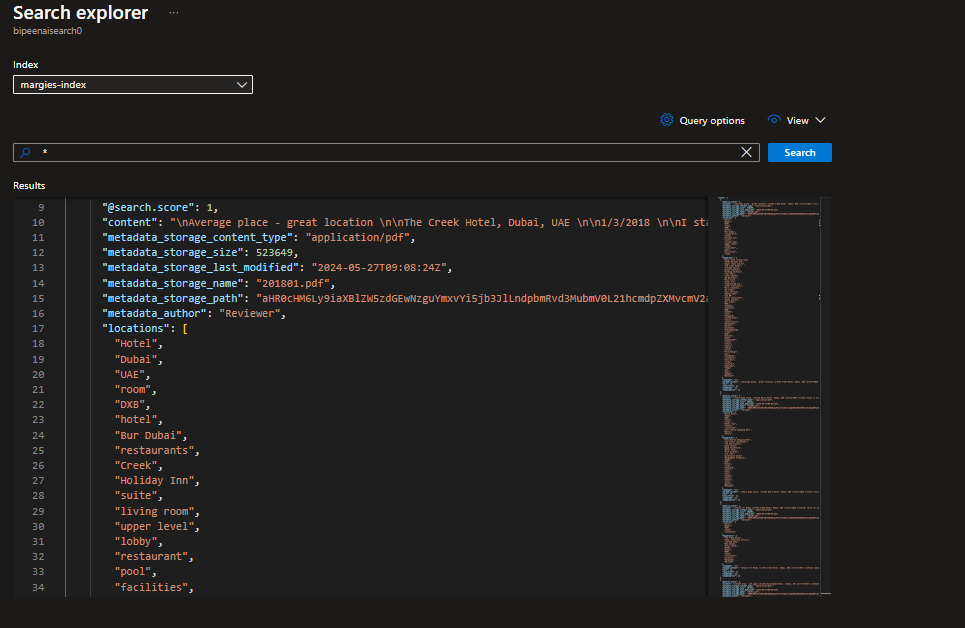
Now that you have an index, you can search it.

1. At the top of the **Overview** page for your Azure AI Search resource, select **Search explorer**.

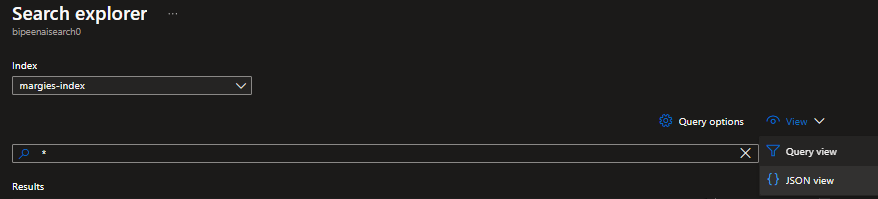


1. In Search explorer, in the **Query string** box, enter \* (a single asterisk), and then select **Search**.

This query retrieves all documents in the index in JSON format. Examine the results and note the fields for each document, which contain document content, metadata, and enriched data extracted by the cognitive skills you selected.



1. In the **View** menu, select **JSON view** and note that the JSON request for the search is shown, like this:

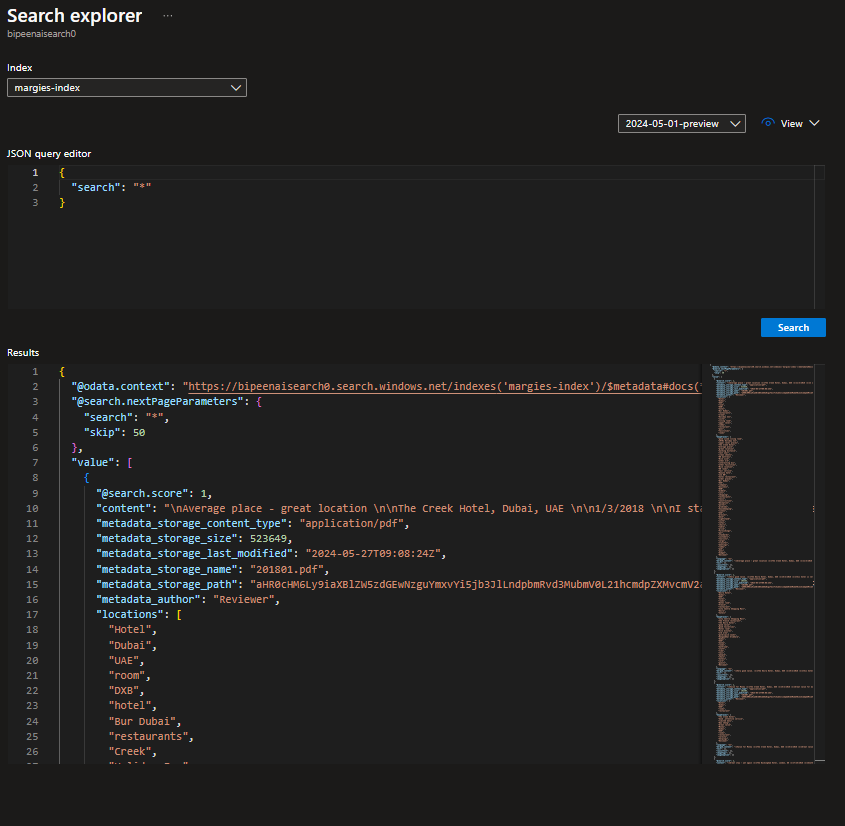


json

{

"search": "\*"

}



1. Modify the JSON request to include the **count** parameter as shown here:

json

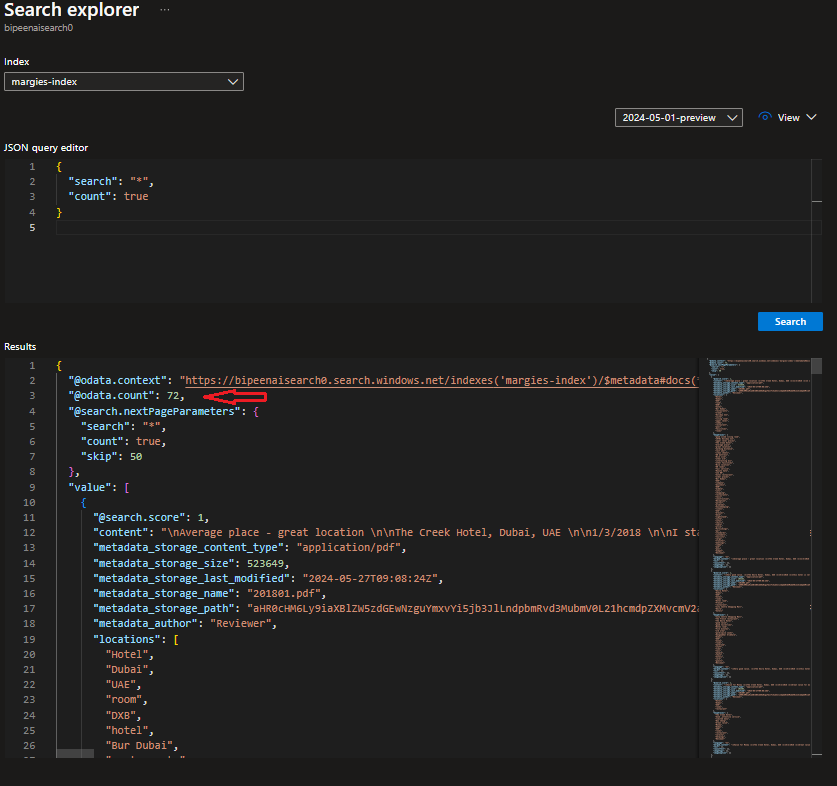
{

"search": "\*",

"count": true

}

1. Submit the modified search. This time, the results include a **@odata.count** field at the top of the results that indicates the number of documents returned by the search.



1. Try the following query:

json

{

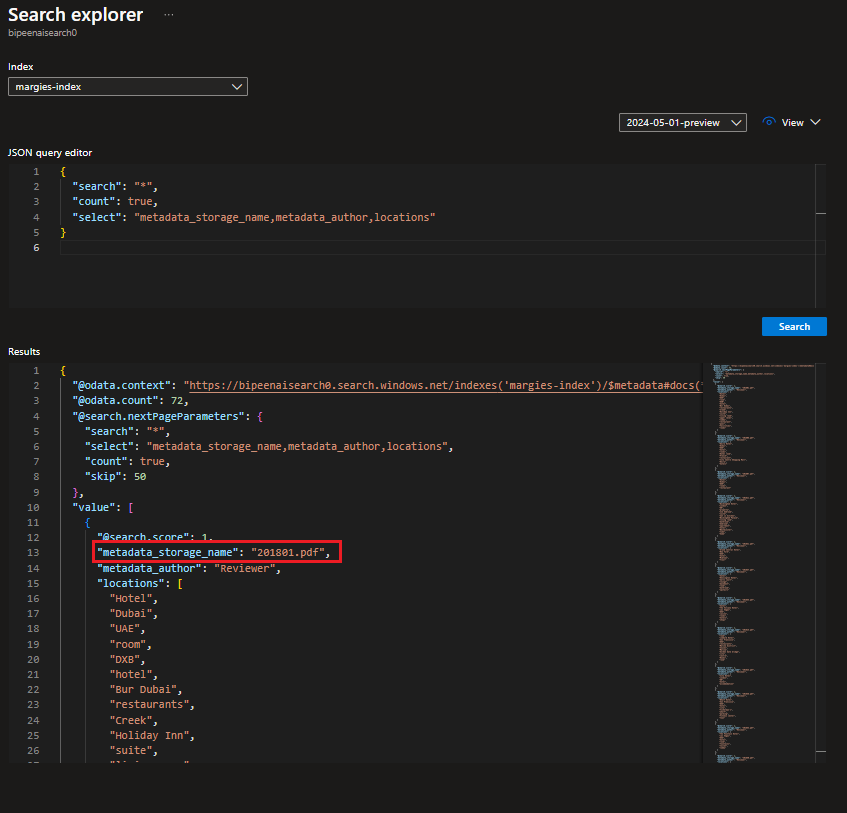
"search": "\*",

"count": true,

"select": "metadata\_storage\_name,metadata\_author,locations"

}

This time the results include only the file name, author, and any locations mentioned in the document content. The file name and author are in the **metadata\_storage\_name** and **metadata\_author** fields, which were extracted from the source document. The **locations** field was generated by a cognitive skill.



1. Now try the following query string:

json

{

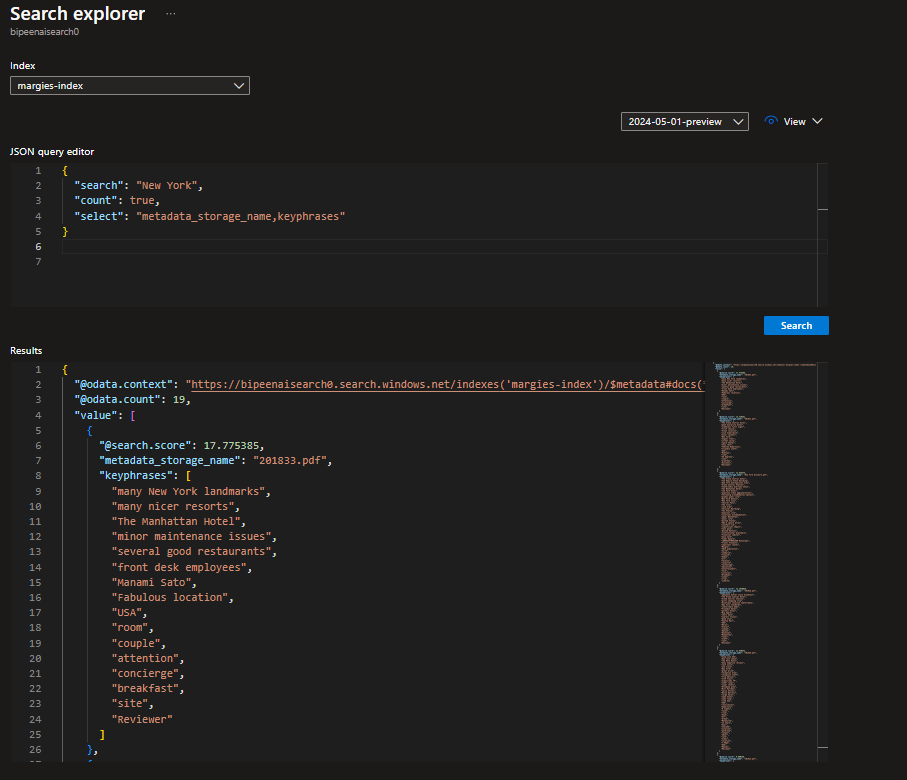
"search": "New York",

"count": true,

"select": "metadata\_storage\_name,keyphrases"

}

This search finds documents that mention "New York" in any of the searchable fields, and returns the file name and key phrases in the document.



1. Let's try one more query:

json

{

"search": "New York",

"count": true,

"select": "metadata\_storage\_name",

"filter": "metadata\_author eq 'Reviewer'"

}

This query returns the filename of any documents authored by Reviewer that mention "New York".

# Create a search client application

Now that you have a useful index, you can use it from a client application. You can do this by consuming the REST interface, submitting requests and receiving responses in JSON format over HTTP; or you can use the software development kit (SDK) for your preferred programming language. In this exercise, we'll use the SDK.

**Note**: You can choose to use the SDK for either **C#** or **Python**. In the steps below, perform the actions appropriate for your preferred language.

# Get the endpoint and keys for your search resource

1. In the Azure portal, on the **Overview** page for your Azure AI Search resource, note the **Url** value, which should be similar to **https://your\_resource\_name.search.windows.net**. This is the endpoint for your search resource.
2. On the **Keys** page, note that there are two **admin** keys, and a single **query** key. An admin key is used to create and manage search resources; a query key is used by client applications that only need to perform search queries.

You will need the endpoint and query key for your client application.

# Prepare to use the Azure AI Search SDK

1. In Visual Studio Code, in the **Explorer** pane, browse to the **01-azure-search** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.
2. Right-click the **margies-travel** folder and open an integrated terminal. Then install the Azure AI Search SDK package by running the appropriate command for your language preference:

**C#**

dotnet add package Azure.Search.Documents --version 11.1.1

**Python**

pip install azure-search-documents==11.0.0

1. View the contents of the **margies-travel** folder, and note that it contains a file for configuration settings:
   * **C#**: appsettings.json
   * **Python**: .env

Open the configuration file and update the configuration values it contains to reflect the **endpoint** and **query key** for your Azure AI Search resource. Save your changes.

### Explore code to search an index

The **margies-travel** folder contains code files for a web application (a Microsoft C# ASP.NET Razor web application or a Python Flask application), which includes search functionality.

1. Open the following code file in the web application, depending on your choice of programming language:
   * **C#**:Pages/Index.cshtml.cs
2. Near the top of the code file, find the comment **Import search namespaces**, and note the namespaces that have been imported to work with the Azure AI Search SDK:
3. In the **search\_query** function, find the comment **Create a search client**, and note that the code creates a **SearchClient** object using the endpoint and query key for your Azure AI Search resource:
4. In the **search\_query** function, find the comment **Submit search query**, and review the code to submit a search for the specified text with the following options:
   * A search mode that requires **all** of the individual words in the search text are found.
   * The total number of documents found by the search is included in the results.
   * The results are filtered to include only documents that match the provided filter expression.
   * The results are sorted into the specified sort order.
   * Each discrete value of the **metadata\_author** field is returned as a facet that can be used to display pre-defined values for filtering.
   * Up to three extracts of the **merged\_content** and **imageCaption** fields with the search terms highlighted are included in the results.
   * The results include only the fields specified.

# Explore code to render search results

The web app already includes code to process and render the search results.

1. Open the following code file in the web application, depending on your choice of programming language:
   * **C#**:Pages/Index.cshtml
2. Examine the code, which renders the page on which the search results are displayed. Observe that:
   * The page begins with a search form that the user can use to submit a new search (in the Python version of the application, this form is defined in the **base.html** template), which is referenced at the beginning of the page.
   * A second form is then rendered, enabling the user to refine the search results. The code for this form:
     + Retrieves and displays the count of documents from the search results.
     + Retrieves the facet values for the **metadata\_author** field and displays them as an option list for filtering.
     + Creates a drop-down list of sort options for the results.
   * The code then iterates through the search results, rendering each result as follows:
     + Display the **metadata\_storage\_name** (file name) field as a link to the address in the **url** field.
     + Displaying highlights for search terms found in the **merged\_content** and **imageCaption** fields to help show the search terms in context.
     + Display the **metadata\_author**, **metadata\_storage\_size**, **metadata\_storage\_last\_modified**, and **language** fields.
     + Display the **sentiment** label for the document. Can be positive, negative, neutral, or mixed.
     + Display the first five **keyphrases** (if any).
     + Display the first five **locations** (if any).
     + Display the first five **imageTags** (if any).

# Run the web app

1. return to the integrated terminal for the **margies-travel** folder, and enter the following command to run the program:

**C#**

dotnet run