Quickstart: Generative search (RAG) using grounding data from Azure AI Search

08/09/2025

In this quickstart, you send queries to a chat completion model for a conversational search experience over your indexed content on Azure AI Search. After setting up Azure OpenAI and Azure AI Search resources in the Azure portal, you run code to call the APIs.

Prerequisites

- An Azure account with an active subscription. Create an account for free .
- An Azure OpenAI resource.
  - Choose a region that supports the chat completion model you want to use (gpt-4o, gpt-4o-mini, or an equivalent model).
  - Deploy the chat completion model in Azure AI Foundry or use another approach.
- An Azure AI Search resource.
  - We recommend using the Basic tier or higher.
  - Enable semantic ranking.
- Visual Studio Code with the Python extension and the Jupyter package . For more information, see Python in Visual Studio Code .

Download file

Download a Jupyter notebook from GitHub to send the requests in this quickstart. For more information, see Downloading files from GitHub .

You can also start a new file on your local system and create requests manually by using the instructions in this article.

Configure access

Requests to the search endpoint must be authenticated and authorized. You can use API keys or roles for this task. Keys are easier to start with, but roles are more secure. This quickstart assumes roles.

You're setting up two clients, so you need permissions on both resources.

Azure AI Search is receiving the query request from your local system. Assign yourself the **Search Index Data Reader** role assignment if the hotels sample index already exists. If it doesn't exist, assign yourself **Search Service Contributor** and **Search Index Data Contributor** roles so that you can create and query the index.

Azure OpenAI is receiving the query and the search results from your local system. Assign yourself the **Cognitive Services OpenAI User** role on Azure OpenAI.

1. Sign in to the Azure portal .

2. Configure Azure AI Search for role-based access:

   a. In the Azure portal, find your Azure AI Search service.

   b. On the left menu, select **Settings** > **Keys**, and then select either **Role-based access control** or **Both**.

3. Assign roles:

   a. On the left menu, select **Access control (IAM)**.

   b. On Azure AI Search, select these roles to create, load, and query a search index, and then assign them to your Microsoft Entra ID user identity:

      - **Search Index Data Contributor**
      - **Search Service Contributor**

   c. On Azure OpenAI, select **Access control (IAM)** to assign this role to yourself on Azure OpenAI:

      - **Cognitive Services OpenAI User**

It can take several minutes for permissions to take effect.

Update the hotels-sample-index

A search index provides grounding data for the chat model. We recommend the hotels-sample-index, which can be created in minutes and runs on any search service tier. This index is created using built-in sample data.

1. In the Azure portal, find your search service .

2. On the **Overview** home page, select **Import data** to start the wizard.

3. On the **Connect to your data** page, select **Samples** from the dropdown list.

4. Choose the **hotels-sample**.

5. Select **Next** through the remaining pages, accepting the default values.

6. Once the index is created, select **Search management** > **Indexes** from the left menu to open the index.

7. Select **Edit JSON**.

8. Scroll to the end of the index, where you can find placeholders for constructs that can be added to an index.

```JSON
"analyzers": [],
"tokenizers": [],
"tokenFilters": [],
"charFilters": [],
"normalizers": [],
```

9. On a new line after "normalizers", paste in the following semantic configuration. This example specifies a `"defaultConfiguration"`, which is important to the running of this quickstart.

```JSON
"semantic":{
   "defaultConfiguration":"semantic-config",
   "configurations":[
      {
         "name":"semantic-config",
         "prioritizedFields":{
            "titleField":{
               "fieldName":"HotelName"
            },
            "prioritizedContentFields":[
               {
                  "fieldName":"Description"
               }
            ],
            "prioritizedKeywordsFields":[
               {
```

```
                "fieldName":"Category"
            },
            {
                "fieldName":"Tags"
            }
        ]
    }
  }
]
},
```

10. **Save** your changes.

11. Run the following query in Search Explorer to test your index: `complimentary breakfast`.

    Output should look similar to the following example. Results that are returned directly from the search engine consist of fields and their verbatim values, along with metadata like a search score and a semantic ranking score and caption if you use semantic ranker. We used a select statement to return just the HotelName, Description, and Tags fields.

```
{
"@odata.count": 18,
"@search.answers": [],
"value": [
   {
      "@search.score": 2.2896252,
      "@search.rerankerScore": 2.506816864013672,
      "@search.captions": [
      {
         "text": "Head Wind Resort. Suite. coffee in lobby\r\nfree wifi\r\nview.
The best of old town hospitality combined with views of the river and cool
breezes off the prairie. Our penthouse suites offer views for miles and the
rooftop plaza is open to all guests from sunset to 10 p.m. Enjoy a
**complimentary continental breakfast** in the lobby, and free Wi-Fi throughout
the hotel..",
         "highlights": ""
      }
      ],
      "HotelName": "Head Wind Resort",
      "Description": "The best of old town hospitality combined with views of
the river and cool breezes off the prairie. Our penthouse suites offer views for
miles and the rooftop plaza is open to all guests from sunset to 10 p.m. Enjoy a
complimentary continental breakfast in the lobby, and free Wi-Fi throughout the
hotel.",
      "Tags": [
      "coffee in lobby",
```

```
      "free wifi",
      "view"
      ]
   },
   {
      "@search.score": 2.2158256,
      "@search.rerankerScore": 2.288334846496582,
      "@search.captions": [
      {
         "text": "Swan Bird Lake Inn. Budget. continental breakfast\r\nfree
wifi\r\n24-hour front desk service. We serve a continental-style breakfast each
morning, featuring a variety of food and drinks. Our locally made, oh-so-soft,
caramel cinnamon rolls are a favorite with our guests. Other breakfast items in-
clude coffee, orange juice, milk, cereal, instant oatmeal, bagels, and
muffins..",
         "highlights": ""
      }
      ],
      "HotelName": "Swan Bird Lake Inn",
      "Description": "We serve a continental-style breakfast each morning, fea-
turing a variety of food and drinks. Our locally made, oh-so-soft, caramel cin-
namon rolls are a favorite with our guests. Other breakfast items include cof-
fee, orange juice, milk, cereal, instant oatmeal, bagels, and muffins.",
      "Tags": [
      "continental breakfast",
      "free wifi",
      "24-hour front desk service"
      ]
   },
   {
      "@search.score": 0.92481667,
      "@search.rerankerScore": 2.221315860748291,
      "@search.captions": [
      {
         "text": "White Mountain Lodge & Suites. Resort and Spa. continental
breakfast\r\npool\r\nrestaurant. Live amongst the trees in the heart of the for-
est. Hike along our extensive trail system. Visit the Natural Hot Springs, or
enjoy our signature hot stone massage in the Cathedral of Firs. Relax in the
meditation gardens, or join new friends around the communal firepit. Weekend
evening entertainment on the patio features special guest musicians or poetry
readings..",
         "highlights": ""
      }
      ],
      "HotelName": "White Mountain Lodge & Suites",
      "Description": "Live amongst the trees in the heart of the forest. Hike
along our extensive trail system. Visit the Natural Hot Springs, or enjoy our
signature hot stone massage in the Cathedral of Firs. Relax in the meditation
gardens, or join new friends around the communal firepit. Weekend evening enter-
tainment on the patio features special guest musicians or poetry readings.",
      "Tags": [
```

```
        "continental breakfast",
        "pool",
        "restaurant"
        ]
    },
    . . .
]}
```

Get service endpoints

In the remaining sections, you set up API calls to Azure OpenAI and Azure AI Search. Get the service endpoints so that you can provide them as variables in your code.

1. Sign in to the Azure portal .
2. Find your search service .
3. On the **Overview** home page, copy the URL. An example endpoint might look like `https://example.search.windows.net`.
4. Find your Azure OpenAI service .
5. On the **Overview** home page, select the link to view the endpoints. Copy the URL. An example endpoint might look like `https://example.openai.azure.com/`.

Create a virtual environment

In this step, switch back to your local system and Visual Studio Code. We recommend that you create a virtual environment so that you can install the dependencies in isolation.

1. In Visual Studio Code, open the folder containing Quickstart-RAG.ipynb.
2. Press Ctrl-shift-P to open the command palette, search for "Python: Create Environment", and then select `Venv` to create a virtual environment in the current workspace.
3. Select Quickstart-RAG\requirements.txt for the dependencies.

It takes several minutes to create the environment. When the environment is ready, continue to the next step.

Sign in to Azure

You're using Microsoft Entra ID and role assignments for the connection. Make sure you're logged in to the same tenant and subscription as Azure AI Search and Azure OpenAI. You can use the Azure CLI on the command line to show current properties, change properties, and to sign in. For more information, see Connect without keys.

Run each of the following commands in sequence.

azure-cli

```
az account show

az account set --subscription <PUT YOUR SUBSCRIPTION ID HERE>

az login --tenant <PUT YOUR TENANT ID HERE>
```

You should now be logged in to Azure from your local device.

Set up the query and chat thread

This section uses Visual Studio Code and Python to call the chat completion APIs on Azure OpenAI.

1. Start Visual Studio Code and open the .ipynb file or create a new Python file.

2. Install the following Python packages.

   Python

   ```
   ! pip install azure-search-documents==11.6.0b5 --quiet
   ! pip install azure-identity==1.16.1 --quiet
   ! pip install openai --quiet
   ! pip install aiohttp --quiet
   ! pip install ipykernel --quiet
   ```

3. Set the following variables, substituting placeholders with the endpoints you collected in the previous step.

   Python

   ```
   AZURE_SEARCH_SERVICE: str = "PUT YOUR SEARCH SERVICE ENDPOINT HERE"
   AZURE_OPENAI_ACCOUNT: str = "PUT YOUR AZURE OPENAI ENDPOINT HERE"
   AZURE_DEPLOYMENT_MODEL: str = "gpt-4o"
   ```

4. Set up clients, the prompt, query, and response.

   For the Azure Government cloud, modify the API endpoint on the token provider to `"https://cognitiveservices.azure.us/.default"`.

Python

```
# Set up the query for generating responses
 from azure.identity import DefaultAzureCredential
 from azure.identity import get_bearer_token_provider
 from azure.search.documents import SearchClient
 from openai import AzureOpenAI

 credential = DefaultAzureCredential()
 token_provider = get_bearer_token_provider(credential,
"https://cognitiveservices.azure.com/.default")
 openai_client = AzureOpenAI(
     api_version="2024-06-01",
     azure_endpoint=AZURE_OPENAI_ACCOUNT,
     azure_ad_token_provider=token_provider
 )

 search_client = SearchClient(
     endpoint=AZURE_SEARCH_SERVICE,
     index_name="hotels-sample-index",
     credential=credential
 )

 # This prompt provides instructions to the model
 GROUNDED_PROMPT="""
 You are a friendly assistant that recommends hotels based on activities and
amenities.
 Answer the query using only the sources provided below in a friendly and con-
cise bulleted manner.
 Answer ONLY with the facts listed in the list of sources below.
 If there isn't enough information below, say you don't know.
 Do not generate answers that don't use the sources below.
 Query: {query}
 Sources:\n{sources}
 """

 # Query is the question being asked. It's sent to the search engine and the
chat model
 query="Can you recommend a few hotels with complimentary breakfast?"

 # Search results are created by the search client
 # Search results are composed of the top 5 results and the fields selected from
the search index
 # Search results include the top 5 matches to your query
 search_results = search_client.search(
```

```
    search_text=query,
    top=5,
    select="Description,HotelName,Tags"
 )
 sources_formatted = "\n".join([f'{document["HotelName"]}:
{document["Description"]}:{document["Tags"]}' for document in search_results])

 # Send the search results and the query to the LLM to generate a response based
on the prompt.
 response = openai_client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": GROUNDED_PROMPT.format(query=query,
sources=sources_formatted)
        }
    ],
    model=AZURE_DEPLOYMENT_MODEL
 )

 # Here is the response from the chat model.
 print(response.choices[0].message.content)
```

Output is from Azure OpenAI, and it consists of recommendations for several hotels. Here's an example of what the output might look like:

```
Sure! Here are a few hotels that offer complimentary breakfast:

- **Head Wind Resort**
- Complimentary continental breakfast in the lobby
- Free Wi-Fi throughout the hotel

- **Double Sanctuary Resort**
- Continental breakfast included

- **White Mountain Lodge & Suites**
- Continental breakfast available

- **Swan Bird Lake Inn**
- Continental-style breakfast each morning with a variety of food and drinks
 such as caramel cinnamon rolls, coffee, orange juice, milk, cereal,
 instant oatmeal, bagels, and muffins
```

If you get a **Forbidden** error message, check Azure AI Search configuration to make sure role-based access is enabled.

If you get an **Authorization failed** error message, wait a few minutes and try again. It can take several minutes for role assignments to become operational.

If you get a **Resource not found** error message, check the resource URIs and make sure the API version on the chat model is valid.

Otherwise, to experiment further, change the query and rerun the last step to better understand how the model works with the grounding data.

You can also modify the prompt to change the tone or structure of the output.

You might also try the query without semantic ranking by setting `use_semantic_reranker=False` in the query parameters step. Semantic ranking can noticably improve the relevance of query results and the ability of the LLM to return useful information. Experimentation can help you decide whether it makes a difference for your content.

Send a complex RAG query

Azure AI Search supports complex types for nested JSON structures. In the hotels-sample-index, `Address` is an example of a complex type, consisting of `Address.StreetAddress`, `Address.City`, `Address.StateProvince`, `Address.PostalCode`, and `Address.Country`. The index also has complex collection of `Rooms` for each hotel.

If your index has complex types, your query can provide those fields if you first convert the search results output to JSON, and then pass the JSON to the chat model. The following example adds complex types to the request. The formatting instructions include a JSON specification.

Python

```
import json

# Query is the question being asked. It's sent to the search engine and the LLM.
query="Can you recommend a few hotels that offer complimentary breakfast?
Tell me their description, address, tags, and the rate for one room that sleeps 4
people."

# Set up the search results and the chat thread.
# Retrieve the selected fields from the search index related to the question.
selected_fields = ["HotelName","Description","Address","Rooms","Tags"]
search_results = search_client.search(
    search_text=query,
    top=5,
```

```
    select=selected_fields,
    query_type="semantic"
)
sources_filtered = [{field: result[field] for field in selected_fields} for result
in search_results]
sources_formatted = "\n".join([json.dumps(source) for source in sources_filtered])

response = openai_client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": GROUNDED_PROMPT.format(query=query, sources=sources_formatted)
        }
    ],
    model=AZURE_DEPLOYMENT_MODEL
)

print(response.choices[0].message.content)
```

Output is from Azure OpenAI, and it adds content from complex types.

```
Here are a few hotels that offer complimentary breakfast and have rooms that sleep 4
people:

1. **Head Wind Resort**
   - **Description:** The best of old town hospitality combined with views of the
river and
   cool breezes off the prairie. Enjoy a complimentary continental breakfast in the
lobby,
   and free Wi-Fi throughout the hotel.
   - **Address:** 7633 E 63rd Pl, Tulsa, OK 74133, USA
   - **Tags:** Coffee in lobby, free Wi-Fi, view
   - **Room for 4:** Suite, 2 Queen Beds (Amenities) - $254.99

2. **Double Sanctuary Resort**
   - **Description:** 5-star Luxury Hotel - Biggest Rooms in the city. #1 Hotel in
the area
   listed by Traveler magazine. Free WiFi, Flexible check in/out, Fitness Center &
espresso
   in room. Offers continental breakfast.
   - **Address:** 2211 Elliott Ave, Seattle, WA 98121, USA
   - **Tags:** View, pool, restaurant, bar, continental breakfast
   - **Room for 4:** Suite, 2 Queen Beds (Amenities) - $254.99

3. **Swan Bird Lake Inn**
   - **Description:** Continental-style breakfast featuring a variety of food and
drinks.
   Locally made caramel cinnamon rolls are a favorite.
```

```
   - **Address:** 1 Memorial Dr, Cambridge, MA 02142, USA
   - **Tags:** Continental breakfast, free Wi-Fi, 24-hour front desk service
   - **Room for 4:** Budget Room, 2 Queen Beds (City View) - $85.99

4. **Gastronomic Landscape Hotel**
   - **Description:** Known for its culinary excellence under the management of
William Dough,
   offers continental breakfast.
   - **Address:** 3393 Peachtree Rd, Atlanta, GA 30326, USA
   - **Tags:** Restaurant, bar, continental breakfast
   - **Room for 4:** Budget Room, 2 Queen Beds (Amenities) - $66.99
...
   - **Tags:** Pool, continental breakfast, free parking
   - **Room for 4:** Budget Room, 2 Queen Beds (Amenities) - $60.99

Enjoy your stay! Let me know if you need any more information.
```

Troubleshooting errors

To debug authentication errors, insert the following code before the step that calls the search engine and the LLM.

Python

```python
import sys
import logging # Set the logging level for all azure-storage-* libraries
logger = logging.getLogger('azure.identity')
logger.setLevel(logging.DEBUG)

handler = logging.StreamHandler(stream=sys.stdout)
formatter = logging.Formatter('[%(levelname)s %(name)s] %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)
```

Rerun the query script. You should now get INFO and DEBUG statements in the output that provide more detail about the issue.

If you see output messages related to ManagedIdentityCredential and token acquisition failures, it could be that you have multiple tenants, and your Azure sign-in is using a tenant that doesn't have your search service. To get your tenant ID, search the Azure portal for "tenant properties" or run `az login tenant list`.

Once you have your tenant ID, run `az login --tenant <YOUR-TENANT-ID>` at a command prompt, and then rerun the script.

Clean up

When you're working in your own subscription, it's a good idea at the end of a project to identify whether you still need the resources you created. Resources left running can cost you money. You can delete resources individually or delete the resource group to delete the entire set of resources.

You can find and manage resources in the Azure portal by using the **All resources** or **Resource groups** link in the leftmost pane.

Related content

- Tutorial: Build a RAG solution in Azure AI Search