Searching Documents

I'll guide you through the process of working with an Azure AI Search service in Python, from setting up the service to querying documents. This will build on your existing Azure resource management knowledge.

Step-by-Step Guide to Using Azure AI Search in Python

1. Install Required Packages

First, make sure you have the necessary packages:

```
pip install azure-identity azure-mgmt-search azure-search-documents
```

2. Create an Azure AI Search Service (if you don't have one)

```
from azure.identity import AzureCliCredential
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.search import SearchManagementClient

# Authenticate
credential = AzureCliCredential()

# Get subscription ID from available subscriptions
from azure.mgmt.resource import SubscriptionClient
subscription_client = SubscriptionClient(credential)
subscription = next(subscription_client.subscriptions.list())
subscription_id = subscription.subscription_id

# Create Search Management client
search_mgmt_client = SearchManagementClient(credential, subscription_id)

# Parameters for creating search service
resource_group_name = "YourResourceGroup"  # Use an existing resource
group
service_name = "yoursearchservice"  # Must be globally unique, lowercase,
no spaces
location = "eastus"  # Choose an appropriate region

# Define the search service
search_service = {
    "location": location,
    "sku": {
        "name": "basic"  # Options: free, basic, standard, standard2,
standard3, storage_optimized_l1, storage_optimized_l2
    }
}

# Create the search service (this may take a few minutes)
```

```
print(f"Creating search service '{service_name}'...")
operation = search_mgmt_client.services.begin_create_or_update(
    resource_group_name=resource_group_name,
    search_service_name=service_name,
    service=search_service
)
search_service = operation.result()
print(f"Search service '{service_name}' created successfully.")

# Get the admin key
admin_key = search_mgmt_client.admin_keys.get(
    resource_group_name=resource_group_name,
    search_service_name=service_name
).primary_key

print(f"Service Endpoint: https://{service_name}.search.windows.net")
print(f"Admin Key: {admin_key}")
```

3. Create an Index and Upload Documents

Now, let's create a simple index and upload some sample documents:

```
from azure.search.documents import SearchClient
from azure.search.documents.indexes import SearchIndexClient
from azure.search.documents.indexes.models import (
    SearchIndex,
    SearchField,
    SearchFieldDataType,
    SimpleField,
    SearchableField
)
from azure.core.credentials import AzureKeyCredential

# Define search service details
service_endpoint = f"https://{service_name}.search.windows.net"
index_name = "sample-index"
credential = AzureKeyCredential(admin_key)

# Create a search index client
index_client = SearchIndexClient(endpoint=service_endpoint,
credential=credential)

# Define the index structure
fields = [
    SimpleField(name="id", type=SearchFieldDataType.String, key=True),
    SearchableField(name="title", type=SearchFieldDataType.String),
    SearchableField(name="content", type=SearchFieldDataType.String),
    SimpleField(name="category", type=SearchFieldDataType.String,
filterable=True, facetable=True)
]
```

```
# Create the index
index = SearchIndex(name=index_name, fields=fields)
result = index_client.create_or_update_index(index)
print(f"Index '{index_name}' created successfully.")

# Create a search client for uploading documents and querying
search_client = SearchClient(endpoint=service_endpoint,
index_name=index_name, credential=credential)

# Sample documents to upload
documents = [
    {
        "id": "1",
        "title": "Azure AI Search Overview",
        "content": "Azure AI Search is a cloud search service that gives
developers APIs and tools for building rich search experiences over
private, heterogeneous content in web, mobile, and enterprise
applications.",
        "category": "Azure Services"
    },
    {
        "id": "2",
        "title": "Python Development Guide",
        "content": "Python is a popular programming language known for its
simplicity and readability. This guide covers the basics of Python
development for Azure services.",
        "category": "Development"
    },
    {
        "id": "3",
        "title": "Machine Learning with Azure",
        "content": "Azure provides various services for machine learning
and AI. This document explains how to build ML models using Azure Machine
Learning.",
        "category": "Machine Learning"
    }
]

# Upload documents to the index
result = search_client.upload_documents(documents=documents)
print(f"Uploaded {len(result)} documents")
```

4. Query Documents from the Index

Now you can search for documents:

```
def search_documents(search_client, query_text, filter_expression=None):
    """Search for documents in the index"""

    # Define search options
    search_options = {
```

```
        "include_total_count": True,
        "select": "id,title,category",
        "highlight_fields": "content",
        "highlight_pre_tag": "<b>",
        "highlight_post_tag": "</b>"
    }

    # Add filter if specified
    if filter_expression:
        search_options["filter"] = filter_expression

    # Execute the search
    results = search_client.search(query_text, **search_options)

    # Process the results
    print(f"\nSearch results for: '{query_text}'")

    count = 0
    for result in results:
        count += 1
        print(f"\nDocument ID: {result['id']}")
        print(f"Title: {result['title']}")
        print(f"Category: {result['category']}")

        # Display highlighted content if available
        if 'content' in result.get('@search.highlights', {}):
            print("Content highlights:")
            for highlight in result['@search.highlights']['content']:
                print(f"  ...{highlight}...")

    print(f"\nFound {count} documents")

# Example 1: Simple query
search_documents(search_client, "Azure")

# Example 2: Query with filter
search_documents(search_client, "development", "category eq
'Development'")
```

5. Perform Semantic Search (if you have Standard tier or higher)

If you have a higher tier Azure AI Search service, you can use semantic search capabilities:

```
def semantic_search(search_client, query_text):
    """Perform semantic search with AI ranking"""

    # Define semantic search options
    semantic_options = {
        "query_type": "semantic",
        "semantic_configuration_name": "default",
        "query_caption": "extractive",
```

```
        "query_answer": "extractive",
        "include_total_count": True
    }

    # Execute the search
    results = search_client.search(query_text, **semantic_options)

    # Process the results
    print(f"\nSemantic search results for: '{query_text}'")

    count = 0
    for result in results:
        count += 1
        print(f"\nDocument ID: {result['id']}")
        print(f"Title: {result['title']}")

        # Display semantic caption if available
        if '@search.captions' in result:
            for caption in result['@search.captions']:
                print(f"Caption: {caption['text']}")

    print(f"\nFound {count} documents")

# Example: Semantic search (Only works with Standard S1 or higher tier)
# Note: You'll need to create a semantic configuration first
# semantic_search(search_client, "How does Azure AI Search work?")
```

6. Complete Script with Error Handling

Here's a complete script that combines all these functionalities with proper error handling:

```
from azure.identity import AzureCliCredential
from azure.mgmt.search import SearchManagementClient
from azure.search.documents import SearchClient
from azure.search.documents.indexes import SearchIndexClient
from azure.search.documents.indexes.models import (
    SearchIndex,
    SearchField,
    SearchFieldDataType,
    SimpleField,
    SearchableField
)
from azure.core.credentials import AzureKeyCredential
from azure.core.exceptions import ResourceNotFoundError, HttpResponseError

def main():
    try:
        # Authenticate
        credential = AzureCliCredential()

        # Get subscription ID
```

```
        from azure.mgmt.resource import SubscriptionClient
        subscription_client = SubscriptionClient(credential)
        subscription = next(subscription_client.subscriptions.list())
        subscription_id = subscription.subscription_id

        print(f"Using subscription: {subscription.display_name}
({subscription_id})")

        # Get your search service details
        resource_group_name = input("Enter your resource group name: ")
        service_name = input("Enter your search service name: ")

        # Create Search Management client
        search_mgmt_client = SearchManagementClient(credential,
subscription_id)

        # Get the admin key
        try:
            admin_key = search_mgmt_client.admin_keys.get(
                resource_group_name=resource_group_name,
                search_service_name=service_name
            ).primary_key

            print(f"Successfully connected to search service:
{service_name}")

            # Work with the search service
            service_endpoint =
f"https://{service_name}.search.windows.net"

            # Use the search service
            work_with_search_service(service_endpoint, admin_key)

        except ResourceNotFoundError:
            print(f"Error: Search service '{service_name}' not found in
resource group '{resource_group_name}'")

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

def work_with_search_service(service_endpoint, admin_key):
    """Work with an existing search service"""
    credential = AzureKeyCredential(admin_key)

    # Create index client
    index_client = SearchIndexClient(endpoint=service_endpoint,
credential=credential)

    # List existing indexes
    print("\nExisting indexes:")
    indexes = list(index_client.list_indexes())
    for index in indexes:
        print(f"- {index.name}")
```

```
    # Create a new index or work with existing one
    use_existing = False
    if indexes:
        response = input("\nDo you want to use an existing index? (y/n):
").lower()
        use_existing = response == 'y'

    if use_existing:
        index_name = input("Enter the name of the existing index: ")
    else:
        index_name = input("Enter a name for the new index: ")
        create_sample_index(index_client, index_name)

    # Create a search client
    search_client = SearchClient(endpoint=service_endpoint,
index_name=index_name, credential=credential)

    # Upload documents if it's a new index
    if not use_existing:
        upload_sample_documents(search_client)

    # Interactive search
    while True:
        query = input("\nEnter a search query (or 'exit' to quit): ")
        if query.lower() == 'exit':
            break

        search_documents(search_client, query)

def create_sample_index(index_client, index_name):
    """Create a sample search index"""

    # Define the index structure
    fields = [
        SimpleField(name="id", type=SearchFieldDataType.String, key=True),
        SearchableField(name="title", type=SearchFieldDataType.String),
        SearchableField(name="content", type=SearchFieldDataType.String),
        SimpleField(name="category", type=SearchFieldDataType.String,
filterable=True, facetable=True)
    ]

    # Create the index
    index = SearchIndex(name=index_name, fields=fields)

    try:
        result = index_client.create_or_update_index(index)
        print(f"Index '{index_name}' created successfully.")
    except HttpResponseError as e:
        print(f"Error creating index: {str(e)}")
        exit(1)

def upload_sample_documents(search_client):
    """Upload sample documents to the index"""
```

```
    # Sample documents
    documents = [
        {
            "id": "1",
            "title": "Azure AI Search Overview",
            "content": "Azure AI Search is a cloud search service that
gives developers APIs and tools for building rich search experiences over
private, heterogeneous content in web, mobile, and enterprise
applications.",
            "category": "Azure Services"
        },
        {
            "id": "2",
            "title": "Python Development Guide",
            "content": "Python is a popular programming language known for
its simplicity and readability. This guide covers the basics of Python
development for Azure services.",
            "category": "Development"
        },
        {
            "id": "3",
            "title": "Machine Learning with Azure",
            "content": "Azure provides various services for machine
learning and AI. This document explains how to build ML models using Azure
Machine Learning.",
            "category": "Machine Learning"
        }
    ]

    try:
        result = search_client.upload_documents(documents=documents)
        print(f"Uploaded {len(result)} documents")
    except HttpResponseError as e:
        print(f"Error uploading documents: {str(e)}")

def search_documents(search_client, query_text, filter_expression=None):
    """Search for documents in the index"""

    try:
        # Define search options
        search_options = {
            "include_total_count": True,
            "highlight_fields": "content",
            "highlight_pre_tag": "<b>",
            "highlight_post_tag": "</b>"
        }

        # Add filter if specified
        if filter_expression:
            search_options["filter"] = filter_expression

        # Execute the search
        results = search_client.search(query_text, **search_options)
```

```
        # Process the results
        print(f"\nSearch results for: '{query_text}'")

        count = 0
        for result in results:
            count += 1
            print(f"\nDocument ID: {result['id']}")
            print(f"Title: {result['title']}")
            print(f"Category: {result['category']}")

            # Display highlighted content if available
            if 'content' in result.get('@search.highlights', {}):
                print("Content highlights:")
                for highlight in result['@search.highlights']['content']:
                    print(f"  ...{highlight}...")

        print(f"\nFound {count} documents")

    except HttpResponseError as e:
        print(f"Search error: {str(e)}")

if __name__ == "__main__":
    main()
```

7. Working with Your Own Documents

To work with your own documents in a real-world scenario, you'd typically:

1. Design an index schema that matches your data structure
2. Upload your documents from a data source (e.g., SQL database, blob storage)
3. Possibly create an indexer to keep your index up-to-date

Here's how you might load documents from a JSON file:

```
import json

def load_documents_from_file(file_path):
    """Load documents from a JSON file"""
    with open(file_path, 'r') as file:
        return json.load(file)

# Example usage:
# documents = load_documents_from_file('your_documents.json')
# result = search_client.upload_documents(documents=documents)
```

This comprehensive guide should help you get started with Azure AI Search in Python. You can expand on these fundamentals to build more complex search solutions for your specific needs.