AI-102-23: AI-102 ラボ 23 (検索カスタムスキル)

1 時間 59 分 残り

指示 リソース ヘルプ  100%

# 開始する前に

**ヒント**: このウィンドウの指示に従って、アイコンが表示されるたびに、それを使用して指示ペインから仮想マシンのインターフェイスにテキストをコピーできます。これは特にコードをコピーする場合に便利です。ただし、実行する前に、インデントレベルや書式を修正するために、貼り付けたコードを修正する必要がある場合があります。

1. プロンプトが表示されたら、パスワード Pa55w.rd を使用して **Student** アカウントにログインします。PC を検出できるように求めるメッセージが表示されたら、[いいえ] を選択**します**。
2. Azure サブスクリプションをまだお持ちでない場合は、https://azure.microsoft.com/free/ で無料試用版にサインアップ[してください](https://azure.microsoft.com/free/" \t "_blank)。

# Azure コグニティブ検索のカスタム スキルを作成する

Azure Cognitive Search では、認知スキルのエンリッチメント パイプラインを使用して、AI で生成されたフィールドをドキュメントから抽出し、検索インデックスに含めます。使用できる組み込みスキルの包括的なセットがありますが、これらのスキルに満たない特定の要件がある場合は、カスタム スキルを作成できます。

この演習では、ドキュメント内の個々の単語の頻度を表すカスタム スキルを作成して、最も使用されている上位 5 単語のリストを生成し、架空の旅行代理店である Margie's Travel の検索ソリューションに追加します。

## このコースのリポジトリのクローンを作成する

**AI-102-AIEngineer** コード リポジトリを、このラボで作業している環境に既にクローンしている場合は、Visual Studio コードで開きます。それ以外の場合は、次の手順に従って、今すぐ複製します。

1. ビジュアル スタジオ コードを起動します。
2. パレットを開き (SHIFT + Ctrl + P) し、**Git: Clone** コマンドを実行してリポジトリをローカル フォルダーに複製します (どのフォルダーにクローンを作成しても構いません)。https://github.com/MicrosoftLearning/AI-102-AIEngineer
3. リポジトリが複製されたら、Visual Studio コードでフォルダーを開きます。
4. 追加のファイルがインストールされ、Repo の C# コード プロジェクトをサポートするまで待ちます。

**注**: ビルドおよびデバッグに必要なアセットを追加するよう求められた場合は、[**今は使用しない**] を選択します。

## Azure リソースの作成

**注**: 以前に Azure [**Cognitive Search ソリューションの作成**](https://raw.githubusercontent.com/MicrosoftLearning/AI-102-AIEngineer/master/Instructions/22-azure-search.md)の演習を完了し、サブスクリプションにこれらの Azure リソースを含めている場合は、このセクションをスキップして**「検索ソリューションの作成**」セクションから開始できます。それ以外の場合は、次の手順に従って、必要な Azure リソースをプロビジョニングします。

1. Web ブラウザーで、Azure ポータル を開き、Azure サブスクリプションに関連付けられている Microsoft アカウントを使用してサインインします。https://portal.azure.com
2. サブスクリプションの**リソース グループ**を表示します。
3. リソース グループが提供されている制限付きサブスクリプションを使用している場合は、リソース グループを選択してプロパティを表示します。それ以外の場合は、任意の名前で新しいリソース グループを作成し、作成後に移動します。
4. リソース グループの [**概要**] ページで、[**サブスクリプション ID**] と **[場所**] をメモします。これらの値は、以降の手順でリソース グループの名前と共に必要になります。
5. Visual Studio コードで、**23 カスタム検索スキル** フォルダーを展開し、**setup.cmd** を選択します。このバッチ スクリプトを使用して、必要な Azure リソースを作成するために必要な Azure コマンド ライン インターフェイス (CLI) コマンドを実行します。
6. **23 カスタム検索スキル**フォルダを右クリックし、[**統合端末で開く**] を選択します。
7. ターミナル ウィンドウで、次のコマンドを入力して、Azure サブスクリプションへの認証済み接続を確立します。

az login --output none

1. プロンプトが表示されたら、Azure サブスクリプションにサインインします。次に、Visual Studio コードに戻り、サインイン プロセスが完了するまで待ちます。
2. 次のコマンドを実行して、Azure の場所を一覧表示します。

az account list-locations -o table

1. 出力で、リソース グループの場所に対応する **Name** 値を見つけます (たとえば、米国東部の場合は、対応する名前は eastus です)。
2. **setup.cmd** スクリプトで、**subscription\_id**、**resource\_group**、および**場所**の変数宣言を、サブスクリプション ID、リソース グループ名、および場所名に適切な値で変更します。その後、変更を保存します。
3. **23 カスタム検索スキル**フォルダのターミナルで、次のコマンドを入力してスクリプトを実行します。

setup

**注**: 検索 CLI モジュールはプレビュー段階にあり、-実行中の .. プロセスでスタックする可能性があります。この現象が 2 分を超える場合は、Ctrl + C キーを押して実行時間の長い操作をキャンセルし、スクリプトを終了するかどうか尋ねられたときに **N** を選択します。その後、正常に完了するはずです。

スクリプトが失敗した場合は、正しい変数名を使用して保存したことを確認してから、もう一度やり直してください。

1. スクリプトが完了したら、表示される出力を確認し、Azure リソースに関する次の情報を確認します (これらの値は後で必要になります)。
   * ストレージ アカウント名
   * ストレージ接続文字列
   * コグニティブ サービス アカウント
   * コグニティブ サービスキー
   * 検索サービス エンドポイント
   * 検索サービス管理キー
   * 検索サービス クエリ キー
2. Azure ポータルで、リソース グループを更新し、Azure ストレージ アカウント、Azure コグニティブ サービス リソース、および Azure コグニティブ 検索リソースが含まれていることを確認します。

## 検索ソリューションを作成する

必要な Azure リソースを用意したので、次のコンポーネントで構成される検索ソリューションを作成できます。

* Azure ストレージ コンテナー内のドキュメントを参照する**データ ソース**。
* AI 生成フィールドをドキュメントから抽出するスキルのエンリッチメント パイプラインを定義する**スキルセット**。
* 検索可能なドキュメント レコードのセットを定義する**インデックス**。
* データ ソースからドキュメントを抽出し、スキルセットを適用し、インデックスを設定する**インデクサー**。

この演習では、Azure コグニティブ検索 REST インターフェイスを使用して、JSON 要求を送信してこれらのコンポーネントを作成します。

1. Visual Studio コードの **23 カスタム検索スキル** フォルダーで、検索フォルダーの**作成**を展開し、**data\_source.json** を選択します。このファイルには、**margies-custom-data** という名前のデータソースの JSON 定義が含まれています。
2. **YOUR\_CONNECTION\_STRING**のプレースホルダーを、次のような Azure ストレージ アカウントの接続文字列に置き換えます。

DefaultEndpointsProtocol=https;AccountName=ai102str123;AccountKey=12345abcdefg...==;EndpointSuffix=core.windows.net

接続文字列は、Azure ポータルのストレージ アカウントの ***[アクセス キー***] ページで確認できます。

1. 更新された JSON ファイルを保存して閉じます。
2. 検索フォルダの**作成**で、**skillset.json を開きます**。このファイルには、**margies-custom-skillset という名前のスキルセットの** JSON 定義が含まれています。
3. スキルセット定義の上部にある **cognitiveServices** 要素で、**YOUR\_COGNITIVE\_SERVICES\_KEY**プレースホルダーをコグニティブ サービス リソースのキーのどちらかに置き換えます。

キーは、Azure ポータルのコグニティブ サービス リソースの ***[キーとエンドポイント***] ページで確認できます。

1. 更新された JSON ファイルを保存して閉じます。
2. 検索フォルダの**作成**で、**index.json を**開きます。このファイルには、**margies-custom-index** という名前のインデックスの JSON 定義が含まれています。
3. インデックスの JSON を確認し、変更を加えずにファイルを閉じます。
4. 検索フォルダの**作成**で、**indexer.json を開きます**。このファイルには、**margies-custom-indexer という名前のインデクサーの** JSON 定義が含まれています。
5. インデクサーの JSON を確認し、変更を加えずにファイルを閉じます。
6. 検索フォルダの**作成**で、**create-search.cmd** を開きます。このバッチ スクリプトでは、cURL ユーティリティを使用して、Azure コグニティブ検索リソースの REST インターフェイスに JSON 定義を送信します。
7. **YOUR\_SEARCH\_URL**と**YOUR\_ADMIN\_KEY**変数のプレースホルダーを**、Azure** Cognitive Search リソースの URL と**いずれかの管理キー**に置き換えます。

これらの値は、Azure ポータルの Azure コグニティブ検索リソースの ***[概要***] ページと ***[キー***] ページで確認できます。

1. 更新されたバッチ ファイルを保存します。
2. **検索フォルダ**を右クリックし、[**統合ターミナルで開く**] を選択します。
3. 検索フォルダの**作成**のターミナル ペインで、次のコマンドを入力してバッチ スクリプトを実行します。

create-search

1. スクリプトが完了したら、Azure ポータルの Azure Cognitive Search リソースのページで[**インデクサー**] ページを選択し、インデックス作成プロセスが完了するまで待ちます。

[***更新]*** を選択すると、インデックス作成操作の進行状況を追跡できます。完了するまでに 1 分ほどかかる場合があります。

## インデックスの検索

これでインデックスが作成され、検索できます。

1. Azure コグニティブ検索リソースのブレードの上部で、[**検索エクスプローラー**] を選択します。
2. 検索エクスプローラーの [**クエリ文字列**] ボックスに次のクエリ文字列を入力し、[検索] を選択**します**。

search=London&$select=url,sentiment,keyphrases&$filter=metadata\_author eq 'Reviewer' and sentiment eq 'positive'

このクエリは、肯定的な**センチメント** ラベルを持つ Reviewer によって作成されたロンドンに言及するすべてのドキュメントの **URL**、**センチメント**、および**キーフレーズ**を取得します (つまり、ロンドンに言及する肯定的なレビュー)。

## カスタム スキル用の Azure 関数を作成する

検索ソリューションには、前のタスクで見たセンチメント スコアやキー フレーズの一覧など、ドキュメントの情報をインデックスに付加する組み込みの認知スキルが数多く含まれています。

カスタム スキルを作成することで、インデックスをさらに強化できます。たとえば、各ドキュメントで最も頻繁に使用される単語を特定すると便利ですが、この機能を提供する組み込みのスキルはありません。

単語カウント機能をカスタム スキルとして実装するには、使用する言語で Azure 関数を作成します。

1. Visual Studio コードで 、[Azure 拡張機能] タブ (**⊞**) を表示し、**Azure 関数**拡張機能がインストールされていることを確認します。この拡張機能を使用すると、Visual Studio コードから Azure 関数を作成およびデプロイできます。
2. [Azure] タブ (**Δ**) の [**Azure 関数**] ウィンドウで、使用する言語に応じて、次の設定を使用して新しいプロジェクト (📂) を作成します。

### ****C#****

* + **フォルダ**: **23 カスタム検索スキル/C-シャープ/ワードカウント**を参照
  + **言語**: C#
  + **テンプレート**: HTTP トリガー
  + **関数名**: ワードカウント
  + **名前空間**: マーギース検索
  + **権限レベル**: 機能

### ****ニシキヘビ****

* + **フォルダ**: **23 カスタム検索スキル/Python/ワードカウント**を参照
  + **言語**: Python
  + **仮想環境**: 仮想環境をスキップする
  + **テンプレート**: HTTP トリガー
  + **関数名**: ワードカウント
  + **権限レベル**: 機能

***launch.json*** を上書きするように求められたら、それを行います。

1. **[エクスプローラー**] タブ (**🗍**) に戻り、**ワードカウント** フォルダーに Azure 関数のコード ファイルが含まれていることを確認します。

Python を選択した場合、コード ファイルはサブフォルダにあり、***ワードカウント***という名前も付けられます。

1. 関数のメイン コード ファイルは自動的に開かれているはずです。選択されていない場合は、選択した言語に適したファイルを開きます。
   * **C#**: ワードカウント.cs
   * **パイソン**:\_\_init\_\_.py
2. 選択した言語の次のコードで、ファイルの内容全体を置き換えます。

### ****C#****

C#

using System.IO;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Extensions.Http;

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

using System.Collections.Generic;

using Microsoft.Extensions.Logging;

using System.Text.RegularExpressions;

using System.Linq;

namespace margies.search

{

public static class wordcount

{

//define classes for responses

private class WebApiResponseError

{

public string message { get; set; }

}

private class WebApiResponseWarning

{

public string message { get; set; }

}

private class WebApiResponseRecord

{

public string recordId { get; set; }

public Dictionary<string, object> data { get; set; }

public List<WebApiResponseError> errors { get; set; }

public List<WebApiResponseWarning> warnings { get; set; }

}

private class WebApiEnricherResponse

{

public List<WebApiResponseRecord> values { get; set; }

}

//function for custom skill

[FunctionName("wordcount")]

public static IActionResult Run(

[HttpTrigger(AuthorizationLevel.Function, "post", Route = null)]HttpRequest req, ILogger log)

{

log.LogInformation("Function initiated.");

string recordId = null;

string originalText = null;

string requestBody = new StreamReader(req.Body).ReadToEnd();

dynamic data = JsonConvert.DeserializeObject(requestBody);

// Validation

if (data?.values == null)

{

return new BadRequestObjectResult(" Could not find values array");

}

if (data?.values.HasValues == false || data?.values.First.HasValues == false)

{

return new BadRequestObjectResult("Could not find valid records in values array");

}

WebApiEnricherResponse response = new WebApiEnricherResponse();

response.values = new List<WebApiResponseRecord>();

foreach (var record in data?.values)

{

recordId = record.recordId?.Value as string;

originalText = record.data?.text?.Value as string;

if (recordId == null)

{

return new BadRequestObjectResult("recordId cannot be null");

}

// Put together response.

WebApiResponseRecord responseRecord = new WebApiResponseRecord();

responseRecord.data = new Dictionary<string, object>();

responseRecord.recordId = recordId;

responseRecord.data.Add("text", Count(originalText));

response.values.Add(responseRecord);

}

return (ActionResult)new OkObjectResult(response);

}

public static string RemoveHtmlTags(string html)

{

string htmlRemoved = Regex.Replace(html, @"<script[^>]\*>[\s\S]\*?</script>|<[^>]+>| ", " ").Trim();

string normalised = Regex.Replace(htmlRemoved, @"\s{2,}", " ");

return normalised;

}

public static List<string> Count(string text)

{

//remove html elements

text=text.ToLowerInvariant();

string html = RemoveHtmlTags(text);

//split into list of words

List<string> list = html.Split(" ").ToList();

//remove any non alphabet characters

var onlyAlphabetRegEx = new Regex(@"^[A-z]+$");

list = list.Where(f => onlyAlphabetRegEx.IsMatch(f)).ToList();

//remove stop words

string[] stopwords = { "", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you",

"you're", "you've", "you'll", "you'd", "your", "yours", "yourself",

"yourselves", "he", "him", "his", "himself", "she", "she's", "her",

"hers", "herself", "it", "it's", "its", "itself", "they", "them",

"their", "theirs", "themselves", "what", "which", "who", "whom",

"this", "that", "that'll", "these", "those", "am", "is", "are", "was",

"were", "be", "been", "being", "have", "has", "had", "having", "do",

"does", "did", "doing", "a", "an", "the", "and", "but", "if", "or",

"because", "as", "until", "while", "of", "at", "by", "for", "with",

"about", "against", "between", "into", "through", "during", "before",

"after", "above", "below", "to", "from", "up", "down", "in", "out",

"on", "off", "over", "under", "again", "further", "then", "once", "here",

"there", "when", "where", "why", "how", "all", "any", "both", "each",

"few", "more", "most", "other", "some", "such", "no", "nor", "not",

"only", "own", "same", "so", "than", "too", "very", "s", "t", "can",

"will", "just", "don", "don't", "should", "should've", "now", "d", "ll",

"m", "o", "re", "ve", "y", "ain", "aren", "aren't", "couldn", "couldn't",

"didn", "didn't", "doesn", "doesn't", "hadn", "hadn't", "hasn", "hasn't",

"haven", "haven't", "isn", "isn't", "ma", "mightn", "mightn't", "mustn",

"mustn't", "needn", "needn't", "shan", "shan't", "shouldn", "shouldn't", "wasn",

"wasn't", "weren", "weren't", "won", "won't", "wouldn", "wouldn't"};

list = list.Where(x => x.Length > 2).Where(x => !stopwords.Contains(x)).ToList();

//get distict words by key and count, and then order by count.

var keywords = list.GroupBy(x => x).OrderByDescending(x => x.Count());

var klist = keywords.ToList();

// return the top 10 words

var numofWords = 10;

if(klist.Count<10)

numofWords=klist.Count;

List<string> resList = new List<string>();

for (int i = 0; i < numofWords; i++)

{

resList.Add(klist[i].Key);

}

return resList;

}

}

}

## ****Python****

Python

import logging

import os

import sys

import json

from string import punctuation

from collections import Counter

import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:

logging.info('Wordcount function initiated.')

# The result will be a "values" bag

result = {

"values": []

}

statuscode = 200

# We're going to exclude words from this list in the word counts

stopwords = ['', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',

"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',

'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',

'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',

'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom',

'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was',

'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',

'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',

'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with',

'about', 'against', 'between', 'into', 'through', 'during', 'before',

'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',

'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',

'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',

'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',

'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',

'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',

'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",

'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",

'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',

"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',

"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

try:

values = req.get\_json().get('values')

logging.info(values)

for rec in values:

# Construct the basic JSON response for this record

val = {

"recordId": rec['recordId'],

"data": {

"text":None

},

"errors": None,

"warnings": None

}

try:

# get the text to be processed from the input record

txt = rec['data']['text']

# remove numeric digits

txt = ''.join(c for c in txt if not c.isdigit())

# remove punctuation and make lower case

txt = ''.join(c for c in txt if c not in punctuation).lower()

# remove stopwords

txt = ' '.join(w for w in txt.split() if w not in stopwords)

# Count the words and get the most common 10

wordcount = Counter(txt.split()).most\_common(10)

words = [w[0] for w in wordcount]

# Add the top 10 words to the output for this text record

val["data"]["text"] = words

except:

# An error occured for this text record, so add lists of errors and warning

val["errors"] =[{"message": "An error occurred processing the text."}]

val["warnings"] = [{"message": "One or more inputs failed to process."}]

finally:

# Add the value for this record to the response

result["values"].append(val)

except Exception as ex:

statuscode = 500

# A global error occurred, so return an error response

val = {

"recordId": None,

"data": {

"text":None

},

"errors": [{"message": ex.args}],

"warnings": [{"message": "The request failed to process."}]

}

result["values"].append(val)

finally:

# Return the response

return func.HttpResponse(body=json.dumps(result), mimetype="application/json", status\_code=statuscode)

1. 更新したファイルを保存します。
2. コード ファイルを含む**ワードカウント** フォルダーを右クリックし、[**関数アプリに展開]** を選択します。次に、次の言語固有の設定で関数をデプロイします (プロンプトが表示されたら Azure にサインインします)。

### ****C#****

* + **サブスクリプション** (要求された場合): Azure サブスクリプションを選択します。
  + **機能**: Azure で新しい関数アプリを作成する (詳細)
  + **関数アプリ名**: グローバルに一意の名前を入力します。
  + **ランタイム**: .NET コア 3.1
  + **OS**: Linux
  + **ホスティングプラン**:消費
  + **リソース グループ**: Azure コグニティブ検索リソースを含むリソース グループ。
    - 注: このリソース グループに既に Windows ベースの Web アプリが含まれている場合、そこに Linux ベースの関数を展開することはできません。既存の Web アプリを削除するか、別のリソース グループに関数をデプロイします。
  + **ストレージ アカウント**: マーギーのトラベル ドキュメントが保存されているストレージ数。
  + **アプリケーションインサイト**: 今はスキップ

Visual Studio Code は、関数プロジェクトを作成したときに保存された ***.vscode*** フォルダーの構成設定に基づいて、関数のコンパイル済みバージョンを (***bin*** サブフォルダーに) 配置します。

### ****ニシキヘビ****

* + **サブスクリプション** (要求された場合): Azure サブスクリプションを選択します。
  + **機能**: Azure で新しい関数アプリを作成する (詳細)
  + **関数アプリ名**: グローバルに一意の名前を入力します。
  + **ランタイム**: Python 3.8
  + **ホスティングプラン**:消費
  + **リソース グループ**: Azure コグニティブ検索リソースを含むリソース グループ。
    - 注: このリソース グループに既に Windows ベースの Web アプリが含まれている場合、そこに Linux ベースの関数を展開することはできません。既存の Web アプリを削除するか、別のリソース グループに関数をデプロイします。
  + **ストレージ アカウント**: マーギーのトラベル ドキュメントが保存されているストレージ数。
  + **アプリケーションインサイト**: 今はスキップ

1. 関数を配置する Visual Studio コードを待機します。展開が完了すると、通知が表示されます。

## 関数をテストする

関数を Azure にデプロイしたので、Azure ポータルでテストできます。

1. [Azure ポータル](https://portal.azure.com/)を開き、関数アプリを作成したリソース グループを参照します。次に、関数アプリのアプリ サービスを開きます。
2. アプリ サービスのブレードの [**関数**] ページで、**ワードカウント**関数を開きます。
3. [**ワードカウント**関数] ブレードで、[**コード + テスト**] ページを表示し、[**テスト/実行**] ウィンドウを開きます。
4. [**テスト/実行**] ウィンドウで、既存の**本文**を次の JSON に置き換えます。
5. {
6. "values": [
7. {
8. "recordId": "a1",
9. "data":
10. {
11. "text": "Tiger, tiger burning bright in the darkness of the night.",
12. "language": "en"
13. }
14. },
15. {
16. "recordId": "a2",
17. "data":
18. {
19. "text": "The rain in spain stays mainly in the plains! That's where you'll find the rain!",
20. "language": "en"
21. }
22. }
23. ]

}

1. [**実行**] をクリックして、関数によって返される HTTP 応答の内容を表示します。これは、スキルを消費する際に Azure Cognitive Search が必要とするスキーマを反映しており、各ドキュメントに対する応答が返されます。この場合、応答は各ドキュメント内の最大 10 の用語から構成され、表示される頻度の降順になります。
2. {
3. "values": [
4. {
5. "recordId": "a1",
6. "data": {
7. "text": [
8. "tiger",
9. "burning",
10. "bright",
11. "darkness",
12. "night"
13. ]
14. },
15. "errors": null,
16. "warnings": null
17. },
18. {
19. "recordId": "a2",
20. "data": {
21. "text": [
22. "rain",
23. "spain",
24. "stays",
25. "mainly",
26. "plains",
27. "thats",
28. "youll",
29. "find"
30. ]
31. },
32. "errors": null,
33. "warnings": null
34. }
35. ]

}

1. **[テスト/実行**] ウィンドウを閉じ、**ワードカウント**関数ブレードで [**関数 URL の取得**] をクリックします。次に、既定のキーの URL をクリップボードにコピーします。これは次の手順で必要になります。

## 検索ソリューションにカスタム スキルを追加する

次に、検索ソリューションのスキルセットにカスタム スキルとして関数を含め、結果をインデックスのフィールドにマップする必要があります。

1. Visual Studio コードの **23 個のカスタム検索スキル/更新検索**フォルダーで、**更新スキルセット.json** ファイルを開きます。これには、スキルセットの JSON 定義が含まれます。
2. スキルセット定義を確認します。以前と同じスキルと、**get-top-words** という名前の新しい **WebApiSkill** スキルが含まれています。
3. **get-top-words** スキル定義を編集して、**URI** 値を Azure 関数の URL に設定します (前の手順でクリップボードにコピーした**) を使用して、関数-APP-URL を**置き換えます。
4. スキルセット定義の上部にある **cognitiveServices** 要素で、**YOUR\_COGNITIVE\_SERVICES\_KEY**プレースホルダーをコグニティブ サービス リソースのキーのどちらかに置き換えます。

キーは、Azure ポータルのコグニティブ サービス リソースの ***[キーとエンドポイント***] ページで確認できます。

1. 更新された JSON ファイルを保存して閉じます。
2. **更新検索**フォルダーで、**更新インデックス.json を**開きます。このファイルには**、マーギーのカスタムインデックス**インデックスの JSON 定義が **top\_words**含まれています。
3. インデックスの JSON を確認し、変更を加えずにファイルを閉じます。
4. **更新検索**フォルダーで、**更新インデクサー.json を開きます**。このファイルには**、margies カスタム インデクサー**の JSON 定義と、**top\_words** フィールドの追加マッピングが含まれています。
5. インデクサーの JSON を確認し、変更を加えずにファイルを閉じます。
6. **更新検索**フォルダで、**更新-検索.cmd を**開きます。このバッチ スクリプトでは、cURL ユーティリティを使用して、更新された JSON 定義を Azure コグニティブ検索リソースの REST インターフェイスに送信します。
7. **YOUR\_SEARCH\_URL**と**YOUR\_ADMIN\_KEY**変数のプレースホルダーを**、Azure** Cognitive Search リソースの URL と**いずれかの管理キー**に置き換えます。

これらの値は、Azure ポータルの Azure コグニティブ検索リソースの ***[概要***] ページと ***[キー***] ページで確認できます。

1. 更新されたバッチ ファイルを保存します。
2. **更新検索**フォルダを右クリックし、[**統合ターミナルで開く**] を選択します。
3. **更新検索**フォルダーのターミナル ウィンドウで、次のコマンドを入力バッチ スクリプトを実行します。

update-search

1. スクリプトが完了したら、Azure ポータルの Azure Cognitive Search リソースのページで[**インデクサー**] ページを選択し、インデックス作成プロセスが完了するまで待ちます。

[***更新]*** を選択すると、インデックス作成操作の進行状況を追跡できます。完了するまでに 1 分ほどかかる場合があります。

## インデックスの検索

これでインデックスが作成され、検索できます。

1. Azure コグニティブ検索リソースのブレードの上部で、[**検索エクスプローラー**] を選択します。
2. 検索エクスプローラーの [**クエリ文字列**] ボックスに次のクエリ文字列を入力し、[検索] を選択**します**。

search=Las Vegas&$select=url,top\_words

このクエリは、ラスベガスに言及するすべてのドキュメントの **URL** と**top\_words**フィールドを取得します。

## 詳細

Azure コグニティブ 検索のカスタム スキルの作成の詳細については、[Azure コグニティブ 検索のドキュメント](https://docs.microsoft.com/azure/search/cognitive-search-custom-skill-interface" \t "_blank)を参照してください。

## マイクロソフトに戻る 学び

演習が完了したら、[マイクロソフトのラーニングに戻り](https://docs.microsoft.com/learn/modules/create-enrichment-pipeline-azure-cognitive-search/12-knowledge-check" \t "_blank)、この章を完了するための知識チェックを完了し、ポイントを獲得します。

AI-102-23: AI-102 Lab 23 (Search Custom Skill)

1 Hr 58 Min Remaining

Instructions Resources Help  100%

# Before You Start

**Tip**: As you follow the instructions in this pane, whenever you see a icon, you can use it to copy text from the instruction pane into the virtual machine interface. This is particularly useful to copy code; but bear in mind you may need to modify the pasted code to fix indent levels or formatting before running it!

1. If prompted, log into the **Student** account with the password Pa55w.rd. If prompted to allow your PC to be discoverable, select **No**.
2. If you do not already have an Azure subscription, sign up for a free trial at <https://azure.microsoft.com/free/>.

# Create a Custom Skill for Azure Cognitive Search

Azure Cognitive Search uses an enrichment pipeline of cognitive skills to extract AI-generated fields from documents and include them in a search index. There's a comprehensive set of built-in skills that you can use, but if you have a specific requirement that isn't met by these skills, you can create a custom skill.

In this exercise, you'll create a custom skill that tabulates the frequency of individual words in a document to generate a list of the top five most used words, and add it to a search solution for Margie's Travel - a fictitious travel agency.

## Clone the repository for this course

If you have already cloned **AI-102-AIEngineer** code repository to the environment where you're working on this lab, open it in Visual Studio Code; otherwise, follow these steps to clone it now.

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the repository to a local folder (it doesn't matter which folder).https://github.com/MicrosoftLearning/AI-102-AIEngineer
3. When the repository has been cloned, open the folder in Visual Studio Code.
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**.

## Create Azure resources

**Note**: If you have previously completed the [**Create an Azure Cognitive Search solution**](https://raw.githubusercontent.com/MicrosoftLearning/AI-102-AIEngineer/master/Instructions/22-azure-search.md) exercise, and still have these Azure resources in your subscription, you can skip this section and start at the **Create a search solution** section. Otherwise, follow the steps below to provision the required Azure resources.

1. In a web browser, open the Azure portal at , and sign in using the Microsoft account associated with your Azure subscription.https://portal.azure.com
2. View the **Resource groups** in your subscription.
3. If you are using a restricted subscription in which a resource group has been provided for you, select the resource group to view its properties. Otherwise, create a new resource group with a name of your choice, and go to it when it has been created.
4. On the **Overview** page for your resource group, note the **Subscription ID** and **Location**. You will need these values, along with the name of the resource group in subsequent steps.
5. In Visual Studio Code, expand the **23-custom-search-skill** folder and select **setup.cmd**. You will use this batch script to run the Azure command line interface (CLI) commands required to create the Azure resources you need.
6. Right-click the the **23-custom-search-skill** folder and select **Open in Integrated Terminal**.
7. In the terminal pane, enter the following command to establish an authenticated connection to your Azure subscription.

az login --output none

1. When prompted, sign into your Azure subscription. Then return to Visual Studio Code and wait for the sign-in process to complete.
2. Run the following command to list Azure locations.

az account list-locations -o table

1. In the output, find the **Name** value that corresponds with the location of your resource group (for example, for East US the corresponding name is eastus).
2. In the **setup.cmd** script, modify the **subscription\_id**, **resource\_group**, and **location** variable declarations with the appropriate values for your subscription ID, resource group name, and location name. Then save your changes.
3. In the terminal for the **23-custom-search-skill** folder, enter the following command to run the script:

setup

**Note**: The Search CLI module is in preview, and may get stuck in the - Running .. process. If this happens for over 2 minutes, press CTRL+C to cancel the long-running operation, and then select **N** when asked if you want to terminate the script. It should then complete successfully.

If the script fails, ensure you saved it with the correct variable names and try again.

1. When the script completes, review the output it displays and note the following information about your Azure resources (you will need these values later):
   * Storage account name
   * Storage connection string
   * Cognitive Services account
   * Cognitive Services key
   * Search service endpoint
   * Search service admin key
   * Search service query key
2. In the Azure portal, refresh the resource group and verify that it contains the Azure Storage account, Azure Cognitive Services resource, and Azure Cognitive Search resource.

## Create a search solution

Now that you have the necessary Azure resources, you can create a search solution that consists of the following components:

* A **data source** that references the documents in your Azure storage container.
* A **skillset** that defines an enrichment pipeline of skills to extract AI-generated fields from the documents.
* An **index** that defines a searchable set of document records.
* An **indexer** that extracts the documents from the data source, applies the skillset, and populates the index.

In this exercise, you'll use the Azure Cognitive Search REST interface to create these components by submitting JSON requests.

1. In Visual Studio Code, in the **23-custom-search-skill** folder, expand the **create-search** folder and select **data\_source.json**. This file contains a JSON definition for a data source named **margies-custom-data**.
2. Replace the **YOUR\_CONNECTION\_STRING** placeholder with the connection string for your Azure storage account, which should resemble the following:

DefaultEndpointsProtocol=https;AccountName=ai102str123;AccountKey=12345abcdefg...==;EndpointSuffix=core.windows.net

You can find the connection string on the ***Access keys*** page for your storage account in the Azure portal.

1. Save and close the updated JSON file.
2. In the **create-search** folder, open **skillset.json**. This file contains a JSON definition for a skillset named **margies-custom-skillset**.
3. At the top of the skillset definition, in the **cognitiveServices** element, replace the **YOUR\_COGNITIVE\_SERVICES\_KEY** placeholder with either of the keys for your cognitive services resources.

You can find the keys on the ***Keys and Endpoint*** page for your cognitive services resource in the Azure portal.

1. Save and close the updated JSON file.
2. In the **create-search** folder, open **index.json**. This file contains a JSON definition for an index named **margies-custom-index**.
3. Review the JSON for the index, then close the file without making any changes.
4. In the **create-search** folder, open **indexer.json**. This file contains a JSON definition for an indexer named **margies-custom-indexer**.
5. Review the JSON for the indexer, then close the file without making any changes.
6. In the **create-search** folder, open **create-search.cmd**. This batch script uses the cURL utility to submit the JSON definitions to the REST interface for your Azure Cognitive Search resource.
7. Replace the **YOUR\_SEARCH\_URL** and **YOUR\_ADMIN\_KEY** variable placeholders with the **Url** and one of the **admin keys** for your Azure Cognitive Search resource.

You can find these values on the ***Overview*** and ***Keys*** pages for your Azure Cognitive Search resource in the Azure portal.

1. Save the updated batch file.
2. Right-click the the **create-search** folder and select **Open in Integrated Terminal**.
3. In the terminal pane for the **create-search** folder, enter the following command run the batch script.

create-search

1. When the script completes, in the Azure portal, on the page for your Azure Cognitive Search resource, select the **Indexers** page and wait for the indexing process to complete.

You can select ***Refresh*** to track the progress of the indexing operation. It may take a minute or so to complete.

## Search the index

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

1. At the top of the blade for your Azure Cognitive Search resource, select **Search explorer**.
2. In Search explorer, in the **Query string** box, enter the following query string, and then select **Search**.

search=London&$select=url,sentiment,keyphrases&$filter=metadata\_author eq 'Reviewer' and sentiment eq 'positive'

This query retrieves the **url**, **sentiment**, and **keyphrases** for all documents that mention London authored by Reviewer that have a positive **sentiment** label (in other words, positive reviews that mention London)

## Create an Azure Function for a custom skill

The search solution includes a number of built-in cognitive skills that enrich the index with information from the documents, such as the sentiment scores and lists of key phrases seen in the previous task.

You can enhance the index further by creating custom skills. For example, it might be useful to identify the words that are used most frequently in each document, but no built-in skill offers this functionality.

To implement the word count functionality as a custom skill, you'll create an Azure Function in your preferred language.

1. In Visual Studio Code, view the Azure Extensions tab (**⊞**), and verify that the **Azure Functions** extension is installed. This extension enables you to create and deploy Azure Functions from Visual Studio Code.
2. On Azure tab (**Δ**), in the **Azure Functions** pane, create a new project (📂) with the following settings, depending on your preferred language:

### ****C#****

* + **Folder**: Browse to **23-custom-search-skill/C-Sharp/wordcount**
  + **Language**: C#
  + **Template**: HTTP trigger
  + **Function name**: wordcount
  + **Namespace**: margies.search
  + **Authorization level**: Function

### ****Python****

* + **Folder**: Browse to **23-custom-search-skill/Python/wordcount**
  + **Language**: Python
  + **Virtual environment**: Skip virtual environment
  + **Template**: HTTP trigger
  + **Function name**: wordcount
  + **Authorization level**: Function

If you are prompted to overwrite ***launch.json***, do so!

1. Switch back to the **Explorer** (**🗍**) tab and verify that the **wordcount** folder now contains the code files for your Azure Function.

If you chose Python, the code files may be in a subfolder, also named ***wordcount***

1. The main code file for your function should have been opened automatically. If not, open the appropriate file for your chosen language:
   * **C#**: wordcount.cs
   * **Python**: \_\_init\_\_.py
2. Replace the entire contents of the file with the following code for your chosen language:

### ****C#****

C#

using System.IO;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Azure.WebJobs;

using Microsoft.Azure.WebJobs.Extensions.Http;

using Microsoft.AspNetCore.Http;

using Newtonsoft.Json;

using System.Collections.Generic;

using Microsoft.Extensions.Logging;

using System.Text.RegularExpressions;

using System.Linq;

namespace margies.search

{

public static class wordcount

{

//define classes for responses

private class WebApiResponseError

{

public string message { get; set; }

}

private class WebApiResponseWarning

{

public string message { get; set; }

}

private class WebApiResponseRecord

{

public string recordId { get; set; }

public Dictionary<string, object> data { get; set; }

public List<WebApiResponseError> errors { get; set; }

public List<WebApiResponseWarning> warnings { get; set; }

}

private class WebApiEnricherResponse

{

public List<WebApiResponseRecord> values { get; set; }

}

//function for custom skill

[FunctionName("wordcount")]

public static IActionResult Run(

[HttpTrigger(AuthorizationLevel.Function, "post", Route = null)]HttpRequest req, ILogger log)

{

log.LogInformation("Function initiated.");

string recordId = null;

string originalText = null;

string requestBody = new StreamReader(req.Body).ReadToEnd();

dynamic data = JsonConvert.DeserializeObject(requestBody);

// Validation

if (data?.values == null)

{

return new BadRequestObjectResult(" Could not find values array");

}

if (data?.values.HasValues == false || data?.values.First.HasValues == false)

{

return new BadRequestObjectResult("Could not find valid records in values array");

}

WebApiEnricherResponse response = new WebApiEnricherResponse();

response.values = new List<WebApiResponseRecord>();

foreach (var record in data?.values)

{

recordId = record.recordId?.Value as string;

originalText = record.data?.text?.Value as string;

if (recordId == null)

{

return new BadRequestObjectResult("recordId cannot be null");

}

// Put together response.

WebApiResponseRecord responseRecord = new WebApiResponseRecord();

responseRecord.data = new Dictionary<string, object>();

responseRecord.recordId = recordId;

responseRecord.data.Add("text", Count(originalText));

response.values.Add(responseRecord);

}

return (ActionResult)new OkObjectResult(response);

}

public static string RemoveHtmlTags(string html)

{

string htmlRemoved = Regex.Replace(html, @"<script[^>]\*>[\s\S]\*?</script>|<[^>]+>| ", " ").Trim();

string normalised = Regex.Replace(htmlRemoved, @"\s{2,}", " ");

return normalised;

}

public static List<string> Count(string text)

{

//remove html elements

text=text.ToLowerInvariant();

string html = RemoveHtmlTags(text);

//split into list of words

List<string> list = html.Split(" ").ToList();

//remove any non alphabet characters

var onlyAlphabetRegEx = new Regex(@"^[A-z]+$");

list = list.Where(f => onlyAlphabetRegEx.IsMatch(f)).ToList();

//remove stop words

string[] stopwords = { "", "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you",

"you're", "you've", "you'll", "you'd", "your", "yours", "yourself",

"yourselves", "he", "him", "his", "himself", "she", "she's", "her",

"hers", "herself", "it", "it's", "its", "itself", "they", "them",

"their", "theirs", "themselves", "what", "which", "who", "whom",

"this", "that", "that'll", "these", "those", "am", "is", "are", "was",

"were", "be", "been", "being", "have", "has", "had", "having", "do",

"does", "did", "doing", "a", "an", "the", "and", "but", "if", "or",

"because", "as", "until", "while", "of", "at", "by", "for", "with",

"about", "against", "between", "into", "through", "during", "before",

"after", "above", "below", "to", "from", "up", "down", "in", "out",

"on", "off", "over", "under", "again", "further", "then", "once", "here",

"there", "when", "where", "why", "how", "all", "any", "both", "each",

"few", "more", "most", "other", "some", "such", "no", "nor", "not",

"only", "own", "same", "so", "than", "too", "very", "s", "t", "can",

"will", "just", "don", "don't", "should", "should've", "now", "d", "ll",

"m", "o", "re", "ve", "y", "ain", "aren", "aren't", "couldn", "couldn't",

"didn", "didn't", "doesn", "doesn't", "hadn", "hadn't", "hasn", "hasn't",

"haven", "haven't", "isn", "isn't", "ma", "mightn", "mightn't", "mustn",

"mustn't", "needn", "needn't", "shan", "shan't", "shouldn", "shouldn't", "wasn",

"wasn't", "weren", "weren't", "won", "won't", "wouldn", "wouldn't"};

list = list.Where(x => x.Length > 2).Where(x => !stopwords.Contains(x)).ToList();

//get distict words by key and count, and then order by count.

var keywords = list.GroupBy(x => x).OrderByDescending(x => x.Count());

var klist = keywords.ToList();

// return the top 10 words

var numofWords = 10;

if(klist.Count<10)

numofWords=klist.Count;

List<string> resList = new List<string>();

for (int i = 0; i < numofWords; i++)

{

resList.Add(klist[i].Key);

}

return resList;

}

}

}

## ****Python****

Python

import logging

import os

import sys

import json

from string import punctuation

from collections import Counter

import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:

logging.info('Wordcount function initiated.')

# The result will be a "values" bag

result = {

"values": []

}

statuscode = 200

# We're going to exclude words from this list in the word counts

stopwords = ['', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',

"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',

'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',

'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',

'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom',

'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was',

'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',

'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',

'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with',

'about', 'against', 'between', 'into', 'through', 'during', 'before',

'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',

'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',

'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',

'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',

'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',

'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',

'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",

'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",

'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',

"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',

"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

try:

values = req.get\_json().get('values')

logging.info(values)

for rec in values:

# Construct the basic JSON response for this record

val = {

"recordId": rec['recordId'],

"data": {

"text":None

},

"errors": None,

"warnings": None

}

try:

# get the text to be processed from the input record

txt = rec['data']['text']

# remove numeric digits

txt = ''.join(c for c in txt if not c.isdigit())

# remove punctuation and make lower case

txt = ''.join(c for c in txt if c not in punctuation).lower()

# remove stopwords

txt = ' '.join(w for w in txt.split() if w not in stopwords)

# Count the words and get the most common 10

wordcount = Counter(txt.split()).most\_common(10)

words = [w[0] for w in wordcount]

# Add the top 10 words to the output for this text record

val["data"]["text"] = words

except:

# An error occured for this text record, so add lists of errors and warning

val["errors"] =[{"message": "An error occurred processing the text."}]

val["warnings"] = [{"message": "One or more inputs failed to process."}]

finally:

# Add the value for this record to the response

result["values"].append(val)

except Exception as ex:

statuscode = 500

# A global error occurred, so return an error response

val = {

"recordId": None,

"data": {

"text":None

},

"errors": [{"message": ex.args}],

"warnings": [{"message": "The request failed to process."}]

}

result["values"].append(val)

finally:

# Return the response

return func.HttpResponse(body=json.dumps(result), mimetype="application/json", status\_code=statuscode)

1. Save the updated file.
2. Right-click the **wordcount** folder containing your code files and select **Deploy to Function App**. Then deploy the function with the following language-specific settings (signing into Azure if prompted):

### ****C#****

* + **Subscription** (if prompted): Select your Azure subscription.
  + **Function**: Create a new Function App in Azure (Advanced)
  + **Function App Name**: Enter a globally unique name.
  + **Runtime**: .NET Core 3.1
  + **OS**: Linux
  + **Hosting plan**: Consumption
  + **Resource group**: The resource group containing your Azure Cognitive Search resource.
    - Note: If this resource group already contains a Windows-based web app, you will not be able to deploy a Linux-based function there. Either delete the existing web app or deploy the function to a different resource group.
  + **Storage account**: The storage count where the Margie's Travel docs are stored.
  + **Application Insights**: Skip for now

Visual Studio Code will deploy the compiled version of the function (in the ***bin*** subfolder) based on the configuration settings in the ***.vscode*** folder that were saved when you created the function project.

### ****Python****

* + **Subscription** (if prompted): Select your Azure subscription.
  + **Function**: Create a new Function App in Azure (Advanced)
  + **Function App Name**: Enter a globally unique name.
  + **Runtime**: Python 3.8
  + **Hosting plan**: Consumption
  + **Resource group**: The resource group containing your Azure Cognitive Search resource.
    - Note: If this resource group already contains a Windows-based web app, you will not be able to deploy a Linux-based function there. Either delete the existing web app or deploy the function to a different resource group.
  + **Storage account**: The storage count where the Margie's Travel docs are stored.
  + **Application Insights**: Skip for now

1. Wait for Visual Studio Code to deploy the function. A notification will appear when deployment is complete.

## Test the function

Now that you've deployed the function to Azure, you can test it in the Azure portal.

1. Open the [Azure portal](https://portal.azure.com/), and browse to the resource group where you created the function app. Then open the app service for your function app.
2. In the blade for your app service, on the **Functions** page, open the **wordcount** function.
3. On the **wordcount** function blade, view the **Code + Test** page and open the **Test/Run** pane.
4. In the **Test/Run** pane, replace the existing **Body** with the following JSON, which reflects the schema expected by an Azure Cognitive Search skill in which records containing data for one or more documents are submitted for processing:
5. {
6. "values": [
7. {
8. "recordId": "a1",
9. "data":
10. {
11. "text": "Tiger, tiger burning bright in the darkness of the night.",
12. "language": "en"
13. }
14. },
15. {
16. "recordId": "a2",
17. "data":
18. {
19. "text": "The rain in spain stays mainly in the plains! That's where you'll find the rain!",
20. "language": "en"
21. }
22. }
23. ]

}

1. Click **Run** and view the HTTP response content that is returned by your function. This reflects the schema expected by Azure Cognitive Search when consuming a skill, in which a response for each document is returned. In this case, the response consists of up to 10 terms in each document in descending order of how frequently they appear:
2. {
3. "values": [
4. {
5. "recordId": "a1",
6. "data": {
7. "text": [
8. "tiger",
9. "burning",
10. "bright",
11. "darkness",
12. "night"
13. ]
14. },
15. "errors": null,
16. "warnings": null
17. },
18. {
19. "recordId": "a2",
20. "data": {
21. "text": [
22. "rain",
23. "spain",
24. "stays",
25. "mainly",
26. "plains",
27. "thats",
28. "youll",
29. "find"
30. ]
31. },
32. "errors": null,
33. "warnings": null
34. }
35. ]

}

1. Close the **Test/Run** pane and in the **wordcount** function blade, click **Get function URL**. Then copy the URL for the default key to the clipboard. You'll need this in the next procedure.

## Add the custom skill to the search solution

Now you need to include your function as a custom skill in the search solution skillset, and map the results it produces to a field in the index.

1. In Visual Studio Code, in the **23-custom-search-skill/update-search** folder, open the **update-skillset.json** file. This contains the JSON definition of a skillset.
2. Review the skillset definition. It includes the same skills as before, as well as a new **WebApiSkill** skill named **get-top-words**.
3. Edit the **get-top-words** skill definition to set the **uri** value to the URL for your Azure function (which you copied to the clipboard in the previous procedure), replacing **YOUR-FUNCTION-APP-URL**.
4. At the top of the skillset definition, in the **cognitiveServices** element, replace the **YOUR\_COGNITIVE\_SERVICES\_KEY** placeholder with either of the keys for your cognitive services resources.

You can find the keys on the ***Keys and Endpoint*** page for your cognitive services resource in the Azure portal.

1. Save and close the updated JSON file.
2. In the **update-search** folder, open **update-index.json**. This file contains the JSON definition for the **margies-custom-index** index, with an additional field named **top\_words** at the bottom of the index definition.
3. Review the JSON for the index, then close the file without making any changes.
4. In the **update-search** folder, open **update-indexer.json**. This file contains a JSON definition for the **margies-custom-indexer**, with an additional mapping for the **top\_words** field.
5. Review the JSON for the indexer, then close the file without making any changes.
6. In the **update-search** folder, open **update-search.cmd**. This batch script uses the cURL utility to submit the updated JSON definitions to the REST interface for your Azure Cognitive Search resource.
7. Replace the **YOUR\_SEARCH\_URL** and **YOUR\_ADMIN\_KEY** variable placeholders with the **Url** and one of the **admin keys** for your Azure Cognitive Search resource.

You can find these values on the ***Overview*** and ***Keys*** pages for your Azure Cognitive Search resource in the Azure portal.

1. Save the updated batch file.
2. Right-click the the **update-search** folder and select **Open in Integrated Terminal**.
3. In the terminal pane for the **update-search** folder, enter the following command run the batch script.

update-search

1. When the script completes, in the Azure portal, on the page for your Azure Cognitive Search resource, select the **Indexers** page and wait for the indexing process to complete.

You can select ***Refresh*** to track the progress of the indexing operation. It may take a minute or so to complete.

## Search the index

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

1. At the top of the blade for your Azure Cognitive Search resource, select **Search explorer**.
2. In Search explorer, in the **Query string** box, enter the following query string, and then select **Search**.

search=Las Vegas&$select=url,top\_words

This query retrieves the **url** and **top\_words** fields for all documents that mention Las Vegas.

## More information

To learn more about creating custom skills for Azure Cognitive Search, see the [Azure Cognitive Search documentation](https://docs.microsoft.com/azure/search/cognitive-search-custom-skill-interface).

## Return to Microsoft Learn

Now that you have completed the exercise, [return to Microsoft Learn](https://docs.microsoft.com/learn/modules/create-enrichment-pipeline-azure-cognitive-search/12-knowledge-check) to complete the knowledge check and earn points for completing this module.