AI-102-22: AI-102 ラボ 22 (認知検索)

残り30分

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/)。

Azure コグニティブ検索ソリューションを作成する

すべての組織は、情報を利用して意思決定を行い、質問に答え、効率的に機能します。ほとんどの組織にとっての問題は、情報の不足ではなく、情報が格納されている膨大なドキュメント、データベース、およびその他のソースから情報を検索して抽出するという課題です。

たとえば、*マーギーの旅行*は、世界中の都市への旅行を整理する専門の旅行代理店であるとします。時間が経つにつれて、同社はパンフレットなどの文書に膨大な量の情報を蓄積し、顧客から提出されたホテルのレビューを蓄積してきました。このデータは、旅行を計画する旅行代理店や顧客にとって貴重な洞察の源ですが、膨大な量のデータが特定の顧客の質問に答えるために関連する情報を見つけるのが難しくなる可能性があります。

この課題に対処するために、Margie の Travel は Azure Cognitive Search を使用して、AI ベースの認知スキルを使用してドキュメントのインデックスを作成し、情報を得て、ドキュメントを検索しやすくするソリューションを実装できます。

このコースのリポジトリのクローンを作成する

このラボで作業している環境に **AI-102-AIEngineer** コードリポジトリをまだクローンしていない場合は、次の手順に従ってください。それ以外の場合は、Visual Studio コードで複製されたフォルダーを開きます。

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

マーギーの旅行のために作成するソリューションには、Azure サブスクリプションに次のリソースが必要です。

* インデックス作成とクエリを管理する **Azure コグニティブ 検索**リソース。
* **認知サービス** リソースは、検索ソリューションが AI 生成されたインサイトを使用してデータ ソース内のデータを強化するために使用できるスキルに対する AI サービスを提供します。
* 検索対象のドキュメントが格納される BLOB コンテナーを持つ**ストレージ アカウント**。

**重要**: Azure の認知検索とコグニティブ サービスのリソースは、同じ場所になければなりません。

Azure コグニティブ検索リソースを作成する

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. **[+リソースの作成**] ボタンを選択し、*検索*を検索して、次の設定で **Azure Cognitive Search** リソースを作成します。
   * **サブスクリプション**: *Azure サブスクリプション*
   * **リソース グループ**: *新しいリソース グループを作成します (制限されたサブスクリプションを使用している場合は、新しいリソース グループを作成するアクセス許可がない可能性があります 。*
   * **サービス名**: *一意の名前を入力してください*
   * **場所**: *場所を選択する - Azure コグニティブ検索とコグニティブ サービス のリソースが同じ場所にある必要があることに注意してください。*
   * **価格レベル**: 基本
3. デプロイが完了するのを待ってから、デプロイされたリソースに移動します。
4. Azure ポータルで、Azure コグニティブ 検索リソースのブレードの **[概要**] ページを確認します。ここでは、ビジュアルインターフェイスを使用して、検索ソリューションのさまざまなコンポーネントを作成、テスト、管理、および監視できます。データ ソース、インデックス、インデクサー、スキルセットを含みます。

コグニティブ サービス リソースの作成

サブスクリプションにまだサブスクリプションがない場合は、**Cognitive サービス** リソースをプロビジョニングする必要があります。検索ソリューションでは、これを使用して、AI 生成されたインサイトを使用してデータストア内のデータを強化します。

1. Azure ポータルのホーム ページに戻り、[**+ リソースの作成**] ボタンを選択し、*コグニティブ サービス*を検索して、次の設定で**コグニティブ サービス** リソースを作成します。
   * **サブスクリプション**: *Azure サブスクリプション*
   * **リソース グループ**: *Azure コグニティブ検索リソースと同じリソース グループ*
   * **リージョン**: *Azure コグニティブ検索リソースと同じ場所*
   * **名前**: *一意の名前を入力してください*
   * **価格レベル**: 標準 S0
2. 必要なチェックボックスを選択し、リソースを作成します。
3. 展開が完了するまで待ってから、展開の詳細を表示します。

ストレージ アカウントを作成する

1. Azure Portal のホーム ページに戻り、[**+リソースの作成**] ボタンを選択し、*ストレージ アカウント*を検索して、次の設定で**ストレージ アカウント** リソースを作成します。
   * **サブスクリプション**: *Azure サブスクリプション*
   * **リソース グループ**: *\*Azure コグニティブ検索およびコグニティブ サービス リソースと同じリソース グループ*
   * **ストレージ アカウント名**: *一意の名前を入力します。*
   * **リージョン**: *使用可能なリージョンを選択します*。
   * **パフォーマンス**: 標準
   * **レプリケーション**: ローカル冗長ストレージ (LRS)
2. デプロイが完了するのを待ってから、デプロイされたリソースに移動します。
3. [**概要**] ページで、**サブスクリプション ID** -this は、ストレージ アカウントがプロビジョニングされているサブスクリプションを識別します。
4. [**アクセス キー**] ページで、ストレージ アカウントに対して 2 つのキーが生成されていることを確認します。次に、[**キーを表示**] を選択してキーを表示します。

**ヒント**: **[ストレージ アカウント**] ブレードを開いたままにしておきます。

Azure ストレージへのドキュメントのアップロード

必要なリソースができたので、Azure ストレージ アカウントにいくつかのドキュメントをアップロードできます。

1. Visual Studio コードの**エクスプローラー** ペインで、**22 個の検索ソリューションフォルダーを**展開し、**UploadDocs.cmd** を選択します。
2. バッチ ファイルを編集して、**YOUR\_SUBSCRIPTION\_ID**、**YOUR\_AZURE\_STORAGE\_ACCOUNT\_NAME**、**およびYOUR\_AZURE\_STORAGE\_KEY**のプレースホルダーを、以前に作成したストレージ アカウントの適切なサブスクリプション ID、Azure ストレージ アカウント名、および Azure ストレージ アカウント キー値に置き換えます。
3. 変更を保存し、**22-create-a-search-solution フォルダーを**右クリックして統合ターミナルを開きます。
4. 次のコマンドを入力して、Azure CLI を使用して Azure サブスクリプションにサインインします。

az login

Web ブラウザー タブが開き、Azure にサインインするように求められます。これを行い、ブラウザー タブを閉じて、Visual Studio コードに戻ります。

1. 次のコマンドを入力して、バッチ ファイルを実行します。これにより、ストレージ アカウントに BLOB コンテナーが作成され、**データ** フォルダー内のドキュメントがアップロードされます。

UploadDocs

ドキュメントのインデックスを作成する

ドキュメントを作成したら、インデックスを作成して検索ソリューションを作成できます。

1. Azure ポータルで、Azure コグニティブ検索リソースを参照します。次に、**その概要**ページで、**データのインポート**を選択します。
2. [**データへの接続**] ページの [**データ ソース**] ボックスの一覧で、[**Azure BLOB ストレージ**] を選択します。次に、データ ストアの詳細を次の値で完了します。
   * **データ ソース**: Azure BLOB ストレージ
   * **データソース名**: マーギーズデータ
   * **抽出するデータ**: コンテンツとメタデータ
   * **解析モード**: デフォルト
   * **接続文字列**: *[****既存の接続を選択します] を選択します****。次に、ストレージ アカウントを選択し、最後に UploadDocs.cmd スクリプトで作成された****margies****コンテナーを選択します。*
   * **マネージド ID 認証**: なし
   * **コンテナ名**:マーギー
   * **BLOB フォルダー**: *この空白のままにします*。
   * **説明**:マーギーの旅行ウェブサイトのパンフレットとレビュー。
3. 次のステップに進みます (*認知能力を追加*します)。
4. [**コグニティブ サービスのアタッチ]** セクションで、コグニティブ サービス リソースを選択します。
5. [**エンリッチメントの追加] セクションで、次の手順を実行**します。
   * **スキルセット名**を**マーギースキルセット**に変更します。
   * [**OCR を有効にする] オプションを選択し、すべてのテキストをフィールドに結合merged\_content**。
   * **[ソース] データ フィールド**が **merged\_content** に設定されていることを確認します。
   * **[エンリッチメントの詳細レベル**] は **[ソース] フィールド**のままにします。しかし、ページや文など、より細かいレベルで情報を抽出するためにこれを変更できることに注意してください。
   * 次のエンリッチフィールドを選択します。

| **認知スキル** | **パラメーター** | **フィールド名** |
| --- | --- | --- |
| 場所名の抽出 |  | 場所 |
| キーフレーズを抽出する |  | キーフレーズ |
| 言語を検出する |  | 言語 |
| 画像からタグを生成する |  | イメージタグ |
| 画像からキャプションを生成する |  | 画像キャプション |

1. 選択内容を再確認します (後で変更するのは難しい場合があります)。次の手順に進みます (*ターゲットインデックスのカスタマイズ*)。
2. **インデックス名**を**マーギーインデックスに変更します**。
3. **キー**が**metadata\_storage\_path**に設定されていることを確認し、**サジェスタ名**を空白のままにし、**検索モード**をデフォルトのままにします。
4. インデックス フィールドに次の変更を加え、その他のすべてのフィールドに既定の設定を残します (**重要**: テーブル全体を表示するには右にスクロールする必要がある場合があります)。

| **フィールド名** | **取得可能** | **濾過** | **ソート** | **フェイステーブル** | **検索** |
| --- | --- | --- | --- | --- | --- |
| metadata\_storage\_size | ✔ | ✔ | ✔ |  |  |
| metadata\_storage\_last\_modified | ✔ | ✔ | ✔ |  |  |
| metadata\_storage\_name | ✔ | ✔ | ✔ |  | ✔ |
| metadata\_author | ✔ | ✔ | ✔ | ✔ | ✔ |
| 場所 | ✔ | ✔ |  |  | ✔ |
| キーフレーズ | ✔ | ✔ |  |  | ✔ |
| 言語 | ✔ | ✔ |  |  |  |

1. 選択内容を再確認し、各フィールドに対して適切な**取得、フィルター、並べ替え可能**、**顔面、検索可能な**各オプションが選択されていることを確認します (後で変更するのは難しい場合があります)。次の手順に進みます (*インデクサーを作成します*)。
2. **インデクサー名**を**マーギース インデクサーに変更します**。
3. **スケジュール**を **[1 回]** に設定したままにします。
4. **[詳細]** オプションを展開し、[**Base-64 エンコード キー**] オプションが選択されていることを確認します (通常、キーをエンコードするとインデックスの効率が向上します)。
5. [**送信]** を選択して、データ ソース、スキルセット、インデックス、およびインデクサーを作成します。インデクサーは自動的に実行され、インデックスパイプラインを実行します。
   * データ ソースからドキュメント メタデータ フィールドとコンテンツを抽出します。
   * コグニティブスキルのスキルセットを実行して、追加のエンリッチフィールドを生成します。
   * 抽出されたフィールドをインデックスにマップします。
6. Azure コグニティブ検索リソースの **[概要**] ページの下半分で、[**インデクサー**] タブを表示**します。**数分待ってから、[状態] が成功↻**示**すまで [**更新]** をクリックします。

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 **[概要**] ページの上部にある **[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で、[**クエリ文字列**] ボックスに「(単一のアスタリスク)」と入力し、[検索] を選択**します**。\*

このクエリは、JSON 形式でインデックス内のすべてのドキュメントを取得します。結果を調べ、選択した認知スキルによって抽出されたドキュメントコンテンツ、メタデータ、およびエンリッチデータを含む各ドキュメントのフィールドを確認します。

1. クエリ文字列を変更し、検索を送信します。search=\*&$count=true

今回の結果には、検索によって返されたドキュメントの数を示す @odata**カウント** フィールドが結果の先頭に含まれます。

1. 次のクエリ文字列を試してください。

search=\*&$count=true&$select=metadata\_storage\_name,metadata\_author,locations

この場合、結果にはファイル名、作成者、およびドキュメントコンテンツに記載されているすべての場所のみが含まれます。ファイル名と作成者は、ソース ドキュメントから抽出された**metadata\_storage\_name**および**metadata\_author**フィールドに含まれています。**ロケーション** フィールドは、認知スキルによって生成されました。

1. Now try the following query string:

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. もう一つクエリ文字列を試してみましょう。

search="New York"&$count=true&$select=metadata\_storage\_name&$filter=metadata\_author eq 'Reviewer'

このクエリは、"ニューヨーク" と言う *Reviewer* によって作成されたドキュメントのファイル名を返します。

検索コンポーネントの定義の調査と変更

検索ソリューションのコンポーネントは、Azure ポータルで表示および編集できる JSON 定義に基づいています。

ポータルを使用して検索ソリューションを作成および変更できますが、多くの場合、検索オブジェクトを JSON で定義し、Azure Cognitive Service REST インターフェイスを使用して検索ソリューションを作成および変更することが望ましい場合があります。

Azure コグニティブ検索リソースのエンドポイントとキーを取得する

1. Azure ポータルで、Azure コグニティブ検索リソースの **[概要**] ページに戻ります。ページの上部セクションで、リソースの **URL** (**https://resource\_name.search.windows.net** のように見えます) を見つけてクリップボードにコピーします。
2. Visual Studio Code の [エクスプローラー] ウィンドウで、**22 個の検索ソリューション**の作成フォルダーとその**変更検索**サブフォルダーを展開し、**modify-search.cmd** を選択して開きます。このスクリプト ファイルを使用して、JSON を Azure コグニティブ サービス REST インターフェイスに送信する *cURL* コマンドを実行します。
3. **modify-search.cmd** で、**YOUR\_SEARCH\_URL**のプレースホルダをクリップボードにコピーした URL に置き換えます。
4. Azure ポータルで、Azure コグニティブ検索リソースの **[キー**] ページを表示し、**プライマリ管理キー**をクリップボードにコピーします。
5. Visual Studio コードで、**YOUR\_ADMIN\_KEY**のプレースホルダーをクリップボードにコピーしたキーに置き換えます。
6. 変更を保存して **search.cmd を変更**します (ただし、まだ実行しないでください)。

スキルセットの確認と変更

1. Visual studio コードで、検索フォルダーの**変更**で **skillset.json を**開きます。これは、**マーギースキルセット**のJSON定義を示しています。
2. スキルセット定義の上部で、**コグニティブ サービス** リソースをスキルセットに接続するために使用される CognitiveServices オブジェクトに注意してください。
3. Azure ポータルで、(Azure コグニティブ検索リソースではなく) コグニティブ サービス リソースを開き、**そのキー** ページを表示します。次に**、キー 1** をクリップボードにコピーします。
4. Visual Studio コードの **skillset.json** で、**YOUR\_COGNITIVE\_SERVICES\_KEY**のプレースホルダーをクリップボードにコピーしたコグニティブ サービス キーに置き換えます。
5. JSON ファイルをスクロールして、Azure ポータルの Azure Cognitive Search ユーザー インターフェイスを使用して作成したスキルの定義が含まれていることに関して説明します。スキルの一覧の下部に、次の定義を使用して追加のスキルが追加されました。
6. {
7. "@odata.type": "#Microsoft.Skills.Text.V3.SentimentSkill",
8. "defaultLanguageCode": "en",
9. "name": "get-sentiment",
10. "description": "New skill to evaluate sentiment",
11. "context": "/document",
12. "inputs": [
13. {
14. "name": "text",
15. "source": "/document/merged\_content"
16. },
17. {
18. "name": "languageCode",
19. "source": "/document/language"
20. }
21. ],
22. "outputs": [
23. {
24. "name": "sentiment",
25. "targetName": "sentimentLabel"
26. }
27. ]

}

新しいスキルには **get-sentiment** という名前が付けられ、ドキュメント内の**各ドキュメント** レベルで、インデックスが作成されているドキュメントの**merged\_content** フィールドに含まれるテキスト (ソース コンテンツ、およびコンテンツ内のイメージから抽出されたテキストを含む) が評価されます。抽出された**言語** (デフォルトは英語) を使用し、コンテンツのセンチメントのラベルを評価します。センチメント ラベルの値は、"正"、"負"、"ニュートラル"、または "混合" になります。このラベルは、**sentimentLabel** という名前の新しいフィールドとして出力されます。

1. **skillset.json** に加えた変更を保存します。

インデックスの確認と変更

1. Visual studio のコードで、検索フォルダーの**変更**で**、index.json を**開きます。これは**、マーギーインデックス**のJSON定義を示しています。
2. インデックスをスクロールして、フィールド定義を表示します。ソース ドキュメント内のメタデータとコンテンツに基づくフィールドもあれば、スキルセット内のスキルの結果であるフィールドもあります。
3. Azure ポータルで定義したフィールドの一覧の最後に、2 つのフィールドが追加されていることに注意してください。
4. {
5. "name": "sentiment",
6. "type": "Edm.String",
7. "facetable": false,
8. "filterable": true,
9. "retrievable": true,
10. "sortable": true
11. },
12. {
13. "name": "url",
14. "type": "Edm.String",
15. "facetable": false,
16. "filterable": true,
17. "retrievable": true,
18. "searchable": false,
19. "sortable": false

}

1. **センチメント**フィールドは、スキルセットを追加した**感情取得**スキルからの出力を追加するために使用されます。**url** フィールドは、データ ソースから抽出された**metadata\_storage\_path**値に基づいて、インデックス付きの各ドキュメントの URL をインデックスに追加するために使用されます。インデックスには既に**metadata\_storage\_path**フィールドが含まれていますが、インデックスキーとして使用され、Base-64はエンコードされているため、キーとして効率的になりますが、実際のURL値をフィールドとして使用する場合は、それをデコードする必要があります。エンコードされていない値に 2 番目のフィールドを追加すると、この問題は解決されます。

インデクサーの確認と変更

1. Visual studio コードで、検索フォルダーの**変更**で**、indexer.json を**開きます。これは、ドキュメントコンテンツとメタデータから抽出されたフィールド(**フィールドマッピング**セクション)、スキルセット内のスキルによって抽出された値(**outputFieldMappings**セクション内)をインデックスのフィールドにマップする**、マーギース・インデクサー**のJSON定義を示しています。
2. **フィールドマッピング**リストで、**metadata\_storage\_path**値と base-64 でエンコードされたキーフィールドへのマッピングをメモします。これは、**キーとしてmetadata\_storage\_path**を割り当て、Azure ポータルでキーをエンコードするオプションを選択したときに作成されました。さらに、新しいマッピングは、同じ値を**明示的に url** フィールドにマップしますが、Base-64 エンコードは使用しません。
3. {
4. "sourceFieldName" : "metadata\_storage\_path",
5. "targetFieldName" : "url"

}

ソース ドキュメント内の他のすべてのメタデータとコンテンツ フィールドは、インデックス内の同じ名前のフィールドに暗黙的にマップされます。

1. スキルセット内のスキルからの出力をインデックスフィールドにマップする**「出力フィールドマッピング**」セクションを確認します。これらのほとんどはユーザー インターフェイスで行った選択を反映していますが、次のマッピングが追加され、センチメント スキルによって抽出された **sentimentLabel** 値を、インデックスに追加した**センチメント** フィールドにマップしています。
2. {
3. "sourceFieldName": "/document/sentimentLabel",
4. "targetFieldName": "sentiment"

}

REST API を使用して検索ソリューションを更新する

1. **[検索の変更**] フォルダを右クリックし、統合ターミナルを開きます。
2. **変更検索**フォルダーのターミナル ペインで、次のコマンドを入力して、REST インターフェイスに JSON 定義を送信し、インデックス作成を開始する **modify-search.cmd** スクリプトを実行します。

modify-search

1. スクリプトが完了したら、Azure ポータルで Azure コグニティブ 検索リソースの **[概要**] ページに戻り、[**インデクサー**] ページを表示します。定期的に [**更新]** を選択して、インデックス作成操作の進行状況を追跡します。完了するまでに 1 分ほどかかる場合があります。

*いくつかのドキュメントに対して、センチメントを評価するには大きすぎる警告が表示されることがあります。多くの場合、センチメント分析は、完全なドキュメントではなく、ページレベルまたは文レベルで実行されます。しかし、この場合、ほとんどのドキュメント(特にホテルのレビュー)は、有用なドキュメントレベルのセンチメントスコアを評価するのに十分短いです。*

変更されたインデックスのクエリ

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)。

1. **[検索エクスプローラー**] ページを閉じて、[**概要**] ページに戻ります。

検索クライアント アプリケーションを作成する

これで、便利なインデックスが作成できたので、それをクライアント アプリケーションから使用できます。これを行うには、REST インターフェイスを使用して要求を送信し、HTTP 経由で JSON 形式で応答を受信します。または、任意のプログラミング言語にソフトウェア開発キット (SDK) を使用できます。この演習では、SDK を使用します。

**注**: **C#** または **Python** のどちらにも SDK を使用できます。以下の手順で、使用する言語に応じた操作を実行します。

検索リソースのエンドポイントとキーを取得する

1. Azure ポータルの Azure コグニティブ 検索リソースの **[概要**] ページで、[**url**] 値を **https:// *your\_resource\_name.search.windows.net*** に似ている必要があります。これは、検索リソースのエンドポイントです。
2. [**キー**] ページには、2 つの**管理**キーと 1 つの**クエリ** キーがあることに注意してください。検索リソースの作成と管理には*、管理*キーを使用します。*クエリ* キーは、検索クエリを実行するだけで済むクライアント アプリケーションで使用されます。

*クライアント アプリケーションのエンドポイントとクエリ キーが必要です。*

Azure コグニティブ 検索 SDK の使用を準備する

1. Visual Studio Code の **[エクスプローラー**] ウィンドウで、**22 個の検索ソリューション**フォルダーを参照し、言語設定に応じて **C-Sharp** フォルダーまたは **Python** フォルダーを展開します。
2. **マーギーズトラベル**フォルダを右クリックし、統合ターミナルを開きます。次に、言語設定に応じて適切なコマンドを実行して、Azure コグニティブ検索 SDK パッケージをインストールします。

**C#**

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

**ニシキヘビ**

pip install azure-search-documents==11.0.0

1. **margies-travel** フォルダの内容を表示し、構成設定用のファイルが含まれていることに注意してください。
   * **C#**: アプリ設定.json
   * **パイソン**:.env

構成ファイルを開き、Azure Cognitive Search リソースの**エンドポイント**と**クエリ キー**を反映するように構成値を更新します。変更を保存します。

インデックスを検索するコードを調べる

**margies-travel** フォルダーには、検索機能を含む Web アプリケーション (Microsoft C# *ASP.NET Razor* Web アプリケーションまたは Python *Flask* アプリケーション) のコード ファイルが含まれています。

1. プログラミング言語の選択に応じて、Web アプリケーションで次のコード ファイルを開きます。
   * **C#**:P ages/インデックス.cshtml.cs
   * **パイソン**:app.py
2. コード ファイルの上部にある "**検索名前空間のインポート"** というコメントを見つけ、Azure Cognitive Search SDK を使用するためにインポートされた名前空間をメモします。
3. **search\_query**関数で、**検索クライアントを作成**するというコメントを見つけ、コードが Azure Cognitive Search リソースのエンドポイントとクエリ キーを使用して **SearchClient** オブジェクトを作成することに注意してください。
4. **search\_query**関数で、"**送信" 検索クエリ**というコメントを見つけ、次のオプションを使用して、指定したテキストの検索を送信するコードを確認します。
   * *検索テキスト*内の個々の単語**をすべて**必要とする検索モードが見つかりました。
   * 検索で見つかったドキュメントの総数が結果に含まれます。
   * 結果は、指定されたフィルター式に一致するドキュメントのみを含むようにフィルター処理されます。
   * 結果は、指定された並べ替え順序に並べ替えられます。
   * **metadata\_author**フィールドの各不連続値は、フィルター処理用に定義済みの値を表示するために使用できる*ファセット*として返されます。
   * 検索語句が強調表示された**merged\_content**および **imageCaption** フィールドの抽出が、最大 3 つまで含まれます。
   * 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
   * **Python**: templates/search.html
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:
     + **url** フィールドのアドレスへのリンクとして**、metadata\_storage\_name** (ファイル名) フィールドを表示します。
     + **merged\_content**および **imageCaption** フィールドに含まれる検索語句の*強調表示*を表示して、検索用語をコンテキストで表示する場合に役立ちます。
     + **metadata\_author**、**metadata\_storage\_size**、**metadata\_storage\_last\_modified**、および**言語**の各フィールドを表示します。
     + ドキュメントの**センチメント** ラベルを表示します。正、負、中立、または混合をすることができます。
     + 最初の 5 つの**キーフレーズ**を表示します (存在する場合)。
     + 最初の 5 つの**場所を**表示します (存在する場合)。
     + 最初の 5 つの**イメージタグ**を表示します(存在する場合)。

Web アプリを実行する

1. **margies-travel** フォルダの統合ターミナルに戻り、次のコマンドを入力してプログラムを実行します。

**C#**

dotnet run

**ニシキヘビ**

pip install flask

pip install python-dotenv

flask run

* 1. アプリが正常に起動したときに表示されるメッセージで、実行中の Web アプリケーション (*http://localhost:5000/* または *http://127.0.0.1:5000/*) へのリンクをたどって、Web ブラウザーで Margies Travel サイトを開きます。
  2. マーギーの旅行ウェブサイトで、検索ボックスに**ロンドンのホテル**を入力し、[**検索**]をクリックします。
  3. 検索結果を確認します。ファイル名 (ファイル URL へのハイパーリンク付き)、検索語句 (*London* と *Hotel*) が強調表示されたファイル コンテンツの抽出、インデックス フィールドからのファイルのその他の属性が含まれます。
  4. 結果ページには、結果を絞り込むためのユーザー インターフェイス要素がいくつか含まれていることを確認します。これには、次のようなものがあります。
  + **metadata\_author** フィールドのファセット値に基づく*フィルター*。これは、*ファセットフィールド*を使用して、ユーザーインターフェイスで潜在的なフィルタ値として表示できる少数の不連続値のセットを持つ*ファセット*のリストを返す方法を示しています。
  + 指定したフィールドと並べ替えの方向 (昇順または降順) に基づいて結果を*並べ替*える機能。デフォルトの順序は、インデックスフィールドの検索語の頻度と重要性を評価する*スコアリングプロファイル*に基づいて**、search.score()** 値として計算される*関連性*に基づいています。
  1. **[レビューア** フィルター] と [**正対負の**並べ替え] オプションを選択し、[**結果をリファイン]** を選択します。
  2. 結果が、レビューのみを含むようにフィルター処理され、センチメント ラベルに基づいて並べ替えられることを確認します。
  3. **検索**ボックスに、**ニューヨークの静かなホテルを**新たに検索し、結果を確認します。
  4. 次の検索語句を試してください。
  + **ロンドン塔** (この用語は、いくつかの文書で*キーフレーズ*として識別されることを観察してください)。
  + **超高層ビル** (この単語は、ドキュメントの実際のコンテンツには表示されませんが、一部のドキュメントの画像に対して生成された*画像キャプション*と*画像タグ*に含まれています)。
  + **モハベ砂漠**(この用語は、いくつかの文書内の*場所*として識別されることを観察してください)。
  1. マーギーの旅行のウェブサイトを含むブラウザタブを閉じて、Visual Studioコードに戻ります。次に、**マーギーストラベル**フォルダ(ドットネットまたはフラスコアプリケーションが実行されている場所)のPythonターミナルで、Ctrl+Cと入力してアプリを停止します。

詳細

Azure コグニティブ 検索の詳細については、[Azure コグニティブ 検索のドキュメント](https://docs.microsoft.com/azure/search/search-what-is-azure-search)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-azure-cognitive-search-solution/10-knowledge-check)、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-102-22: AI-102 Lab 22 (Cognitive Search)

30 Minutes 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 an Azure Cognitive Search solution

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 Cognitive Search to implement a solution in which the documents are indexed and enriched by using AI-based cognitive skills to make them easier to search.

Clone the repository for this course

If you have not already cloned **AI-102-AIEngineer** code repository to the environment where you're working on this lab, follow these steps to do so. Otherwise, open the cloned folder in Visual Studio Code.

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

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

* An **Azure Cognitive Search** resource, which will manage indexing and querying.
* A **Cognitive 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.

**Important**: Your Azure Cognitive Search and Cognitive Services resources must be in the same location!

Create an Azure Cognitive Search resource

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. Select the **＋Create a resource** button, search for *search*, and create a **Azure Cognitive 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 Cognitive Search and Cognitive Services resources must be in the same location*
   * **Pricing tier**: Basic
3. Wait for deployment to complete, and then go to the deployed resource.
4. Review the **Overview** page on the blade for your Azure Cognitive 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.

Create a Cognitive Services resource

If you don't already have one in your subscription, you'll need to provision a **Cognitive Services** resource. Your search solution will use this to enrich the data in the datastore with AI-generated insights.

1. Return to the home page of the Azure portal, and then select the **＋Create a resource** button, search for *cognitive services*, and create a **Cognitive Services** resource with the following settings:
   * **Subscription**: *Your Azure subscription*
   * **Resource group**: *The same resource group as your Azure Cognitive Search resource*
   * **Region**: *The same location as your Azure Cognitive Search resource*
   * **Name**: *Enter a unique name*
   * **Pricing tier**: Standard S0
2. Select the required checkboxes and create the resource.
3. Wait for deployment to complete, and then view the deployment details.

Create a storage account

1. 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 Cognitive Search and Cognitive Services resources*
   * **Storage account name**: *Enter a unique name*
   * **Region**: *Choose any available region*
   * **Performance**: Standard
   * **Replication**: Locally-redundant storage (LRS)
2. Wait for deployment to complete, and then go to the deployed resource.
3. On the **Overview** page, note the **Subscription ID** -this identifies the subscription in which the storage account is provisioned.
4. 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.

Upload Documents to Azure Storage

Now that you have the required resources, you can upload some documents to your Azure Storage account.

1. In Visual Studio Code, in the **Explorer** pane, expand the **22-create-a-search-solution** folder and select **UploadDocs.cmd**.
2. Edit the batch file to replace the **YOUR\_SUBSCRIPTION\_ID**, **YOUR\_AZURE\_STORAGE\_ACCOUNT\_NAME**, and **YOUR\_AZURE\_STORAGE\_KEY** placeholders with the appropriate subscription ID, Azure storage account name, and Azure storage account key values for the storage account you created previously.
3. Save your changes, and then right-click the **22-create-a-search-solution** folder and open an integrated terminal.
4. Enter the following command to sign into your Azure subscription by using the Azure CLI.

az login

A web browser tab will open and prompt you to sign into Azure. Do so, and then close the browser tab and return to Visual Studio Code.

1. Enter the following command to run the batch file. This will create a blob container in your storage account and upload the documents in the **data** folder to it.

UploadDocs

Index the documents

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 Cognitive Search resource. Then, on its **Overview** page, select **Import data**.
2. On the **Connect to your data** page, in the **Data Source** list, select **Azure Blob Storage**. 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.
3. Proceed to the next step (*Add cognitive skills*).
4. in the **Attach Cognitive Services** section, select your cognitive services resource.
5. 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*).
2. Change the **Index name** to **margies-index**.
3. Ensure that the **Key** is set to **metadata\_storage\_path** and leave the **Suggester name** blank and **Search mode** at its default.
4. 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*).
2. Change the **Indexer name** to **margies-indexer**.
3. Leave the **Schedule** set to **Once**.
4. Expand the **Advanced** options, and ensure that the **Base-64 encode keys** option is selected (generally encoding keys make the index more efficient).
5. 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.
6. In the bottom half of the **Overview** page for your Azure Cognitive 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 Cognitive Search resource, select **Search explorer**.
2. 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. Modify the query string to and submit the search.search=\*&$count=true

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 string:

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:

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 string:

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".

Explore and modify definitions of search components

The components of the search solution are based on JSON definitions, which you can view and edit in the Azure portal.

While you can use the portal to create and modify search solutions, it's often desirable to define the search objects in JSON and use the Azure Cognitive Service REST interface to create and modify them.

Get the endpoint and key for your Azure Cognitive Search resource

1. In the Azure portal, return to the **Overview** page for your Azure Cognitive Search resource; and in the top section of the page, find the **Url** for your resource (which looks like **https://resource\_name.search.windows.net**) and copy it to the clipboard.
2. In Visual Studio Code, in the Explorer pane, expand the **22-create-a-search-solution** folder and its **modify-search** subfolder, and select **modify-search.cmd** to open it. You will use this script file to run *cURL* commands that submit JSON to the Azure Cognitive Service REST interface.
3. In **modify-search.cmd**, replace the **YOUR\_SEARCH\_URL** placeholder with the URL you copied to the clipboard.
4. In the Azure portal, view the **Keys** page for your Azure Cognitive Search resource, and copy the **Primary admin key** to the clipboard.
5. In Visual Studio Code, replace the **YOUR\_ADMIN\_KEY** placeholder with the key you copied to the clipboard.
6. Save the changes to **modify-search.cmd** (but don't run it yet!)

Review and modify the skillset

1. In Visual studio Code, in the **modify-search** folder, open **skillset.json**. This shows a JSON definition for **margies-skillset**.
2. At the top of the skillset definition, note the **cognitiveServices** object, which is used to connect your Cognitive Services resource to the skillset.
3. In the Azure portal, open your Cognitive Services resource (not your Azure Cognitive Search resource!) and view its **Keys** page. Then copy **Key 1** to the clipboard.
4. In Visual Studio Code, in **skillset.json**, replace the **YOUR\_COGNITIVE\_SERVICES\_KEY** placeholder with the Cognitive Services key you copied to the clipboard.
5. Scroll through the JSON file, noting that it includes definitions for the skills you created using the Azure Cognitive Search user interface in the Azure portal. At the bottom of the list of skills, an additional skill has been added with the following definition:
6. {
7. "@odata.type": "#Microsoft.Skills.Text.V3.SentimentSkill",
8. "defaultLanguageCode": "en",
9. "name": "get-sentiment",
10. "description": "New skill to evaluate sentiment",
11. "context": "/document",
12. "inputs": [
13. {
14. "name": "text",
15. "source": "/document/merged\_content"
16. },
17. {
18. "name": "languageCode",
19. "source": "/document/language"
20. }
21. ],
22. "outputs": [
23. {
24. "name": "sentiment",
25. "targetName": "sentimentLabel"
26. }
27. ]

}

The new skill is named **get-sentiment**, and for each **document** level in a document, it, will evaluate the text found in the **merged\_content** field of the document being indexed (which includes the source content as well as any text extracted from images in the content). It uses the extracted **language** of the document (with a default of English), and evaluates a label for the sentiment of the content. Values for the sentiment label can be "positive", "negative", "neutral", or "mixed". This label is then output as a new field named **sentimentLabel**.

1. Save the changes you've made to **skillset.json**.

Review and modify the index

1. In Visual studio Code, in the **modify-search** folder, open **index.json**. This shows a JSON definition for **margies-index**.
2. Scroll through the index and view the field definitions. Some fields are based on metadata and content in the source document, and others are the results of skills in the skillset.
3. At the end of the list of fields that you defined in the Azure portal, note that two additional fields have been added:
4. {
5. "name": "sentiment",
6. "type": "Edm.String",
7. "facetable": false,
8. "filterable": true,
9. "retrievable": true,
10. "sortable": true
11. },
12. {
13. "name": "url",
14. "type": "Edm.String",
15. "facetable": false,
16. "filterable": true,
17. "retrievable": true,
18. "searchable": false,
19. "sortable": false

}

1. The **sentiment** field will be used to add the output from the **get-sentiment** skill that was added the skillset. The **url** field will be used to add the URL for each indexed document to the index, based on the **metadata\_storage\_path** value extracted from the data source. Note that index already includes the **metadata\_storage\_path** field, but it's used as the index key and Base-64 encoded, making it efficient as a key but requiring client applications to decode it if they want to use the actual URL value as a field. Adding a second field for the unencoded value resolves this problem.

Review and modify the indexer

1. In Visual studio Code, in the **modify-search** folder, open **indexer.json**. This shows a JSON definition for **margies-indexer**, which maps fields extracted from document content and metadata (in the **fieldMappings** section), and values extracted by skills in the skillset (in the **outputFieldMappings** section), to fields in the index.
2. In the **fieldMappings** list, note the mapping for the **metadata\_storage\_path** value to the base-64 encoded key field. This was created when you assigned the **metadata\_storage\_path** as the key and selected the option to encode the key in the Azure portal. Additionally, a new mapping explicitly maps the same value to the **url** field, but without the Base-64 encoding:
3. {
4. "sourceFieldName" : "metadata\_storage\_path",
5. "targetFieldName" : "url"

}

All of the other metadata and content fields in the source document are implicitly mapped to fields of the same name in the index.

1. Review the **ouputFieldMappings** section, which maps outputs from the skills in the skillset to index fields. Most of these reflect the choices you made in the user interface, but the following mapping has been added to map the **sentimentLabel** value extracted by your sentiment skill to the **sentiment** field you added to the index:
2. {
3. "sourceFieldName": "/document/sentimentLabel",
4. "targetFieldName": "sentiment"

}

Use the REST API to update the search solution

1. Right-click the **modify-search** folder and open an integrated terminal.
2. In the terminal pane for the **modify-search** folder, enter the following command to run the **modify-search.cmd** script, which submits the JSON definitions to the REST interface and initiates the indexing.

modify-search

1. When the script has finished, return to the **Overview** page for your Azure Cognitive Search resource in the Azure portal and view the **Indexers** page. The periodically select **Refresh** to track the progress of the indexing operation. It may take a minute or so to complete.

*There may be some warnings for a few documents that are too large to evaluate sentiment. Often sentiment analysis is performed at the page or sentence level rather than the full document; but in this case scenario, most of the documents - particularly the hotel reviews, are short enough for useful document-level sentiment scores to be evaluated.*

Query the modified index

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)

1. Close the **Search explorer** page to return to the **Overview** page.

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 Cognitive 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 Cognitive Search SDK

1. In Visual Studio Code, in the **Explorer** pane, browse to the **22-create-a-search-solution** 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 Cognitive 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 Cognitive 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
   * **Python**: app.py
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 Cognitive 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 Cognitive 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
   * **Python**: templates/search.html
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

**Python**

flask run

* 1. In the message that is displayed when the app starts successfully, follow the link to the running web application (*http://localhost:5000/* or *http://127.0.0.1:5000/*) to open the Margies Travel site in a web browser.
  2. In the Margie's Travel website, enter **London hotel** into the search box and click **Search**.
  3. Review the search results. They include the file name (with a hyperlink to the file URL), an extract of the file content with the search terms (*London* and *hotel*) emphasized, and other attributes of the file from the index fields.
  4. Observe that the results page includes some user interface elements that enable you to refine the results. These include:
  + A *filter* based on a facet value for the **metadata\_author** field. This demonstrates how you can use *facetable* fields to return a list of *facets* - fields with a small set of discrete values that can displayed as potential filter values in the user interface.
  + The ability to *order* the results based on a specified field and sort direction (ascending or descending). The default order is based on *relevancy*, which is calculated as a **search.score()** value based on a *scoring profile* that evaluates the frequency and importance of search terms in the index fields.
  1. Select the **Reviewer** filter and the **Positive to negative** sort option, and then select **Refine Results**.
  2. Observe that the results are filtered to include only reviews, and sorted based on the sentiment label.
  3. In the **Search** box, enter a new search for **quiet hotel in New York** and review the results.
  4. Try the following search terms:
  + **Tower of London** (observe that this term is identified as a *key phrase* in some documents).
  + **skyscraper** (observe that this word doesn't appear in the actual content of any documents, but is found in the *image captions* and *image tags* that were generated for images in some documents).
  + **Mojave desert** (observe that this term is identified as a *location* in some documents).
  1. Close the browser tab containing the Margie's Travel web site and return to Visual Studio Code. Then in the Python terminal for the **margies-travel** folder (where the dotnet or flask application is running), enter Ctrl+C to stop the app.

More information

To learn more about Azure Cognitive Search, see the [Azure Cognitive Search documentation](https://docs.microsoft.com/azure/search/search-what-is-azure-search).

Return to Microsoft Learn

Now that you have completed the exercise, [return to Microsoft Learn](https://docs.microsoft.com/learn/modules/create-azure-cognitive-search-solution/10-knowledge-check) to complete the knowledge check and earn points for completing this module.