4.4 - Enrich your data with Azure AI Language

Overview

Azure AI Language gives you the power of Natural Language Processing (NLP) to automatically understand and analyze text. You can use that power to enhance your search solutions.

In this module, you'll learn how to:

- Use Azure AI Language to enrich Azure AI Search indexes.
- Enrich an AI Search index with custom classes.

Introduction

Language Studio is included with Azure AI Language. Language Studio lets you use UI tools to explore and build AI models focused in language into your solutions.

These AI models have endpoints that can be used in search solutions to enrich indexes.

This module builds on Create a custom skill for Azure AI Search but uses a **custom text classification model** to enrich a search index.

Here you'll explore the features of Azure AI Language, see how to manually enrich an existing search index, and finally complete an exercise to create and enrich your own search index.

By the end of this module you'll learn to:

- Use Azure AI Language to enrich Azure AI Search indexes.
- Enrich a search index with custom classes.

> Note: This module assumes you already know how to create and use an Azure AI Search solution. If not, complete the Create an Azure AI Search solution module first.

Explore the available features of Azure AI Language

Here you'll explore the features Azure AI Language offers and then use the demo-like environment to test a preconfigured sentiment model. You'll then see the steps to create, train and deploy a custom model for conversational language understanding.

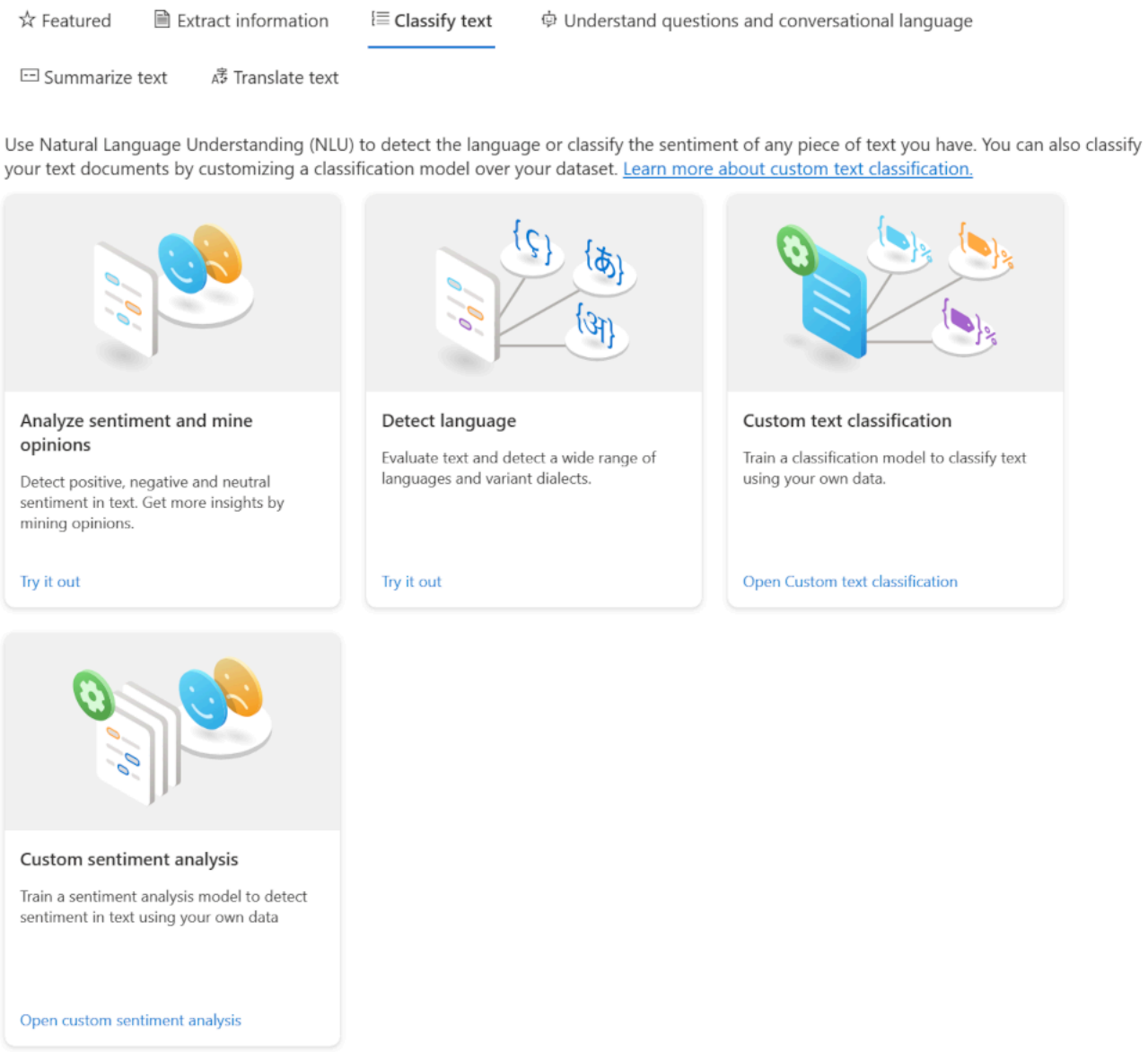
Azure AI Language features

Azure AI Language groups its features into the following areas:

1. Classify text
2. Understand questions and conversational language
3. Extract information
4. Summarize text

5. Translate text

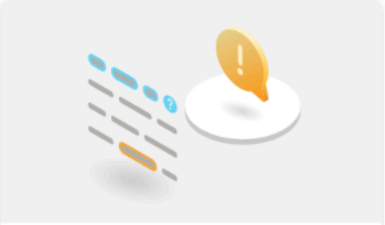
Classify text



Understand questions and conversational language

☆ Featured | Extract information | Classify text | Understand questions and conversational language | Summarize text | Translate text

Retrieve the most appropriate answer to questions using question answering (CQA) or classify intents and extract entities for conversational utterances using conversational language understanding (CLU). Use orchestration workflow to create one project that routes queries between multiple CQA and CLU projects. Learn more about understanding conversational language.



Answer questions

Use the prebuilt question answering API to get answers to questions over unstructured text.

Try it out

Custom question answering

Next generation of QnAMaker

Customize the list of questions and answers extracted from your content corpus to provide a conversational experience that suits your needs.

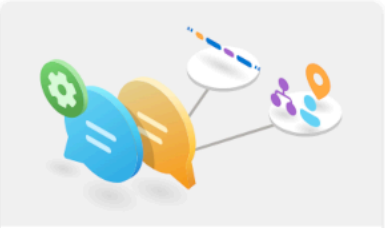
Open Custom question answering

Conversational language understanding

Next generation of LUIS

Classify utterances into intents and extract information with entities to build natural language into apps, bots, and IoT devices.

Open Conversational language understanding

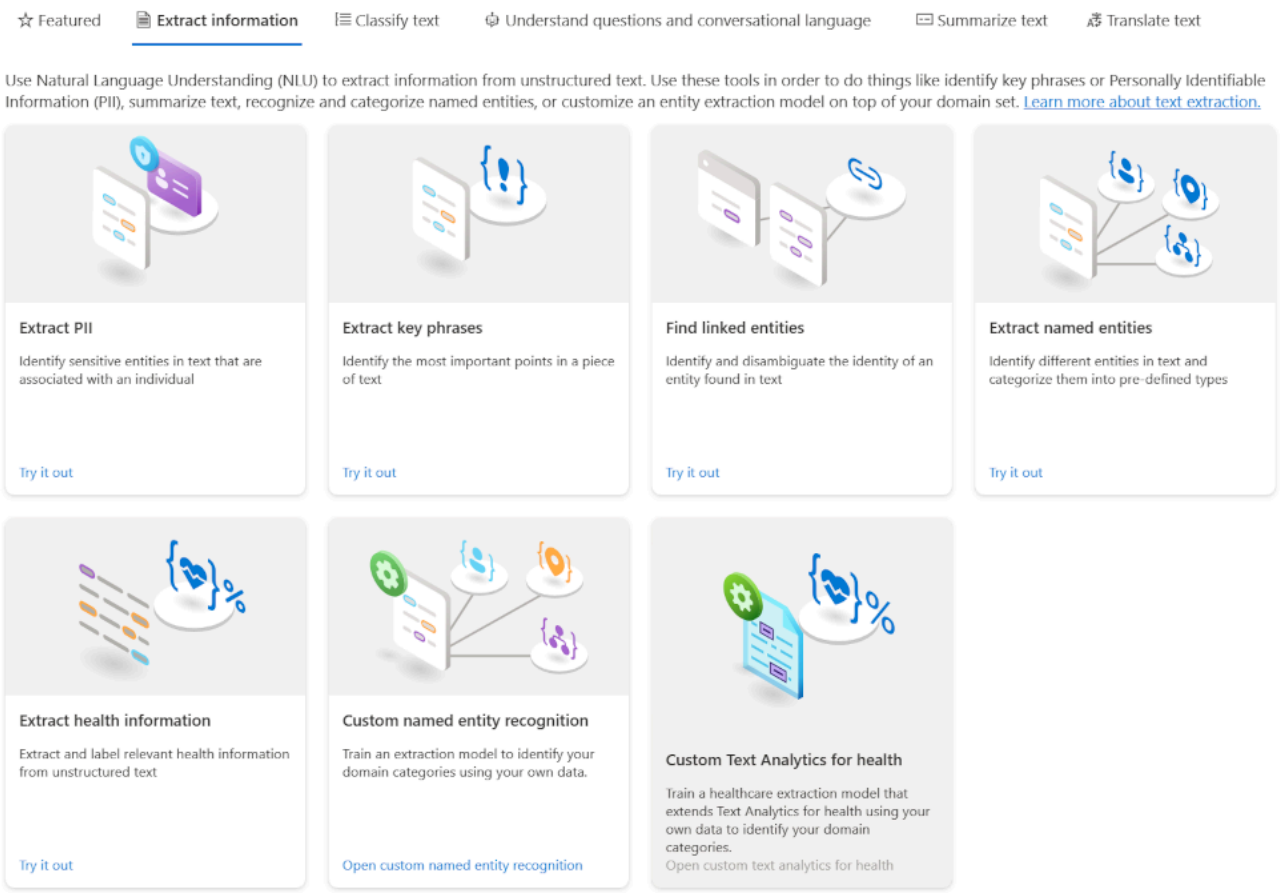


Orchestration workflow

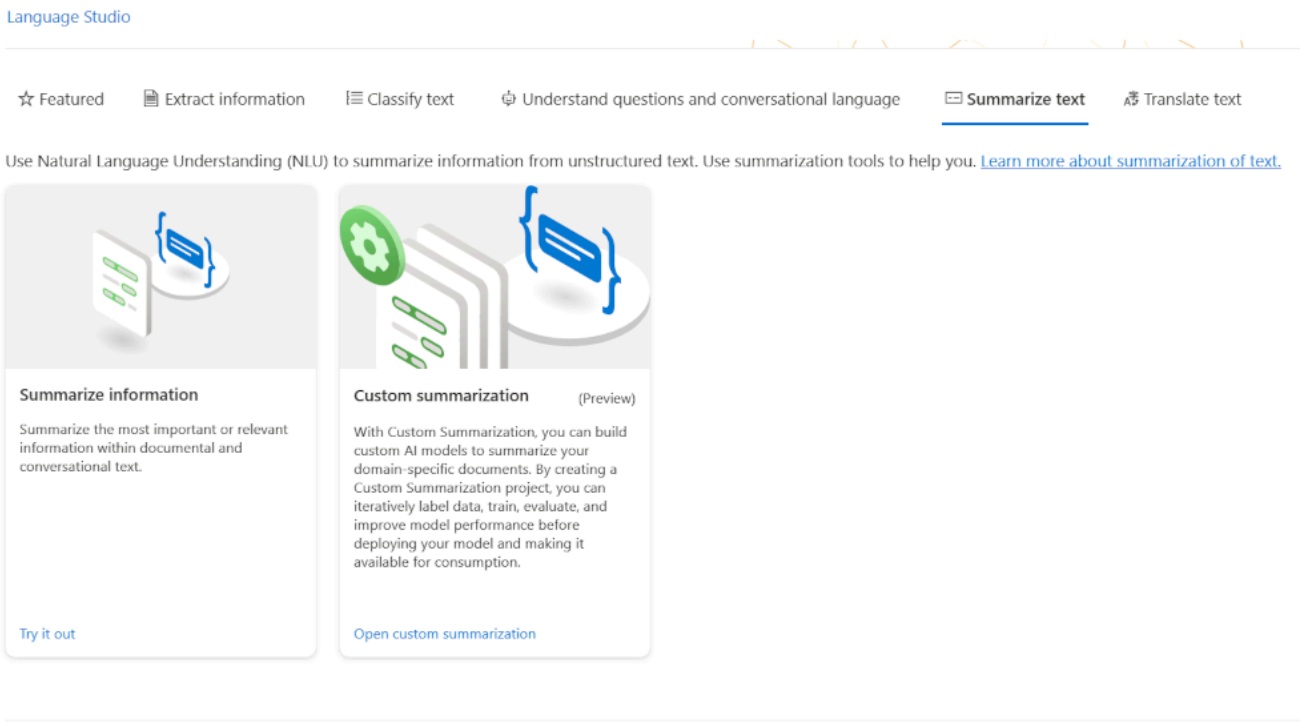
Connect and orchestrate CLU, Custom question answering & LUIS projects together in one single project.

Open Orchestration workflow

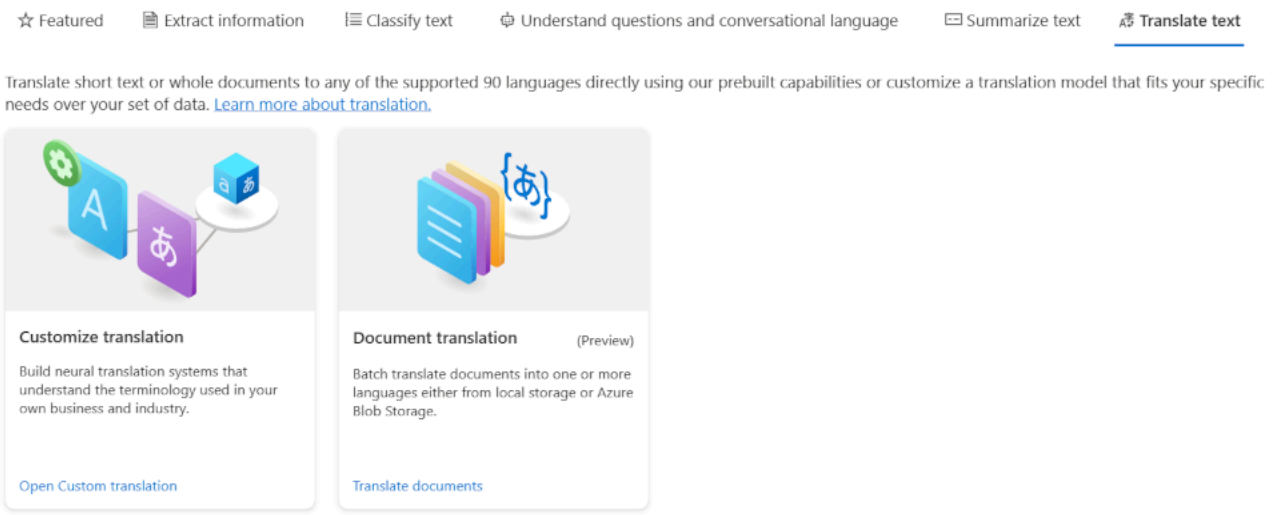
Extract information



Summarize text



Translate text



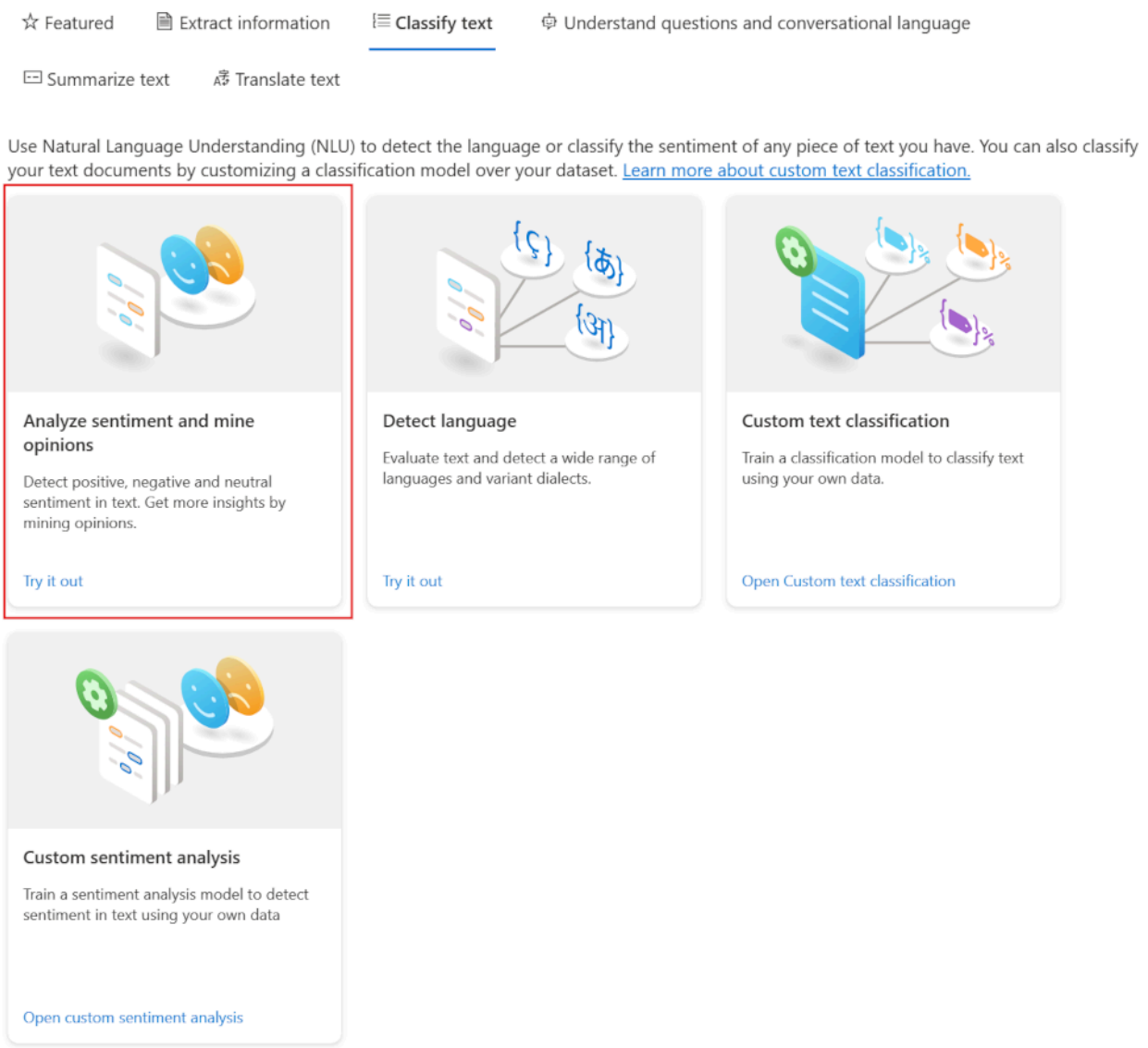
Features can be either preconfigured or customizable. Preconfigured features can be tested straight away with a demo-like environment directly inside Language Studio. You can use them straight out of the box.

The other features with * and green cogs in their logo need user customization. They require you to train their models so they fit your data better. After you have train them, you deploy and can then use them to power your apps or use the same demo-like testing environment.

Test and use preconfigured language features

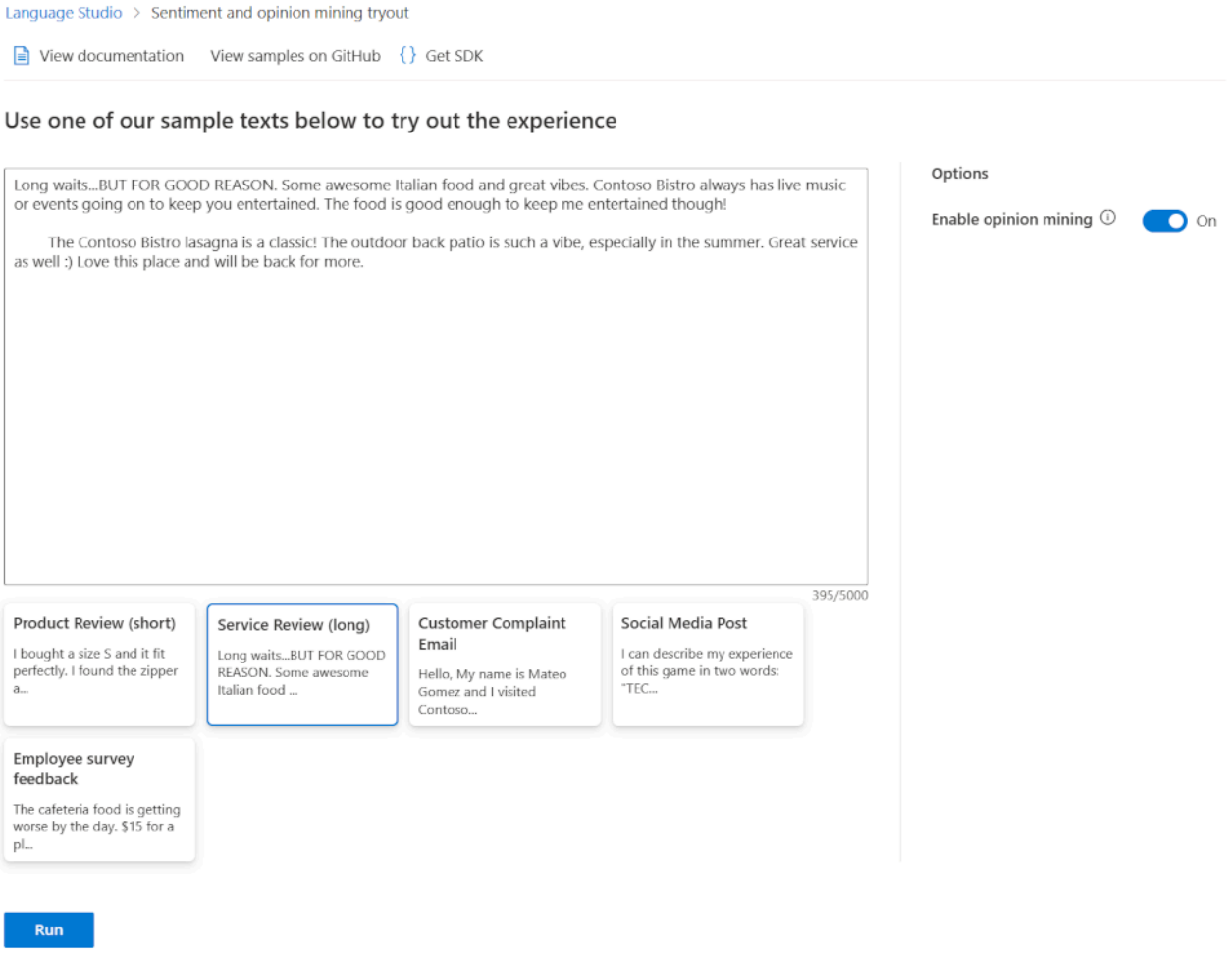
1. Go to the Language Studio and sign in with your Azure account.

2. If you haven't got a language resource, create one.



3. Scroll down to the classify text section, then select **Analyze sentiment and mine options**.

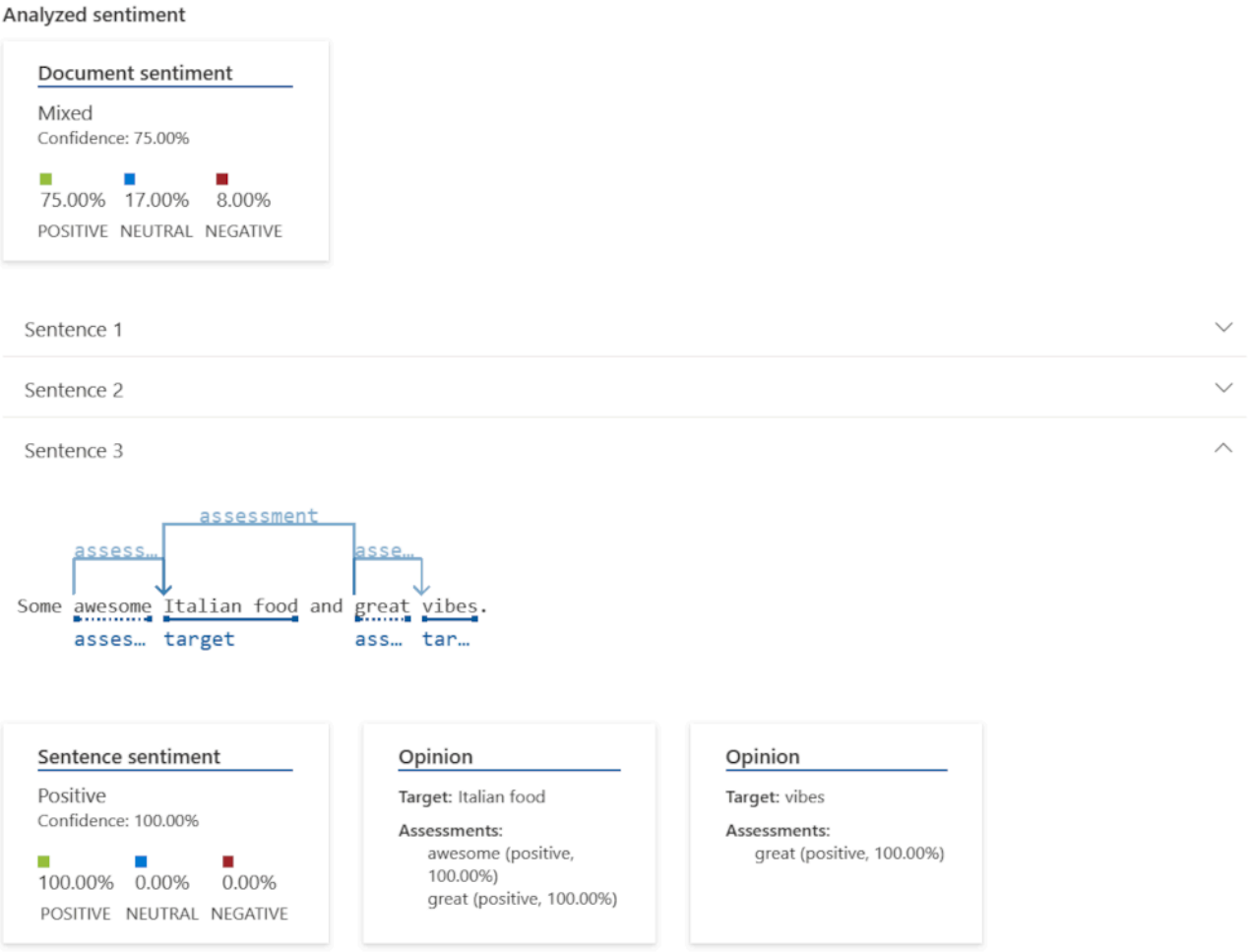
4. Scroll down, then select **Service Review (long)**.



Read the review text, how positive do you think it is?

5. Select **Run**.

6. Examine the results.



You can see how the sentiment model has identified words that help it calculate an overall positive score. In the sample text, the model returns an **75% positive** result for the whole document.

You can also drill down into individual sentences for more detail. Azure AI Language then gives you guidance for how to use the model in your own apps.

For example, you can get the same JSON response from the model by using this curl statement in your development environment:

```
curl -v -X POST "https://<YOUR_ENDPOINT_HERE>/language/:analyze-text?api-version=2022-05-01" -H "Content-Type: application/json" -H "Ocp-Apim-Subscription-Key: subscription key" --data-ascii "{\"kind\":\"SentimentAnalysis\",\"analysisInput\":{\"documents\":[{\"id\":\"documentId\",\"text\":\"Long waits...BUT FOR GOOD REASON. Some awesome Italian food and great vibes. Contoso Bistro always has live music or events going on to keep you entertained. The food is good enough to keep me entertained though!\n\n
The Contoso Bistro lasagna is a classic! The outdoor back patio is such a vibe, especially in the summer. Great service as well :) Love this place and will be back for more.\",\"language\":\"en\"}]},\"parameters\":{\"opinionMining\":true}}"
```

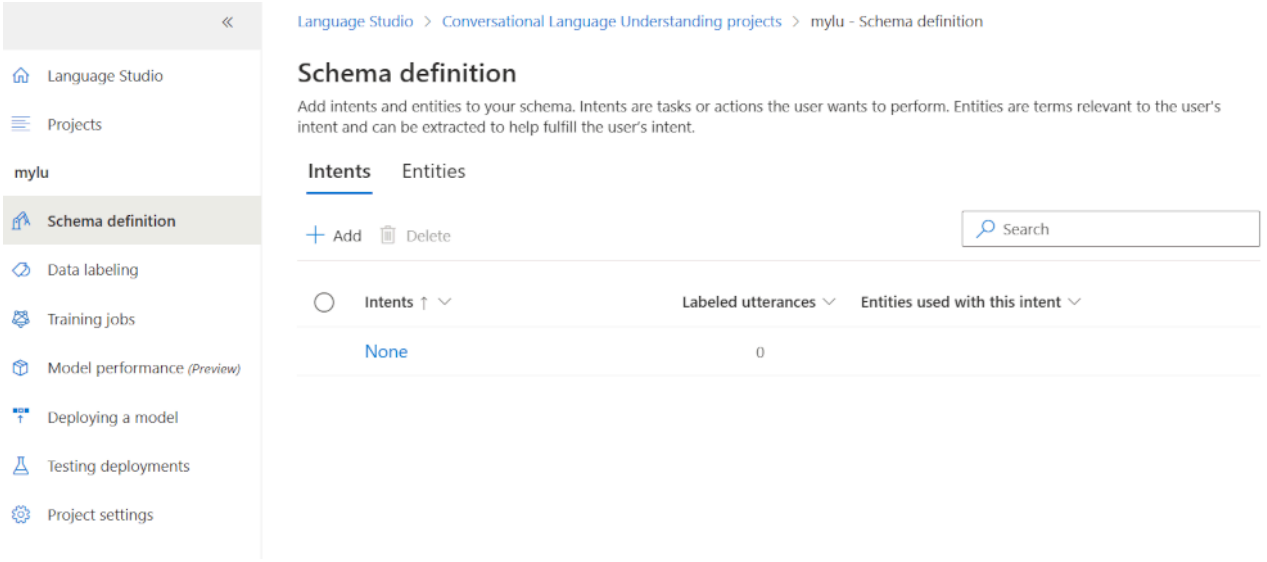
Create, train, and deploy a conversation language understanding model

Each of the customizable features in Azure AI Language needs different steps to create the models. In this example, you'll see how to create a conversation language understanding model.

Conversational language understanding aims to build a model that **predicts intention from conversational text.** For example, imagine an email app that you can chat with to send email messages or flag emails. You train the model on sentences like "please add a flag to that email" or "okay, ready to send". These sentences would be translated to intents for `flag email` and `send email`.

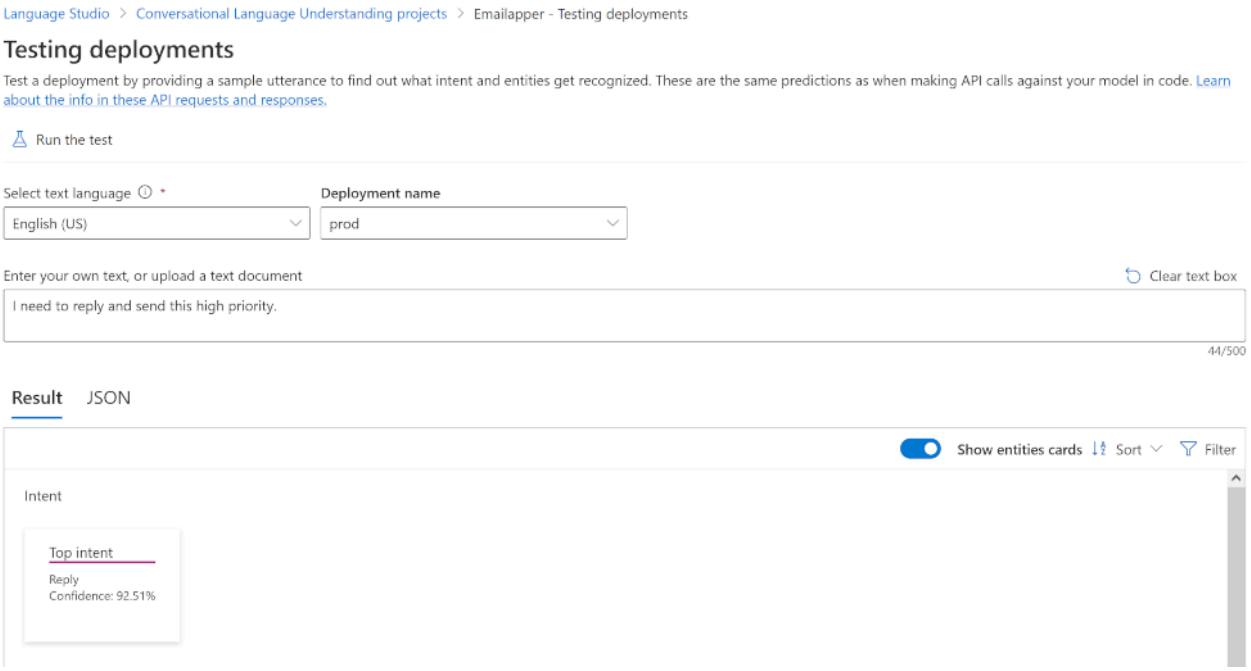
To use language understanding you'll need to have an Azure AI Language resource already created in Azure, then you can carry out the following steps in Language Studio:

1. From the home page, you select **Conversational Language Understanding** in the **Understand questions and conversational language** tab.
2. Select **+ Create new project**.
3. Enter a name for the new project.
4. Select your language.
5. Enter a description, then select **Next**.
6. Select **Create**.



Language Studio guides you through the remaining steps. Follow the left navigation from top to bottom to:

1. Create your schema definition. This involves adding all the intents and entities that your app is interested in.
2. Label data. You **provide example chats and utterances along with how they map to entities and intents.**
3. Train your model. Once you've added the data labeling information, you can start training your model. You can either split all your data with 80% for training and 20% for testing. Or you can create your own manual split.
4. Review the performance of your model.
5. Deploy your model. When you're happy with the performance of your model, you deploy it. This makes it available to be called as an API from your app and test it.
6. Test your deployment. This option allows you to test your model in the same way as preconfigured models.

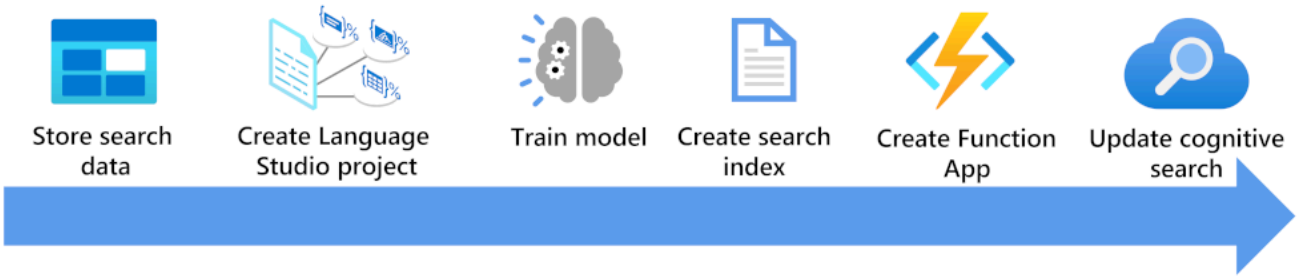


Whichever feature your use you end up with a model that you can use in apps to add language understanding.

In the context of Azure AI Search, these models can help us enrich our search indexes to create better search experiences. Or provide answers when users ask questions.

Enrich a search index in Azure AI Search with custom classes and Azure AI Language

Custom text classification allows you to map a passage of text to different user defined classes. For example, you could train a model on the synopsis on the back cover of books to automatically identify a books genre. You then use that identified genre to enrich your online shop search engine with a genre facet.



Here, you'll see what you need to consider to enrich a search index using a custom text classification model.

- Store your documents so they can be accessed by Language Studio and Azure AI Search indexers
- Create a custom text classification project
- Train and test your model
- Create a **search index** based on your stored documents
- Create a **function** app that uses your deployed trained model

- Update your search solution, your index, indexer, and custom skillset

Store your data

Azure Blob storage can be accessed from both Language Studio and Azure AI Services. The container needs to be accessible, so the simplest option is to choose Container, but it's also possible to use private containers with some additional configuration.

Along with your data, you also need a way to assign classifications for each document. Language Studio provides a graphical tool that you can use to classify each document one at a time manually.

You can choose between two different kinds of project, if a document maps to a single class use a single label classification project. If you could map a document to more than one class, use the multi label classification project.

If you don't want to manually classify each document, you can label all your documents before you create your Azure AI Language project. This process involves creating a labels JSON document in this format:

```
{
    "projectFileVersion": "2022-05-01",
    "stringIndexType": "Utf16CodeUnit",
    "metadata": {
      "projectKind": "CustomMultiLabelClassification",
      "storageInputContainerName": "{CONTAINER-NAME}",
      "projectName": "{PROJECT-NAME}",
      "multilingual": false,
      "description": "Project-description",
      "language": "en-us"
    },
    "assets": {
      "projectKind": "CustomMultiLabelClassification",
      "classes": [
        {
          "category": "Class1"
        },
        {
          "category": "Class2"
        }
      ],
      "documents": [
          {
              "location": "{DOCUMENT-NAME}",
              "language": "{LANGUAGE-CODE}",
              "dataset": "{DATASET}",
              "classes": [
                  {
                      "category": "Class1"
                  },
                  {
                      "category": "Class2"
```

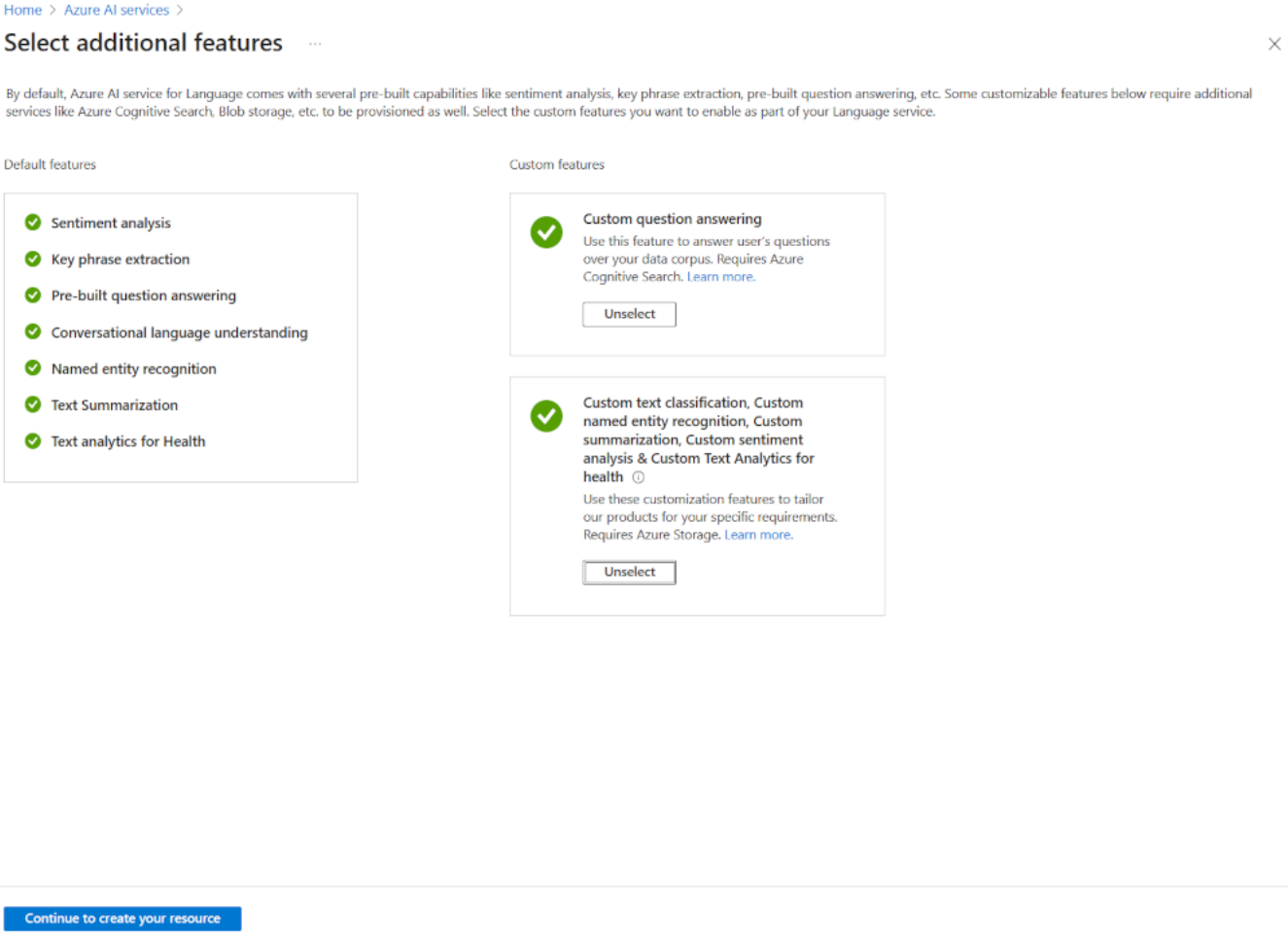
```
                }
            ]
        }
    ]
}
```

You add as many classes as you have to the `classes` array. You add an entry for each document in the `documents` array including which classes the document matches.

Create your Azure AI Language project

There are two ways to create your Azure AI Language project. If you start using the Language Studio without first creating a language service in the Azure portal, Language Studio will offer to create one for you.

The most flexible way to create a Azure AI Language project is to first create your language service using the Azure portal. If you choose this option, you get the option to add custom features.



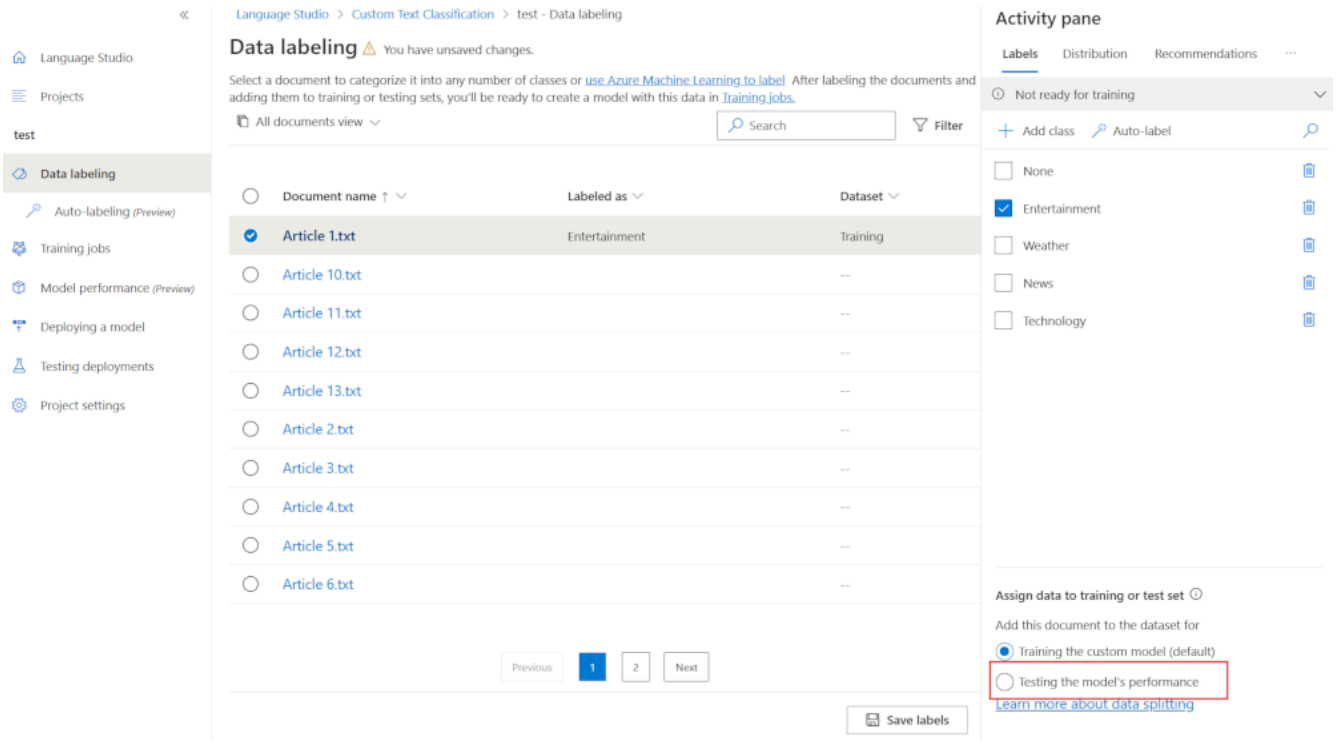
As you were going to create a custom text classification, select that custom feature when creating your language service. You'll also link the language service to a storage account using this method.

Once the resource has been deployed, you can navigate directly to the Language Studio from the overview pane of the language service. You can then create a new custom text classification project.

> Note: If you have created your language service from Language Studio you might need to follow these steps, Set roles for your Azure Language resource and storage account, to connect your storage container to your custom text classification project.

Train your classification model

As with all AI models, you need to have identified data that you can use to train it. The model needs to see examples of how to map data to a class and have some examples it can use to test the model. You can choose to let the model automatically split your training data, be default it will use 80% of the documents to train the model and 20% to blind test it. If you have some specific documents that you want to test your model with, you can label documents for testing.



In Language Studio, in your project, select Data labeling. You'll see all your documents. Select each document you'd like to add to the testing set, then select **Testing the model's performance.** Save your updated labels and then create a new training job.

Create search index

There isn't anything specific you need to do to create a search index that will be enriched by a custom text classification model. Follow the steps in Create an Azure AI Search solution. You'll be updating the index, indexer, and custom skill after you've created a function app.

Create an Azure function app

You can choose the language and technologies you want for your function app. The app needs to be able to **pass JSON to the custom text classification endpoint**, for example:

```
{
    "displayName": "Extracting custom text classification",
    "analysisInput": {
        "documents": [
            {
                "id": "1",
                "language": "en-us",
                "text": "This film takes place during the events of Get Smart. Bruce and
Lloyd have been testing out an invisibility cloak, but during a party, Maraguayan agent
```

```
Isabelle steals it for El Presidente. Now, Bruce and Lloyd must find the cloak on their
own because the only non-compromised agents, Agent 99 and Agent 86  are in Russia"
            }
        ]
      },
    "tasks": [
        {
        "kind": "CustomMultiLabelClassification",
        "taskName": "Multi Label Classification",
        "parameters": {
            "project-name": "movie-classifier",
            "deployment-name": "test-release"}
        }
    ]
}
```

Then process the JSON response from the model, for example:

```
{
  "jobId": "be1419f3-61f8-481d-8235-36b7a9335bb7",
  "lastUpdatedDateTime": "2022-06-13T16:24:27Z",
  "createdDateTime": "2022-06-13T16:24:26Z",
  "expirationDateTime": "2022-06-14T16:24:26Z",
  "status": "succeeded",
  "errors": [],
  "displayName": "Extracting custom text classification",
  "tasks": {
    "completed": 1,
    "failed": 0,
    "inProgress": 0,
    "total": 1,
    "items": [
      {
        "kind": "CustomMultiLabelClassificationLROResults",
        "taskName": "Multi Label Classification",
        "lastUpdateDateTime": "2022-06-13T16:24:27.7912131Z",
        "status": "succeeded",
        "results": {
          "documents": [
            {
              "id": "1",
              "class": [
                {
                  "category": "Action",
                  "confidenceScore": 0.99
                },
                {
                  "category": "Comedy",
                  "confidenceScore": 0.96
```

```
                }
              ],
              "warnings": []
            }
          ],
          "errors": [],
          "projectName": "movie-classifier",
          "deploymentName": "test-release"
        }
      }
    ]
  }
}
```

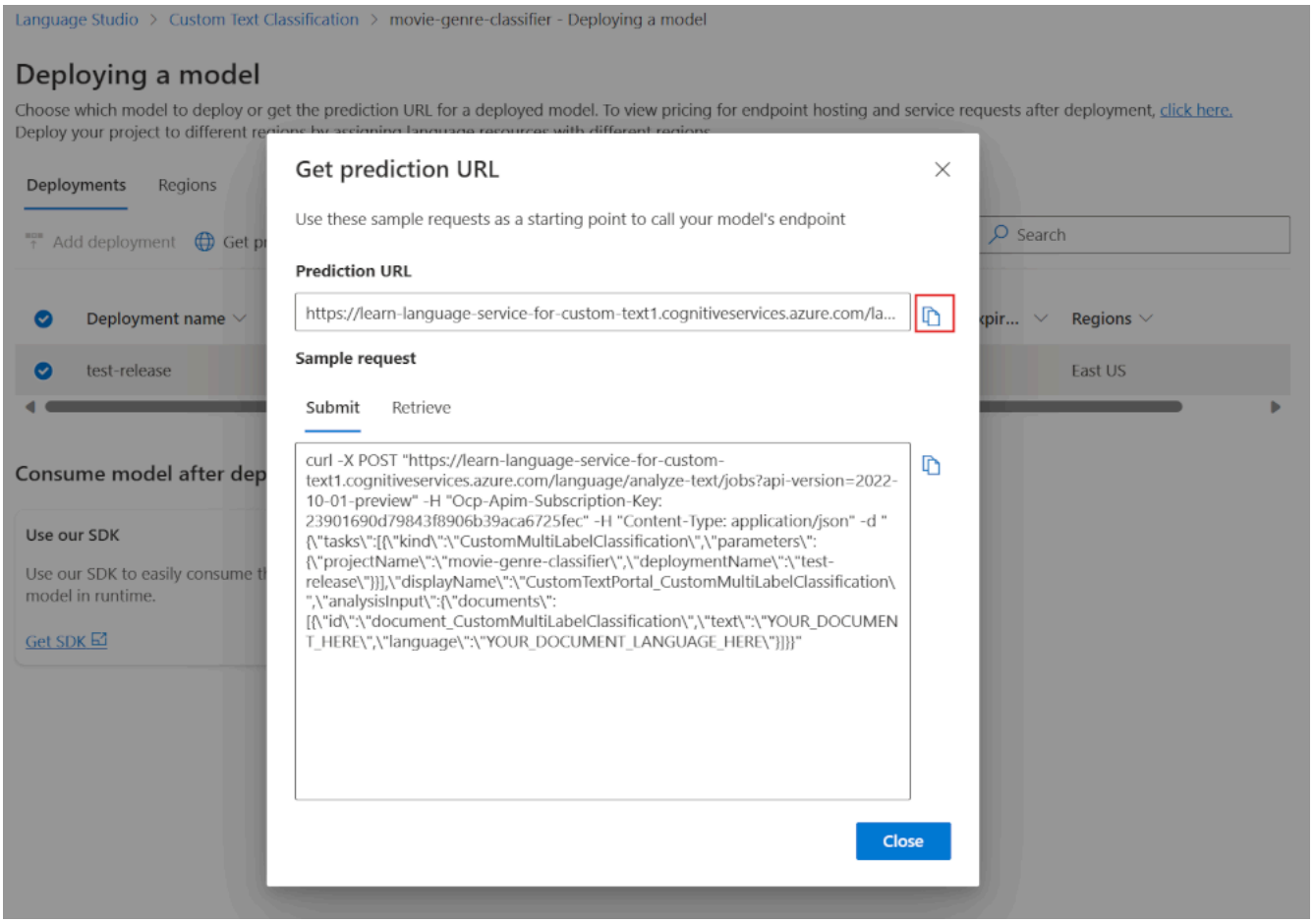
The **function then returns a structured JSON message back to a custom skillset in AI Search**, for example:

```
[{"category": "Action", "confidenceScore": 0.99}, {"category": "Comedy",
"confidenceScore": 0.96}]
```

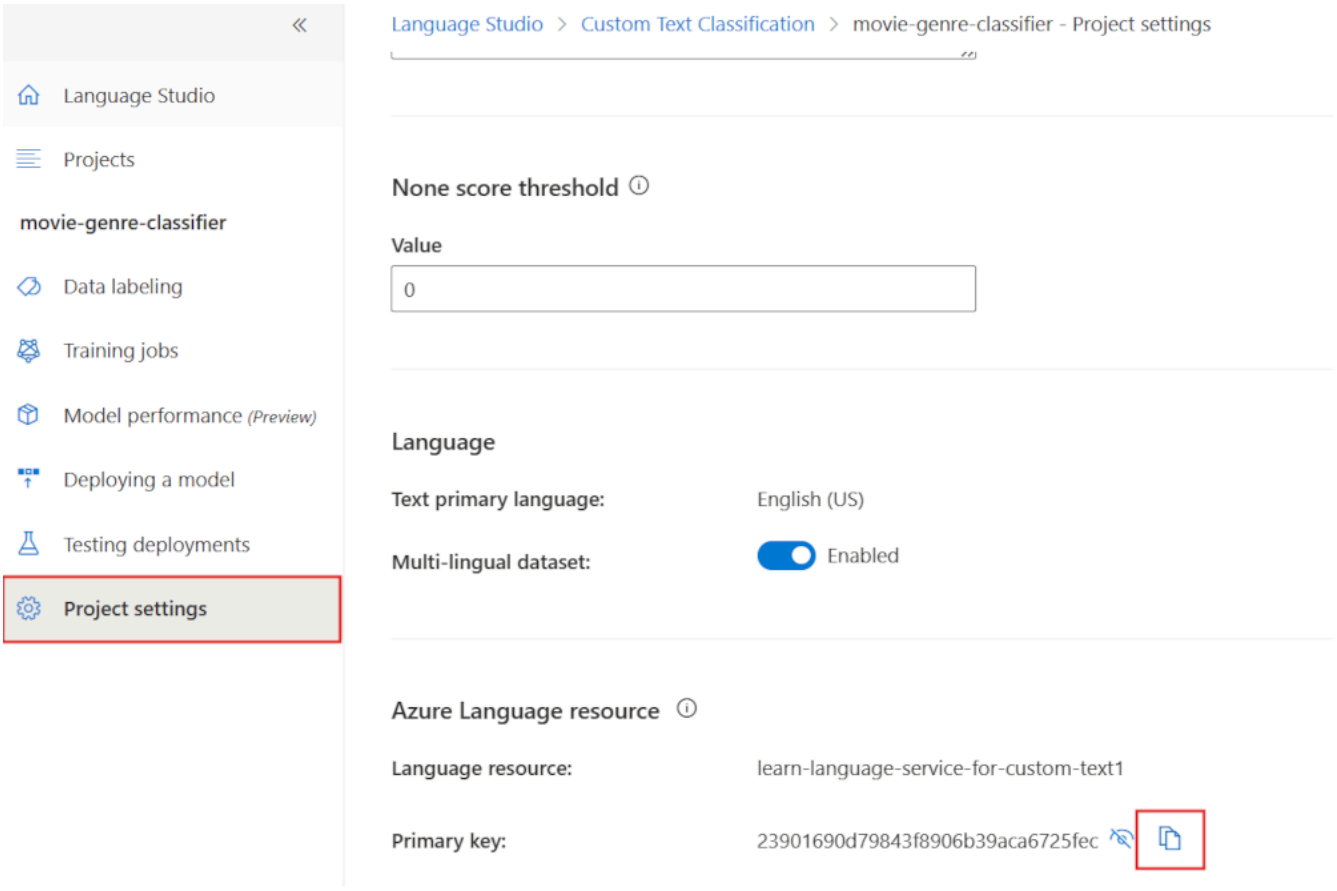
There are five things the function app needs to know:

1. The text to be classified.
2. The endpoint for your trained custom text classification deployed model.
3. The primary key for the custom text classification project.
4. The project name.
5. The deployment name.

The first is passed from your custom skillset in AI Search to the function as input. The remaining four can be found in Language Studio.



The endpoint and deployment name is on the deploying a model pane.



The project name and primary key are on the project settings pane.

Update your Azure AI Search solution

There are three changes in the Azure portal you need to make to enrich your search index.

1. You need to **add a field to your index** to store the custom text classification enrichment.
2. You need to **add a custom skillset to call your function app** with the text to classify.
3. You need to **map the response from the skillset into the index**.

Add a field to an existing index

In the Azure portal, go to your AI Search resource, select the index and you'll add JSON for in this format:

```
{
  "name": "classifiedtext",
  "type": "Collection(Edm.ComplexType)",
  "analyzer": null,
  "synonymMaps": [],
  "fields": [
    {
      "name": "category",
      "type": "Edm.String",
      "facetable": true,
      "filterable": true,
      "key": false,
      "retrievable": true,
      "searchable": true,
      "sortable": false,
      "analyzer": "standard.lucene",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
    {
      "name": "confidenceScore",
      "type": "Edm.Double",
      "facetable": true,
      "filterable": true,
      "retrievable": true,
      "sortable": false,
      "analyzer": null,
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    }
  ]
}
```

This JSON adds a compound field to the index to store the class in a `category` field that is searchable. The second `confidenceScore` field stores the confidence percentage in a double field.

Edit the custom skillset

In the Azure portal, select the skillset and add JSON in this format:

```
{
  "@odata.type": "#Microsoft.Skills.Custom.WebApiSkill",
  "name": "Genre Classification",
  "description": "Identify the genre of your movie from its summary",
  "context": "/document",
  "uri": "https://learn-acs-lang-serives.cognitiveservices.azure.com/language/analyze-
text/jobs?api-version=2022-05-01",
  "httpMethod": "POST",
  "timeout": "PT30S",
  "batchSize": 1,
  "degreeOfParallelism": 1,
  "inputs": [
    {
      "name": "lang",
      "source": "/document/language"
    },
    {
      "name": "text",
      "source": "/document/content"
    }
  ],
  "outputs": [
    {
      "name": "text",
      "targetName": "class"
    }
  ],
  "httpHeaders": {}
}
```

This `WebApiSill` skill definition specifies that the language and the contents of a document are passed as inputs to the function app. The app will return JSON text named `class`.

Map the output from the function app into the index

The last change is to map the output into the index. In the Azure portal, select the indexer and edit the JSON to have a new output mapping:

```
{
  "sourceFieldName": "/document/class",
  "targetFieldName": "classifiedtext"
}
```

The indexer now knows that the output from the function app `document/class` should be stored in the `classifiedtext` field. As this has been defined as a compound field, the function app has to return a JSON array containing a `category` and `confidenceScore` field.

You can now search an enriched search index for your custom classified text.

Exercise: Enrich a search index in Azure AI Search with custom classes

You've built a search solution and now want to add Azure AI Services for language enrichments to your indexes.

In this exercise, you'll create an Azure AI Search solution and enrich an index with the results from a Language Studio custom text classification project. You'll create a function app to connect search and your classification model together.

> **Note** To complete this exercise, you will need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free.

Set up your development environment with Python, VS Code and VS Code Extensions

Install these tools to complete this exercise. You can still follow along with the steps without these tools.

1. Install VS Code
2. Install Azure Core Functions Tool
3. Install Azure Tools Extensions for VS Code
4. Install Python 3.8 for your operating system.
5. Install Python Extension for VS Code

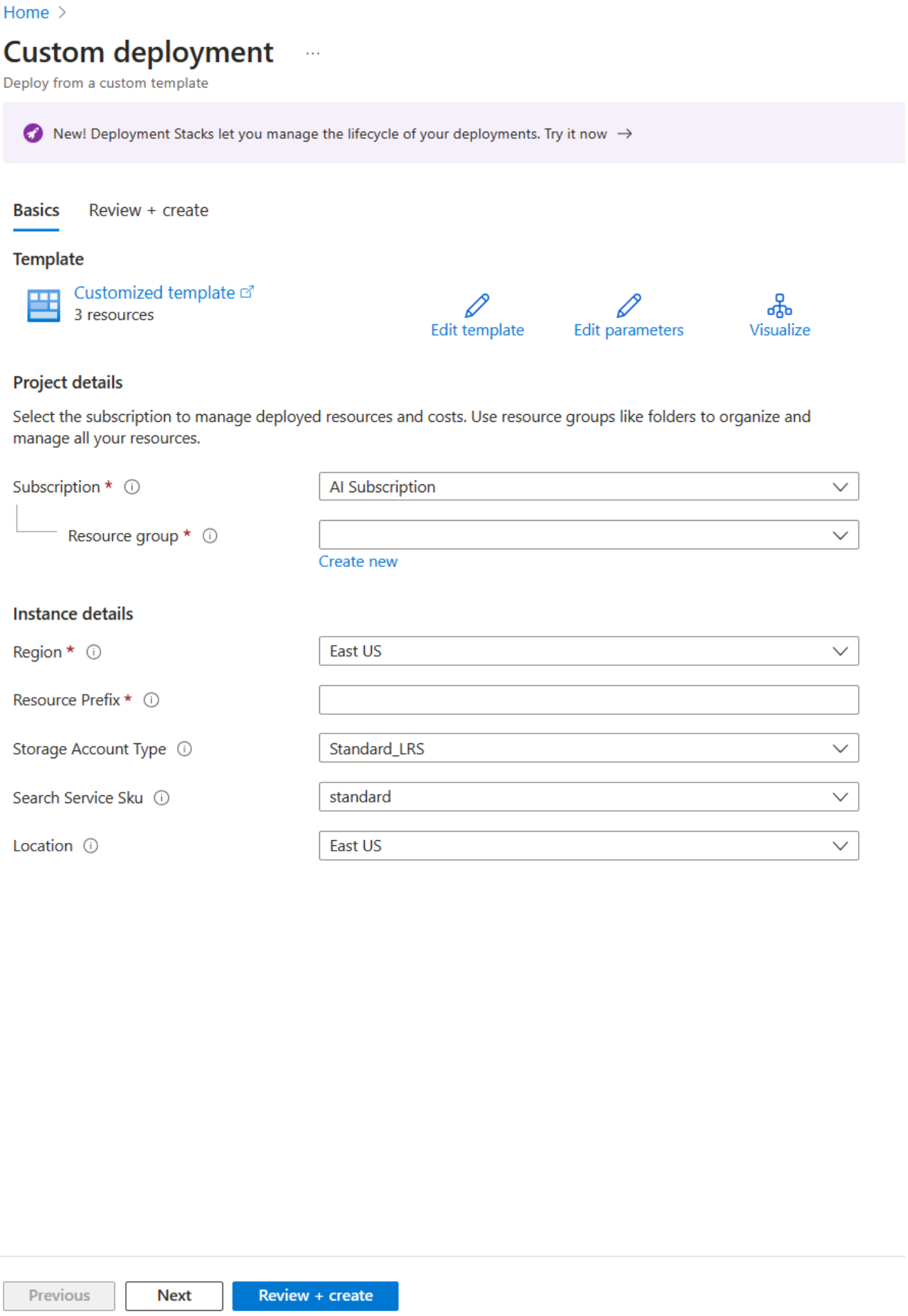
Set up your Azure resources

To save you time, select this Azure ARM template to create resources you'll need later in the exercise.

Deploy a pre-built ARM template

1. 

select this link to create your starting resources. You might need to copy and paste the direct link into your search bar.

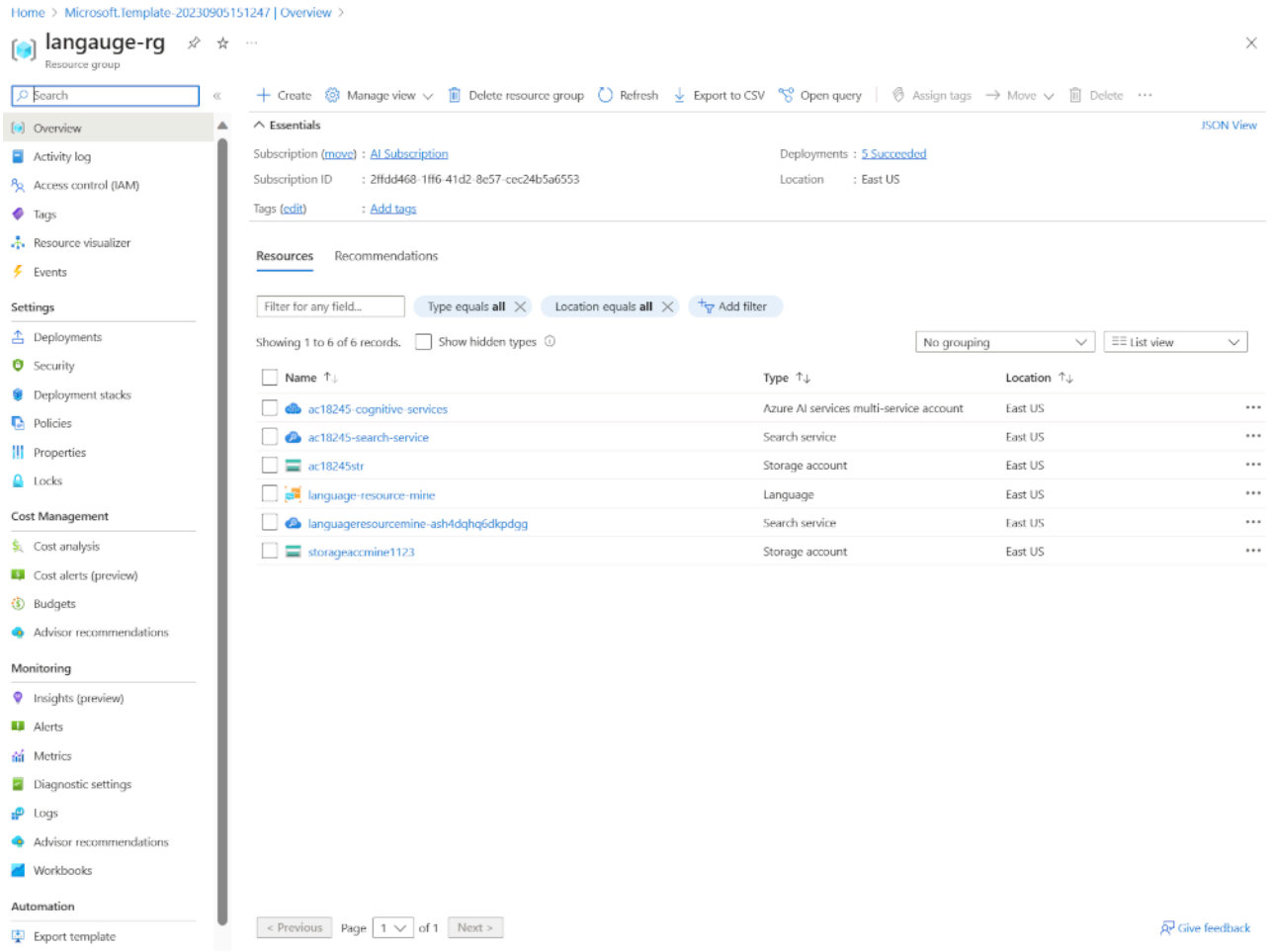


2. In **Resource group**, select **Create new**, name it **cog-search-language-exe**.
3. In **Region**, select a supported region that is close to you.
4. The **Resource Prefix** needs to be globally unique, enter a random numeric and lower-case character prefix, for example **acs18245**.

5. In **Location**, select the same region you chose above.
6. Select **Review + create**.
7. Select **Create**.

> **Note** There's an error shown, **You will need to Agree to the terms of service below to create this resource successfully.**, by selecting **Create** you are agreeing to them.

8. Select **Go to resource group** to see all the resources that you've created.



You'll be setting up an Azure Cognitive Search index, creating an Azure function, and creating a Language Studio project to identify movie genres from their summaries.

Upload sample data to train language services

This exercise uses 210 text files that contain a plot summary for a movie. The text files name is the movie title. The folder also contains a **movieLabels.json** file that maps the genres of a movie to the file, for each file there's a JSON entry like this:

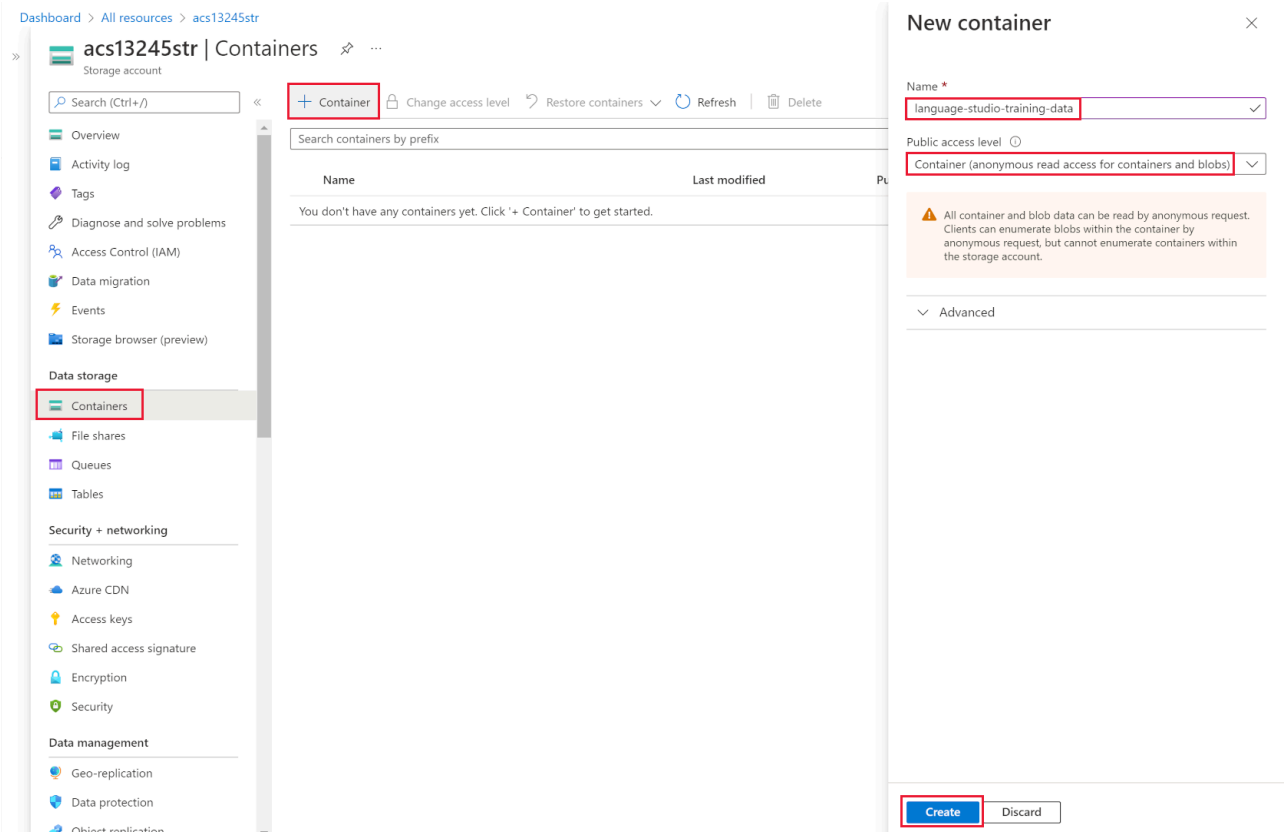
```
{
    "location": "And_Justice_for_All.txt",
    "language": "en-us",
    "classifiers": [
        {
            "classifierName": "Mystery"
        },
        {
            "classifierName": "Drama"
        },
        {
```

```
            "classifierName": "Thriller"
        },
        {
            "classifierName": "Comedy"
        }
    ]
},
```

1. Navigate to **Labfiles/04-enrich-custom-classes** and extract the **movies summary.zip** folder containing all the files.

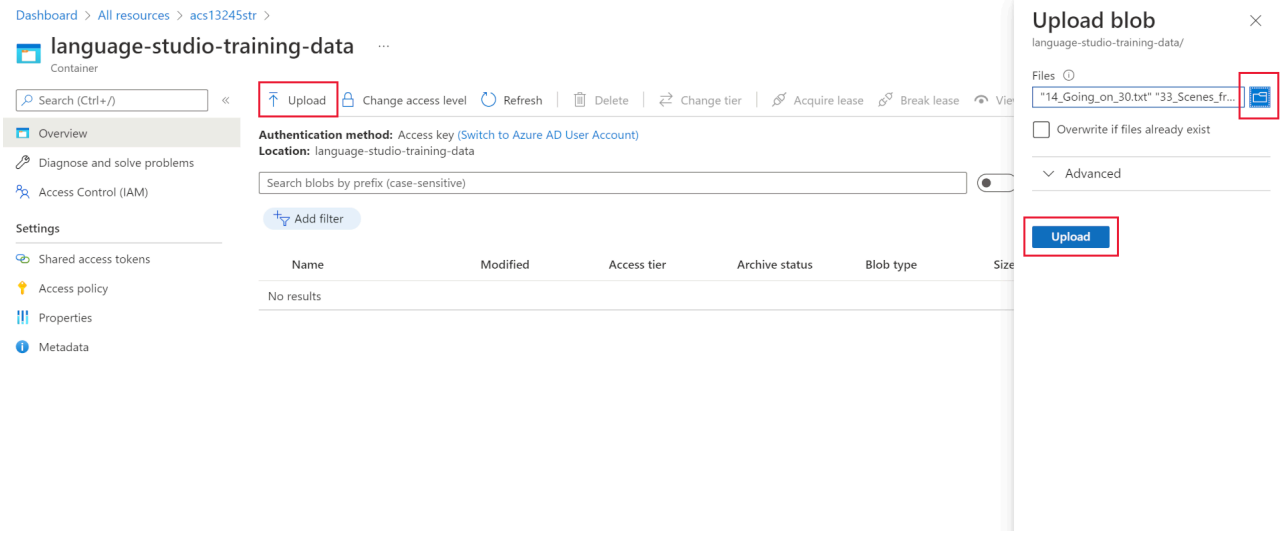
> **Note** You use these files to train a model in Language Studio, and will also index all the files in Azure AI Search.

2. In the Azure portal, select **Resource groups**, then select your resource group.
3. Select the storage account you created, for example **acs18245str**.
4. Select **Configuration** from the left pane, select the **Enable** option for the *Allow Blob anonymous access* setting and then select **Save** at the top of the page.



5. Select **Containers** from the left, then select **+ Container**.
6. In the **New container** pane, in **Name**, enter **language-studio-training-data**.
7. In **Anonymous access level**, choose **Container (anonymous read access for containers and blobs)** and select **Create**.

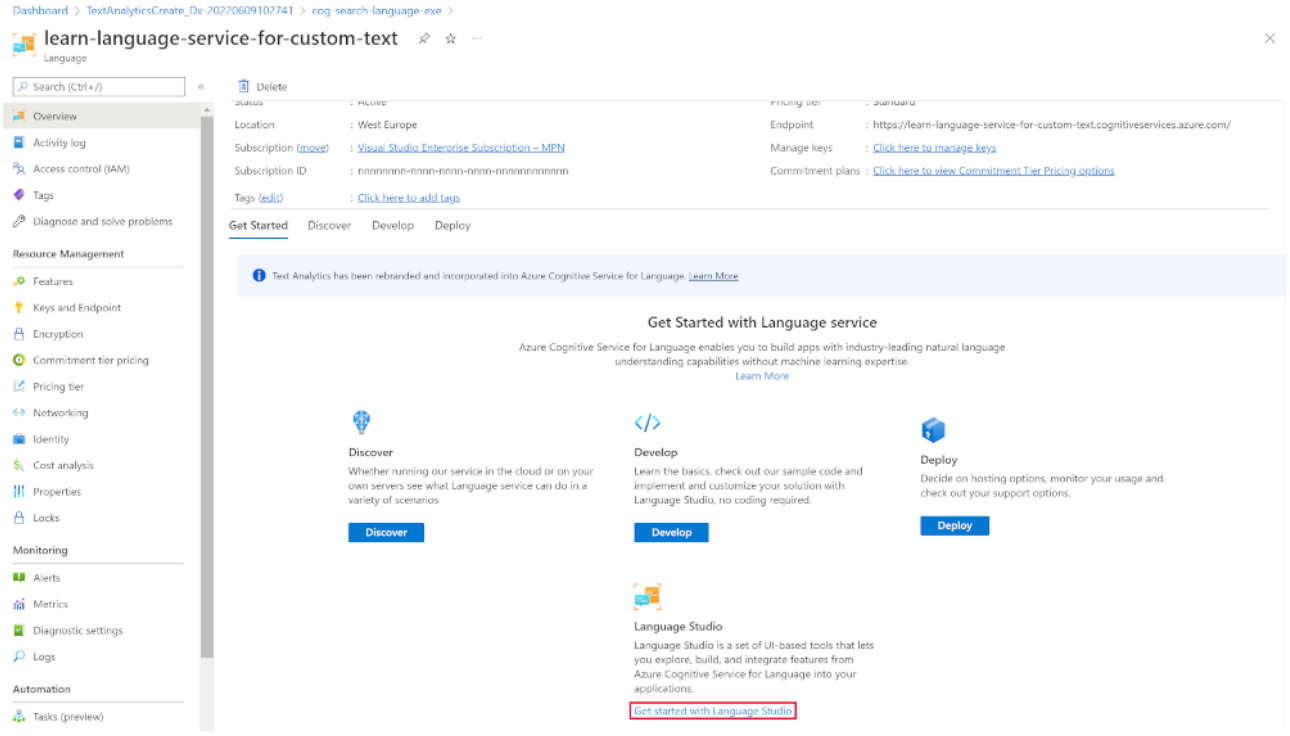
8. Select the new container you just created, **language-studio-training-data**.



9. Select **Upload** at the top of the pane.
10. In the **Upload blob** pane, select **Browse for files**.
11. Navigate to where you extracted the sample files, select all the text (`.txt`) and json (`.json`) files.
12. Select **Upload** in the pane.
13. Close the **Upload blob** pane.

Create a language resource

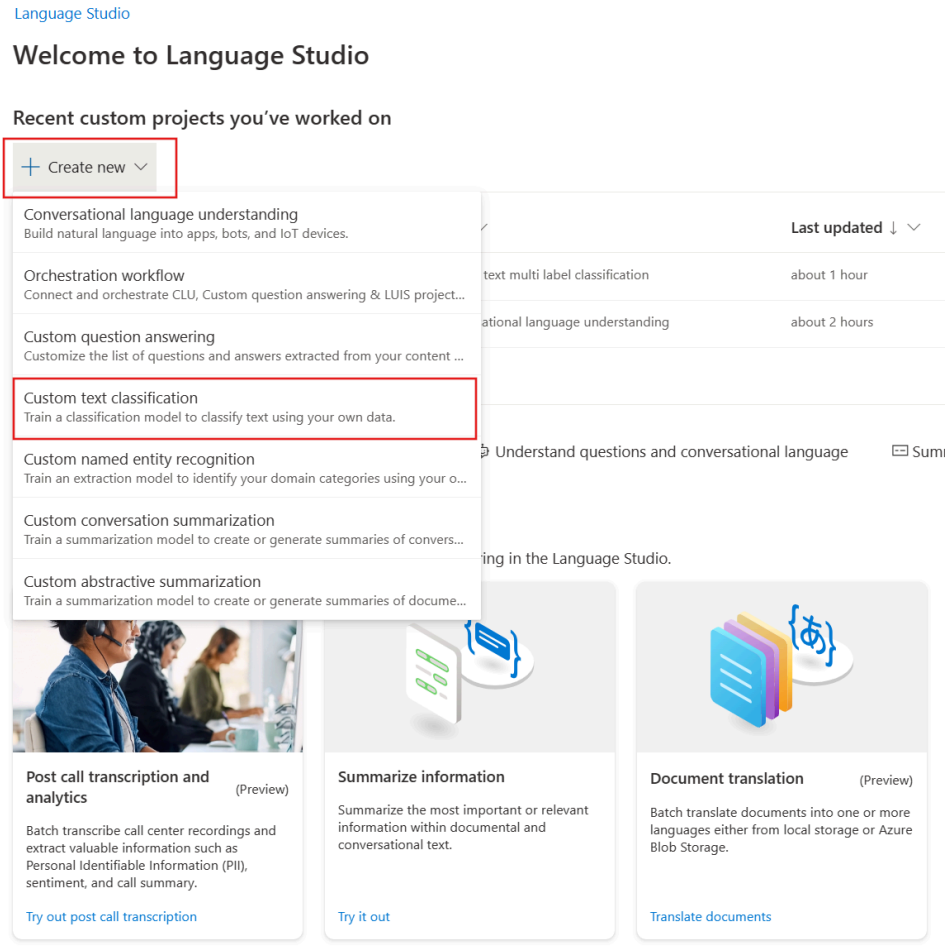
1. In the breadcrumb link at the top of the page, select **Home**.
2. Select **+ Create a resource** and search for *Language service*.
3. Select **Create** under **Language Service**.
4. Select the option that includes **Custom text classification and Custom named entity recognition**.
5. Select **Continue to create your resource**.
6. In **Resource group**, choose **cog-search-language-exe**.
7. In **Region**, select the region you used above.
8. In **Name**, enter **learn-language-service-for-custom-text**. This needs to be unique globally, so you might need to add a random numbers or characters at the end of it.
9. In **Pricing tier**, select **S**.
10. In **New/Existing storage account**, select **Existing storage account**.
11. In **Storage account in current selected subscription and resource region**, select the storage account you created, for example **acs18245str**.
12. Agree to the **Responsible AI Notice** terms, then select **Review + create**.
13. Select **Create**.
14. Wait for the resources to be deployed, then select **Go to resource group**.
15. Select **learn-language-service-for-custom-text**.



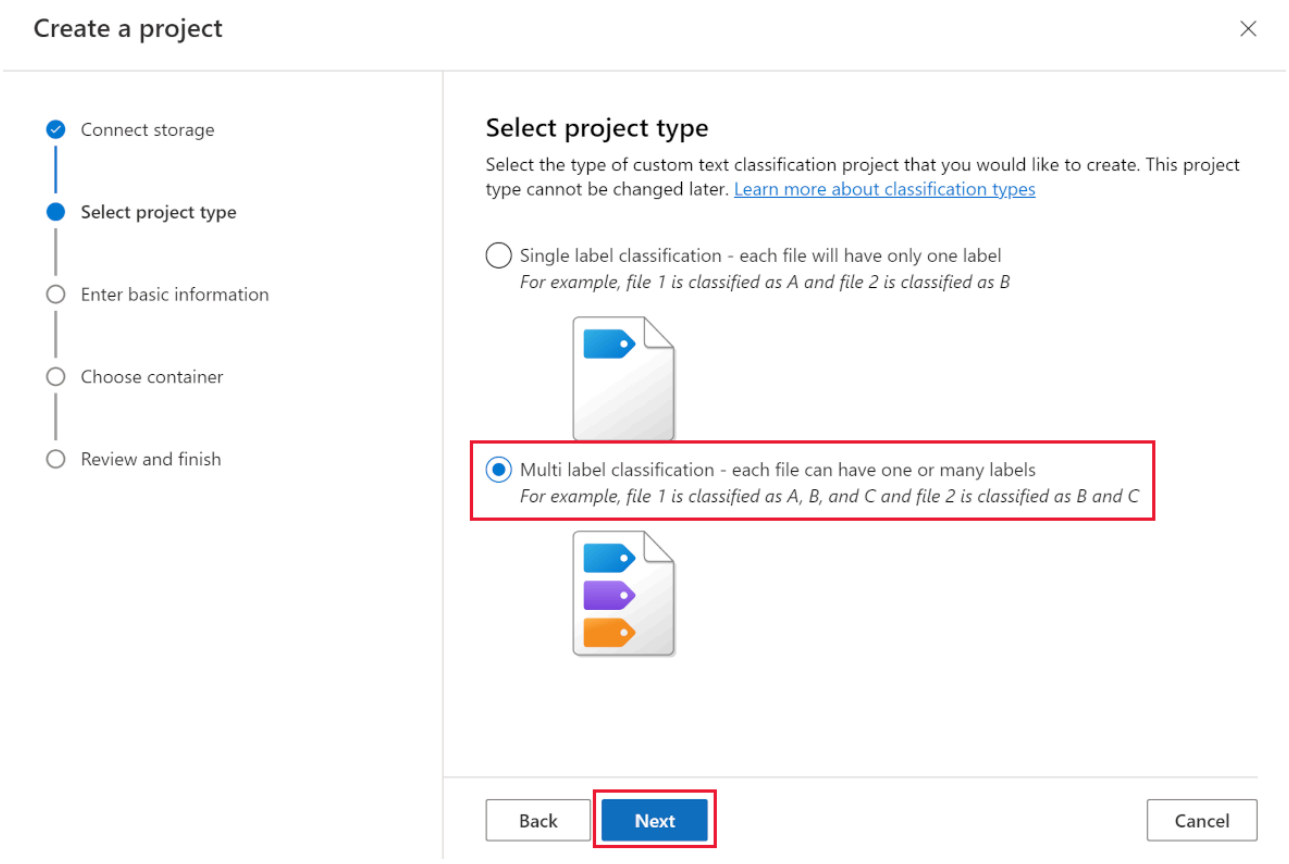
16. Scroll down on the **Overview** pane, and select **Get started with Language Studio**.
17. Sign in the language studio. If you are prompted to choose a Language resource select the resource you created earlier.

Create a custom text classification project in Language Studio

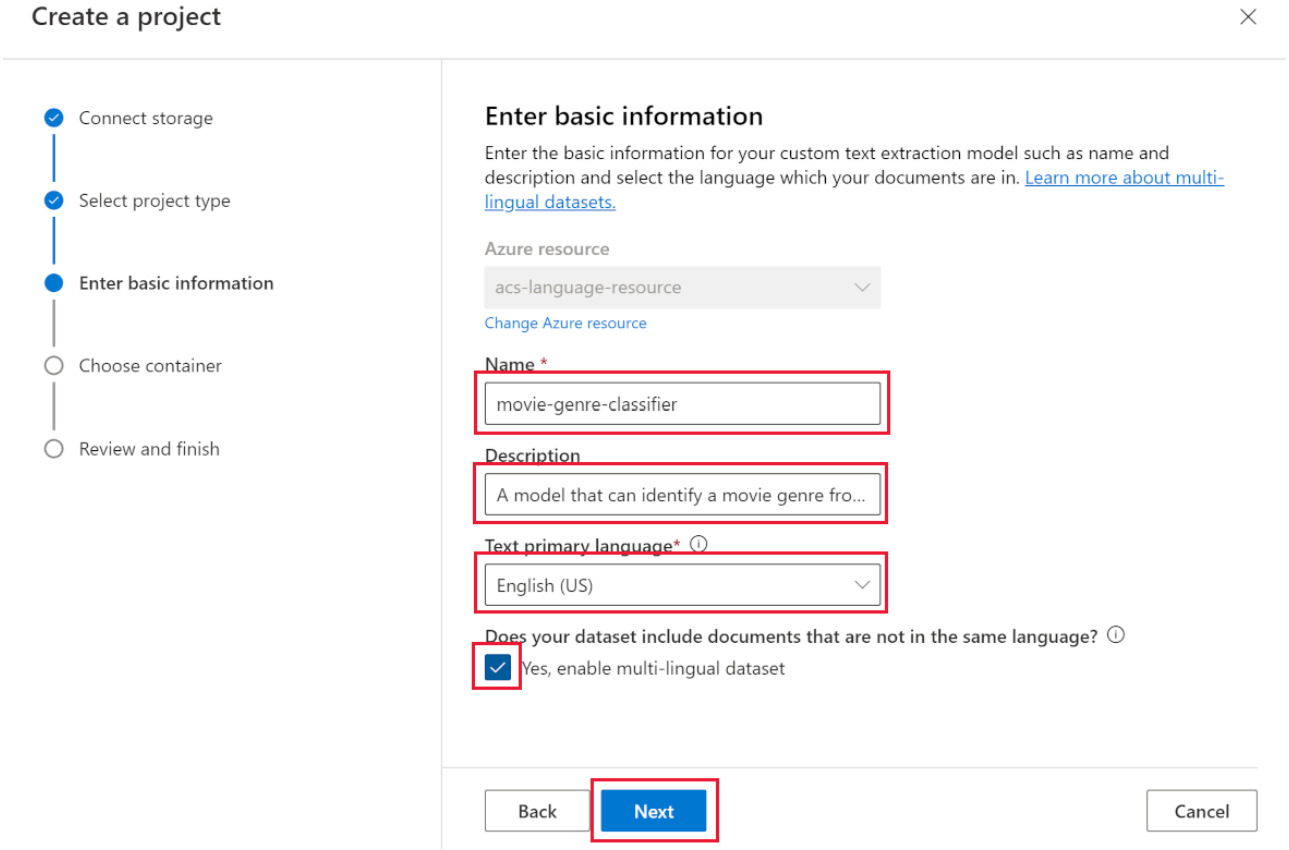
1. On the Language Studio home page, select **Create new**, then select **Custom text classification**.



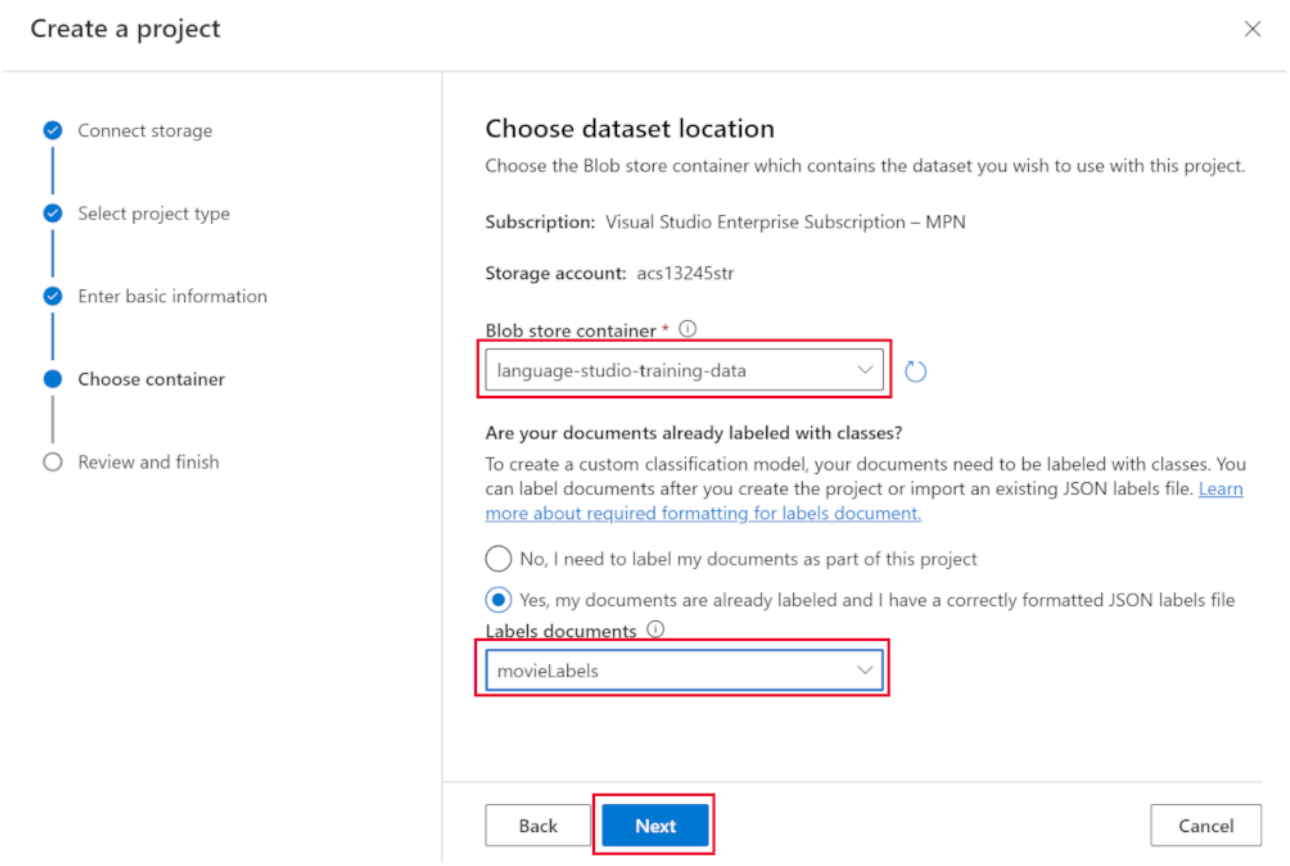
2. Select **Next**.



3. Select **Multi label classification**, then select **Next**.



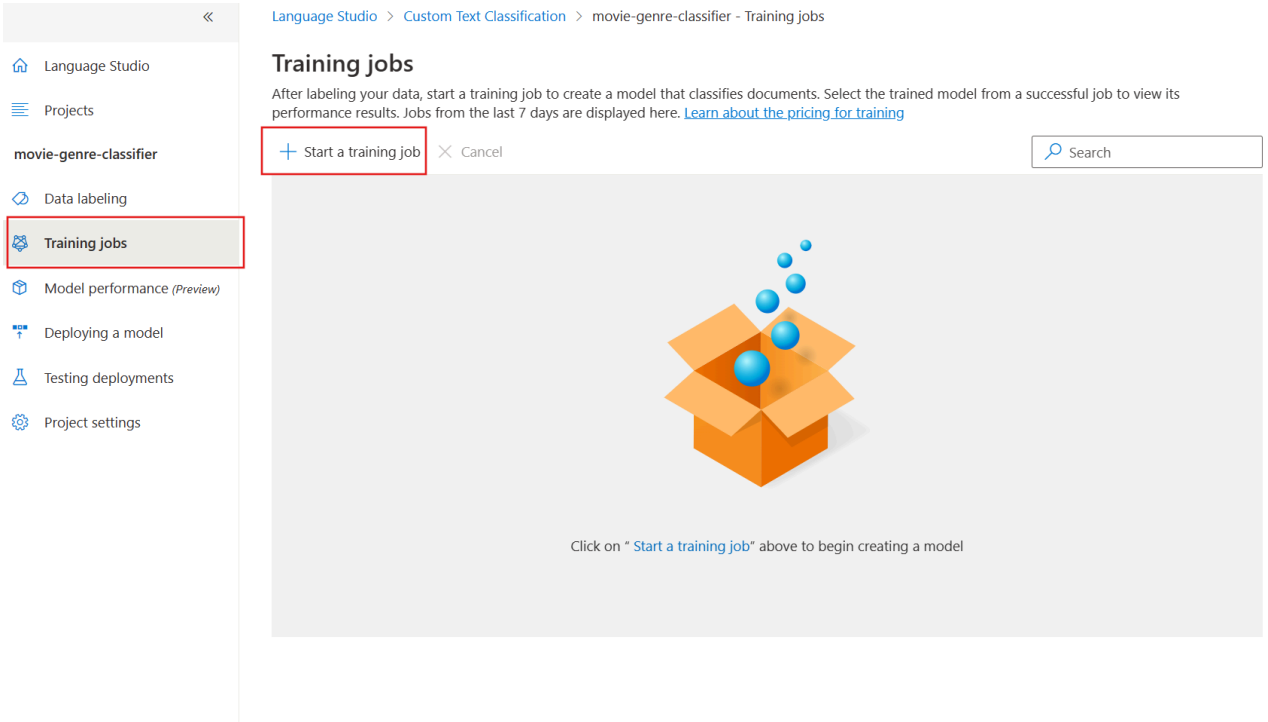
4. In **Name**, enter **movie-genre-classifier**.
5. In **Text primary language**, select **English (US)**.
6. In **Description**, enter **A model that can identify a movie genre from the summary**.
7. Select **Yes, enable multi-lingual dataset**.
8. Select **Next**.



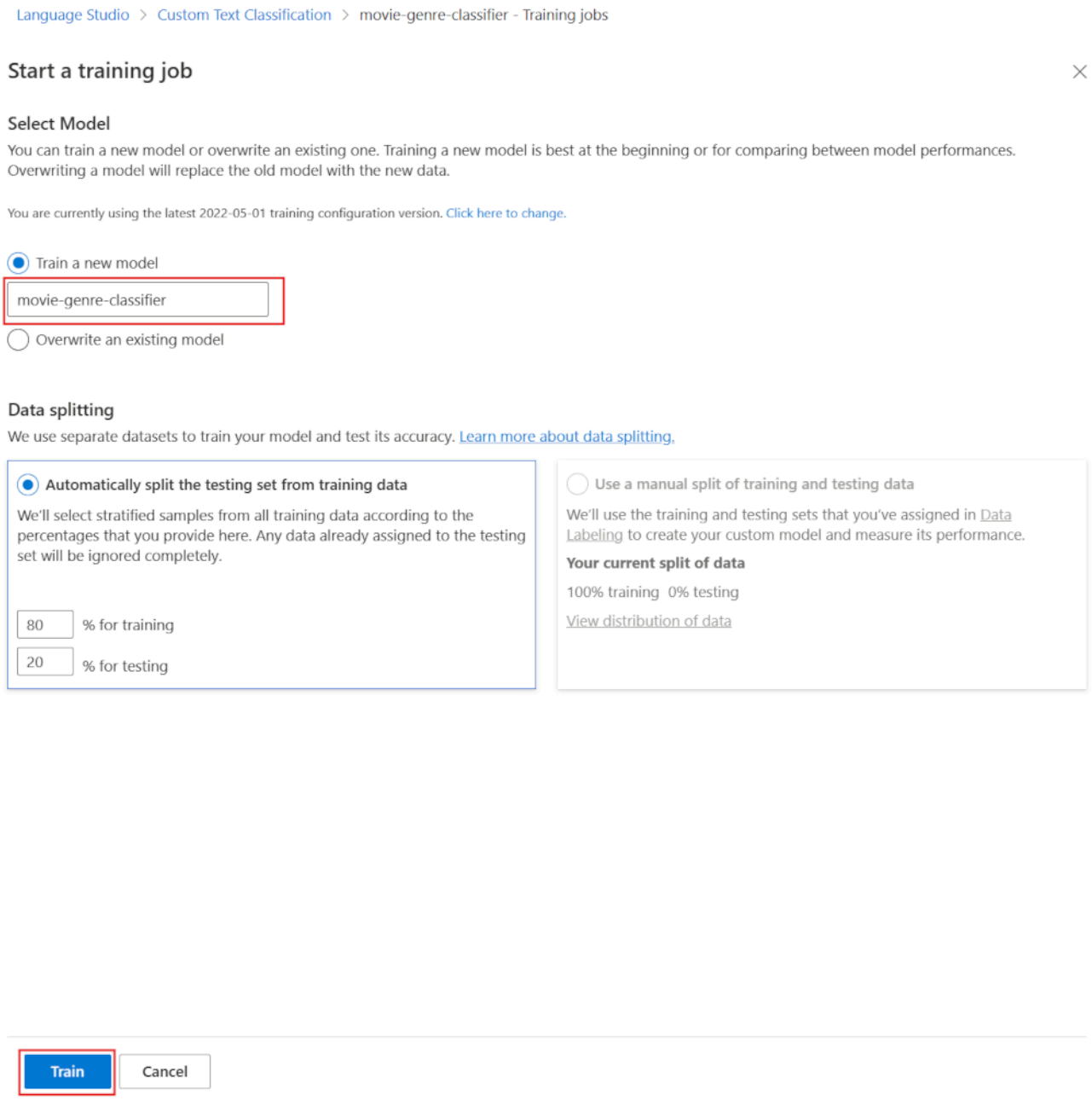
9. In **Blob storage container**, choose **language-studio-training-data**.
10. Select **Yes, my documents are already labeled and I have a correctly formatted JSON labels file**.
11. In **Label documents**, choose **movieLabels**.
12. Select **Next**.
13. Select **Create project**.

Train your custom text classification AI model

1. On the left, select **Training jobs**.



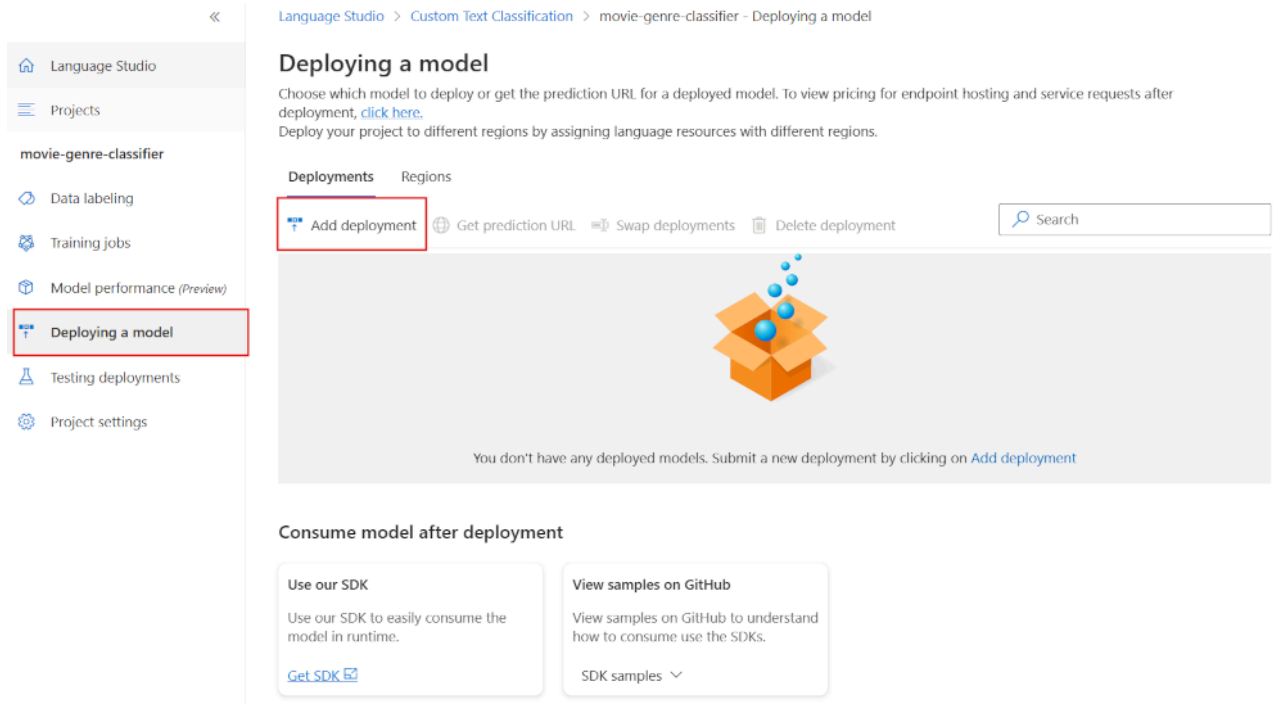
2. Select **+ Start a training job**.



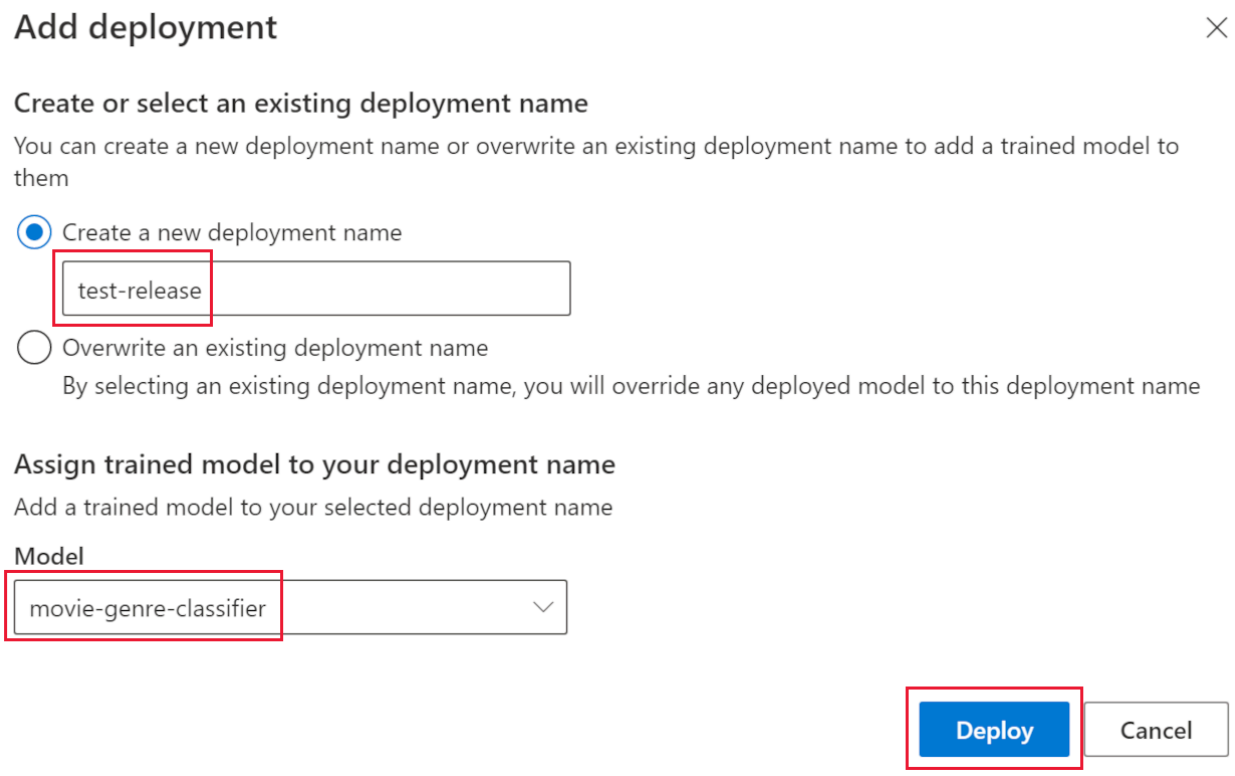
3. In **Train a new modal**, enter **movie-genre-classifier**.
4. Select **Train**.
5. Training the classifier model should take less than 10 minutes. Wait for the status to change to **Training succeeded**.

Deploy your custom text classification AI model

1. On the left, select **Deploying a model**.



2. Select **Add a deployment**.



3. In **Create a new deployment name**, enter **test-release**.
4. In **Model**, select **movie-genre-classifier**.
5. Select **Deploy**.

Leave this web page open for later in this exercise.

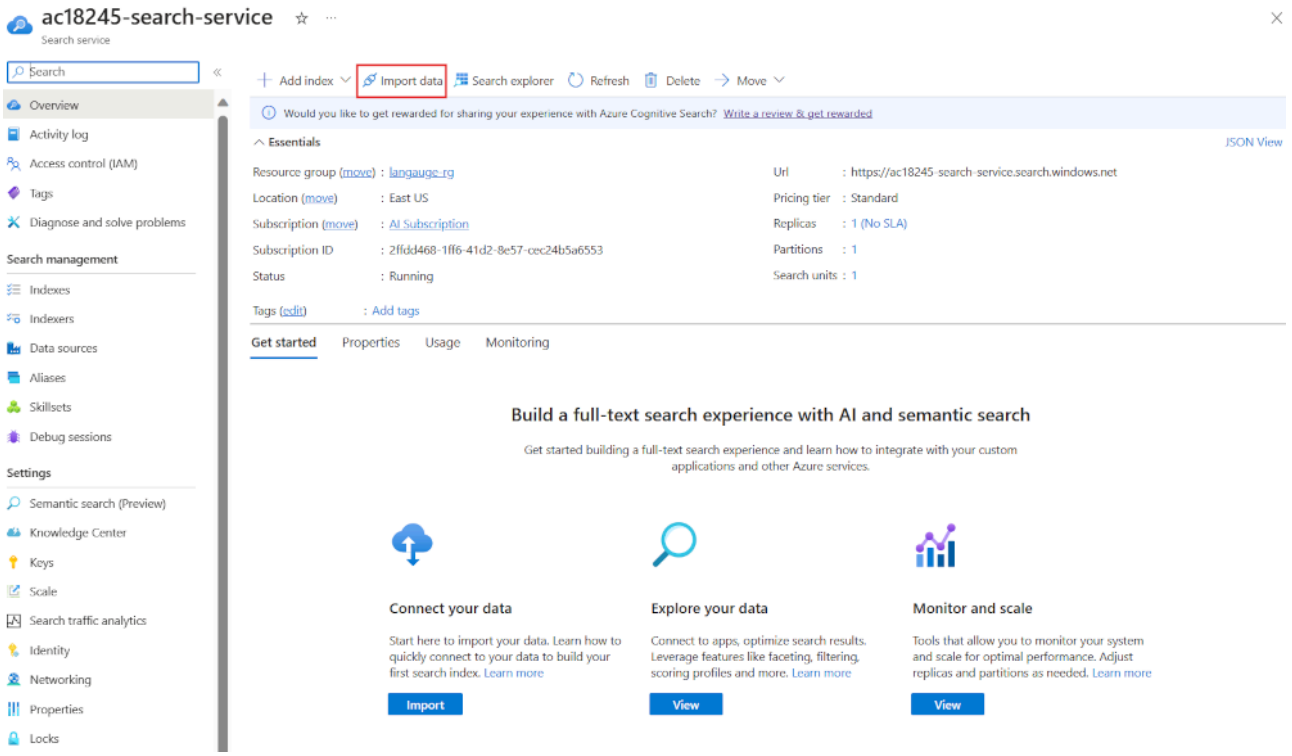
Create an Azure AI Search index

Create a search index that you can enrich with this model, you'll index all the text files that contain the movie summaries you've already downloaded.

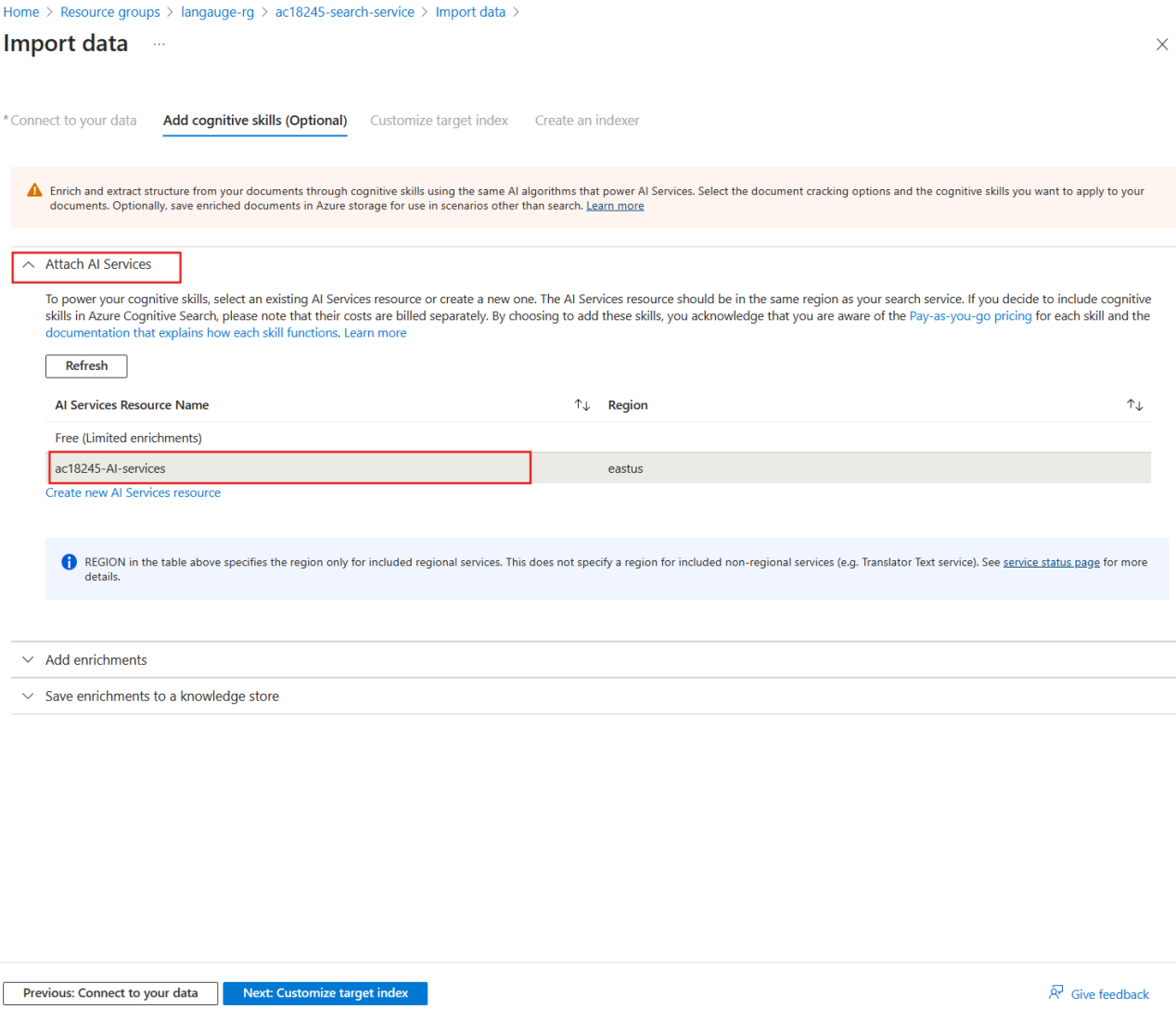
1. In the Azure portal, select **Resource groups**, select your resource group, then select the storage account you created, for example **acs18245str**.
2. Select **Containers** from the left, then select **+ Container**.
3. In the **New container** pane, in **Name**, enter **search-data**.
4. In **Anonymous access level**, choose **Container**.
5. Select **Create**.
6. Select the new container you just created, **search-data**.
7. Select **Upload** at the top of the pane.
8. In the **Upload blob** pane, select **Browse for files**.
9. Navigate to where you downloaded the sample files, select **ONLY** the text (`.txt`) files.
10. Select **Upload** in the pane.
11. Close the **Upload blob** pane.

Import documents into Azure AI Search

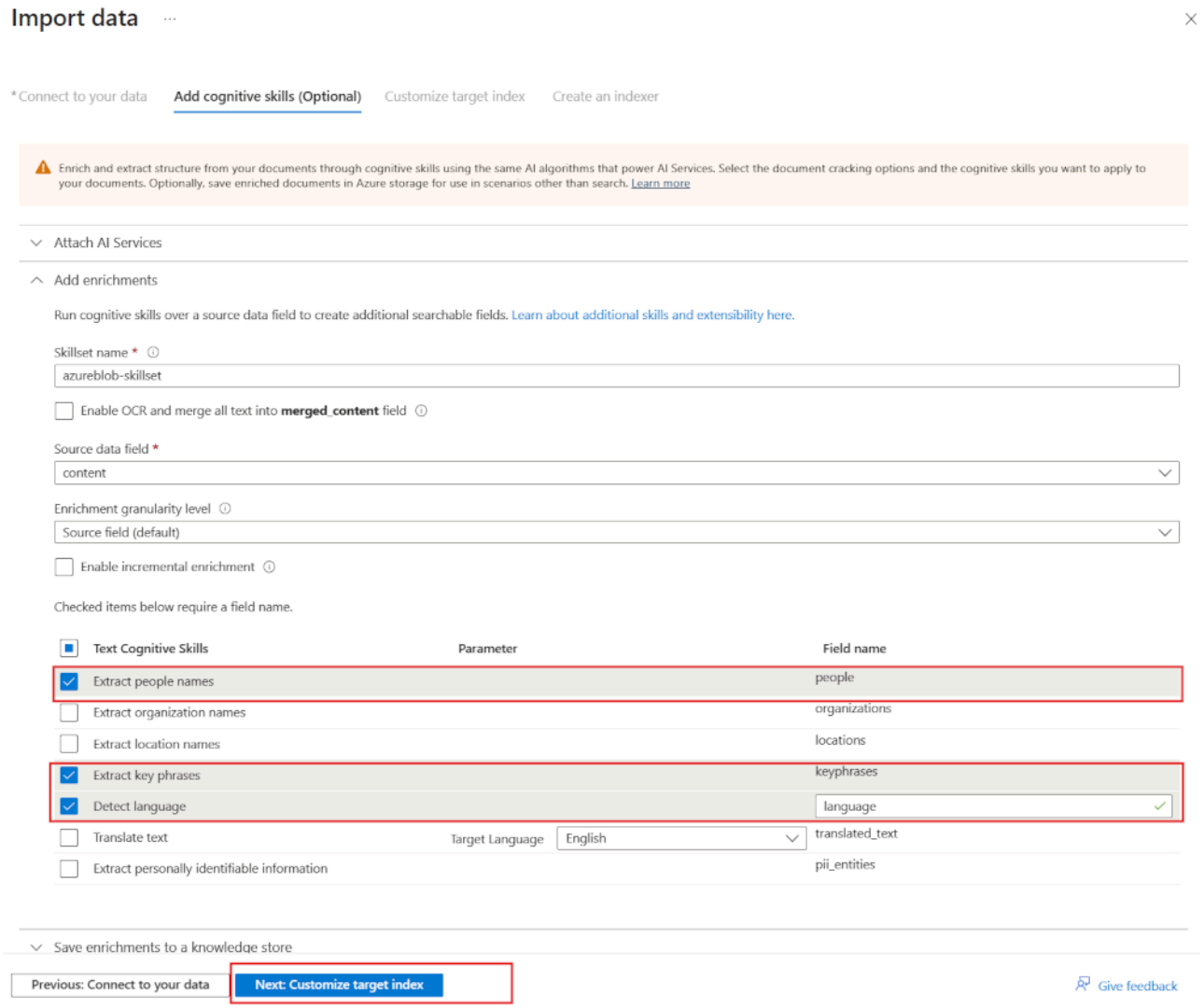
1. On the left, select **Resource groups**, select your resource group, then select your search service.
2. Select **Import data**.



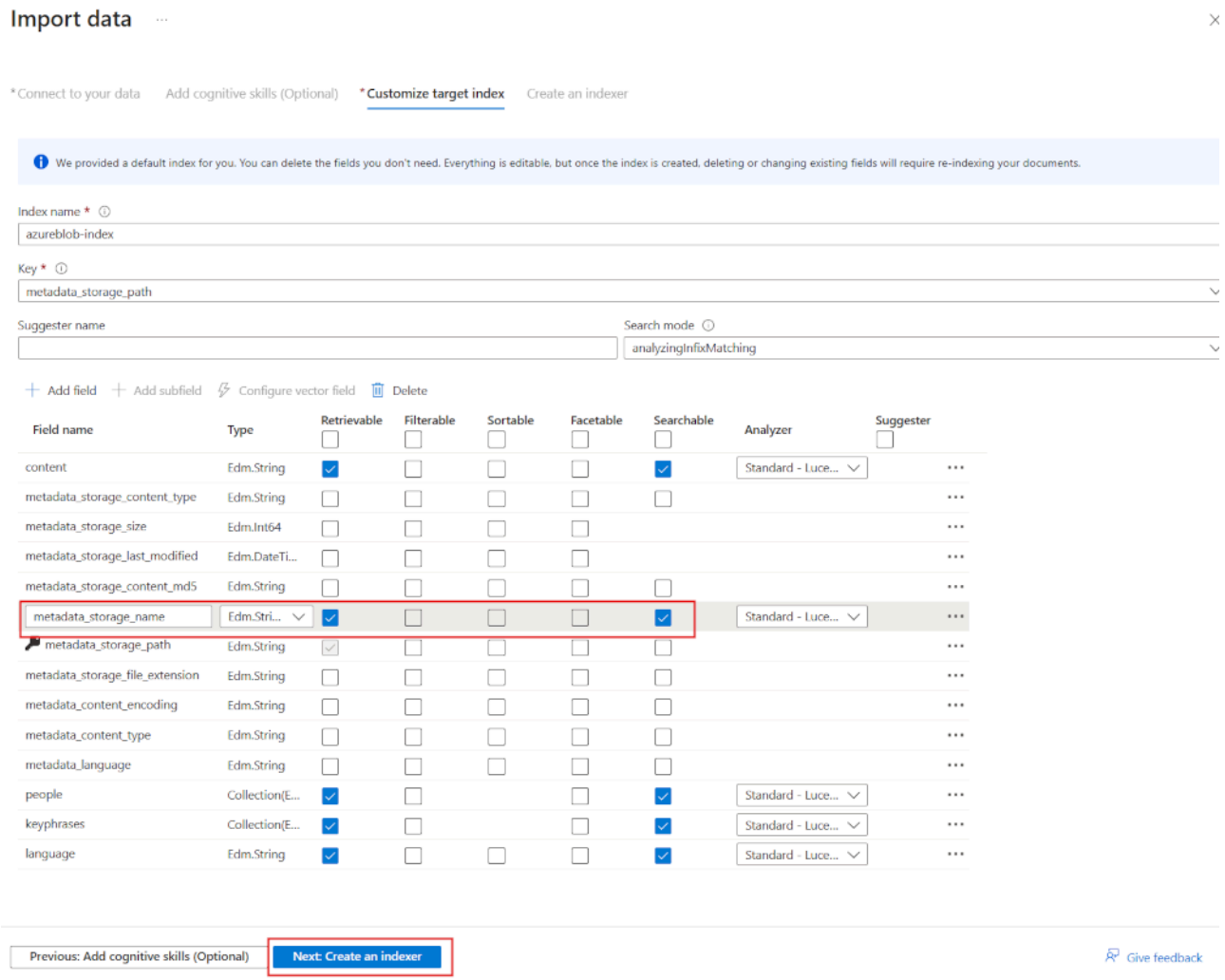
3. In **Data Source**, select **Azure Blob Storage**.
4. In **Data source name**, enter **movie-summaries**.
5. Select **Choose an existing connection**, select your storage account, then select the container you just created, **search-data**.
6. Select **Add cognitive skills (optional)**.
7. Expand the **Attach AI Services** section, then select the Azure AI service you created earlier.



8. Expand the **Add enrichments** section.



9. Leave all the fields with their default values, then select **Extract people names**.
10. Select **Extract key phrases**.
11. Select **Detect language**.
12. Select **Next: Customize target index**.



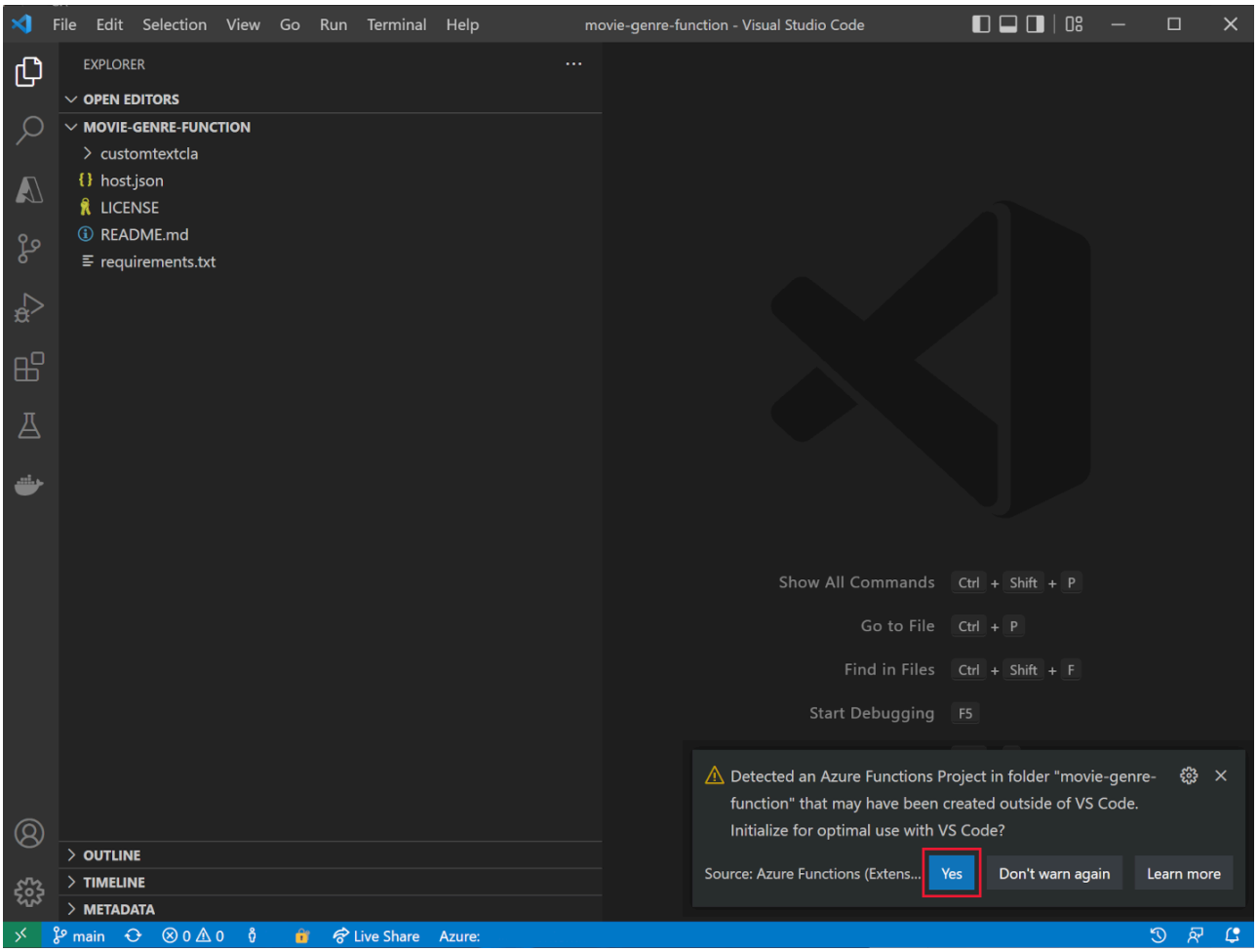
13. Leave all the fields with their default values,
    for **metadata_storage_name** select **Retrievable** and **Searchable**.
14. Select **Next: Create an indexer**.
15. Select **Submit**.

The indexer will run and create an index of the 210 text files. You don't need to wait for it to continue with the next steps.

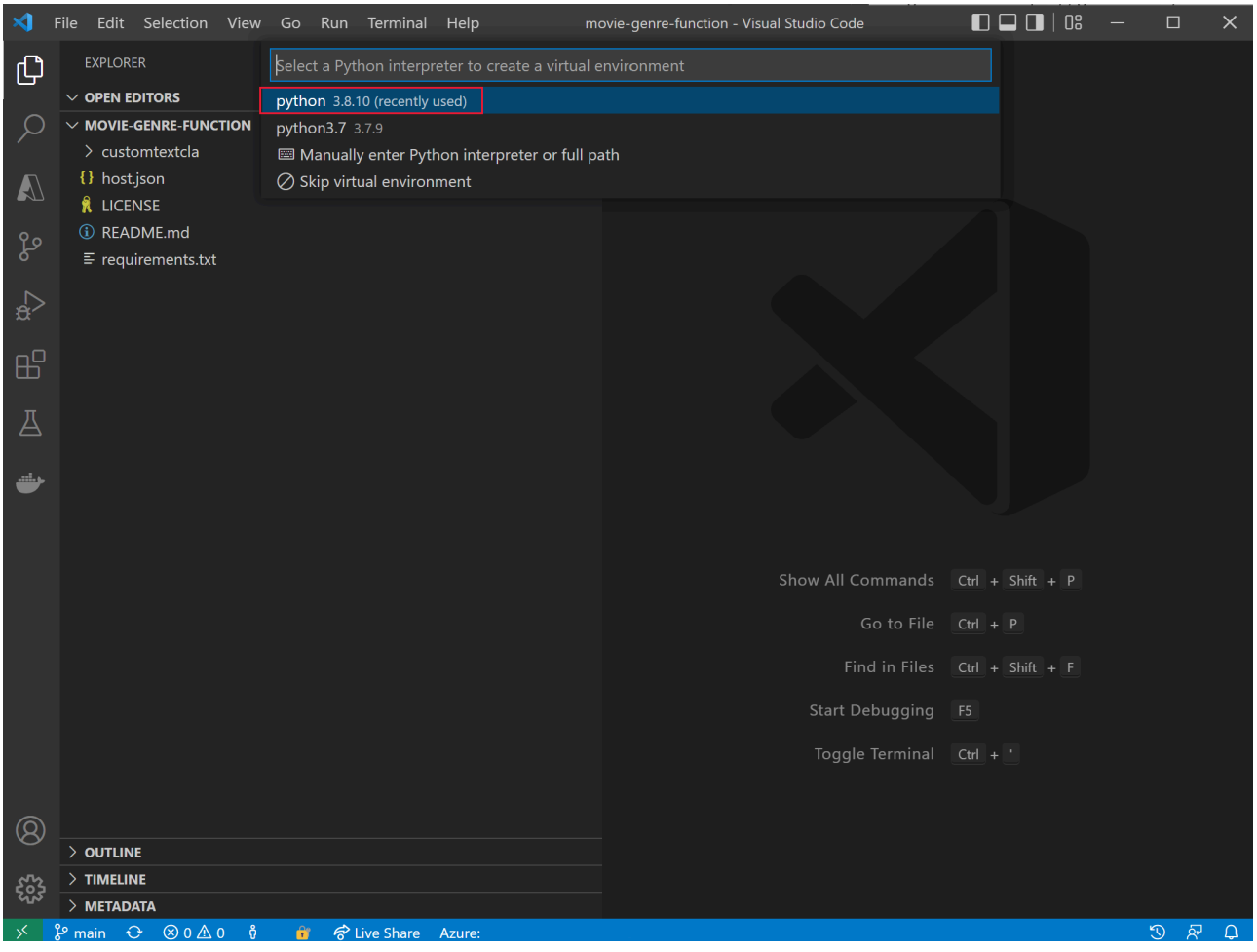
Create a function app to enrich your search index

You'll now create a Python function app that your cognitive search custom skillset will call. The function app will use your custom text classifier model to enrich your search index.

1. Download required files and extract the folder containing all the files.
2. Open Visual Studio Code, open the **movie-genre-function** folder you've just downloaded.

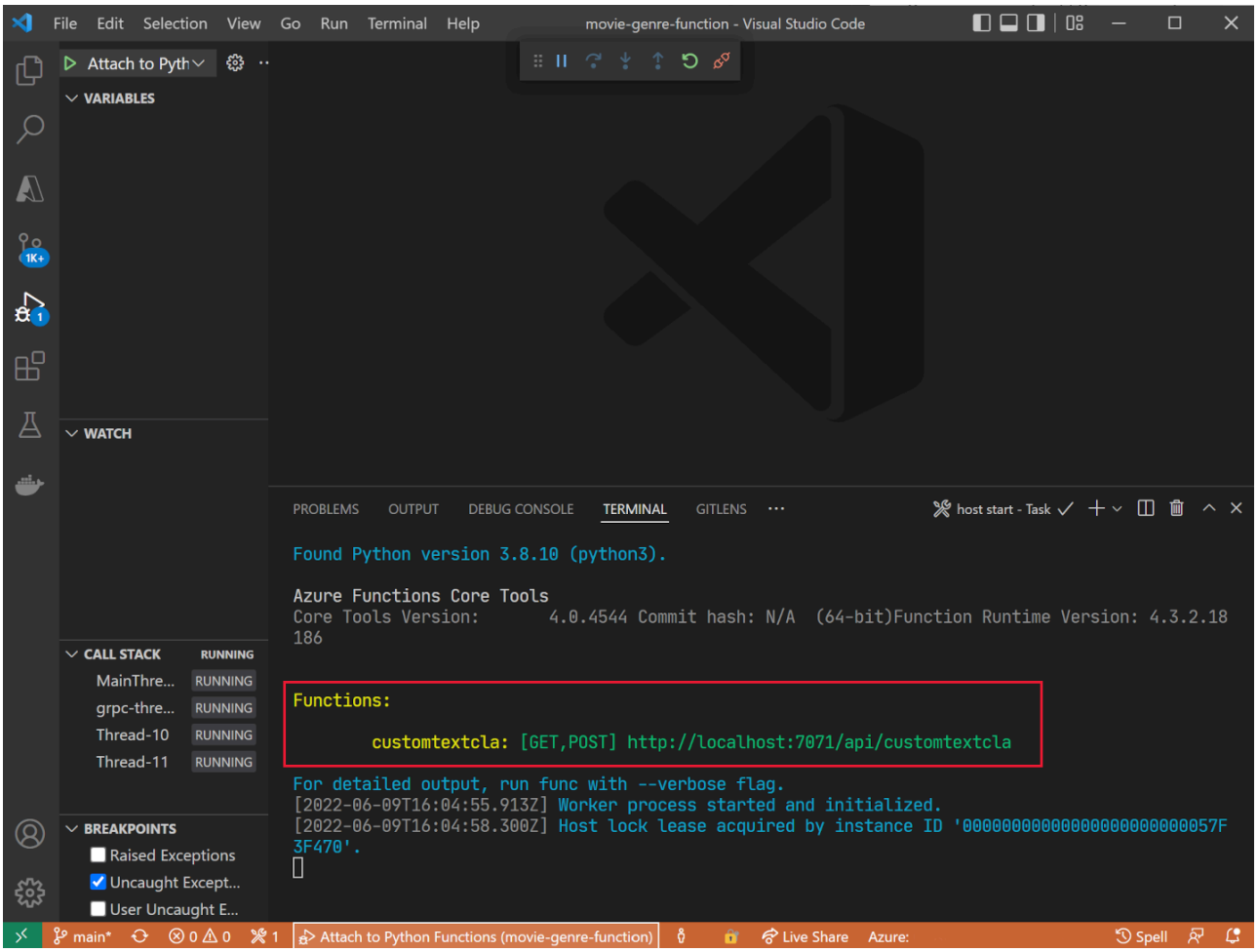


3. If you've installed all the required extensions, you're prompted to optimize the project. Select **Yes**.



4. Select your Python interpreter, it should be version 3.8.
5. The workspace will be updated, if you're asked to connect it to the workspace folder, select **Yes**

6. Press **F5** to debug the app.



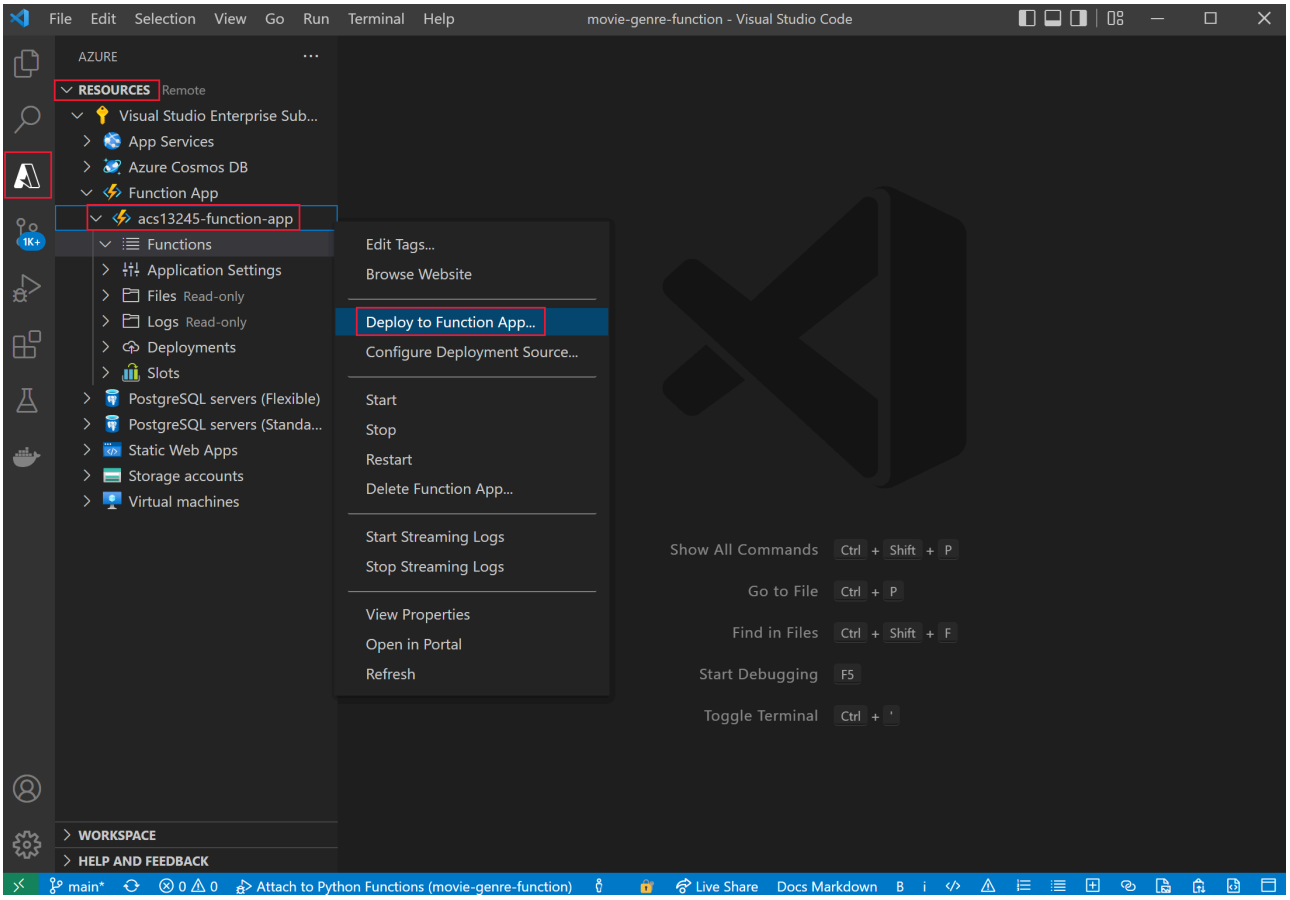
If the app is running you should see a localhost URL that you can use for local testing.

7. Stop debugging the app, press **SHIFT** + **F5**.

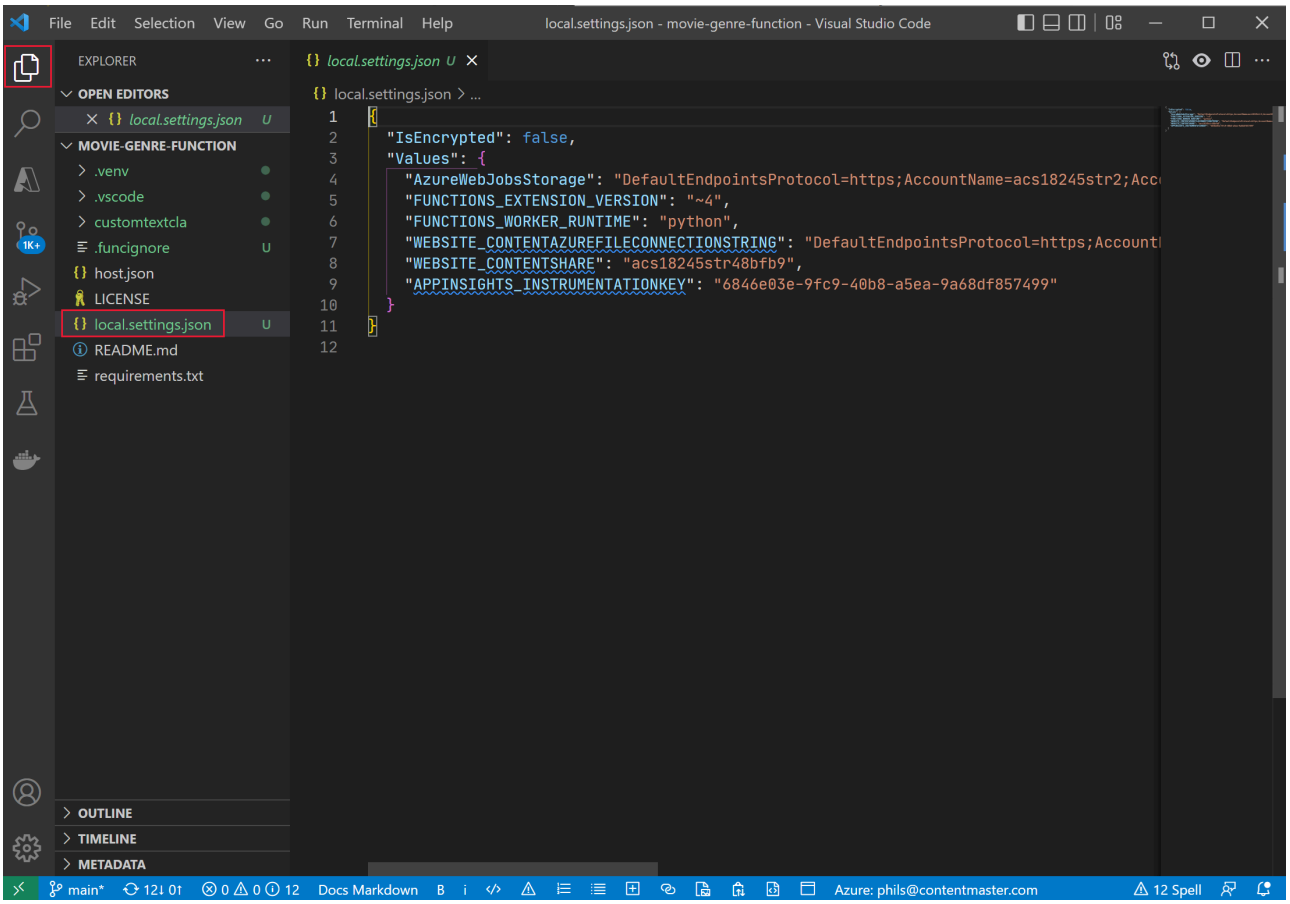
Deploy your local function app to Azure

1. In Visual Studio Code, press **F1** to open the command palette.
2. In the command palette, search for and select `Azure Functions: Create Function App in Azure...`.
3. Enter a globally unique name for your function app, for example **acs13245str-function-app**.
4. In **Select a runtime stack**, select **Python 3.8**.
5. Select the same location you used above.

6. In the left navigation, select the **Azure** extension.

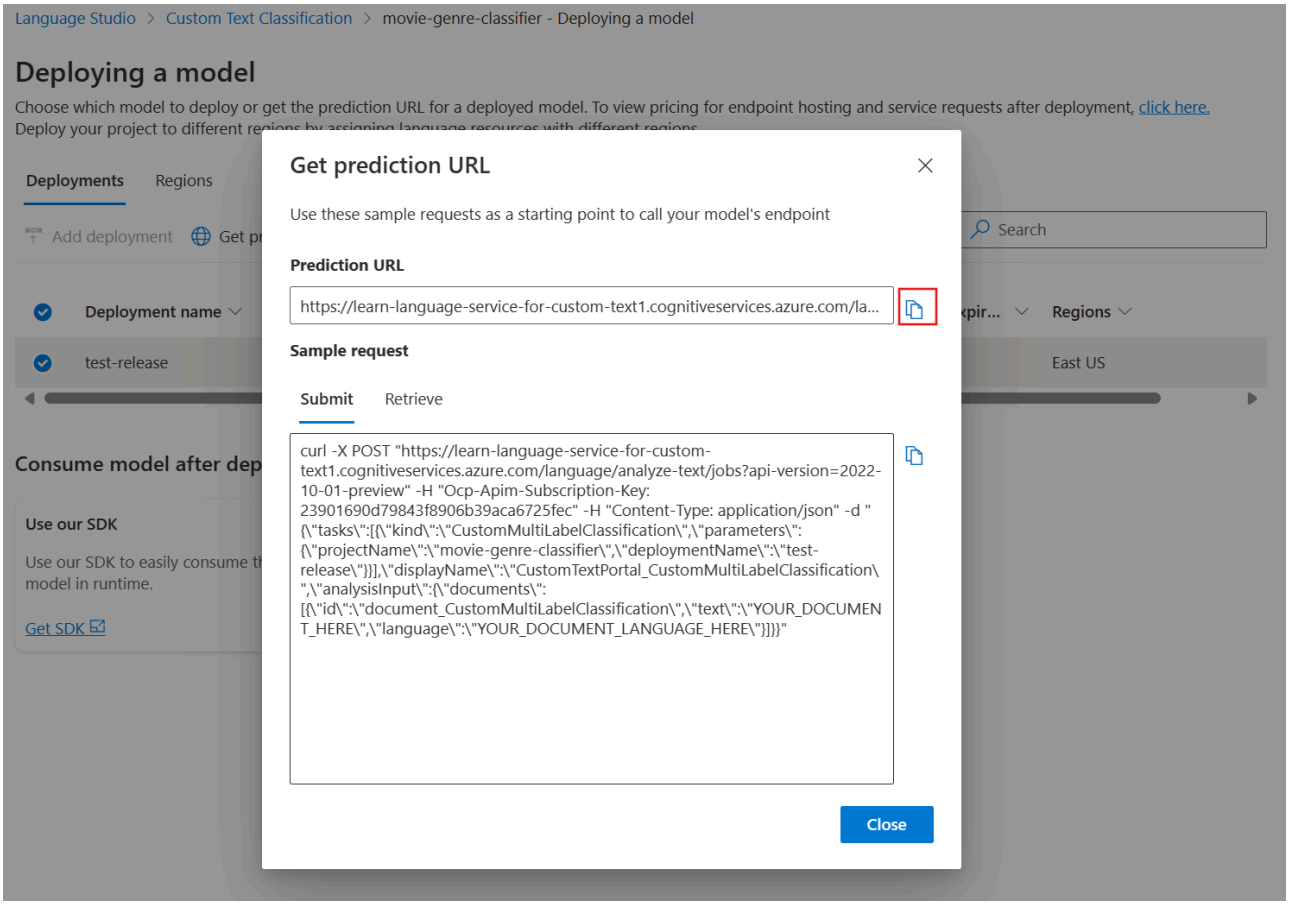


7. Expand **Resources**, expand **Function App** under your subscription, then right-click on the function, for example **acs13245-function-app**.
8. Select **Deploy to Function App**. Wait for the app to be deployed.
9. Expand the app, right-click on **Application Settings**, select **Download Remote Settings**.
10. On the left, select **Explorer**, then select **local.settings.json**.



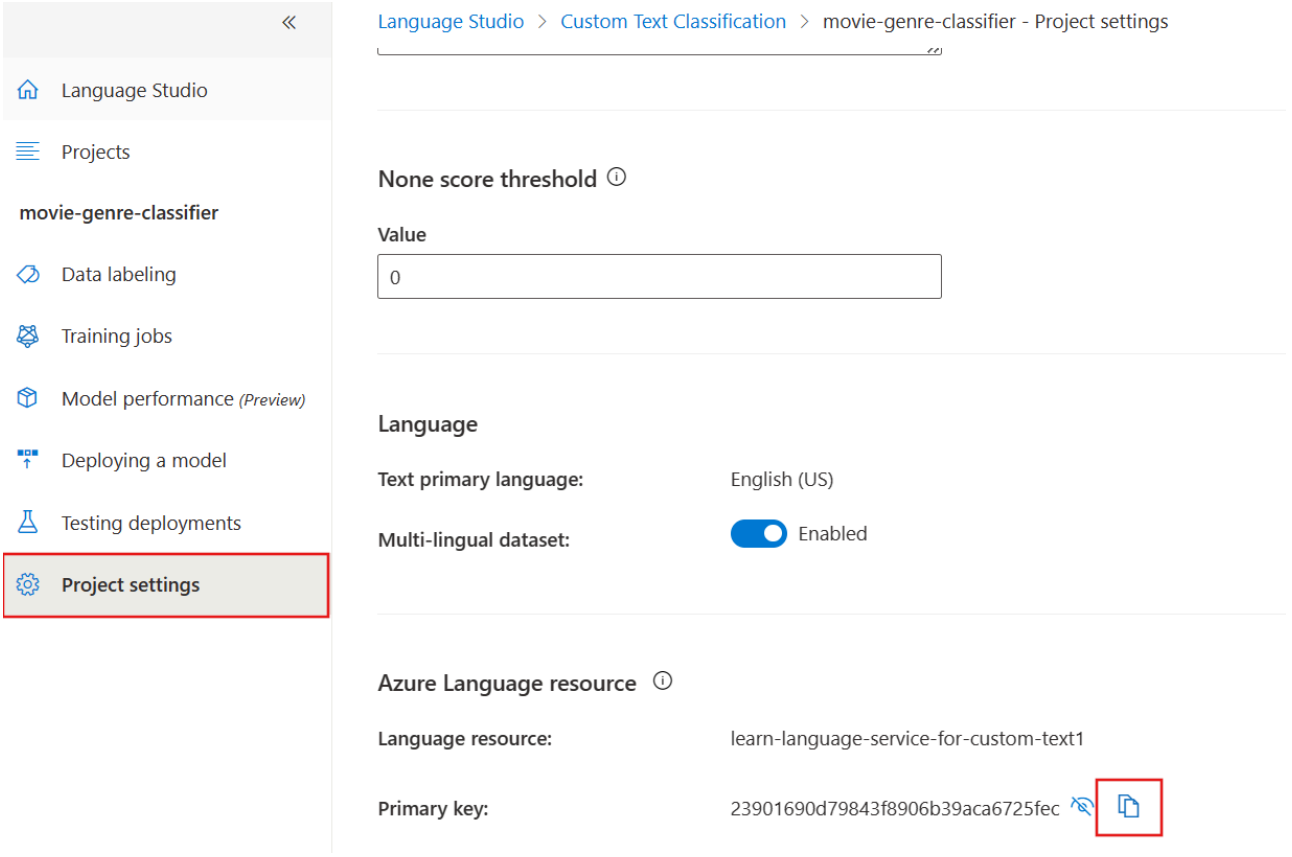
The function app needs to be connected to your custom text classification model. Follow these steps to get the configuration settings.

11. In your browser, navigate to **Language Studio**, you should be on the **Deploying a model** page.



12. Select your model. Then select **Get prediction URL**.
13. Select the copy icon next to the **Prediction URL**.

14. In Visual Studio Code, at the bottom of **local.settings.json**, paste the prediction URL.
15. In **Language Studio**, on the left, select **Project settings**.



16. Select the copy icon next to the **Primary key**.
17. In Visual Studio Code, at the bottom of **local.settings.json**, paste the primary key.
18. Edit the settings to add these four lines at the bottom, copy the endpoint into the `TA_ENDPOINT` value.

```
,
"TA_ENDPOINT": " [your endpoint] ",
"TA_KEY": " [your key] ",
"DEPLOYMENT": "test-release",
"PROJECT_NAME": "movie-genre-classifier"
```

19. Copy the primary key into the `TA_KEY` value.

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=...",
    "FUNCTIONS_EXTENSION_VERSION": "~4",
    "FUNCTIONS_WORKER_RUNTIME": "python",
    "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING":
"DefaultEndpointsProtocol=https;AccountName=...",
    "WEBSITE_CONTENTSHARE": "acs...",
    "APPINSIGHTS_INSTRUMENTATIONKEY": "6846...",
    "TA_ENDPOINT": "https://learn-languages-service-for-custom-
text.cognitiveservices.azure.com/language/analyze-text/jobs?api-version=2022-05-01",
    "TA_KEY": "7105e938ce1...",
    "DEPLOYMENT": "test-release",
    "PROJECT_NAME": "movie-genre-classifier"
  }
```

```
}
```

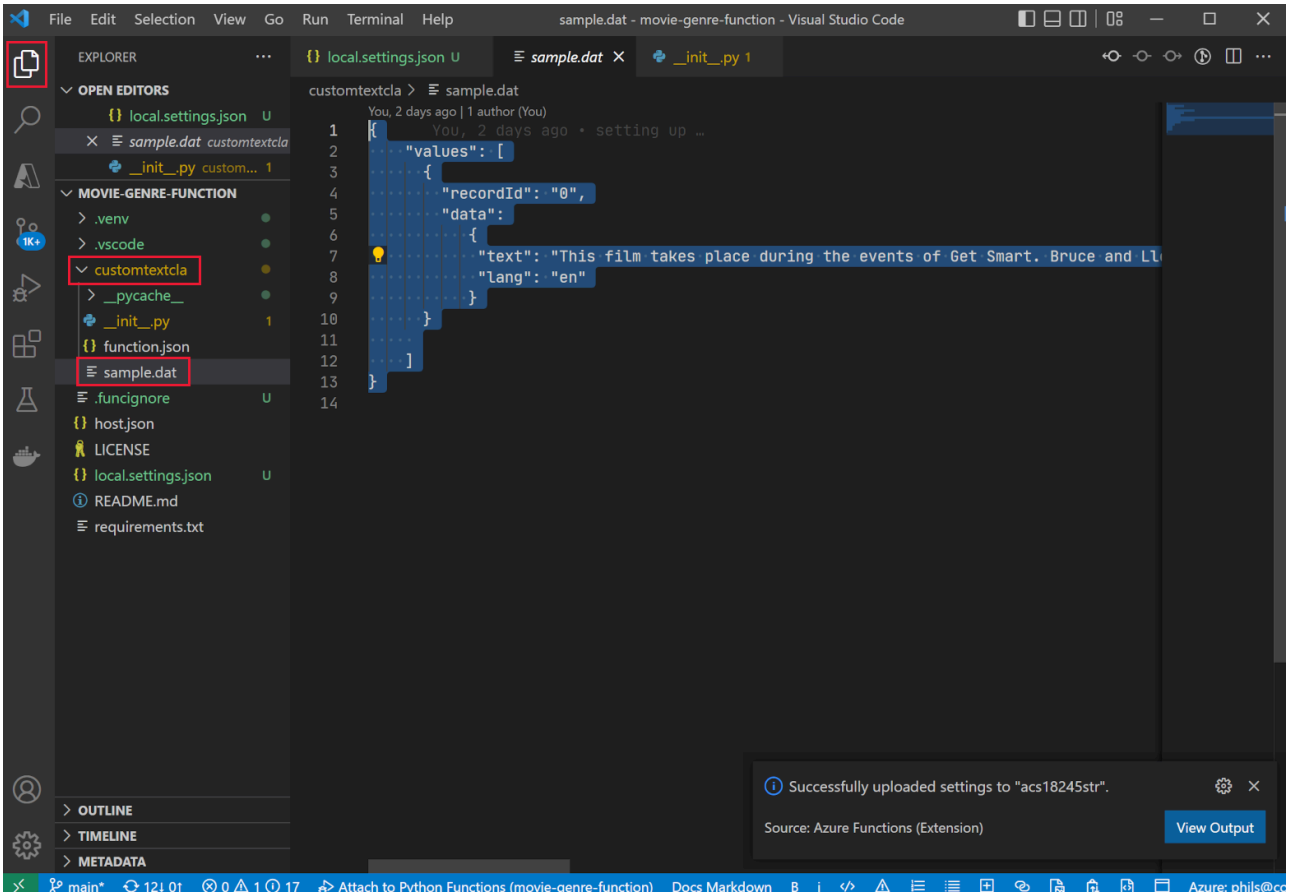
The settings should look like above, with the values of your project.

20. Press **CTRL+S** to save your **local.settings.json** changes.
21. In the left navigation, select the **Azure** extension.
22. Expand **Resources**, and under your subscription, expand **Function App**, then right-click on **Application Settings**, select **Upload Local Settings**.

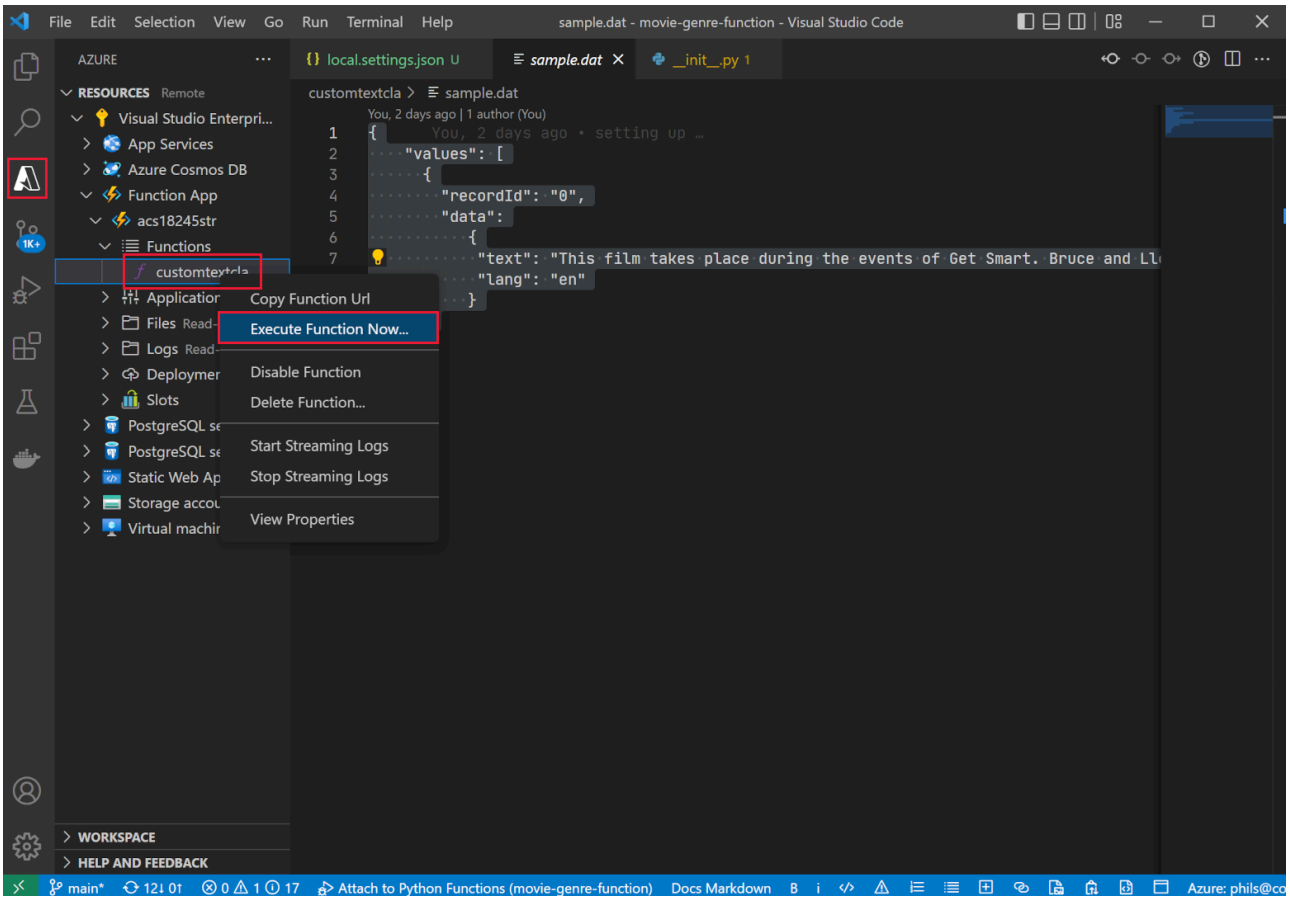
Test your remote function app

There's a sample query you can use to test that your function app and classifier model are working correctly.

1. On the left, select **Explorer**, expand the **customtextcla** folder, then select **sample.dat**.

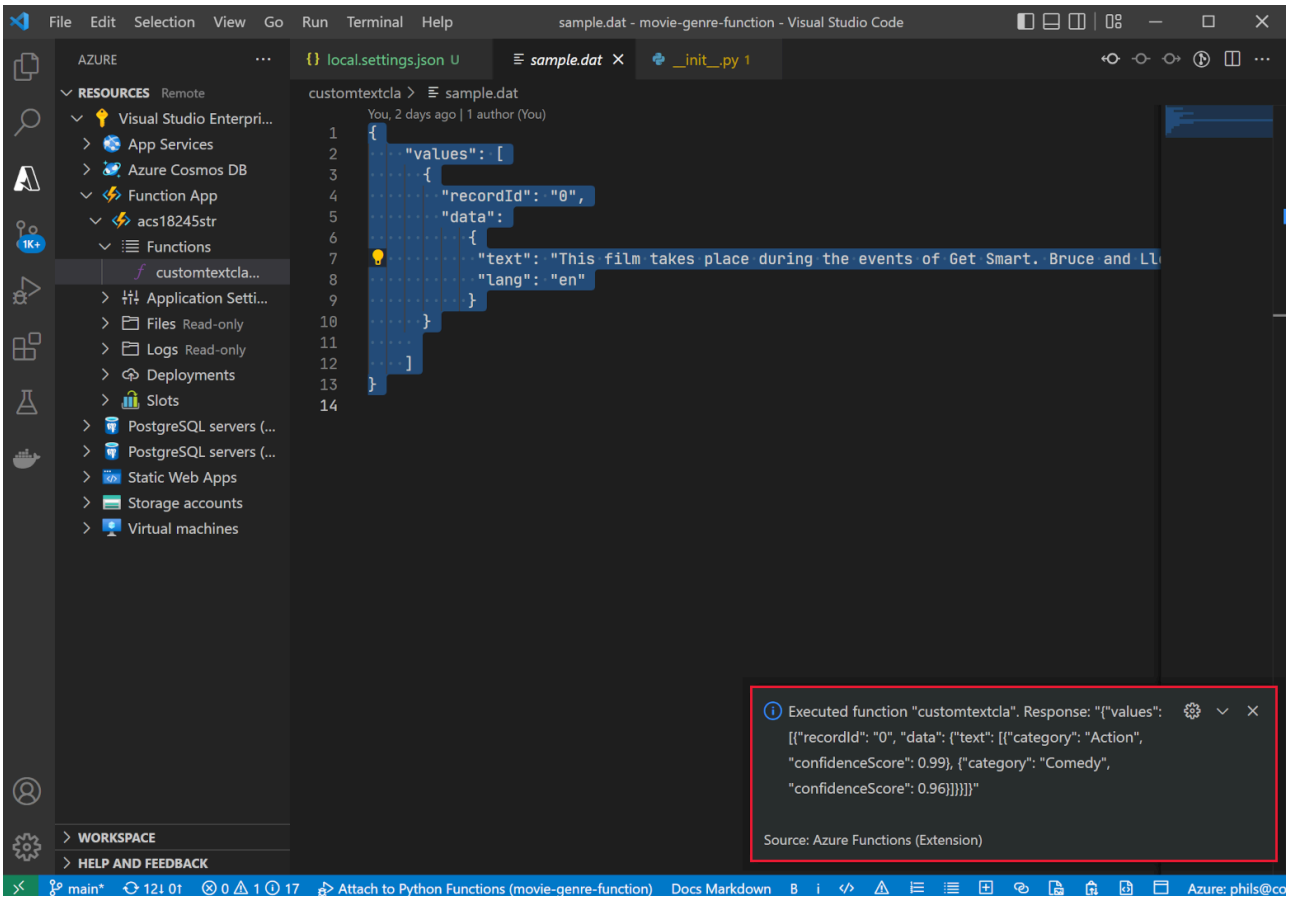


2. Copy the contents of the file.
3. On the left, select the **Azure** extension.



4. Under the **Function App**, expand **Functions**, right-click on **customtextcla**, then select **Execute Function now**.
5. In **Enter request body**, paste the sample data you copied, then press **Enter**.

   The function app will respond with JSON results.
6. Expand the notification to see the whole results.



The JSON response should look like this:

```
{"values":
    [
        {"recordId": "0",
        "data": {"text":
        [
            {"category": "Action", "confidenceScore": 0.99},
            {"category": "Comedy", "confidenceScore": 0.96}
        ]}}
    ]
}
```

Add a field to your search index

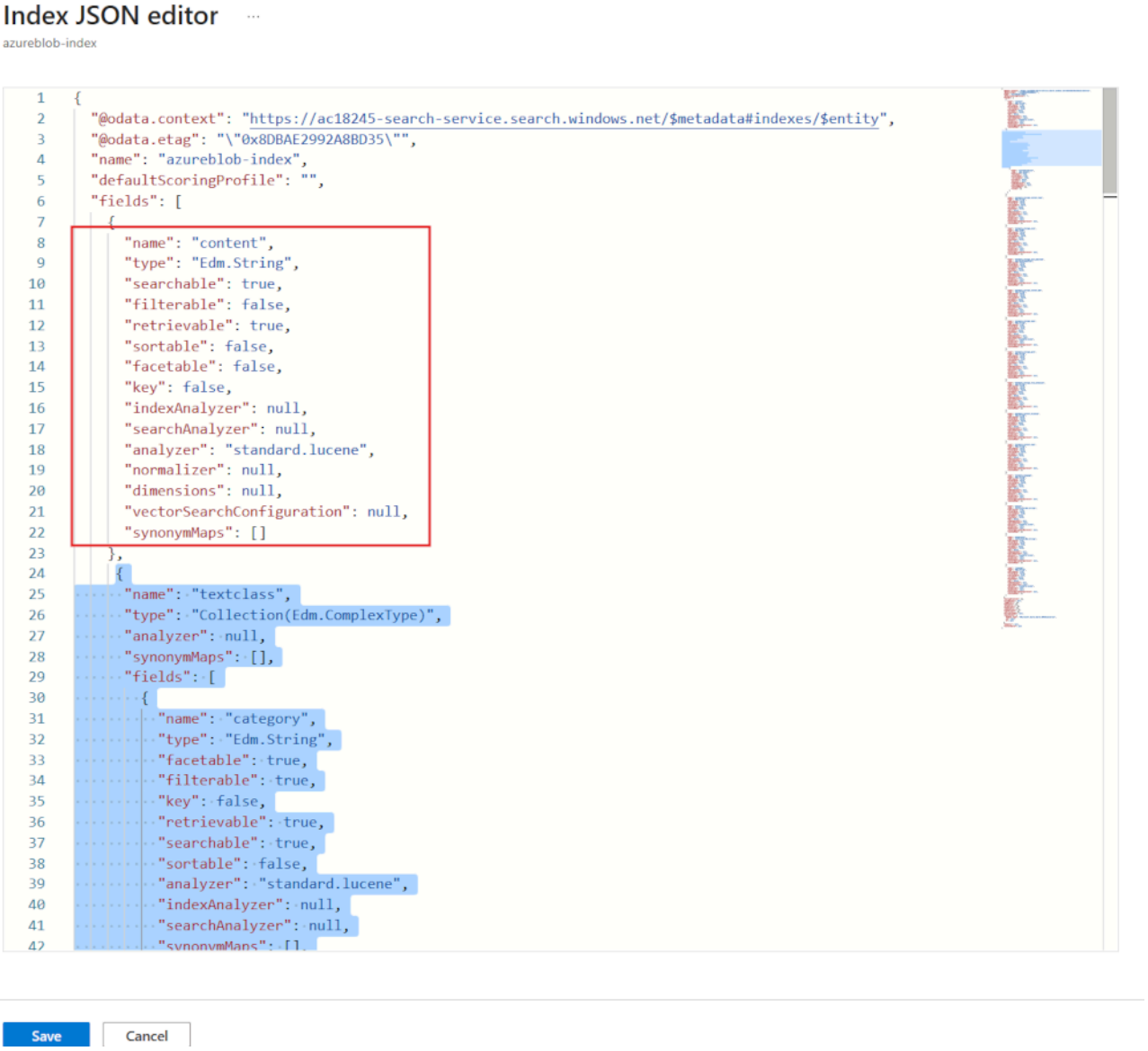
You need a place to store the enrichment returned by your new function app. Follow these steps to add a new compound field to store the text classification and confidence score.

1. In the Azure portal, go to the resource group that contains your search service, then select the cognitive search service you created, for example **acs18245-search-service**.
2. On the **Overview** pane, select **Indexes**.
3. Select **azurebob-index**.
4. Select **Edit JSON**.
5. Add the new fields to the index, paste the JSON below the content field.

```
{
  "name": "textclass",
  "type": "Collection(Edm.ComplexType)",
```

```
    "analyzer": null,
    "synonymMaps": [],
    "fields": [
      {
        "name": "category",
        "type": "Edm.String",
        "facetable": true,
        "filterable": true,
        "key": false,
        "retrievable": true,
        "searchable": true,
        "sortable": false,
        "analyzer": "standard.lucene",
        "indexAnalyzer": null,
        "searchAnalyzer": null,
        "synonymMaps": [],
        "fields": []
      },
      {
        "name": "confidenceScore",
        "type": "Edm.Double",
        "facetable": true,
        "filterable": true,
        "retrievable": true,
        "sortable": false,
        "analyzer": null,
        "indexAnalyzer": null,
        "searchAnalyzer": null,
        "synonymMaps": [],
        "fields": []
      }
    ]
  },
```

Your index should now look like this.

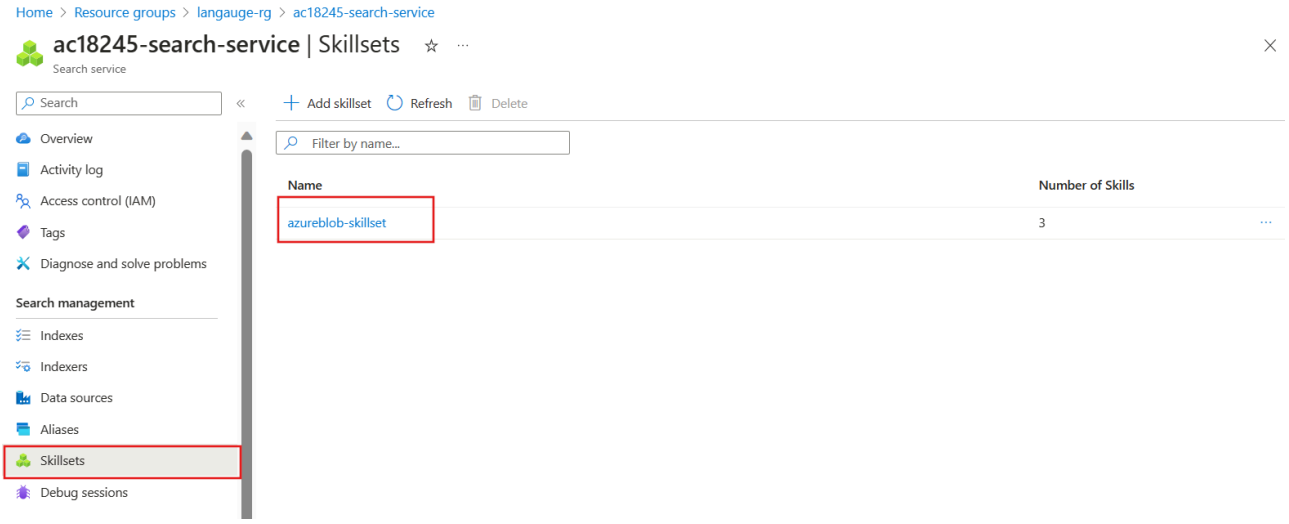


6. Select **Save**.

Edit the custom skillset to call your function app

The cognitive search index needs a way to have these new fields populated. Edit the skillset you created earlier to call your function app.

1. At the top of the page, select the search service link, for example **acs18245-search-service|Indexes**.
2. On the **Overview** pane, select **Skillsets**.

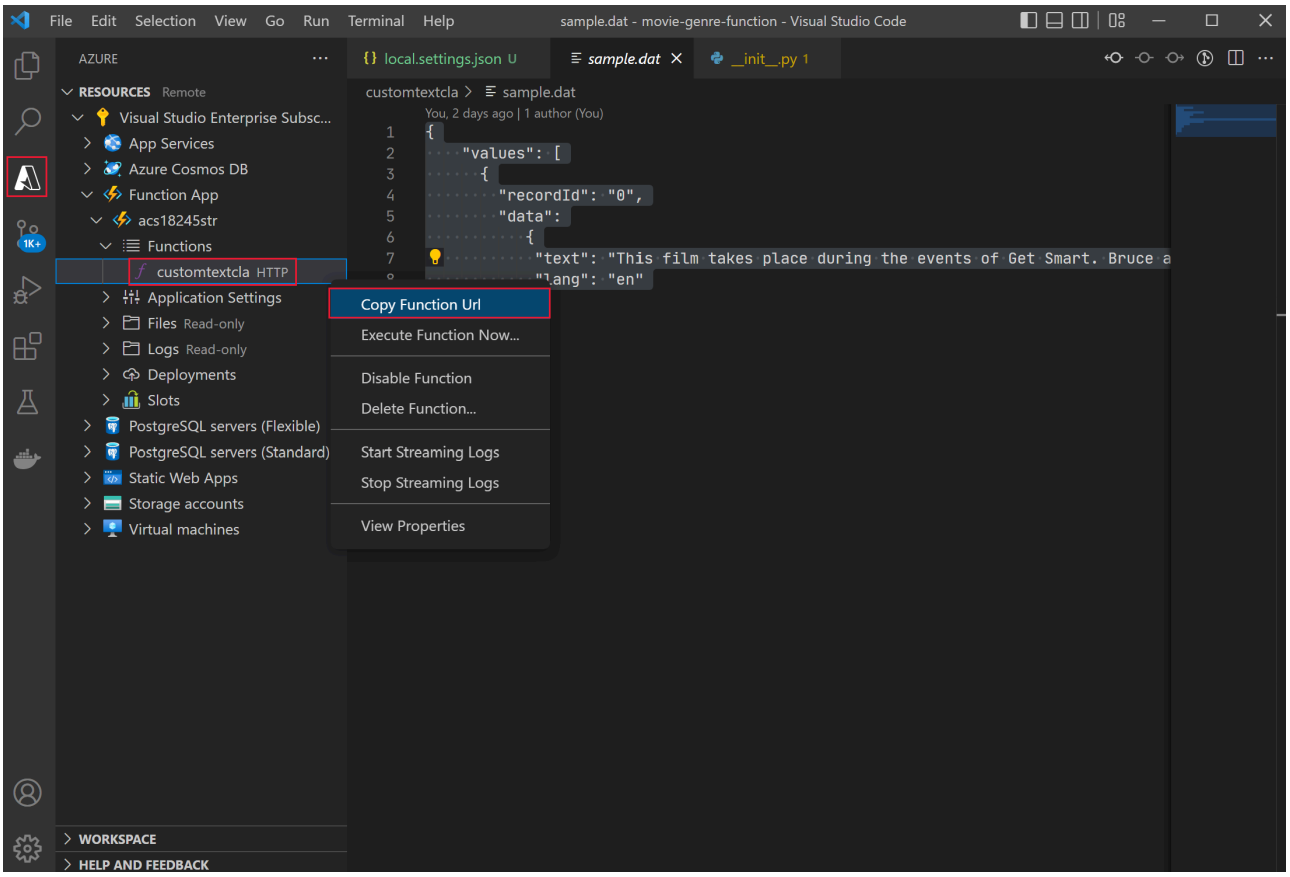


3. Select **azureblob-skillset**.
4. Add the custom skillset definition below, by pasting it as the first skillset.

```
{
  "@odata.type": "#Microsoft.Skills.Custom.WebApiSkill",
  "name": "Genre Classification",
  "description": "Identify the genre of your movie from its summary",
  "context": "/document",
  "uri": "URI",
  "httpMethod": "POST",
  "timeout": "PT30S",
  "batchSize": 1,
  "degreeOfParallelism": 1,
  "inputs": [
    {
      "name": "lang",
      "source": "/document/language"
    },
    {
      "name": "text",
      "source": "/document/content"
    }
  ],
  "outputs": [
    {
      "name": "text",
      "targetName": "class"
    }
  ],
  "httpHeaders": {}
},
```

You need to change the `"uri": "URI"` to point to your function app.

1. In Visual Studio Code, select the **Azure** extension.

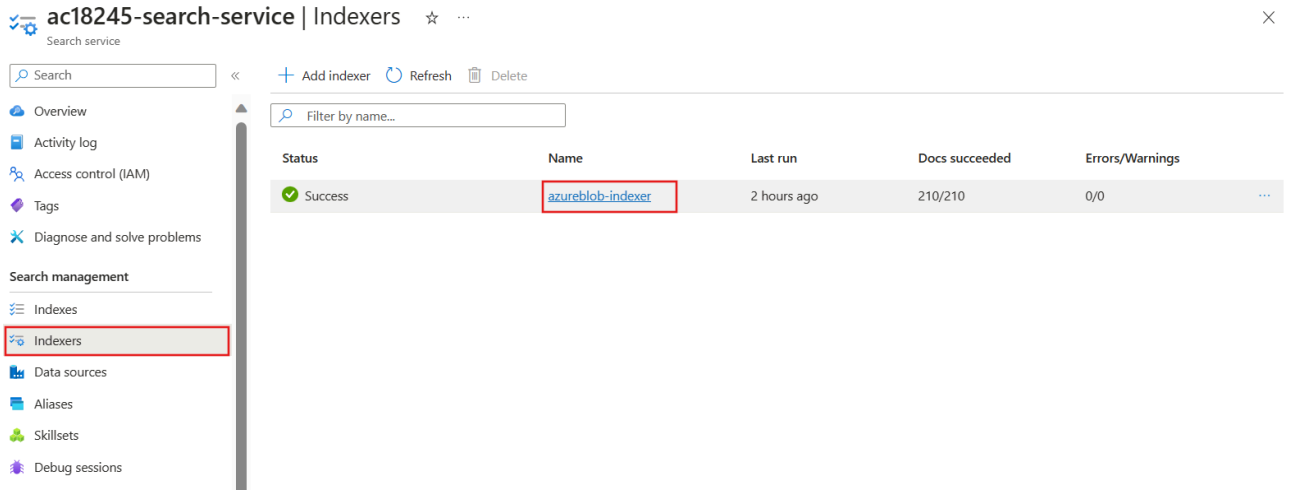


2. Under **Functions**, right-click **customtextcla**, then select **Copy Function Url**.
3. On the Azure portal, replace the URI with the copied function URL.
4. Select **Save**.

Edit the field mappings in the indexer

You now have fields to store the enrichment, a skillset to call your function app, the last step is to tell the cognitive search where to put the enrichment.

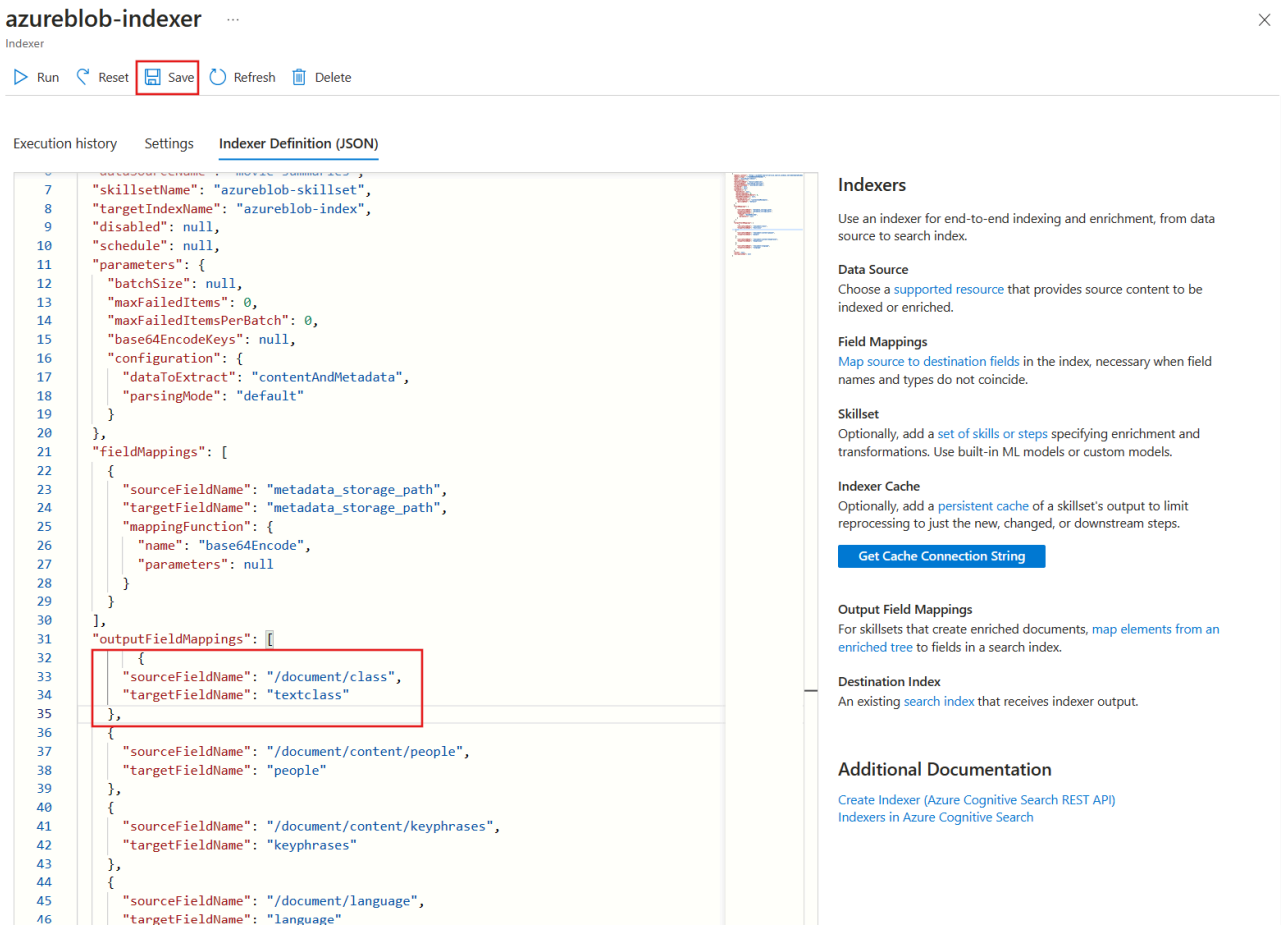
1. At the top of the page, select the search service, for example, **acs18245-search-service|Skillsets** link.|



2. On the **Overview** pane, select **Indexers**.
3. Select **azureblob-indexer**.
4. Select **Indexer Definition (JSON)**.
5. Add a new output field mapping, by pasting this field definition to the top of the output field section.

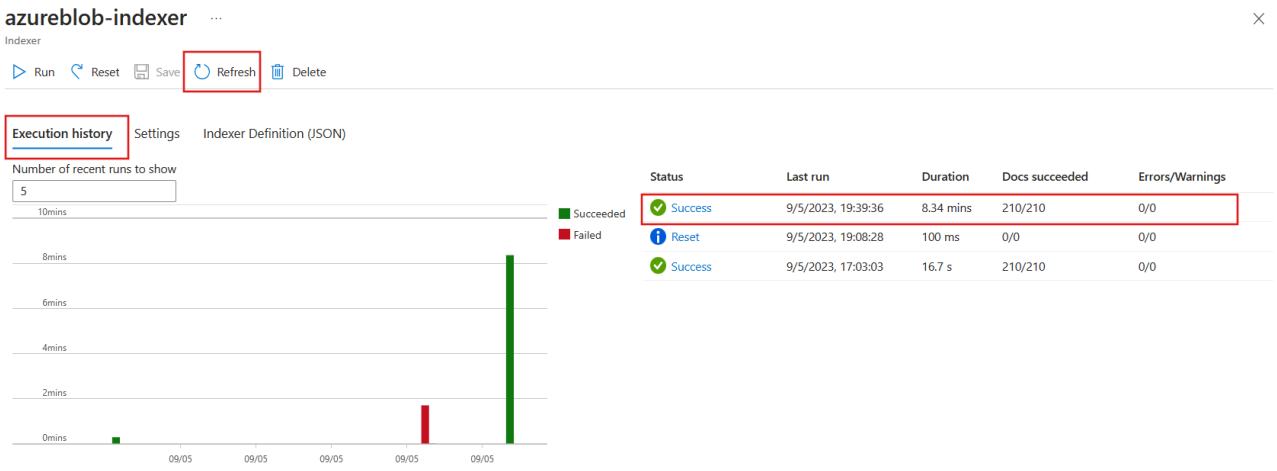
```
{
  "sourceFieldName": "/document/class",
  "targetFieldName": "textclass"
},
```

The indexer JSON definition should now look like this:



6. Select **Save**.
7. Select **Reset**, then select **Yes**.
8. Select **Run**, then select **Yes**.

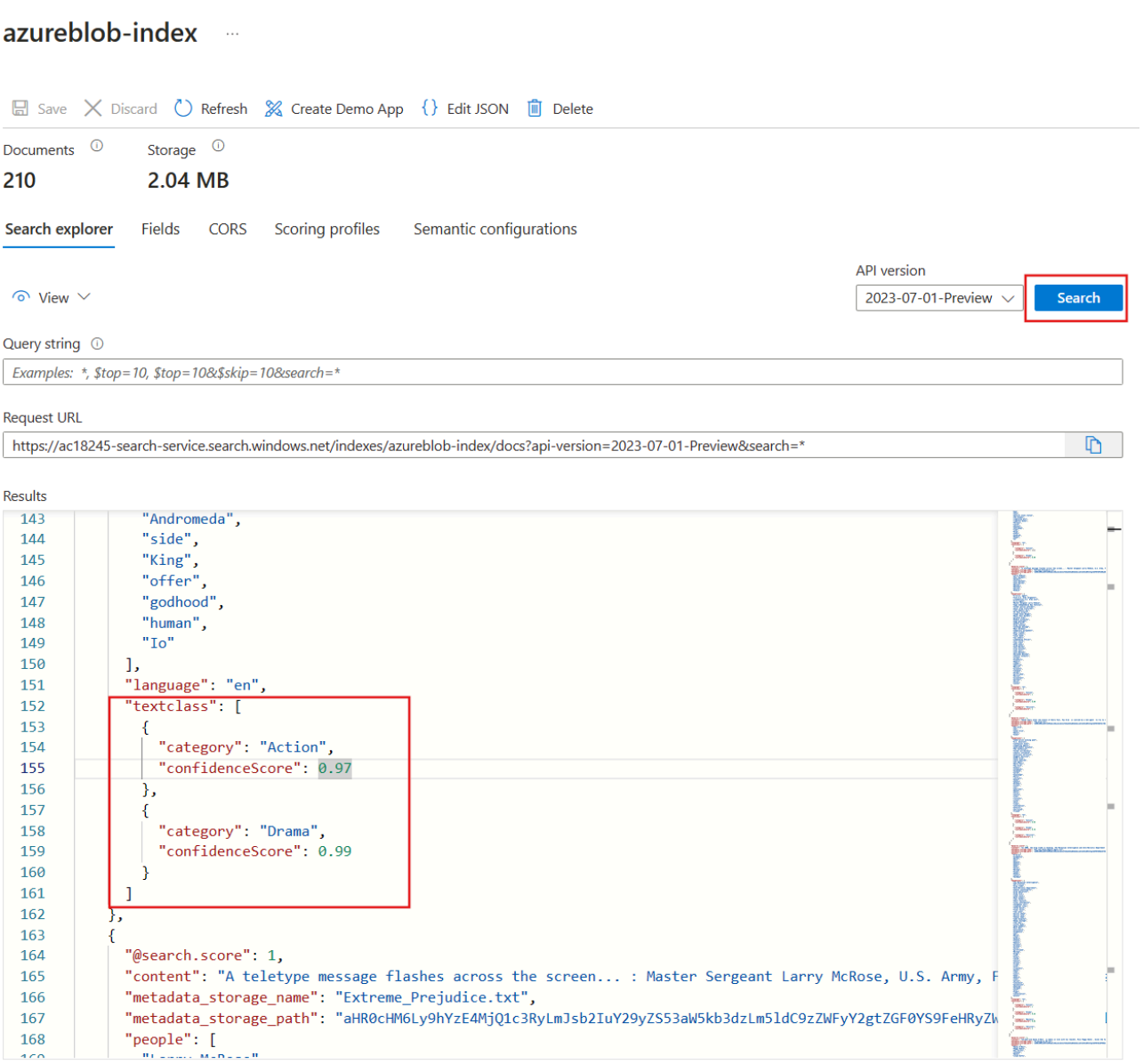
   Your Azure cognitive search service runs your updated indexer. The indexer uses the edited custom skillset. The skillset calls your function app with the document being indexed. The custom text classifier model uses the text in the document to try and identify the genre of the movie. The model returns a JSON document with genres and confidence levels. The indexer maps the JSON results to the fields in your index using the new output field mapping.
9. Select **Execution history**.
10. Check that the indexer has successfully run against the 210 documents.



You might need to select **Refresh** to update the status of the indexer.

Test your enriched search index

1. At the top of the page, select the search service, for example **acs18245-search-service|Indexers**.
2. On the **Overview** pane, select **Indexes**.
3. Select **azureblob-index**.



4. Select **Search**.
5. Explore the search results.

Each document in the index should have a new `textclass` field that can be searched. It contains a category field with the movies genres. It can be more than one. It also shows how confident the custom text classification model is about the identified genre.

Now that you've completed the exercise, delete all the resources you no longer need.

Delete exercise resources

1. In the Azure portal, go to the Home page, and select **Resource groups**.
2. Select the resource groups you don't need, then select **Delete resource group**.

Reference: mslearn-knowledge-mining

Knowledge Check

1. Which of the following features of Azure AI Language can you use out of box without needing to train a model? *

- ○ Conversational language understanding.
- ◉ Analyze sentiment.
  - ✔ Correct. The analyze sentiment feature has a prebuilt model that you can use without any training.
- ○ Custom text classification.

2. Which step enables an AI Search index to store the enrichments from an Azure AI Language project? *

- ◉ Update your Azure AI Search solution.
  - ✔ Correct. You need to edit the index, indexer, and custom skillset to store the enrichments.
- ○ Create a function app.
- ○ Train a custom text classification model.

3. Where do you copy the full endpoint of the function app to use in the custom web skill that includes the api-version and path to the actual function name? *

- ○ You can find the endpoint in the Azure portal.
- ○ You can find the endpoint in Language Studio.
- ◉ You can find the endpoint in the Azure extension inside VS Code.
  - ✔ Correct. You right click on the function name in the Azure extension and select Copy function URL.

Summary

By completing this module, you've learned to:

- Use Azure AI Language to enrich Azure AI Search indexes.
- Enrich an AI Search index with custom classes.

The flexibility of Azure AI Language and AI Search gives you the power to improve the search experience in your apps.

If you'd like to learn more about Language Studio, see the docs overview.

✍️ Compiled by Kenneth Leung (2025)