5.1 Create a multimodal analysis solution with Azure AI Content Understanding

Use **Azure AI Content Understanding** for **multimodal content analysis and information extraction**.

Learning objectives

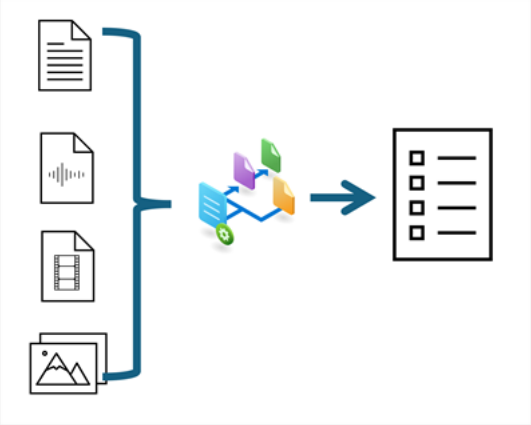
After completing this module, you will be able to:

- Describe capabilities of Azure AI Content Understanding.
- Use Azure AI Content Understanding to build a **content analyzer**.
- Consume a **Content Understanding analyzer** by using the REST API.

Introduction

Organizations today rely on information that is often locked up in content assets such as documents, images, videos, and audio recordings. Extracting information from this content can be challenging, laborious, and time-consuming, and organizations often need to build solutions based on multiple technologies for content analysis depending on the formats being used.

Azure AI Content Understanding is **a multimodal service that simplifies the creation of AI-powered analyzers that can extract information from content** in practically any format.



In this module, you'll explore the capabilities of Azure AI Content Understanding, and learn how to use it to build custom analyzers.

> Note: Azure AI Content Understanding is currently in public preview. Details described in this module are subject to change.

What is Azure AI Content Understanding?

Azure AI Content Understanding is *a generative AI service that you can use to extract insights and data from multiple kinds of content*. With Content Understanding, you can quickly build applications that analyze complex data and generate outputs that can be used to automate and optimize processes.

Content Understanding is a component of Azure AI services. To use it, you need to provision an Azure AI services resource in your Azure subscription. You can develop and manage a Content Understanding solution:

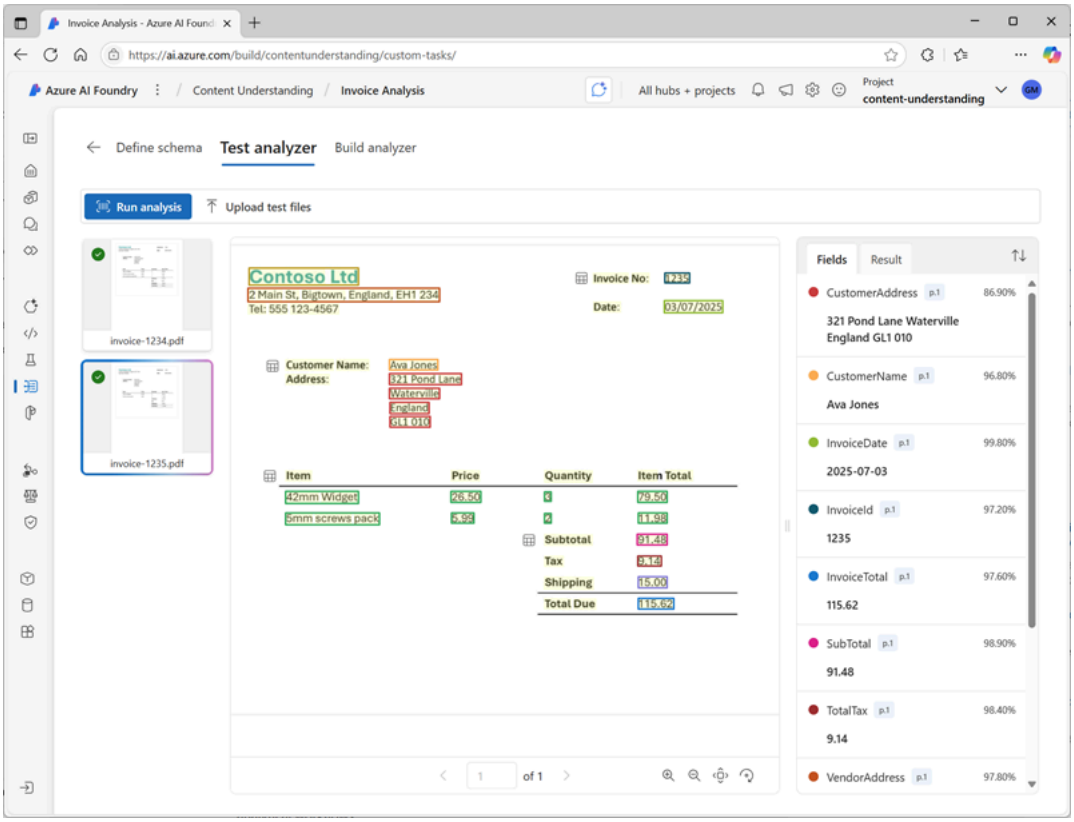
- In the **Azure AI Foundry portal**
- By using the **Content Understanding REST API**

Multimodal content analysis

Content Understanding can extract information from common kinds of content, enabling you to use a single service with a straightforward and consistent development process to **build multimodal content analysis solutions**.

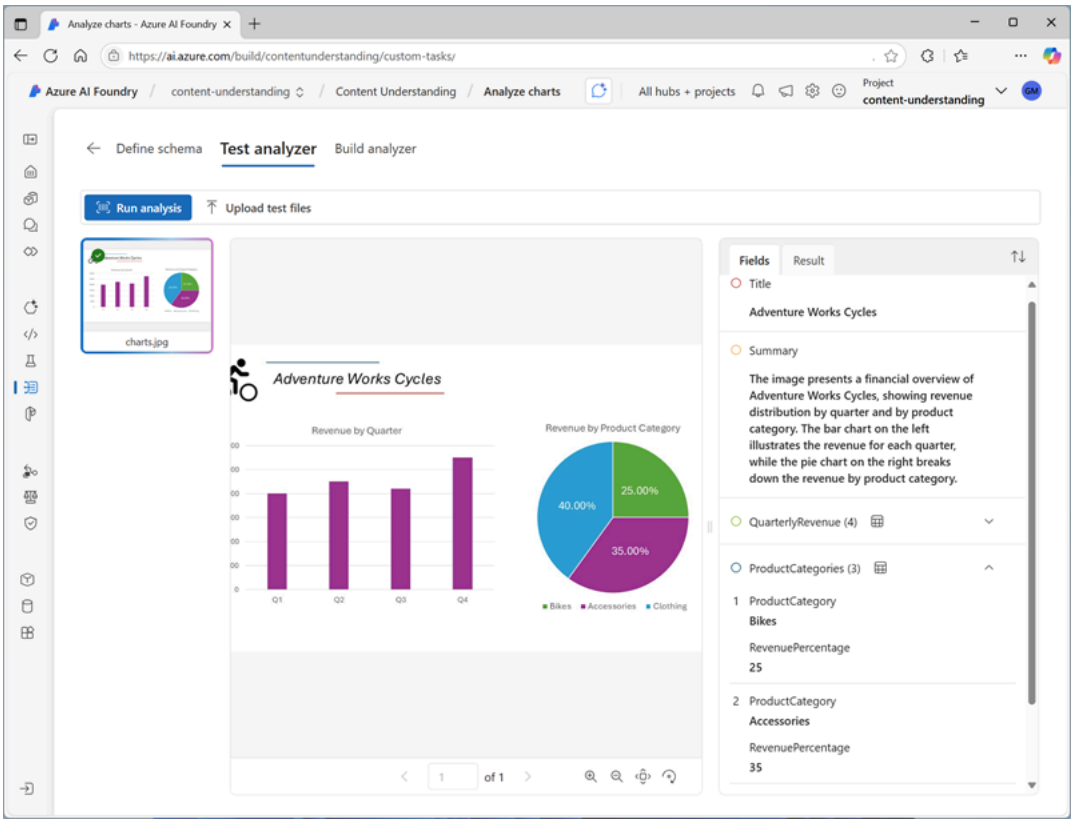
Documents and forms

You can use Content Understanding to **analyze documents and forms and retrieve specific field values**. For example, you could extract key data values from an invoice to automate payment processing.



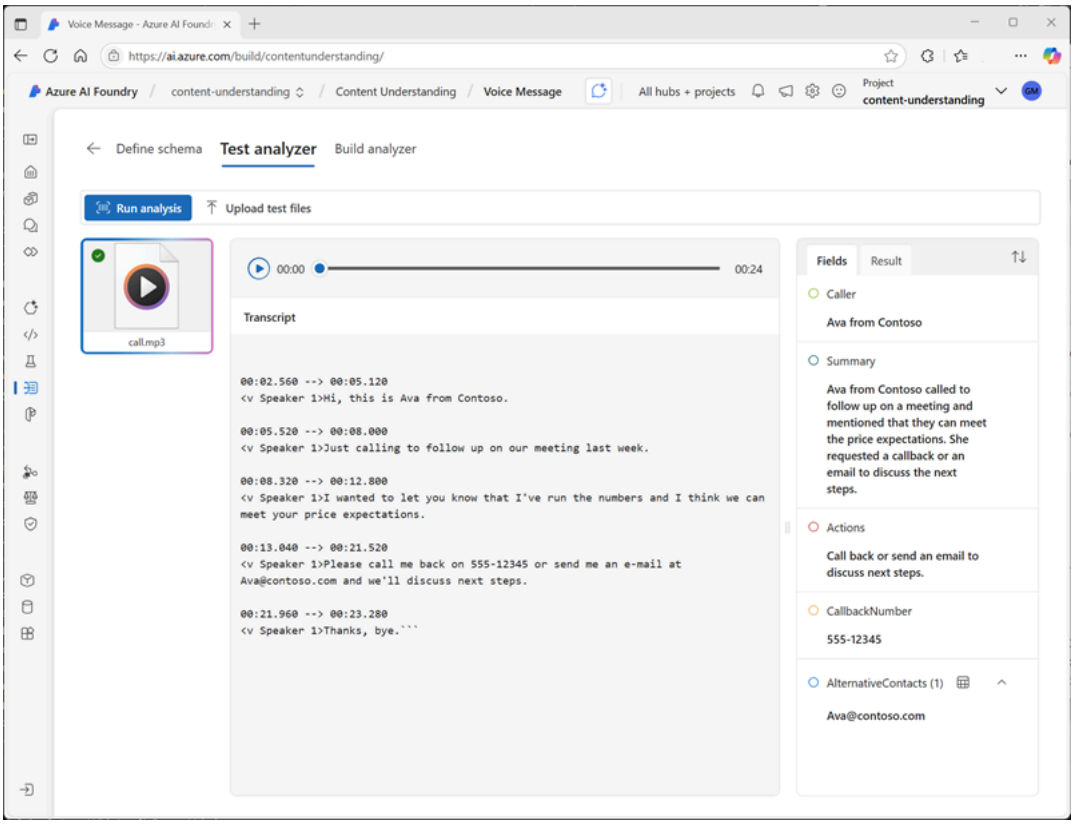
Images

You can analyze images to *infer information from visuals such as charts, identify physical defects in products or other items, detect the presence of specific objects or people, or determine other information visually*.



Audio

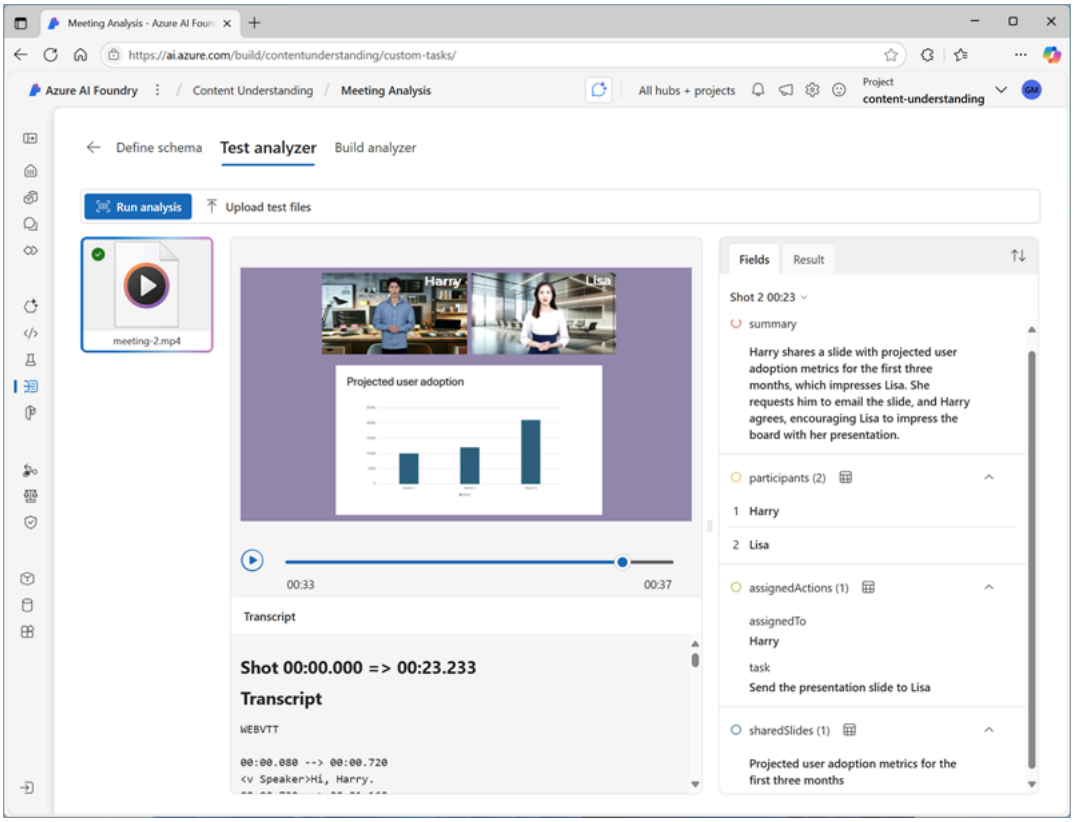
Analysis of audio enables you to automate tasks like *summarizing conference calls, determining sentiment of recorded customer conversations, or extracting key data from telephone messages*.



Video

Video accounts for a large volume of the data captured today, and you can use Content Understanding to **analyze and extract insights from video** to support many scenarios. For example, to *extract key points from video conference recordings, to summarize presentations, or to detect the presence of*

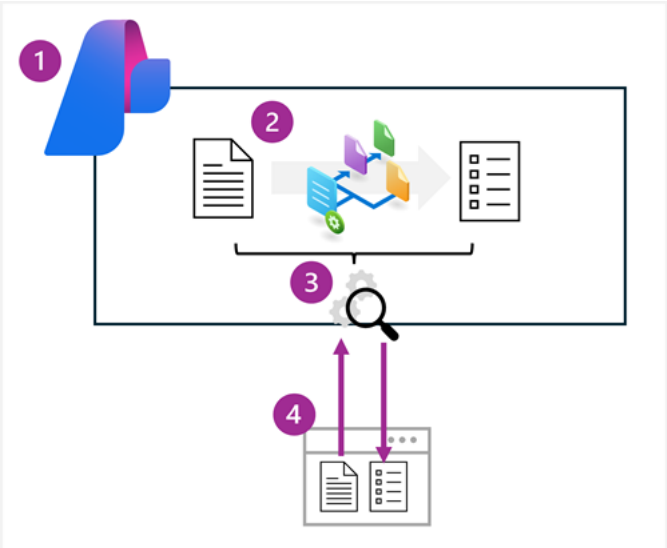
specific activity in security footage.



Create a Content Understanding analyzer

Content Understanding solutions are based on the creation of an **analyzer**; which is trained to extract specific information from a particular type of content based on a schema that you define.

The high-level process for creating a Content Understanding solution includes the following steps:



1. Create an *Azure AI services resource*.
2. **Define a Content Understanding schema** for the information to be extracted. This can be based on **a content sample** and **an analyzer template**.
3. **Build an analyzer** based on the completed schema.
4. Use the **analyzer** to extract or generate fields from new content.

Numerous **analyzer templates** are provided to help you develop an appropriate analyzer for your needs quickly. Additionally, because of the generative AI capabilities of Content Understanding, you can use minimal training data to define a schema by example. In many cases, the service accurately identifies the data values in the sample content that map to the schema elements automatically, though you can also explicitly label fields in content such as documents to improve the performance of your analyzer.

Creating an analyzer with Azure AI Foundry

While you can provision an Azure AI services resource and develop a complete Content Understanding solution through the REST API, **the preferred approach for AI development projects is to use zure AI Foundry**. Specifically, you can **use the Azure AI Foundry portal to create a project, define a Content Understanding schema, and build and test an analyzer**.

1. Creating a Content Understanding project

In Azure AI Foundry, you can *create a project* in an existing AI hub, or you can create a new hub as you create the project. In addition to the AI hub itself, creating a hub provisions the Azure resources needed to support one or more projects; including an Azure AI services resource, storage, and a key vault resource to store sensitive details like credentials and keys.

> Note: Content Understanding schemas can only be created in Azure locations where the service is supported. For more information, see Content Understanding region and language support.

2. Defining a schema

After creating a project, **the first step in building an analyzer is to define a schema** for the content the analyzer will process, and the information it will extract. **Azure AI Foundry provides a schema editor interface** in which you can upload a file (document, image, audio, or video) on which the schema should be based. You can then apply an appropriate schema template and define the specific fields you want the analyzer to identify.

> Note: The templates and field types available in a schema depend on the content type of the file on which the schema is based. Some content types support additional optional functionality, such as extracting barcodes and formulae from text in documents. For more information about using Content Understanding with different content types, see the following articles in the product documentation:
>
> - Content Understanding document solutions
> - Content Understanding image solutions
> - Content Understanding audio solutions
> - Content Understanding video solutions

3. Testing

You can **test the analyzer schema** at any time during the development process by running analysis on the sample file used to define the schema or other uploaded files. **The test results include the extracted field values and the JSON format output** returned by the analyzer to client applications.

4. Building an analyzer

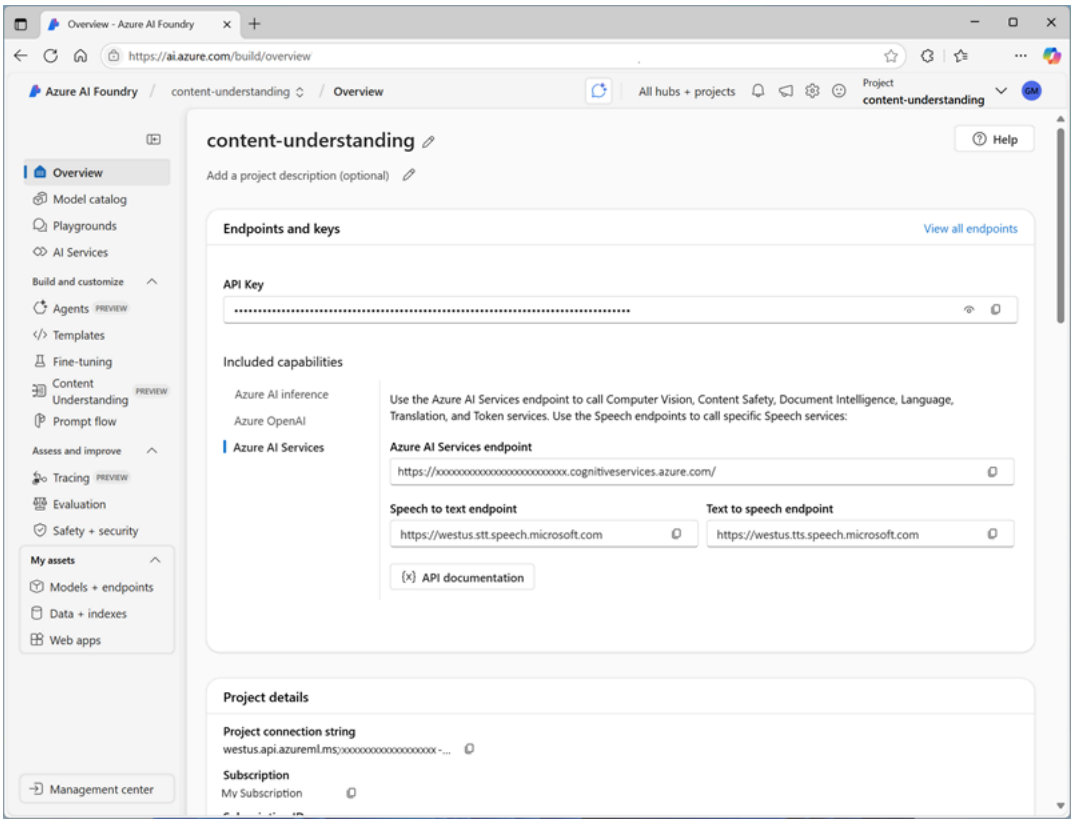
When you're satisfied with the performance of your schema, you can build your analyzer. Building an analyzer makes it **accessible to client applications through Content Understanding endpoint** for the Azure AI services resource associated with your project.

After building your analyzer, you can continue to test it in the Azure AI Foundry portal, and refine the schema to create new named versions with different capabilities.

Use the Content Understanding REST API

The **Content Understanding REST API** provides a programmatic interface that you can use to **create, manage, and consume analyzers**.

To use the REST API, your *client application submits HTTP calls to the Content Understanding* **endpoint** for your Azure AI services resource, **passing one of the *authorization keys* in the header**. You can obtain the endpoint and keys in the Azure portal or in the Azure AI Foundry portal. You can also use the Azure AI Foundry API to connect to the project and retrieve the endpoint and key for your Azure AI Services resource programmatically.



Using the REST API to analyze content

One of the most common uses of the REST API is to submit content to an existing analyzer that you have previously built, and retrieve the results of analysis. The **analysis request returns an operation ID** value that represents an asynchronous task. Your client application must then use another request to **pass the operation ID back to the endpoint and retrieve the operation status** - potentially polling multiple times until the operation is complete and **the results are returned in JSON format**.

For example, to analyze a document, a client application might submit a POST request to the analyze function containing the following JSON body:

```
POST {endpoint}/contentunderstanding/analyzers/{analyzer}:analyze?api-version={api version} {   "url": "https://host.com/doc.pdf" }
```

> Note: You can **specify a URL for the content file location**, or you can **include the binary contents of the file**.

Assuming the request is authenticated and initiated successfully, the response will be similar to this example:

```
Operation-Id: 1234abcd-1234-abcd-1234-abcd1234abcd
Operation-Location: {endpoint}/contentunderstanding/analyzers/{analyzer}/results/1234abcd-1234-abcd-1234-abcd1234abcd?api-version={api version} {
```

Your client application must then use the **operation ID** that has been returned to **check the status of the operation until it has succeeded (or failed)** by submitting a GET request to the results function.

```
GET {endpoint}/contentunderstanding/analyzers/{analyzer}/results/1234abcd-1234-abcd-1234-abcd1234abcd?api-version={api version}
```

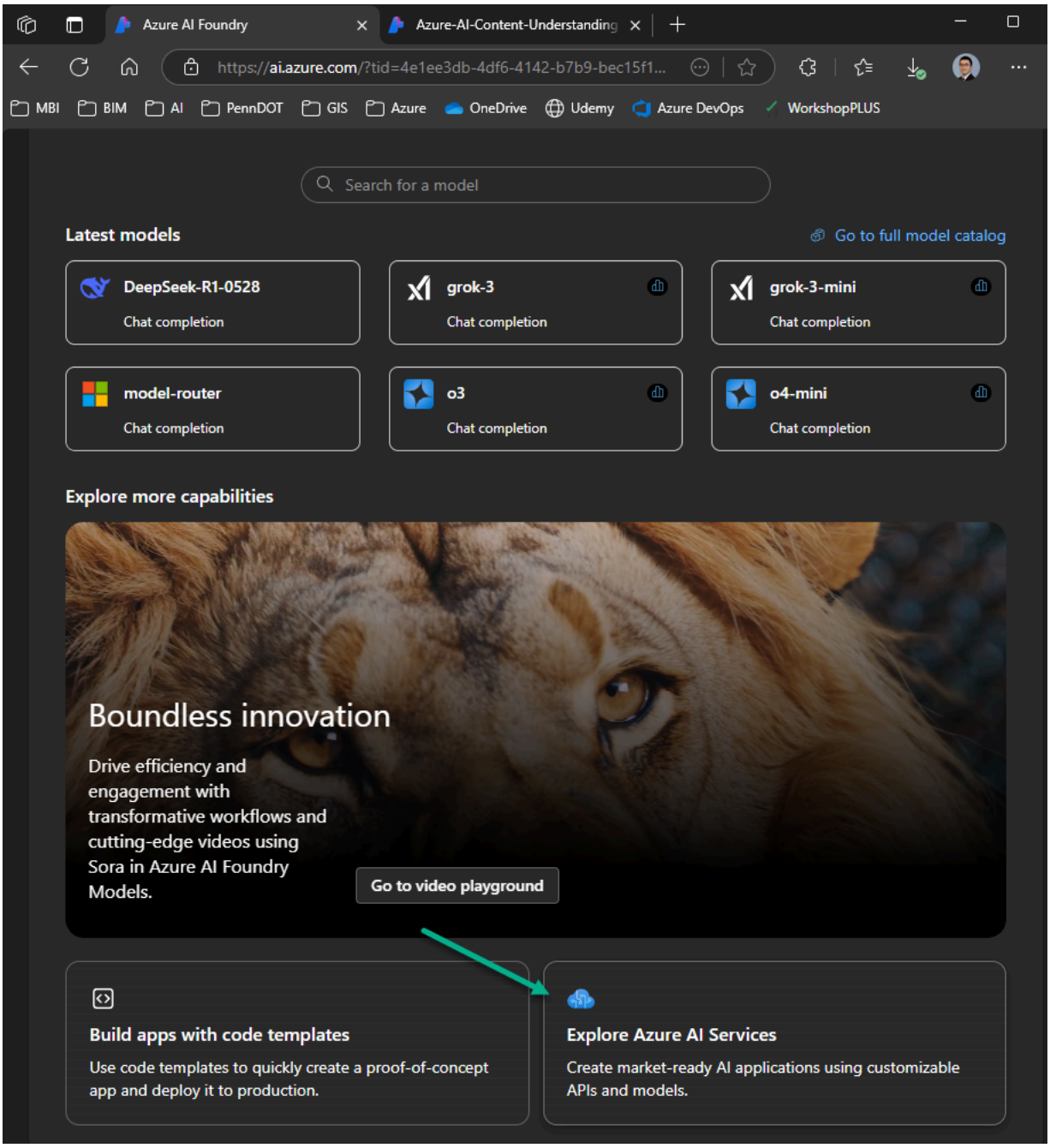
When the operation has completed successfully, **the response contains a JSON payload representing the results of the analysis**. The specific results depend on the content and schema.

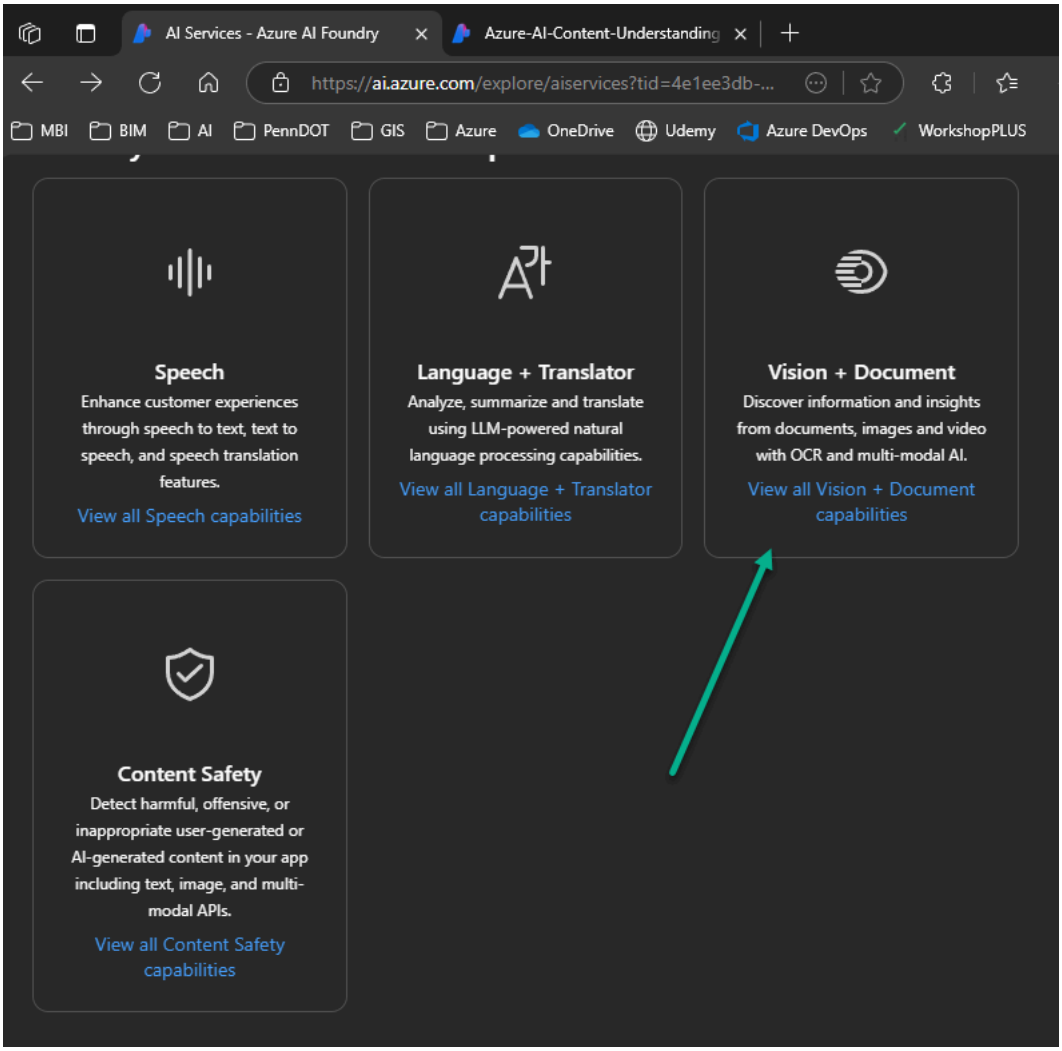
> Note: For more information about the Content Understanding REST API, see the reference documentation.

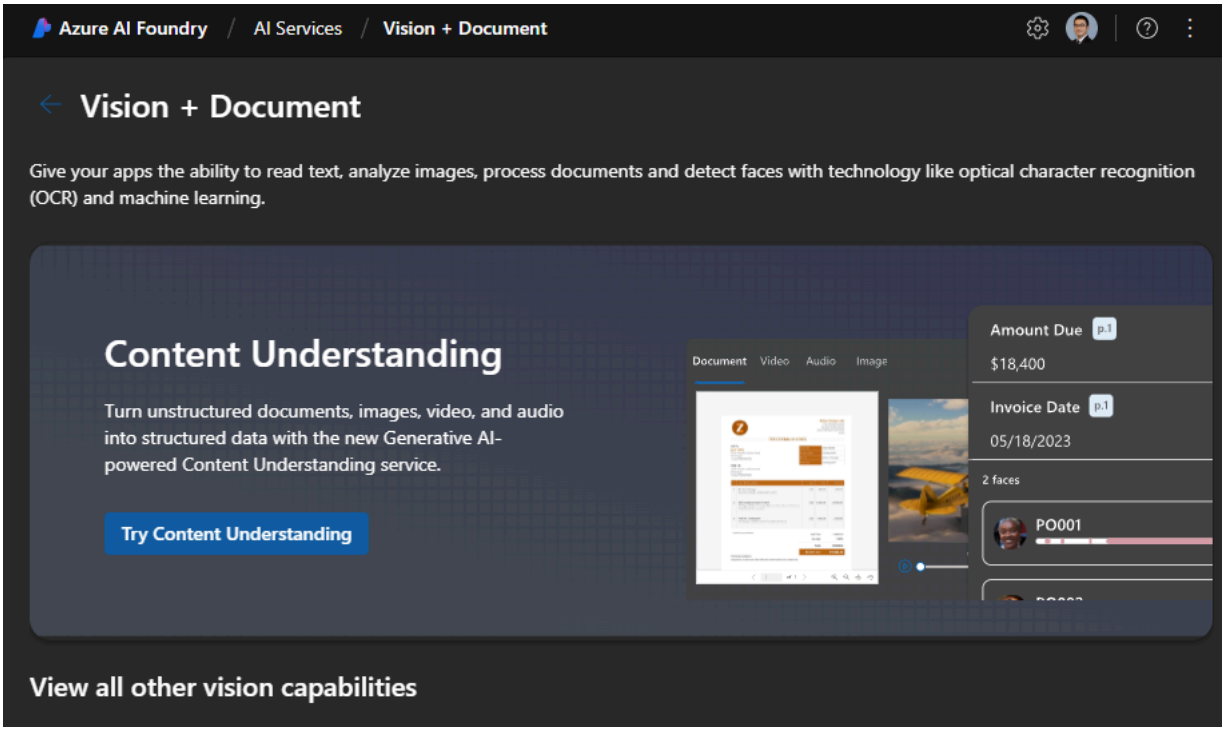
Exercise - Extract information from multimodal content

Extract information from multimodal content

In this exercise, you use Azure Content Understanding to extract information from a variety of content types; including an invoice, an images of a slide containing charts, an audio recording of a voice messages, and a video recording of a conference call.


Azure AI Foundry
Azure-AI-Content-Understanding
https://ai.azure.com/?tid=4e1ee3db-4df6-4142-b7b9-bec15f1...
MBI
BIM
AI
PennDOT
GIS
Azure
OneDrive
Udemy
Azure DevOps
WorkshopPLUS
Search for a model
Latest models
Go to full model catalog
DeepSeek-R1-0528
Chat completion
grok-3
Chat completion
grok-3-mini
Chat completion
model-router
Chat completion
o3
Chat completion
o4-mini
Chat completion
Explore more capabilities
Boundless innovation
Drive efficiency and engagement with transformative workflows and cutting-edge videos using Sora in Azure AI Foundry Models.
Go to video playground
Build apps with code templates
Use code templates to quickly create a proof-of-concept app and deploy it to production.
Explore Azure AI Services
Create market-ready AI applications using customizable APIs and models.


AI Services - Azure AI Foundry
Azure-AI-Content-Understanding
https://ai.azure.com/explore/aiservices?tid=4e1ee3db-...
MBI
BIM
AI
PennDOT
GIS
Azure
OneDrive
Udemy
Azure DevOps
WorkshopPLUS
Speech
Enhance customer experiences through speech to text, text to speech, and speech translation features.
View all Speech capabilities
Language + Translator
Analyze, summarize and translate using LLM-powered natural language processing capabilities.
View all Language + Translator capabilities
Vision + Document
Discover information and insights from documents, images and video with OCR and multi-modal AI.
View all Vision + Document capabilities
Content Safety
Detect harmful, offensive, or inappropriate user-generated or AI-generated content in your app including text, image, and multi-modal APIs.
View all Content Safety capabilities


Azure AI Foundry / AI Services / Vision + Document
Vision + Document
Give your apps the ability to read text, analyze images, process documents and detect faces with technology like optical character recognition (OCR) and machine learning.
Content Understanding
Turn unstructured documents, images, video, and audio into structured data with the new Generative AI-powered Content Understanding service.
Try Content Understanding
Document
Video
Audio
Image
Amount Due
$18,400
Invoice Date
05/18/2023
2 faces
PO001
View all other vision capabilities

Module assessment

1. What kinds of AI solution is Azure AI Content Understanding designed to help you build? **Analyzers that extract information from documents, images, videos, and audio files**.
2. Which graphical tool should you use to create an Azure AI Content Understanding project? **Azure AI Foundry protal**.
3. What should you define for the information you want to extract from content? **A schema**

Summary

Azure AI Content Understanding is a multimodal AI service that enables you to extract information from many different kinds of content.

In this module, you learned how to use the **Azure AI Foundry portal** to create a Content Understanding project and build an **analyzer**.

> Note: For more information about Azure AI Content Understanding, see Azure AI Content Understanding documentation.

Extract information from multimodal content

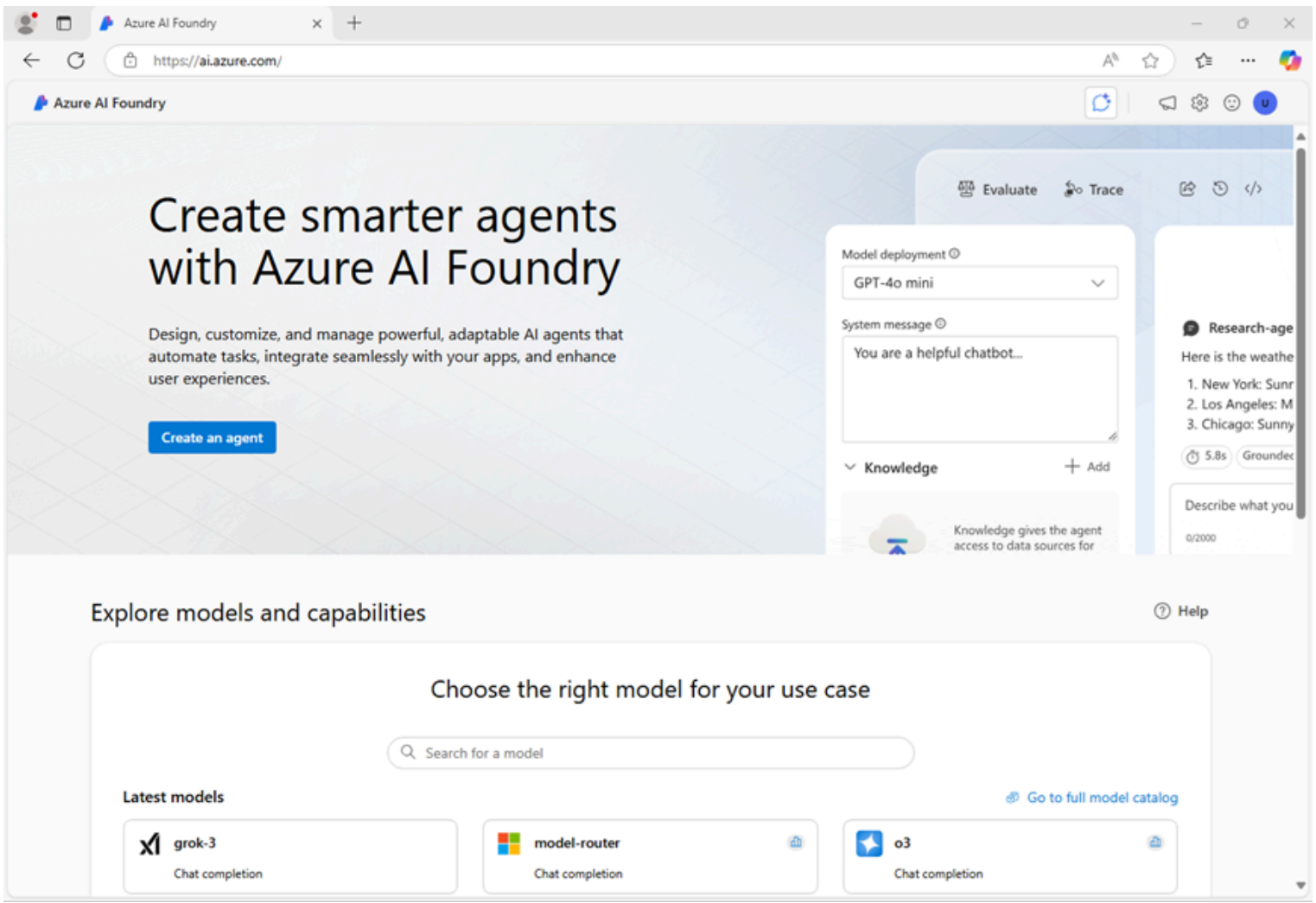
In this exercise, you use Azure Content Understanding to extract information from a variety of content types; including an invoice, an images of a slide containing charts, an audio recording of a voice messages, and a video recording of a conference call.

This exercise takes approximately **40** minutes.

Create an Azure AI Foundry hub and project

The features of Azure AI Foundry we're going to use in this exercise require a project that is based on an Azure AI Foundry *hub* resource.

1. In a web browser, open the Azure AI Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Azure AI Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):



2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: Select one of the following locations (*At the time of writing, Azure AI Content understanding is only available in these regions*):
        - Australia East
        - Sweden Central
        - West US

> **Note**: If you're working in an Azure subscription in which policies are used to restrict allowable resource names, you may need to use the link at the bottom of the **Create a new project** dialog box to create the hub using the Azure portal.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created.

Download content

The content you're going to analyze is in a .zip archive. Download it and extract it in a local folder.

1. In a new browser tab, download content.zip from
   ```
   https://github.com/microsoftlearning/mslearn-ai-information-extraction/raw/main/Labfiles/content/content.zip
   ```
   and save it in a local folder.
2. Extract the downloaded *content.zip* file and view the files it contains. You'll use these files to build various Content Understanding analyzers in this exercise.

> **Note**: If you're only interested in exploring analysis of a specific modality (documents, images, video, or audio), you can skip to the relevant task below. For the best experience, go through each task to learn how to extract information from different types of content.

Extract information from invoice documents

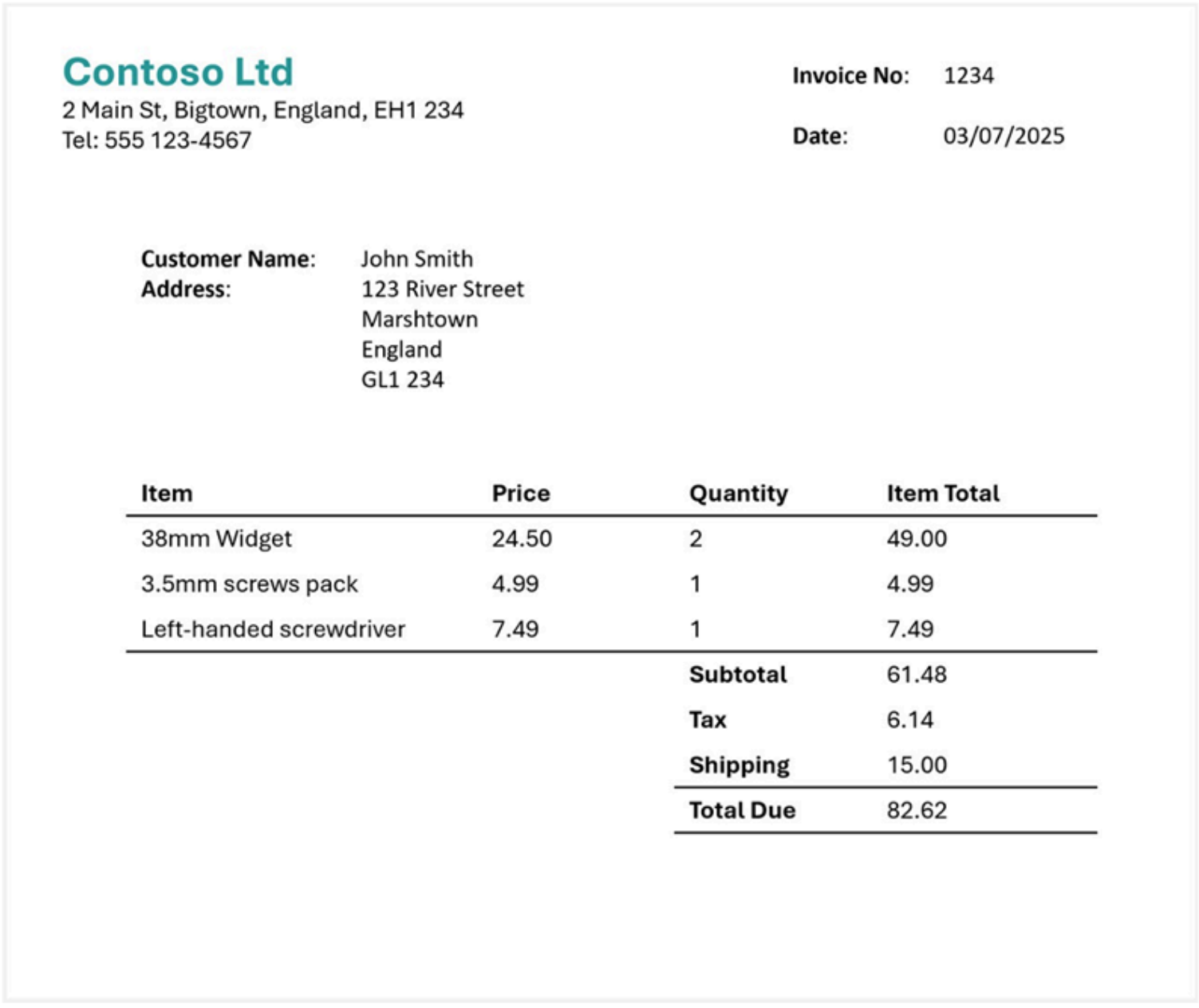
You are going to build an Azure AI Content Understanding analyzer that can extract information from invoices. You'll start by defining a schema based on a sample invoice.

Define a schema for invoice analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
   - **Task name**: `Invoice analysis`
   - **Description**: `Extract data from an invoice`
   - **Single file content analysis**: *Selected*
   - **Advanced settings**:
     - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
     - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **invoice-1234.pdf** file from the folder where you extracted content files. This file contains the following invoice:



Contoso Ltd
2 Main St, Bigtown, England, EH1 234
Tel: 555 123-4567

Invoice No: 1234
Date: 03/07/2025

Customer Name: John Smith
Address: 123 River Street
Marshtown
England
GL1 234

Item	Price	Quantity	Item Total
38mm Widget	24.50	2	49.00
3.5mm screws pack	4.99	1	4.99
Left-handed screwdriver	7.49	1	7.49
		Subtotal	61.48
		Tax	6.14
		Shipping	15.00
		Total Due	82.62

6. On the **Define schema** page, after uploading the invoice file, select the **Invoice data extraction** template and select **Create**.

   The *Invoice analysis* template includes common fields that are found in invoices. You can use the schema editor to delete any of the suggested fields that you don't need, and add any custom fields that you do.
7. In the list of suggested fields, select **BillingAddress**. This field is not needed for the invoice format you have uploaded, so use the **Delete field** (🗑) icon that appears in the selected field row to delete it.
8. Now delete the following suggested fields, which aren't needed for your invoice schema:

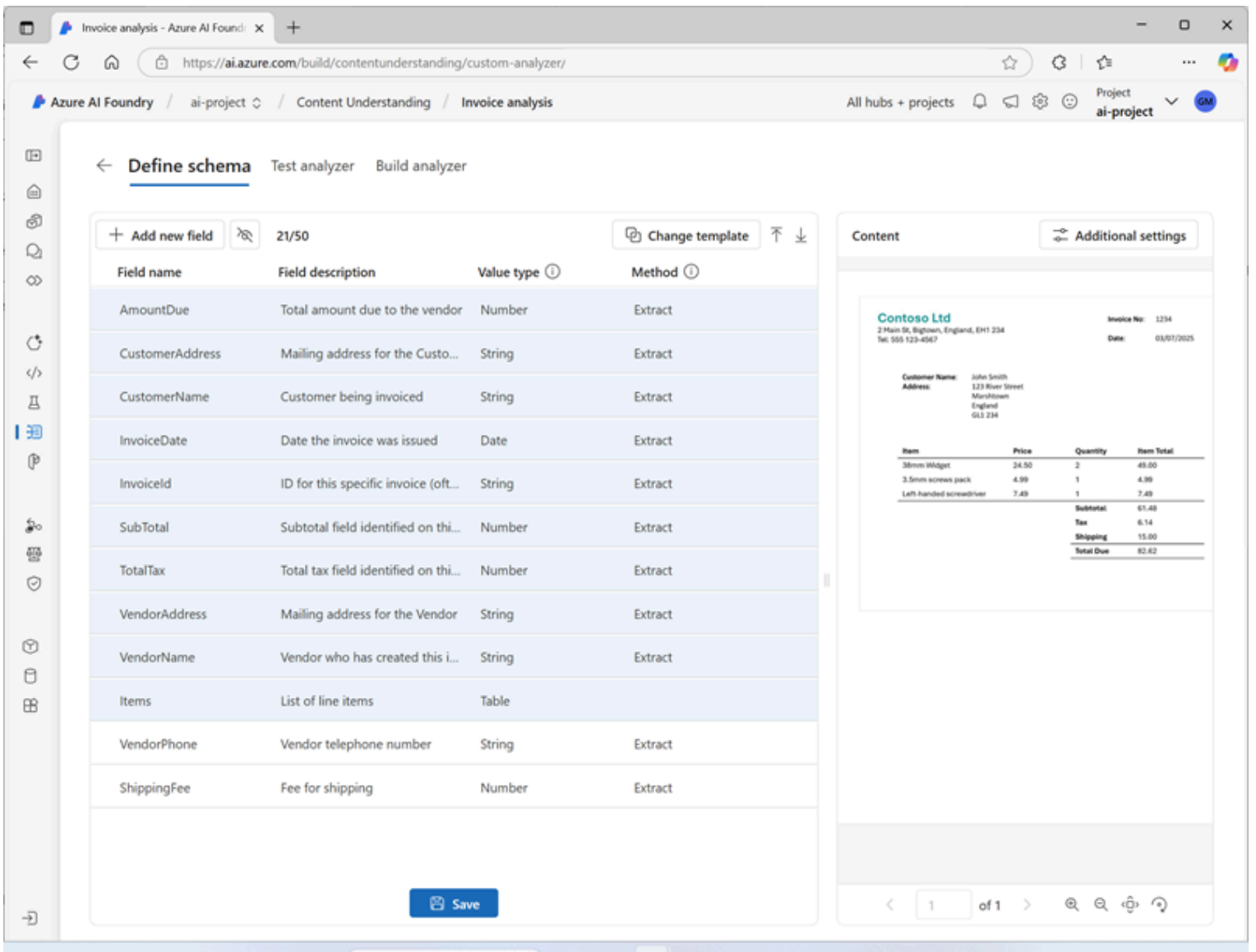
   - BillingAddressRecipient
   - CustomerAddressRecipient
   - CustomerId
   - CustomerTaxId
   - DueDate
   - InvoiceTotal
   - PaymentTerm
   - PreviousUnpaidBalance
   - PurchaseOrder
   - RemittanceAddress
   - RemittanceAddressRecipient
   - ServiceAddress
   - ServiceAddressRecipient
   - ShippingAddress
   - ShippingAddressRecipient
   - TotalDiscount
   - VendorAddressRecipient
   - VendorTaxId
   - TaxDetails

9. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

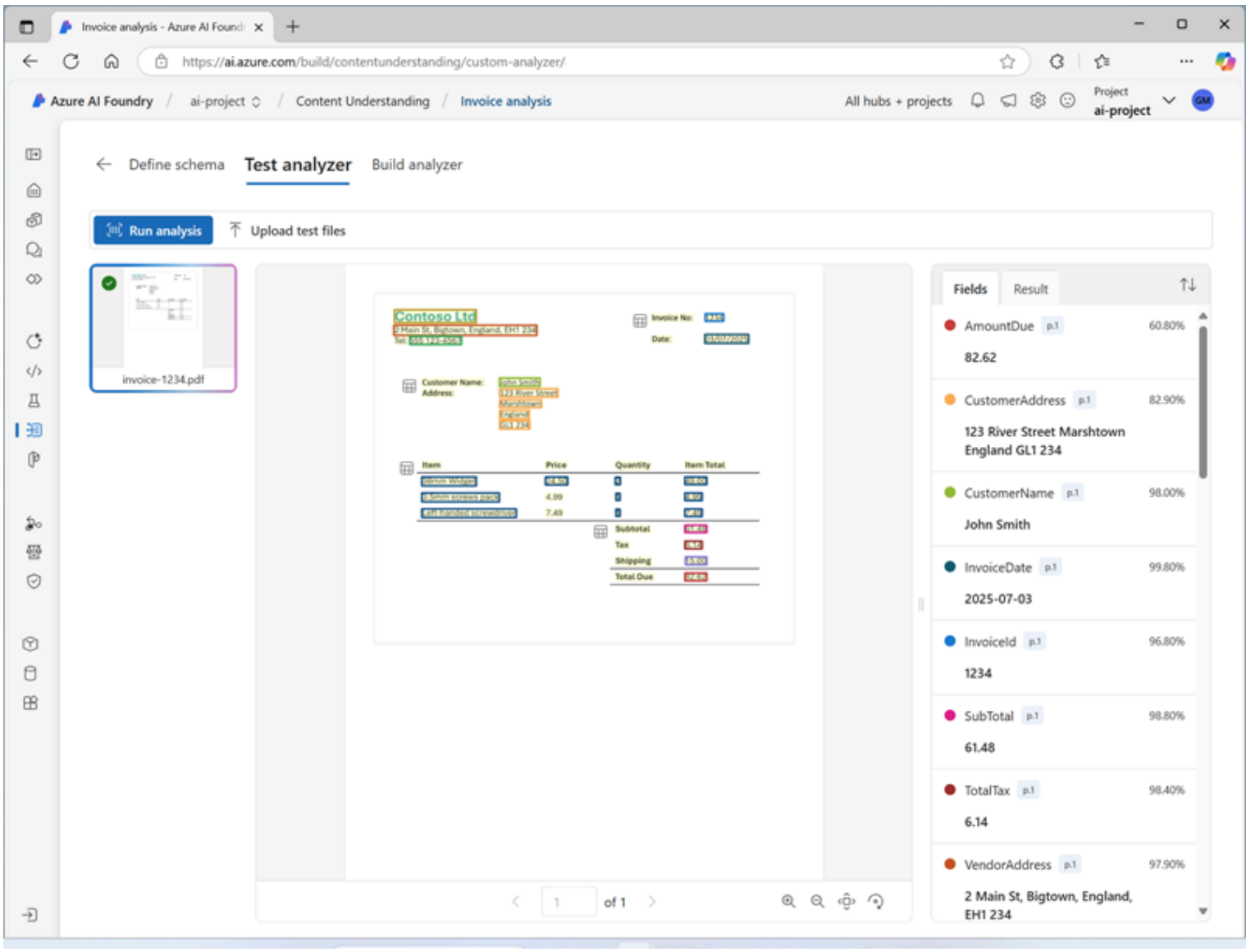
Field name	Field description	Value type	Method
`VendorPhone`	`Vendor telephone number`	String	Extract

Field name	Field description	Value type	Method
`ShippingFee`	`Fee for shipping`	Number	Extract

10. In the row for the **Items** field, note that this field is a *table* (it contains the collection of items in the invoice). Select it's **Edit** (▦) icon to open a new page with its subfields.
11. Remove the following subfields from the **Items** table:
    - Date
    - ProductCode
    - Unit
    - TaxAmount
    - TaxRate
12. Use the **OK** button to confirm the changes and return to the top-level of the invoice schema.
13. Verify that your completed schema looks like this, and select **Save**.



14. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.
15. Review the analysis results, which should look similar to this:



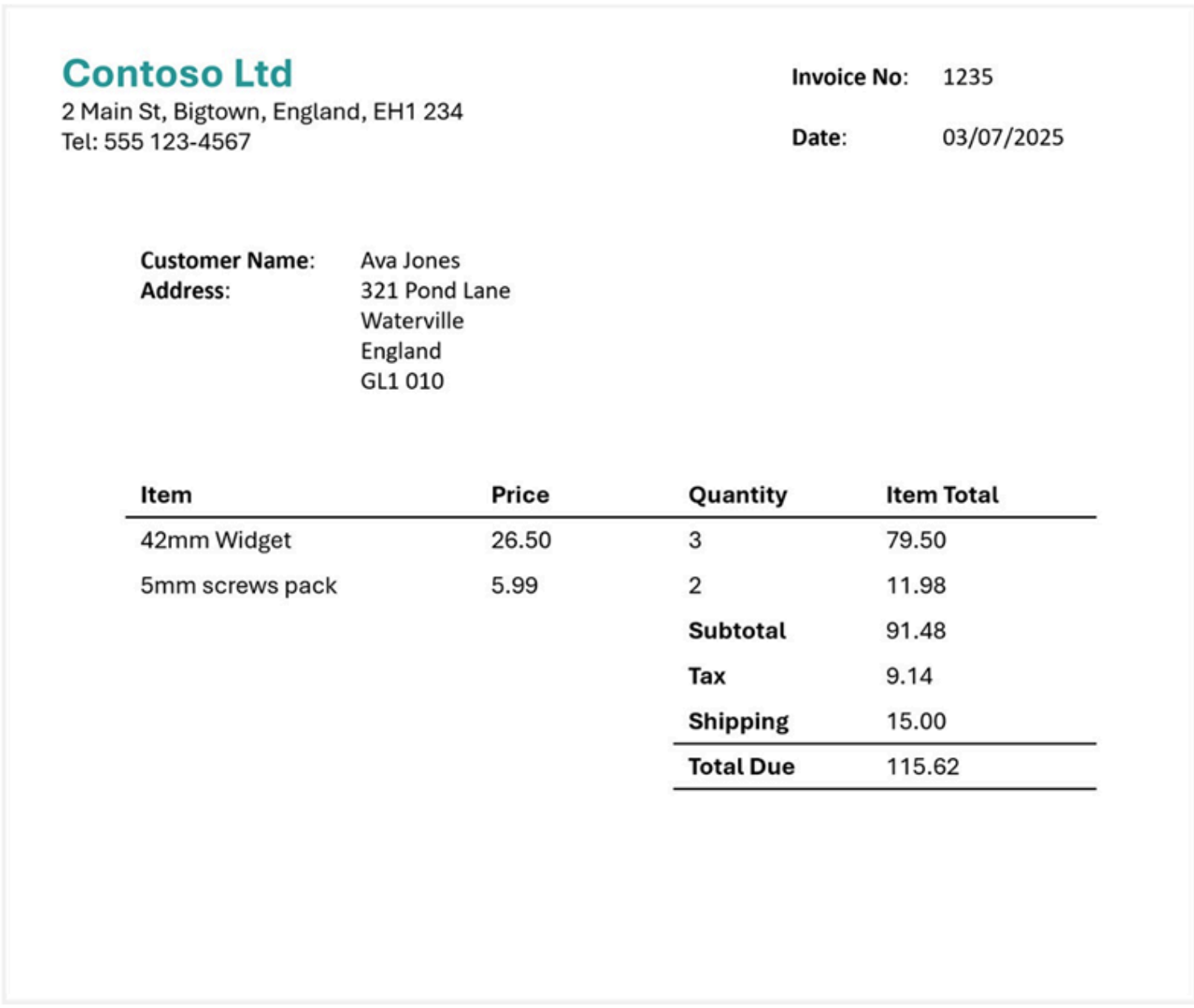
16. View the details of the fields that were identified in the **Fields** pane.

Build and test an analyzer for invoices

Now that you have trained a model to extract fields from invoices, you can build an analyzer to use with similar documents.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `invoice-analyzer`
    - **Description**: `Invoice analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **invoice-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **invoice-analyzer** page, select the **Test** tab.
5. Use the **+ Upload test files** button to upload **invoice-1235.pdf** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the invoice.

The invoice being analyzed looks like this:



Contoso Ltd
2 Main St, Bigtown, England, EH1 234
Tel: 555 123-4567

Invoice No: 1235
Date: 03/07/2025

Customer Name: Ava Jones
Address: 321 Pond Lane
Waterville
England
GL1 010

Item	Price	Quantity	Item Total
42mm Widget	26.50	3	79.50
5mm screws pack	5.99	2	11.98
		Subtotal	91.48
		Tax	9.14
		Shipping	15.00
		Total Due	115.62

6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the test invoice.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content Understanding REST interface to call your analyzer.
9. Close the **invoice-analyzer** page.

Extract information from a slide image

You are going to build an Azure AI Content Understanding analyzer that can extract information from a slide containing charts.

Define a schema for image analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
    - **Task name**: `Slide analysis`
    - **Description**: `Extract data from an image of a slide`
    - **Single file content analysis**: *Selected*
    - **Advanced settings**:
        - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
        - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **slide-1.jpg** file from the folder where you extracted content files. Then select the **Image analysis** template and select **Create**.

The *Image analysis* template doesn't include any predefined fields. You must define fields to describe the information you want to extract.

6. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

Field name	Field description	Value type	Method
`Title`	`Slide title`	String	Generate
`Summary`	`Summary of the slide`	String	Generate
`Charts`	`Number of charts on the slide`	Integer	Generate

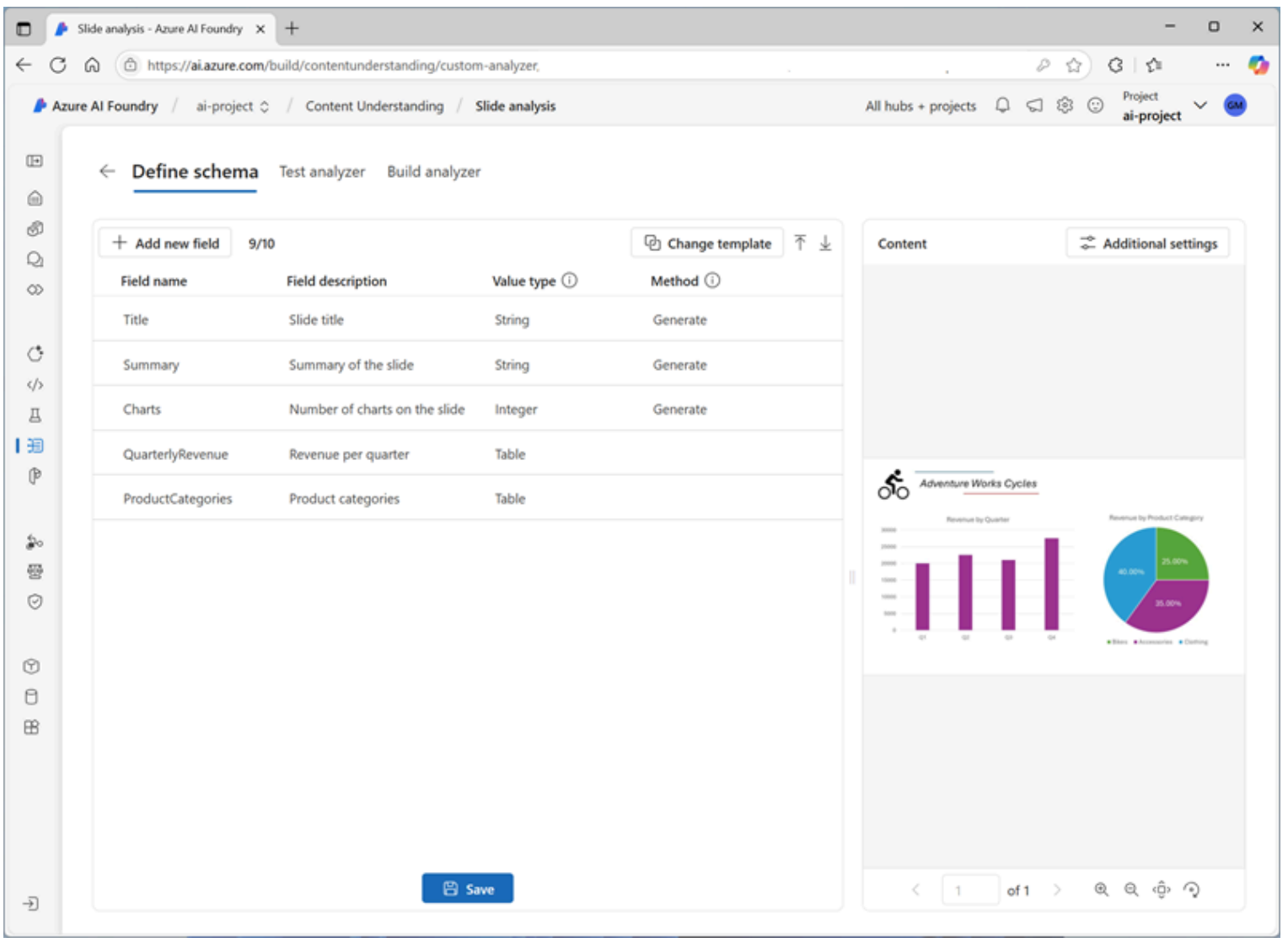
7. Use **+ Add new field** button to add a new field named `QuarterlyRevenue` with the description `Revenue per quarter` with the value type **Table**, and save the new field (✓). Then, in the new page for the table subfields that opens, add the following subfields:

Field name	Field description	Value type	Method
`Quarter`	`Which quarter?`	String	Generate
`Revenue`	`Revenue for the quarter`	Number	Generate

8. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and use **+ Add new field** button to add a new field named `ProductCategories` with the description `Product categories` with the value type **Table**, and save the new field (✓). Then, in the new page for the table subfields that opens, add the following subfields:

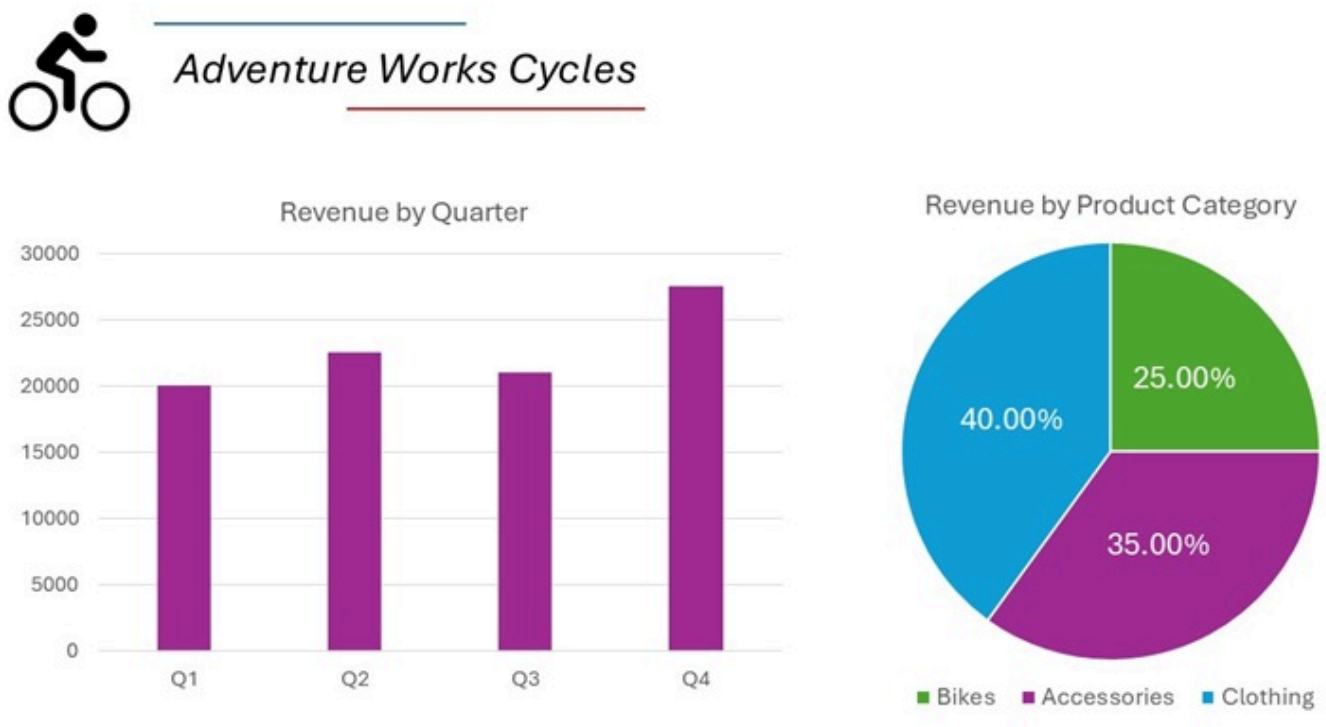
Field name	Field description	Value type	Method
`ProductCategory`	`Product category name`	String	Generate
`RevenuePercentage`	`Percentage of revenue`	Number	Generate

9. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and verify that it looks like this. Then select **Save**.

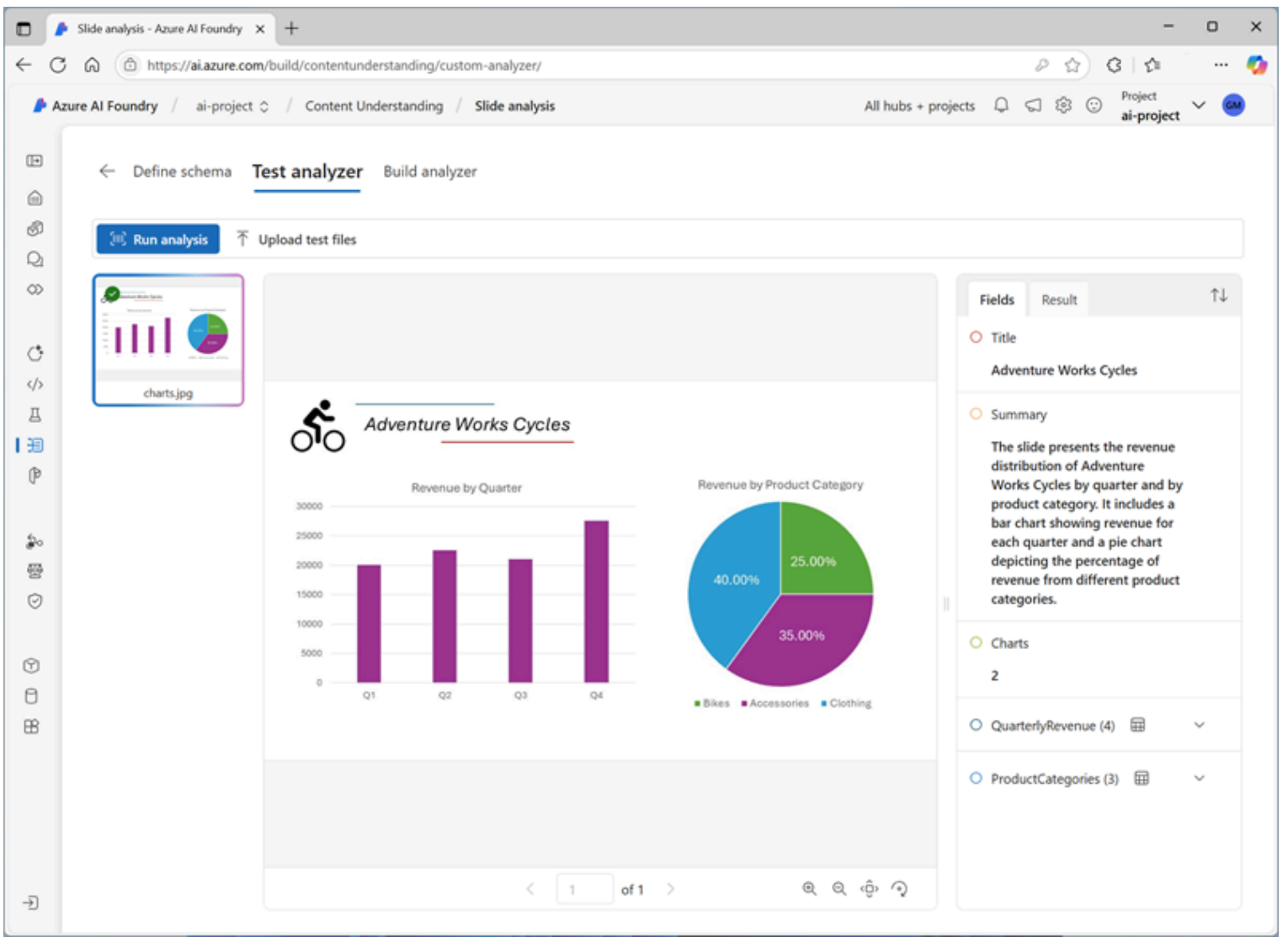


10. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

The slide being analyzed looks like this:



11. Review the analysis results, which should look similar to this:



12. View the details of the fields that were identified in the **Fields** pane, expanding the **QuarterlyRevenue** and **ProductCategories** fields to see the subfield values.

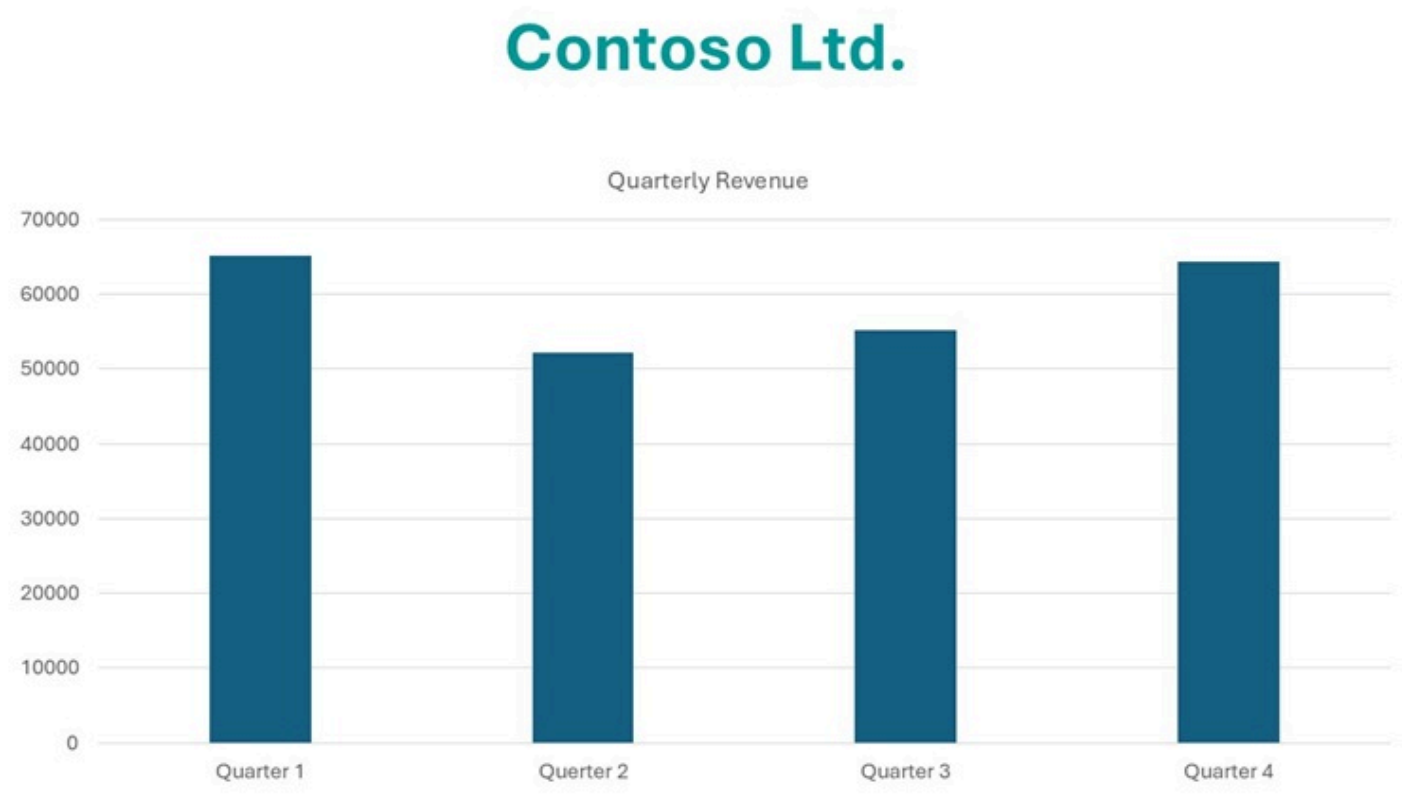
Build and test an analyzer

Now that you have trained a model to extract fields from slides, you can build an analyzer to use with similar slide images.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `slide-analyzer`
    - **Description**: `Slide image analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **slide-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **slide-analyzer** page, select the **Test** tab.

5. Use the **+ Upload test files** button to upload **slide-2.jpg** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the image.

   The slide being analyzed looks like this:



6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the slide image.

   > **Note**: Slide 2 doesn't include a breakfown by product category, so the product category revenue data is not found.

7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content understanding REST interface to call your analyzer.
9. Close the **slide-analyzer** page.

Extract information from a voicemail audio recording

You are going to build an Azure AI Content Understanding analyzer that can extract information from an audio recording of a voicemail message.

Define a schema for audio analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
   - **Task name**: `Voicemail analysis`
   - **Description**: `Extract data from a voicemail recording`
   - **Single file content analysis**: *Selected*
   - **Advanced settings**:
     - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
     - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

   > **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **call-1.mp3** file from the folder where you extracted content files. Then select the **Speech transcript analysis** template and select **Create**.

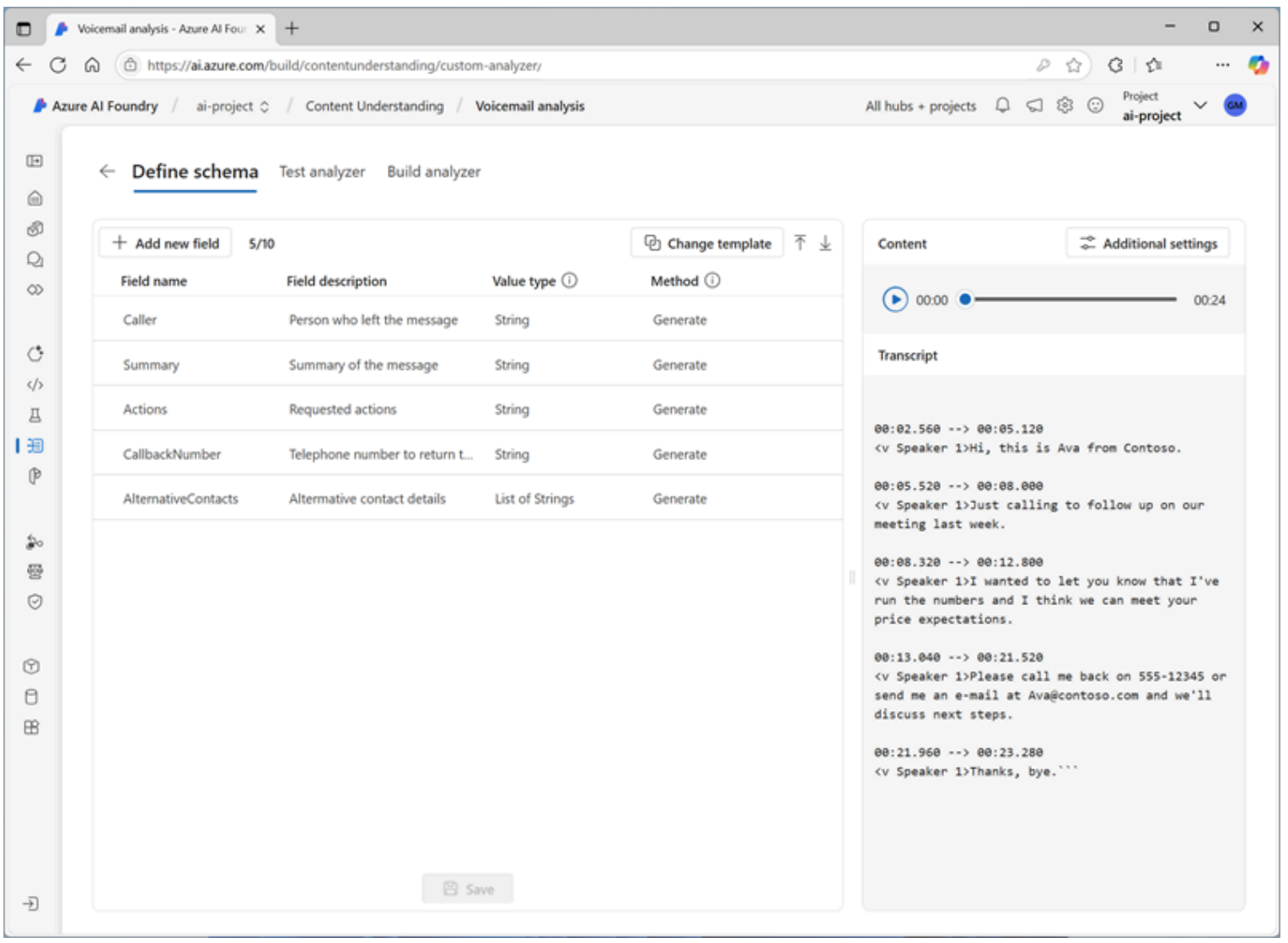
6. In the **Content** pane on the right, select **Get transcription preview** to see a transcription of the recorded message.

   The *Speech transcript analysis* template doesn't include any predefined fields. You must define fields to describe the information you want to extract.

7. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

Field name	Field description	Value type	Method
`Caller`	`Person who left the message`	String	Generate
`Summary`	`Summary of the message`	String	Generate
`Actions`	`Requested actions`	String	Generate
`CallbackNumber`	`Telephone number to return the call`	String	Generate
`AlternativeContacts`	`Alternative contact details`	List of Strings	Generate

8. Verify that your schema looks like this. Then select **Save**.



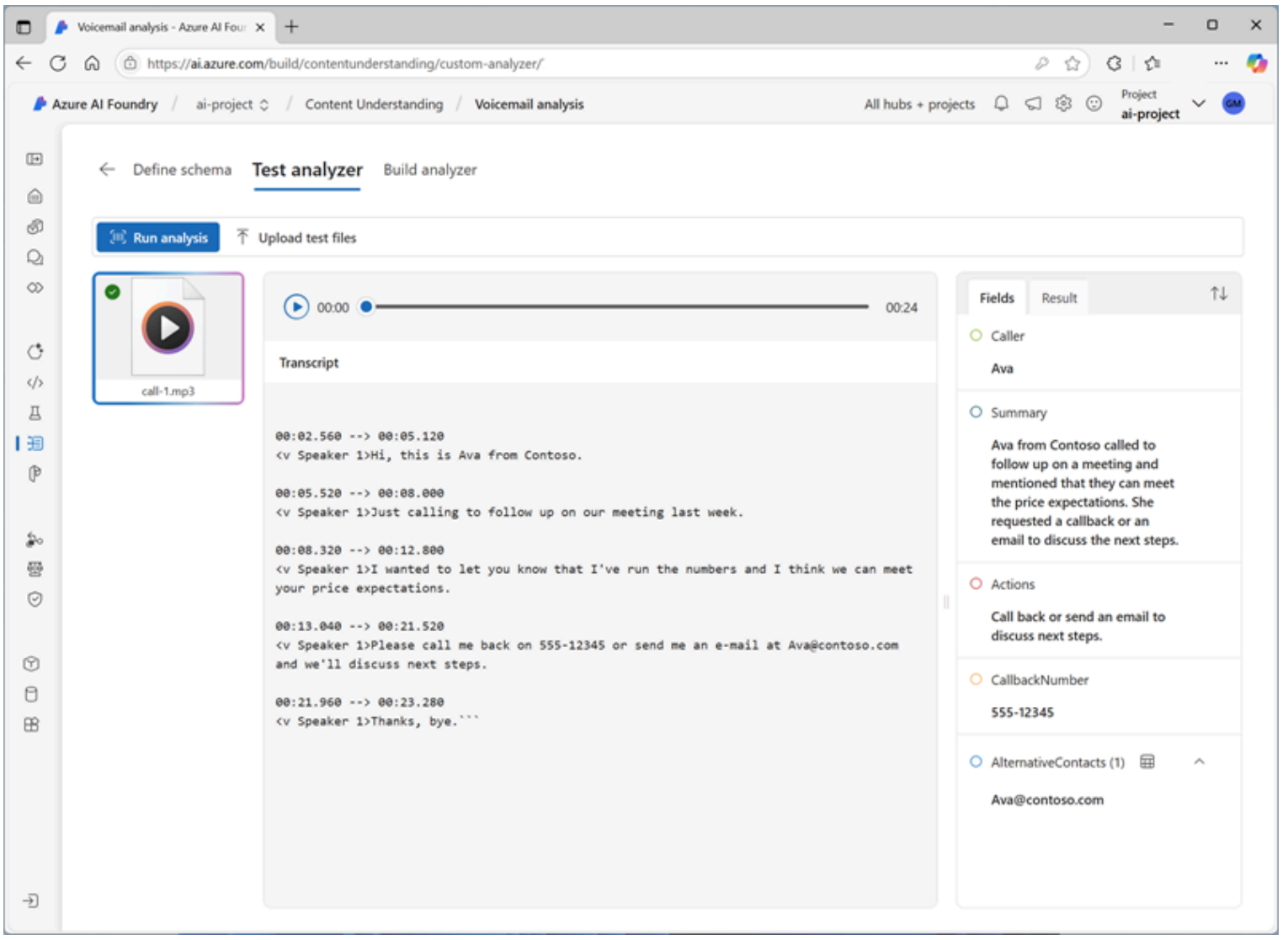
9. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

   Audio analysis can take some time. While you're waiting, you can play the audio file below:

   0:00

   Note: This audio was generated using AI.

10. Review the analysis results, which should look similar to this:



11. View the details of the fields that were identified in the **Fields** pane, expanding the **AlternativeContacts** field to see the listed values.

Build and test an analyzer

Now that you have trained a model to extract fields from voice messages, you can build an analyzer to use with similar audio recordings.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `voicemail-analyzer`
    - **Description**: `Voicemail audio analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **voicemail-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **voicemail-analyzer** page, select the **Test** tab.
5. Use the **+ Upload test files** button to upload **call-2.mp3** from the folder where you extracted the content files, and click on **Run analysis** to extract field data from the audio file.

   Audio analysis can take some time. While you're waiting, you can play the audio file below:

   0:00

   Note: This audio was generated using AI.
6. Review the **Fields** pane, and verify that the analyzer extracted the correct fields from the voice message.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. On the **Code example** tab, view the sample code that you could use to develop a client application that uses the Content understanding REST interface to call your analyzer.
9. Close the **voicemail-analyzer** page.

Extract information from a video conference recording

You are going to build an Azure AI Content Understanding analyzer that can extract information from an video recording of a conference call.

Define a schema for video analysis

1. In the browser tab containing the home page for your Azure AI Foundry project; in the navigation pane on the left, select **Content Understanding**.
2. On the **Content Understanding** page, select the **Custom task** tab at the top.
3. On the Content Understanding custom task page, select **+ Create**, and create a task with the following settings:
    - **Task name**: `Conference call video analysis`
    - **Description**: `Extract data from a video conference recording`
    - **Single file content analysis**: *Selected*
    - **Advanced settings**:
        - **Azure AI services connection**: *The Azure AI Services resource in your Azure AI Foundry hub*
        - **Azure Blob Storage account**: *The default storage account in your Azure AI Foundry hub*
4. Wait for the task to be created.

> **Tip**: If an error accessing storage occurs, wait a minute and try again. Permissions for a new hub may take a few minutes to propagate.

5. On the **Define schema** page, upload the **meeting-1.mp4** file from the folder where you extracted content files. Then select the **Video analysis** template and select **Create**.
6. In the **Content** pane on the right, select **Get transcription preview** to see a transcription of the recorded message.

   The *Video analysis* template extracts data for the video. It doesn't include any predefined fields. You must define fields to describe the information you want to extract.

7. Use **+ Add new field** button to add the following fields, selecting **Save changes** (✓) for each new field:

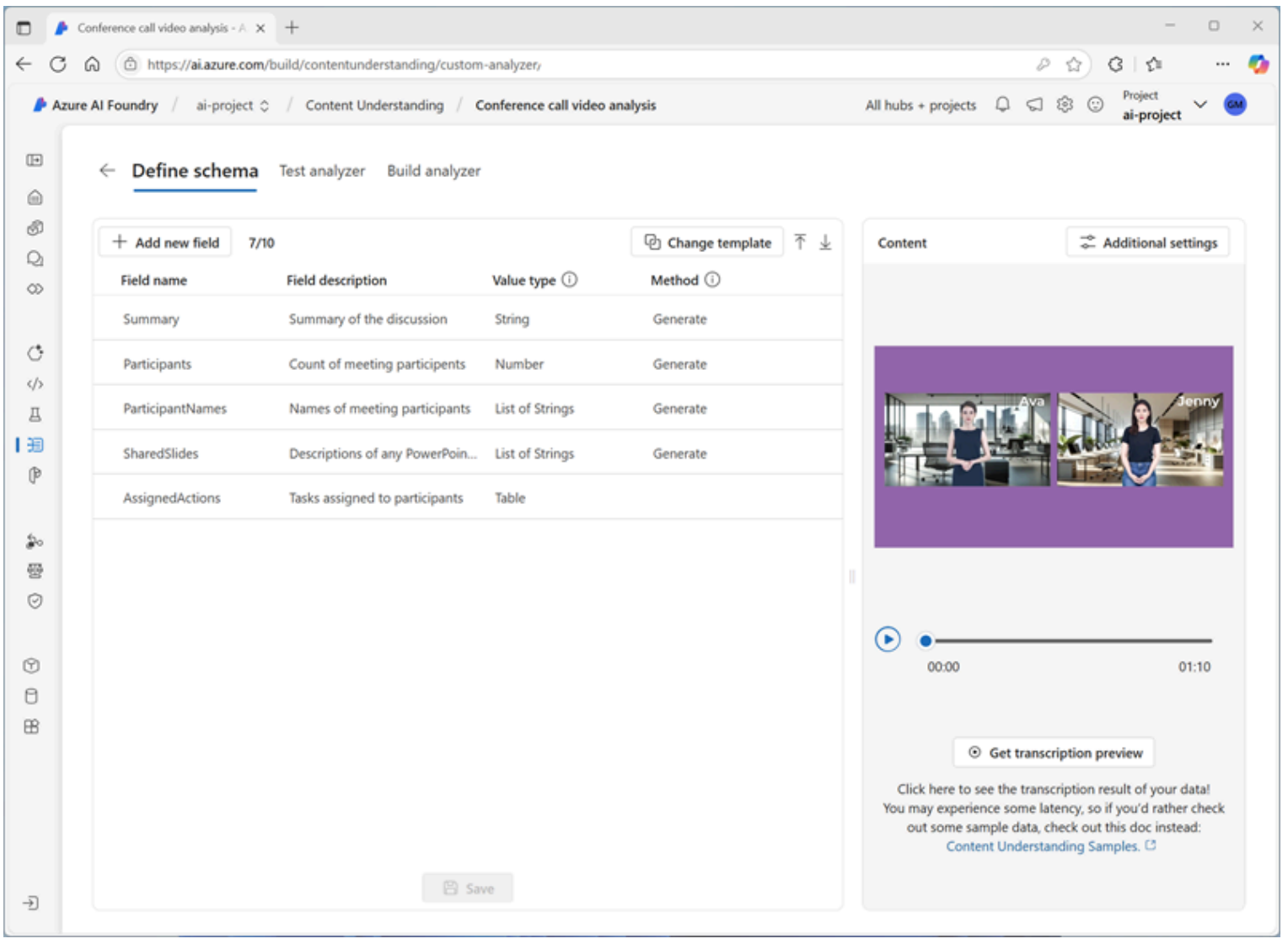
Field name	Field description	Value type	Method
`Summary`	`Summary of the discussion`	String	Generate
`Participants`	`Count of meeting participants`	Integer	Generate
`ParticipantNames`	`Names of meeting participants`	List of Strings	Generate
`SharedSlides`	`Descriptions of any PowerPoint slides presented`	List of Strings	Generate
`AssignedActions`	`Tasks assigned to participants`	Table	

8. When you enter the **AssignedActions** field, in the table of subfields that appears, create the following subfields:

Field name	Field description	Value type	Method
`Task`	`Description of the task`	String	Generate
`AssignedTo`	`Who the task is assigned to`	String	Generate

9. Select **Back** (the arrow icon near the **Add new subfield** button) or ✓ **OK** to return to the top level of your schema, and verify that it looks like this. Then select **Save**.

10. Verify that your schema looks like this. Then select **Save**.



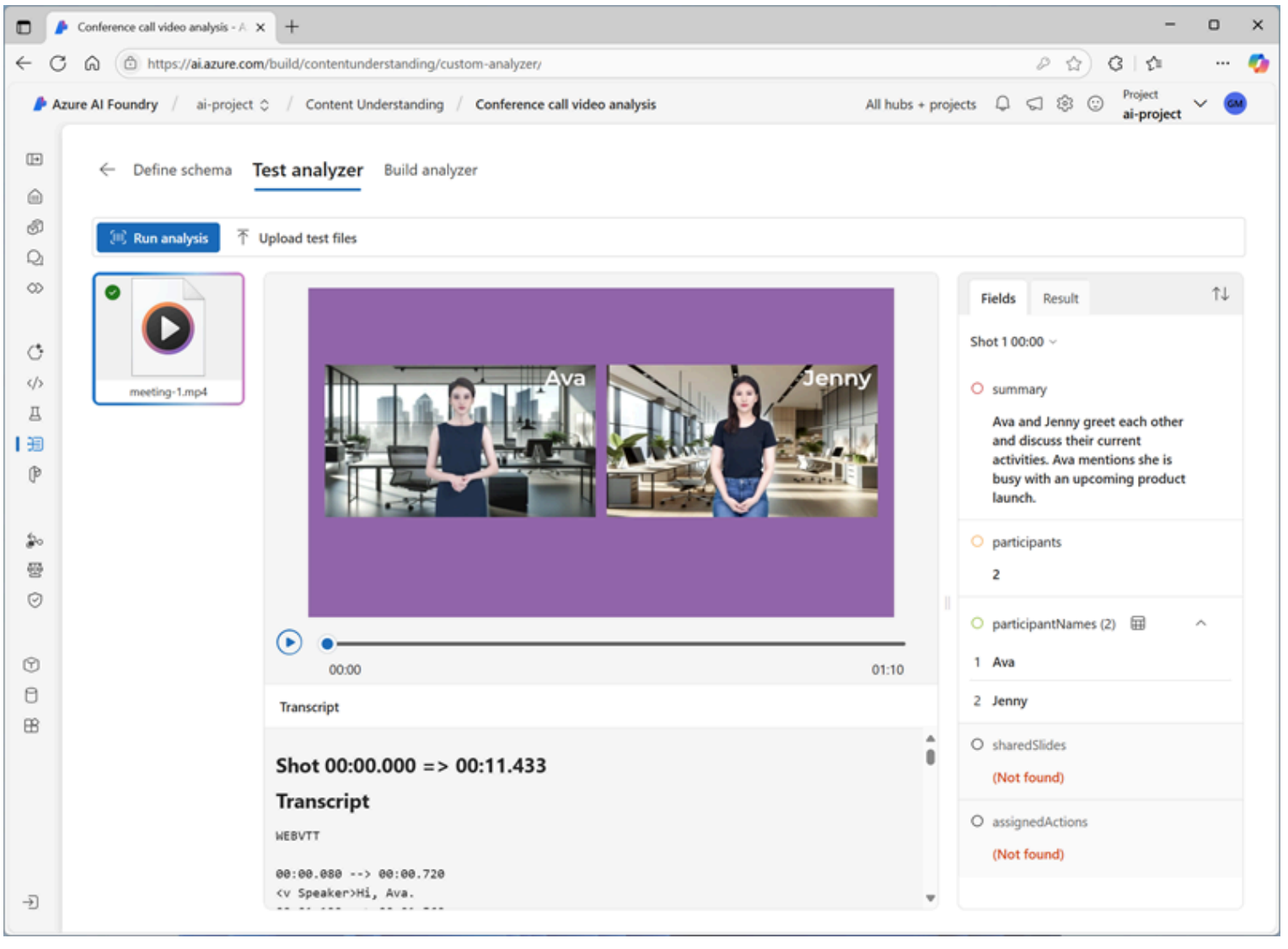
11. On the **Test Analyzer** page, if analysis does not begin automatically, select **Run analysis**. Then wait for analysis to complete.

    Video analysis can take some time. While you're waiting, you can view the video below:

    0:00

    Note: This video was generated using AI.

12. When analysis is complete, review the results, which should look similar to this:



13. In the **Fields** pane, view the extracted data for the video, including the fields you added. View the field values that were generated, expanding list and table fields as necessary.

Build and test an analyzer

Now that you have trained a model to extract fields from conference call recordings, you can build an analyzer to use with similar videos.

1. Select the **Analyzer list** page, and then select **+ Build analyzer** and build a new analyzer with the following properties (typed exactly as shown here):
    - **Name**: `conference-call-analyzer`
    - **Description**: `Conference call video analyzer`
2. Wait for the new analyzer to be ready (use the **Refresh** button to check).
3. When the analyzer has been built, select the **conference-call-analyzer** link. The fields defined in the analyzer's schema will be displayed.
4. In the **conference-call-analyzer** page, select the **Test** tab.
5. Use the **Upload test files** button to upload **meeting-2.mp4** from the folder where you extracted the content files, and run the analysis to extract field data from the audio file.

    Video analysis can take some time. While you're waiting, you can view the video below:

    0:00

    Note: This video was generated using AI.
6. Review the **Fields** pane, and view the fields that the analyzer extracted for the conference call video.
7. Review the **Results** pane to see the JSON response that the analyzer would return to a client application.
8. Close the **conference-call-analyzer** page.

Clean up

If you've finished working with the Content Understanding service, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. In the Azure AI Foundry portal, navigate to your hub, in the overview page, select your project and delete it.
2. In the Azure portal, delete the resource group you created in this exercise.

Create an Azure AI Content Understanding client application

Use the **Azure AI Content Understanding *REST API*** for **multimodal content analysis and information extraction**.

Learning objectives

After completing this module, you will be able to:

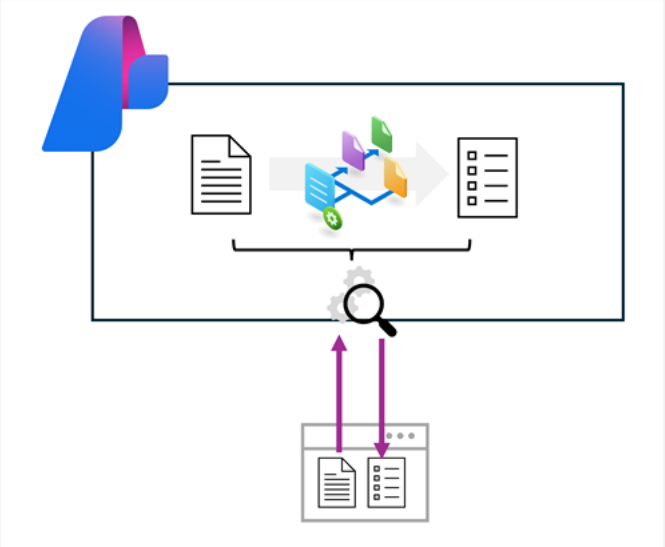
- Use the Azure AI Content Understanding REST API to **build a content analyzer**.
- Use the Azure AI Content Understanding REST API to **consume a content analyzer**.

Introduction

Azure AI Content Understanding is a multimodal service that simplifies the creation of AI-powered analyzers that can extract information from multiple content formats, including *documents, images, audio files, and videos*.

> Tip: To learn how to build Azure AI Content Understanding analyzers, complete the Create a multimodal analysis solution with Azure AI Content Understanding module.

You can develop client applications that use **Azure AI Content Understanding analyzers** by using the **Azure AI Content Understanding REST API**; which is the focus of this module.



In this module, you'll learn how to write code that **uses the REST API to submit a content file to an analyzer and process the results**.

> Note: Azure AI Content Understanding is currently in public preview. Details described in this module are subject to change.

Prepare to use the AI Content Understanding REST API

Before you can use the **Azure AI Content Understanding REST API**, you need an **Azure AI services multi-services resource** in your Azure subscription. You can provision this resource in the following ways:

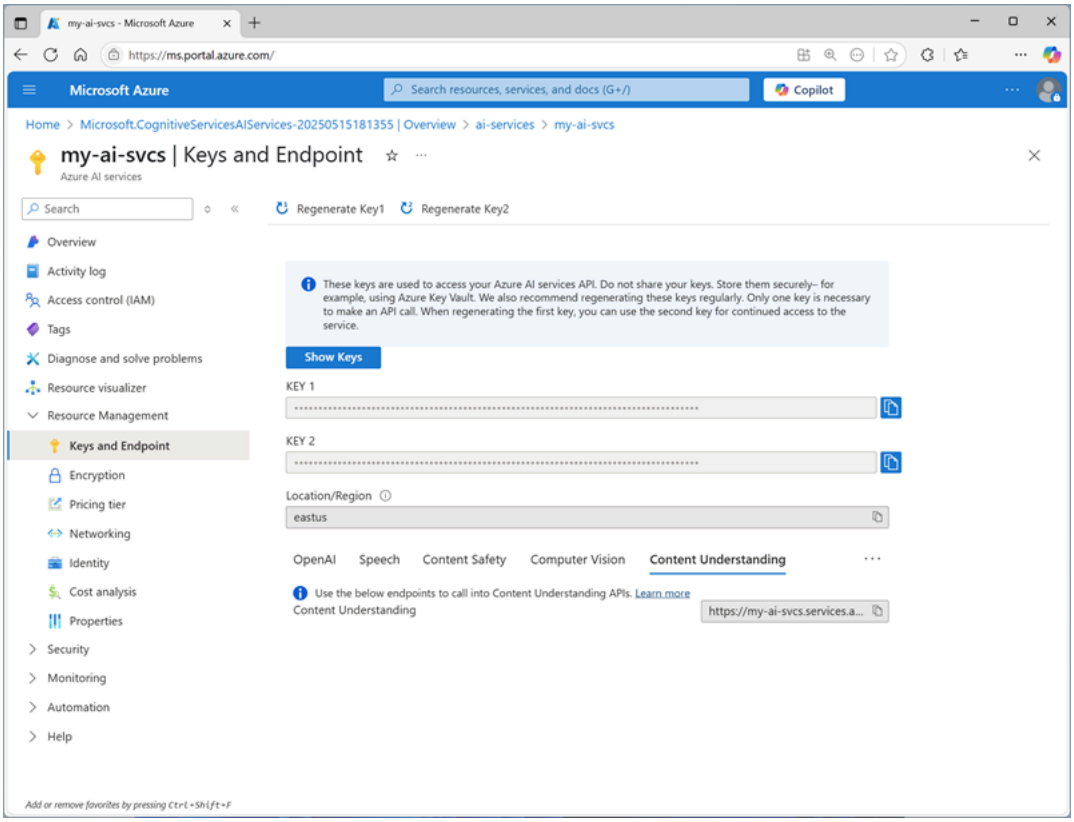
- Create an **Azure AI services** resource in the Azure portal.
- Create an **Azure AI Foundry** hub, which includes an Azure AI services resource by default.

> Tip: Creating an Azure AI Foundry hub enables you to work in an Azure AI Foundry project, in which you can use visual tools to create and manage Azure AI Content Understanding schemas and analyzers.

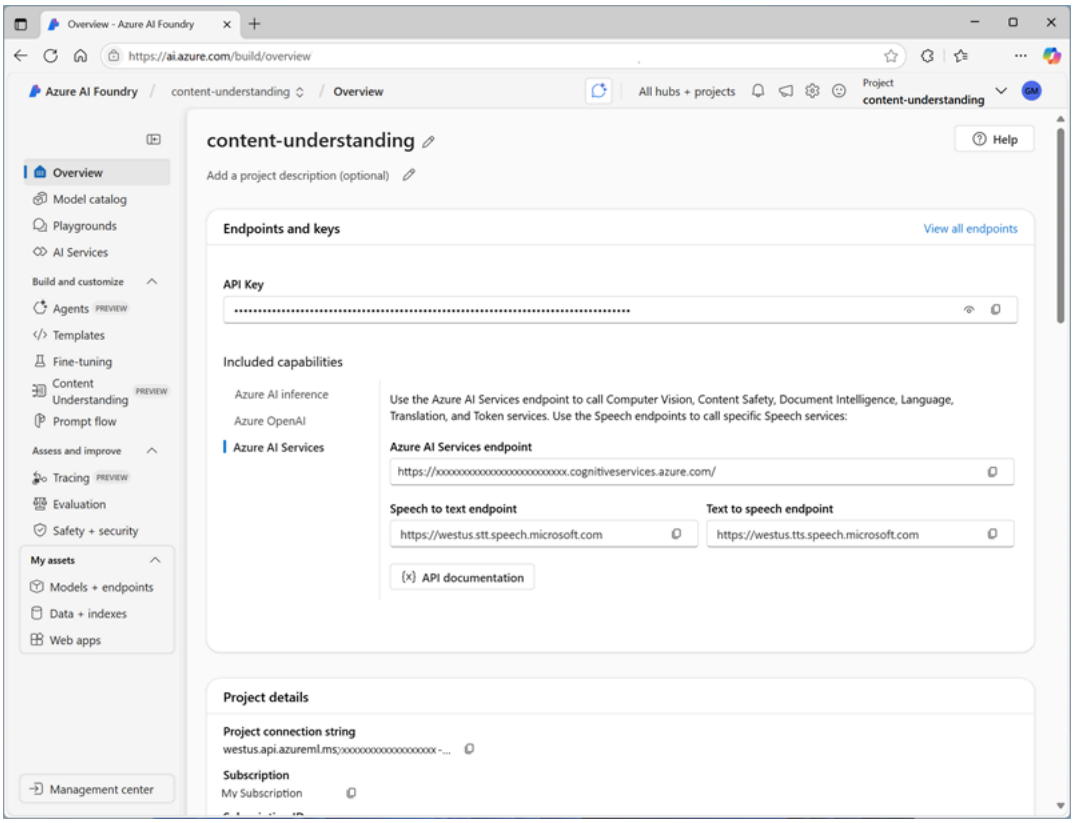
After you've provisioned an Azure AI services resource, you need the following information to connect to the **Azure AI Content Understanding REST API** from a client application:

- The Azure AI services resource **endpoint**
- One of the **API keys** associated with the endpoint.

You can obtain these values from the Azure portal, as shown in the following image:



If you're working within an Azure AI Foundry project, you can find the **endpoint** and **key** for the associated *Azure AI services resource* in the *Azure AI Foundry portal*, as shown in the following image:



When working in an Azure AI Foundry project, you can also write code that uses the **Azure AI Foundry SDK** to connect to the project using **Microsoft Entra ID authentication**, and retrieve the connection details for the Azure AI services resource; including the endpoint and key.

> Tip: To learn more about programming with the Azure AI Foundry SDK, complete the Develop an AI app with the Azure AI Foundry SDK module.

Create a Content Understanding analyzer

In most scenarios, you should consider creating and testing analyzers using the **visual interface in the Azure AI Foundry portal**. However, in some cases you might want to create an analyzer by submitting **a JSON definition of the schema** for your desired content fields to the **REST API**.

Defining a schema for an analyzer

Analyzers are based on **schemas that define the fields you want to extract or generate from a content file**. At its simplest, a schema is a set of fields, which can be specified in a JSON document, as shown in this example of an analyzer definition:

```
{     "description": "Simple business card",     "baseAnalyzerId": "prebuilt-documentAnalyzer",     "config": {         "returnDetails": true     }
```

This example of a custom analyzer schema is based on the **pre-built document analyzer**, and describes *two fields* that you would expect to find on a business card: *ContactName* and *EmailAddress*. Both fields are defined as string data types, and are expected to be extracted from a document (in other words, the string values are expected to exist in the document so they can be "read"; rather than being fields that can be generated by inferring information about the document).

> Note: This example is deliberately simple, with the minimal information needed to create a working analyzer. In reality, the schema would likely include more fields of different types, and the analyzer definition would include more configuration settings. The JSON might even include a sample document. See the Azure AI Content Understanding REST API documentation for more details.

Using the REST API to create an analyzer

With your analyzer definition in place, you can use the REST API to submit it to Azure AI Content Understanding to be created. The JSON data is submitted as a `PUT` request to the **endpoint** with the **API key** *in the request header* to start the analyzer creation operation.

The response from the `PUT` request includes a Operation-Location in the header, which provides a **callback URL** that you can use to **check on the status of the request** by submitting a `GET` request.

You can use any HTTP-capable client tool or language to submit the request. For example, the following Python code submits a request to create an analyzer based on the contents of a file named `card.json` (which is assumed to contain the JSON definition described previously)

```python
import json
import requests

# Get the buisness card schema
with open("card.json", "r") as file:
    schema_json = json.load(file)

# Use a PUT request to submit the schema for a new analyzer
analyzer_name = "business_card_analyser"

headers = {
    "Ocp-Apim-Subscription-Key": "<YOUR_API_KEY>",
    "Content-Type": "application/json"}

url = f"{<YOUR_ENDPOINT>}/contentunderstanding/analyzers/{analyzer_name}?api-version=2025-05-01-preview"

response = requests.put(url, headers=headers, data=json.dumps(schema_json))

# Get the response and extract the ID assigned to the operation
callback_url = response.headers["Operation-Location"]

# Use a GET request to check the status of the operation
result_response = requests.get(callback_url, headers=headers)

# Keep polling until the operation is complete
status = result_response.json().get("status")
while status == "Running":
    result_response = requests.get(callback_url, headers=headers)
    status = result_response.json().get("status")

print("Done!")
```

Analyze content

To analyze the contents of a file, you can use the **Azure AI Content Understanding REST API** to submit it to the **endpoint** using a `POST` request. You can specify the **content as a URL** (for a file hosted in an Internet-accessible location) or as the **binary contents of the file** (for example, a .pdf document, a .png image, an .mp3 audio file, or an .mp4 video file). The request header must include the **API key**, and the **endpoint** address for the analyze request includes the analyzer to be used.

As with the request to create an analyzer, the analyze request starts an **asynchronous** operation. The `POST` request returns a unique operation ID, which you can then use in a `GET` request to **check the status of the analysis operation**.

For example, suppose you want to use the *business card analyzer* discussed previously to extract the name and email address from the following scanned business card image:

The following Python code submits a request for analysis, and then polls the service until the operation is complete and the results are returned.

```python
import json
import requests

# Read the image data
with open("business-card.png", "rb") as file:
        image_data = file.read()

## Use a POST request to submit the image data to the analyzer
analyzer_name = "business_card_analyser"

headers = {
        "Ocp-Apim-Subscription-Key": "<YOUR_API_KEY>",
        "Content-Type": "application/octet-stream"}

url = f"{<YOUR_ENDPOINT>}/contentunderstanding/analyzers/{analyzer_name}:analyze?
api-version=2025-05-01-preview"

response = requests.post(url, headers=headers, data=image_data)

# Get the response and extract the ID assigned to the analysis operation
response_json = response.json()
id_value = response_json.get("id")

# Use a GET request to check the status of the analysis operation
result_url = f"{<YOUR_ENDPOINT>}/contentunderstanding/analyzerResults/{id_value}?
api-version=2025-05-01-preview"

result_response = requests.get(result_url, headers=headers)

# Keep polling until the analysis is complete
status = result_response.json().get("status")
while status == "Running":
        result_response = requests.get(result_url, headers=headers)
        status = result_response.json().get("status")

# Get the analysis results
if status == "Succeeded":
    result_json = result_response.json()
```

Processing analysis results

The results in the response JSON depend on:

- The **kind of content** the *analyzer* is designed to analyze (for example, document, video, image, or audio).
- The **schema** for the analyzer.
- The **contents of the file** that was analyzed.

For example, the response from the document-based business card analyzer when analyzing the business card described previously contain:

- The extracted fields
- The *optical character recognition (OCR)* layout of the document, including locations of lines of text, individual words, and paragraphs on each page.

Here's the complete JSON response for the business card analysis:

- Confidence score
- Span
- Content
- Source

Your application must typically parse the JSON to retrieve field values. For example, the following python code extracts all of the string values:

```python
# (continued from previous code example)

# Iterate through the fields and extract the names and type-specific values
contents = result_json["result"]["contents"]
for content in contents:
    if "fields" in content:
        fields = content["fields"]
        for field_name, field_data in fields.items():
            if field_data['type'] == "string":
                print(f"{field_name}: {field_data['valueString']}")
```

The output from this code is shown here:

```
ContactName: John Smith
EmailAddress: john@contoso.com
```

Exercise - Develop a Content Understanding client application

Now it's your turn to build your own Content Understanding client application!

In this exercise, you use the Azure AI Content Understanding REST API to extract information from content by submitting a file to an analyzer.

Develop a Content Understanding client application

In this exercise, you use Azure AI Content Understanding to create an analyzer that extracts information from business cards. You'll then develop a client application that uses the analyzer to extract contact details from scanned business cards.

Module assessment

1. What configuration values are needed to use the Azure AI Content Understanding REST API? **The endpoint and key for the Azure AI service**.
2. What must be specified when calling the analyze method to extract fields from content? **The name of the analyzer**.
3. How are the extracted fields returned? **As type-specific values**.

Summary

Azure AI Content Understanding is a **multimodal AI service** that enables you to **extract information from many different kinds of content**. The REST API for the service enables you to create client applications that analyze content to extract and generate field values.

> Note: For more information about Azure AI Content Understanding, see Azure AI Content Understanding documentation.

Develop a Content Understanding client application

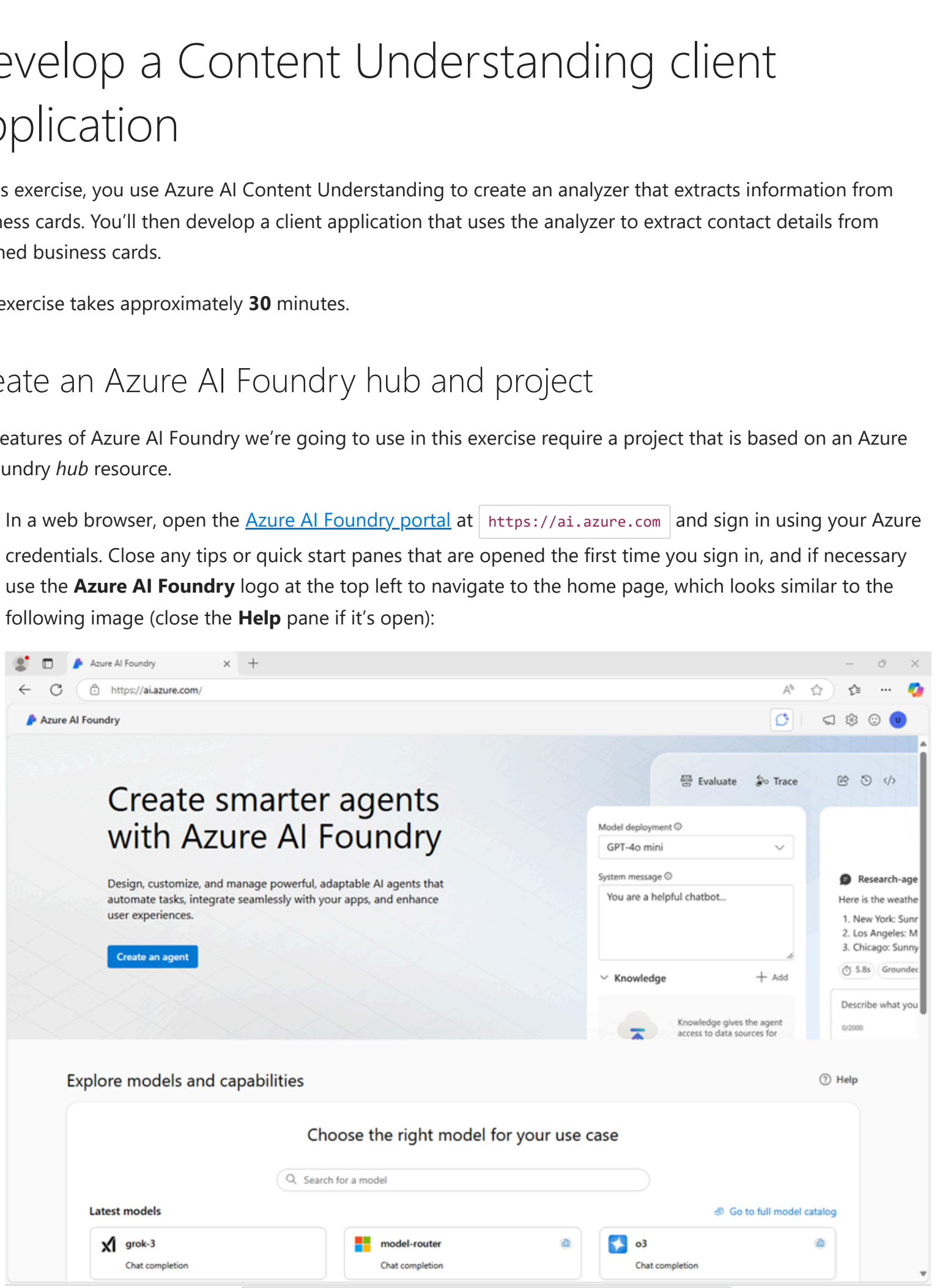
In this exercise, you use Azure AI Content Understanding to create an analyzer that extracts information from business cards. You'll then develop a client application that uses the analyzer to extract contact details from scanned business cards.

This exercise takes approximately **30** minutes.

Create an Azure AI Foundry hub and project

The features of Azure AI Foundry we're going to use in this exercise require a project that is based on an Azure AI Foundry *hub* resource.

1. In a web browser, open the Azure AI Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Azure AI Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):



2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Hub name**: A valid name for your hub
    - **Location**: Choose one of the following locations:*
        - Australia East
        - Sweden Central
        - West US

> *At the time of writing, Azure AI Content understanding is only available in these regions.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created, and then navigate to your project overview page.

Use the REST API to create a Content Understanding analyzer

You're going to use the REST API to create an analyzer that can extract information from images of business cards.

1. Open a new browser tab (keeping the Azure AI Foundry portal open in the existing tab). Then in the new tab, browse to the Azure portal at `https://portal.azure.com`; signing in with your Azure credentials if prompted.

   Close any welcome notifications to see the Azure portal home page.

2. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment with no storage in your subscription.

   The cloud shell provides a command-line interface in a pane at the bottom of the Azure portal. You can resize or maximize this pane to make it easier to work in.

   > **Tip**: Resize the pane so you can work mostly in the cloud shell but still see the keys and endpoint in the Azure portal page - you'll need to copy them into your code.

3. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

4. In the cloud shell pane, enter the following commands to clone the GitHub repo containing the code files for this exercise (type the command, or copy it to the clipboard and then right-click in the command line and paste as plain text):

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you enter commands into the cloudshell, the output may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

5. After the repo has been cloned, navigate to the folder containing the code files for your app:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/content-app
   ls -a -l
   ```

   The folder contains two scanned business card images as well as the Python code files you need to build your app.

6. In the cloud shell command-line pane, enter the following command to install the libraries you'll use:

   Code Copy

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt
```

7. Enter the following command to edit the configuration file that has been provided:

Code Copy

```
code .env
```

The file is opened in a code editor.

8. In the code file, replace the **YOUR_ENDPOINT** and **YOUR_KEY** placeholders with the your Azure AI services endpoint and either of its keys (copied from the Azure portal), and ensure that **ANALYZER_NAME** is set to `business-card-analyzer`.

9. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

> **Tip**: You can maximize the cloud shell pane now.

10. In the cloud shell command line, enter the following command to view the **biz-card.json** JSON file that has been provided:

Code Copy

```
cat biz-card.json
```

Scroll the cloud shell pane to view the JSON in the file, which defines an analyzer schema for a business card.

11. When you've viewed the JSON file for the analyzer, enter the following command to edit the **create-analyzer.py** Python code file that has been provided:

Code Copy

```
code create-analyzer.py
```

The Python code file is opened in a code editor.

12. Review the code, which:
    - Loads the analyzer schema from **biz-card.json** file.
    - Retrieves the endpoint, key, and analyzer name from the environment configuration file.
    - Calls a function named **create_analyzer**, which is currently not implemented

13. In the **create_analyzer** function, find the comment **Create a Content Understanding analyzer** and add the following code (being careful to maintain the correct indentation):

Code Copy

```
# Create a Content Understanding analyzer
print (f"Creating {analyzer}")

# Set the API version
CU_VERSION = "2025-05-01-preview"

# initiate the analyzer creation operation
headers = {
     "Ocp-Apim-Subscription-Key": key,
     "Content-Type": "application/json"}

url = f"{endpoint}/contentunderstanding/analyzers/{analyzer}?api-version={CU_VERSION}"

# Delete the analyzer if it already exists
response = requests.delete(url, headers=headers)
print(response.status_code)
time.sleep(1)

# Now create it
response = requests.put(url, headers=headers, data=(schema))
print(response.status_code)

# Get the response and extract the callback URL
callback_url = response.headers["Operation-Location"]

# Check the status of the operation
time.sleep(1)
result_response = requests.get(callback_url, headers=headers)

# Keep polling until the operation is no longer running
status = result_response.json().get("status")
while status == "Running":
     time.sleep(1)
     result_response = requests.get(callback_url, headers=headers)
     status = result_response.json().get("status")

result = result_response.json().get("status")
print(result)
if result == "Succeeded":
     print(f"Analyzer '{analyzer}' created successfully.")
else:
     print("Analyzer creation failed.")
     print(result_response.json())
```

14. Review the code you added, which:
    - Creates appropriate headers for the REST requests
    - Submits an HTTP *DELETE* request to delete the analyzer if it already exists.
    - Submits an HTTP *PUT* request to initiate the creation of the analyzer.
    - Checks the response to retrieve the *Operation-Location* callback URL.
    - Repeatedly submits an HTTP *GET* request to the callback URL to check the operation status until it is no longer running.
    - Confirms success (or failure) of the operation to the user.

> **Note**: The code includes some deliberate time delays to avoid exceeding the request rate limit foe the service.

15. Use the **CTRL+S** command to save the code changes, but keep the code editor pane open in case you need to correct any errors in the code. Resize the panes so you can clearly see the command line pane.

16. In the cloud shell command line pane, enter the following command to run the Python code:

```
python create-analyzer.py
```

17. Review the output from the program, which should hopefully indicate that the analyzer has been created.

Use the REST API to analyze content

Now that you've created an analyzer, you can consume it from a client application through the Content Understanding REST API.

1. In the cloud shell command line, enter the following command to edit the **read-card.py** Python code file that has been provided:

```
code read-card.py
```

   The Python code file is opened in a code editor:

2. Review the code, which:
   - Identifies the image file to be analyzed, with a default of **biz-card-1.png**.
   - Retrieves the endpoint and key for your Azure AI Services resource from the project (using the Azure credentials from the current cloud shell session to authenticate).
   - Calls a function named **analyze_card**, which is currently not implemented

3. In the **analyze_card** function, find the comment **Use Content Understanding to analyze the image** and add the following code (being careful to maintain the correct indentation):

```
# Use Content Understanding to analyze the image
print (f"Analyzing {image_file}")

# Set the API version
CU_VERSION = "2025-05-01-preview"

# Read the image data
with open(image_file, "rb") as file:
     image_data = file.read()

## Use a POST request to submit the image data to the analyzer
print("Submitting request...")
headers = {
     "Ocp-Apim-Subscription-Key": key,
     "Content-Type": "application/octet-stream"}
url = f'{endpoint}/contentunderstanding/analyzers/{analyzer}:analyze?api-version=
{CU_VERSION}'
response = requests.post(url, headers=headers, data=image_data)

# Get the response and extract the ID assigned to the analysis operation
print(response.status_code)
response_json = response.json()
id_value = response_json.get("id")

# Use a GET request to check the status of the analysis operation
print ('Getting results...')
time.sleep(1)
result_url = f'{endpoint}/contentunderstanding/analyzerResults/{id_value}?api-version=
{CU_VERSION}'
result_response = requests.get(result_url, headers=headers)
print(result_response.status_code)

# Keep polling until the analysis is complete
status = result_response.json().get("status")
while status == "Running":
     time.sleep(1)
     result_response = requests.get(result_url, headers=headers)
     status = result_response.json().get("status")

# Process the analysis results
if status == "Succeeded":
     print("Analysis succeeded:\n")
     result_json = result_response.json()
     output_file = "results.json"
     with open(output_file, "w") as json_file:
         json.dump(result_json, json_file, indent=4)
         print(f"Response saved in {output_file}\n")

     # Iterate through the fields and extract the names and type-specific values
     contents = result_json["result"]["contents"]
     for content in contents:
         if "fields" in content:
             fields = content["fields"]
             for field_name, field_data in fields.items():
                 if field_data['type'] == "string":
                     print(f"{field_name}: {field_data['valueString']}")
                 elif field_data['type'] == "number":
                     print(f"{field_name}: {field_data['valueNumber']}")
```

```
        elif field_data['type'] == "integer":
            print(f"{field_name}: {field_data['valueInteger']}")
        elif field_data['type'] == "date":
            print(f"{field_name}: {field_data['valueDate']}")
        elif field_data['type'] == "time":
            print(f"{field_name}: {field_data['valueTime']}")
        elif field_data['type'] == "array":
            print(f"{field_name}: {field_data['valueArray']}")
```

4. Review the code you added, which:
    - Reads the contents of the image file
    - Sets the version of the Content Understanding REST API to be used
    - Submits an HTTP *POST* request to your Content Understanding endpoint, instructing the to analyze the image.
    - Checks the response from the operation to retrieve an ID for the analysis operation.
    - Repeatedly submits an HTTP *GET* request to your Content Understanding endpoint to check the operation status until it is no longer running.
    - If the operation has succeeded, saves the JSON response, and then parses the JSON and displays the values retrieved for each type-specific field.

> **Note**: In our simple business card schema, all of the fields are strings. The code here illustrates the need to check the type of each field so that you can extract values of different types from a more complex schema.

5. Use the **CTRL+S** command to save the code changes, but keep the code editor pane open in case you need to correct any errors in the code. Resize the panes so you can clearly see the command line pane.

6. In the cloud shell command line pane, enter the following command to run the Python code:

Code Copy

```
python read-card.py biz-card-1.png
```

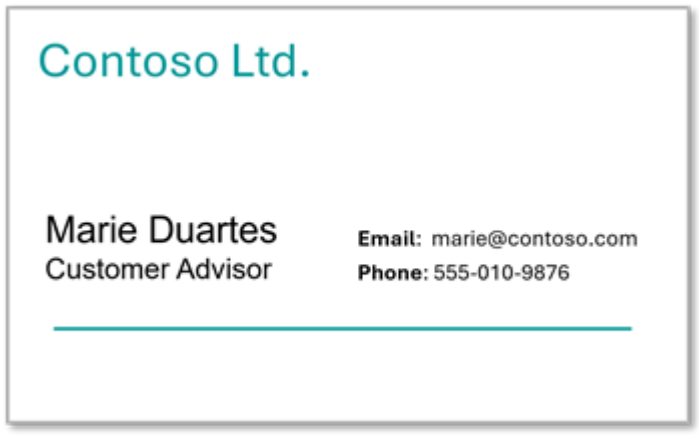
7. Review the output from the program, which should show the values for the fields in the following business card:

8. Use the following command to run the program with a different business card:

Code Copy

```
python read-card.py biz-card-2.png
```

9. Review the results, which should reflect the values in this business card:



10. In the cloud shell command line pane, use the following command to view the full JSON response that was returned:

```
cat results.json
```

Scroll to view the JSON.

Clean up

If you've finished working with the Content Understanding service, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. In the Azure portal, delete the resources you created in this exercise.

Use prebuilt Document intelligence models

Learn what data you can analyze by choosing **prebuilt Forms Analyzer models** and how to deploy these models in a **Document intelligence solution**.

Learning objectives

In this module, you'll learn to:

- Identify business problems that you can solve by using prebuilt models in **Forms Analyzer**.
- Analyze forms by using the **General Document, Read, and Layout models**.
- Analyze forms by using **financial, ID, and tax prebuilt models**.

Introduction

Many forms and documents that businesses handle are common across disparate companies in different sectors. For example, most companies use invoices and receipts. Microsoft **Azure AI Document Intelligence** includes **prebuilt models** so you can handle common document types easily.

Imagine you conduct polls for private companies and political parties. Participants submit their responses as paper forms or as online PDFs. You've decided to deploy **Azure AI Document Intelligence** to streamline data entry. You want to know if you can use the prebuilt models to generate meaningful data from your forms.

In this module, you'll learn about the capabilities of the **prebuilt models in Azure AI Document Intelligence** and how to use them.

Understand prebuilt models

Prebuilt models in Azure AI Document Intelligence enable you to **extract data from common forms and documents without training your own models**.

In your polling company, polling forms are unique to each survey project, but you also use invoices and receipts to record financial transactions and you have many unstructured documents. You want to know how much work is required to extract names, addresses, amounts, and other information from these documents.

Here, you learn how prebuilt models can help you analyze common document types.

What are prebuilt models?

The general approach used in AI solutions is to provide a large quantity of sample data and then train an optimized model by trying different data features, parameters, and statistical treatments. The combination that best predicts the values that interest you constitute the trained model, and you can use this model to predict values from new data.

Many of the forms that businesses use from day to day are of a few common types. For example, most businesses issue or receive invoices and receipts. Any business that has employees in the United States must use the W-2 tax declaration form. Also you often have more general documents that you might want to extract data from. For these cases, Microsoft helps you by providing prebuilt models. **Prebuilt models are already trained on large numbers of their target form type**.

If you want to use Document Intelligence to extract data from one of these common forms or documents, you can choose to use a prebuilt model and you don't have to train your own. Because Microsoft trains these models on a large corpus of examples, you can expect them to provide accurate and reliable results when dealing with their intended forms.

Several of the prebuilt models are trained on specific form types:

- **Invoice** model. Extracts common fields and their values from invoices.
- **Receipt** model. Extracts common fields and their values from receipts.
- **US Tax** model. Unified US tax model that can extract from forms such as W-2, 1098, 1099, and 1040.
- **ID document** model. Extracts common fields and their values from US drivers' licenses, European Union IDs and drivers license, and international passports.
- **Business card** model. Extracts common fields and their values from business cards.
- **Health insurance card** model. Extracts common fields and their values from health insurance cards.
- **Marriage certificate**. Extracts information from marriage certificates.
- **Credit/Debit card** model. Extracts common information from bank cards.
- **Mortgage documents**. Extracts information from mortgage closing disclosure, Uniform Residential Loan Application (Form 1003), Appraisal (Form 1004), Validation of Employment (Form 1005), and Uniform Underwriting and Transmittal Summary (Form 1008).
- **Bank statement** model. Extracts account information including beginning and ending balances, transaction details from bank statements.
- **Pay Stub** model. Extracts wages, hours, deductions, net pay, and other common pay stub fields.
- **Check** model. Extracts payee, amount, date, and other relevant information from checks.

The other models are designed to extract values from documents with less specific structures:

- **Read model**. Extracts text and languages from documents.
- **General document model**. Extract text, keys, values, entities, and selection marks from documents.
- **Layout model**. Extracts text and structure information from documents.

Features of prebuilt models

The prebuilt models are designed to extract different types of data from the documents and forms users submit. To select the right model for your requirements, you must understand these features:

- **Text extraction**. All the prebuilt models extract lines of text and words from hand-written and printed text.
- **Key-value pairs**. Many models extract spans of text within a document that identify a label or key and its response or value as key-values pairs. For example, *a typical key might be Weight and its value might be 31 kg.*
- **Entities**. Text that includes common, more complex data structures can be extracted as entities. **Entity types include people, locations, and dates**.
- **Selection marks**. Some models extract spans of text that indicate a choice as selection marks. These **marks include radio buttons and check boxes**.
- **Tables**. Many models can extract tables in scanned forms included the data contained in cells, the numbers of columns and rows, and column and row headings. Tables with merged cells are supported.
- **Fields**. Models trained for a specific form type identify the values of a fixed set of fields. For example, the Invoice model includes **CustomerName** and **InvoiceTotal** fields.

Also consider that prebuilt models are designed for and trained on generic document and form types. If you have an industry-specific or unique form type that you use often, you might be able to obtain more reliable and predictable results by using a custom model. However, **custom models take time to develop because you must invest the time and resources to train them on example forms before you can use it**. *The larger the number of example forms you provide for training, the better the model is at predicting form content accurately*.

Input requirements

The prebuilt models are flexible but you can help them to return accurate and helpful results by **submitting one clear photo or high-quality scan for each document**.

You must also comply with these requirements when you submit a form for analysis:

- The file must be in **JPEG, PNG, BMP, TIFF, or PDF** format. Additionally, the Read model can accept Microsoft Office files.
- The file must be smaller than **500 MB for the standard tier**, and **4 MB for the free tier**.
- Images must have dimensions **between 50 x 50 pixels and 10,000 x 10,000 pixels**.
- PDF documents must have dimensions **less than 17 x 17 inches or A3 paper size**.
- PDF documents must **not be protected with a password**.

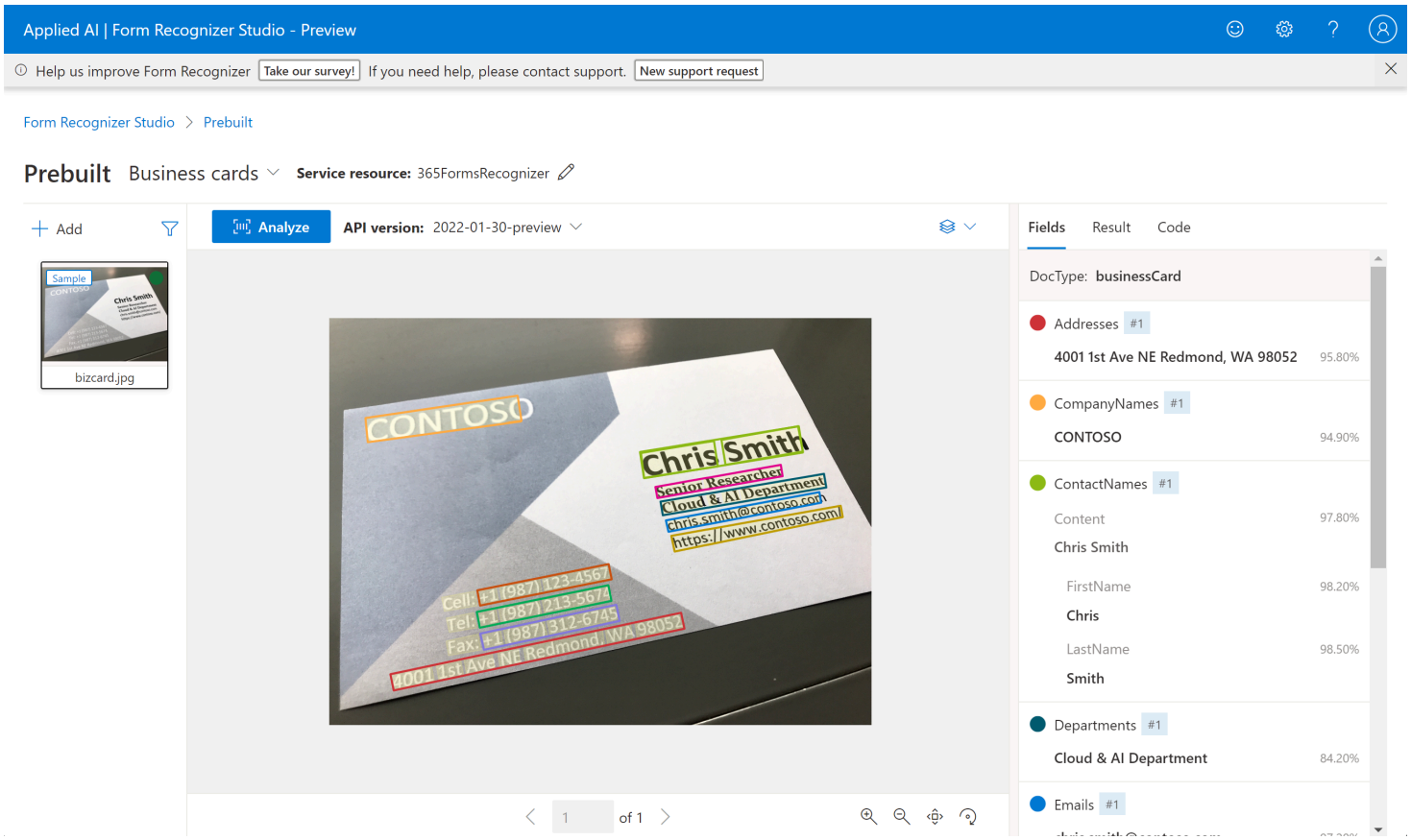
> Note: If you can, submit *text-embedded PDF files* because they eliminate errors in character recognition.

PDF and TIFF files can have any number of pages but, in the **standard tier**, only the **first 2,000 pages** are analyzed. In the **free tier**, only the **first two pages** are analyzed.

Try out prebuilt models with Azure AI Document Intelligence Studio

Azure AI Document Intelligence is designed as a web service you can call using code in your custom applications. However, it's often helpful to explore the models and how they behave with your forms visually. You can perform such experiments by using Azure AI Document Intelligence Studio and use the experience to help design and write your code.

You can choose any of the prebuilt models in Azure AI Document Intelligence Studio. Microsoft provides some sample documents for use with each model or you can add your own documents and analyze them.



Calling prebuilt models by using APIs

Because **Azure AI Document Intelligence implements RESTful web services**, you can use web service calls from any language that supports them. However, when you use Microsoft's Azure AI Document Intelligence APIs, security and session management is simplified and you have to write less code.

Azure AI Document Intelligence is available for:

- C# and other .NET languages.

- Java.
- Python.
- JavaScript.

Whenever you want to call Azure AI Document Intelligence, you must start by **connecting and authenticating with the service in your Azure subscription**. To make that connection, you need:

- The service **endpoint**. This value is the URL where the service is published.
- The **API key**. This value is a unique key that grants access.

You obtain both of these values from the Azure portal.

Because the service can take a few seconds to respond, it's best to use **asynchronous** calls to submit a form and then obtain results from the analysis:

```
poller = document_analysis_client.begin_analyze_document(         "prebuilt-layout", AnalyzeDocumentRequest(url_source=docUrl     ))
result: AnalyzeResult = poller.result()
```

The details you can extract from these results depend on the model you used.

Learn more

- What is Azure AI Document Intelligence?
- Azure AI Document Intelligence models

Use the General Document, Read, and Layout models

If you want to extract text, languages, and other information from documents with unpredictable structures, you can use the **read, general document, or layout models**.

In your polling company, customers and partners often send specifications, tenders, statements of work, and other documents with unpredictable structures. You want to know if Azure AI Document Intelligence can analyze and extract values from these documents.

Here, you'll learn about the **prebuilt models** that Microsoft provides **for general documents**.

Using the read model

The **Azure AI Document Intelligence read model** extracts printed and handwritten text from documents and images. It's used to provide *text extraction* in all the other prebuilt models.

The **read model** can also **detect the language** that a line of text is written in and **classify whether it's handwritten or printed text**.

> Note: The read model supports more languages for printed text than handwritten text. Check the documentation to see the current list of supported languages.

For multi-page PDF or TIFF files, you can use the `pages` parameter in your request to fix a page range for the analysis.

The read model is ideal if you want to extract words and lines from documents with no fixed or predictable structure.

Using the general document model

The general document model extends the functionality of the read model by adding the **detection of key-value pairs, entities, selection marks, and tables**. The model can extract these values from **structured, semi-structured, and unstructured documents**.

The general document model is the only prebuilt model to support entity extraction. It can recognize entities such as **people, organizations, and dates** and it runs against the whole document, not just key-value pairs. This approach ensures that, **when structural complexity has prevented the model extracting a key-value pair, an entity can be extracted instead**. Remember, however, that sometimes a single piece of text *might return both a key-value pair and an entity*.

The types of entities you can detect include:

- `Person` . The name of a person.
- `PersonType` . A job title or role.
- `Location` . Buildings, geographical features, geopolitical entities.
- `Organization` . Companies, government bodies, sports clubs, musical bands, and other groups.
- `Event` . Social gatherings, historical events, anniversaries.
- `Product` . Objects bought and sold.
- `Skill` . A capability belonging to a person.
- `Address` . Mailing address for a physical location.
- `Phone number` . Dialing codes and numbers for mobile phones and landlines.
- `Email` . Email addresses.
- `URL` . Webpage addresses.
- `IP Address` . Network addresses for computer hardware.
- `DateTime` . Calendar dates and times of day.
- `Quantity` . Numerical measurements with their units.

Using the layout model

As well as extracting text, the layout model returns **selection marks** and **tables** from the input image or PDF file. It's a good model to use when you need **rich information about the structure of a document**.

When you digitize a document, it can be at an odd angle. Tables can have complicated structures with or without headers, cells that span columns or rows, and incomplete columns or rows. The layout model can handle all of these difficulties to extract the complete document structure.

For example, each **table cell** is extracted with:

- Its content text.
- The size and position of its bounding box.
- If it's part of a header column.
- Indexes to indicate its row and column position in the table.

Selection marks are extracted with their **bounding box, a confidence indicator, and whether they're selected or not**.

Learn more

- Language support for Azure AI Document Intelligence
- Azure AI Document Intelligence read model
- Azure AI Document Intelligence general document model
- Azure AI Document Intelligence layout model

Use financial, ID, and tax models

Azure AI Document Intelligence includes some prebuilt models that are trained on common form types. You can use these models to obtain the values of common fields from **invoices, receipts, business cards**, and more.

In your polling company, invoices and receipts are often submitted as photos or scans of the paper documents. Sometimes the scan is poor and the paper is creased or damaged. You want to know if Azure AI Document Intelligence can get this information into your databases more efficiently than manual data entry.

Here, you'll learn about the prebuilt models that handle financial, identity, and tax documents.

Using the invoice model

Your business both issues invoices and receives them from partner organization. There might be many different formats on paper or in digitized forms and some will have been scanned poorly at odd angles or from creased paper.

The invoice model in Azure AI Document Intelligence can handle these challenges and uses the features of the read model to extract text from invoice scans. In addition, it extracts specific fields that are commonly used on invoices including:

- Customer name and reference ID
- Purchase order number
- Invoice and due dates
- Details about the vendor, such as name, tax ID, physical address.
- Similar details about the customer.
- Billing and shipping addresses.
- Amounts such as total tax, invoice total, and amount due.

Invoices also feature lines, usually in a table, each of which is one purchased item. **For each line**, the invoice model identifies details including:

- The description and product code of the product or service invoiced.
- Amounts such as the unit price, the quantity of items, the tax incurred, and the line total.

Using the receipt model

Receipts have similar fields and structures to invoices, but they record amounts paid instead of amounts charged. Azure AI Document Intelligence faces the same challenges of poor scanning or digitization but can reliably identify fields including:

- **Merchant details** such a name, phone number, and address.
- **Amounts** such as receipt total, tax, and tip.
- The **date and time of the transaction**.

As for invoices, receipts often include a table of items, each of which is a product or service purchased. **For each of these lines**, the model recognizes:

- The name of the item.
- The quantity of the item purchased.
- The unit price of the item.
- The total price for that quantity.

> Note: In Azure AI Document Intelligence v3.0 and later, the receipt model supports single-page hotel receipt processing. If a receipt is classified as a hotel receipt, the model extracts extra relevant fields such as **arrival and departure dates**.

Using the ID document model

The ID document model is trained to analyze two types of identity document:

- **United States drivers licenses**.
- **International passports**.

> Note: Only the biographical pages of passports can be analyzed. Visas and other travel documents are not supported.

The ID document model can extract fields including:

- First and last names.
- Personal information such as sex, date of birth, and nationality.
- The country and region where the document was issued.
- Unique numbers such as the document number and machine readable zone.
- Endorsements, restrictions, and vehicle classifications.

> **Important**: Since much of the data extracted by the ID document model is personal, it is of a sensitive nature and covered by data protection laws in most jurisdictions. Be sure that you have the permission of the individual to store their data and comply with all legal requirements in the way you handle this information.

Using the business card model

Business cards are a popular way to exchange contact information quickly and often include branding, unusual fonts, and graphic design elements. Fields that the business card model can extract include:

- First and last names.
- Postal addresses.
- Email and website addresses.
- Various telephone numbers.

Using other prebuilt models

Azure AI Document Intelligence offers several prebuilt models, with new models being released regularly. Before training a custom model, it's worth verifying if your use case can be analyzed accurately with one of these prebuilt models. Using a prebuilt model will benefit from rigorous testing, updated model versions, and reduced cost compared to a custom model.

Learn more

- Azure AI Document Intelligence model overview

Exercise - Analyze a document using Azure AI Document Intelligence

Use prebuilt Document Intelligence models

In this exercise, you'll set up an Azure AI Document Intelligence resource in your Azure subscription. You'll use both the Azure AI Document Intelligence Studio and C# or Python to submit forms to that resource for analysis.

Module assessment

1. You have a large set of documents with varying structures that contain customer name and address information. You want to extract entities for each customer. Which prebuilt model should you use? **General document model**.
2. You are using the prebuilt layout model to analyze a document with many checkboxes. You want to find out whether each box is checked or empty. What object should you use in the returned JSON code? **Selection marks**.
3. You submit a Word document to the Azure AI Document Intelligence general document model for analysis but you receive an error. The file is A4 size, contains 1 MB of data, and is not password-protected. How should you resolve the error? **Convert the document to PDF format**.

Summary

There are many document types that are common to most business and Azure AI Document Intelligence includes prebuilt models to handle them. If you have a collection of such forms that you want to analyze, you can extract data by using these prebuilt models and you don't have to train your own models. You can get up and running very quickly by submitting photos and scans to the most appropriate prebuilt model.

Now that you've completed this module, you can:

- Identify business problems that you can solve by using prebuilt models in Azure AI Document Intelligence.
- Analyze forms by using the General Document, Read, and Layout models.
- Analyze forms by using financial, ID, and tax prebuilt models.

Learn more

- What is Azure AI Document Intelligence?
- Azure AI Document Intelligence models
- Language support for Azure AI Document Intelligence
- Azure AI Document Intelligence read model
- Azure AI Document Intelligence general document model
- Azure AI Document Intelligence layout model
- Azure AI Document Intelligence invoice model
- Azure AI Document Intelligence receipt model
- Azure AI Document Intelligence ID document model
- Azure AI Document Intelligence business card model
- Azure AI Document Intelligence W-2 model

Analyze forms with prebuilt Azure AI Document Intelligence models

In this exercise, you'll set up an Azure AI Foundry project with all the necessary resources for document analysis. You'll use both the Azure AI Foundry portal and the Python SDK to submit forms to that resource for analysis.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

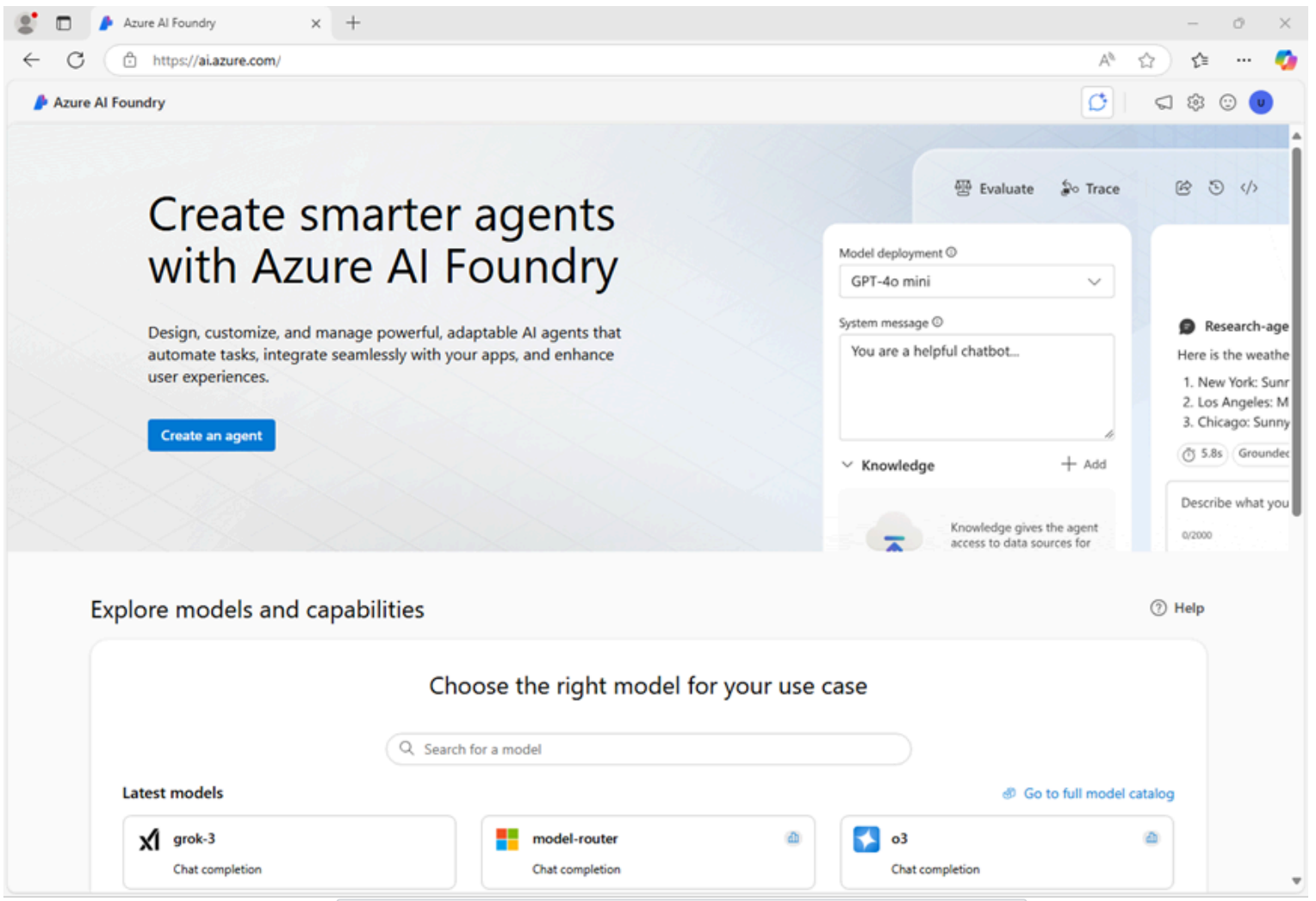
- Azure AI Document Intelligence client library for Python
- Azure AI Document Intelligence client library for Microsoft .NET
- Azure AI Document Intelligence client library for JavaScript

This exercise takes approximately **30** minutes.

Create an Azure AI Foundry project

Let's start by creating an Azure AI Foundry project.

1. In a web browser, open the Azure AI Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Azure AI Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):

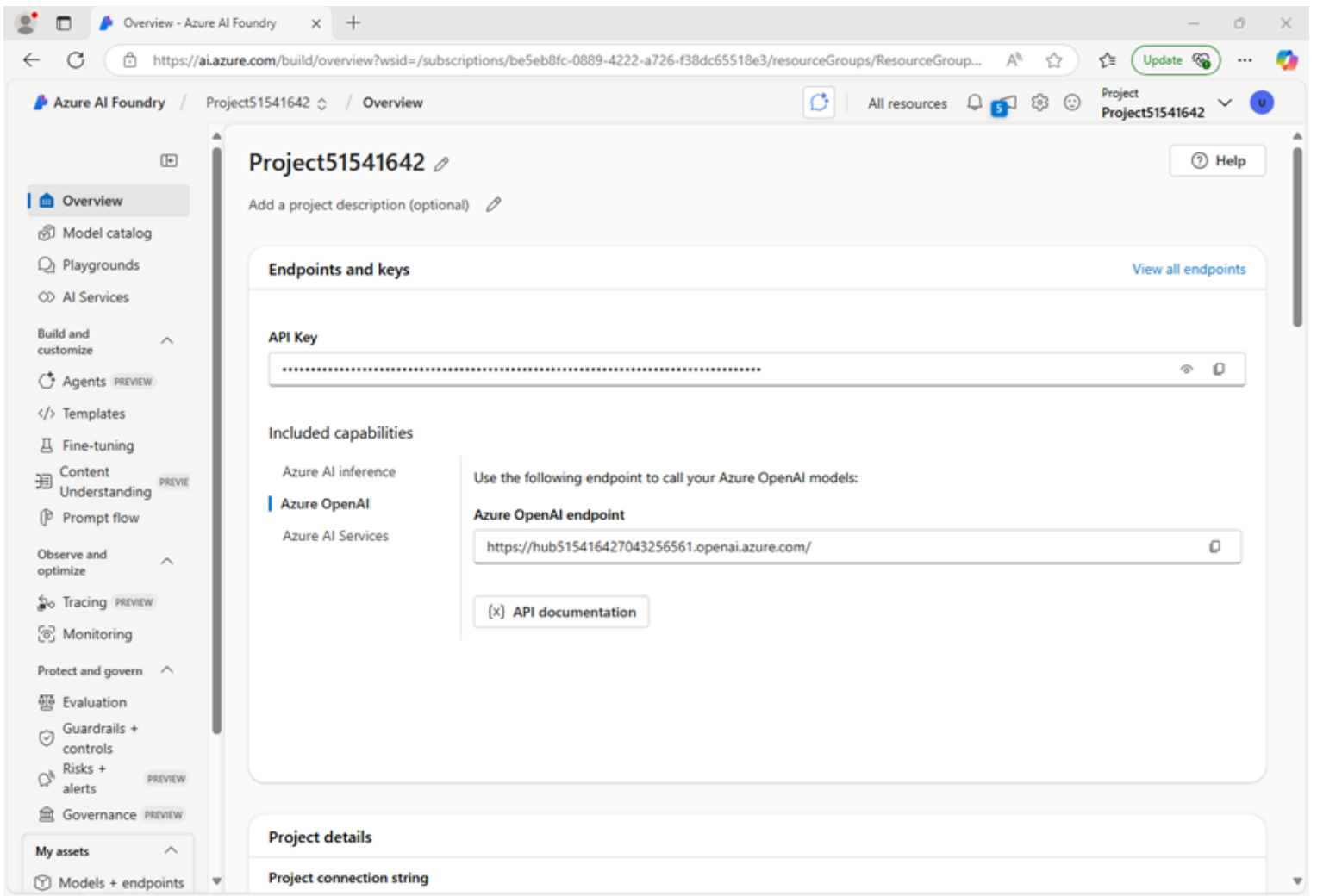


2. In the browser, navigate to `https://ai.azure.com/managementCenter/allResources` and select **Create new**. Then choose the option to create a new **AI hub resource**.
3. In the **Create a project** wizard, enter a valid name for your project, and select the option to create a new hub. Then use the **Rename hub** link to specify a valid name for your new hub, expand **Advanced options**, and specify the following settings for your project:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Any available region*

> **Note**: If you're working in an Azure subscription in which policies are used to restrict allowable resource names, you may need to use the link at the bottom of the **Create a new project** dialog box to create the hub using the Azure portal.

> **Tip**: If the **Create** button is still disabled, be sure to rename your hub to a unique alphanumeric value.

4. Wait for your project to be created.
5. When your project is created, close any tips that are displayed and review the project page in Azure AI Foundry portal, which should look similar to the following image:

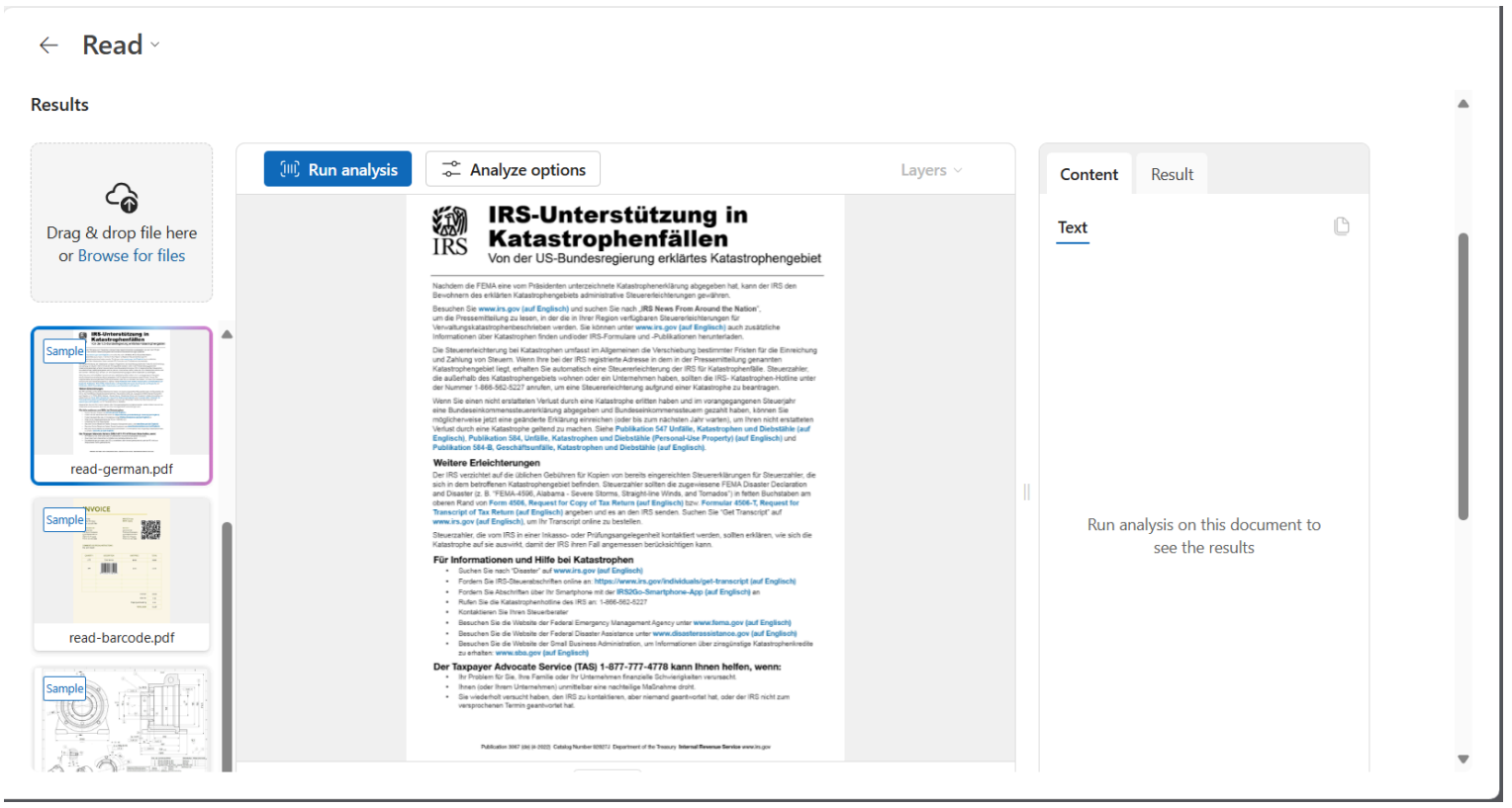


Use the Read model

Let's start by using the **Azure AI Foundry** portal and the Read model to analyze a document with multiple languages:

1. In the navigation panel on the left, select **AI Services**.
2. In the **Azure AI Services** page, select the **Vision + Document** tile.
3. In the **Vision + Document** page, verify that the **Document** tab is selected, then select the **OCR/Read** tile.

   In the **Read** page, the Azure AI Services resource created with your project should already be connected.
4. In the list of documents on the left, select **read-german.pdf**.



5. At the top toolbar, select **Analyze options**, then enable the **Language** check-box (under **Optional detection**) in the **Analyze options** pane and select **Save**.
6. At the top-left, select **Run Analysis**.
7. When the analysis is complete, the text extracted from the image is shown on the right in the **Content** tab. Review this text and compare it to the text in the original image for accuracy.
8. Select the **Result** tab. This tab displays the extracted JSON code.

Prepare to develop an app in Cloud Shell

Now let's explore the app that uses the Azure Document Intelligence service SDK. You'll develop your app using Cloud Shell. The code files for your app have been provided in a GitHub repo.

This is the invoice that your code will analyze.

CONTOSO LTD.

INVOICE

Contoso Headquarters
123 456th St
New York, NY, 10001

INVOICE: INV-100
DATE: 11/15/2019
DUE DATE: 12/15/2019
CUSTOMER NAME: MICROSOFT CORPORATION
CUSTOMER ID: CID-12345

Microsoft Corp
123 Other St,
Redmond WA, 98052

BILL TO:
Microsoft Finance
123 Bill St,
Redmond WA, 98052

SHIP TO:
Microsoft Delivery
123 Ship St,
Redmond WA, 98052

SERVICE ADDRESS:
Microsoft Services
123 Service St,
Redmond WA, 98052

SALESPERSON	P.O. NUMBER	REQUISITIONER	SHIPPED VIA	F.O.B. POINT	TERMS
	PO-3333				

QUANTITY	DESCRIPTION	UNIT PRICE	TOTAL
1	Test for 23 fields	1	$100.00
		SUBTOTAL	$100.00
		SALES TAX	$10.00
		TOTAL	$110.00

1. In the Azure AI Foundry portal, view the **Overview** page for your project.

2. In the **Endpoints and keys** area, select the **Azure AI Services** tab, and note the **API Key** and **Azure AI Services endpoint**. You'll use these credentials to connect to your Azure AI Services in a client application.
3. Open a new browser tab (keeping the Azure AI Foundry portal open in the existing tab). Then in the new tab, browse to the Azure portal at `https://portal.azure.com`; signing in with your Azure credentials if prompted.
4. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

   > **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

5. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

6. In the PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you paste commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

   Now follow the steps for your chosen programming language.

7. After the repo has been cloned, navigate to the folder containing the code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/prebuilt-doc-intelligence/Python
   ```

8. In the cloud shell command line pane, enter the following command to install the libraries you'll use:

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-ai-formrecognizer==3.3.3
   ```

9. Enter the following command to edit the configuration file that has been provided:

   Code Copy

   ```
   code .env
   ```

   The file is opened in a code editor.
10. In the code file, replace the **YOUR_ENDPOINT** and **YOUR_KEY** placeholders with your Azure AI services endpoint and its API key (copied from the Azure AI Foundry portal).
11. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

Add code to use the Azure Document Intelligence service

Now you're ready to use the SDK to evaluate the pdf file.

1. Enter the following command to edit the app file that has been provided:

```
code document-analysis.py
```

   The file is opened in a code editor.

2. In the code file, find the comment **Import the required libraries** and add the following code:

```
# Add references
from azure.core.credentials import AzureKeyCredential
from azure.ai.formrecognizer import DocumentAnalysisClient
```

3. Find the comment **Create the client** and add the following code (being careful to maintain the correct indentation level):

```
# Create the client
document_analysis_client = DocumentAnalysisClient(
     endpoint=endpoint, credential=AzureKeyCredential(key)
)
```

4. Find the comment **Analyze the invoice** and add the following code:

```
# Analyse the invoice
poller = document_analysis_client.begin_analyze_document_from_url(
     fileModelId, fileUri, locale=fileLocale
)
```

5. Find the comment **Display invoice information to the user**and add the following code:

```
# Display invoice information to the user
receipts = poller.result()

for idx, receipt in enumerate(receipts.documents):

     vendor_name = receipt.fields.get("VendorName")
     if vendor_name:
         print(f"\nVendor Name: {vendor_name.value}, with confidence
{vendor_name.confidence}.")

     customer_name = receipt.fields.get("CustomerName")
     if customer_name:
         print(f"Customer Name: '{customer_name.value}, with confidence
{customer_name.confidence}.")


     invoice_total = receipt.fields.get("InvoiceTotal")
     if invoice_total:
         print(f"Invoice Total: '{invoice_total.value.symbol}{invoice_total.value.amount},
 with confidence {invoice_total.confidence}.")
```

6. In the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes. Keep the code editor open in case you need to fix any errors in the code, but resize the panes so you can see the command line pane clearly.

7. In the command line pane, enter the following command to run the application.

Code Copy

```
python document-analysis.py
```

The program displays the vendor name, customer name, and invoice total with confidence levels. Compare the values it reports with the sample invoice you opened at the start of this section.

Clean up

If you’re done with your Azure resource, remember to delete the resource in the Azure portal (`https://portal.azure.com`) to avoid further charges.

5.4 Extract data from forms with Azure Document intelligence

Document intelligence **uses machine learning technology** to **identify and extract key-value pairs and table data from form documents** with accuracy, at scale. This module teaches you how to use the **Azure Document intelligence cognitive service**.

Learning objectives

In this module, you learn how to:

- Identify how **Document intelligence's layout service, prebuilt models, and custom models** can automate processes.
- Use Document intelligence's capabilities with **SDKs, REST API, and Document Intelligence Studio**.
- Develop and test custom models.

Introduction

Forms are used to communicate information in every industry, every day. Many people still manually **extract data from forms** to exchange information.

Consider some of the instances when a person needs to process form data:

- When filing claims
- When enrolling new patients in an online management system
- When entering data from receipts to an expense report
- When reviewing an operations report for anomalies
- When selecting data from a report to give to a stakeholder

Without AI services, people need to manually sort through form documents to identify important information and then manually reenter data to record it. Some may also need to complete these tasks in real-time with a customer.

Azure Document Intelligence services provide the building blocks for automation by using intelligent services to extract data at scale and with accuracy. **Azure Document Intelligence** is **a Vision API that extracts key-value pairs and table data from form documents**.

Uses of the Azure Document Intelligence service include:

- Process automation
- Knowledge mining

- Industry-specific applications

In this module, you'll learn how to:

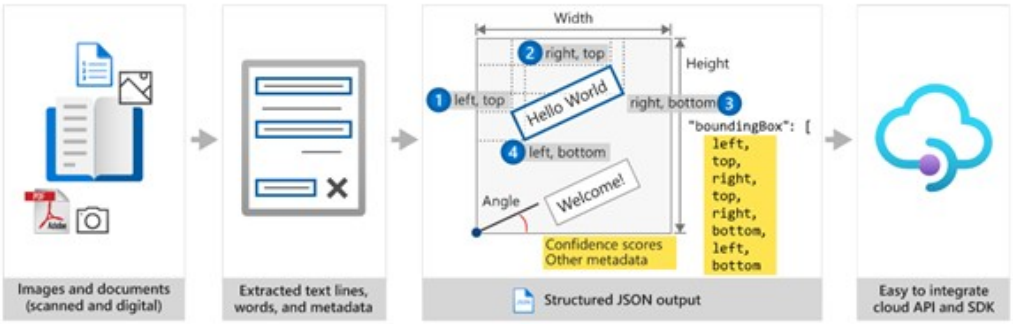
- Identify how Azure Document Intelligence's document analysis, prebuilt, and custom models can automate processes
- Use **Azure Document Intelligence**'s **Optical Character Recognition (OCR)** capabilities with **SDKs and REST API**.
- Develop and test a custom Azure Document Intelligence model

To complete this module, you'll need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.microsoft.com.

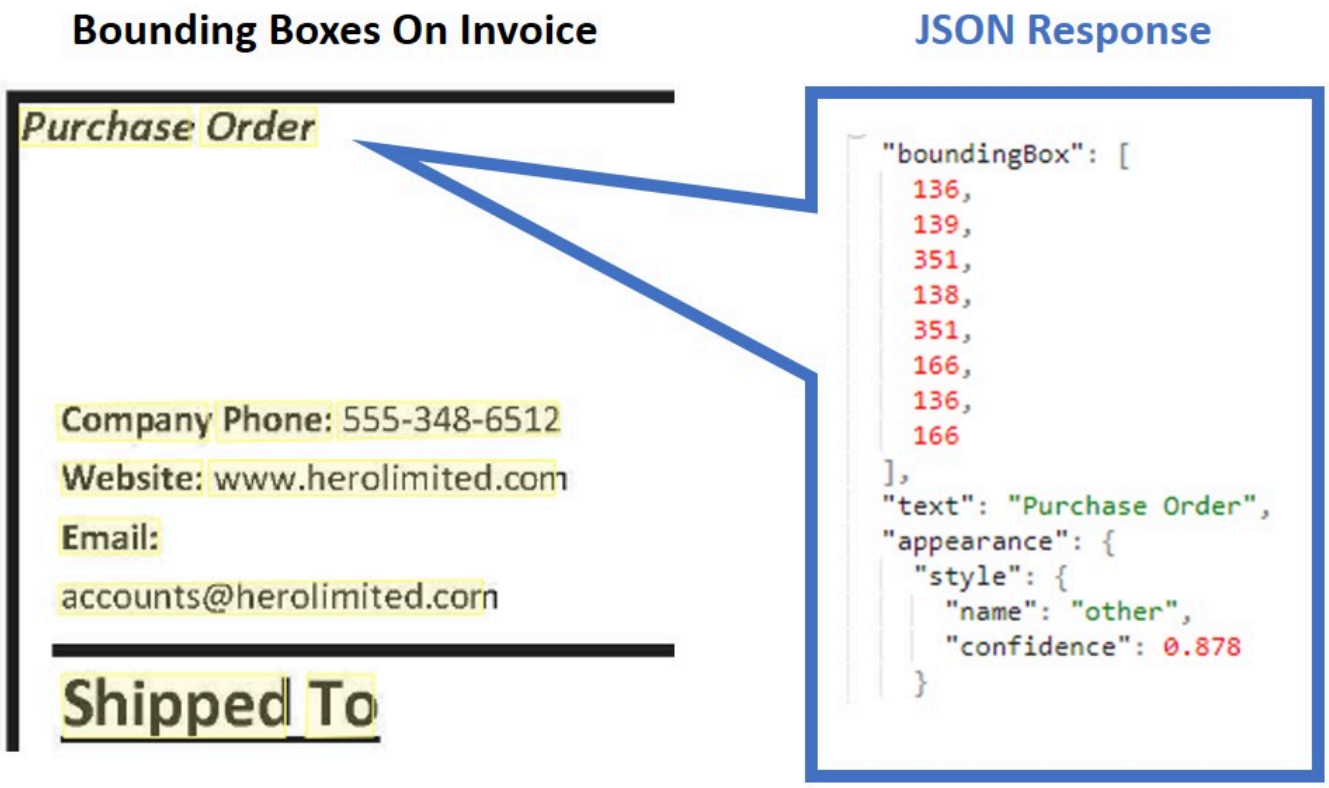
What is Azure Document Intelligence?

Azure Document Intelligence is one of many Azure AI Services, cloud-based artificial intelligence (AI) services with *REST APIs and client library SDKs* that can be used to build intelligence into your applications.

Azure Document Intelligence uses **Optical Character Recognition (OCR) capabilities and deep learning models** to **extract text, key-value pairs, selection marks, and tables from documents**.



OCR captures document structure by **creating bounding boxes around detected objects in an image**. The locations of the bounding boxes are recorded as coordinates in relation to the rest of the page. Azure Document Intelligence services return bounding box data and other information in a structured form with the relationships from the original file.



To build a high-accuracy model from scratch, people need to build deep learning models, use a large amount of compute resources, and face long model training times. These factors could make a project infeasible. Azure Document Intelligence provides underlying models that have been trained on thousands of form examples. The underlying models enable you to do high-accuracy data extraction from your forms with little to no model training.

Azure Document Intelligence service components

Azure Document Intelligence is composed of the following services:

- **Document analysis models**: which *take an input of* **JPEG, PNG, PDF, and TIFF** files and **return a JSON file** with **the location of text in bounding boxes, text content, tables, selection marks (also known as checkboxes or radio buttons), and document structure**.
- **Prebuilt models**: which detect and extract information from document images and return the extracted data in a structured JSON output. Azure Document Intelligence currently supports prebuilt models for several forms, including:
  - W-2 forms
  - Invoices
  - Receipts
  - ID documents
  - Business cards
- **Custom models**: custom models extract data from forms specific to your business. Custom models can be trained through the Azure Document Intelligence Studio.

> Note: Some Azure Document Intelligence features are in preview, as of the time this content was authored, and as a result, features and usage details might change. You should refer to the official page for up-to-date information.

Access services

You can access Azure Document Intelligence services in several ways. These options include using:

- A **REST API**
- Client library **SDKs**
- **Azure Document Intelligence Studio**
- **Azure AI Foundry**

> Tip: This module's exercise focuses on the Python and .NET SDKs. The underlying REST services can be used by any language.

Check out the documentation for quick start guides on all the available SDKs and the REST API.

Get started with Azure Document Intelligence

To start a project with Azure Document Intelligence services, you need an Azure resource and selection of form files for data extraction.

Subscribe to a resource

You can access Azure Document Intelligence services via:

- An **Azure AI Service resource**: a *multi-service subscription key* (used across multiple Azure AI Services)

OR

- An **Azure Document Intelligence resource**: a *single-service subscription key* (used only with a specific Azure AI Service)

> Note: Create an **Azure AI Services resource** if you plan to access multiple Azure AI services **under a single endpoint/key**. For Azure Document Intelligence access only, create an Azure Document Intelligence resource. Note that **you need a single-service resource if you intend to use Microsoft Entra authentication**.

You can subscribe to a service in the Azure portal or with the Azure Command Line Interface (CLI). You can learn more about the CLI commands here.

Understand Azure Document Intelligence file input requirements

Azure Document Intelligence works on input documents that meet these requirements:

- Format must be **JPG, PNG, BMP, PDF (text or scanned), or TIFF**.
- The file size must be **less than 500 MB for paid (S0) tier and 4 MB for free (F0) tier**.
- Image dimensions must be **between 50 x 50 pixels and 10,000 x 10,000 pixels**.
- The total size of the training data set must be **500 pages or less**.

More input requirements can be found in the documentation for specific models.

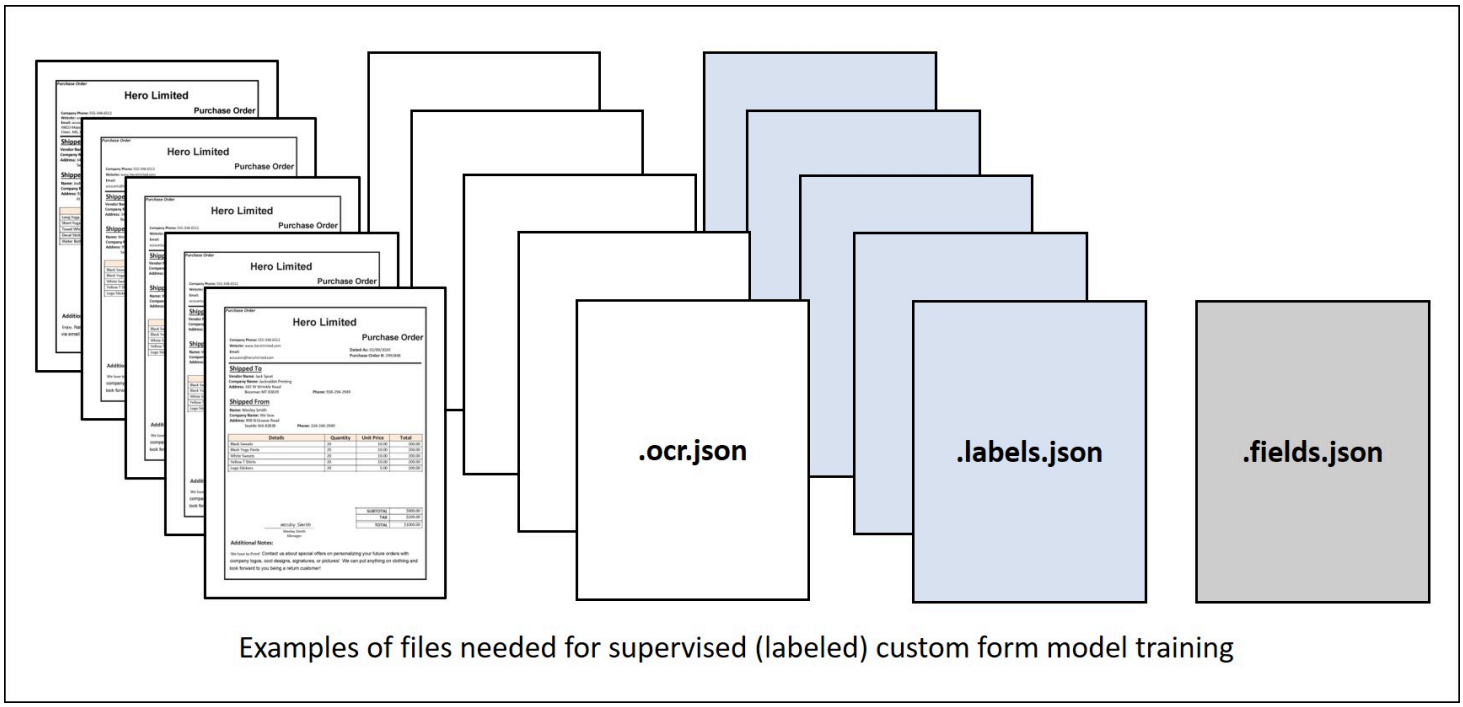
Decide what component of Azure Document Intelligence to use

After you collect your files, decide what you need to accomplish.

Use case	Recommended features to use
Use **OCR** capabilities to capture document analysis	Use the **Layout model, Read model, or General Document model**.
Create an application that extracts data from **W-2s, Invoices, Receipts, ID documents, Health insurance, vaccination, and business cards**	Use a **prebuilt model**. These models don't need to be trained. Azure Document Intelligence services analyze the documents and **return a JSON output**.
Create an application to extract data from your **industry-specific forms**	Create a **custom model**. This model needs to be trained on sample documents. After you train the custom model, it can analyze new documents and **return a JSON output**.

Train custom models

Azure's **Azure Document Intelligence service supports supervised machine learning**. You can train custom models and create composite models with form documents and JSON documents that contain labeled fields.



Examples of files needed for supervised (labeled) custom form model training

To train a custom model:

- Store sample forms in an **Azure blob container**, along with **JSON files containing layout and label field information**.
  - You can generate an **ocr.json** file for each sample form using the Azure Document Intelligence's Analyze document function. Additionally, you need a single **fields.json** file describing **the fields you want to extract**, and a **labels.json** file for **each sample form mapping the fields to their location in that form**.
- Generate a **shared access security (SAS) URL** for the container.
- Use the **Build model** *REST API function (or equivalent SDK method)*.
- Use the **Get model** *REST API function (or equivalent SDK method)* to get **the trained model ID**.

OR

- Use the Azure Document Intelligence Studio to label and train. There are two types of underlying models for custom forms custom template models or custom neural models.
  - **Custom template models** accurately extract **labeled key-value pairs, selection marks, tables, regions, and signatures from documents**. Training only takes a few minutes, and more than 100 languages are supported.
  - **Custom neural models** are deep learned models that combine layout and language features to accurately **extract labeled fields from documents**.This model is **best for semi-structured or unstructured documents**.

Use Azure Document Intelligence models

Using the API

To extract form data using a custom model, use the analyze document function of either a **supported SDK, or the REST API**, while **supplying model ID** (generated during model training). This function starts the form analysis. which you can then request the result to get the analysis.

Example code to call your model:

```
endpoint = "YOUR_DOC_INTELLIGENCE_ENDPOINT"
key = "YOUR_DOC_INTELLIGENCE_KEY"

model_id = "YOUR_CUSTOM_BUILT_MODEL_ID"
formUrl = "YOUR_DOCUMENT"

document_analysis_client = DocumentAnalysisClient(     endpoint=endpoint, credential=AzureKeyCredential(key) )

# Make sure your document's type is included in the list of document types the custom model can analyze
task = document_analysis_client.begin_analyze_document_from_url(model_id, formUrl)
result = task.result()
```

A successful JSON response contains **analyzeResult** that contains **the content extracted** and **an array of pages containing information about the document content**.

Example analyze document JSON response:

Explore the documentation for supported language quickstarts.

Understanding confidence scores

If the confidence values of the **analyzeResult** are low, try to improve the quality of your input documents.

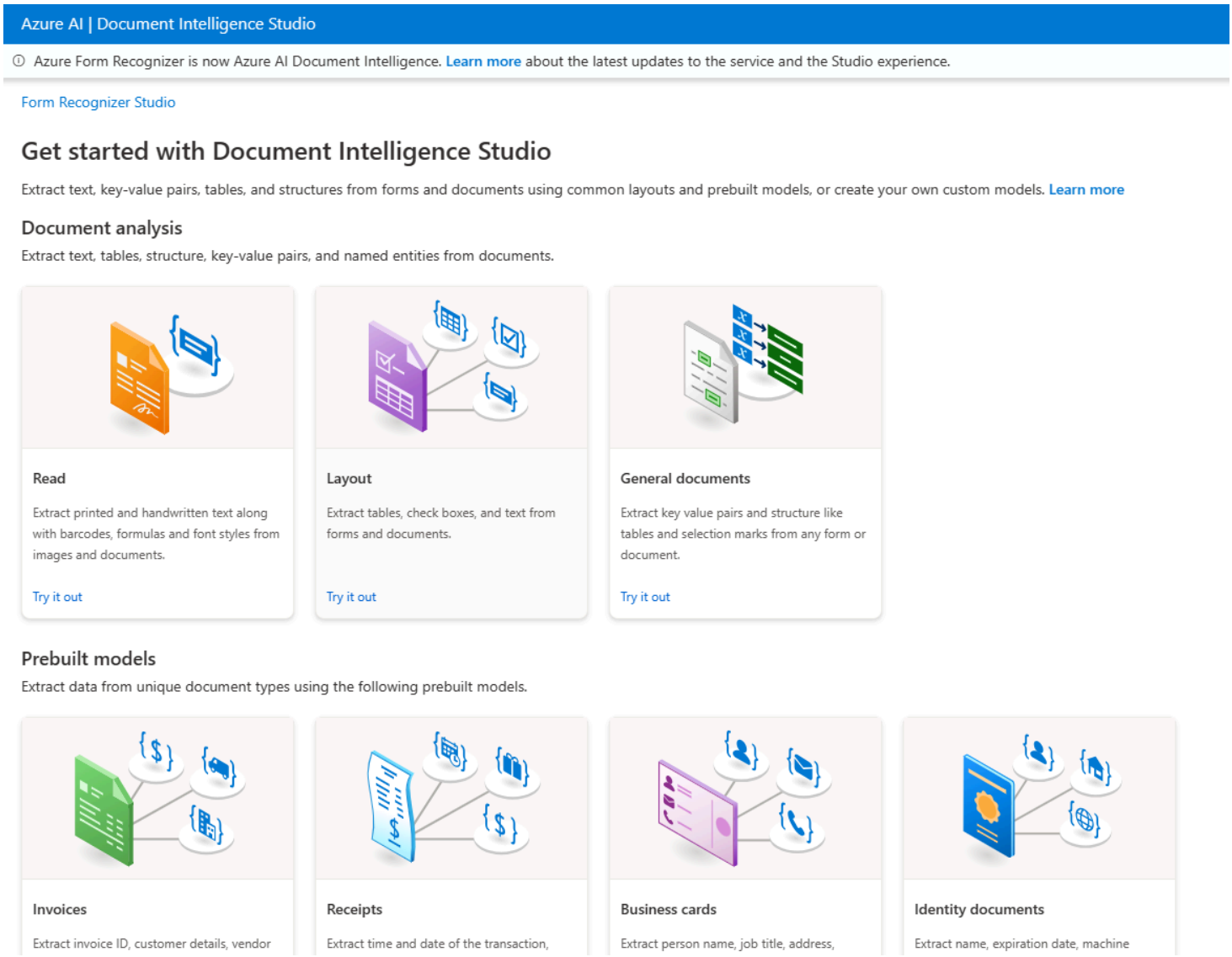
You want to make sure that the form you're analyzing has a similar appearance to forms in the training set if the confidence values are low. If the form appearance varies, **consider training more than one model, with each model focused on one form format**.

Depending on the use case, you might find that **a confidence score of 80% or higher is acceptable for a low-risk application**. **For more sensitive cases, like reading medical records or billing statements, a score of 100% is recommended**.

Use the Azure Document Intelligence Studio

In addition to SDKs and the REST API, **Azure Document Intelligence** services can be accessed through a user interface called the **Azure Document Intelligence Studio**, an online tool for visually exploring, understanding, and

integrating features from the Azure Document Intelligence service. The Studio can be used to **analyze form layouts, extract data from prebuilt models, and train custom models**.



The Azure Document Intelligence Studio currently supports the following projects:

- **Document analysis models**
  - **Read**: Extract **printed and handwritten text lines, words, locations, and detected languages from documents and images**.
  - **Layout**: Extract **text, tables, selection marks, and structure information from documents (PDF and TIFF) and images (JPG, PNG, and BMP)**.
  - **General Documents**: Extract **key-value pairs, selection marks, and entities from documents**.
- **Prebuilt models**
- **Custom models**

Build Document analysis model projects

To **extract text, tables, structure, key-value pairs, and named entities** with **document analysis models**:

- Create an Azure Document Intelligence or Azure AI Services resource.

- Select either "**Read**", "**Layout**", or "**General Documents**" under the Document analysis models category.
- Analyze your document. You'll need your Azure Document Intelligence or Azure AI service **endpoint and key**.

Build prebuilt model projects

To extract data from common forms with prebuilt models:

- Create an **Azure Document Intelligence** or **Azure AI Services** resource
- Select one of the "**prebuilt models**" including **W-2s, Invoices, Receipts, ID documents, Health insurance, vaccination, and business cards**.
- Analyze your document. You'll need your Azure Document Intelligence or Azure AI service **endpoint and key**.

Build custom model projects

You can use **Azure Document Intelligence Studio's custom service** for the entire process of training and testing custom models.

When you use Azure Document Intelligence Studio to build custom models, the **ocr.json** files, **labels.json** files, and **fields.json** file needed for training are automatically created and stored in your storage account.

To train a custom model and use it to extract data with custom models:

- Create an Azure Document Intelligence or Azure AI Services resource
- Collect **at least 5-6 sample forms** for training and upload them to your storage account container.
- Configure **cross-domain resource sharing (CORS)**. CORS enables Azure Document Intelligence Studio to store labeled files in your storage container.
- **Create a custom model project in Azure Document Intelligence Studio**. You'll need to provide configurations linking your storage container and Azure Document Intelligence or Azure AI Service resource to the project.
- Use Azure Document Intelligence Studio to **apply labels to text**.
- Train your model. Once the model is trained, you'll receive a **Model ID and Average Accuracy for tags**.
- Test your model by analyzing a new form that wasn't used in training.

Exercise - Extract data from custom forms

In this exercise, you'll use the Azure Document Intelligence service to train and test a custom model using the Python or .NET SDK.

To complete the exercise for this module, launch the VM and follow the instructions.

Extract Data from Forms

Suppose a company currently requires employees to manually purchase order sheets and enter the data into a database. They would like you to utilize AI services to improve the data entry process. You decide to build a machine

learning model that will read the form and produce structured data that can be used to automatically update a database.

Azure AI Document Intelligence is an Azure AI service that enables users to build automated data processing software. This software can **extract text, key/value pairs, and tables from form documents using optical character recognition (OCR)**. Azure AI Document Intelligence has **pre-built models for recognizing invoices, receipts, and business cards**. The service also provides the capability to train **custom models**. In this exercise, we will focus on building custom models.

Module assessment

In this module, you have learned how to use the Azure Document Intelligence service to extract data from forms.

Consider the following review questions to check your understanding of the topics discussed in this module.

1. A person plans to use an Azure Document Intelligence prebuilt invoice model. To extract document data using the model and REST API language, what are two calls they need to make to the API? **Analyze Invoice and Get Analyze Invoice Result**.
2. A person needs to build an application that submits expense claims and extracts the merchant, date, and total from scanned receipts. What's the best way to build the application? **Use Azure Document Intelligence's prebuilt receipts model**.
3. A person is building a custom model with Azure Document Intelligence services. What is required to train a model? **Along with the form to analyze, JSON files need to be provided**.

Summary

This module focused on Azure Document Intelligence's **prebuilt service, custom service, and its client library SDKs and REST API**. We also introduced the **Azure Document Intelligence Studio** to label and train your model.

Azure Document Intelligence services can be integrated with other Azure AI Services. For example, you can try using this tutorial with **Azure Document Intelligence and Cognitive Search**.

Document intelligence is just one part of the overall Vision API in Azure AI Services. Azure Document Intelligence services are ever evolving. You can read about the latest updates here.

Analyze forms with custom Azure AI Document Intelligence models

Suppose a company currently requires employees to manually purchase order sheets and enter the data into a database. They would like you to utilize AI services to improve the data entry process. You decide to build a machine learning model that will read the form and produce structured data that can be used to automatically update a database.

Azure AI Document Intelligence is an Azure AI service that enables users to build automated data processing software. This software can extract text, key/value pairs, and tables from form documents using optical character recognition (OCR). Azure AI Document Intelligence has pre-built models for recognizing invoices, receipts, and business cards. The service also provides the capability to train custom models. In this exercise, we will focus on building custom models.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

- Azure AI Document Intelligence client library for Python
- Azure AI Document Intelligence client library for Microsoft .NET
- Azure AI Document Intelligence client library for JavaScript

This exercise takes approximately **30** minutes.

Create a Azure AI Document Intelligence resource

To use the Azure AI Document Intelligence service, you need a Azure AI Document Intelligence or Azure AI Services resource in your Azure subscription. You'll use the Azure portal to create a resource.

1. In a browser tab, open the Azure portal at `https://portal.azure.com`, signing in with the Microsoft account associated with your Azure subscription.
2. On the Azure portal home page, navigate to the top search box and type **Document Intelligence** and then press **Enter**.
3. On the **Document Intelligence** page, select **Create**.
4. On the **Create Document Intelligence** page, create a new resource with the following settings:
    - **Subscription**: Your Azure subscription.
    - **Resource group**: Create or select a resource group
    - **Region**: Any available region
    - **Name**: A valid name for your Document Intelligence resource
    - **Pricing tier**: Free F0 (*if you don't have a Free tier available, select* Standard S0).
5. When the deployment is complete, select **Go to resource** to view the resource's **Overview** page.

Prepare to develop an app in Cloud Shell

You'll develop your text translation app using Cloud Shell. The code files for your app have been provided in a GitHub repo.

1. In the Azure Portal, use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

    > **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

2. Size the cloud shell pane so you can see both the command line console and the Azure portal. You'll need to use the the split bar to switch as you switch between the two panes.

3. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

4. In the PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you paste commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

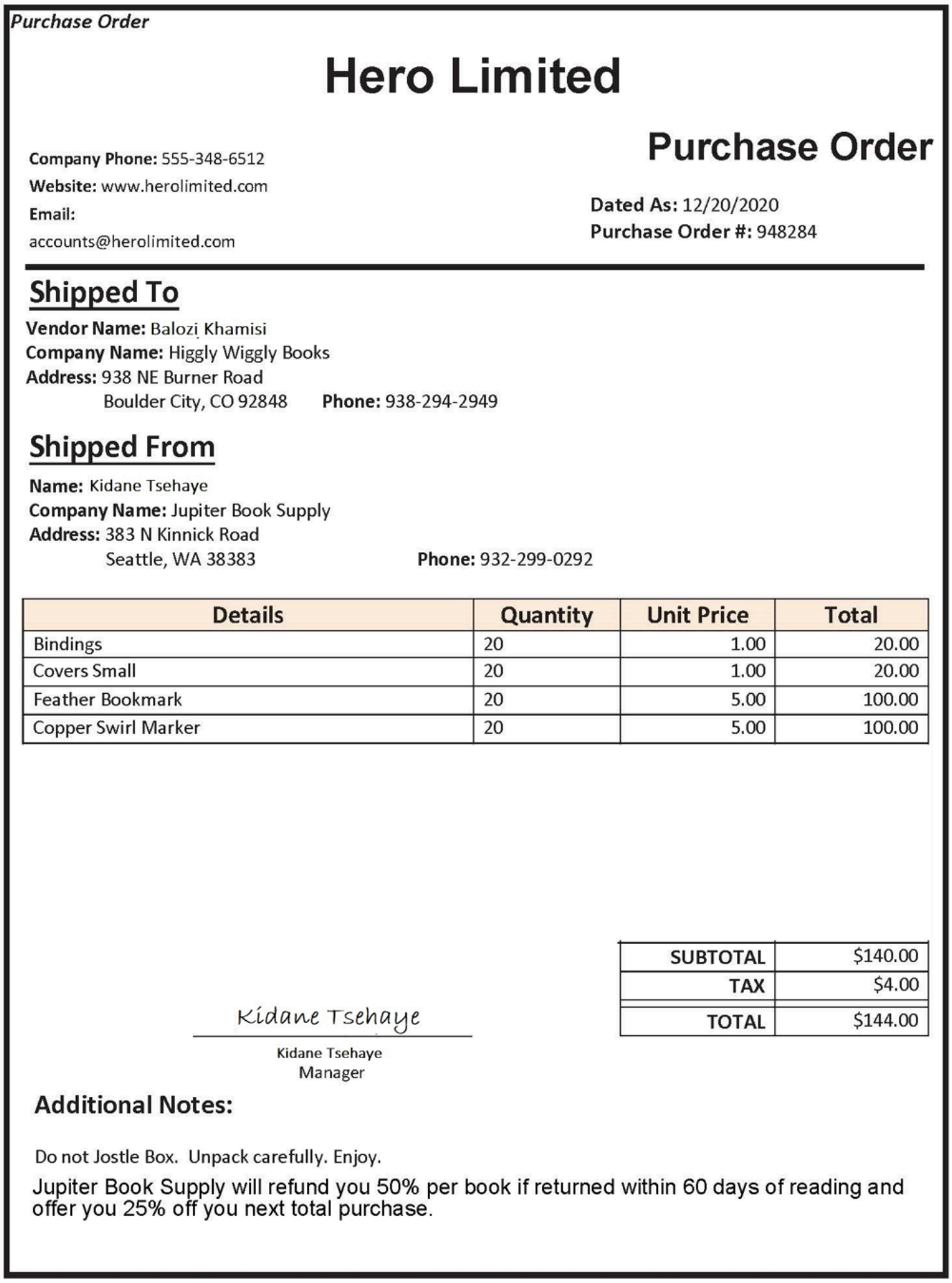
5. After the repo has been cloned, navigate to the folder containing the application code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/custom-doc-intelligence
   ```

Gather documents for training

You'll use the sample forms such as this one to train a test a model:



Purchase Order

Hero Limited

Company Phone: 555-348-6512
Website: www.herolimited.com
Email:
accounts@herolimited.com

Purchase Order

Dated As: 12/20/2020
Purchase Order #: 948284

Shipped To

Vendor Name: Balozi Khamisi
Company Name: Higgly Wiggly Books
Address: 938 NE Burner Road
Boulder City, CO 92848 **Phone:** 938-294-2949

Shipped From

Name: Kidane Tsehaye
Company Name: Jupiter Book Supply
Address: 383 N Kinnick Road
Seattle, WA 38383 **Phone:** 932-299-0292

Details	Quantity	Unit Price	Total
Bindings	20	1.00	20.00
Covers Small	20	1.00	20.00
Feather Bookmark	20	5.00	100.00
Copper Swirl Marker	20	5.00	100.00

SUBTOTAL	$140.00
TAX	$4.00
TOTAL	$144.00

Kidane Tsehaye

Kidane Tsehaye
Manager

Additional Notes:

Do not Jostle Box. Unpack carefully. Enjoy.

Jupiter Book Supply will refund you 50% per book if returned within 60 days of reading and offer you 25% off you next total purchase.

1. In the command line, run `ls ./sample-forms` to list the content in the **sample-forms** folder. Notice there are files ending in **.json** and **.jpg** in the folder.

   You will use the **.jpg** files to train your model.

   The **.json** files have been generated for you and contain label information. The files will be uploaded into your blob storage container alongside the forms.
2. In the **Azure portal** and navigate to your resource's **Overview** page if you're not already there. Under the *Essentials* section, note the **Resource group**, **Subscription ID**, and **Location**. You will need these values in subsequent steps.

3. Run the command `code setup.sh` to open **setup.sh** in a code editor. You will use this script to run the Azure command line interface (CLI) commands required to create the other Azure resources you need.
4. In the **setup.sh** script, review the commands. The program will:

   - Create a storage account in your Azure resource group
   - Upload files from your local *sampleforms* folder to a container called *sampleforms* in the storage account
   - Print a Shared Access Signature URI

5. Modify the **subscription_id**, **resource_group**, and **location** variable declarations with the appropriate values for the subscription, resource group, and location name where you deployed the Document Intelligence resource.

   > **Important**: For your **location** string, be sure to use the code version of your location. For example, if your location is "East US", the string in your script should be `eastus`. You can see that version is the **JSON View** button on the right side of the **Essentials** tab of your resource group in Azure portal.

   If the **expiry_date** variable is in the past, update it to a future date. This variable is used when generating the Shared Access Signature (SAS) URI. In practice, you will want to set an appropriate expiry date for your SAS. You can learn more about SAS here.

6. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes and then use the **CTRL+Q** command or **Right-click > Quit** to close the code editor while keeping the cloud shell command line open.

7. Enter the following commands to make the script executable and to run it:

   Code Copy

   ```
   chmod +x ./setup.sh
   ./setup.sh
   ```

8. When the script completes, review the displayed output.
9. In the Azure portal, refresh your resource group and verify that it contains the Azure Storage account just created. Open the storage account and in the pane on the left, select **Storage browser**. Then in Storage Browser, expand **Blob containers** and select the **sampleforms** container to verify that the files have been uploaded from your local **custom-doc-intelligence/sample-forms** folder.

Train the model using Document Intelligence Studio

Now you will train the model using the files uploaded to the storage account.

1. Open a new browser tab, and navigate to the Document Intelligence Studio at `https://documentintelligence.ai.azure.com/studio`.
2. Scroll down to the **Custom models** section and select the **Custom extraction model** tile.
3. If prompted, sign in with your Azure credentials.
4. If you are asked which Azure AI Document Intelligence resource to use, select the subscription and resource name you used when you created the Azure AI Document Intelligence resource.
5. Under **My Projects**, Create a new project with the following configuration:
   - **Enter project details**:
     - **Project name**: A valid name for your project
   - **Configure service resource**:
     - **Subscription**: Your Azure subscription
     - **Resource group**: The resource group where you deployed your Document Intelligence resource
     - **Document intelligence resource** Your Document Intelligence resource (select the *Set as default* option and use the default API version)
   - **Connect training data source**:
     - **Subscription**: Your Azure subscription
     - **Resource group**: The resource group where you deployed your Document Intelligence resource
     - **Storage account**: The storage account that was created by the setup script (select the *Set as default* option, select the `sampleforms` blob container, and leave the folder path blank)
6. When your project is created, on the top right of the page, select **Train** to train your model. Use the following configurations:
   - **Model ID**: A valid name for your model (*you'll need the model ID name in the next step*)

- **Build Mode**: Template.

7. Select **Go to Models**.
8. Training can take some time. Wait until the status is **succeeded**.

Test your custom Document Intelligence model

1. Return to the browser tab containing the Azure Portal and cloud shell. In the command line, run the following command to change to the folder containing the application code files:

```
cd Python
```

2. Install the Document Intelligence package by running the following command:

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-ai-formrecognizer==3.3.3
```

3. Enter the following command to edit the configuration file that has been provided:

```
code .env
```

4. In the pane containing the Azure portal, on the **Overview** page for your Document Intelligence resource, select **Click here to manage keys** to see the endpoint and keys for your resource. Then edit the configuration file with the following values:
   - Your Document Intelligence endpoint
   - Your Document Intelligence key
   - The Model ID you specified when training your model
5. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.
6. Open the code file for your client application (`code Program.cs` for C#, `code test-model.py` for Python) and review the code it contains, particularly that the image in the URL refers to the file in this GitHub repo on the web. Close the file without making any changes.
7. In the command line, and enter the following command to run the program:

```
python test-model.py
```

8. View the output and observe how the output for the model provides field names like `Merchant` and `CompanyPhoneNumber` .

Clean up

If you're done with your Azure resource, remember to delete the resource in the Azure portal to avoid further charges.

More information

For more information about the Document Intelligence service, see the Document Intelligence documentation.

5.5 Create a knowledge mining solution with Azure AI Search

Unlock the hidden insights in your data with **Azure AI Search**. In this module, you'll learn how to implement a **knowledge mining solution** that extracts and enriches data, making it searchable and ready for deeper analysis.

Learning objectives

After completing this module, you'll be able to:

- Implement **indexing** with **Azure AI Search**
- Use AI skills to **enrich data in an index**
- Search an index to find relevant information
- Persist extracted information in a knowledge store

Introduction

Azure AI Search is a powerful cloud-based service that enables you to **extract, enrich, and explore information** from a wide variety of data sources. In this module, you'll learn how to build **intelligent search and knowledge mining solutions** using Azure AI Search.

We'll start by introducing the core concepts of Azure AI Search, including how to **connect to data sources and create indexes**. You'll discover how the indexing process works, and how AI skills can be used to enrich your data with insights such as **language detection, key phrase extraction, and image analysis**.

After learning how to implement an index, you'll explore how to **query and filter results** using full-text search.

Finally, you'll see how to **use the knowledge store to persist enriched data** for further analysis and integration with other systems.

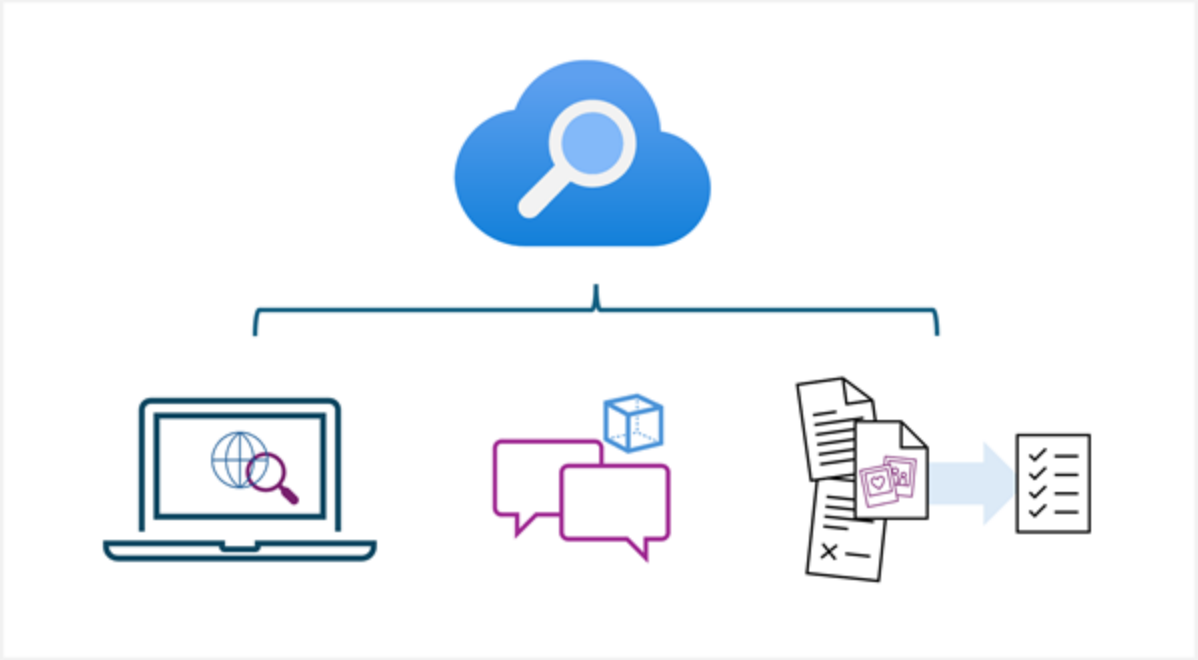
What is Azure AI Search?

Azure AI Search provides a cloud-based solution for **indexing and querying a wide range of data sources**, and **creating comprehensive and high-scale search solutions**. It provides the infrastructure and tools to create search solutions that **extract data from structured, semi-structured, and non-structured documents and other data sources**.

With Azure AI Search, you can:

- **Index documents and data** from a range of sources.
- Use AI skills to **enrich index data**.
- **Store extracted insights in a knowledge store** for analysis and integration.

Azure AI Search indexes contain insights extracted from your data; which can include **text inferred or read using OCR from images, entities and key phrases detection through text analytics**, and other derived information based on AI skills that are integrated into the indexing process.



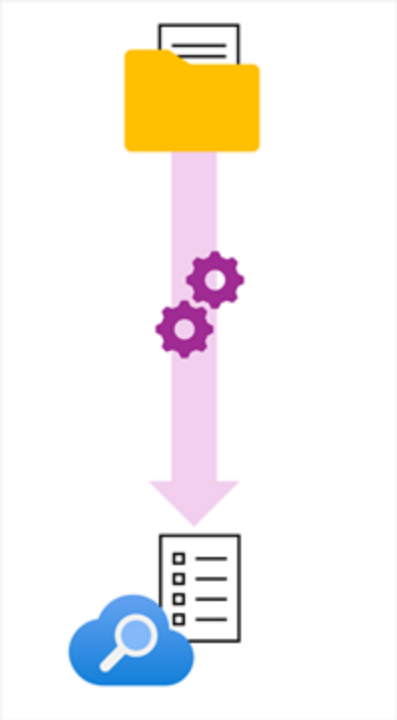
Azure AI search has many applications, including:

- Implementing an enterprise search solution to help employees or customers find information in websites or applications.
- **Supporting retrieval augmented generation (RAG) in generative AI applications** by using vector-based indexes for prompt grounding data.
- Creating **knowledge mining solutions** in which the indexing process is used to infer insights and extract granular data assets from documents to support data analytics.

In this module, we'll focus on Azure AI Search in knowledge mining scenarios.

Extract data with an indexer

At the heart of Azure AI Search solutions is the creation of an index. **An index contains your searchable content and is created and updated, unsurprisingly, by an indexer**.



The indexing process starts with a data source: the storage location of your original data artifacts; for example, an Azure blob store container full of documents, a database, or some other store.

The Indexer automates the extraction and indexing of data fields through an enrichment pipeline, in which it applies *document cracking to extract the contents of the source documents and applies incremental steps to create a hierarchical (JSON-based) document with the required fields for the index definition*.

The result is a populated index, which can be queried to return specified fields from documents that match the query criteria.

How documents are constructed during indexing

The indexing process works by creating a document for each indexed entity. *During indexing, an enrichment pipeline iteratively builds the documents that combine metadata from the data source with enriched fields extracted or generated by skills*. You can think of **each indexed document as a JSON**

structure, which initially consists of a document with the index fields you have mapped to fields extracted directly from the source data, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content

When the documents in the data source contain images, you can configure the indexer to extract the image data and place each image in a **normalized_images** collection, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content
  - normalized_images
    - image0
    - image1

Normalizing the image data in this way enables you to use the collection of images as an input for skills that extract information from image data.

Each skill adds fields to the document, so for example a skill that detects the language in which a document is written might store its output in a **language** field, like this:

- document
  - metadata_storage_name
  - metadata_author
  - content
  - normalized_images
    - image0
    - image1
  - language

The document is structured hierarchically, and the skills are applied to a specific context within the hierarchy, enabling you to run the skill for each item at a particular level of the document. For example, you could run an **optical character recognition (OCR)** skill for each image in the normalized images collection to extract any **text** they contain:

- document
  - metadata_storage_name

    - metadata_author
    - content
    - normalized_images
        - image0
            - Text
        - image1
            - Text
    - language

The output fields from each skill can be used as inputs for other skills later in the pipeline, which in turn store their outputs in the document structure. For example, we could use a merge skill to combine the original text content with the text extracted from each image to create a new **merged_content** field that contains all of the text in the document, including image text.

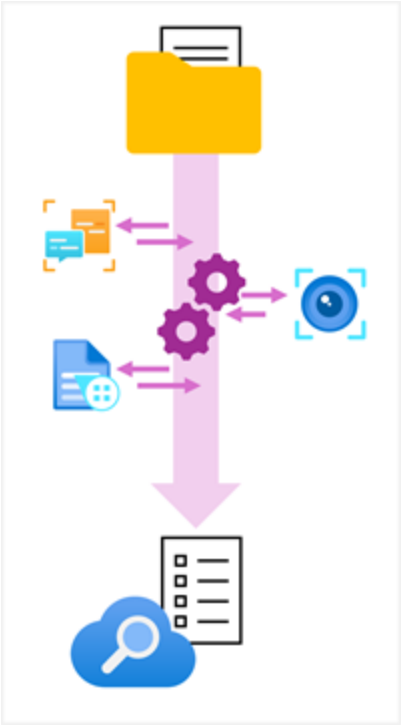
- document
    - metadata_storage_name
    - metadata_author
    - content
    - normalized_images
        - image0
            - Text
        - image1
            - Text
    - language
    - merged_content

The fields in the final document structure at the end of the pipeline are mapped to index fields by the indexer in one of two ways:

- **Fields extracted directly from the source data are all mapped to index fields**. These mappings can be **implicit** *(fields are automatically mapped to in fields with the same name in the index)* or **explicit** *(a mapping is defined to match a source field to an index field, often to rename the field to something more useful or to apply a function to the data value as it is mapped)*.
- Output fields from the skills in the skillset are explicitly mapped from their hierarchical location in the output to the target field in the index.

Enrich extracted data with AI skills

The enrichment pipeline that is orchestrated by an indexer uses a skillset of AI skills to create AI-enriched fields. **The indexer applies each skill in order, refining the index document at each step.**



Built-in skills

Azure AI Search provides a collection of built-in skills that you can include in a skillset for your indexer. **Built-in skills include functionality from Azure AI services such as Azure AI Vision and Azure AI Language**, enabling you to apply enrichments such as:

- Detecting the **language** that text is written in.
- Detecting and extracting **places, locations, and other entities in the text**.
- Determining and extracting **key phrases** within a body of text.
- **Translating** text.
- Identifying and extracting (or removing) **personally identifiable information (PII)** within the text.
- Extracting **text from images**.
- Generating **captions and tags** to describe images.

To use the built-in skills, your indexer must have access to an Azure AI services resource. You can use a restricted Azure AI search resource that is included in Azure AI Search (and which is limited to indexing 20 or fewer documents) or you can attach an Azure AI services resource in your Azure subscription (which must be in the same region as your Azure AI Search resource).

Custom skills

You can further extend the enrichment capabilities of your index by creating custom skills. As the name suggests, custom skills perform custom logic on input data from your index document to return new field values that can be incorporated into the index. Often, **custom skills are "wrappers" around services that are specifically designed to extract data from documents**. For example, you could implement a custom skill as an Azure Function, and use it to pass data from your index document to an **Azure AI Document Intelligence** model, which can extract fields from a form.

> Tip: To learn mode about using custom skills with Azure AI Search, see Add a custom skill to an Azure AI Search enrichment pipeline in the Azure AI Search documentation.

Search an index

The index is the searchable result of the indexing process. It consists of *a collection of JSON documents, with fields that contain the values extracted during indexing*. Client applications can query the index to retrieve, filter, and sort information.

Each index field can be configured with the following attributes:

- **key**: Fields that define a unique key for index records.
- **searchable**: Fields that can be queried using full-text search.

- **filterable**: Fields that can be included in filter expressions to return only documents that **match specified constraints**.
- **sortable**: Fields that can be used to **order** the results.
- **facetable**: Fields that can be used to determine values for facets (user interface elements used to filter the results based on a list of **known field values**).
- **retrievable**: Fields that can be included in search results (by default, all fields are retrievable unless this attribute is explicitly removed).

Full-text search

While you could retrieve index entries based on simple field value matching, most search solutions use full-text search semantics to query an index.

Full-text search describes search solutions that parse text-based document contents to find query terms. Full-text search queries in Azure AI Search are based on the Lucene query syntax, which provides a rich set of query operations for searching, filtering, and sorting data in indexes. Azure AI Search supports two variants of the Lucene syntax:

- **Simple** - An intuitive syntax that makes it easy to perform basic searches that match literal query terms submitted by a user.
- **Full** - An extended syntax that supports complex filtering, regular expressions, and other more sophisticated queries.

Client applications submit queries to Azure AI Search by specifying a search expression along with other parameters that determine how the expression is evaluated and the results returned. Some common parameters submitted with a query include:

- **search** - A search expression that includes the terms to be found.
- **queryType** - The **Lucene syntax** to be evaluated (simple or full).
- **searchFields** - The index fields to be searched.
- **select** - The fields to be included in the results.
- **searchMode** - Criteria for including results based on multiple search terms. For example, suppose you search an index of travel-related documents for comfortable hotel. A searchMode value of Any returns documents that contain "comfortable", "hotel", or both; while a searchMode value of All restricts results to documents that contain both "comfortable" and "hotel".

Query processing consists of four stages:

1. **Query parsing**. The search expression is *evaluated and reconstructed as a tree of appropriate subqueries*. Subqueries might include **term queries** (finding specific individual words in the

search expression - for example hotel), **phrase queries** (finding multi-term phrases specified in quotation marks in the search expression - for example, "free parking"), and **prefix queries** (finding terms with a specified prefix - for example air*, which would match airway, air-conditioning, and airport).

2. **Lexical analysis** - The query terms are analyzed and refined based on linguistic rules. For example, *text is converted to lower case and nonessential stopwords (such as "the", "a", "is", and so on) are removed*. Then words are *converted to their root form* (for example, "comfortable" might be simplified to "comfort") and *composite words are split into their constituent terms*.
3. **Document retrieval** - The *query terms are matched against the indexed terms*, and the set of matching documents is identified.
4. **Scoring** - A *relevance score* is assigned to each result based on a term frequency/inverse document frequency (TF/IDF) calculation.

> Tip: For more information about querying an index, and details about simple and full syntax, see Query types and composition in Azure AI Search in the Azure AI Search documentation.

It's common in a search solution for users to want to refine query results by filtering and sorting based on field values. Azure AI Search supports both of these capabilities through the search query API.

Filtering results

You can apply filters to queries in two ways:

- By including filter criteria in a simple search expression.
- By providing an OData filter expression as a **$filter** parameter with a *full* syntax search expression.

You can apply a filter to any *filterable* field in the index.

For example, suppose you want to find documents containing the text *London* that have an **author** field value of *Reviewer*.

You can achieve this result by submitting the following simple search expression:

```
search=London+author='Reviewer'
queryType=Simple
```

Alternatively, you can use an OData filter in a **$filter** parameter with a **full Lucene search expression** like this:

```
search=London
$filter=author eq 'Reviewer'
```

```
queryType=Full
```

> Note: OData **$filter** expressions are case-sensitive!

Filtering with facets

Facets are a useful way to present users with filtering criteria based on field values in a result set. They work best when a field has a small number of discrete values that can be displayed as links or options in the user interface.

To use facets, you must specify **facetable** fields for which you want to retrieve the possible values in an initial query. For example, you could use the following parameters to return all of the possible values for the author field:

```
search=*
facet=author
```

The results from this query include a collection of discrete facet values that you can display in the user interface for the user to select. Then in a subsequent query, you can use the selected facet value to filter the results:

```
search=*
$filter=author eq 'selected-facet-value-here'
```

Sorting results

By default, results are sorted based on the relevancy score assigned by the query process, with the highest scoring matches listed first. However, you can override this sort order by including an OData **orderby** parameter that specifies one or more **sortable** fields and a **sort order (asc or desc)**.

For example, to sort the results so that the most recently modified documents are listed first, you could use the following parameter values:

```
search=*
$orderby=last_modified desc
```

> Tip: For more information about using filters, see Filters in Azure AI Search in the Azure AI Search documentation.

Persist extracted information in a knowledge store

While the index might be considered the primary output from an indexing process, the enriched data it contains might also be useful in other ways. For example:

- Since the index is essentially a collection of JSON objects, each representing an indexed record, it might be useful to *export the objects as JSON files for integration into a data orchestration process for* **extract, transform, and load (ETL) operations**.
- You may want to *normalize the index records into* **a relational schema of tables** for analysis and reporting.
- Having *extracted embedded images from documents during the indexing process*, you might want to save those images as files.

Azure AI Search supports these scenarios by enabling you to define a **knowledge** store in the skillset that encapsulates your enrichment pipeline. The knowledge store consists of *projections of the enriched data*, which can be *JSON objects, tables, or image files*. When an indexer runs the pipeline to create or update an index, the projections are generated and persisted in the knowledge store.

> Tip: To learn more about using a knowledge store, see Knowledge store in Azure AI Search in the Azure AI Search documentation.

Exercise - Create a knowledge mining solution

It's time to out what you've learned into practice!

In this exercise, you use Azure AI Search to extract and enrich information from documents into a searchable index and a knowledge store.

Create an knowledge mining solution

In this exercise, you use AI Search to index a set of documents maintained by Margie's Travel, a fictional travel agency. The indexing process involves using AI skills to extract key information to make them searchable, and generating a knowledge store containing data assets for further analysis.

Module assessment

1. Which component of an Azure AI Search solution is scheduled to extract and enrich data to populate an index? **Indexer**.
2. Which service supports built-in AI skills in Azure AI Search? **Azure AI Services**
3. Which kind of projection results in a relational data schema for extracted fields? **Table**

Summary

In this module, you've learned how Azure AI Search enables you to build **intelligent search and knowledge mining** solutions by **indexing and enriching data** from various sources. You *explored the indexing process, the use of AI skills for data enrichment, and how to persist enriched data in a knowledge store for further analysis and integration*.

With these skills, you're now equipped to design and implement solutions that unlock valuable insights from your data using Azure AI Search.

> Tip: To learn more about Azure AI Search, see the Azure AI Search documentation.

Create an knowledge mining solution

In this exercise, you use AI Search to index a set of documents maintained by Margie's Travel, a fictional travel agency. The indexing process involves using AI skills to extract key information to make them searchable, and generating a knowledge store containing data assets for further analysis.

While this exercise is based on Python, you can develop similar applications using multiple language-specific SDKs; including:

- Azure AI Search client library for Python
- Azure AI Search client library for Microsoft .NET
- Azure AI Search client library for JavaScript

This exercise takes approximately **40** minutes.

Create Azure resources

The solution you will create for Margie's Travel requires multiple resources in your Azure subscription. In this exercise, you'll create them directly in the Azure portal. You could also create them by using a script, or an ARM or BICEP template; or you could create an Azure AI Foundry project that includes an Azure AI Search resource.

> **Important**: Your Azure resources should be created in the same location!

Create an Azure AI Search resource

1. In a web browser, open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials.
2. Select the **+ Create a resource** button, search for `Azure AI Search`, and create an **Azure AI Search** resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Service name**: *A valid name for your search resource*
    - **Location**: *Any available location*
    - **Pricing tier**: Free
3. Wait for deployment to complete, and then go to the deployed resource.
4. Review the **Overview** page on the blade for your Azure AI 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 storage account

1. Return to the home page, and then create a **Storage account** resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *The same resource group as your Azure AI Search and Azure AI Services resources*
    - **Storage account name**: *A valid name for your storage resource*
    - **Region**: *The same region as your Azure AI Search resource*
    - **Primary service**: Azure Blob Storage or Azure Data Lake Storage Gen 2
    - **Performance**: Standard
    - **Redundancy**: Locally-redundant storage (LRS)
2. Wait for deployment to complete, and then go to the deployed resource.

> **Tip**: Keep the storage account portal page open - you will use it in the next procedure.

Upload documents to Azure Storage

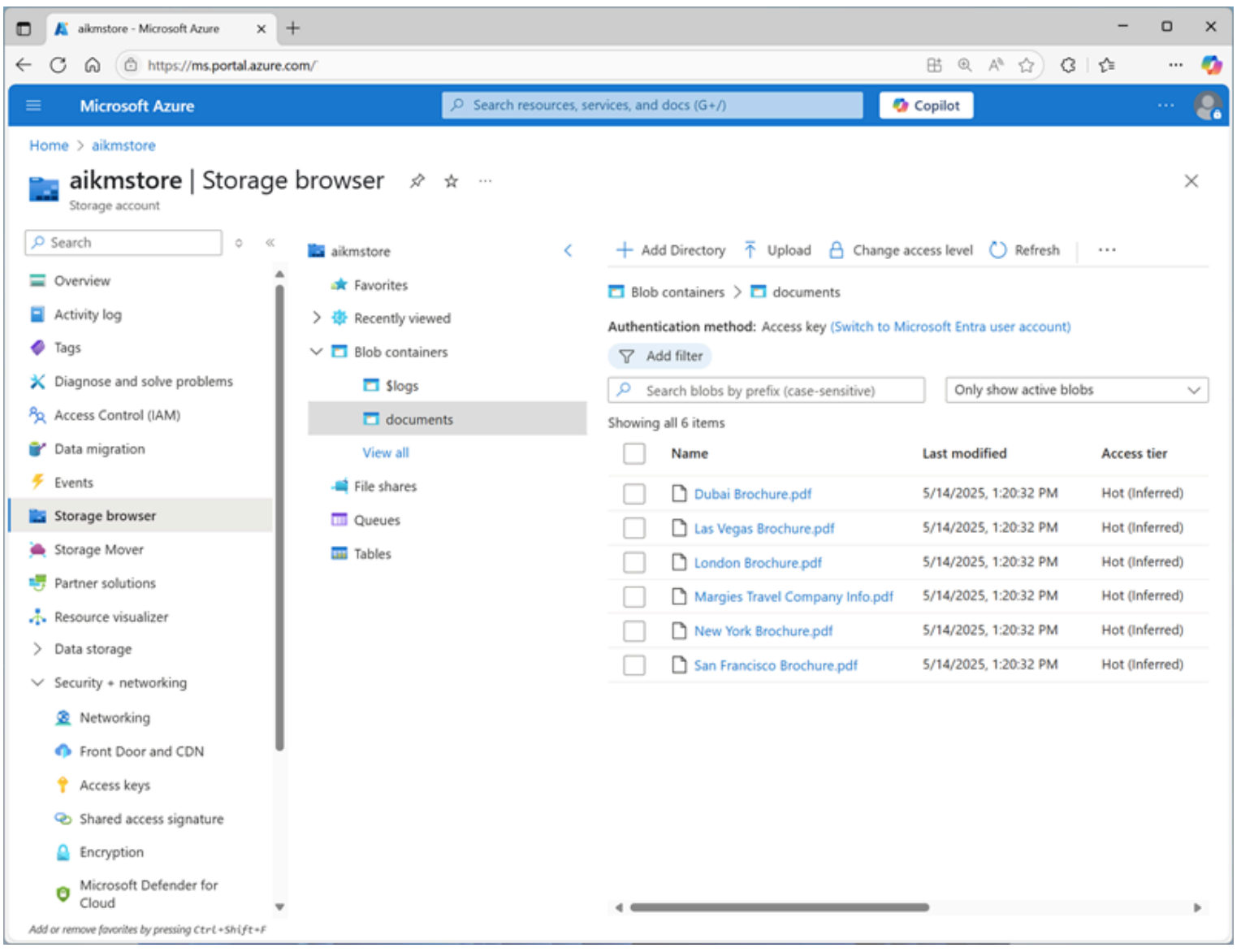
Your knowledge mining solution will extract information from travel brochure documents in an Azure Storage blob container.

1. In a new browser tab, download documents.zip from
   ```
   https://github.com/microsoftlearning/mslearn-ai-information-extraction/raw/main/Labfiles/knowledge/documents.zip
   ```
   and save it in a local folder.
2. Extract the downloaded *documents.zip* file and view the travel brochure files it contains. You'll extract and index information from these files.
3. In the browser tab containing the Azure portal page for your storage account, in the navigation pane on the left, select **Storage browser**.
4. In the storage browser, select **Blob containers**.

   Currently, your storage account should contain only the default **$logs** container.
5. In the toolbar, select **+ Container** and create a new container with the following settings:
   - **Name**: `documents`
   - **Anonymous access level**: Private (no anonymous access)*

> **Note**: *Unless you enabled the option to allow anonymous container access when creating your storage account, you won't be able to select any other setting!

6. Select the **documents** container to open it, and then use the **Upload** toolbar button to upload the .pdf files you extracted from **documents.zip** previously into the root of the container, as shown here:



Create and run an indexer

Now that you have the documents in place, you can create an indexer to extract information from them.

1. In the Azure portal, browse to your Azure AI 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-documents`
- **Data to extract**: Content and metadata
- **Parsing mode**: Default
- **Subscription**: *Your Azure subscription*
- **Connection string**:
    - Select **Choose an existing connection**
    - Select your storage account
    - Select the **documents** container
- **Managed identity authentication**: None
- **Container name**: documents
- **Blob folder**: *Leave this blank*
- **Description**: `Travel brochures`

3. Proceed to the next step (**Add cognitive skills**), which has three expandable sections to complete.

4. In the **Attach Azure AI Services** section, select **Free (limited enrichments)***.

> **Note**: *The free Azure AI Services resource for Azure AI Search can be used to index a maximum of 20 documents. In a real solution, you should create an Azure AI Services resource in your subscription to enable AI enrichment for a larger number of documents.

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
Text Cognitive Skills		
Extract people names		people
Extract location names		locations
Extract key phrases		keyphrases
Image Cognitive Skills		
Generate tags from images		imageTags
Generate captions from images		imageCaption

    Double-check your selections (it can be difficult to change them later).

6. In the **Save enrichments to a knowledge store** section:

    - Select only the following checkboxes (an error will be displayed, you'll resolve that shortly):
        - **Azure file projections**:
            - Image projections
        - **Azure table projections**:
            - Documents
                - Key phrases
        - **Azure blob projections**:

        - Document
    - Under **Storage account connection string** (beneath the error messages):
        - Select **Choose an existing connection**
        - Select your storage account
        - Select the **documents** container (*this is only required to select the storage account in the browse interface - you'll specify a different container name for the extracted knowledge assets!*)
    - Change the **Container name** to `knowledge-store`.
7. Proceed to the next step (**Customize target index**), where you'll specify the fields for your index.
8. Change the **Index name** to `margies-index`.
9. Ensure that the **Key** is set to **metadata_storage_path**, leave the **Suggester name** blank, and ensure **Search mode** is **analyzingInfixMatching**.
10. 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	✓	✓	✓		✓
locations	✓	✓			✓
people	✓	✓			✓
keyphrases	✓	✓			✓

    Double-check your selections, paying particular attention to ensure that the correct **Retrievable**, **Filterable**, **Sortable**, **Facetable**, and **Searchable** options are selected correctly for each field (it can be difficult to change them later).
11. Proceed to the next step (**Create an indexer**), where you'll create and schedule the indexer.
12. Change the **Indexer name** to `margies-indexer`.
13. Leave the **Schedule** set to **Once**.
14. 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.
    - Saves the extracted data assets to the knowledge store.
15. In the navigation pane on the left, under **Search management** view the **Indexers** page, 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. Return to the **Overview** page for your Azure AI Search resource, and on the toolbar, 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.
3. In the **View** menu, select **JSON view** and note that the JSON request for the search is shown, like this:

Code Copy

```
{
  "search": "*",
  "count": true
}
```

4. The results include a **@odata.count** field at the top of the results that indicates the number of documents returned by the search.

5. Modify the JSON request to include the **select** parameter as shown here:

Code Copy

```
{
  "search": "*",
  "count": true,
  "select": "metadata_storage_name,locations"
}
```

This time the results include only the file name and any locations mentioned in the document content. The file name is in the **metadata_storage_name** field, which was extracted from the source document. The **locations** field was generated by an AI skill.

6. Now try the following query string:

Code Copy

```
{
  "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.

7. Let’s try one more query:

Code Copy

```
{
    "search": "New York",
    "count": true,
    "select": "metadata_storage_name,keyphrases",
    "filter": "metadata_storage_size lt 380000"
}
```

This query returns the filename and key phrases for any documents mentioning “New York” that are smaller than 380,000 bytes in size.

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, close the search explorer page and return to the **Overview** page for your Azure AI 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. In the navigation pane on the left, expand **Settings** and view 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 AI Search SDK

1. Use the **[>_]** button to the right of the search bar at the top of the Azure portal to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment with no storage in your subscription.

   The cloud shell provides a command-line interface in a pane at the bottom of the Azure portal. You can resize or maximize this pane to make it easier to work in. Initially, you'll need to see both the cloud shell and the Azure portal (so you can find and copy the endpoint and key you'll need).

2. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   Ensure you've switched to the classic version of the cloud shell before continuing.

3. In the cloud shell pane, enter the following commands to clone the GitHub repo containing the code files for this exercise (type the command, or copy it to the clipboard and then right-click in the command line and paste as plain text):

   Code Copy

   ```
   rm -r mslearn-ai-info -f
   git clone https://github.com/microsoftlearning/mslearn-ai-information-extraction mslearn-ai-info
   ```

   > **Tip**: As you enter commands into the cloudshell, the output may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

4. After the repo has been cloned, navigate to the folder containing the application code files:

   Code Copy

   ```
   cd mslearn-ai-info/Labfiles/knowledge/python
   ls -a -l
   ```

5. Install the Azure AI Search SDK and Azure identity packages by running the following commands:

   Code Copy

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-identity azure-search-documents==11.5.1
```

6. Run the following command to edit the configuration file for your app:

Code Copy

```
code .env
```

The configuation file is opened in a code editor.

7. Edit the configuration file to replace the following placeholder values:
    - **your_search_endpoint** (*replace with the endpoint for your Azure AI Search resource*)
    - **your_query_key** *(replace with the query key for your Azure AI Search resource*)
    - **your_index_name** (*replace with the name of your index, which should be* `margies-index`)
8. When you've updated the placeholders, use the **CTRL+S** command to save the file and then use the **CTRL+Q** command to close it.

> **Tip**: Now that you've copied the endpoint and key from the Azure portal, you might want to maximize the cloud shell pane to make it easier to work in.

9. Run the following command to open the code file for your app:

Code Copy

```
code search-app.py
```

The code file is opened in a code editor.

10. Review the code, and note that it performs the following actions:
    - Retrieves the configuration settings for your Azure AI Search resource and index from the configuration file you edited.
    - Creates a **SearchClient** with the endpoint, key, and index name to connect to your search service.
    - Prompts the user for a search query (until they enter "quit")
    - Searches the index using the query, returning the following fields (ordered by metadata_storage_name):
        - metadata_storage_name
        - locations
        - people
        - keyphrases
    - Parses the search results that are returned to display the fields returned for each document in the result set.
11. Close the code editor pane (*CTRL+Q*), keeping the cloud shell command line console pane open
12. Enter the following command to run the app:

Code Copy

```
python search-app.py
```

13. When prompted, enter a query such as `London` and view the results.
14. Try another query, such as `flights` .
15. When you're finished testing the app, enter `quit` to close it.

16. Close the Cloud shell, returning to the Azure portal.

View the knowledge store

After you have run an indexer that uses a skillset to create a knowledge store, the enriched data extracted by the indexing process is persisted in the knowledge store projections.

View object projections

The *object* projections defined in the Margie's Travel skillset consist of a JSON file for each indexed document. These files are stored in a blob container in the Azure Storage account specified in the skillset definition.

1. In the Azure portal, view the Azure Storage account you created previously.
2. Select the **Storage browser** tab (in the pane on the left) to view the storage account in the storage explorer interface in the Azure portal.
3. Expand **Blob containers** to view the containers in the storage account. In addition to the **documents** container where the source data is stored, there should be two new containers: **knowledge-store** and **margies-skillset-image-projection**. These were created by the indexing process.
4. Select the **knowledge-store** container. It should contain a folder for each indexed document.
5. Open any of the folders, and then select the **objectprojection.json** file it contains and use the **Download** button on the toolbar to download and open it. Each JSON file contains a representation of an indexed document, including the enriched data extracted by the skillset as shown here (formatted to make it easier to read).

Code Copy

```
{
    "metadata_storage_content_type": "application/pdf",
    "metadata_storage_size": 388622,
    "<more_metadata_fields>": "...",
    "key_phrases":[
        "Margie’s Travel",
        "Margie's Travel",
        "best travel experts",
        "world-leading travel agency",
        "international reach"
        ],
    "locations":[
        "Dubai",
        "Las Vegas",
        "London",
        "New York",
        "San Francisco"
        ],
    "image_tags":[
        "outdoor",
        "tree",
        "plant",
        "palm"
        ],
    "more fields": "..."
}
```

The ability to create *object* projections like this enables you to generate enriched data objects that can be incorporated into an enterprise data analysis solution.

View file projections

The *file* projections defined in the skillset create JPEG files for each image that was extracted from the documents during the indexing process.

1. In the *Storage browser* interface in the Azure portal, select the **margies-skillset-image-projection** blob container. This container contains a folder for each document that contained images.
2. Open any of the folders and view its contents - each folder contains at least one *.jpg file.
3. Open any of the image files, and download and view it to see the image.

The ability to generate *file* projections like this makes indexing an efficient way to extract embedded images from a large volume of documents.

View table projections

The *table* projections defined in the skillset form a relational schema of enriched data.

1. In the *Storage browser* interface in the Azure portal, expand **Tables**.
2. Select the **margiesSkillsetDocument** table to view data. This table contains a row for each document that was indexed:
3. View the **margiesSkillsetKeyPhrases** table, which contains a row for each key phrase extracted from the documents.

The ability to create *table* projections enables you to build analytical and reporting solutions that query a relational schema. The automatically generated key columns can be used to join the tables in queries - for example to return all of the key phrases extracted from a specific document.

Clean-up

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

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

More information

To learn more about Azure AI Search, see the Azure AI Search documentation.