Provision an Azure AI Vision resource

Develop an image analysis app with the Azure AI Vision SDK

Clean up resources

Analyze images

Azure AI Vision is an artificial intelligence capability that enables software systems to interpret visual input by analyzing images. In Microsoft Azure, the **Vision** Azure AI service provides pre-built models for common computer vision tasks, including analysis of images to suggest captions and tags, detection of common objects and people. You can also use the Azure AI Vision service to remove the background or create a foreground matting of images.

> **Note**: This exercise is based on pre-release SDK software, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

While this exercise is based on the Azure Vision Python SDK, you can develop vision applications using multiple language-specific SDKs; including:

- Azure AI Vision Analysis for JavaScript
- Azure AI Vision Analysis for Microsoft .NET
- Azure AI Vision Analysis for Java

This exercise takes approximately **30** minutes.

Provision an Azure AI Vision resource

If you don't already have one in your subscription, you'll need to provision an Azure AI Vision resource.

> **Note**: In this exercise, you'll use a standalone **Computer Vision** resource. You can also use Azure AI Vision services in an *Azure AI Services* multi-service resource, either directly or in an *Azure AI Foundry* project.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Close any welcome messages or tips that are displayed.
2. Select **Create a resource**.
3. In the search bar, search for `Computer Vision`, select **Computer Vision**, and create the resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Choose from **East US**, **West US**, **France Central**, **Korea Central**, **North Europe**, **Southeast Asia**, **West Europe**, or **East Asia****
    - **Name**: *A valid name for your Computer Vision resource*
    - **Pricing tier**: Free F0

    *Azure AI Vision 4.0 full feature sets are currently only available in these regions.
4. Select the required checkboxes and create the resource.
5. Wait for deployment to complete, and then view the deployment details.
6. When the resource has been deployed, go to it and under the **Resource management** node in the navigation pane, view its **Keys and Endpoint** page. You will need the endpoint and one of the keys from this page in the next procedure.

Develop an image analysis app with the Azure AI Vision SDK

In this exercise, you'll complete a partially implemented client application that uses the Azure AI Vision SDK to analyze images.

Prepare the application configuration

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 with no storage in your subscription.

   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. 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. Resize the cloud shell pane so you can still see the **Keys and Endpoint** page for your Computer Vision resource.

   > **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

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-vision -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
   ```

   > **Tip**: As you paste 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, use the following command to navigate to and view the folder containing the application code files:

   Code Copy

   ```
   cd mslearn-ai-vision/Labfiles/analyze-images/python/image-analysis
   ls -a -l
   ```

   The folder contains application configuration and code files for your app. It also contains a **/images** subfolder, which contains some image files for your app to analyze.

6. Install the Azure AI Vision SDK package and other required packages by running the following commands:

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-ai-vision-imageanalysis==1.0.0
   ```

7. Enter the following command to edit the configuration file for your app:

   Code Copy

```
code .env
```

The file is opened in a code editor.

8. In the code file, update the configuration values it contains to reflect the **endpoint** and an authentication **key** for your Computer Vision resource (copied from its **Keys and Endpoint** page in the Azure portal).
9. After you've replaced the placeholders, 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 suggest a caption

1. In the cloud shell command line, enter the following command to open the code file for the client application:

Code Copy

```
code image-analysis.py
```

> **Tip**: You might want to maximize the cloud shell pane and move the split-bar between the command line cosole and the code editor so you can see the code more easily.

2. In the code file, find the comment **Import namespaces**, and add the following code to import the namespaces you will need to use the Azure AI Vision SDK:

Code Copy

```
# import namespaces
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential
```

3. In the **Main** function, note that the code to load the configuration settings and determine the image file to be analyzed has been provided. Then find the comment **Authenticate Azure AI Vision client** and add the following code to create and authenticate a Azure AI Vision client object (be sure to maintain the correct indentation levels):

Code Copy

```
# Authenticate Azure AI Vision client
cv_client = ImageAnalysisClient(
     endpoint=ai_endpoint,
     credential=AzureKeyCredential(ai_key))
```

4. In the **Main** function, under the code you just added, find the comment **Analyze image** and add the following code:

Code Copy

```
# Analyze image
with open(image_file, "rb") as f:
     image_data = f.read()
print(f'\nAnalyzing {image_file}\n')

result = cv_client.analyze(
     image_data=image_data,
     visual_features=[
         VisualFeatures.CAPTION,
         VisualFeatures.DENSE_CAPTIONS,
         VisualFeatures.TAGS,
         VisualFeatures.OBJECTS,
         VisualFeatures.PEOPLE],
)
```

5. Find the comment **Get image captions**, add the following code to display image captions and dense captions:

Code Copy

```
# Get image captions
if result.caption is not None:
     print("\nCaption:")
     print(" Caption: '{}' (confidence: {:.2f}%)".format(result.caption.text,
result.caption.confidence * 100))

if result.dense_captions is not None:
     print("\nDense Captions:")
     for caption in result.dense_captions.list:
         print(" Caption: '{}' (confidence: {:.2f}%)".format(caption.text,
caption.confidence * 100))
```

6. Save your changes (*CTRL+S*) and resize the panes so you can clearly see the command line console while keeping the code editor open. Then enter the following command to run the program with the argument **images/street.jpg**:

Code Copy

```
python image-analysis.py images/street.jpg
```

7. Observe the output, which should include a suggested caption for the **street.jpg** image, which looks like this:

8. Run the program again, this time with the argument **images/building.jpg** to see the caption that gets generated for the **building.jpg** image, which looks like this:

9. Repeat the previous step to generate a caption for the **images/person.jpg** file, which looks like this:

Add code to generate suggested tags

It can sometimes be useful to identify relevant *tags* that provide clues about the contents of an image.

1. In the code editor, in the **AnalyzeImage** function, find the comment **Get image tags** and add the following code:

Code Copy

```
# Get image tags
if result.tags is not None:
     print("\nTags:")
     for tag in result.tags.list:
         print(" Tag: '{}' (confidence: {:.2f}%)".format(tag.name, tag.confidence * 100))
```

2. Save your changes (*CTRL+S*) and run the program with the argument **images/street.jpg**, observing that in addition to the image caption, a list of suggested tags is displayed.
3. Rerun the program for the **images/building.jpg** and **images/person.jpg** files.

Add code to detect and locate objects

1. In the code editor, in the **AnalyzeImage** function, find the comment **Get objects in the image** and add the following code to list the objects detected in the image, and call the provided function to annotate an image with the detected objects:

Code Copy

```
# Get objects in the image
if result.objects is not None:
     print("\nObjects in image:")
     for detected_object in result.objects.list:
         # Print object tag and confidence
         print(" {} (confidence: {:.2f}%)".format(detected_object.tags[0].name,
detected_object.tags[0].confidence * 100))
     # Annotate objects in the image
     show_objects(image_file, result.objects.list)
```

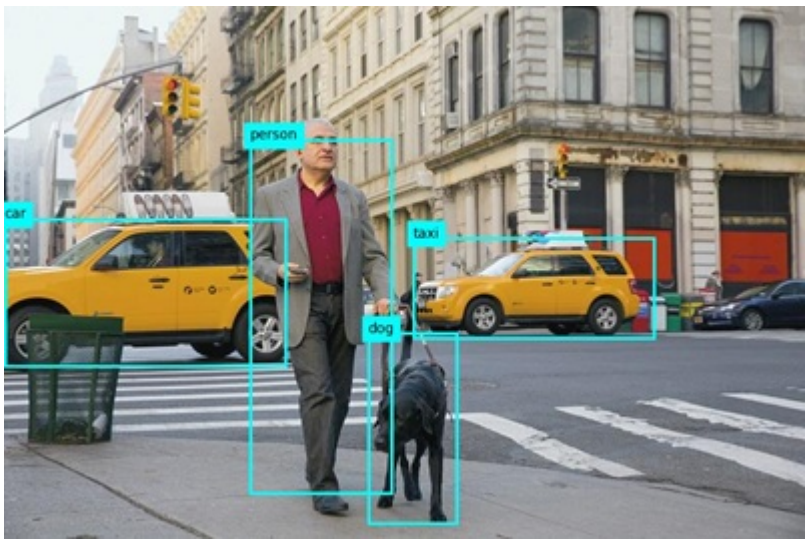
2. Save your changes (*CTRL+S*) and run the program with the argument **images/street.jpg**, observing that in addition to the image caption and suggested tags; a file named **objects.jpg** is generated.

3. Use the (Azure cloud shell-specific) **download** command to download the **objects.jpg** file:

Code Copy

```
download objects.jpg
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file. The image should look similar to this:



4. Rerun the program for the **images/building.jpg** and **images/person.jpg** files, downloading the generated objects.jpg file after each run.

Add code to detect and locate people

1. In the code editor, in the **AnalyzeImage** function, find the comment **Get people in the image** and add the following code to list any detected people with a confidence level of 20% or more, and call a provided function to annotate them in an image:

Code Copy

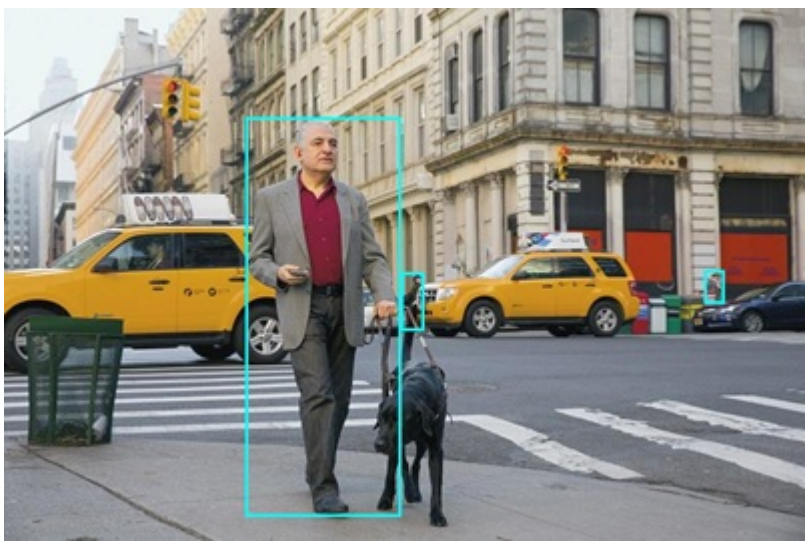
```
# Get people in the image
if result.people is not None:
     print("\nPeople in image:")

     for detected_person in result.people.list:
         if detected_person.confidence > 0.2:
             # Print location and confidence of each person detected
             print(" {} (confidence: {:.2f}%)".format(detected_person.bounding_box,
detected_person.confidence * 100))
     # Annotate people in the image
     show_people(image_file, result.people.list)
```

2. Save your changes (*CTRL+S*) and run the program with the argument **images/street.jpg**, observing that in addition to the image caption, suggested tags, and objects.jpg file; a list of person locations and file named **people.jpg** is generated.

3. Use the (Azure cloud shell-specific) **download** command to download the **objects.jpg** file:

```
download people.jpg
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file. The image should look similar to this:



4. Rerun the program for the **images/building.jpg** and **images/person.jpg** files, downloading the generated people.jpg file after each run.

> **Tip:** If you see bounding boxes returned from the model that don't make sense, check the JSON confidence score and try increasing the confidence score filtering in your app.

Clean up resources

If you've finished exploring Azure AI Vision, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs:

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.

2. In the top search bar, search for *Computer Vision*, and select the Computer Vision resource you created in this lab.

3. On the resource page, select **Delete** and follow the instructions to delete the resource.

Provision an Azure AI Vision resource

Develop a text extraction app with the Azure AI Vision SDK

Clean up resources

Read text in images

Optical character recognition (OCR) is a subset of computer vision that deals with reading text in images and documents. The **Azure AI Vision** Image Analysis service provides an API for reading text, which you'll explore in this exercise.

> **Note**: This exercise is based on pre-release SDK software, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

While this exercise is based on the Azure Vision Analysis Python SDK, you can develop vision applications using multiple language-specific SDKs; including:

- Azure AI Vision Analysis for JavaScript
- Azure AI Vision Analysis for Microsoft .NET
- Azure AI Vision Analysis for Java

This exercise takes approximately **30** minutes.

Provision an Azure AI Vision resource

If you don't already have one in your subscription, you'll need to provision an Azure AI Vision resource.

> **Note**: In this exercise, you'll use a standalone **Computer Vision** resource. You can also use Azure AI Vision services in an *Azure AI Services* multi-service resource, either directly or in an *Azure AI Foundry* project.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Close any welcome messages or tips that are displayed.
2. Select **Create a resource**.
3. In the search bar, search for `Computer Vision`, select **Computer Vision**, and create the resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Choose from **East US**, **West US**, **France Central**, **Korea Central**, **North Europe**, **Southeast Asia**, **West Europe**, or **East Asia****
    - **Name**: *A valid name for your Computer Vision resource*
    - **Pricing tier**: Free F0

    *Azure AI Vision 4.0 full feature sets are currently only available in these regions.
4. Select the required checkboxes and create the resource.
5. Wait for deployment to complete, and then view the deployment details.
6. When the resource has been deployed, go to it and under the **Resource management** node in the navigation pane, view its **Keys and Endpoint** page. You will need the endpoint and one of the keys from this page in the next procedure.

Develop a text extraction app with the Azure AI Vision SDK

In this exercise, you'll complete a partially implemented client application that uses the Azure AI Vision SDK to extract text from images.

Prepare the application configuration

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 with no storage in your subscription.

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. 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. Resize the cloud shell pane so you can still see the **Keys and Endpoint** page for your Computer Vision resource.

   > **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

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-vision -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
   ```

   > **Tip**: As you paste 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, use the following command to navigate to the application code files:

   Code Copy

   ```
   cd mslearn-ai-vision/Labfiles/ocr/python/read-text
   ls -a -l
   ```

   The folder contains application configuration and code files for your app. It also contains an **/images** subfolder, which contains some image files for your app to analyze.

6. Install the Azure AI Vision SDK package and other required packages by running the following commands:

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-ai-vision-imageanalysis==1.0.0
   ```

7. Enter the following command to edit the configuration file for your app:

   Code Copy

   ```
   code .env
   ```

   The file is opened in a code editor.

8. In the code file, update the configuration values it contains to reflect the **endpoint** and an authentication **key** for your Computer Vision resource (copied from its **Keys and Endpoint** page in the Azure portal).

9. After you've replaced the placeholders, 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 read text from an image

1. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code read-text.py
```

> **Tip**: You might want to maximize the cloud shell pane and move the split-bar between the command line cosole and the code editor so you can see the code more easily.

2. In the code file, find the comment **Import namespaces**, and add the following code to import the namespaces you will need to use the Azure AI Vision SDK:

```
# import namespaces
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential
```

3. In the **Main** function, the code to load the configuration settings and determine the file to be analyzed has been provided. Then find the comment **Authenticate Azure AI Vision client** and add the following language-specific code to create and authenticate an Azure AI Vision Image Analysis client object:

```
# Authenticate Azure AI Vision client
cv_client = ImageAnalysisClient(
     endpoint=ai_endpoint,
     credential=AzureKeyCredential(ai_key))
```

4. In the **Main** function, under the code you just added, find the comment **Read text in image** and add the following code to use the Image Analysis client to read the text in the image:

```
# Read text in image
with open(image_file, "rb") as f:
     image_data = f.read()
print (f"\nReading text in {image_file}")

result = cv_client.analyze(
     image_data=image_data,
     visual_features=[VisualFeatures.READ])
```

5. Find the comment **Print the text** and add the following code (including the final comment) to print the lines of text that were found and call a function to annotate them in the image (using the **bounding_polygon** returned for each line of text):

```
# Print the text
if result.read is not None:
     print("\nText:")

     for line in result.read.blocks[0].lines:
         print(f" {line.text}")
     # Annotate the text in the image
     annotate_lines(image_file, result.read)

     # Find individual words in each line
```

6. Save your changes (*CTRL+S*) but keep the code editor open in case you need to fix any typo's.

7. Resize the panes so you can see more of the console, then enter the following command to run the program:

Code Copy

```
python read-text.py images/Lincoln.jpg
```

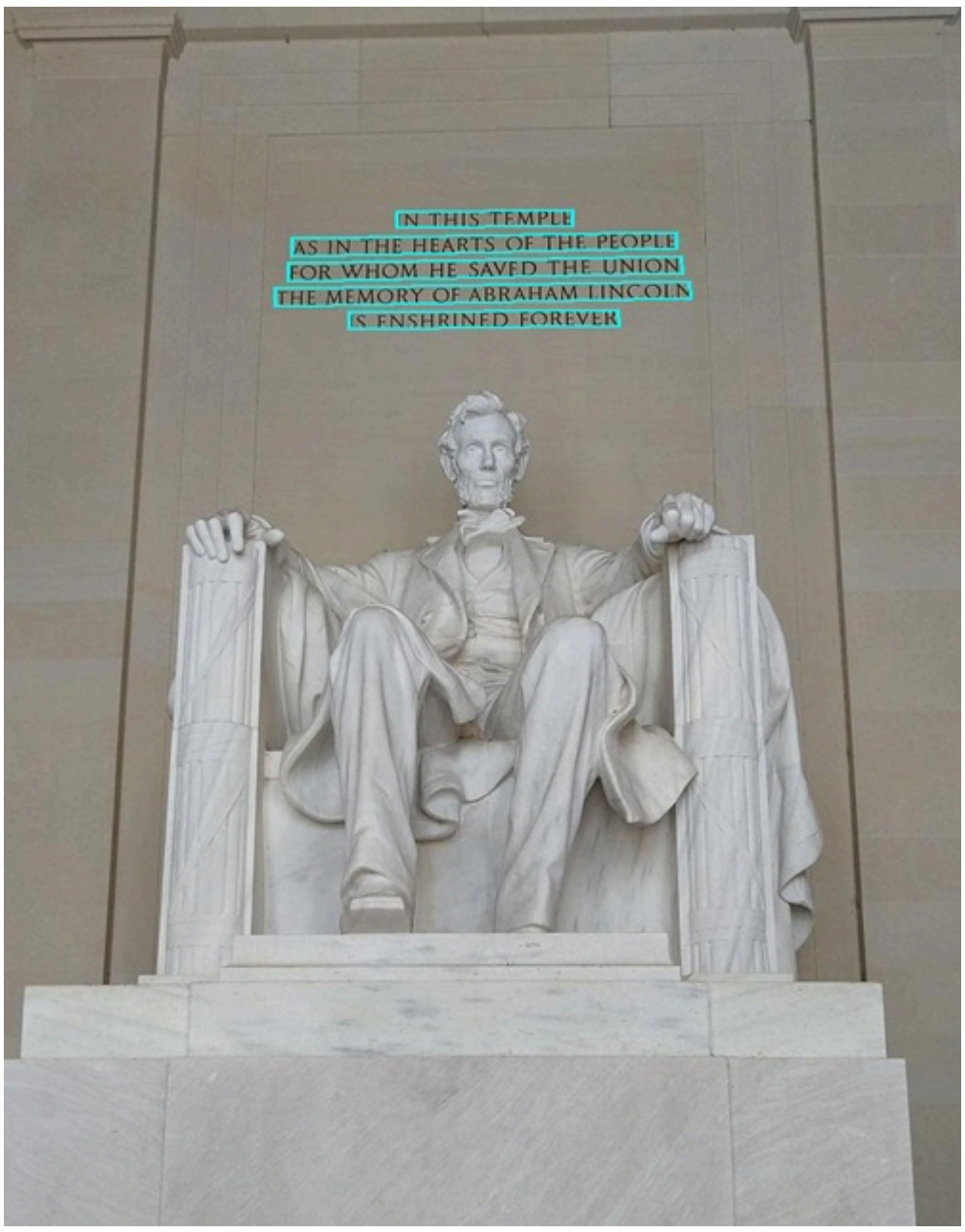
8. The program reads the text in the specified image file (*images/Lincoln.jpg*), which looks like this:

9. In the **read-text** folder, a **lines.jpg** image has been created. Use the (Azure cloud shell-specific) **download** command to download it:

Code Copy

```
download lines.jpg
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file. The image should look simlar to this:



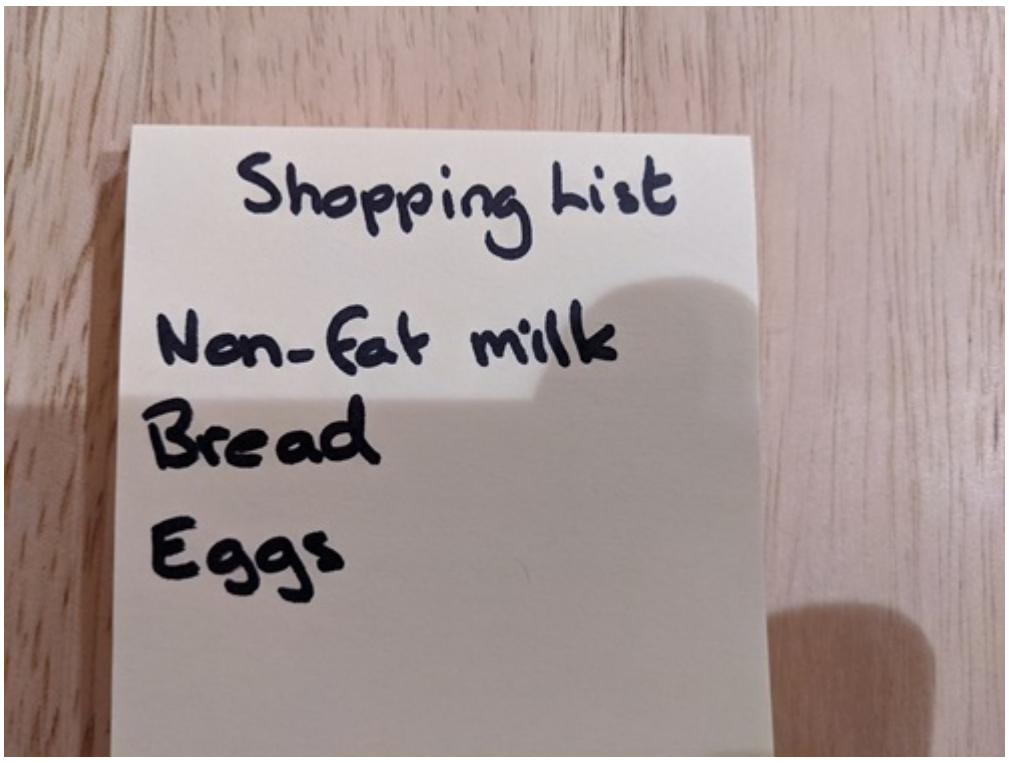
10. Run the program again, this time specifying the parameter *images/Business-card.jpg* to extract text from the following image:


```
python read-text.py images/Business-card.jpg
```

11. Download and view the resulting **lines.jpg** file:

```
download lines.jpg
```

12. Run the program one more time, this time specifying the parameter *images/Note.jpg* to extract text from this image:



Code Copy

```
python read-text.py images/Note.jpg
```

13. Download and view the resulting **lines.jpg** file:

Code Copy

```
download lines.jpg
```

Add code to return the position of individual words

1. Resize the panes so you can see more of the code file. Then find the comment **Find individual words in each line** and add the following code (being careful to maintain the correct indentation level):

Code Copy

```
# Find individual words in each line
print ("\nIndividual words:")
for line in result.read.blocks[0].lines:
     for word in line.words:
          print(f"  {word.text} (Confidence: {word.confidence:.2f}%)")
# Annotate the words in the image
annotate_words(image_file, result.read)
```

2. Save your changes (*CTRL+S*). Then, in the command line pane, rerun the program to extract text from *images/Lincoln.jpg*.
3. Observe the output, which should include each individual word in the image, and the confidence associated with their prediction.
4. In the **read-text** folder, a **words.jpg** image has been created. Use the (Azure cloud shell-specific) **download** command to download and view it:

Code Copy

```
download words.jpg
```

5. Rerun the program for *images/Business-card.jpg* and *images/Note.jpg*; viewing the **words.jpg** file generated for each image.

Clean up resources

If you've finished exploring Azure AI Vision, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs:

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. In the top search bar, search for *Computer Vision*, and select the Computer Vision resource you created in this lab.
3. On the resource page, select **Delete** and follow the instructions to delete the resource.

Provision an Azure AI Face API resource

Develop a facial analysis app with the Face SDK

Clean up resources

Detect and analyze faces

The ability to detect and analyze human faces is a core AI capability. In this exercise, you'll explore the **Face** service to work with faces.

> **Note**: This exercise is based on pre-release SDK software, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

While this exercise is based on the Azure Vision Face Python SDK, you can develop vision applications using multiple language-specific SDKs; including:

- Azure AI Vision Face for JavaScript
- Azure AI Vision Face for Microsoft .NET
- Azure AI Vision Face for Java

This exercise takes approximately **30** minutes.

> **Note**: Capabilities of Azure AI services that return personally identifiable information are restricted to customers who have been granted limited access. This exercise does not include facial recognition tasks, and can be completed without requesting any additional access to restricted features.

Provision an Azure AI Face API resource

If you don't already have one in your subscription, you'll need to provision an Azure AI Face API resource.

> **Note**: In this exercise, you'll use a standalone **Face** resource. You can also use Azure AI Face services in an *Azure AI Services* multi-service resource, either directly or in an *Azure AI Foundry* project.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Close any welcome messages or tips that are displayed.
2. Select **Create a resource**.
3. In the search bar, search for `Face`, select **Face**, and create the resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Choose any available region*
    - **Name**: *A valid name for your Face resource*
    - **Pricing tier**: Free F0
4. Create the resource and wait for deployment to complete, and then view the deployment details.
5. When the resource has been deployed, go to it and under the **Resource management** node in the navigation pane, view its **Keys and Endpoint** page. You will need the endpoint and one of the keys from this page in the next procedure.

Develop a facial analysis app with the Face SDK

In this exercise, you'll complete a partially implemented client application that uses the Azure Face SDK to detect and analyze human faces in images.

Prepare the application configuration

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 with no storage in your subscription.

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. 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. Resize the cloud shell pane so you can still see the **Keys and Endpoint** page for your Face resource.

   > **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

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-vision -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
   ```

   > **Tip**: As you paste 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, use the following command to navigate to the application code files:

   Code Copy

   ```
   cd mslearn-ai-vision/Labfiles/face/python/face-api
   ls -a -l
   ```

   The folder contains application configuration and code files for your app. It also contains an **/images** subfolder, which contains some image files for your app to analyze.

6. Install the Azure AI Vision SDK package and other required packages by running the following commands:

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-ai-vision-face==1.0.0b2
   ```

7. Enter the following command to edit the configuration file for your app:

   Code Copy

   ```
   code .env
   ```

   The file is opened in a code editor.

8. In the code file, update the configuration values it contains to reflect the **endpoint** and an authentication **key** for your Face resource (copied from its **Keys and Endpoint** page in the Azure portal).
9. After you've replaced the placeholders, 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 create a Face API client

1. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code analyze-faces.py
```

> **Tip**: You might want to maximize the cloud shell pane and move the split-bar between the command line cosole and the code editor so you can see the code more easily.

2. In the code file, find the comment **Import namespaces**, and add the following code to import the namespaces you will need to use the Azure AI Vision SDK:

```
# Import namespaces
from azure.ai.vision.face import FaceClient
from azure.ai.vision.face.models import FaceDetectionModel, FaceRecognitionModel,
FaceAttributeTypeDetection01
from azure.core.credentials import AzureKeyCredential
```

3. In the **Main** function, note that the code to load the configuration settings and determine the image to be analyzed has been provided. Then find the comment **Authenticate Face client** and add the following code to create and authenticate a **FaceClient** object:

```
# Authenticate Face client
face_client = FaceClient(
     endpoint=cog_endpoint,
     credential=AzureKeyCredential(cog_key))
```

Add code to detect and analyze faces

1. In the code file for your application, in the **Main** function, find the comment **Specify facial features to be retrieved** and add the following code:

```
# Specify facial features to be retrieved
features = [FaceAttributeTypeDetection01.HEAD_POSE,
            FaceAttributeTypeDetection01.OCCLUSION,
            FaceAttributeTypeDetection01.ACCESSORIES]
```

2. In the **Main** function, under the code you just added, find the comment **Get faces** and add the following code to print the facial feature information and call a function that annotates the image with the bounding box for each detected face (based on the **face_rectangle** property of each face):

```
# Get faces
with open(image_file, mode="rb") as image_data:
     detected_faces = face_client.detect(
         image_content=image_data.read(),
         detection_model=FaceDetectionModel.DETECTION01,
         recognition_model=FaceRecognitionModel.RECOGNITION01,
         return_face_id=False,
         return_face_attributes=features,
     )

face_count = 0
if len(detected_faces) > 0:
     print(len(detected_faces), 'faces detected.')
     for face in detected_faces:

         # Get face properties
         face_count += 1
         print('\nFace number {}'.format(face_count))
         print(' - Head Pose (Yaw): {}'.format(face.face_attributes.head_pose.yaw))
         print(' - Head Pose (Pitch): {}'.format(face.face_attributes.head_pose.pitch))
         print(' - Head Pose (Roll): {}'.format(face.face_attributes.head_pose.roll))
         print(' - Forehead occluded?:
{}'.format(face.face_attributes.occlusion["foreheadOccluded"]))
         print(' - Eye occluded?: {}'.format(face.face_attributes.occlusion["eyeOccluded"]))
         print(' - Mouth occluded?:
{}'.format(face.face_attributes.occlusion["mouthOccluded"]))
         print(' - Accessories:')
         for accessory in face.face_attributes.accessories:
             print('    - {}'.format(accessory.type))
         # Annotate faces in the image
         annotate_faces(image_file, detected_faces)
```

3. Examine the code you added to the **Main** function. It analyzes an image file and detects any faces it contains, including attributes for head pose, occlusion, and the presence of accessories such as glasses. Additionally, a function is called to annotate the original image with a bounding box for each detected face.

4. Save your changes (*CTRL+S*) but keep the code editor open in case you need to fix any typo's.

5. Resize the panes so you can see more of the console, then enter the following command to run the program with the argument *images/face1.jpg*:

Code Copy

```
python analyze-faces.py images/face1.jpg
```

The app runs and analyzes the following image:

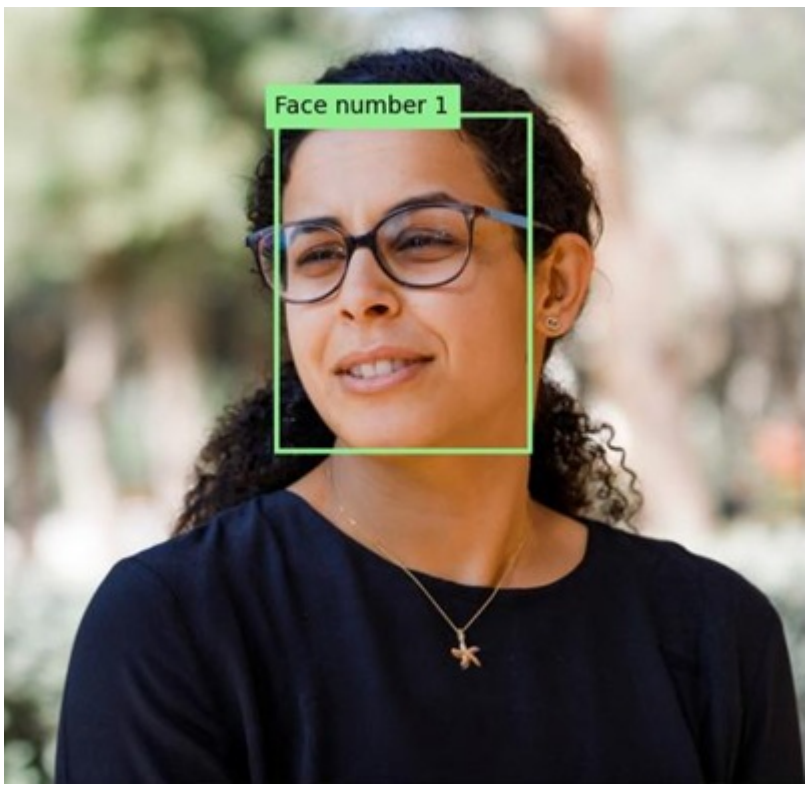
6. Observe the output, which should include the ID and attributes of each face detected.

7. Note that an image file named **detected_faces.jpg** is also generated. Use the (Azure cloud shell-specific) **download** command to download it:

Code Copy

```
download detected_faces.jpg
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file. The image should look simlar to this:



8. Run the program again, this time specifying the parameter *images/face2.jpg* to extract text from the following image:

Code Copy

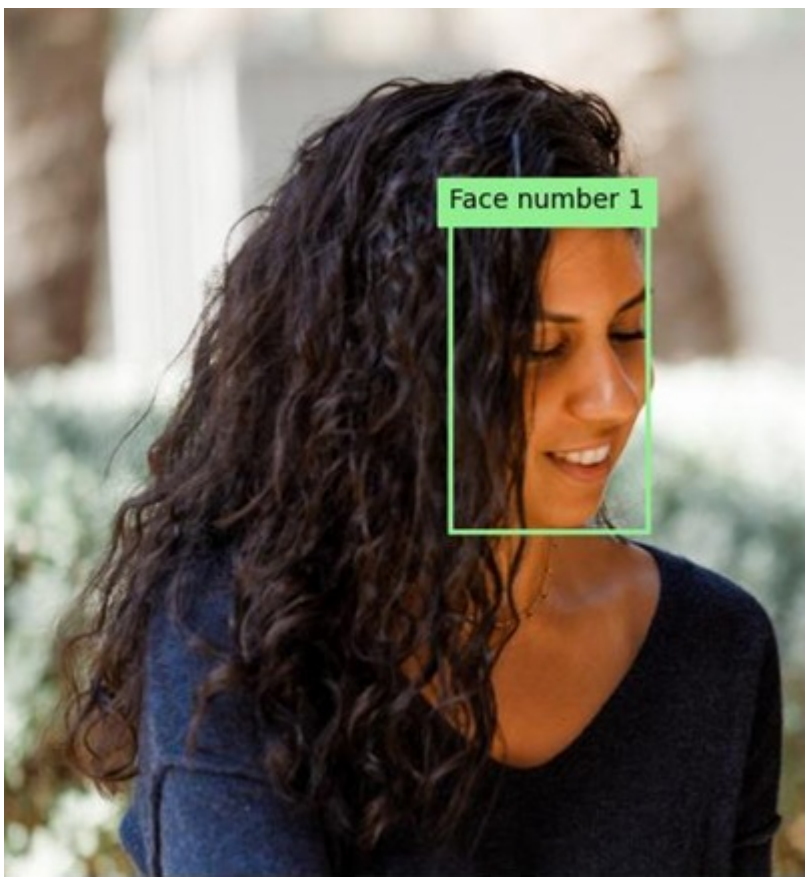
```
python analyze-faces.py images/face2.jpg
```

9. Download and view the resulting **detected_faces.jpg** file:

Code Copy

```
download detected_faces.jpg
```

The resulting image should look like this:



10. Run the program one more time, this time specifying the parameter *images/faces.jpg* to extract text from this image:

Code Copy

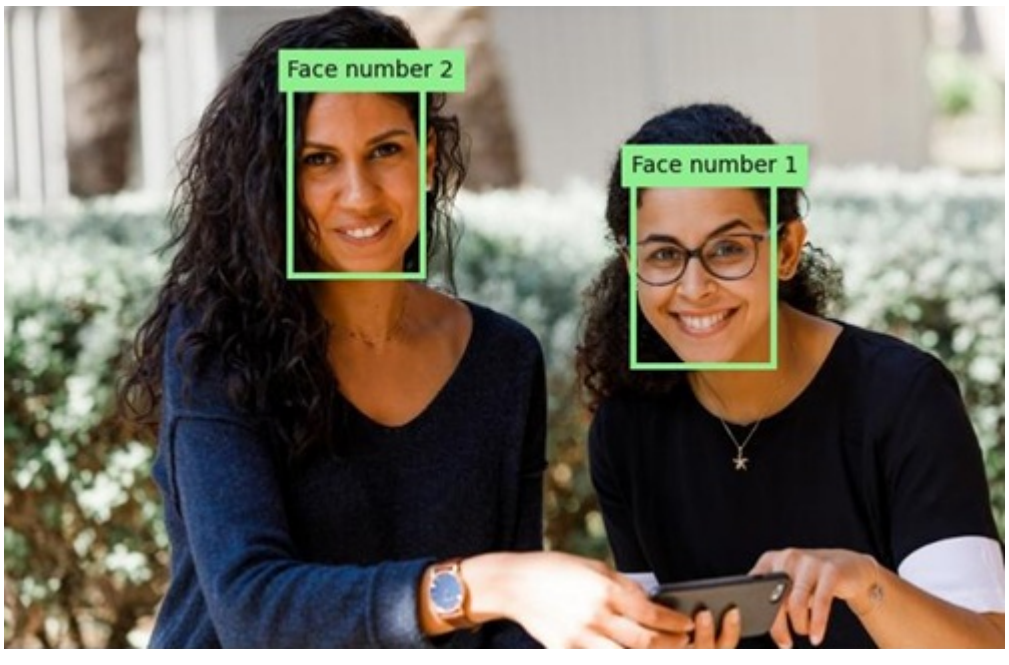
```
python analyze-faces.py images/faces.jpg
```

11. Download and view the resulting **detected_faces.jpg** file:

Code Copy

```
download detected_faces.jpg
```

The resulting image should look like this:



Clean up resources

If you've finished exploring Azure AI Vision, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs:

1. Open the Azure portal at `https://portal.azure.com`, and in the top search bar, search for the resources you created in this lab.
2. On the resource page, select **Delete** and follow the instructions to delete the resource. Alternatively, you can delete the entire resource group to clean up all resources at the same time.

Create Custom Vision resources

Create a Custom Vision project in the Custom Vision portal

Use the training API

Use the image classifier in a client application

Clean up resources

Classify images

The **Azure AI Custom Vision** service enables you to create computer vision models that are trained on your own images. You can use it to train *image classification* and *object detection* models; which you can then publish and consume from applications.

In this exercise, you will use the Custom Vision service to train an image classification model that can identify three classes of fruit (apple, banana, and orange).

While this exercise is based on the Azure Custom Vision Python SDK, you can develop vision applications using multiple language-specific SDKs; including:

- Azure Custom Vision for JavaScript (training)
- Azure Custom Vision for JavaScript (prediction)
- Azure Custom Vision for Microsoft .NET (training)
- Azure Custom Vision for Microsoft .NET (prediction)
- Azure Custom Vision for Java (training)
- Azure Custom Vision for Java (prediction)

This exercise takes approximately **45** minutes.

Create Custom Vision resources

Before you can train a model, you will need Azure resources for *training* and *prediction*. You can create **Custom Vision** resources for each of these tasks, or you can create a single resource and use it for both. In this exercise, you'll create **Custom Vision** resources for training and prediction.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Close any welcome messages or tips that are displayed.
2. Select **Create a resource**.
3. In the search bar, search for `Custom Vision`, select **Custom Vision**, and create the resource with the following settings:
    - **Create options**: Both
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Choose any available region*
    - **Name**: *A valid name for your Custom Vision resource*
    - **Training pricing tier**: F0
    - **Prediction pricing tier**: F0
4. Create the resource and wait for deployment to complete, and then view the deployment details. Note that two Custom Vision resources are provisioned; one for training, and another for prediction.

    > **Note**: Each resource has its own *endpoint* and *keys*, which are used to manage access from your code. To train an image classification model, your code must use the *training* resource (with its endpoint and key); and to use the trained model to predict image classes, your code must use the *prediction* resource (with its endpoint and key).

5. When the resources have been deployed, go to the resource group to view them. You should see two custom vision resources, one with the suffix ***-Prediction***.

Create a Custom Vision project in the Custom Vision portal

To train an image classification model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

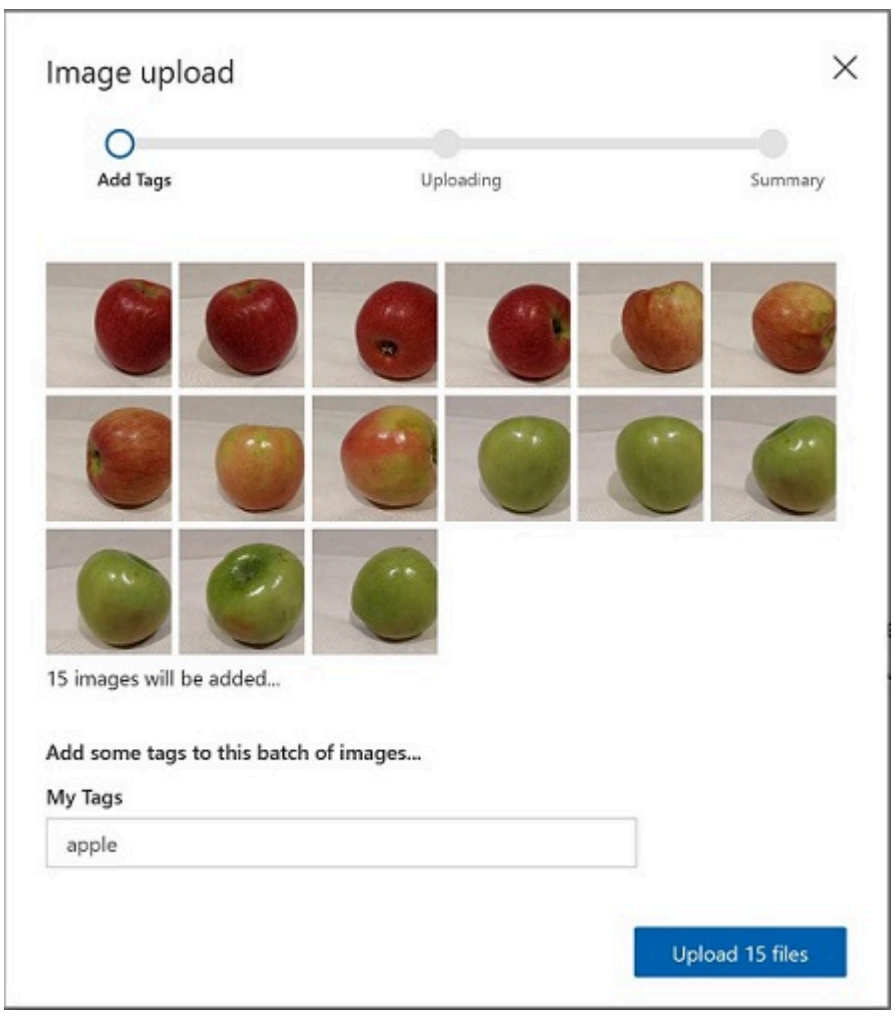
1. Open a new browser tab (keeping the Azure portal tab open - you'll return to it later).

2. In the new browser tab, open the Custom Vision portal at `https://customvision.ai`. If prompted, sign in using your Azure credentials and agree to the terms of service.
3. In the Custom Vision portal, create a new project with the following settings:
    - **Name**: `Classify Fruit`
    - **Description**: `Image classification for fruit`
    - **Resource**: *Your Custom Vision resource*
    - **Project Types**: Classification
    - **Classification Types**: Multiclass (single tag per image)
    - **Domains**: Food

Upload and tag images

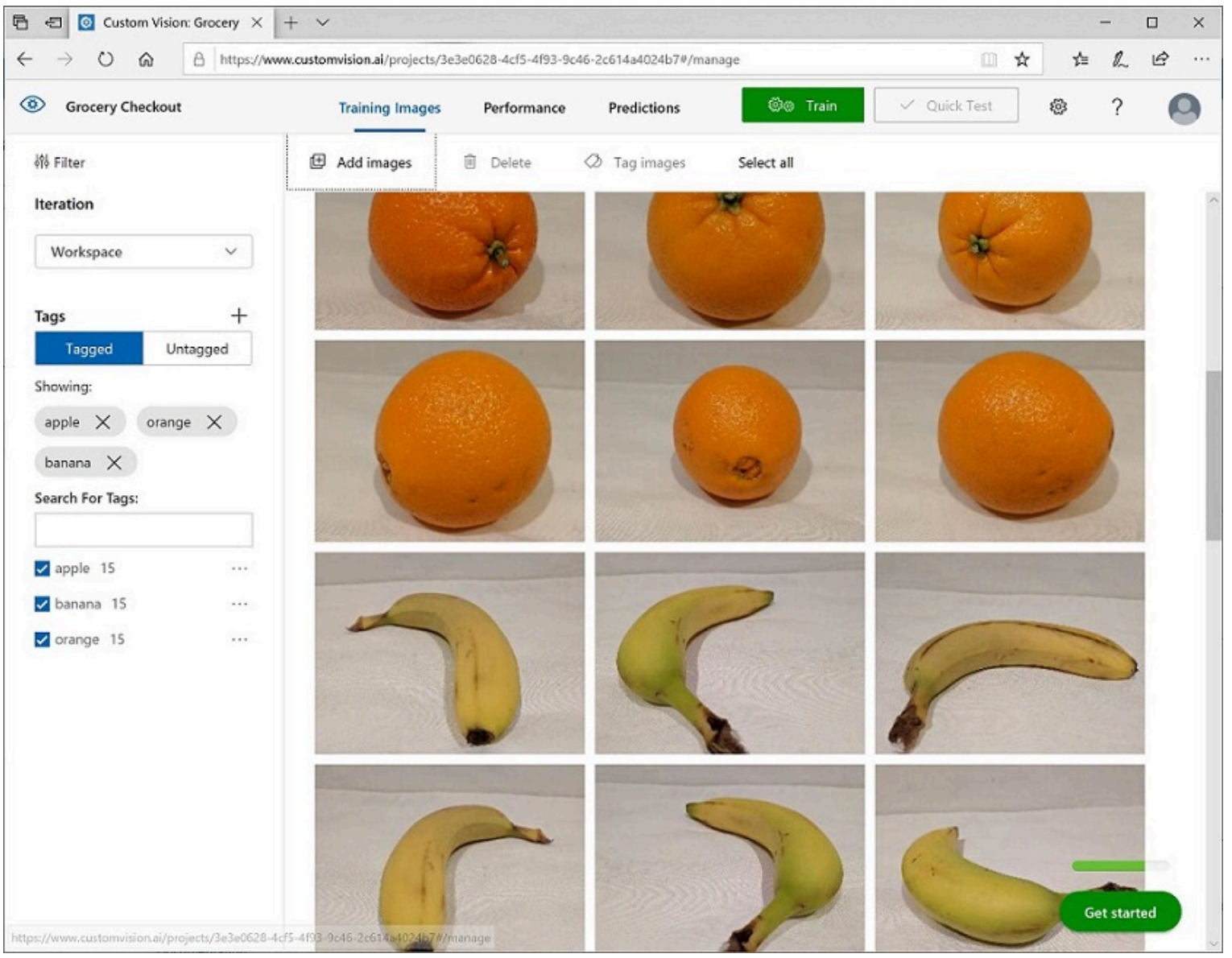
1. In a new browser tab, download the training images from `https://github.com/MicrosoftLearning/mslearn-ai-vision/raw/main/Labfiles/image-classification/training-images.zip` and extract the zip folder to view its contents. This folder contains subfolders of apple, banana, and orange images.

2. In the Custom Vision portal, in your image classification project, click **Add images**, and select all of the files in the **training-images/apple** folder you downloaded and extracted previously. Then upload the image files, specifying the tag `apple`, like this:



3. Use the **Add Images** (**[+]**) toolbar icon to repeat the previous step to upload the images in the **banana** folder with the tag `banana`, and the images in the **orange** folder with the tag `orange`.

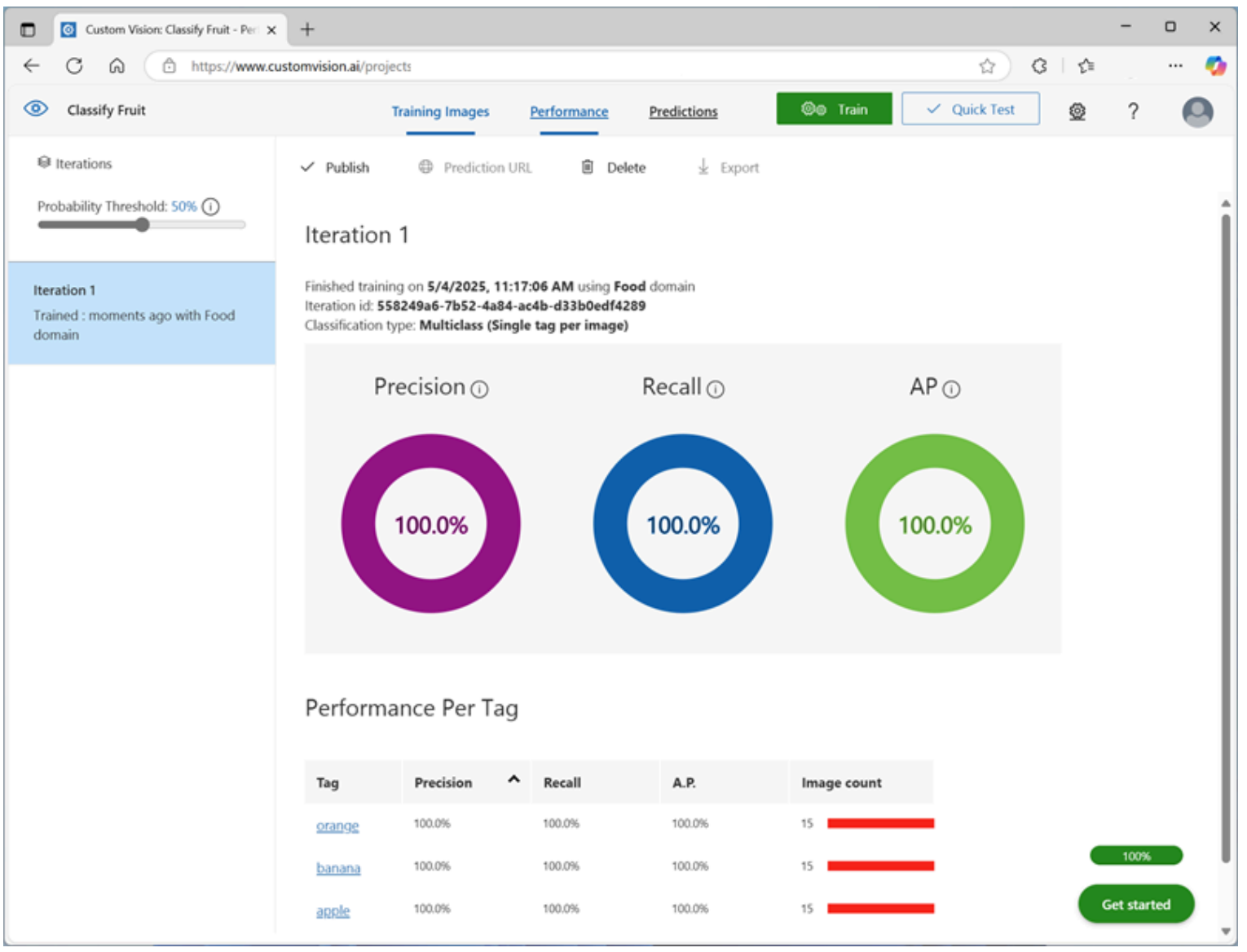
4. Explore the images you have uploaded in the Custom Vision project - there should be 15 images of each class, like this:



Train a model

1. In the Custom Vision project, above the images, click **Train** (⚙) to train a classification model using the tagged images. Select the **Quick Training** option, and then wait for the training iteration to complete (this may take a minute or so).

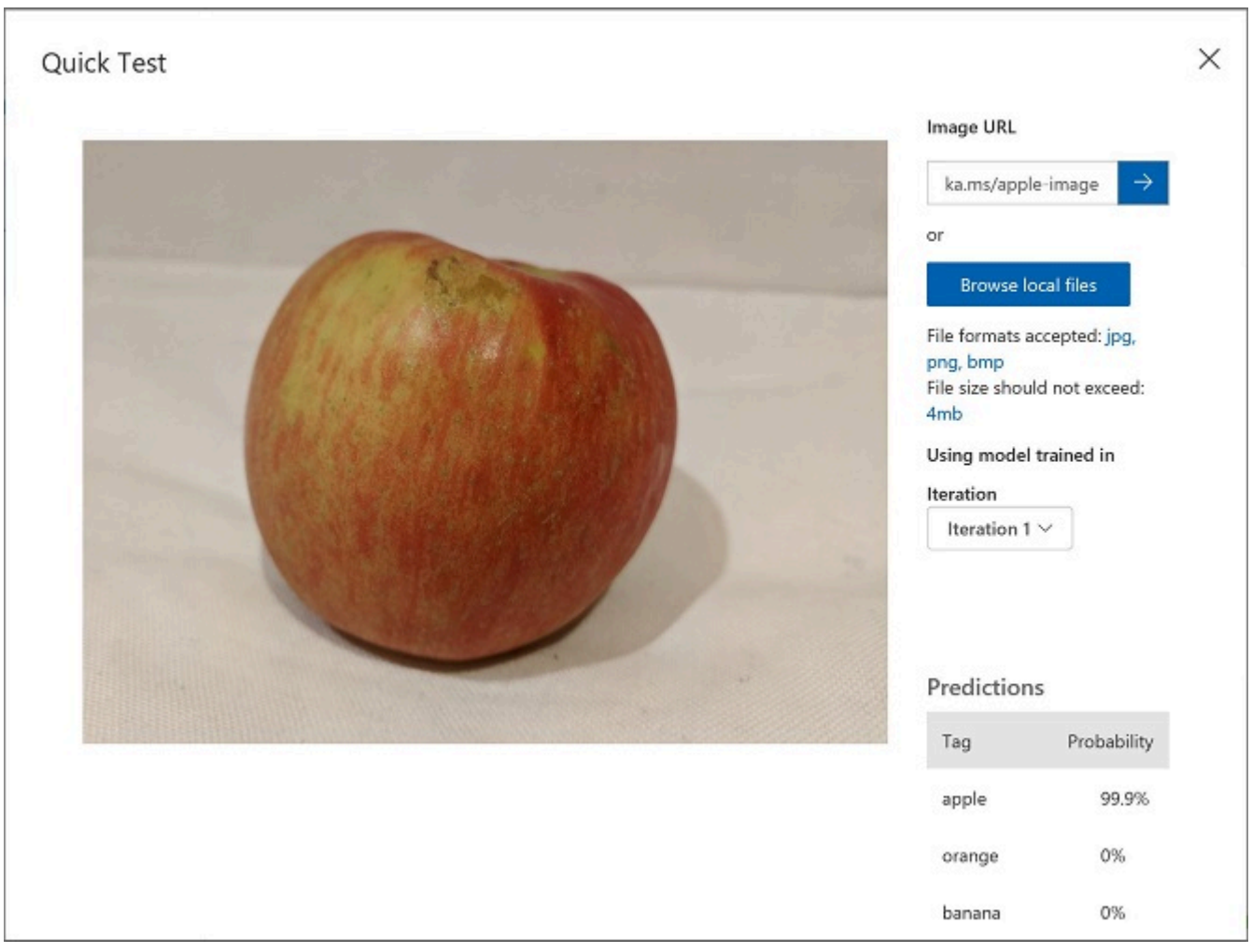
2. When the model iteration has been trained, review the *Precision*, *Recall*, and *AP* performance metrics - these measure the prediction accuracy of the classification model, and should all be high.



> **Note**: The performance metrics are based on a probability threshold of 50% for each prediction (in other words, if the model calculates a 50% or higher probability that an image is of a particular class, then that class is predicted). You can adjust this at the top-left of the page.

Test the model

1. Above the performance metrics, click **Quick Test**.
2. In the **Image URL** box, type `https://aka.ms/test-apple` and click the *quick test image* (➔) button.
3. View the predictions returned by your model - the probability score for *apple* should be the highest, like this:



4. Try testing the following images:
   - `https://aka.ms/test-banana`
   - `https://aka.ms/test-orange`
5. Close the **Quick Test** window.

View the project settings

The project you have created has been assigned a unique identifier, which you will need to specify in any code that interacts with it.

1. Click the *settings* (⚙) icon at the top right of the **Performance** page to view the project settings.
2. Under **General** (on the left), note the **Project Id** that uniquely identifies this project.
3. On the right, under **Resources** note that the key and endpoint are shown. These are the details for the *training* resource (you can also obtain this information by viewing the resource in the Azure portal).

Use the *training* API

The Custom Vision portal provides a convenient user interface that you can use to upload and tag images, and train models. However, in some scenarios you may want to automate model training by using the Custom Vision training API.

Prepare the application configuration

1. Return to the browser tab containing the Azure portal (keeping the Custom Vision portal tab open - you'll return to it later).

2. 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 with no storage in your subscription.

   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***.

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. Resize the cloud shell pane so you can see more of it.

   > **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

5. 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-vision -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
   ```

   > **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.

6. After the repo has been cloned, use the following command to navigate to the application code files:

   Code Copy

   ```
   cd mslearn-ai-vision/Labfiles/image-classification/python/train-classifier
   ls -a -l
   ```

   The folder contains application configuration and code files for your app. It also contains an **/more-training-images** subfolder, which contains some image files you'll use to perform additional training of your model.

7. Install the Azure AI Custom Vision SDK package for training and any other required packages by running the following commands:

   Code Copy

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

8. Enter the following command to edit the configuration file for your app:

   Code Copy

   ```
   code .env
   ```

The file is opened in a code editor.

9. In the code file, update the configuration values it contains to reflect the **Endpoint** and an authentication **Key** for your Custom Vision *training* resource, and the **Project ID** for the custom vision project you created previously.
10. 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.

Write code to perform model training

1. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code train-classifier.py
```

2. Note the following details in the code file:
   - The namespaces for the Azure AI Custom Vision SDK are imported.
   - The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated.
   - **CustomVisionTrainingClient**, which is then used with the project ID to create a **Project** reference to your project.
   - The **Upload_Images** function retrieves the tags that are defined in the Custom Vision project and then uploads image files from correspondingly named folders to the project, assigning the appropriate tag ID.
   - The **Train_Model** function creates a new training iteration for the project and waits for training to complete.
3. Close the code editor (*CTRL+Q*) and enter the following command to run the program:

```
python train-classifier.py
```

4. Wait for the program to end. Then return to the browser tab containing the Custom Vision portal, and view the **Training Images** page for your project (refreshing the browser if necessary).
5. Verify that some new tagged images have been added to the project. Then view the **Performance** page and verify that a new iteration has been created.

Use the image classifier in a client application

Now you're ready to publish your trained model and use it in a client application.

Publish the image classification model

1. In the Custom Vision portal, on the **Performance** page, click ✓ **Publish** to publish the trained model with the following settings:
   - **Model name**: `fruit-classifier`
   - **Prediction Resource**: *The **prediction** resource you created previously which ends with "-Prediction" (not the training resource)*.
2. At the top left of the **Project Settings** page, click the *Projects Gallery* (👁) icon to return to the Custom Vision portal home page, where your project is now listed.
3. On the Custom Vision portal home page, at the top right, click the *settings* (⚙) icon to view the settings for your Custom Vision service. Then, under **Resources**, find your *prediction* resource which ends with "-Prediction" (not the training resource) to determine its **Key** and **Endpoint** values (you can also obtain this information by viewing the resource in the Azure portal).

Use the image classifier from a client application

1. Return to the browser tab containing the Azure portal and the cloud shell pane.
2. In cloud shell, run the following commands to switch to the folder for your client application and view the files it contains:

```
cd ../test-classifier
ls -a -l
```

   The folder contains application configuration and code files for your app. It also contains a **/test-images** subfolder, which contains some image files you'll use to test your model.

3. Install the Azure AI Custom Vision SDK package for prediction and any other required packages by running the following commands:

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

4. Enter the following command to edit the configuration file for your app:

```
code .env
```

   The file is opened in a code editor.
5. Update the configuration values to reflect the **Endpoint** and **Key** for your Custom Vision *prediction* resource, the **Project ID** for the classification project, and the name of your published model (which should be *fruit-classifier*). Save your changes (*CTRL+S*) and close the code editor (*CTRL+Q*).
6. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code test-classifier.py
```

7. Review the code, noting the following details:
   - The namespaces for the Azure AI Custom Vision SDK are imported.
   - The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated **CustomVisionPredictionClient**.
   - The prediction client object is used to predict a class for each image in the **test-images** folder, specifying the project ID and model name for each request. Each prediction includes a probability for each possible class, and only predicted tags with a probability greater than 50% are displayed.
8. Close the code editor and enter the following command to run the program:

```
python test-classifier.py
```

   The program submits each of the following images to the model for classification:

IMG_TEST_1.jpg

IMG_TEST_2.jpg

IMG_TEST_3.jpg

9. View the label (tag) and probability scores for each prediction.

Clean up resources

If you've finished exploring Azure AI Custom Vision, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs:

1. Open the Azure portal at `https://portal.azure.com`, and in the top search bar, search for the resources you created in this lab.

2. On the resource page, select **Delete** and follow the instructions to delete the resource. Alternatively, you can delete the entire resource group to clean up all resources at the same time.

Create Custom Vision resources

Create a Custom Vision project in the Custom Vision portal

Upload and tag images

Train and test a model

Use the object detector in a client application

Use the image classifier from a client application

Clean up resources

More information

Detect objects in images

The **Azure AI Custom Vision** service enables you to create computer vision models that are trained on your own images. You can use it to train *image classification* and *object detection* models; which you can then publish and consume from applications.

In this exercise, you will use the Custom Vision service to train an *object detection* model that can detect and locate three classes of fruit (apple, banana, and orange) in an image.

While this exercise is based on the Azure Custom Vision Python SDK, you can develop vision applications using multiple language-specific SDKs; including:

- Azure Custom Vision for JavaScript (training)
- Azure Custom Vision for JavaScript (prediction)
- Azure Custom Vision for Microsoft .NET (training)
- Azure Custom Vision for Microsoft .NET (prediction)
- Azure Custom Vision for Java (training)
- Azure Custom Vision for Java (prediction)

This exercise takes approximately **45** minutes.

Create Custom Vision resources

Before you can train a model, you will need Azure resources for *training* and *prediction*. You can create **Custom Vision** resources for each of these tasks, or you can create a single resource and use it for both. In this exercise, you'll create **Custom Vision** resources for training and prediction.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Close any welcome messages or tips that are displayed.
2. Select **Create a resource**.
3. In the search bar, search for `Custom Vision`, select **Custom Vision**, and create the resource with the following settings:
    - **Create options**: Both
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Choose any available region*
    - **Name**: *A valid name for your Custom Vision resource*
    - **Training pricing tier**: F0
    - **Prediction pricing tier**: F0
4. Create the resource and wait for deployment to complete, and then view the deployment details. Note that two Custom Vision resources are provisioned; one for training, and another for prediction.

> **Note**: Each resource has its own *endpoint* and *keys*, which are used to manage access from your code. To train an image classification model, your code must use the *training* resource (with its endpoint and key); and to use the trained model to predict image classes, your code must use the *prediction* resource (with its endpoint and key).

5. When the resources have been deployed, go to the resource group to view them. You should see two custom vision resources, one with the suffix ***-Prediction***.

Create a Custom Vision project in the Custom Vision portal

To train an object detection model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

1. Open a new browser tab (keeping the Azure portal tab open - you'll return to it later).

2. In the new browser tab, open the Custom Vision portal at `https://customvision.ai`. If prompted, sign in using your Azure credentials and agree to the terms of service.
3. Create a new project with the following settings:
    - **Name**: `Detect Fruit`
    - **Description**: `Object detection for fruit.`
    - **Resource**: *Your Custom Vision resource*
    - **Project Types**: Object Detection
    - **Domains**: General
4. Wait for the project to be created and opened in the browser.

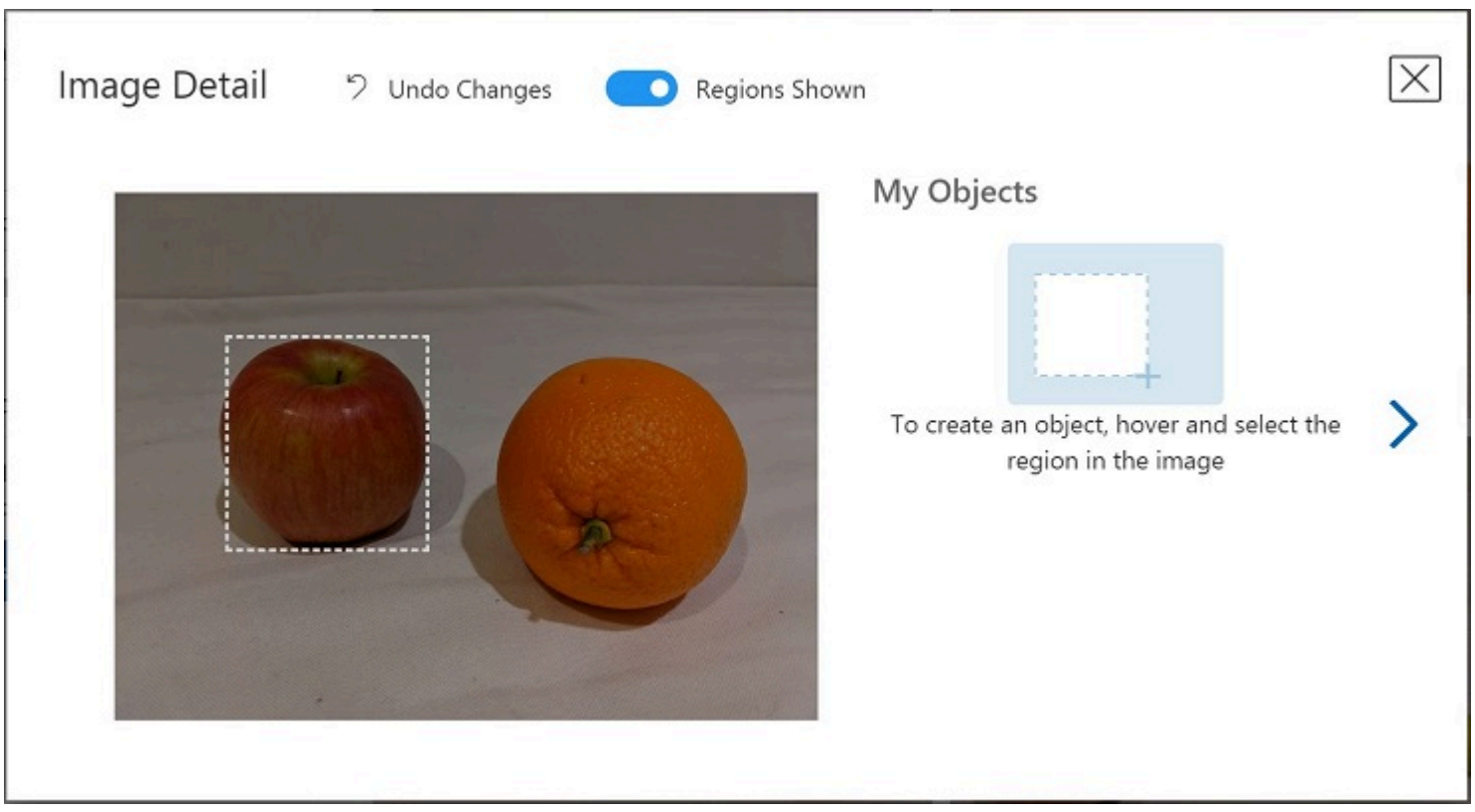
Upload and tag images

Now that you have an object detection project, you can upload and tag images to train a model.

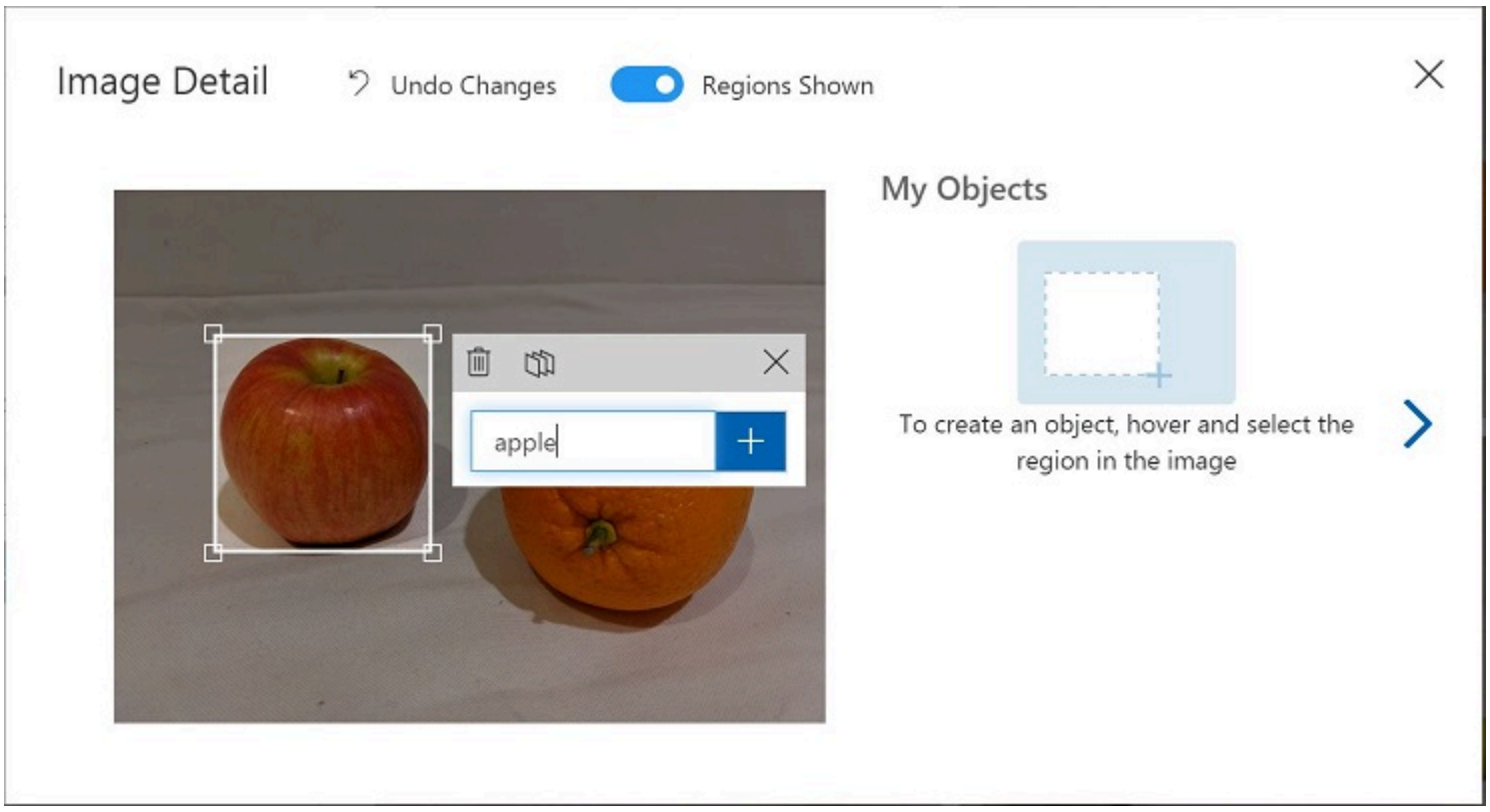
Upload and tag images in the Custom Vision portal

The Custom Vision portal includes visual tools that you can use to upload images and tag regions within them that contain multiple types of object.

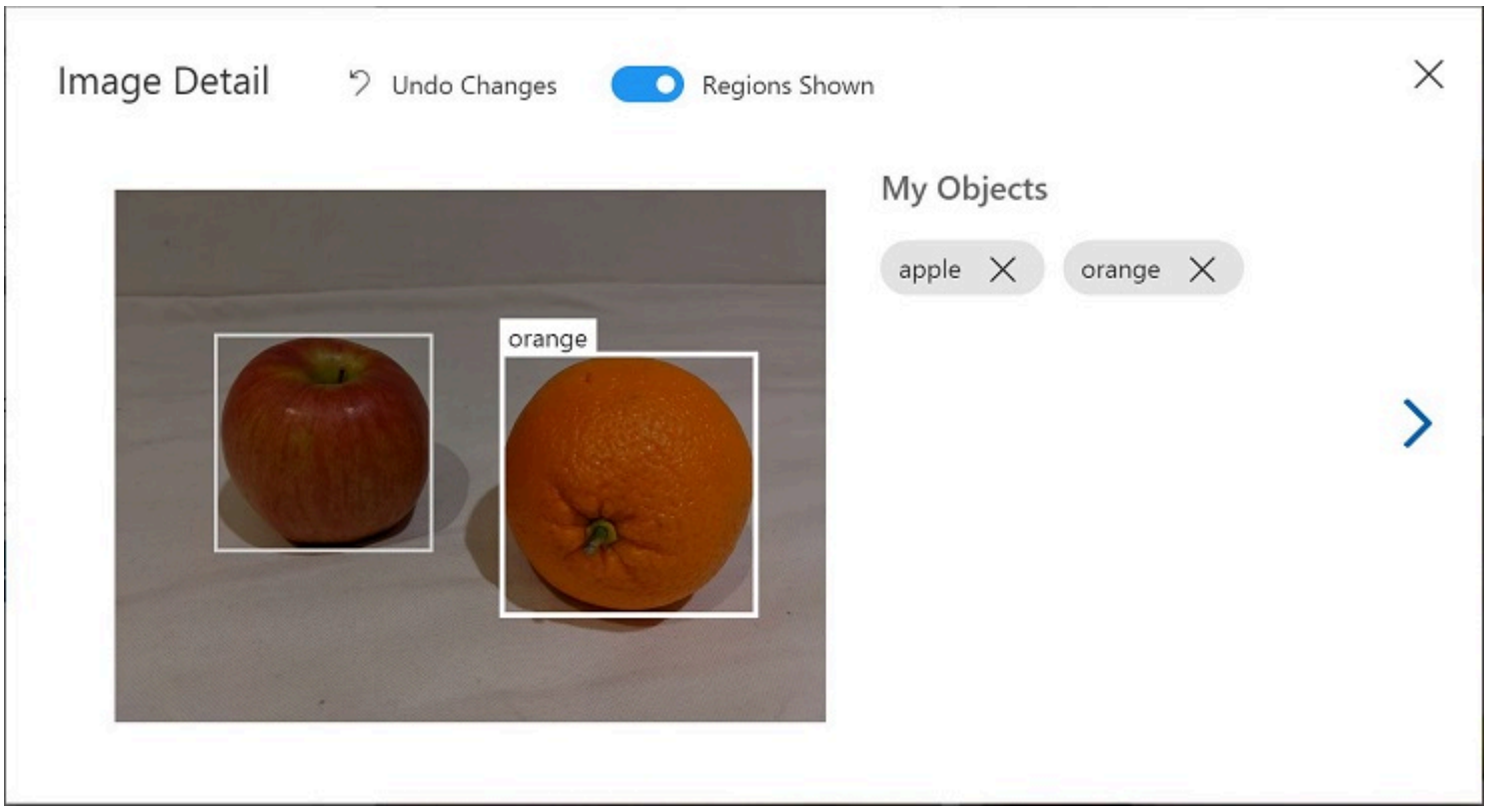
1. In a new browser tab, download the training images from `https://github.com/MicrosoftLearning/mslearn-ai-vision/raw/main/Labfiles/object-detection/training-images.zip` and extract the zip folder to view its contents. This folder contains images of fruit.
2. In the Custom Vision portal, in your object detection project, select **Add images** and upload all of the images in the extracted folder.
3. After the images have been uploaded, select the first one to open it.
4. Hold the mouse over any object in the image until an automatically detected region is displayed like the image below. Then select the object, and if necessary resize the region to surround it.



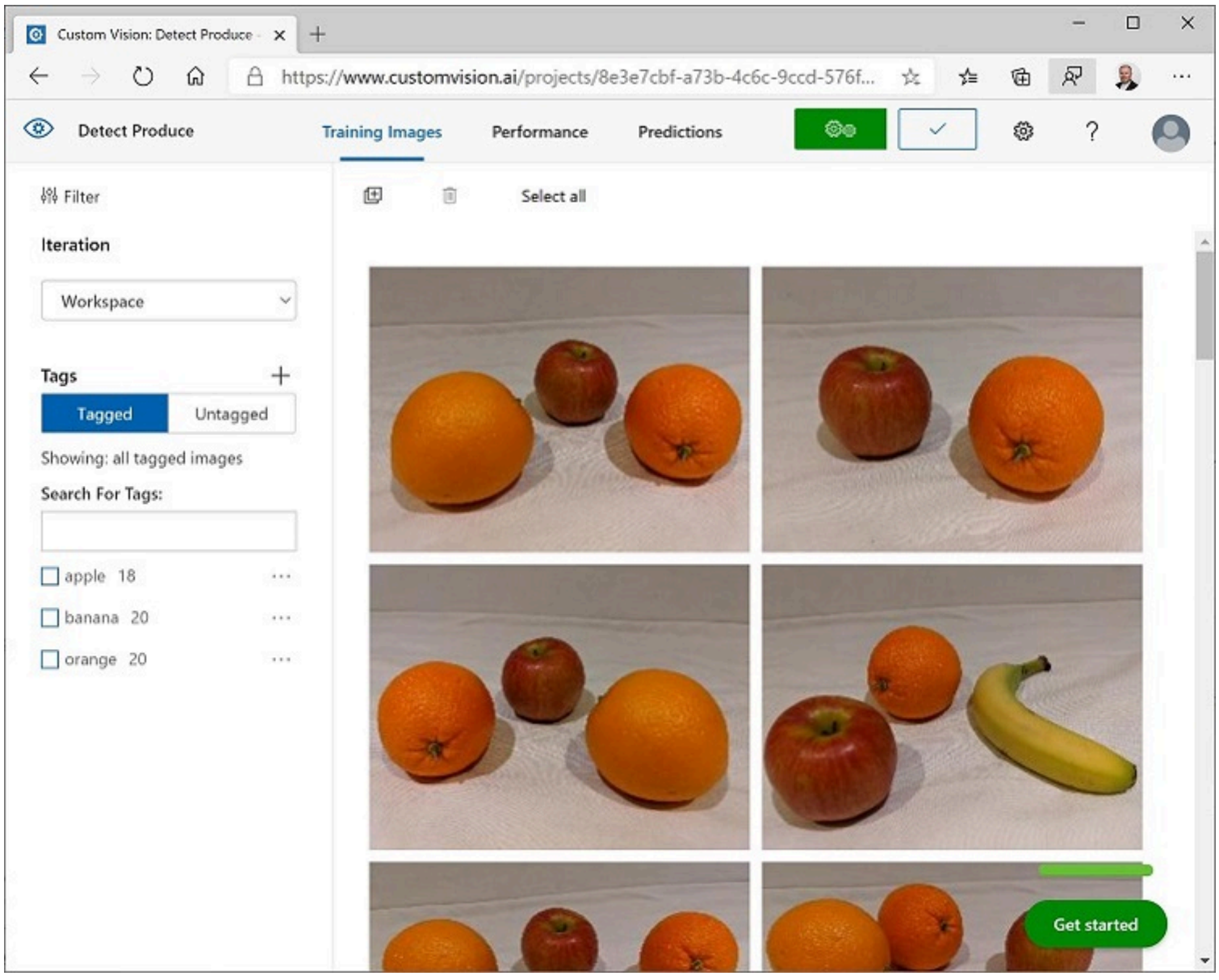
    Alternatively, you can simply drag around the object to create a region.
5. When the region surrounds the object, add a new tag with the appropriate object type (*apple*, *banana*, or *orange*) as shown here:



6. Select and tag each other object in the image, resizing the regions and adding new tags as required.



7. Use the **>** link on the right to go to the next image, and tag its objects. Then just keep working through the entire image collection, tagging each apple, banana, and orange.
8. When you have finished tagging the last image, close the **Image Detail** editor. On the **Training Images** page, under **Tags**, select **Tagged** to see all of your tagged images:



Use the Custom Vision SDK to upload images

You can use the UI in the Custom Vision portal to tag your images, but many AI development teams use other tools that generate files containing information about tags and object regions in images. In scenarios like this, you can use the Custom Vision training API to upload tagged images to the project.

1. Click the *settings* (⚙) icon at the top right of the **Training Images** page in the Custom Vision portal to view the project settings.
2. Under **General** (on the left), note the **Project Id** that uniquely identifies this project.
3. On the right, under **Resources** note that the **Key** and **Endpoint** are shown. These are the details for the *training* resource (you can also obtain this information by viewing the resource in the Azure portal).
4. Return to the browser tab containing the Azure portal (keeping the Custom Vision portal tab open - you'll return to it later).

5. 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 with no storage in your subscription.

   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***.

6. 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.

7. Resize the cloud shell pane so you can see more of it.

   > **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

8. 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-vision -f
git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
```

> **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.

9. After the repo has been cloned, use the following command to navigate to the application code files:

Code Copy

```
cd mslearn-ai-vision/Labfiles/object-detection/python/train-detector
ls -a -l
```

The folder contains application configuration and code files for your app. It also contains a **tagged-images.json** file which contains bounding box coordinates for objects in multiple images, and an **/images** subfolder, which contains the images.

10. Install the Azure AI Custom Vision SDK package for training and any other required packages by running the following commands:

Code Copy

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

11. Enter the following command to edit the configuration file for your app:

Code Copy

```
code .env
```

The file is opened in a code editor.

12. In the code file, update the configuration values it contains to reflect the **Endpoint** and an authentication **Key** for your Custom Vision *training* resource, and the **Project ID** for the custom vision project you created previously.
13. 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.

14. In the cloud shell command line, enter the following command to open the **tagged-images.json** file to see the tagging information for the image files in the **/images** subfolder:

Code Copy

```
code tagged-images.json
```

JSON defines a list of images, each containing one or more tagged regions. Each tagged region includes a tag name, and the top and left coordinates and width and height dimensions of the bounding box containing the tagged object.

> **Note**: The coordinates and dimensions in this file indicate relative points on the image. For example, a *height* value of 0.7 indicates a box that is 70% of the height of the image. Some tagging tools generate other formats of file in which the coordinate and dimension values represent pixels, inches, or other units of measurements.

15. Close the JSON file without saving any changes (*CTRL_Q*).
16. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code add-tagged-images.py
```

17. Note the following details in the code file:
    - The namespaces for the Azure AI Custom Vision SDK are imported.
    - The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated **CustomVisionTrainingClient**, which is then used with the project ID to create a **Project** reference to your project.
    - The **Upload_Images** function extracts the tagged region information from the JSON file and uses it to create a batch of images with regions, which it then uploads to the project.
18. Close the code editor (*CTRL+Q*) and enter the following command to run the program:

```
python add-tagged-images.py
```

19. Wait for the program to end.
20. Switch back to the browser tab containing the Custom Vision portal (keeping the Azure portal cloud shell tab open), and view the **Training Images** page for your project (refreshing the browser if necessary).
21. Verify that some new tagged images have been added to the project.

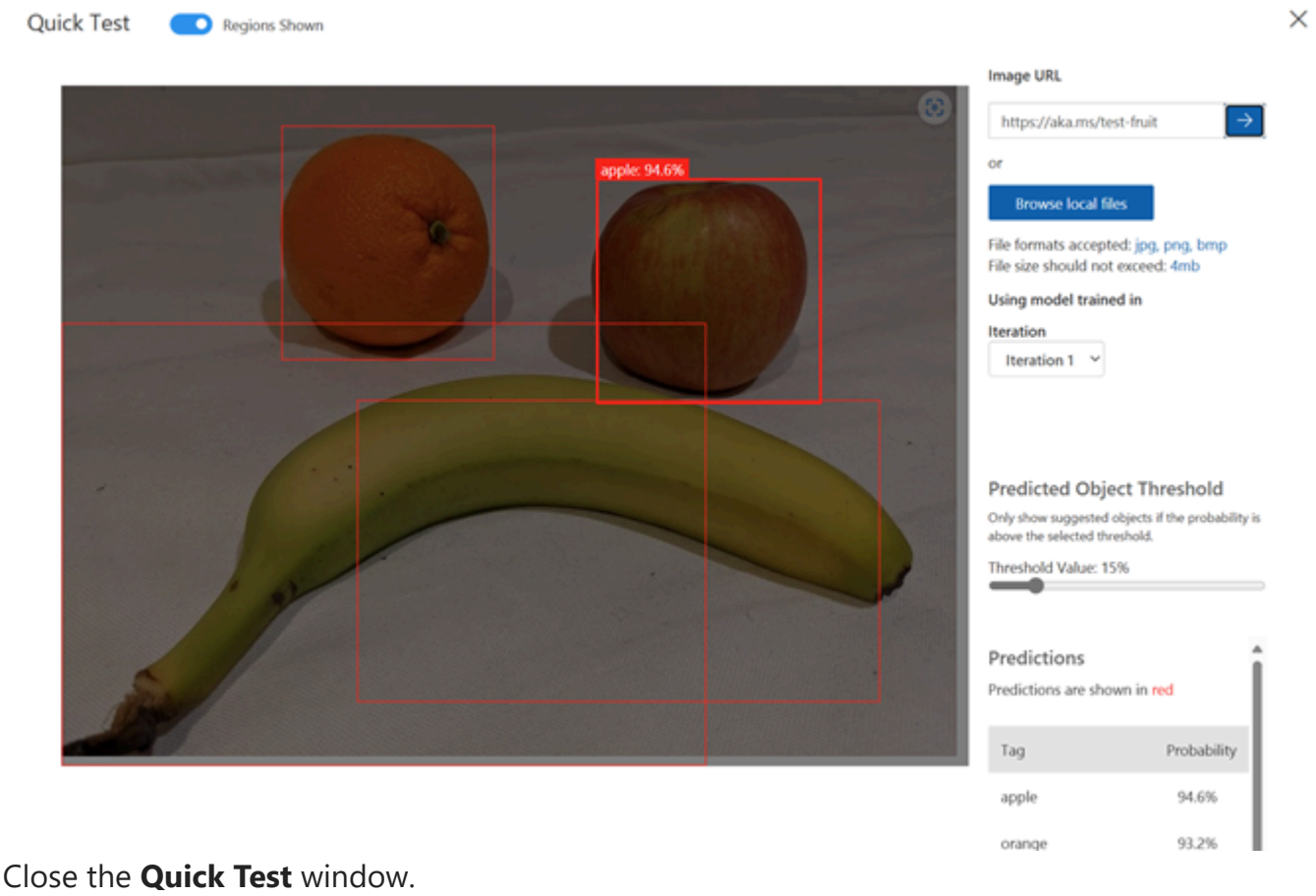
Train and test a model

Now that you've tagged the images in your project, you're ready to train a model.

1. In the Custom Vision project, click **Train** (⚙) to train an object detection model using the tagged images. Select the **Quick Training** option.
2. Wait for training to complete (it might take ten minutes or so).

> **Tip**: The Azure cloud shell has a 20-minute inactivity timeout, after which the session is abandoned. While you wait for training to finish, occassionally return to the cloud shell and enter a colland like `ls` to keep the session active.

3. In the Custom Vision portal, when training has finished, review the *Precision*, *Recall*, and *mAP* performance metrics - these measure the prediction accuracy of the object detection model, and should all be high.
4. At the top right of the page, click **Quick Test**, and then in the **Image URL** box, type `https://aka.ms/test-fruit` and click the *quick test image* (➔) button.
5. View the prediction that is generated.



6. Close the **Quick Test** window.

Use the object detector in a client application

Now you're ready to publish your trained model and use it in a client application.

Publish the object detection model

1. In the Custom Vision portal, on the **Performance** page, click ✓ **Publish** to publish the trained model with the following settings:
    - **Model name**: `fruit-detector`
    - **Prediction Resource**: *The **prediction** resource you created previously which ends with "-Prediction" (not the training resource)*.
2. At the top left of the **Project Settings** page, click the *Projects Gallery* (👁) icon to return to the Custom Vision portal home page, where your project is now listed.
3. On the Custom Vision portal home page, at the top right, click the *settings* (⚙) icon to view the settings for your Custom Vision service. Then, under **Resources**, find your *prediction* resource which ends with "-Prediction" (not the training resource) to determine its **Key** and **Endpoint** values (you can also obtain this information by viewing the resource in the Azure portal).

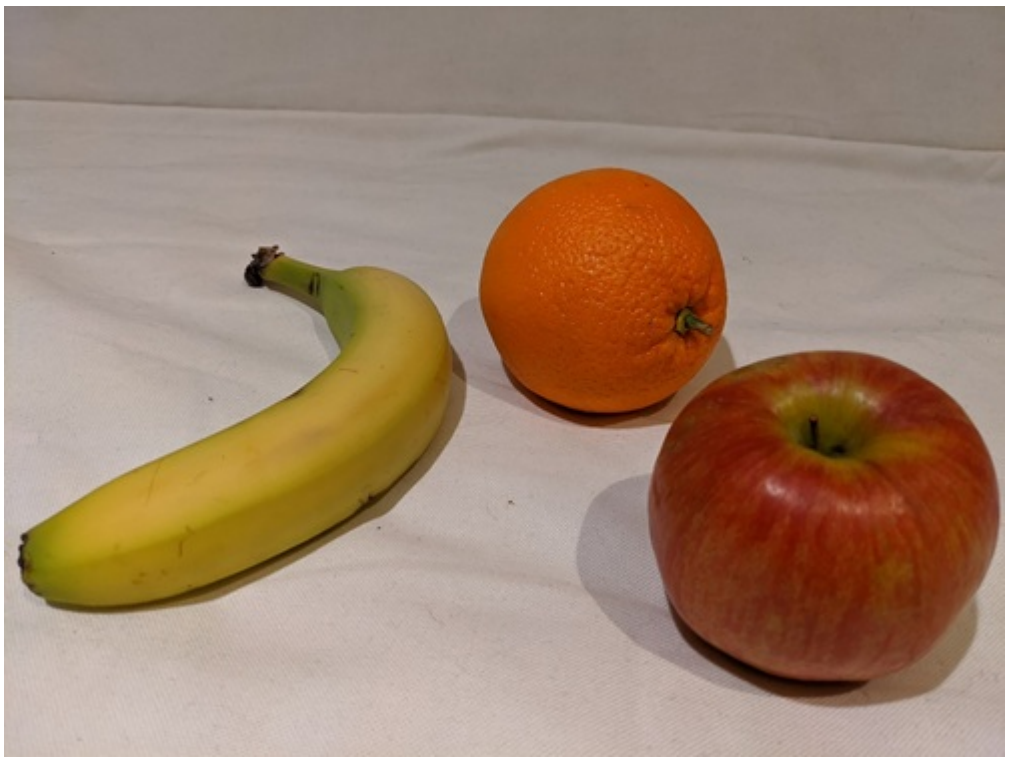
Use the image classifier from a client application

Now that you've published the image classification model, you can use it from a client application. Once again, you can choose to use **C#** or **Python**.

1. Return to the browser tab containing the Azure portal and the cloud shell pane.
2. In cloud shell, run the following commands to switch to the folder for you client application and view the files it contains:

```
cd ../test-detector
ls -a -l
```

The folder contains application configuration and code files for your app. It also contains the following **produce.jpg** image file, which you'll use to test your model.



3. Install the Azure AI Custom Vision SDK package for prediction and any other required packages by running the following commands:

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

4. Enter the following command to edit the configuration file for your app:

```
code .env
```

The file is opened in a code editor.

5. Update the configuration values to reflect the **Endpoint** and **Key** for your Custom Vision *prediction* resource, the **Project ID** for the object detection project, and the name of your published model (which should be *fruit-detector*). Save your changes (*CTRL*+*S*) and close the code editor (*CTRL*+*Q*).

6. In the cloud shell command line, enter the following command to open the code file for the client application:

```
code test-detector.py
```

7. Review the code, noting the following details:

   - The namespaces for the Azure AI Custom Vision SDK are imported.
   - The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated **CustomVisionPredictionClient**.
   - The prediction client object is used to get object detection predictions for the **produce.jpg** image, specifying the project ID and model name in the request. The predicted tagged regions are then drawn on the image, and the result is saved as **output.jpg**.

8. Close the code editor and enter the following command to run the program:

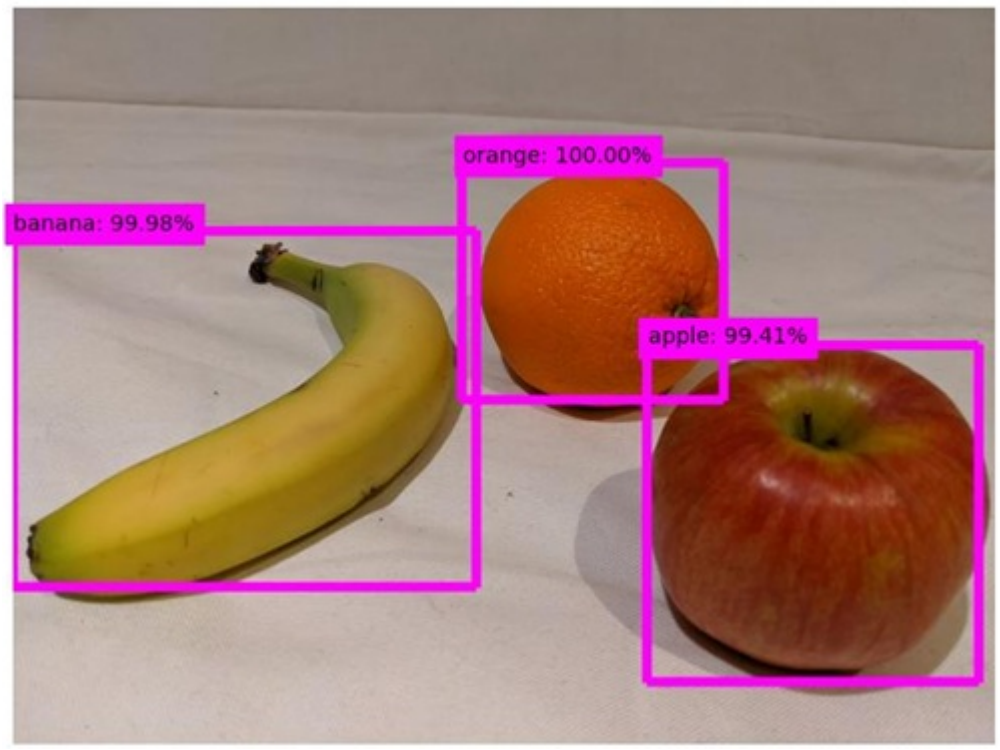
```
python test-detector.py
```

9. Review the program output, which lists each object detected in the image.

10. Note that an image file named **output.jpg** is generated. Use the (Azure cloud shell-specific) **download** command to download it:

```
download output.jpg
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file. The image should look simlar to this:



Clean up resources

If you're not using the Azure resources created in this lab for other training modules, you can delete them to avoid incurring further charges.

1. Open the Azure portal at `https://portal.azure.com`, and in the top search bar, search for the resources you created in this lab.
2. On the resource page, select **Delete** and follow the instructions to delete the resource. Alternatively, you can delete the entire resource group to clean up all resources at the same time.

More information

For more information about object detection with the Custom Vision service, see the Custom Vision documentation.

Upload a video to Video Indexer

Review video insights

Search for insights

Use the Video Indexer REST API

Use Video Indexer widgets

Analyze video

A large proportion of the data created and consumed today is in the format of video. **Azure AI Video Indexer** is an AI-powered service that you can use to index videos and extract insights from them.

> **Note**: From June 21st 2022, capabilities of Azure AI services that return personally identifiable information are restricted to customers who have been granted limited access. Without getting limited access approval, recognizing people and celebrities with Video Indexer for this lab is not available. For more details about the changes Microsoft has made, and why - see Responsible AI investments and safeguards for facial recognition.

Upload a video to Video Indexer

First, you'll need to sign into the Video Indexer portal and upload a video.

1. In your browser, open the Video Indexer portal at `https://www.videoindexer.ai`.
2. If you have an existing Video Indexer account, sign in. Otherwise, sign up for a free account and sign in using your Microsoft account (or any other valid account type). If you have difficulty signing in, try opening a private browser session.

   > Note: If this is your first time signing in you might see a pop-up form asking you to verify how you're going to use the service.

3. In a new tab, download the Responsible AI video by visiting `https://aka.ms/responsible-ai-video`. Save the file.
4. In Video Indexer, select the **Upload** option. Then select the option to **Browse for files**, select the downloaded video, and click **Add**. Change the text in the **File name** field to **Responsible AI**. Select **Review + upload**, review the summary overview, select the checkbox to verify compliance with Microsoft's policies for facial recognition, and upload the file.
5. After the file has uploaded, wait a few minutes while Video Indexer automatically indexes it.

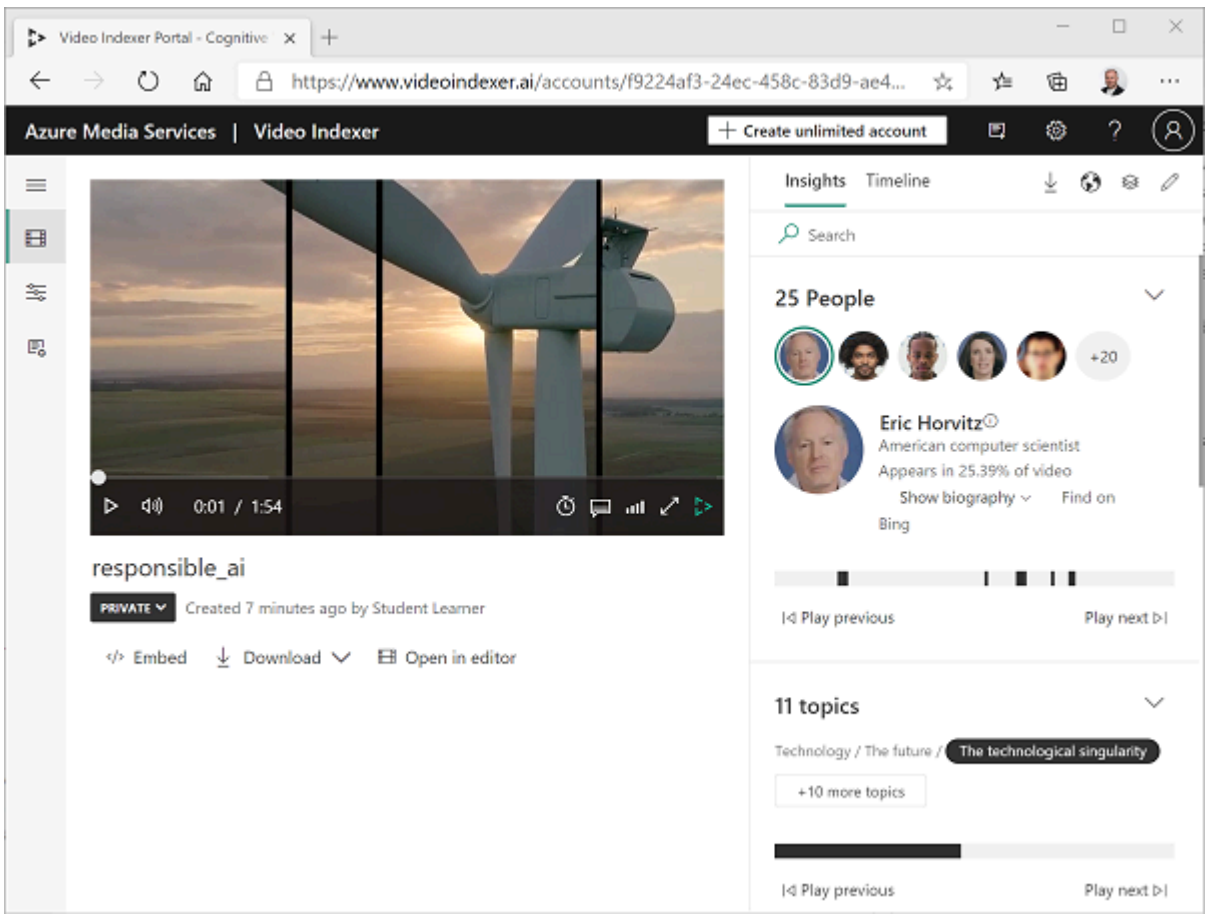
> **Note**: In this exercise, we're using this video to explore Video Indexer functionality; but you should take the time to watch it in full when you've finished the exercise as it contains useful information and guidance for developing AI-enabled applications responsibly!

Review video insights

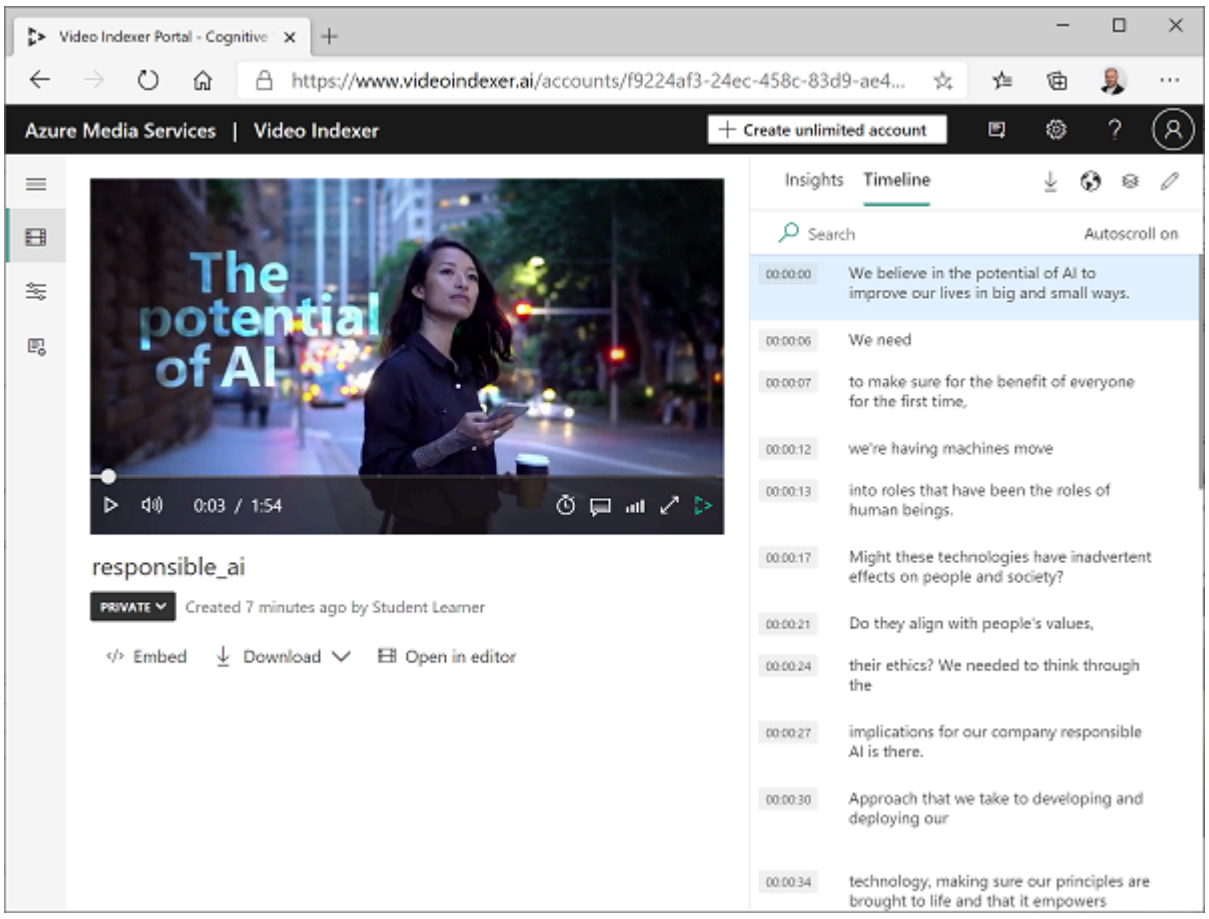
The indexing process extracts insights from the video, which you can view in the portal.

1. In the Video Indexer portal, when the video is indexed, select it to view it. You'll see the video player alongside a pane that shows insights extracted from the video.

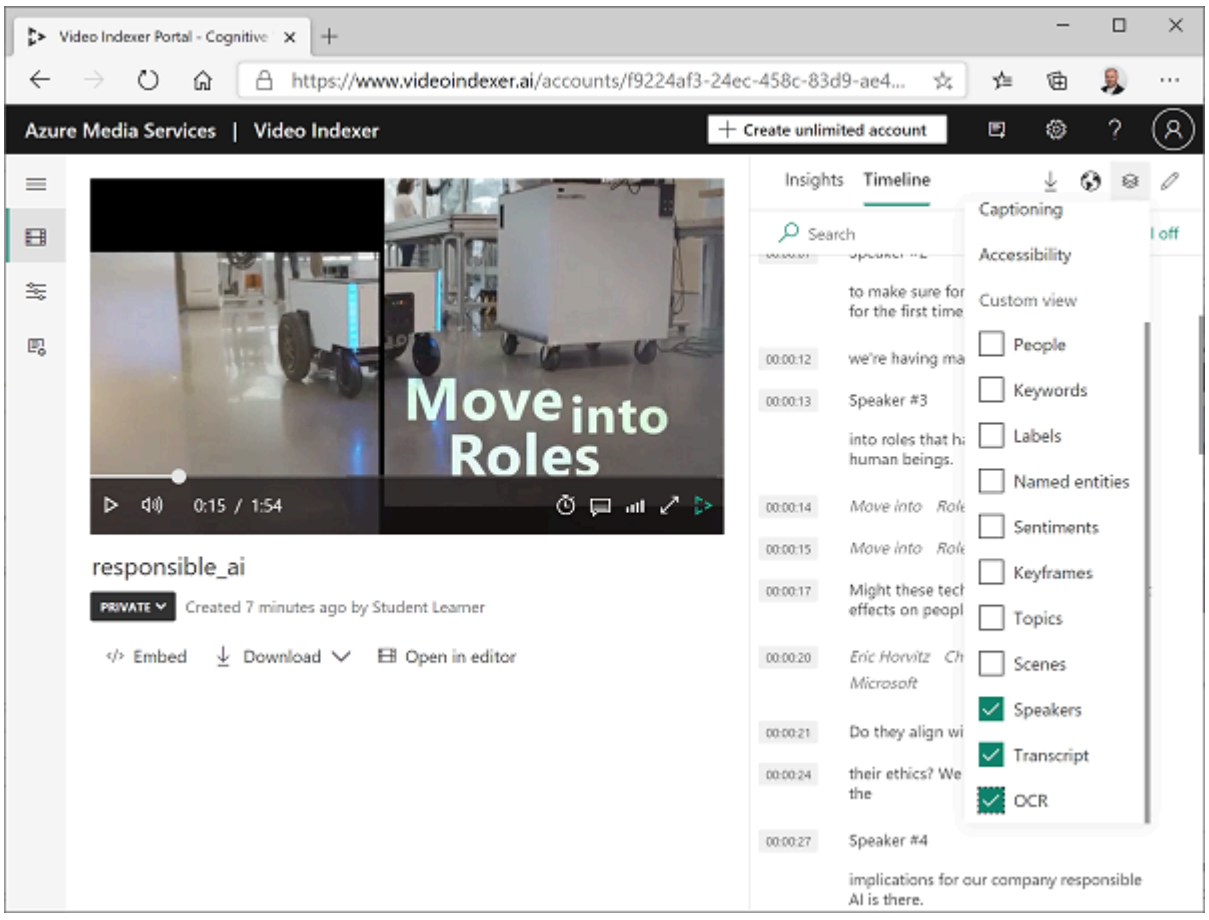
   > **Note**: Due to the limited access policy to protect individuals identities, you may not see names when you index the video.



2. As the video plays, select the **Timeline** tab to view a transcript of the video audio.



3. At the top right of the portal, select the **View** symbol (which looks similar to), and in the list of insights, in addition to **Transcript**, select **OCR** and **Speakers**.



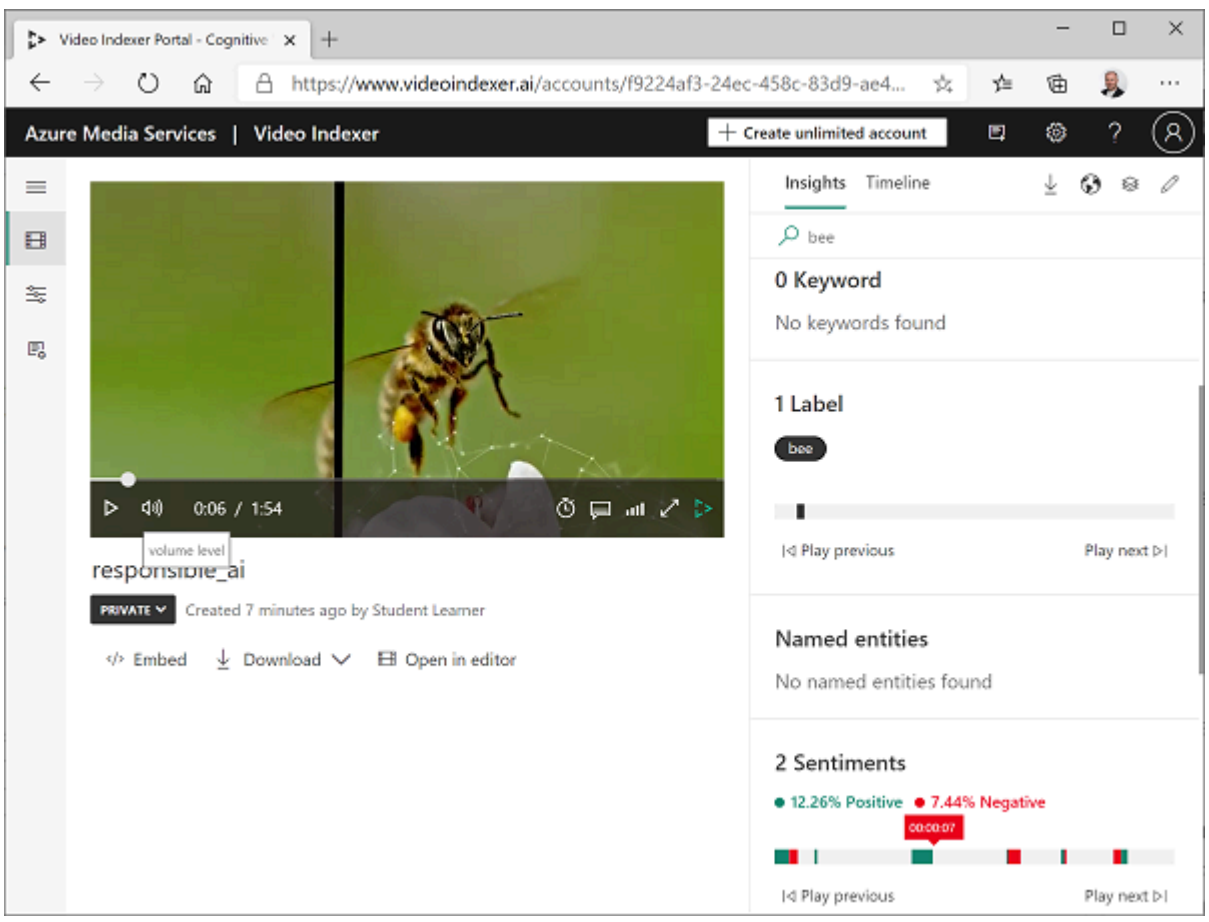
4. Observe that the **Timeline** pane now includes:
    - Transcript of audio narration.
    - Text visible in the video.
    - Indications of speakers who appear in the video. Some well-known people are automatically recognized by name, others are indicated by number (for example *Speaker #1*).
5. Switch back to the **Insights** pane and view the insights show there. They include:
    - Individual people who appear in the video.
    - Topics discussed in the video.
    - Labels for objects that appear in the video.
    - Named entities, such as people and brands that appear in the video.
    - Key scenes.
6. With the **Insights** pane visible, select the **View** symbol again, and in the list of insights, add **Keywords** and **Sentiments** to the pane.

    The insights found can help you determine the main themes in the video. For example, the **topics** for this video show that it is clearly about technology, social responsibility, and ethics.

Search for insights

You can use Video Indexer to search the video for insights.

1. In the **Insights** pane, in the **Search** box, enter *Bee*. You may need to scroll down in the Insights pane to see results for all types of insight.
2. Observe that one matching *label* is found, with its location in the video indicated beneath.
3. Select the beginning of the section where the presence of a bee is indicated, and view the video at that point (you may need to pause the video and select carefully - the bee only appears briefly!)



4. Clear the **Search** box to show all insights for the video.

Use the Video Indexer REST API

Video Indexer provides a REST API that you can use to upload and manage videos in your account.

1. In a new browser tab, open the Azure portal at `https://portal.azure.com`, and sign in using your Azure credentials. Keep the existing tab with the Video Indexer portal open.
2. 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 with no storage in your subscription.

    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***.

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. Resize the cloud shell pane so you can see more of it.

> **Tip**" You can resize the pane by dragging the top border. You can also use the minimize and maximize buttons to switch between the cloud shell and the main portal interface.

5. 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-vision -f
git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
```

> **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.

6. After the repo has been cloned, navigate to the folder containing the application code file for this exercise:

```
Code                                                                    Copy

cd mslearn-ai-vision/Labfiles/video-indexer
```

Get your API details

To use the Video Indexer API, you need some information to authenticate requests:

1. In the Video Indexer portal, expand the left pane and select the **Account settings** page.
2. Note the **Account ID** on this page - you will need it later.
3. Open a new browser tab and go to the Video Indexer developer portal at `https://api-portal.videoindexer.ai, signing with your Azure credentials.
4. On the **Profile** page, view the **Subscriptions** associated with your profile.
5. On the page with your subscription(s), observe that you have been assigned two keys (primary and secondary) for each subscription. Then select **Show** for any of the keys to see it. You will need this key shortly.

Use the REST API

Now that you have the account ID and an API key, you can use the REST API to work with videos in your account. In this procedure, you'll use a PowerShell script to make REST calls; but the same principles apply with HTTP utilities such as cURL or Postman, or any programming language capable of sending and receiving JSON over HTTP.

All interactions with the Video Indexer REST API follow the same pattern:

- An initial request to the **AccessToken** method with the API key in the header is used to obtain an access token.
- Subsequent requests use the access token to authenticate when calling REST methods to work with videos.

1. In the cloud shell, use the following command to open the PowerShell script:

Code Copy

```
code get-videos.ps1
```

2. In the PowerShell script, replace the **YOUR_ACCOUNT_ID** and **YOUR_API_KEY** placeholders with the account ID and API key values you identified previously.
3. Observe that the *location* for a free account is "trial". If you have created an unrestricted Video Indexer account (with an associated Azure resource), you can change this to the location where your Azure resource is provisioned (for example "eastus").
4. Review the code in the script, noting that invokes two REST methods: one to get an access token, and another to list the videos in your account.

5. Save your changes (press *CTRL*+*S*), close the code editor (press *CTRL*+*Q*) and then run the following command to execute the script:

Code Copy

```
./get-videos.ps1
```

6. View the JSON response from the REST service, which should contain details of the **Responsible AI** video you indexed previously.

Use Video Indexer widgets

The Video Indexer portal is a useful interface to manage video indexing projects. However, there may be occasions when you want to make the video and its insights available to people who don't have access to your Video Indexer account. Video Indexer provides widgets that you can embed in a web page for this purpose.

1. Use the `ls` command,to view the contents of the **video-indexer** folder. Note that it contains a **analyze-video.html** file. This is a basic HTML page to which you will add the Video Indexer **Player** and **Insights** widgets.

2. Enter the following command to edit the file:

Code Copy

```
code analyze-video.html
```

The file is opened in a code editor.

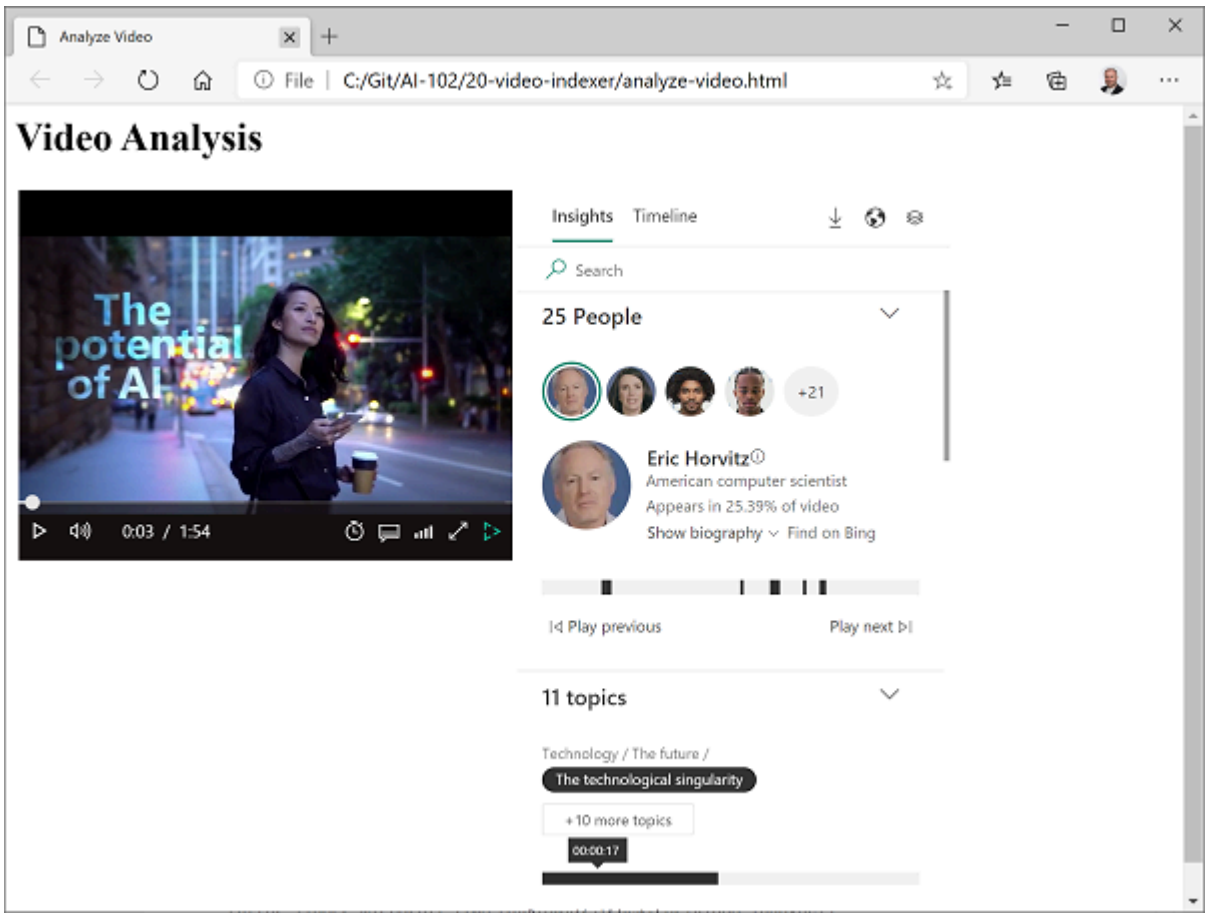
3. Note the reference to the **vb.widgets.mediator.js** script in the header - this script enables multiple Video Indexer widgets on the page to interact with one another.
4. In the Video Indexer portal, return to the **Media files** page and open your **Responsible AI** video.
5. Under the video player, select **</> Embed** to view the HTML iframe code to embed the widgets.
6. In the **Share and Embed** dialog box, select the **Player** widget, set the video size to 560 x 315, and then copy the embed code to the clipboard.
7. In the Azure portal cloud shell, in the code editor for the **analyze-video.html** file, paste the copied code under the comment **<– Player widget goes here – >**.
8. Back in the Video Indexer portal, in the **Share and Embed** dialog box, select the **Insights** widget and then copy the embed code to the clipboard. Then close the **Share and Embed** dialog box, switch back to Azure portal, and paste the copied code under the comment **<– Insights widget goes here – >**.
9. After editing the file, within the code editor, save your changes (*CTRL*+*S*) and then close the code editor (*CTRL*+*Q*) while keeping the cloud shell command line open.

10. In the cloud shell toolbar, enter the following (Cloud shell-specific) command to download the HTML file you edited:

Code Copy

```
download analyze-video.html
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file; which should look like this:



11. Experiment with the widgets, using the **Insights** widget to search for insights and jump to them in the video.

Open Azure AI Foundry portal

Choose a model to start a project

Test the model in the playground

Create a client application

Sign into Azure and run the app

Clean up

Develop a vision-enabled chat app

In this exercise, you use the *Phi-4-multimodal-instruct* generative AI model to generate responses to prompts that include images. You'll develop an app that provides AI assistance with fresh produce in a grocery store by using Azure AI Foundry and the Azure AI Model Inference service.

> **Note**: This exercise is based on pre-release SDK software, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

While this exercise is based on the Azure AI Foundry Python SDK, you can develop AI chat applications using multiple language-specific SDKs; including:

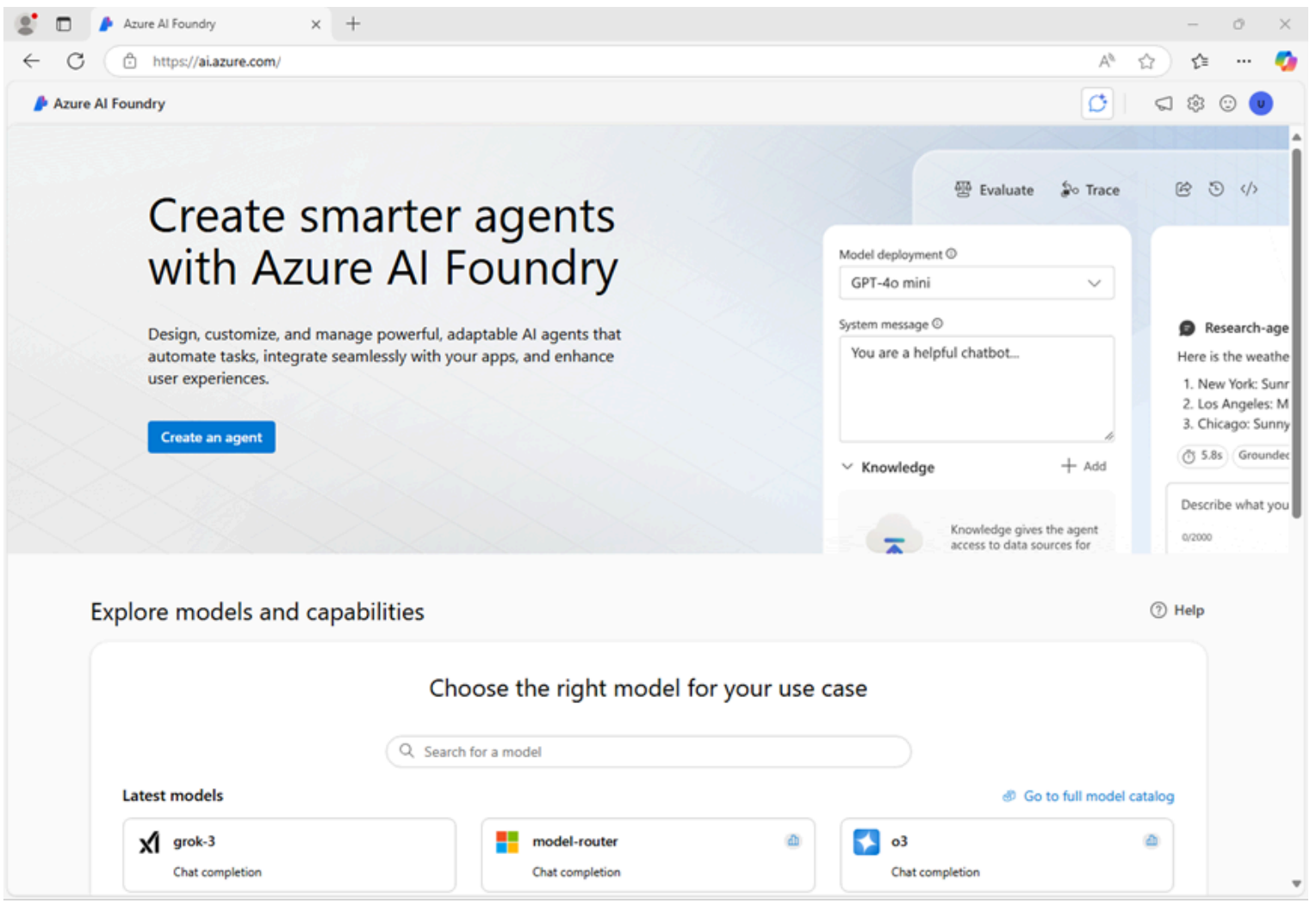
- Azure AI Projects for Python
- Azure AI Projects for Microsoft .NET
- Azure AI Projects for JavaScript

This exercise takes approximately **30** minutes.

Open Azure AI Foundry portal

Let's start by signing into Azure AI Foundry portal.

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. Review the information on the home page.

Choose a model to start a project

An Azure AI *project* provides a collaborative workspace for AI development. Let's start by choosing a model that we want to work with and creating a project to use it in.

> **Note**: AI Foundry projects can be based on an *Azure AI Foundry* resource, which provides access to AI models (including Azure OpenAI), Azure AI services, and other resources for developing AI agents and chat solutions. Alternatively, projects can be based on *AI hub* resources; which include connections to Azure resources for secure storage, compute, and specialized tools. Azure AI Foundry based projects are great for developers who want to manage resources for AI agent or chat app development. AI hub based projects are more suitable for enterprise development teams working on complex AI solutions.

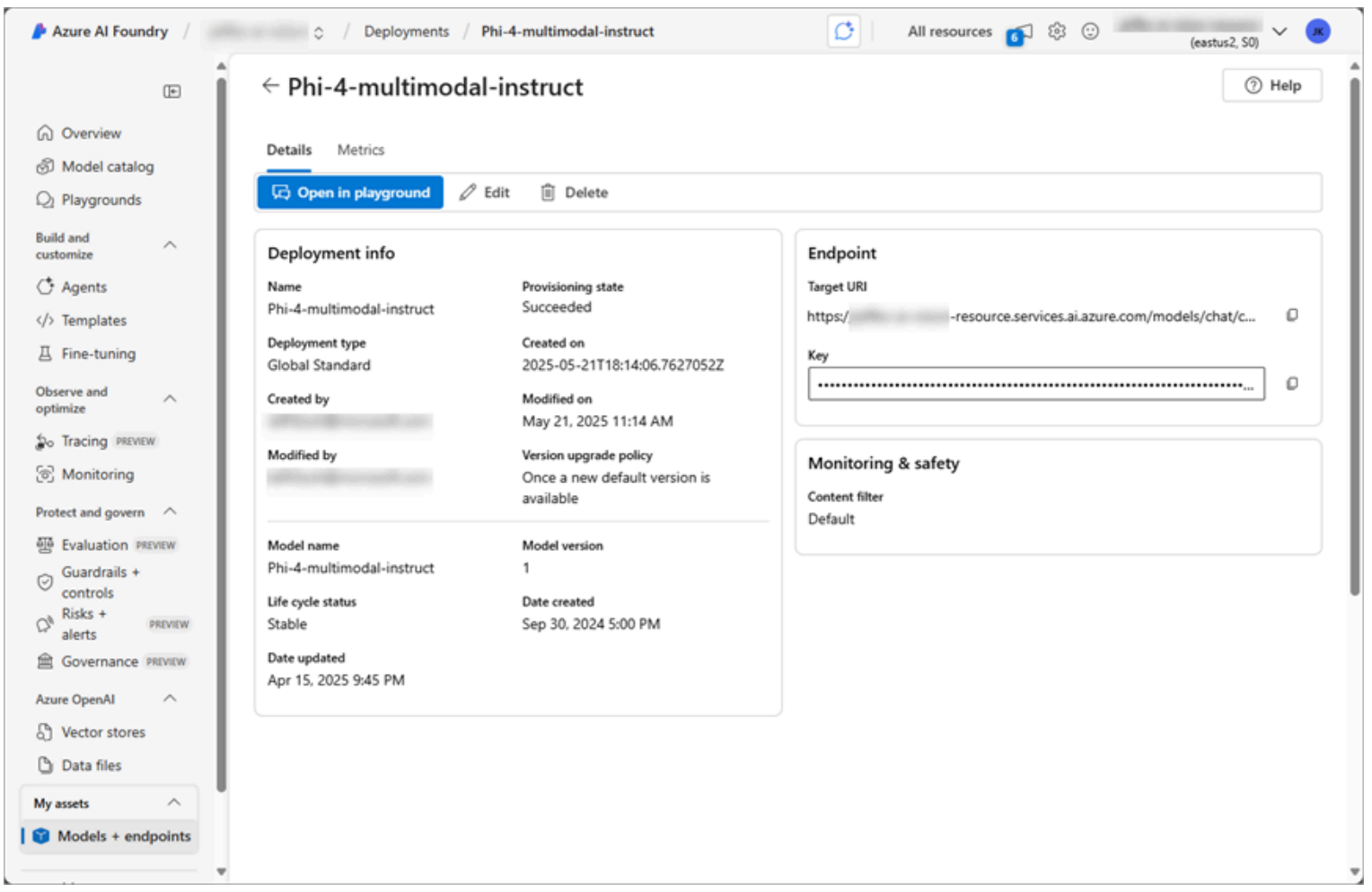
1. In the home page, in the **Explore models and capabilities** section, search for the `Phi-4-multimodal-instruct` model; which we'll use in our project.
2. In the search results, select the **Phi-4-multimodal-instruct** model to see its details, and then at the top of the page for the model, select **Use this model**.
3. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
4. Select **Customize** and specify the following settings for your hub:
    - **Azure AI Foundry resource**: *A valid name for your Azure AI Foundry resource*
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Select any **AI Foundry recommended****

> * Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

5. Select **Create** and wait for your project, including the Phi-4-multimodal-instruct model deployment you selected, to be created.

> Note: Depending on your model selection you might receive additional prompts during the project creation process. Agree to the terms and finalize the deployment.

6. When your project is created, your model will be displayed in the **Models + endpoints** page:



Test the model in the playground

Now you can test your multimodal model deployment with an image-based prompt in the chat playground.

1. Select **Open in playground** on the model deployment page.

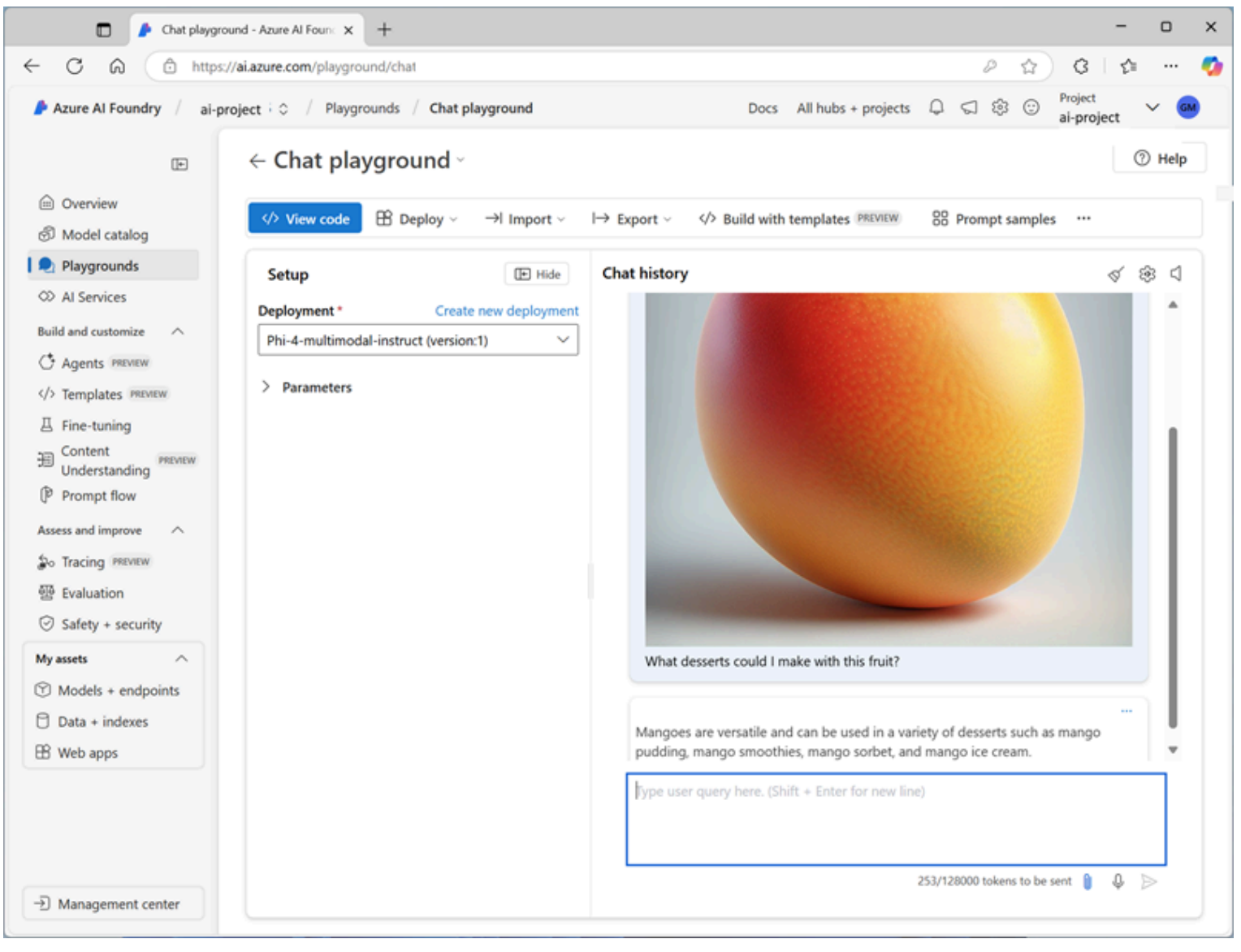
2. In a new browser tab, download mango.jpeg from

   ```
   https://github.com/MicrosoftLearning/mslearn-ai-vision/raw/refs/heads/main/Labfiles/gen-ai-vision/mango.jpeg
   ```

   and save it to a folder on your local file system.

3. On the chat playground page, in the **Setup** pane, ensure that your **Phi-4-multimodal-instruct** model model deployment is selected.

4. In the main chat session panel, under the chat input box, use the attach button (📎) to upload the *mango.jpeg* image file, and then add the text `What desserts could I make with this fruit?` and submit the prompt.



5. Review the response, which should hopefully provide relevant guidance for desserts you can make using a mango.

Create a client application

Now that you've deployed the model, you can use the deployment in a client application.

Prepare the application configuration

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

2. In the **Endpoints and keys** area, ensure the **Azure AI Foundry** library is selected, and note the **Azure AI Foundry project endpoint**. You'll use this connection string to connect to your project 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.

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

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 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.

> **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 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-vision -f
   git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
   ```

   > **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.

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

   Code Copy

   ```
   cd mslearn-ai-vision/Labfiles/gen-ai-vision/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-identity azure-ai-projects openai
   ```

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_project_endpoint** placeholder with the Foundry project endpoint (copied from the project **Overview** page in the Azure AI Foundry portal), and the **your_model_deployment** placeholder with the name you assigned to your Phi-4-multimodal-instruct model deployment.

11. After you've replaced the placeholders, in 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.

Write code to connect to your project and get a chat client for your model

> **Tip**: As you add code, be sure to maintain the correct indentation.

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

Code Copy

```
code chat-app.py
```

2. In the code file, note the existing statements that have been added at the top of the file to import the necessary SDK namespaces. Then, Find the comment **Add references**, add the following code to reference the namespaces in the libraries you installed previously:

Code Copy

```
# Add references
from azure.identity import DefaultAzureCredential
from azure.ai.projects import AIProjectClient
from openai import AzureOpenAI
```

3. In the **main** function, under the comment **Get configuration settings**, note that the code loads the project connection string and model deployment name values you defined in the configuration file.
4. In the **main** function, under the comment **Get configuration settings**, note that the code loads the project connection string and model deployment name values you defined in the configuration file.
5. Find the comment **Initialize the project client**, and add the following code to connect to your Azure AI Foundry project:

> **Tip**: Be careful to maintain the correct indentation level for your code.

Code Copy

```
# Initialize the project client
project_client = AIProjectClient(
         credential=DefaultAzureCredential(
             exclude_environment_credential=True,
             exclude_managed_identity_credential=True
         ),
         endpoint=project_endpoint,
     )
```

6. Find the comment **Get a chat client**, and add the following code to create a client object for chatting with a model:

Code Copy

```
# Get a chat client
openai_client = project_client.get_openai_client(api_version="2024-10-21")
```

Write code to submit a URL-based image prompt

1. Note that the code includes a loop to allow a user to input a prompt until they enter “quit”. Then in the loop section, find the comment **Get a response to image input**, add the following code to submit a prompt that includes the following image:

Code Copy

```
# Get a response to image input
image_url = "https://github.com/MicrosoftLearning/mslearn-ai-vision/raw/refs/heads/main/Labfiles/gen-ai-vision/orange.jpeg"
image_format = "jpeg"
request = Request(image_url, headers={"User-Agent": "Mozilla/5.0"})
image_data = base64.b64encode(urlopen(request).read()).decode("utf-8")
data_url = f"{image_data}"

response = openai_client.chat.completions.create(
     model=model_deployment,
     messages=[
         {"role": "system", "content": system_message},
         { "role": "user", "content": [
             { "type": "text", "text": prompt},
             { "type": "image_url", "image_url": {"url": data_url}}
         ] }
     ]
)
print(response.choices[0].message.content)
```

2. Use the **CTRL+S** command to save your changes to the code file - don't close it yet though.

Sign into Azure and run the app

1. In the cloud shell command-line pane, enter the following command to sign into Azure.

Code Copy

```
az login
```

You must sign into Azure - even though the cloud shell session is already authenticated.

> **Note**: In most scenarios, just using *az login* will be sufficient. However, if you have subscriptions in multiple tenants, you may need to specify the tenant by using the *–tenant* parameter. See Sign into Azure interactively using the Azure CLI for details.

2. When prompted, follow the instructions to open the sign-in page in a new tab and enter the authentication code provided and your Azure credentials. Then complete the sign in process in the command line, selecting the subscription containing your Azure AI Foundry hub if prompted.
3. After you have signed in, enter the following command to run the application:

```
python chat-app.py
```

4. When prompted, enter the following prompt:

```
Suggest some recipes that include this fruit
```

5. Review the response. Then enter `quit` to exit the program.

Modify the code to upload a local image file

1. In the code editor for your app code, in the loop section, find the code you added previously under the comment **Get a response to image input**. Then modify the code as follows, to upload this local image file:

Code Copy

```
# Get a response to image input
script_dir = Path(__file__).parent  # Get the directory of the script
image_path = script_dir / 'mystery-fruit.jpeg'
mime_type = "image/jpeg"

# Read and encode the image file
with open(image_path, "rb") as image_file:
     base64_encoded_data = base64.b64encode(image_file.read()).decode('utf-8')

# Include the image file data in the prompt
data_url = f"data:{mime_type};base64,{base64_encoded_data}"
response = openai_client.chat.completions.create(
         model=model_deployment,
         messages=[
             {"role": "system", "content": system_message},
             { "role": "user", "content": [
                 { "type": "text", "text": prompt},
                 { "type": "image_url", "image_url": {"url": data_url}}
             ] }
         ]
)
print(response.choices[0].message.content)
```

2. Use the **CTRL+S** command to save your changes to the code file. You can also close the code editor (**CTRL+Q**) if you like.
3. In the cloud shell command line pane beneath the code editor, enter the following command to run the app:

Code Copy

```
python chat-app.py
```

4. When prompted, enter the following prompt:

Code Copy

```
What is this fruit? What recipes could I use it in?
```

5. Review the response. Then enter `quit` to exit the program.

> **Note**: In this simple app, we haven't implemented logic to retain conversation history; so the model will treat each prompt as a new request with no context of the previous prompt.

Clean up

If you've finished exploring Azure AI Foundry portal, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. Open the Azure portal and view the contents of the resource group where you deployed the resources used in this exercise.
2. On the toolbar, select **Delete resource group**.
3. Enter the resource group name and confirm that you want to delete it.

Generate images with AI

In this exercise, you use the OpenAI DALL-E generative AI model to generate images. You also use the OpenAI Python SDK to create a simple app to generate images based on your prompts.

> **Note**: This exercise is based on pre-release SDK software, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

While this exercise is based on the OpenAI Python SDK, you can develop AI chat applications using multiple language-specific SDKs; including:

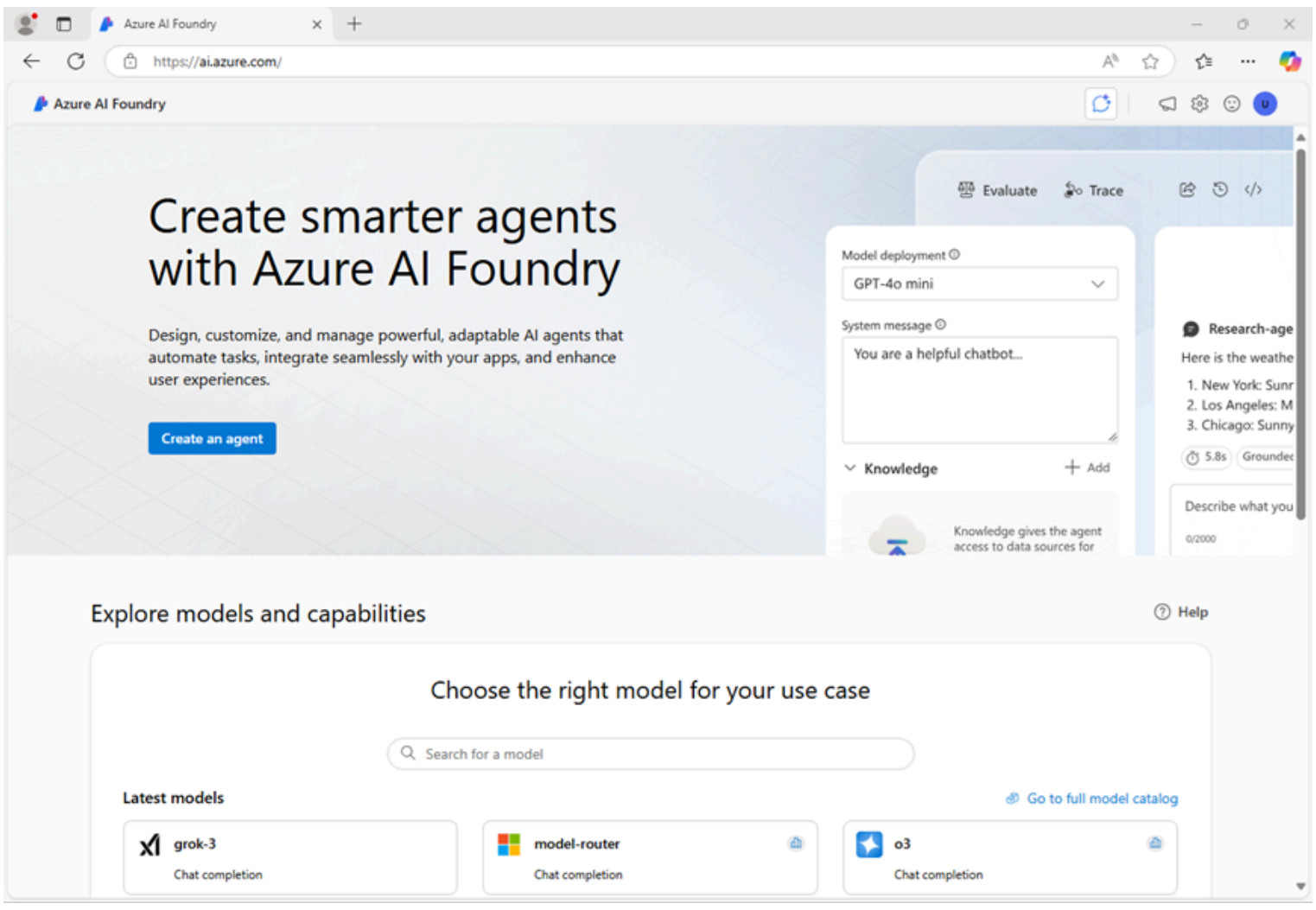
- OpenAI Projects for Microsoft .NET
- OpenAI Projects for JavaScript

This exercise takes approximately **30** minutes.

Open Azure AI Foundry portal

Let's start by signing into Azure AI Foundry portal.

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. Review the information on the home page.

Choose a model to start a project

An Azure AI *project* provides a collaborative workspace for AI development. Let's start by choosing a model that we want to work with and creating a project to use it in.

> **Note**: AI Foundry projects can be based on an *Azure AI Foundry* resource, which provides access to AI models (including Azure OpenAI), Azure AI services, and other resources for developing AI agents and chat solutions. Alternatively, projects can be based on *AI hub* resources; which include connections to Azure resources for secure storage, compute, and specialized tools. Azure AI Foundry based projects are great for developers who want to manage resources for AI agent or chat app development. AI hub based projects are more suitable for enterprise development teams working on complex AI solutions.

1. In the home page, in the **Explore models and capabilities** section, search for the `dall-e-3` model; which we'll use in our project.
2. In the search results, select the **dall-e-3** model to see its details, and then at the top of the page for the model, select **Use this model**.
3. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
4. Select **Customize** and specify the following settings for your hub:
    - **Azure AI Foundry resource**: *A valid name for your Azure AI Foundry resource*
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Create or select a resource group*
    - **Region**: *Select any **AI Foundry recommended****

    > * Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

5. Select **Create** and wait for your project, including the dall-e-3 model deployment you selected, to be created.

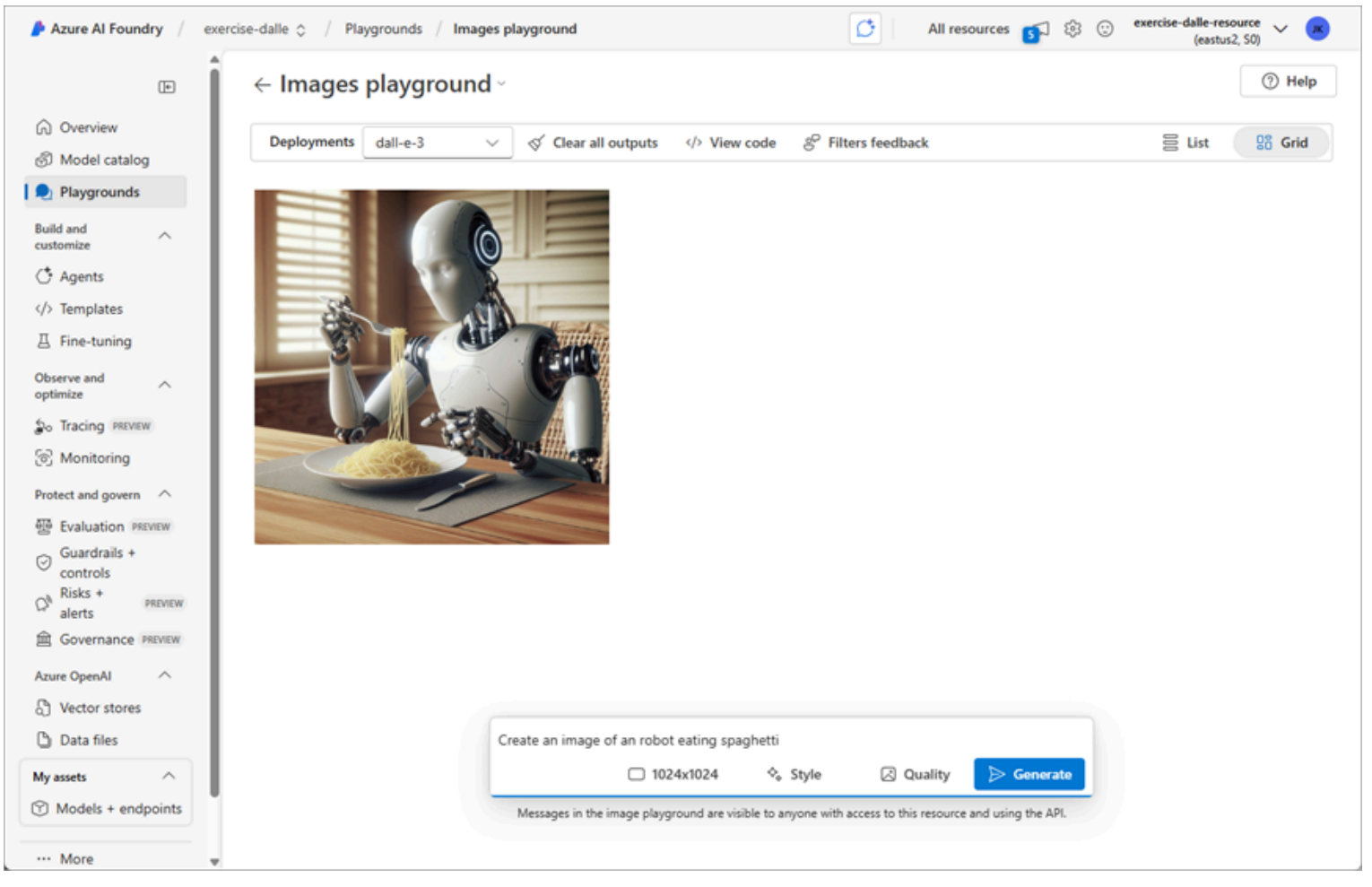
    > Note: Depending on your model selection you might receive additional prompts during the project creation process. Agree to the terms and finalize the deployment.

6. When your project is created, your model will be displayed in the **Models + endpoints** page.

Test the model in the playground

Before creating a client application, let's test the DALL-E model in the playground.

1. Select **Playgrounds**, and then **Images playground**.
2. Ensure your DALL-E model deployment is selected. Then, in the box near the bottom of the page, enter a prompt such as `Create an image of an robot eating spaghetti` and select **Generate**.
3. Review the resulting image in the playground:



4. Enter a follow-up prompt, such as `Show the robot in a restaurant` and review the resulting image.
5. Continue testing with new prompts to refine the image until you are happy with it.
6. Select the **</> View Code** button and ensure you are on the **Entra ID authentication** tab. Then record the following information for use later in the exercise. Note the values are examples, be sure to record the information from your deployment.
    - OpenAI Endpoint: *https://dall-e-aus-resource.cognitiveservices.azure.com/*
    - OpenAI API version: *2024-04-01-preview*
    - Deployment name (model name): *dall-e-3*

Create a client application

The model seems to work in the playground. Now you can use the OpenAI SDK to use it in a client application.

Prepare the application configuration

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.
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. 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***.

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-vision -f
git clone https://github.com/MicrosoftLearning/mslearn-ai-vision
```

> **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-vision/Labfiles/dalle-client/python
```

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 azure-identity openai requests
```

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. Replace the **your_endpoint**, **your_model_deployment**, and **your_api_version** placeholders with the values you recorded from the from the **Images playground**.

9. After you've replaced the placeholders, 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.

Write code to connect to your project and chat with your model

> **Tip**: As you add code, be sure to maintain the correct indentation.

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

Code Copy

```
code dalle-client.py
```

2. In the code file, note the existing statements that have been added at the top of the file to import the necessary SDK namespaces. Then, under the comment **Add references**, add the following code to reference the namespaces in the libraries you installed previously:

Code Copy

```
# Add references
from dotenv import load_dotenv
from azure.identity import DefaultAzureCredential, get_bearer_token_provider
from openai import AzureOpenAI
import requests
```

3. In the **main** function, under the comment **Get configuration settings**, note that the code loads the endpoint, API version, and model deployment name values you defined in the configuration file.

4. Under the comment **Initialize the client**, add the following code to connect to your model using the Azure credentials you are currently signed in with:

Code Copy

```
# Initialize the client
token_provider = get_bearer_token_provider(
    DefaultAzureCredential(exclude_environment_credential=True,
        exclude_managed_identity_credential=True),
    "https://cognitiveservices.azure.com/.default"
)

client = AzureOpenAI(
    api_version=api_version,
    azure_endpoint=endpoint,
    azure_ad_token_provider=token_provider
)
```

Open Azure AI Foundry portal

Choose a model to start a project

Test the model in the playground

Create a client application

Summary

Clean up

5. Note that the code includes a loop to allow a user to input a prompt until they enter “quit”. Then in the loop section, under the comment **Generate an image**, add the following code to submit the prompt and retrieve the URL for the generated image from your model:

Python

Code Copy

```
# Generate an image
result = client.images.generate(
     model=model_deployment,
     prompt=input_text,
     n=1
 )

json_response = json.loads(result.model_dump_json())
image_url = json_response["data"][0]["url"]
```

6. Note that the code in the remainder of the **main** function passes the image URL and a filename to a provided function, which downloads the generated image and saves it as a .png file.

7. Use the **CTRL+S** command to save your changes to the code file and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

Run the client application

1. In the cloud shell command-line pane, enter the following command to sign into Azure.

Code Copy

```
az login
```

You must sign into Azure - even though the cloud shell session is already authenticated.

> **Note**: In most scenarios, just using *az login* will be sufficient. However, if you have subscriptions in multiple tenants, you may need to specify the tenant by using the *–tenant* parameter. See Sign into Azure interactively using the Azure CLI for details.

2. When prompted, follow the instructions to open the sign-in page in a new tab and enter the authentication code provided and your Azure credentials. Then complete the sign in process in the command line, selecting the subscription containing your Azure AI Foundry hub if prompted.

3. In the cloud shell command line pane, enter the following command to run the app:

Code Copy

```
python dalle-client.py
```

4. When prompted, enter a request for an image, such as `Create an image of a robot eating pizza`. After a moment or two, the app should confirm that the image has been saved.

5. Try a few more prompts. When you're finished, enter `quit` to exit the program.

> **Note**: In this simple app, we haven't implemented logic to retain conversation history; so the model will treat each prompt as a new request with no context of the previous prompt.

6. To download and view the images that were generated by your app, use the cloud shell **download** command - specifying the .png file that was generated:

Code Copy

```
download ./images/image_1.png
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file.

Summary

In this exercise, you used Azure AI Foundry and the Azure OpenAI SDK to create a client application uses a DALL-E model to generate images.

Clean up

If you've finished exploring DALL-E, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. Return to the browser tab containing the Azure portal (or re-open the Azure portal at `https://portal.azure.com` in a new browser tab) and view the contents of the resource group where you deployed the resources used in this exercise.
2. On the toolbar, select **Delete resource group**.
3. Enter the resource group name and confirm that you want to delete it.