

# Develop natural language processing solutions



# Agenda



- Analyzing and translating text
- Build a question answering solution
- Build a conversational language understanding app
- Custom classification and named entity extraction
- Speech recognition, synthesis, and translation

# Analyzing text



# Learning Objectives

After completing this module, you will be able to:

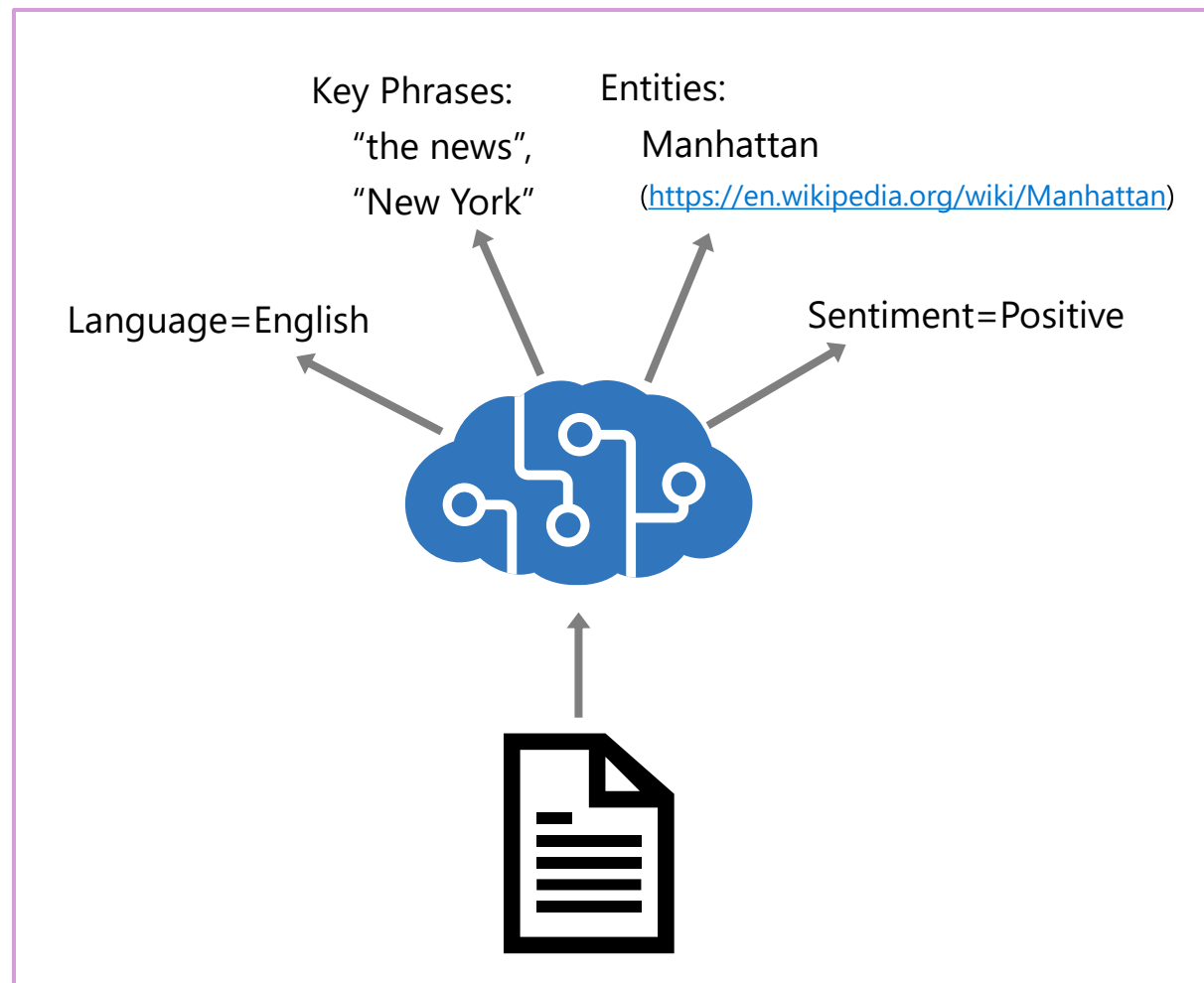
- 1 Detect language and extract key phrases
- 2 Analyze sentiment and detect PII
- 3 Summarize text
- 4 Extract entities and linked entities
- 5 Translate text

# The Azure AI Language Service

Preconfigured features:

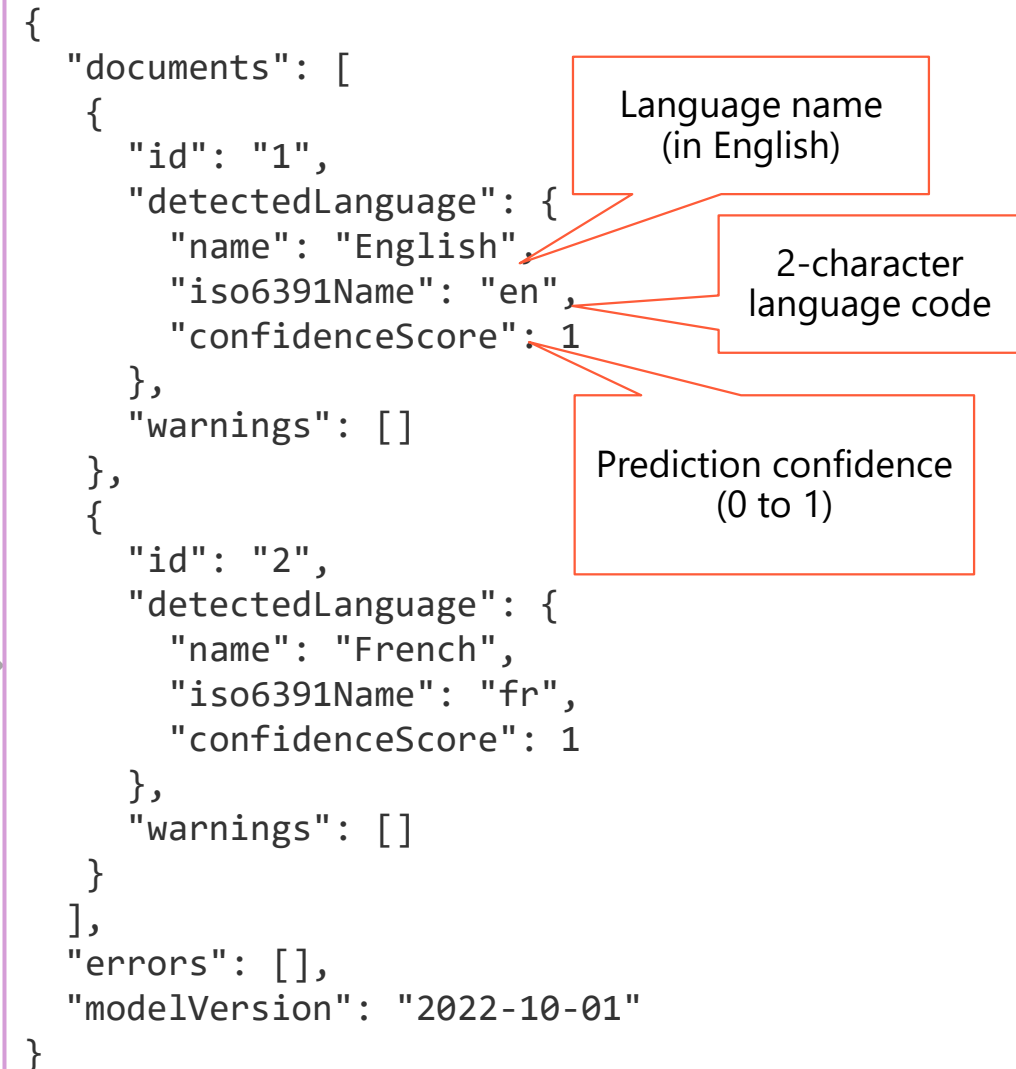
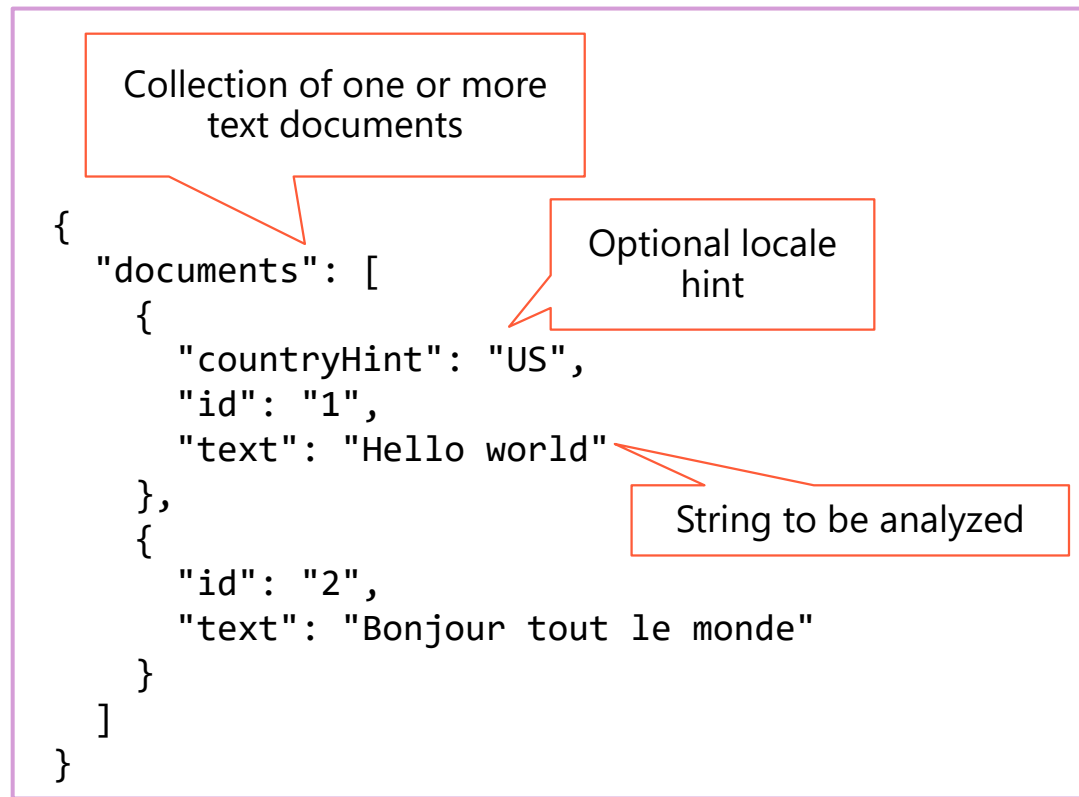
- Language detection
- Key phrase extraction
- Sentiment analysis
- Named entity recognition
- Entity linking
- Summarization
- Personal Identifying Information (PII) detection

Customizable features are covered in another section



# Language detection

- Determine the language in which text is written
- Often useful as a pre-cursor to further analysis that requires a known language



# Key phrase extraction

- Identify the main “talking points” of the text
- Works best with larger documents (up to 5,120 characters)

```
{
  "documents": [
    {
      "id": "1",
      "language": "en",
      "text": "You must be the change you wish
               to see in the world."
    },
    {
      "id": "2",
      "language": "en",
      "text": "The journey of a thousand miles
               begins with a single step."
    }
  ]
}
```

Language (defaults to English if not present)



```
{
  "documents": [
    {
      "id": "1",
      "keyPhrases": [
        "change",
        "world"
      ],
      "warnings": []
    },
    {
      "id": "2",
      "keyPhrases": [
        "miles",
        "single step",
        "journey"
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "2021-06-01"
}
```

List of key phrases in document 1

List of key phrases in document 2

# Sentiment analysis

- Scores overall document sentiment and individual sentence sentiment
- Sentence sentiment is based on confidence scores for *positive*, *negative*, and *neutral*
- Overall document sentiment is based on sentences:
  - All sentences are *neutral* = **neutral**
  - Sentences include *positive* and *neutral* = **positive**
  - Sentences include *negative* and *neutral* = **negative**
  - Sentences include *positive* and *negative* = **mixed**

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "Smile! Life is good!"
    }
  ]
}
```



```
{
  "documents": [
    {
      "id": "1",
      "sentiment": "positive",
      "confidenceScores": {
        "positive": 0.99,
        "neutral": 0.01,
        "negative": 0.00
      },
      "sentences": [
        {
          "text": "Smile!",
          "sentiment": "positive",
          "confidenceScores": {
            "positive": 0.97,
            "neutral": 0.02,
            "negative": 0.01
          },
          "offset": 0,
          "length": 6
        },
        {
          "text": "Life is good!",
          ...
        }
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "2022-11-01"
}
```

Overall sentiment

Overall confidence

Breakdown by sentence

Sentence sentiment

Sentence confidence

Sentence location

Next sentence



# Named entity recognition

- Identifies entities that are mentioned in the text
- Entities are grouped into categories and subcategories, for example:
  - Person
  - Location
  - DateTime
  - Organization
  - Address
  - Email
  - URL
  - Others...

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "Joe went to London on Saturday"
    }
  ]
}
```



```
{
  "documents": [
    {
      "id": "1",
      "entities": [
        {
          "text": "Joe",
          "category": "Person",
          "offset": 0,
          "length": 3,
          "confidenceScore": 0.62
        },
        {
          "text": "London",
          "category": "Location",
          "subcategory": "GPE",
          "offset": 12,
          "length": 6,
          "confidenceScore": 0.88
        },
        {
          "text": "Saturday",
          "category": "DateTime",
          "subcategory": "Date",
          "offset": 22,
          "length": 8,
          "confidenceScore": 0.8
        }
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "2021-01-15"
}
```

*Person entity*

*Location entity*

*DateTime entity*

# Entity Linking

- Used to disambiguate entities of the same name
  - For example, is "Venus" a planet or a goddess?
- Wikipedia provides the knowledge base
- Specific article links are determined based on entity context within the text

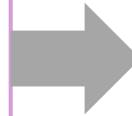
"I saw Venus shining in the sky":

<https://en.wikipedia.org/wiki/Venus>

"Venus, the goddess of beauty":

[https://en.wikipedia.org/wiki/Venus\\_\(mythology\)](https://en.wikipedia.org/wiki/Venus_(mythology))

```
{
  "documents": [
    {
      "language": "en",
      "id": "1",
      "text": "I saw Venus shining in the sky"
    }
  ]
}
```



```
{
  "documents":
  [
    {
      "id": "1",
      "entities": [
        {
          "bingId": "89253af3-5b63-e620-9227-f839138139f6"
          "name": "Venus"
          "matches": [
            {
              "text": "Venus",
              "offset": 6,
              "length": 5,
              "confidenceScore": 0.01
            }
          ],
          "language": "en",
          "id": "Venus",
          "url": "https://en.wikipedia.org/wiki/Venus",
          "dataSource": "Wikipedia"
        }
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "2021-06-01"
}
```

Named entity

Wikipedia unique article ID

Article link

# Summarization

- Can provide two different types of summarization
  - Extractive summarization: Produces summary by using most important sentences
  - Abstractive summarization: Produces a summary capturing the main idea, but not necessarily using the same words as the source document
- Can be customized by training on your own data

```
{
  "analysisInput": {
    "documents": [{
      "language": "en",
      "id": "1",
      "text": "<long paragraph about Microsoft and
technology>"
    }]
  },
  "tasks": [{
    "kind": "ExtractiveSummarization",
    "taskName": "docExtSummary1",
    "parameters": {
      "sentenceCount": 2
    }
  }]
}
```



```
{
  "documents":
  [
    {
      "id": "1",
      "sentences": [
        {
          "text": "<first sentence best summarizing document>"
          "rankScore": 0.71
          "offset": 0
          "length": 135
        },
        {
          "text": "<first sentence best summarizing document>"
          "rankScore": "0.67",
          "offset": 721
          "length": 203
        }
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "latest"
}
```

Array of sentences specified

Sentence rank score

# Personally Identifiable Information detection

- Used to detect and remove sensitive information
- Entity categories include Person, PhoneNumber, Email, Address, Credit card, and financial account identification
- Can be used in situations like applying sensitivity labels, removing information to reduce bias, and clean data for data science

```
{
  "documents": [
    {
      "id": "1",
      "language": "en",
      "text": "Call our office at 312-555-1234, or send an email to support@contoso.com"
    }
  ]
}
```



```
{
  "documents": [
    {
      "redactedText": "Call our office at *****, or send an email to ****",
      "id": "1",
      "entities": [
        {
          "text": "312-555-1234",
          "category": "PhoneNumber",
          "offset": 19,
          "length": 12,
          "confidenceScore": 0.8
        },
        {
          "text": "support@contoso.com",
          "category": "Email",
          "offset": 53,
          "length": 19,
          "confidenceScore": 0.8
        }
      ],
      "warnings": []
    }
  ],
  "errors": [],
  "modelVersion": "2021-06-01"
}
```

Text with PII removed

All entities detected

Type of PII detected

Confidence score

# Exercise – Analyze Text



**Detect Language**

**Evaluate Sentiment**

**Identify Key Phrases**

**Extract Entities**

**Extract Linked Entities**

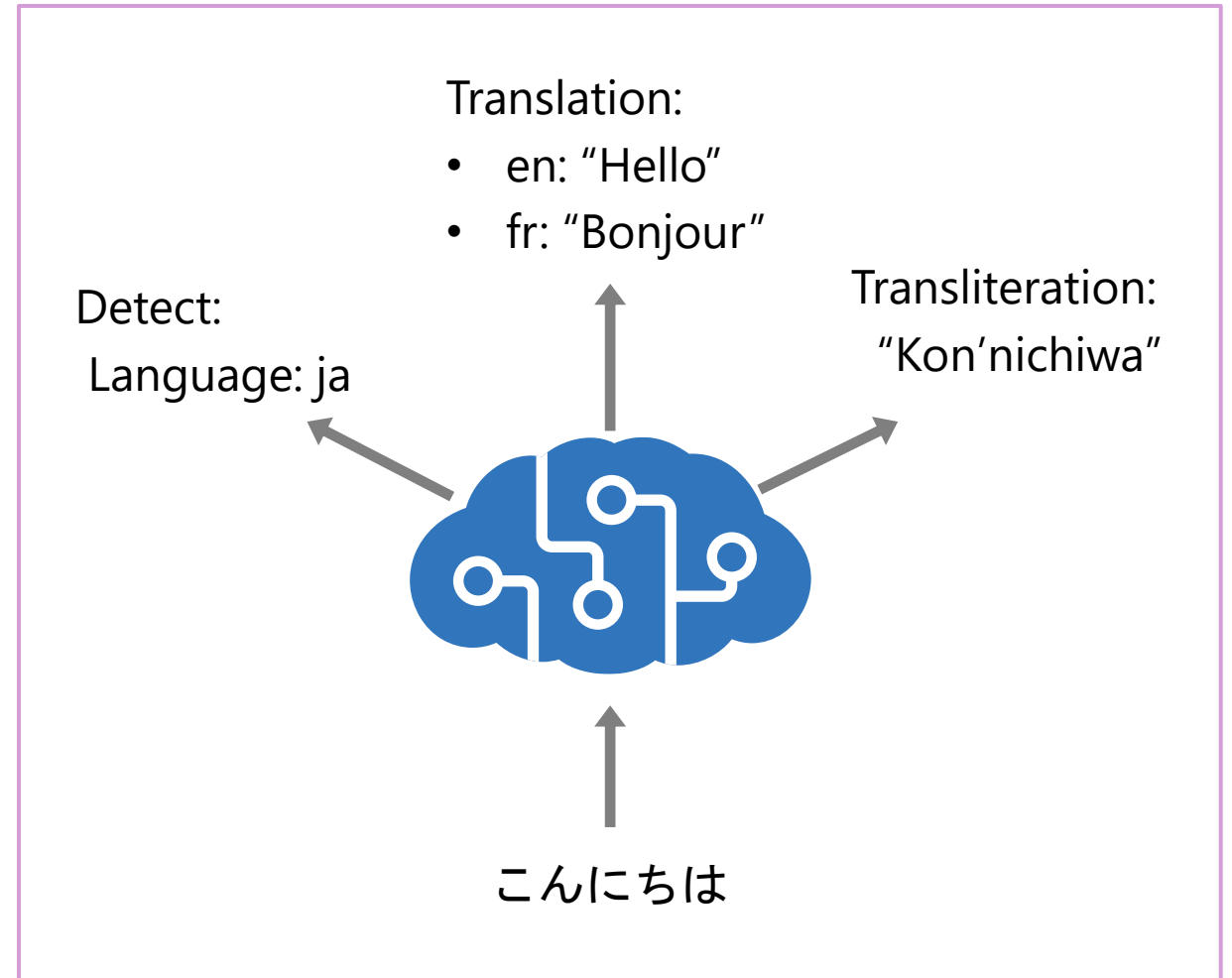
# Translating Text



# The Translator Service

## Multilingual text translation REST API

- Language *detection*
- One-to-many *translation*
- Script *transliteration*



# Detection, Translation, and Transliteration

## Detection

`https://api.cognitive.microsofttranslator.com/detect?api-version=3.0`

Body: [  
 { 'Text': 'こんにちは' }  
]

[  
 {  
 "isTranslationSupported": true,  
 "isTransliterationSupported": true,  
 "language": "ja",  
 "score": 1.0  
 }  
]

ISO Language code

## Translation

`https://api.cognitive.microsofttranslator.com/translate?api-version=3.0`  
`&from=ja&to=en&to=fr`

Body: [  
 { 'Text': 'こんにちは' }  
]

Add **to** parameters for each target language

[  
 { 'translations':  
 [  
 { 'text': 'Hello', 'to': 'en' },  
 { 'text': 'Bonjour', 'to': 'fr' }  
 ]  
 }  
]

## Transliteration

`https://api.cognitive.microsofttranslator.com/transliterate?api-version=3.0`  
`&language=ja&fromScript=Jpan&toScript=Latn`

Body: [  
 { 'Text': 'こんにちは' }  
]

Source text  
language code

Source text  
script

Target text  
script

[  
 {  
 "script": "Latn",  
 "text": "Kon'nichiwa"  
 }  
]



# Translation Options

## Word Alignment

`https://api.cognitive.microsofttranslator.com/translate?api-version=3.0  
&from=en&to=zh&includeAlignment=true`

Body: [  
  { 'Text': 'Smart Services' }  
]

[  
  { 'translations':  
    [  
      { 'text': '智能服务', 'to': 'zh-Hans',  
        'alignment': { 'proj': '0:4-0:1 6:13-2:3' }  
      }  
    ]  
  }  
]

Chars 0-4 in the source  
are chars 0-1 in the  
translation

Chars 6-13 in the  
source are chars 2-3  
in the translation

## Sentence Length

`https://api.cognitive.microsofttranslator.com/translate?api-version=3.0  
&from=en&to=fr&includeSentenceLength=true`

Body: [  
  { 'Text': 'Hello world!' }  
]

[  
  { 'translations':  
    [  
      { 'text': 'Salut tout le monde!', 'to': 'fr',  
        'sentLen': { 'srcSentLen': [12], 'transSentLen': [20] }  
      }  
    ]  
  }  
]

Source is 12  
characters

Translation is 20  
characters

## Profanity filtering

`https://api.cognitive.microsofttranslator.com/translate?api-version=3.0  
&from=en&to=de&profanityAction=Marked`

Body: [  
  { 'Text': 'JSON is [REDACTED] great!' }  
]

[  
  { 'translations':  
    [  
      { 'text': 'JSON ist \*\*\* erstaunlich.', 'to': 'de' }  
    ]  
  }  
]

Default marker for obscenity  
is asterisk

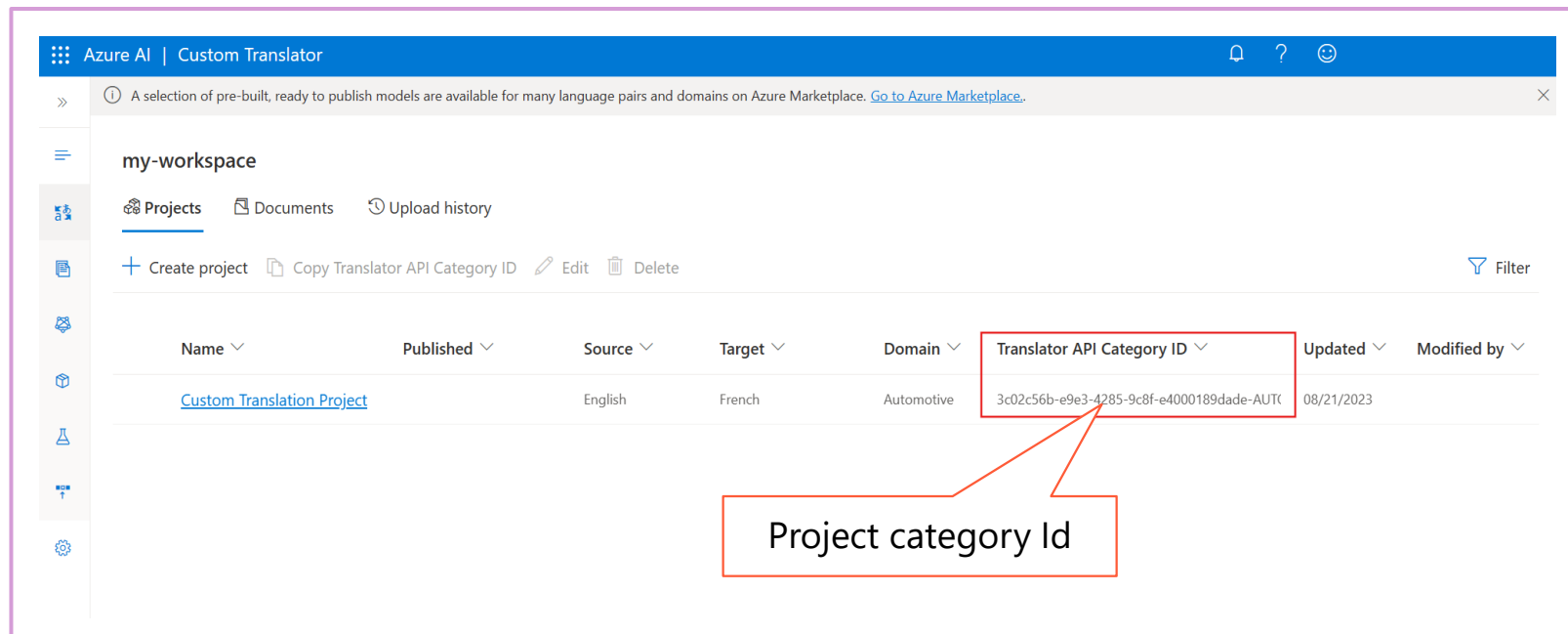
# Custom Translation

## Create a custom translation model

1. Use the Custom Translator portal
2. Link a workspace to your Azure AI Translator resource
3. Create a project
4. Upload training data files
5. Train a model

## Call your model through the Translator API

- Specify a **category** parameter with the project category Id



The screenshot shows the Azure AI Custom Translator interface. The top navigation bar is blue with the text 'Azure AI | Custom Translator'. Below it, a message states: 'A selection of pre-built, ready to publish models are available for many language pairs and domains on Azure Marketplace. [Go to Azure Marketplace.](#)'. The main content area is titled 'my-workspace' and has tabs for 'Projects', 'Documents', and 'Upload history'. The 'Projects' tab is active, showing a table of projects. The table has columns: Name, Published, Source, Target, Domain, Translator API Category ID, Updated, and Modified by. A single project is listed: 'Custom Translation Project'. The 'Translator API Category ID' for this project is '3c02c56b-e9e3-4285-9c8f-e4000189dade-AUT', which is highlighted with a red box. A red arrow points from this box to a separate box labeled 'Project category Id'.

Name	Published	Source	Target	Domain	Translator API Category ID	Updated	Modified by
<a href="#">Custom Translation Project</a>		English	French	Automotive	3c02c56b-e9e3-4285-9c8f-e4000189dade-AUT	08/21/2023	

Project category Id

# [Optional Exercise]– Translate Text



Detect language

Translate text

[mslearn-ai-language/Instructions/Exercises at main · MicrosoftLearning/mslearn-ai-language \(github.com\)](https://github.com/microsoft/mslearn-ai-language/blob/main/Instructions/Exercises%20at%20main.md)

# Build a question answering solution



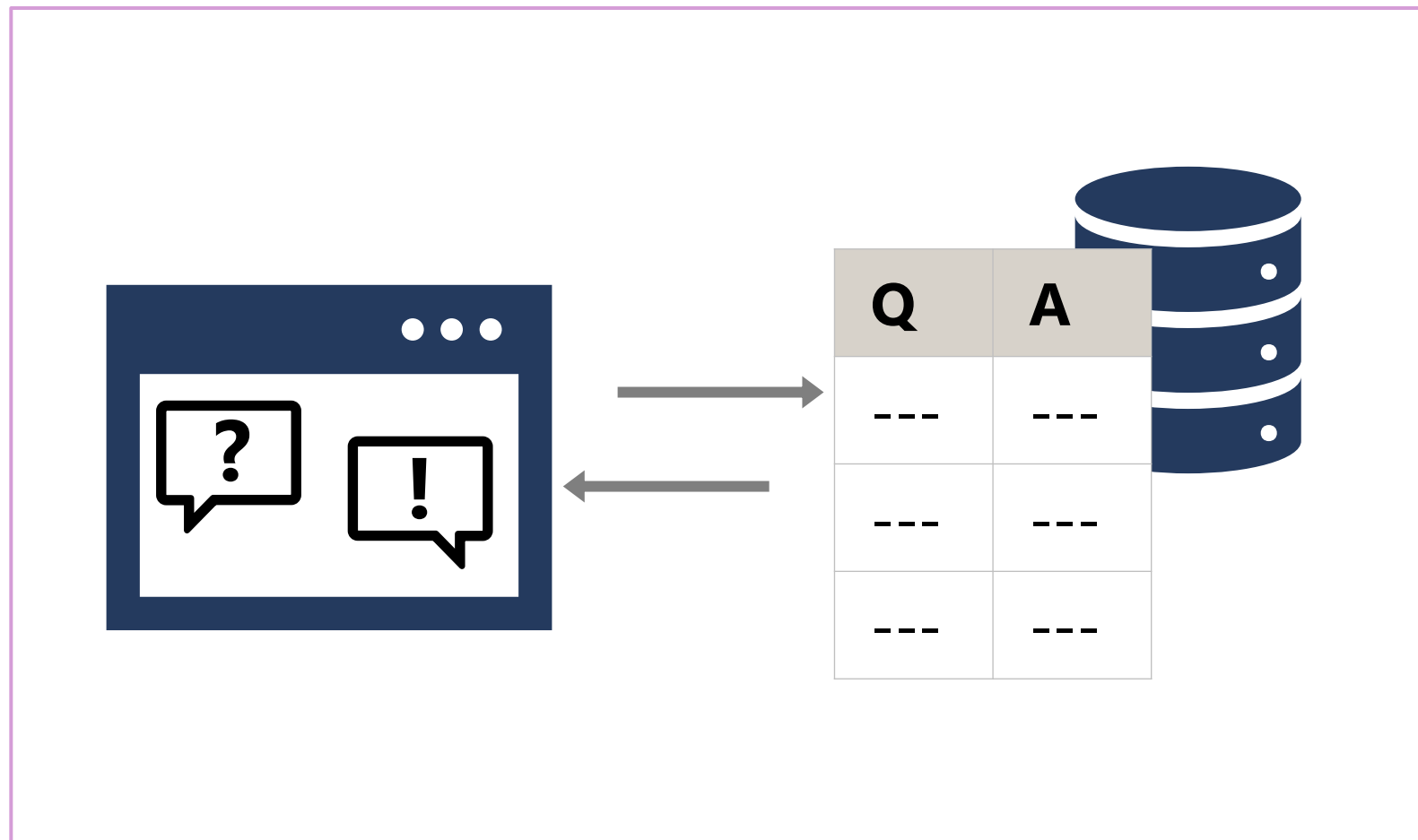
# Learning Objectives

After completing this module, you will be able to:

- 1 Describe the question answering capabilities of Azure AI Language.
- 2 Describe the differences between question answering and conversational language understanding.
- 3 Create a knowledge base.
- 4 Implement multi-turn conversation.
- 5 Test and publish a knowledge base.
- 6 Consume a published knowledge base.
- 7 Implement active learning.

# Introduction to Question Answering

- Knowledge base of question and answer pairs with natural language understanding
- Published as a REST endpoint for applications to consume
- Available through language specific SDKs



# Question Answering vs Language Understanding

## Question answering

- User submits a question, expecting an answer
- Service uses natural language understanding to match the question to an answer in the knowledge base
- Response is a static answer to a known question
- Client application presents the answer to the user

## Language understanding

- User submits an utterance, expecting an appropriate response or action
- Service uses natural language understanding to interpret the utterance, match it to an intent, and identify entities
- Response indicates the most likely intent and referenced entities
- Client application is responsible for performing appropriate action based on the detected intent

# Creating a Knowledge Base

## Use the Language Studio portal

1. Create an **Azure AI Language service** resource in your Azure subscription
2. In Language Studio, select your Azure AI Language resource and **create a Custom question answering** project.
3. Populate the knowledge base:
  - Import from existing FAQ web page
  - Upload document files
  - Add pre-defined "chit-chat" pairs
5. Create the knowledge base and edit question and answer pairs



# Multi-turn conversation

## Add follow-up prompts to define multi-turn exchanges

- Can reference existing question and answer pairs
- Can be restricted to follow-up responses only



# Testing and publishing a Knowledge Base

## Test interactively in Language Studio

- Inspect results to see confidence scores
- Add alternative phrases to improve scores as necessary

## Publish the trained knowledge base

- Creates an HTTP REST-based endpoint for client apps to consume
- Published knowledge base can be used with SDKs within your app

# Creating client apps

- REST Interface or SDKs
- Submit questions to the endpoint

```
{
  "question": "What do I need to do to
cancel a reservation?",
  "top": 2,
  "scoreThreshold": 20,
  "strictFilters": [
    {
      "name": "category",
      "value": "api"
    }
  ]
}
```



```
{
  "answers": [
    {
      "score": 27.74823341616769,
      "id": 20,
      "answer": "Call us on 555 123 4567 to
cancel a reservation.",
      "questions": [
        "How can I cancel a reservation?"
      ],
      "metadata": [
        {
          "name": "category",
          "value": "api"
        }
      ]
    }
  ]
}
```

Confidence score

Answer text

Best question match

# Improving Question Answering Performance

Enable *Active Learning* to suggest alternatives when multiple questions have similar scores for user input

- **Implicit:** The service identifies potential alternative phrases for questions; and presents suggestions in the Language Studio. Periodically review and accept/reject the suggestions.
- **Explicit:** The service returns multiple possible question matches to the user, and the user identifies the correct one. The client app then uses the API to submit feedback items, identifying the correct answer.

Create *Synonyms* for terms with the same meaning

- Add synonyms to the knowledge base through the API or Language Studio interface.

```
{
  "answers": [
    {
      "questions": ["How do I book a hotel?"],
      "answer": "Call 555-123-4567 to book.",
      "score": 76.55,
      "id": 2,
      ...
    }
  ]
}
```

```
{
  "feedbackRecords": [
    {
      "userId": "user1",
      "userQuestion": "I want to book a hotel.",
      "qnaId": 2
    }
  ]
}
```

```
{
  "synonyms": [
    {
      "alterations": [
        "reservation",
        "booking"
      ]
    }
  ]
}
```

# Exercise – Create a Question Answering solution



Create and edit a knowledge base

Train, test, and deploy the knowledge base

# Build a conversational language understanding app



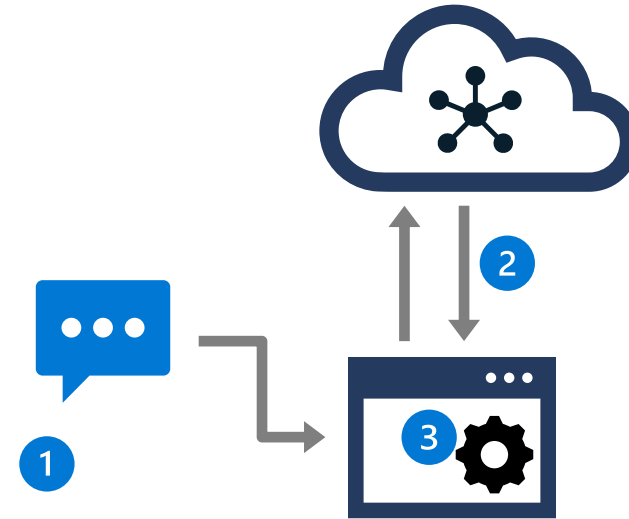
# Learning objectives

After completing this module, you will be able to:

- 1 Provision an Azure AI Language resource
- 2 Define intents, entities, and utterances
- 3 Use patterns to differentiate similar utterances and use pre-built entity components
- 4 Train, test, publish, and review a model
- 5 Describe Azure AI Language Understanding features

# Introduction to language understanding

- 1 An app accepts natural language input from a user
- 2 A language model is used to determine semantic meaning (the user's *intent*)
- 3 The app performs an appropriate action



## Natural Language Processing (NLP) requires a language model to interpret user input

Often this activity is referred to as *natural language understanding* (NLU)

*Conversational language understanding* (CLU) is an Azure service to enable you to build natural language understanding component to be used in an end-to-end conversational application.



# Intents and utterances

To train a language understanding model:

- Specify *utterances* that represent expected natural language input
- Map utterances to *intents* that assign semantic meaning

Utterance	Intent
What time is it?	GetTime
Tell me the time.	
What is the weather forecast?	GetWeather
Do I need an umbrella?	
Turn the light on.	TurnOnDevice
Switch on the fan.	
Hello	None

# Entities

Define *entities* to add specific context to intents

Utterance	Intent	Entities
What is the time?	GetTime	
What time is it in <u>London</u> ?	GetTime	Location (London)
What's the weather forecast for <u>Paris</u> ?	GetWeather	Location (Paris)
Will I need an umbrella <u>tonight</u> ?	GetWeather	Time (tonight)
What's the forecast for <u>Seattle tomorrow</u> ?	GetWeather	Location (Seattle), Time (tomorrow)
Turn the <u>light</u> on.	TurnOnDevice	Device (light)
Switch on the <u>fan</u> .	TurnOnDevice	Device (fan)

Entity types:

Learned	List	Prebuilt
Machine learned through training	Term in a defined list	Common types like numbers and date/times

# Prebuilt entity components

Prebuilt components automatically predict common types from utterances:

## Quantities

- Age, Number, Percentage, Currency, others...

## Datetime

- "June 23, 1976", "7 AM", "6:49 PM", "Tomorrow at 7 PM", "Next Week".

## Email

- "user@contoso.com"

## Phone number

- US Phone Numbers such as "+1 123 456 7890" or "(123)456-7890".

## URL

- "https://learn.microsoft.com/"

# Azure AI Language service capabilities

Features fall into two categories:

**Preconfigured features** – Can be used without labeling or training:

- Summarization
- Named entity recognition
- PII detection
- Key phrase detection
- Sentiment analysis
- Language detection

**Learned features** – Require labeling, training, and deploying to utilize

- Conversational language understanding
- Custom named entity recognition
- Custom text classification
- Question answering

# Processing predictions

Submit a request to a published slot, specifying:

- **Kind** – Indicates which language feature you're requesting. For example, **kind** is defined as *Conversation* for conversational language understanding, or *EntityRecognition* to detect entities
- **Parameters** – Indicates the values for various input parameters. These parameters vary depending on the feature.
- **Analysis input** – Specifies the input documents or text strings to be analyzed by the Azure AI Language service.

```
{
  "query": "What's the time in Edinburgh?",
  "prediction": {
    "topIntent": "GetTime",
    "projectKind": "Conversation",
    "intents": [
      {
        "category": "GetTime"
        "confidenceScore": 0.9
      },
      Any other predicted intents with scores
    ],
    "entities": [
      {
        "text": "Edinburgh",
        "category": "location",
        "offset": 18,
        "length": 9
        <entity location information>
      }, Any other predicted entities
    ]
  }
}
```

Query text is included in response

Highest scoring intent

All possible intents and their scores

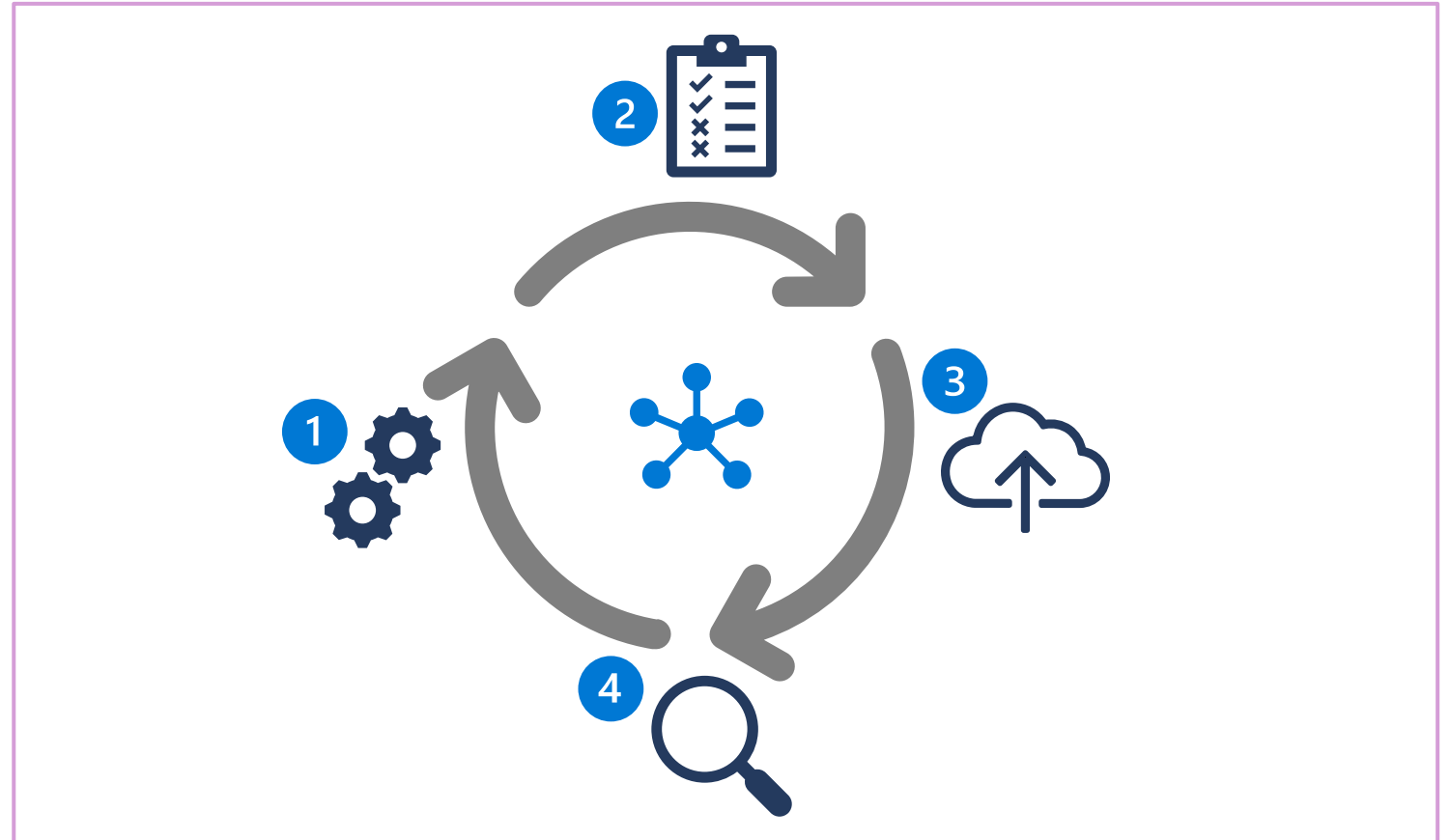
Entities detected

Text of detected entity

Type of entity detected

# Training, testing, publishing, and reviewing

- 1 Train a model to learn intents and entities from sample utterances
- 2 Test the model interactively or using a testing dataset with known labels
- 3 Deploy a trained model to a public endpoint so client apps can use it
- 4 Review predictions and iterate on utterances to train your model



# Exercise – Create a conversational language understanding app



Create intents

Create entities

Test and publish a language model

Query your model from a client app

# Custom classification and named entity extraction





# Learning Objectives

After completing this module, you will be able to:

- 1 Label documents, train and deploy models for custom classification
- 2 Understand model performance and see where to improve your model
- 3 Use your custom model in an app

# Custom Text Classification

Assign custom labels to documents

1. Connect to documents in Azure
2. Define class labels to assign to your documents
3. Label documents
4. Train your model

Call your model through the Language API

- Specify project and deployment name

Can be single label or multi label projects

Language Studio > Custom Text Classification > ClassifyLab - Data labeling

**Data labeling** ✓ Saved

Select a document to categorize it into a class or [use Azure Machine Learning to label](#). After labeling the documents and adding them to training or testing sets, you'll be ready to create a model with this data in [Training jobs](#).

All documents view ▾ Search Filter

<input type="radio"/> Document name ↑ ▾	Labeled as ▾	Dataset ▾
<input checked="" type="radio"/> Article 1.txt	Sports	Training
<input type="radio"/> Article 10.txt	News	Training
<input type="radio"/> Article 11.txt	Entertainment	Testing
<input type="radio"/> Article 12.txt	News	Testing
<input type="radio"/> Article 13.txt	Sports	Testing

**Activity pane**

Labels Distribution Recommendations ...

✓ Ready for training ▾

+ Add class Auto-label 🔍

☐ None

☐ Classifieds

☒ Sports

☐ News

☐ Entertainment

Add additional classes

# Custom Named Entity Recognition

Assign custom labels to entities in your documents

1. Connect to documents in Azure
2. Define entity labels to assign to your documents
3. Label documents completely and consistently
4. Train your model

Call your model through the Language API

- Specify project and deployment name

The screenshot shows the 'Data labeling' interface in Azure Language Studio. The breadcrumb path is 'Language Studio > Custom Named Entity Recognition > Demo - Data labeling'. The page title is 'Data labeling' with a green checkmark and 'Saved'. Below the title, there is a description: 'Select a document to annotate its text with entity labels or [use Azure Machine Learning to label](#). After labeling the documents and adding them to training or testing sets, you'll be ready to create a model with this data in [Training jobs](#).' The document name is 'Article 1.txt'. The text content is: 'How The Footballers Completed One Of The Worst Collapses In Just 4 Minutes' followed by a paragraph: 'With four minutes left in the Championship Game, the [San Francisco] Footballers had the 19-7 lead on Seattle.' The word 'Seattle' is labeled as 'City'. A search box for 'San Francisco' is open, showing 'City' as a suggestion. On the right, the 'Activity pane' has tabs for 'Labels', 'Distribution', and 'Recommendations'. The 'Labels' tab is active, showing 'Not ready for training' and a '+ Add entity' button. A red arrow points from the 'Add additional entities' text to the '+ Add entity' button.

Language Studio > Custom Named Entity Recognition > Demo - Data labeling

**Data labeling** ✓ Saved

Select a document to annotate its text with entity labels or [use Azure Machine Learning to label](#). After labeling the documents and adding them to training or testing sets, you'll be ready to create a model with this data in [Training jobs](#).

Single document view Document name: Article 1.txt Accept all Reject all

How The Footballers Completed One Of The Worst Collapses In Just 4 Minutes

With four minutes left in the Championship Game, the [San Francisco] Footballers had the 19-7 lead on Seattle.

City

From that point on, everything that could possibly go wrong for the Footballers went wr

**Activity pane**

Labels Distribution Recommendations

Not ready for training

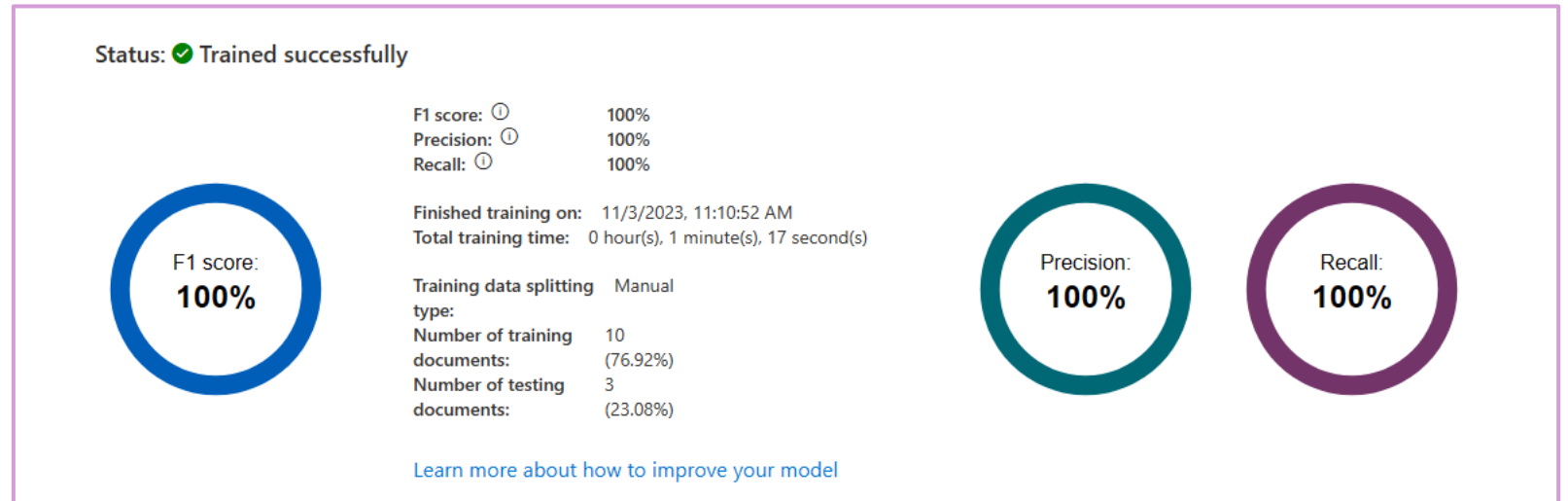
+ Add entity Auto-label

City (1)

Add additional entities

# Review and improve a model

- 1 Train a model to teach labels or entities
- 2 Review model performance to determine how to improve performance, including Confusion matrix
- 3 Determine what cases need to be added to your training data
- 4 Retrain your model with new data included, and repeat as necessary



# [Optional Exercise] – Extract custom entities



Create a custom named entity recognition project

Create and label entities

Test and publish a custom model

Query your model from a client app

[mslearn-ai-language/Instructions/Exercises at main · MicrosoftLearning/mslearn-ai-language \(github.com\)](https://github.com/microsoft/mslearn-ai-language/Instructions/Exercises%20at%20main)

# Speech recognition, translation and synthesis



# Learning Objectives

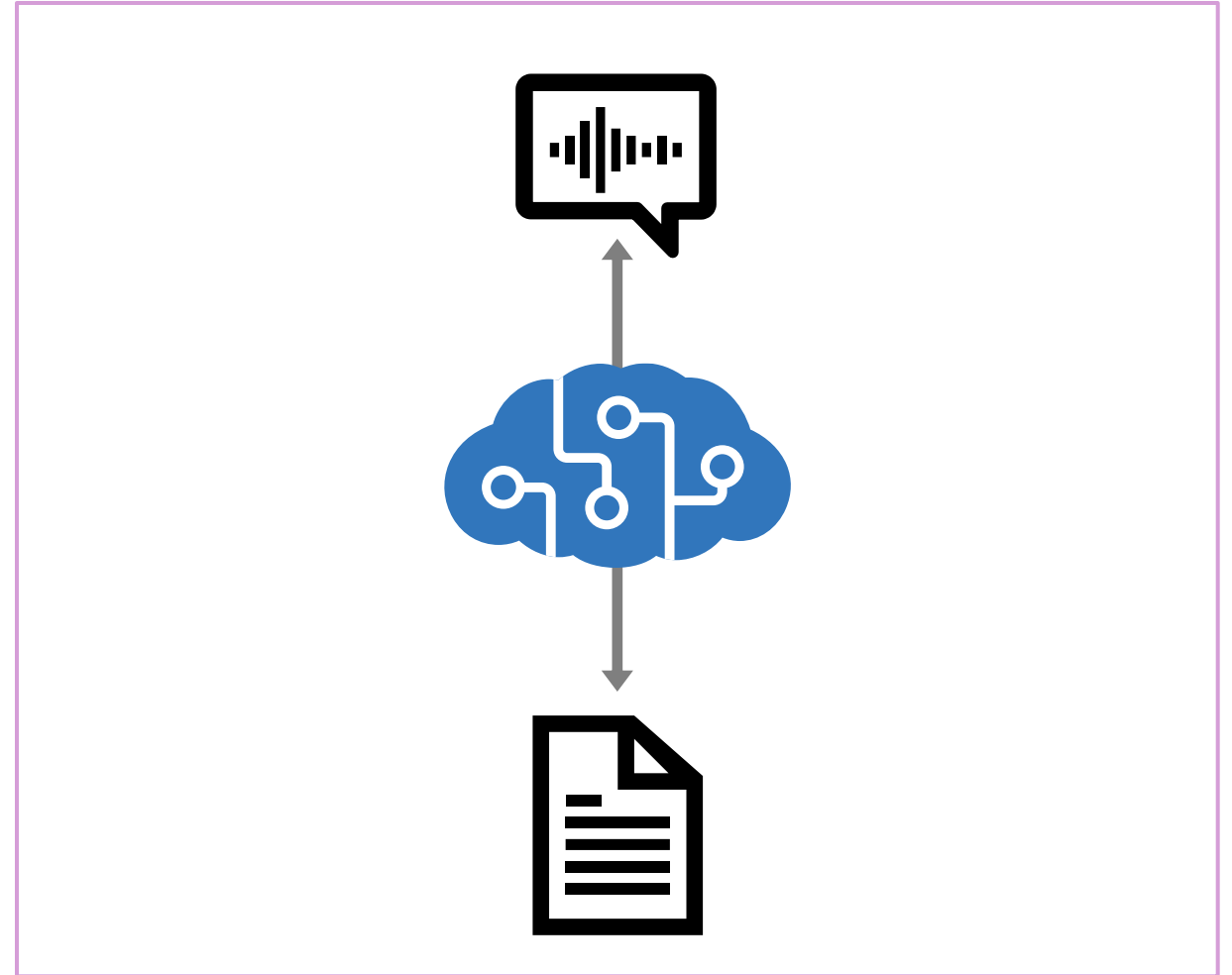
After completing this module, you will be able to:

- 1 Provision an Azure resource for the Azure AI Speech service
- 2 Use the Speech to text API to implement speech recognition
- 3 Use the Text to speech API to implement speech synthesis
- 4 Configure audio format and voices
- 5 Use Speech Synthesis Markup Language (SSML)

# The Speech Service

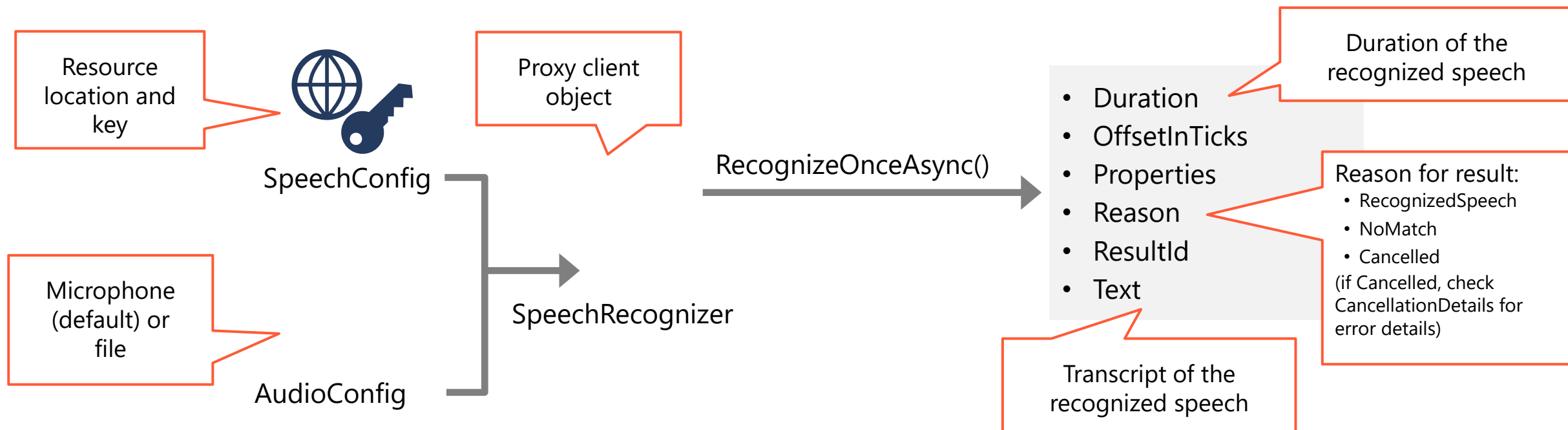
## Speech APIs

- Speech-to-Text API (speech recognition)
- Text-to-Speech API (speech synthesis)
- Speech Translation API
- Speaker Recognition API
- Intent Recognition (uses conversational language understanding)





# Speech-to-Text

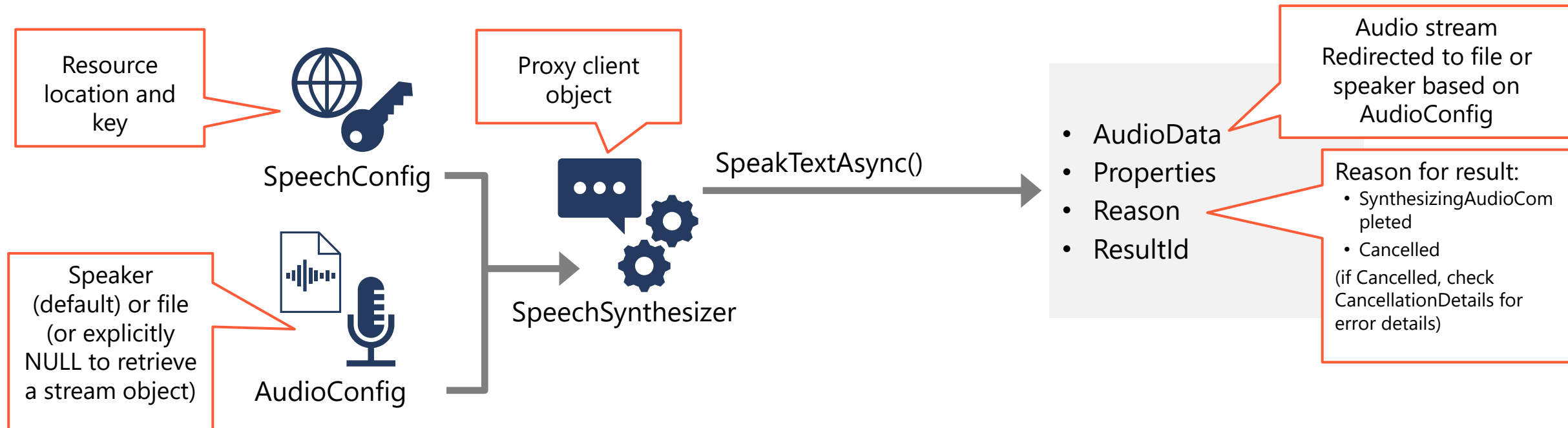


## Two REST APIs:

- Speech-to-text API – Used by Azure AI Speech SDK – preferred for most scenarios
- Speech-to-text Short Audio API – Useful for short (up to 60s) of audio

## Azure AI Speech SDK (.NET, Python, JavaScript, etc.)

# Text-to-Speech



## Two REST APIs:

- Text-to-speech API – Suitable for most scenarios
- Batch synthesis API – Convert large volumes of text to audio files

## Azure AI Speech SDK (.NET, Python, JavaScript, etc.)

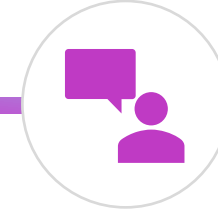
# Audio Format and Voices



## Audio Format

Select an audio format to specify:

- Audio file type
- Sample-rate
- Bit-depth



## Voices

- Standard voices: Synthetic voices created from audio samples
- Neural voices: Natural sounding voices created using deep neural networks

```
speechConfig.SetSpeechSynthesisOutputFormat(SpeechSynthesisOutputFormat.Riff24Khz16BitMonoPcm);
```

```
speechConfig.SpeechSynthesisVoiceName = "en-GB-George";
```

# Speech Synthesis Markup Language (SSML)

XML-based language with customization options:

- Speaking styles (Neural voices only)
- Pauses and silence
- Phonemes (phonetic pronunciations)
- Prosody (speaking pitch, range, rate, etc.)
- "say-as" (number, date, time, address, etc.)
- Insert recorded speech or background audio

```
SpeakSsmlAsync( ssml-string );
```

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
      xmlns:mstts="https://www.w3.org/2001/mstts" xml:lang="en-US">
  <voice name="en-US-AriaNeural">
    <mstts:express-as style="cheerful">
      I say tomato
    </mstts:express-as>
  </voice>
  <voice name="en-US-GuyNeural">
    I say <phoneme alphabet="sapi" ph="t ao m ae t ow"> tomato </phoneme>.
    <break strength="weak"/> Lets call the whole thing off!
  </voice>
</speak>
```

Multiple voices in a single synthesis

Speaking style

Phonetic pronunciation

Pause

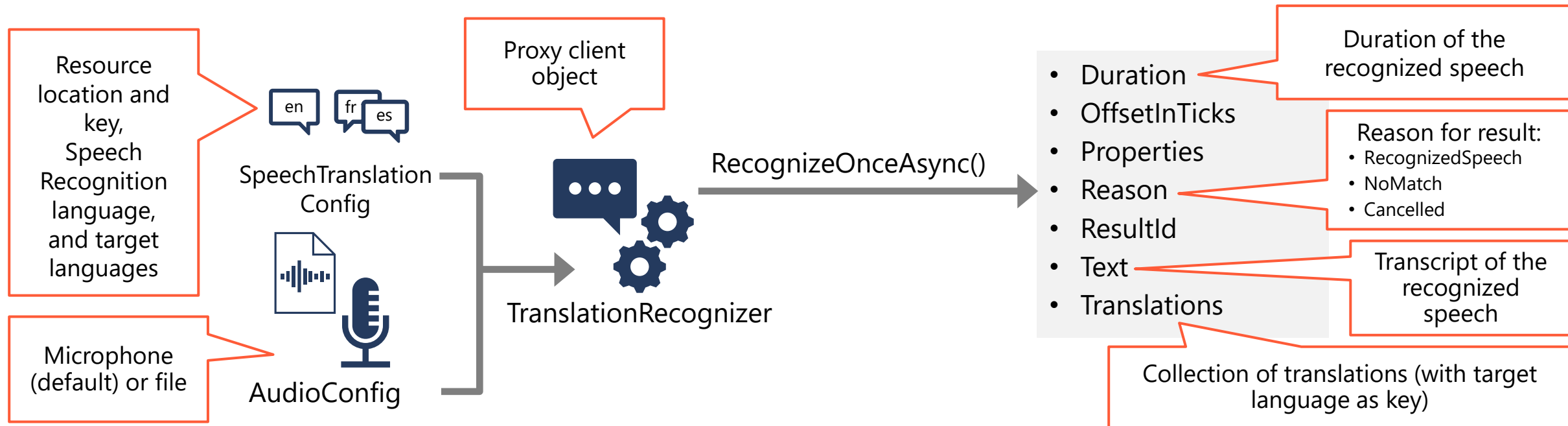
# Exercise – Recognize and Synthesize Speech



Recognize Speech

Synthesize Speech

# Translating Speech to Text



## Translation builds on speech recognition:

1. Recognize and transcribe spoken input in speech recognition language
2. Return translations for one or more target languages

# Synthesizing Translations as Speech

## Event-based synthesis

- Only supported for 1:1 translation (single target language)
- Specify desired voice in the **TranslationConfig**
- Use the **Synthesizing** event to retrieve audio stream
- Create an event handler
- Use `Result.GetAudio()` to retrieve byte stream

## Manual synthesis

- Use for multiple target languages
- Translate to text then use Text-to-Speech API to synthesize each translation in the results

# Extended interactive exercises



Custom text classification

Translate speech

<https://aka.ms/azure-ai-language-lp>



# Knowledge check



1

Which object should you use to specify that the speech input to be transcribed to text is in an audio file?

- ☐ SpeechConfig
- ☒ AudioConfig
- ☐ SpeechRecognizer

2

You have analyzed text that contains the word “Paris”. How might you determine if this word refers to the French city or the character in Homer’s *The Iliad*?

- ☐ Use the Azure AI Language service to extract key phrases.
- ☐ Use the Azure AI Language service to analyze sentiment.
- ☒ Use the Azure AI Language service to extract linked entities.

3

When translating speech, in which cases can you use the Synthesizing event to synthesize the translations and speech?

- ☒ Only when translating to a single target language
- ☐ Only when translating to multiple target languages
- ☐ When translating to one or more target languages

# Knowledge check



- 4** Your app must interpret a command to book a flight to a specified city, such as “Book a flight to Paris.” How should you model the city element of the command?
- ☐ As an intent.
  - ☐ As an utterance.
  - ☒ As an entity.
- 5** Your language model needs to detect an email when present in an utterance. What is the simplest way to extract that email?
- ☐ Use Regular Expression entities.
  - ☒ Use Prebuilt entity components
  - ☐ Use Learned entity components.
- 6** How should you create an application that monitors the comments on your company’s web site and flags any indication that customers are unhappy?
- ☐ Use the Azure AI Translator service to detect profanities in comments.
  - ☒ Use the Azure AI Language service to perform sentiment analysis of the comments.
  - ☐ Use the Azure AI Language service to extract named entities from the comments

# Learning Path Recap

In this learning path, we learned to:

- Analyze and translate text
- Build a conversational language understanding model
- Build a question answering solution
- Speech recognition, synthesis, and translation
- Connect an app to Azure AI Language resources

