**Implement intent recognition**

An intent is something the user wants to do: book a flight, check the weather, or make a call. With intent recognition, your applications, tools, and devices can determine what the user wants to initiate or do based on options. You define user intent in the intent recognizer or conversational language understanding (CLU) model.

**Pattern matching**

The Speech SDK provides an embedded pattern matcher that you can use to recognize intents in a strict way. This is useful for when you need a quick offline solution. This works especially well when the user is going to be trained in some way or can be expected to use specific phrases to trigger intents. For example: "Go to floor seven", or "Turn on the lamp" etc. It's recommended to start here and if it no longer meets your needs, switch to using [CLU](https://learn.microsoft.com/en-us/azure/ai-services/speech-service/intent-recognition#conversational-language-understanding) or a combination of the two.

Use pattern matching if:

* You're only interested in matching strictly what the user said. These patterns match more aggressively than [conversational language understanding (CLU)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/overview).
* You don't have access to a CLU model, but still want intents.

**Conversational Language Understanding**

Conversational language understanding (CLU) enables users to build custom natural language understanding models to predict the overall intention of an incoming utterance and extract important information from it.

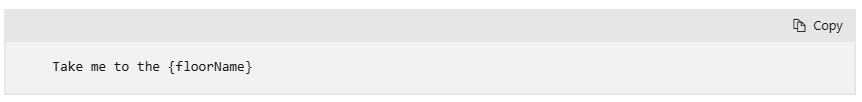
Both a Speech resource and Language resource are required to use CLU with the Speech SDK. The Speech resource is used to transcribe the user's speech into text, and the Language resource is used to recognize the intent of the utterance.

**Recognize intents with simple language pattern matching**

**Speech and simple patterns**

The simple patterns are a feature of the Speech SDK and need an Azure AI services resource or a Unified Speech resource.

A pattern is a phrase that includes an Entity somewhere within it. An Entity is defined by wrapping a word in curly brackets. This example defines an Entity with the ID "floorName", which is case-sensitive:

****

All other special characters and punctuation are ignored.

Intents are added using calls to the IntentRecognizer->AddIntent() API.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with a black border

Description automatically generated with medium confidence

**Build and run your app**

Now you're ready to build your app and test our speech recognition using the Speech service.

1. **Compile the code** - From the menu bar of Visual Studio, choose **Build** > **Build Solution**.
2. **Start your app** - From the menu bar, choose **Debug** > **Start Debugging** or press F5.
3. **Start recognition** - It will prompt you to say something. The default language is English. Your speech is sent to the Speech service, transcribed as text, and rendered in the console.

For example, if you say, "Take me to floor 7", this should be the output:

A screenshot of a computer error

Description automatically generated

**Recognize intents with Conversational Language Understanding**

**Prerequisites**

* An Azure subscription. You can [create one for free](https://azure.microsoft.com/free/cognitive-services).
* [Create a Language resource](https://portal.azure.com/#create/Microsoft.CognitiveServicesTextAnalytics) in the Azure portal.
* Get the Language resource key and endpoint. After your Language resource is deployed, select **Go to resource** to view and manage keys.
* [Create a Speech resource](https://portal.azure.com/#create/Microsoft.CognitiveServicesSpeechServices) in the Azure portal.
* Get the Speech resource key and region. After your Speech resource is deployed, select **Go to resource** to view and manage keys.

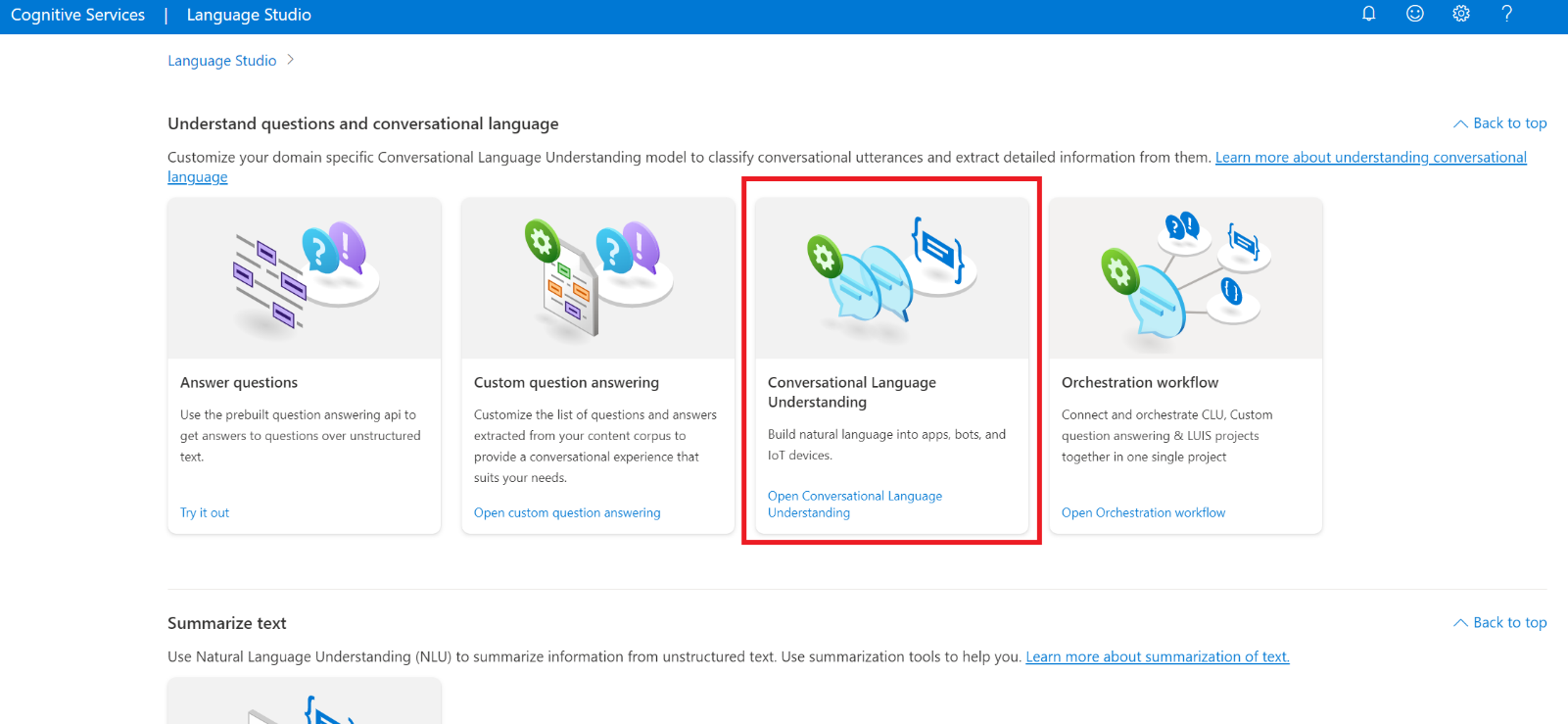
**Set environment variables**

* To set the **LANGUAGE\_KEY** environment variable, replace your-language-key with one of the keys for your resource.
* To set the **LANGUAGE\_ENDPOINT** environment variable, replace your-language-endpoint with one of the regions for your resource.
* To set the **SPEECH\_KEY** environment variable, replace your-speech-key with one of the keys for your resource.
* To set the **SPEECH\_REGION** environment variable, replace your-speech-region with one of the regions for your resource.

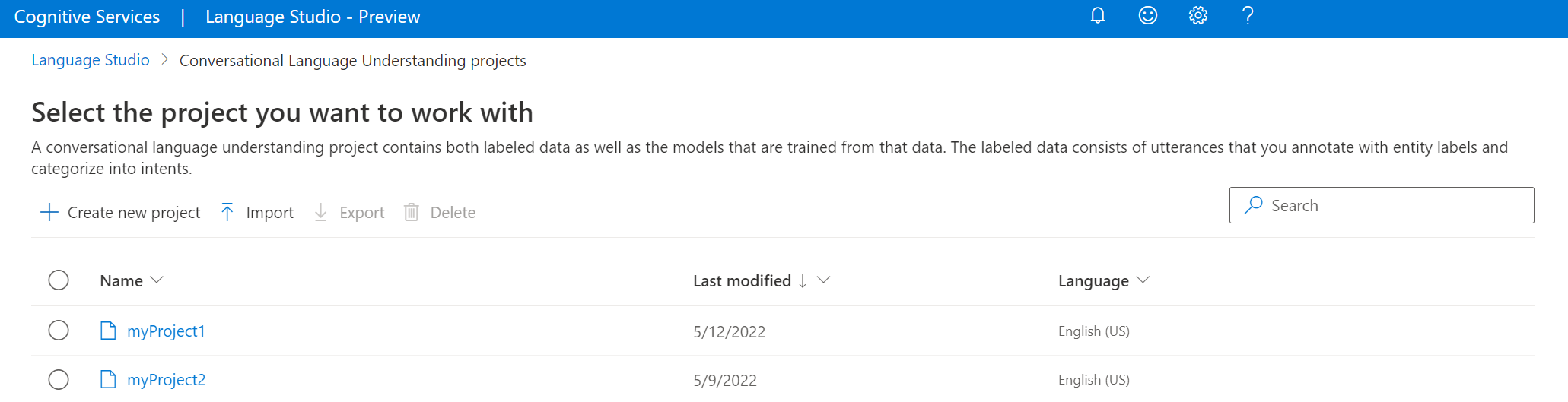
**Create a conversational language understanding project**

For this quickstart, you can download [this sample home automation project](https://github.com/Azure-Samples/cognitive-services-sample-data-files/blob/master/language-service/CLU/HomeAutomationDemo.json) and import it. This project can predict the intended commands from user input, such as turning lights on and off.

1. Under the **Understand questions and conversational language** section of Language Studio, select **Conversational language understanding**.

[](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/media/select-custom-clu.png#lightbox)

1. This will bring you to the **Conversational language understanding projects** page. Next to the **Create new project** button select **Import**.

[](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/media/projects-page.png#lightbox)

1. In the window that appears, upload the JSON file you want to import. Make sure that your file follows the [supported JSON format](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/concepts/data-formats).

Once the upload is complete, you will land on **Schema definition** page. For this quickstart, the schema is already built, and utterances are already labeled with intents and entities.

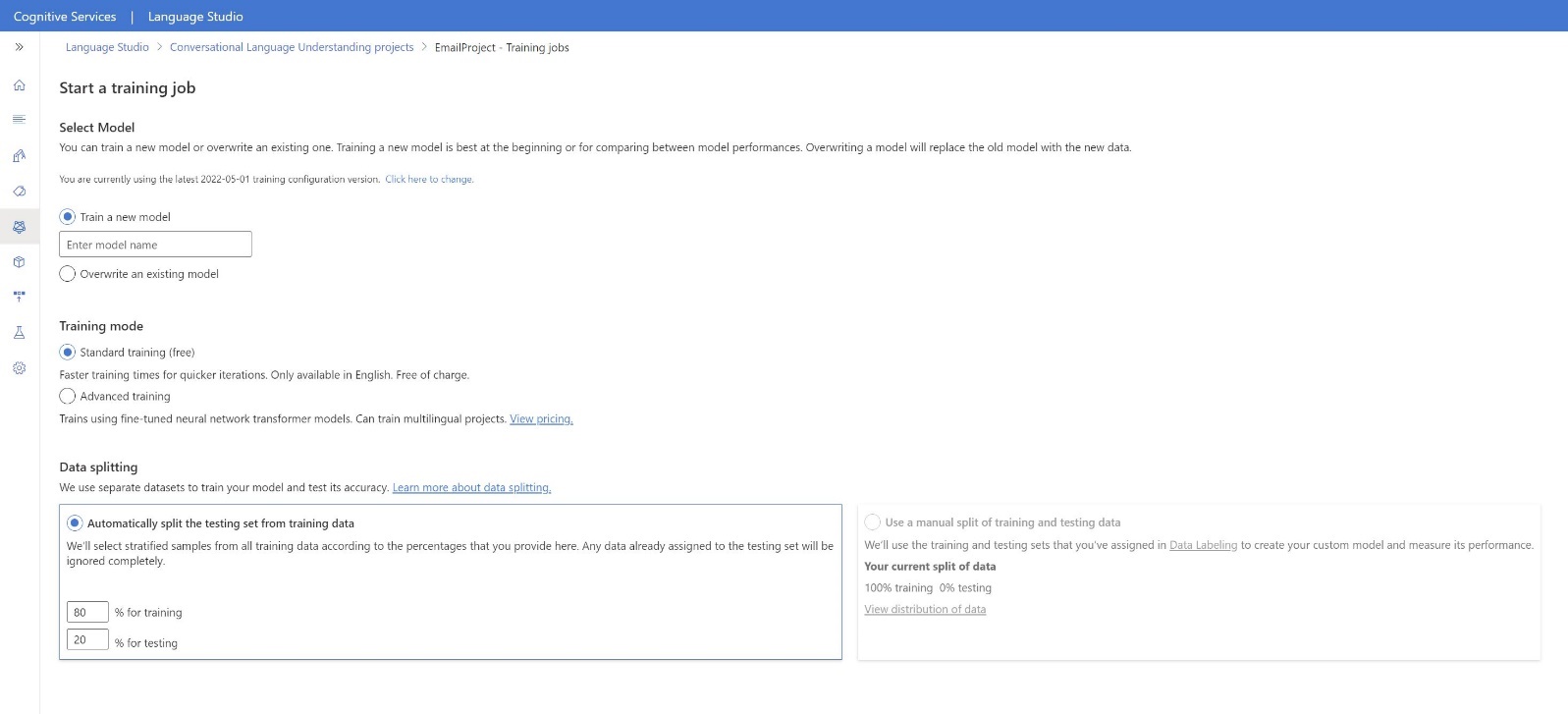
**Train your model**

Typically, after you create a project, you should [build a schema](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/how-to/build-schema) and [label utterances](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/how-to/tag-utterances). For this quickstart, we already imported a ready project with built schema and labeled utterances.

To train a model, you need to start a training job. The output of a successful training job is your trained model.

To start training your model from within the [Language Studio](https://aka.ms/languageStudio):

1. Select **Train model** from the left side menu.
2. Select **Start a training job** from the top menu.
3. Select **Train a new model** and enter a new model name in the text box. Otherwise to replace an existing model with a model trained on the new data, select **Overwrite an existing model** and then select an existing model. Overwriting a trained model is irreversible, but it won't affect your deployed models until you deploy the new model.
4. Select training mode. You can choose **Standard training** for faster training, but it is only available for English. Or you can choose **Advanced training** which is supported for other languages and multilingual projects, but it involves longer training times. Learn more about [training modes](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/how-to/train-model#training-modes).
5. Select a [data splitting](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/how-to/train-model#data-splitting) method. You can choose **Automatically splitting the testing set from training data** where the system will split your utterances between the training and testing sets, according to the specified percentages. Or you can **Use a manual split of training and testing data**, this option is only enabled if you have added utterances to your testing set when you [labeled your utterances](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/how-to/tag-utterances).
6. Select the **Train** button.

[](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/media/train-model.png#lightbox)

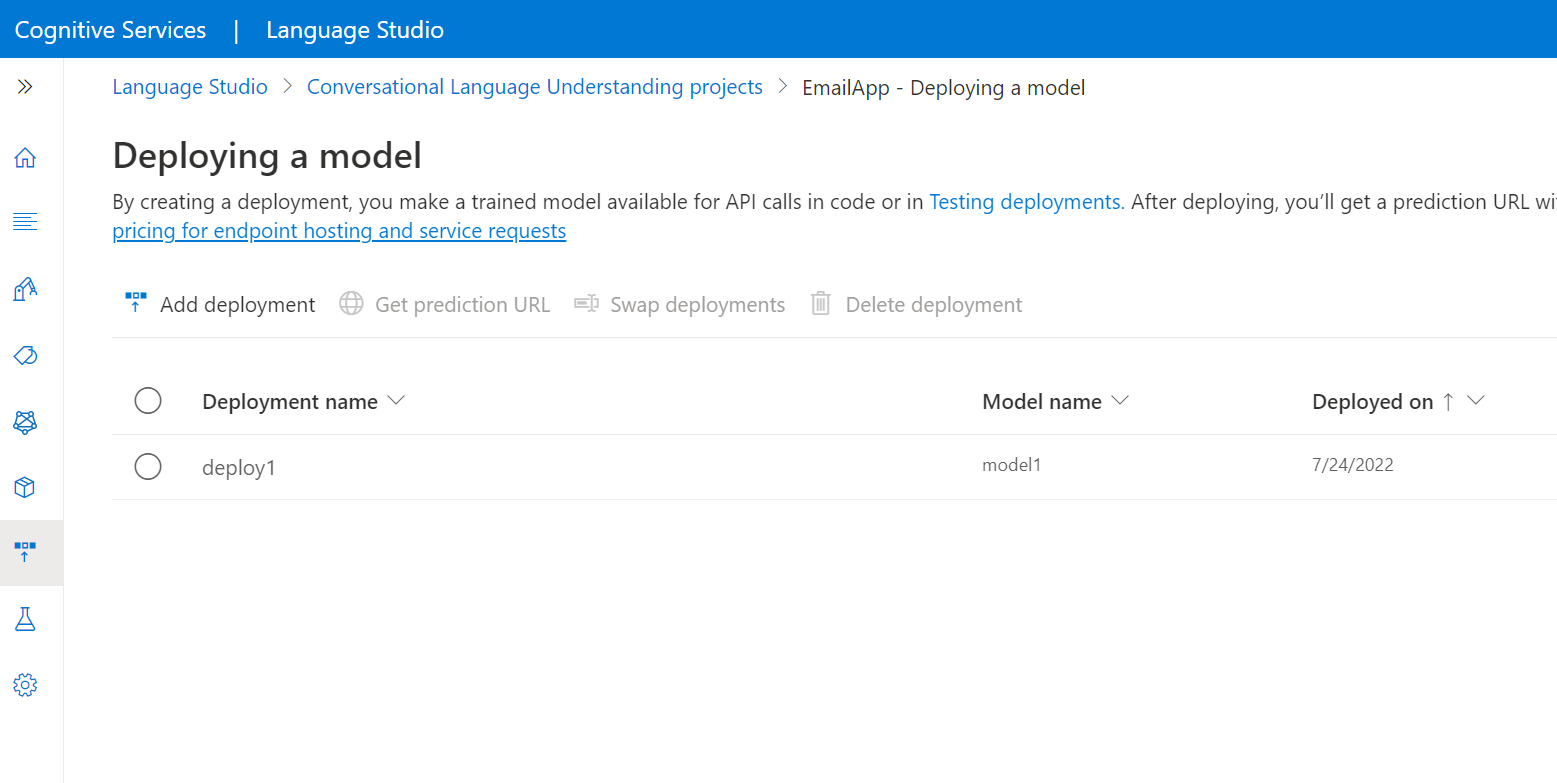
1. Select the training job ID from the list. A panel will appear where you can check the training progress, job status, and other details for this job.

**Deploy your model**

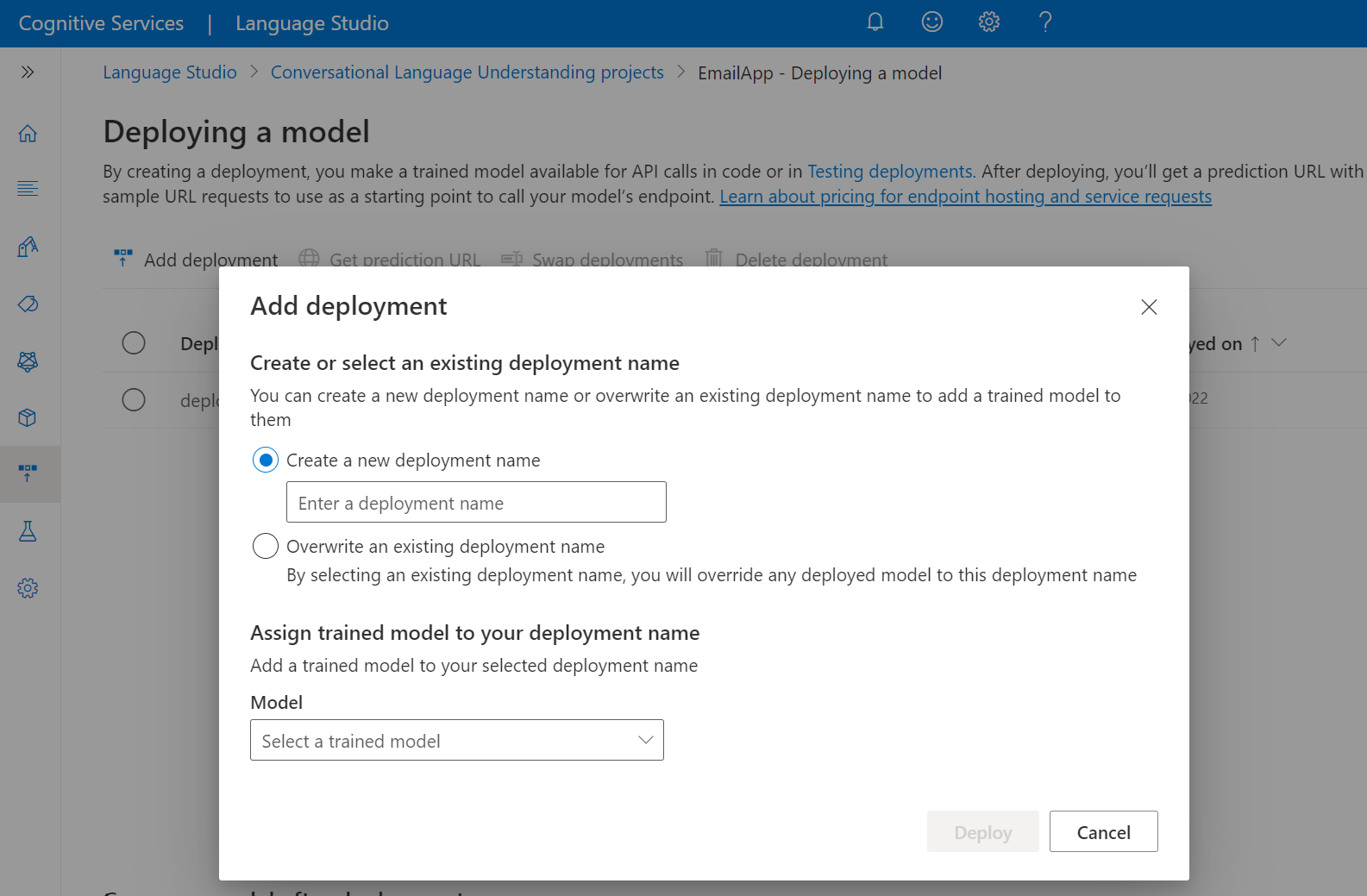
Generally after training a model you would review its evaluation details. In this quickstart, you will just deploy your model, and make it available for you to try in Language studio, or you can call the [prediction API](https://aka.ms/clu-apis).

To deploy your model from within the [Language Studio](https://aka.ms/LanguageStudio):

1. Select **Deploying a model** from the left side menu.
2. Select **Add deployment** to start the **Add deployment** wizard.

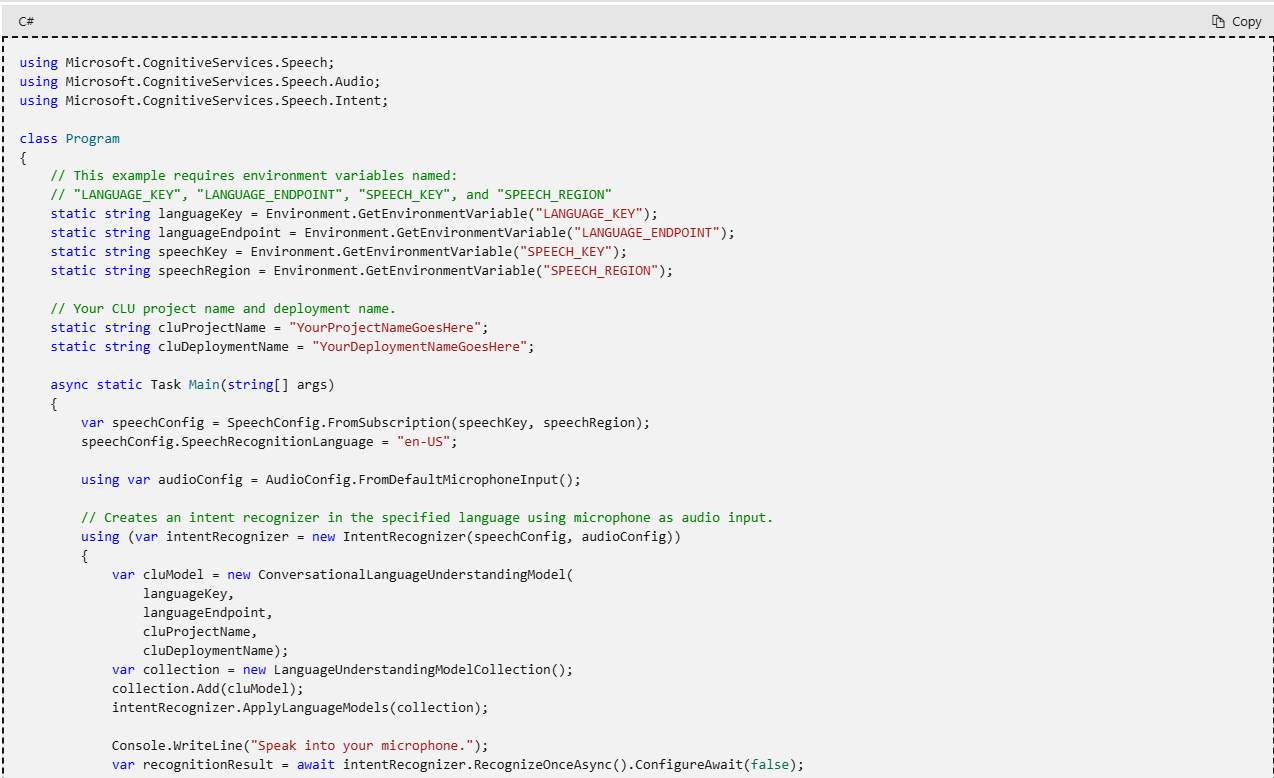
[](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/media/add-deployment-model.png#lightbox)

1. Select **Create a new deployment name** to create a new deployment and assign a trained model from the dropdown below. You can otherwise select **Overwrite an existing deployment name** to effectively replace the model that's used by an existing deployment.

[](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/media/create-deployment-job.png#lightbox)

1. Select a trained model from the **Model** dropdown.
2. Select **Deploy** to start the deployment job.
3. After deployment is successful, an expiration date will appear next to it. [Deployment expiration](https://learn.microsoft.com/en-us/azure/ai-services/language-service/concepts/model-lifecycle#expiration-timeline) is when your deployed model will be unavailable to be used for prediction, which typically happens **twelve** months after a training configuration expires.

**Recognize intents from a microphone**



A screenshot of a computer

Description automatically generated

A close-up of a white box

Description automatically generated