Develop an Azure AI Voice Live agent

1. Introduction

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/1-introduction

Introduction

Completed

- 3 minutes

Voice-enabled applications are transforming how we interact with technology, and this module guides you through building a real-time, interactive voice solutions using advanced APIs and tools. The Azure AI Voice live API is a solution enabling low-latency, high-quality speech to speech interactions for voice agents. The API is designed for developers seeking scalable and efficient voice-driven experiences as it eliminates the need to manually orchestrate multiple components.

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

- Implement the Azure AI Voice Live API to enable real-time, bidirectional communication.
- Set up and configure the agent session.
- Develop and manage event handlers to create dynamic and interactive user experiences.
- Build and deploy a Python-based web app with real-time voice interaction capabilities to Azure.

2. Explore the Azure Voice Live API

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/2-voice-live-api

Explore the Azure Voice Live API

Completed

- 5 minutes

The Voice live API enables developers to create voice-enabled applications with real-time, bidirectional communication. This unit explores its architecture, configuration, and implementation.

Key features of the Voice Live API

The Voice live API provides real-time communication using WebSocket connections. It supports advanced features such as speech recognition, text-to-speech synthesis, avatar streaming, and audio processing.

- JSON-formatted events manage conversations, audio streams, and responses.
- Events are categorized into client events (sent from client to server) and server events (sent from server to client).

Key features include:

- Real-time audio processing with support for multiple formats like PCM16 and G.711.
- Advanced voice options, including OpenAI voices and Azure custom voices.
- Avatar integration using WebRTC for video and animation.
- Built-in noise reduction and echo cancellation.

> **Note**
>
> Voice Live API is optimized for Microsoft Foundry resources. We recommend using Microsoft Foundry resources for full feature availability and best Microsoft Foundry integration experience.

For a table of supported models and regions, visit the Voice Live API overview.

Connect to the Voice Live API

The Voice live API supports two authentication methods: Microsoft Entra (keyless) and API key. Microsoft Entra uses token-based authentication for a Microsoft Foundry resource. You apply a retrieved authentication token using a `Bearer` token with the `Authorization` header.

For the recommended keyless authentication with Microsoft Entra ID, you need to assign the **Cognitive Services User** role to your user account or a managed identity. You generate a token using the Azure CLI or Azure SDKs. The token must be generated with the `https://ai.azure.com/.default` scope, or the legacy `https://cognitiveservices.azure.com/.default`

scope. Use the token in the `Authorization` header of the WebSocket connection request, with the format `Bearer <token>`.

For key access, an API key can be provided in one of two ways. You can use an `api-key` connection header on the prehandshake connection. This option isn't available in a browser environment. Or, you can use an `api-key` query string parameter on the request URI. Query string parameters are encrypted when using https/wss.

> **Note**
>
> The `api-key` connection header on the prehandshake connection isn't available in a browser environment.

WebSocket endpoint

The endpoint to use varies depending on how you want to access your resources. You can access resources through a connection to the AI Foundry project (Agent), or through a connection to the model.

- **Project connection:** The endpoint is `wss://<your-ai-foundry-resource-name>.services.ai.azure.com/voice-live/realtime?api-version=2025-10-01`
- **Model connection:** The endpoint is `wss://<your-ai-foundry-resource-name>.cognitiveservices.azure.com/voice-live/realtime?api-version=2025-10-01`.

The endpoint is the same for all models. The only difference is the required `model` query parameter, or, when using the Agent service, the `agent_id` and `project_id` parameters.

Voice Live API events

Client and server events facilitate communication and control within the Voice live API. Key client events include:

- `session.update`: Modify session configurations.
- `input_audio_buffer.append`: Add audio data to the buffer.
- `response.create`: Generate responses via model inference.

Server events provide feedback and status updates:

- `session.updated`: Confirm session configuration changes.
- `response.done`: Indicate response generation completion.
- `conversation.item.created`: Notify when a new conversation item is added.

For a full list of client/server events, visit Voice live API Reference.

> **Note**
>
> Proper handling of events ensures seamless interaction between client and server.

Configure session settings for the Voice live API

Often, the first event sent by the caller on a newly established Voice live API session is the `session.update` event. This event controls a wide set of input and output behavior. Session settings can be updated dynamically using the `session.update` event. Developers can configure voice types, modalities, turn detection, and audio formats.

Example configuration:

```
{
  "type": "session.update",
  "session": {
    "modalities": ["text", "audio"],
    "voice": {
      "type": "openai",
      "name": "alloy"
    },
    "instructions": "You are a helpful assistant. Be concise and friendly.",
    "input_audio_format": "pcm16",
    "output_audio_format": "pcm16",
    "input_audio_sampling_rate": 24000,
    "turn_detection": {
      "type": "azure_semantic_vad",
      "threshold": 0.5,
      "prefix_padding_ms": 300,
      "silence_duration_ms": 500
    },
    "temperature": 0.8,
    "max_response_output_tokens": "inf"
  }
}
```

> **Tip**
>
> Use Azure semantic VAD for intelligent turn detection and improved conversational flow.

Implement real-time audio processing with the Voice live API

Real-time audio processing is a core feature of the Voice live API. Developers can append, commit, and clear audio buffers using specific client events.

- **Append audio:** Add audio bytes to the input buffer.
- **Commit audio:** Process the audio buffer for transcription or response generation.
- **Clear audio:** Remove audio data from the buffer.

Noise reduction and echo cancellation can be configured to enhance audio quality. For example:

```
{
  "type": "session.update",
  "session": {
    "input_audio_noise_reduction": {
      "type": "azure_deep_noise_suppression"
    },
    "input_audio_echo_cancellation": {
      "type": "server_echo_cancellation"
    }
  }
}
```

> **Note**
>
> Noise reduction improves VAD accuracy and model performance by filtering input audio.

Integrate avatar streaming using the Voice live API

The Voice live API supports WebRTC-based avatar streaming for interactive applications. Developers can configure video, animation, and blendshape settings.

- Use the `session.avatar.connect` event to provide the client's SDP offer.
- Configure video resolution, bitrate, and codec settings.
- Define animation outputs such as blendshapes and visemes.

Example configuration:

```
{
  "type": "session.avatar.connect",
  "client_sdp": "<client_sdp>"
}
```

> **Tip**
>
> Use high-resolution video settings for enhanced visual quality in avatar interactions.

3. Explore the AI Voice Live client library for Python

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/3-voice-live-sdk

Explore the AI Voice Live client library for Python

Completed

- 5 minutes

The Azure AI Voice Live client library for Python provides a real-time, speech-to-speech client for Azure AI Voice Live API. It opens a WebSocket session to stream microphone audio to the service and receives server events for responsive conversations.

Important

As of version 1.0.0, this SDK is async-only. The synchronous API is deprecated to focus exclusively on async patterns. All examples and samples use async/await syntax.

In this unit, you learn how to use the SDK to implement authentication and handle events. You also see a minimal example of creating a session. For a full reference to the Voice Live package, visit the voice live Package reference.

Implement authentication

You can implement authentication with an API key or a Microsoft Entra ID token. The following code sample shows an API key implementation. It assumes environment variables are set in a `.env` file, or directly in your environment.

```
 import asyncio
from azure.core.credentials import AzureKeyCredential
from azure.ai.voicelive import connect

async def main():
    async with connect(
        endpoint="your-endpoint",
        credential=AzureKeyCredential("your-api-key"),
```

```
        model="gpt-4o"
    ) as connection:
        # Your async code here
        pass

asyncio.run(main())
```

For production applications, Microsoft Entra authentication is recommended. The following code sample shows implementing the `DefaultAzureCredential` for authentication:

```
 import asyncio
from azure.identity.aio import DefaultAzureCredential
from azure.ai.voicelive import connect

async def main():
    credential = DefaultAzureCredential()

    async with connect(
        endpoint="your-endpoint",
        credential=credential,
        model="gpt-4o"
    ) as connection:
        # Your async code here
        pass

asyncio.run(main())
```

Handling events

Proper handling of events ensures a more seamless interaction between the client and agent. For example, when handling a user interrupting the voice agent you need to cancel agent audio playback immediately in the client. If you don't, the client continues to play the last agent response until the interrupt is processed in the API - resulting in the agent "talking over" the user.

The following code sample shows some basic event handling:

```
 async for event in connection:
    if event.type == ServerEventType.SESSION_UPDATED:
        print(f"Session ready: {event.session.id}")
        # Start audio capture

    elif event.type == ServerEventType.INPUT_AUDIO_BUFFER_SPEECH_STARTED:
        print("User started speaking")
        # Stop playback and cancel any current response

    elif event.type == ServerEventType.RESPONSE_AUDIO_DELTA:
```

```
        # Play the audio chunk
        audio_bytes = event.delta

    elif event.type == ServerEventType.ERROR:
        print(f"Error: {event.error.message}")
```

Minimal example

The following code sample shows authenticating to the API and configuring the session.

```
 import asyncio
from azure.core.credentials import AzureKeyCredential
from azure.ai.voicelive.aio import connect
from azure.ai.voicelive.models import (
    RequestSession, Modality, InputAudioFormat, OutputAudioFormat, ServerVad, ServerEventType
)

API_KEY = "your-api-key"
ENDPOINT = "your-endpoint"
MODEL = "gpt-4o"

async def main():
    async with connect(
        endpoint=ENDPOINT,
        credential=AzureKeyCredential(API_KEY),
        model=MODEL,
    ) as conn:
        session = RequestSession(
            modalities=[Modality.TEXT, Modality.AUDIO],
            instructions="You are a helpful assistant.",
            input_audio_format=InputAudioFormat.PCM16,
            output_audio_format=OutputAudioFormat.PCM16,
            turn_detection=ServerVad(
                threshold=0.5,
                prefix_padding_ms=300,
                silence_duration_ms=500
            ),
        )
        await conn.session.update(session=session)

        # Process events
        async for evt in conn:
            print(f"Event: {evt.type}")
            if evt.type == ServerEventType.RESPONSE_DONE:
                break

asyncio.run(main())
```

4. Exercise - Develop an Azure AI Voice Live agent

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/4-exercise-develop-agent

Exercise - Develop an Azure AI Voice Live agent

Completed

- 30 minutes

In this exercise, you complete a Flask-based Python web app based that enables real-time voice interactions with an agent. You add the code to initialize the session, and handle session events. You use a deployment script that: deploys the AI model; creates an image of the app in Azure Container Registry (ACR) using ACR tasks; and then creates an Azure App Service instance that pulls the image. To test the app, you need an audio device with microphone and speaker capabilities.

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

- Azure VoiceLive client library for .NET

Tasks performed in this exercise:

- Download the base files for the app
- Add code to complete the web app
- Review the overall code base
- Update and run the deployment script
- View and test the application

This exercise takes approximately **30** minutes to complete.

Before you start

To complete the exercise, you need:

- An Azure subscription. If you don't already have one, you can sign up for one https://azure.microsoft.com/.

- An audio device with microphone and speaker capabilities.

Get started

Select the **Launch Exercise** button to open the exercise instructions in a new browser window. When you're finished with the exercise, return here to:

- Complete the module
- Earn a badge for completing this module

Launch Exercise

5. Module assessment

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/5-knowledge-check

Module assessment

Completed

- 5 minutes

6. Summary

https://learn.microsoft.com/en-us/training/modules/develop-voice-live-agent/6-summary

Summary

Completed

- 3 minutes

In this module, you learned about the Voice live API's features, including WebSocket connections, speech recognition, text-to-speech synthesis, and avatar streaming. You also explored Azure AI Voice Live for creating real-time speech-to-speech applications using Python, including setting up the client library and managing sessions. Additionally, you learned how to implement event handlers in Python for dynamic responses and real-time audio processing. Finally, you developed a Python-based web application using Flask, integrated it with Azure resources, and tested the application.

Additional reading

- What is the Speech service?
- How to customize voice live input and output