Build intelligent apps on .NET using Azure Communication Services

Milan Kaur | LinkedIn @milanKaurInTech
Product Manager, Azure Communication Services

Agenda

Logical layers of an intelligent app

Azure Communication channels

WhatsApp channel for an AI bot

Code walkthrough

PSTN channel for an AI bot

Code walkthrough

Logical layers of an intelligent app

Source

Source system where documents are stored

Ingest

Extract and load large volume of data to Store layer

It's possible to use 3rd party 'connector' to ingest data from source and build index in Store layer

Store

Store extracted data and vector data to provides relevant data for users

Store operational data

Processing

Compute data and run real-time and batch requests

AI for text understanding, summarization and embeddings

Serve

User facing layer for example, Web App or Chat

https://aka.ms/raghack/streams

Monitoring

Monitor users' prompt and response

Identity & Access

Access controls on the service and contents

Azure Communication Services

A fully managed communication platform that enables developers and organizations to securely build communications features and connected user experiences across applications and devices.

Core communication services

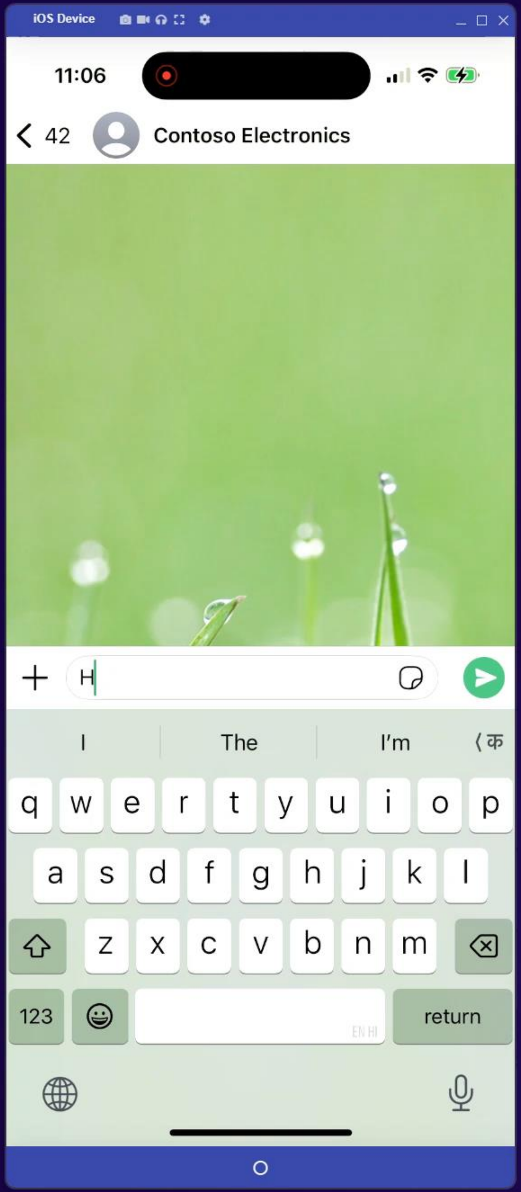
REST API | Web, Mobile, Desktop SDKs

Voice & Video Calling	Chat	Telephony	SMS	Email	WhatsApp

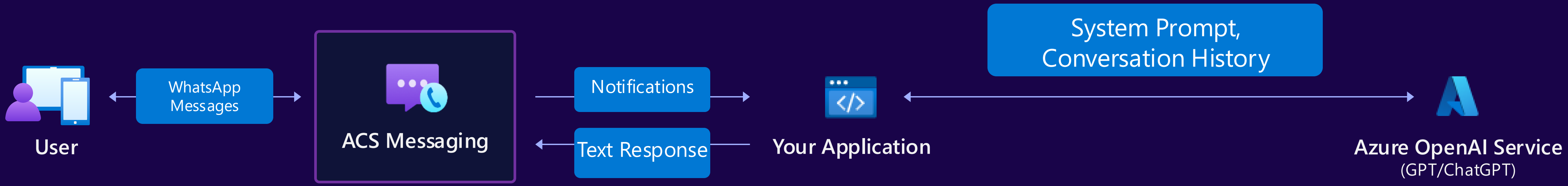
Demo:
WhatsApp customer service bot

Azure Open AI Service

Azure Communication Service


iOS Device
11:06
42
Contoso Electronics
H
I
The
I'm
return
123

Architecture diagram



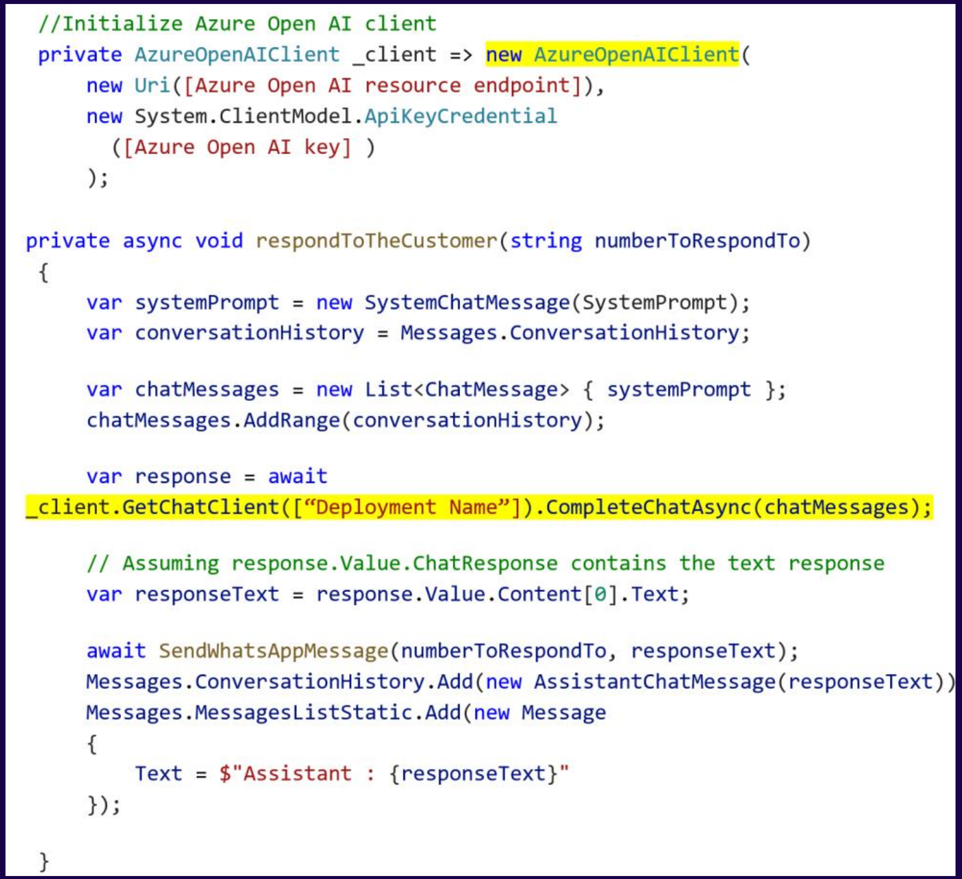
Steps for adding intelligence to your app

Step 1: Create an Azure Open AI resource on the Azure portal

Step 2: Create an AI model deployment in Azure AI Studio

Step 3: Write and test a system prompt in Azure AI Studio

Step 4: Pass the system prompt and conversation history to a chat completion API in code



```
 //Initialize Azure Open AI client
 private AzureOpenAIClient _client => new AzureOpenAIClient(
     new Uri([Azure Open AI resource endpoint]),
     new System.ClientModel.ApiKeyCredential
       ([Azure Open AI key] )
     );

private async void respondToTheCustomer(string numberToRespondTo)
 {
     var systemPrompt = new SystemChatMessage(SystemPrompt);
     var conversationHistory = Messages.ConversationHistory;

     var chatMessages = new List<ChatMessage> { systemPrompt };
     chatMessages.AddRange(conversationHistory);

     var response = await
_client.GetChatClient([“Deployment Name”]).CompleteChatAsync(chatMessages);

     // Assuming response.Value.ChatResponse contains the text response
     var responseText = response.Value.Content[0].Text;

     await SendWhatsAppMessage(numberToRespondTo, responseText);
     Messages.ConversationHistory.Add(new AssistantChatMessage(responseText))
     Messages.MessagesListStatic.Add(new Message
     {
         Text = $"Assistant : {responseText}"
     });

 }
```

Steps for adding WhatsApp channel

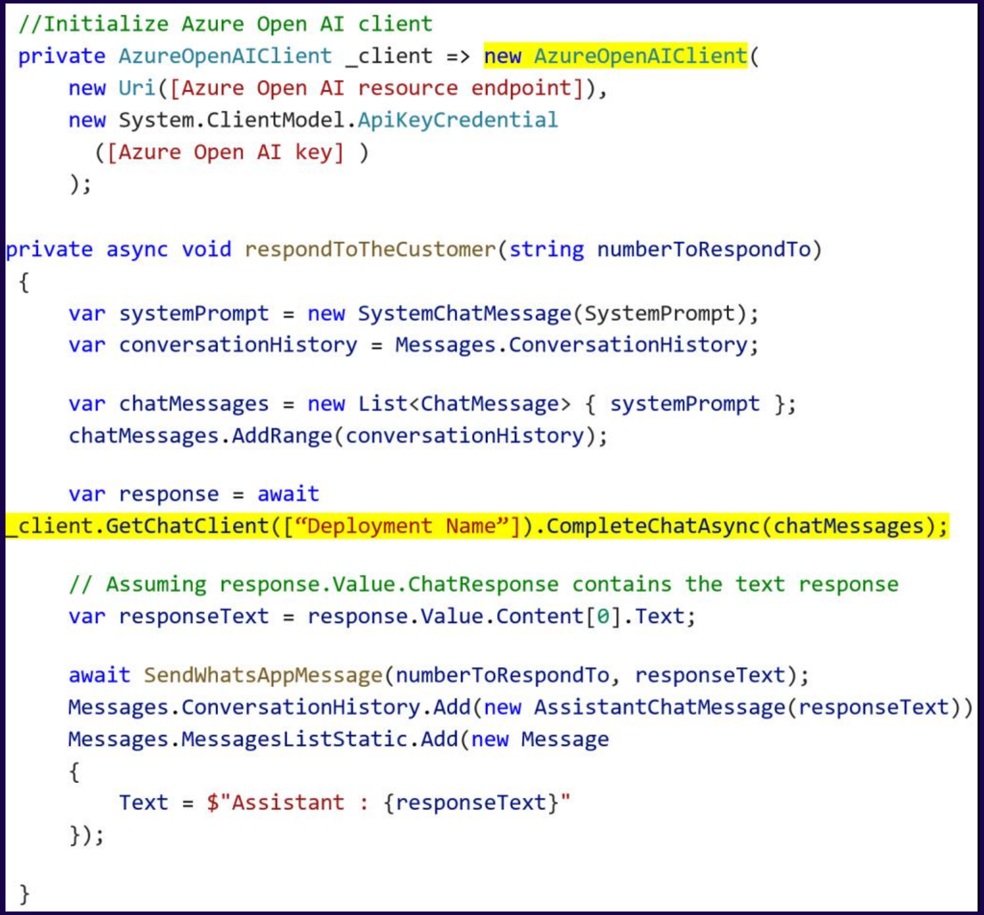
Step 1: **Create an ACS resource**

Step 2: **Create a WhatsApp business account**

Step 3: **Connect your number or an ACS number with WhatsApp account**

Step 4: **Add code to handle events and sending WhatsApp message**

Step 5: **Register your local or server url in ACS event grid for receiving WhatsApp messages**



```
//Initialize Azure Open AI client
private AzureOpenAIClient _client => new AzureOpenAIClient(
    new Uri([Azure Open AI resource endpoint]),
    new System.ClientModel.ApiKeyCredential
      ([Azure Open AI key] )
    );

private async void respondToTheCustomer(string numberToRespondTo)
{
    var systemPrompt = new SystemChatMessage(SystemPrompt);
    var conversationHistory = Messages.ConversationHistory;

    var chatMessages = new List<ChatMessage> { systemPrompt };
    chatMessages.AddRange(conversationHistory);

    var response = await
_client.GetChatClient([“Deployment Name”]).CompleteChatAsync(chatMessages);

    // Assuming response.Value.ChatResponse contains the text response
    var responseText = response.Value.Content[0].Text;

    await SendWhatsAppMessage(numberToRespondTo, responseText);
    Messages.ConversationHistory.Add(new AssistantChatMessage(responseText))
    Messages.MessagesListStatic.Add(new Message
    {
        Text = $"Assistant : {responseText}"
    });

}
```

Demo: Phone calling (PSTN) bot

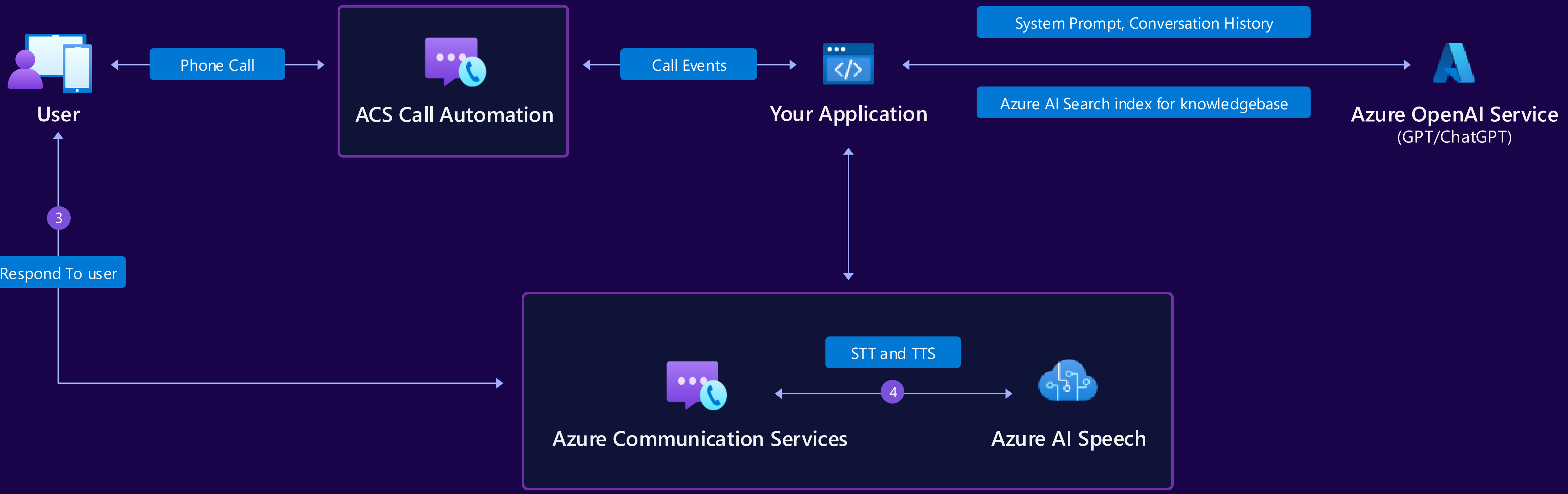
Azure Open AI Service

Azure Communication Service

Azure AI Search

Architecture diagram

https://aka.ms/AcsIgnite2024



Steps for adding intelligence to your app

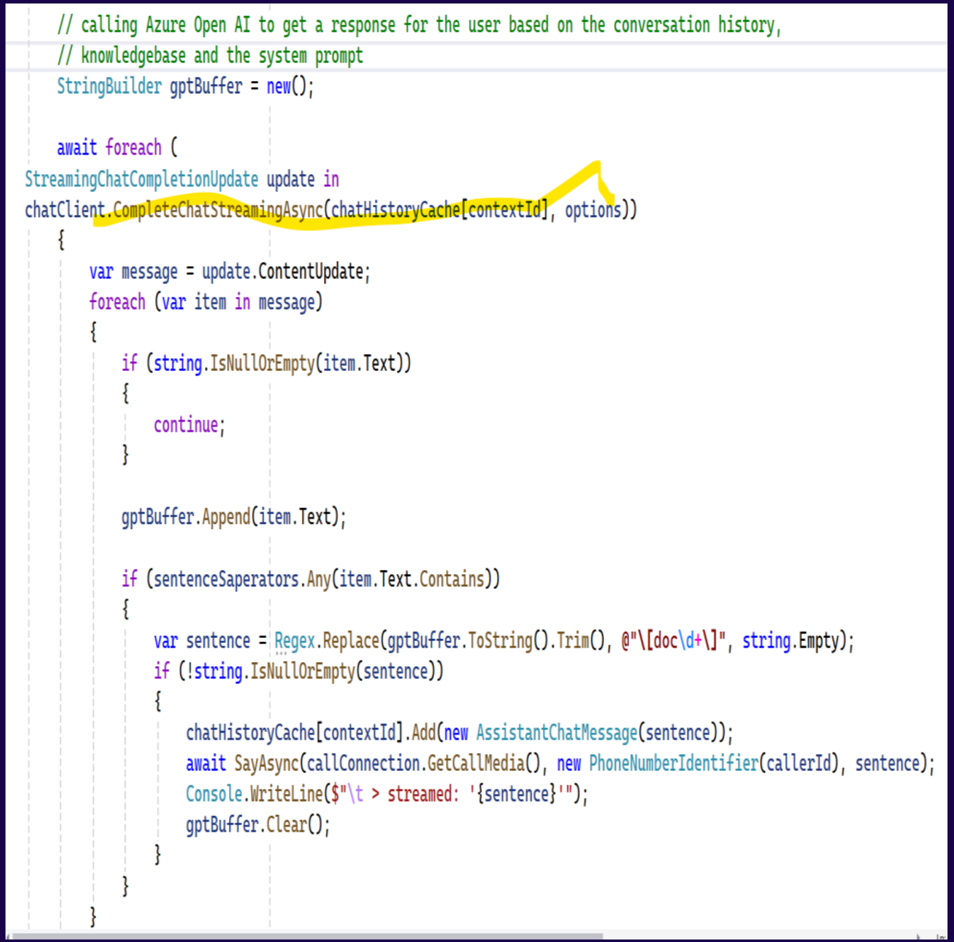
Step 1: Create an Azure Open AI resource on the Azure portal

Step 2: Create an AI model deployment in Azure Open AI Studio

Step 3: Add your data using Azure OpenAI Studio

Step 4: Test system prompt with your data in Azure Open AI Studio

Step 4: Pass the system prompt and conversation history to a streaming chat completion API in code



```
    // calling Azure Open AI to get a response for the user based on the conversation history,
    // knowledgebase and the system prompt
    StringBuilder gptBuffer = new();

    await foreach (
StreamingChatCompletionUpdate update in
chatClient.CompleteChatStreamingAsync(chatHistoryCache[contextId], options))
    {
        var message = update.ContentUpdate;
        foreach (var item in message)
        {
            if (string.IsNullOrEmpty(item.Text))
            {
                continue;
            }

            gptBuffer.Append(item.Text);

            if (sentenceSaperators.Any(item.Text.Contains))
            {
                var sentence = Regex.Replace(gptBuffer.ToString().Trim(), @"\[doc\d+\]", string.Empty);
                if (!string.IsNullOrEmpty(sentence))
                {
                    chatHistoryCache[contextId].Add(new AssistantChatMessage(sentence));
                    await SayAsync(callConnection.GetCallMedia(), new PhoneNumberIdentifier(callerId), sentence);
                    Console.WriteLine($"\t > streamed: '{sentence}'");
                    gptBuffer.Clear();
                }
            }
        }
```

Steps for adding telephony channel

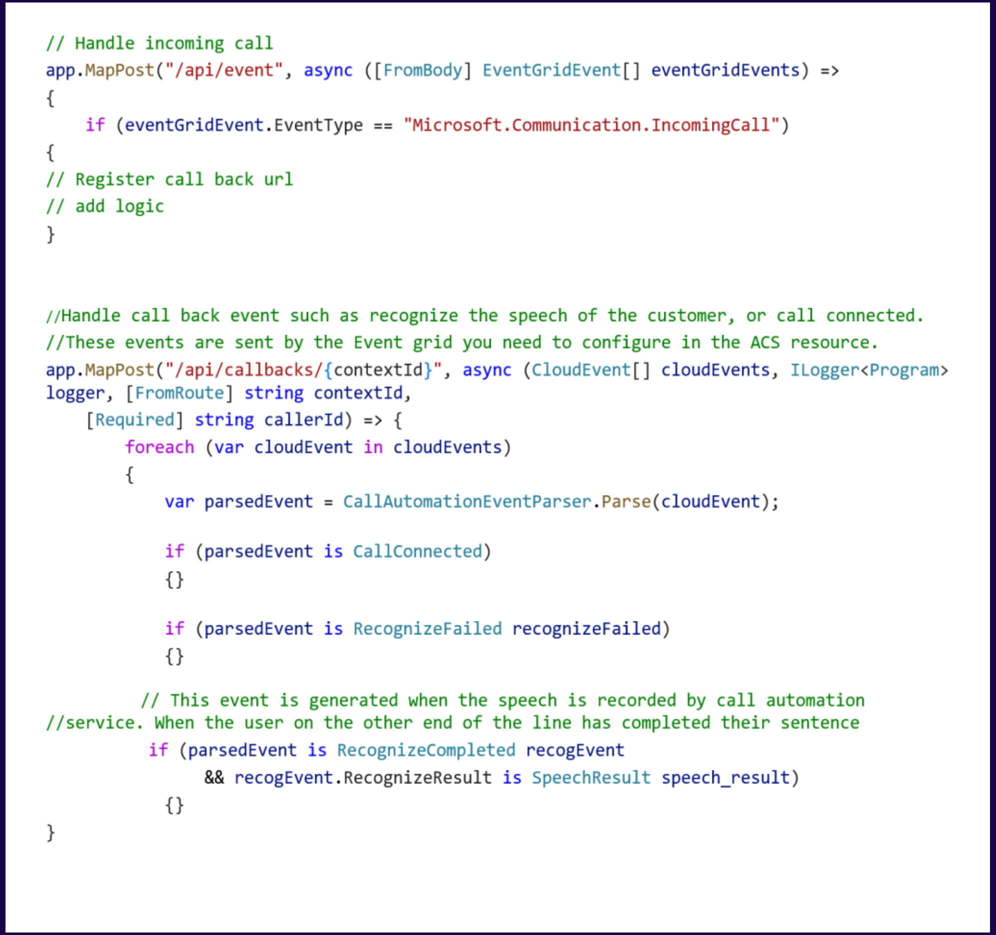
Step 1: Create an ACS resource and get a number

Step 2: Add ACS Call Automation package.

Step 3: Create an endpoint for receiving incoming calls.

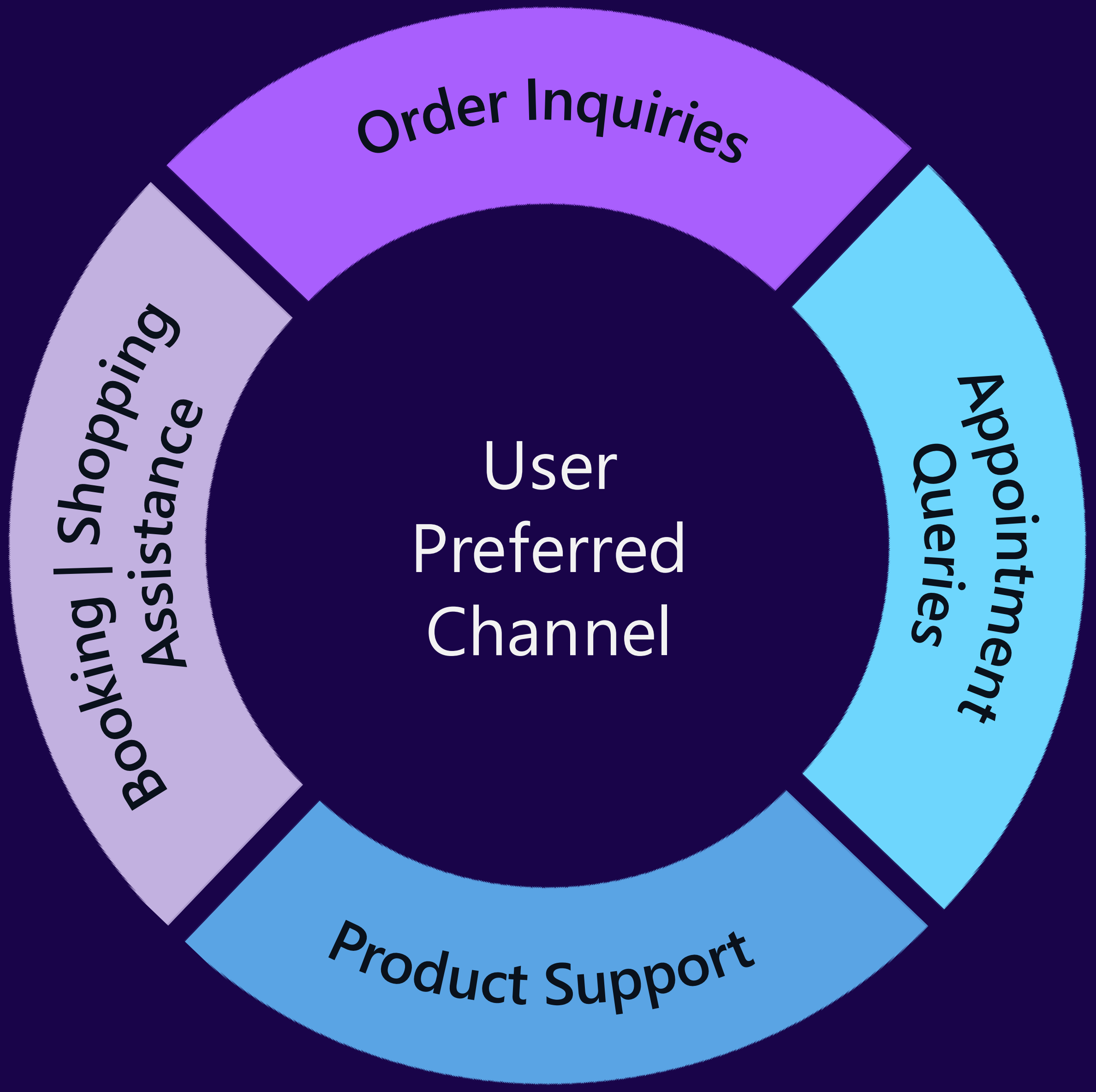
Step 4: Create an endpoint for receiving call back events.

Step 5: Deploy/run your app and register the call back url with ACS event grid.



```
// Handle incoming call
app.MapPost("/api/event", async ([FromBody] EventGridEvent[] eventGridEvents) =>
{
    if (eventGridEvent.EventType == "Microsoft.Communication.IncomingCall")
{
// Register call back url
// add logic
}


//Handle call back event such as recognize the speech of the customer, or call connected.
//These events are sent by the Event grid you need to configure in the ACS resource.
app.MapPost("/api/callbacks/{contextId}", async (CloudEvent[] cloudEvents, ILogger<Program>
logger, [FromRoute] string contextId,
    [Required] string callerId) => {
        foreach (var cloudEvent in cloudEvents)
        {
            var parsedEvent = CallAutomationEventParser.Parse(cloudEvent);

            if (parsedEvent is CallConnected)
            {}

            if (parsedEvent is RecognizeFailed recognizeFailed)
            {}

          // This event is generated when the speech is recorded by call automation
//service. When the user on the other end of the line has completed their sentence
           if (parsedEvent is RecognizeCompleted recogEvent
                && recogEvent.RecognizeResult is SpeechResult speech_result)
            {}
}
```

Scenarios


Order Inquiries
Appointment Queries
Product Support
Booking | Shopping Assistance
User Preferred Channel

Resources

aka.ms/cakeShopSample
Phone call sample

aka.ms/whatsAppRag
WhatsApp bot sample

aka.ms/acs-ai-samples
More ACS & AI samples

aka.ms/whatsappUsagePricing
WhatsApp usage pricing

Aka.ms/ragHack/streams
Content on Retrieval Augmented Generation

aka.ms/AcsIgnite2024
ACS announcements at Ignite

Get .NET 9

Download .NET 9

aka.ms/get-dotnet-9

Thank you