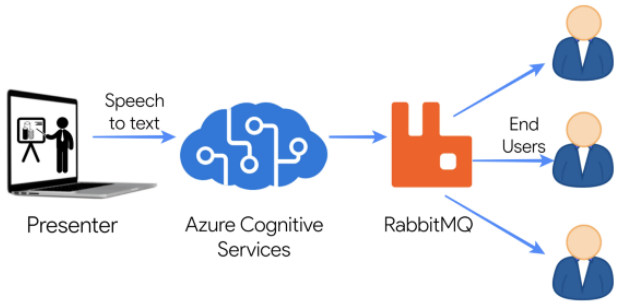
# Utilising Azure cognitive services with Javascript (Vue.JS) to enable real-time translations

> Problem statement: *Back in 2021, when all boardroom meetings were shifted to online meetings, I came across a business use case during Covid-19 wherein my co-workers were unable to understand the meetings and webinars due to a language barrier. There were a lot of newly hired people around that time who were working remotely. All of them understood English but some were unable to grasp words quickly which was causing communication gaps in the team. Third-party translation services were not reliable and would not be cost-effective.*

> Solution: *I was talking to a co-worker when he casually said "Online video streaming services are so much better to watch movies. At least they have captions to make the video understandable." He said as a joke. However, to me, it was a great idea! To solve the problem related to online meetings, I decided to create a solution which would translate the meetings in real-time. We were using an internal platform to conduct meetings. Hence, integrating it with the existing setup using a Vue.JS application was possible. I used Azure Cognitive Service to translate speech-to-text in real-time which was transmitted to the end users by an API call to RabbitMQ. The end users were connected to the RabbitMQ server which was used to establish a connection.*

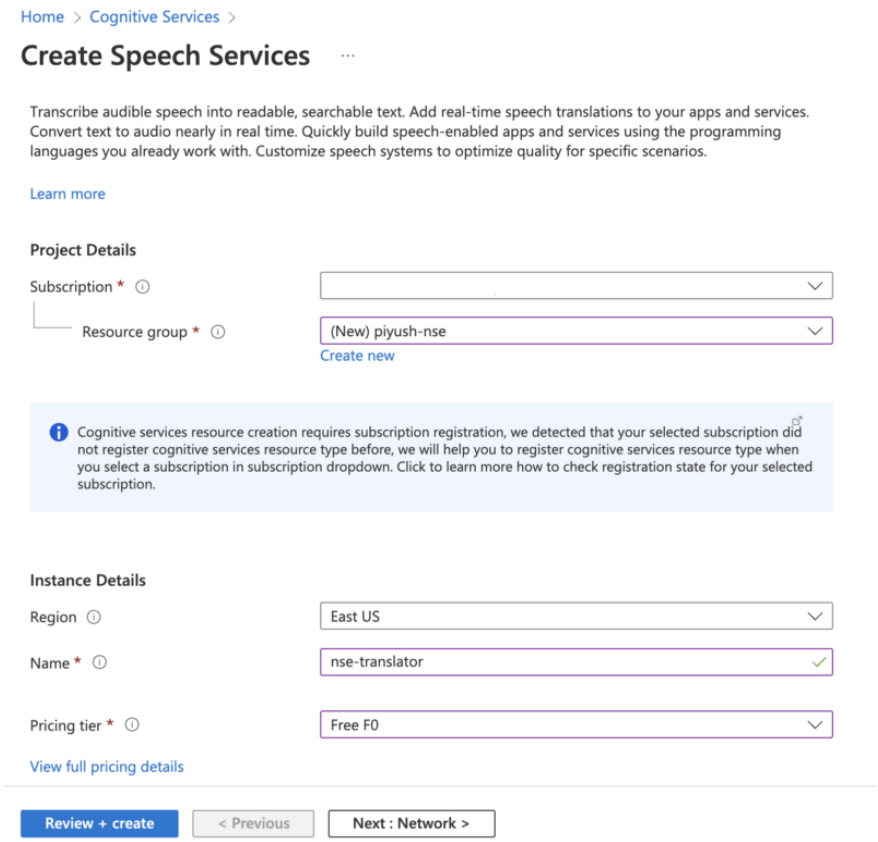
> Solution Architecture:




Above is the architecture diagram of the implemented solution. The presenter's speech is converted to text. The users are connected to the RabbitMQ server which places requests for the translated text and gives the response to the end users.

## Implementation:

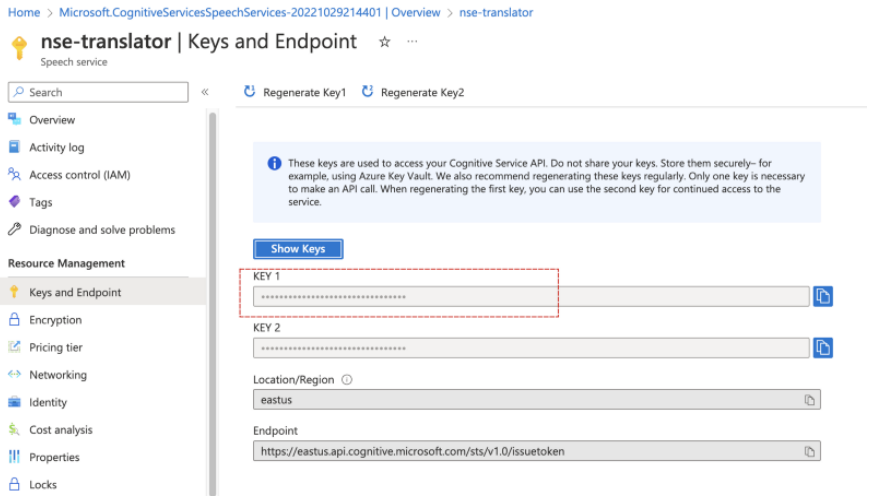
For using Azure Cognitive Services, I created a cognitive service account from the Azure portal



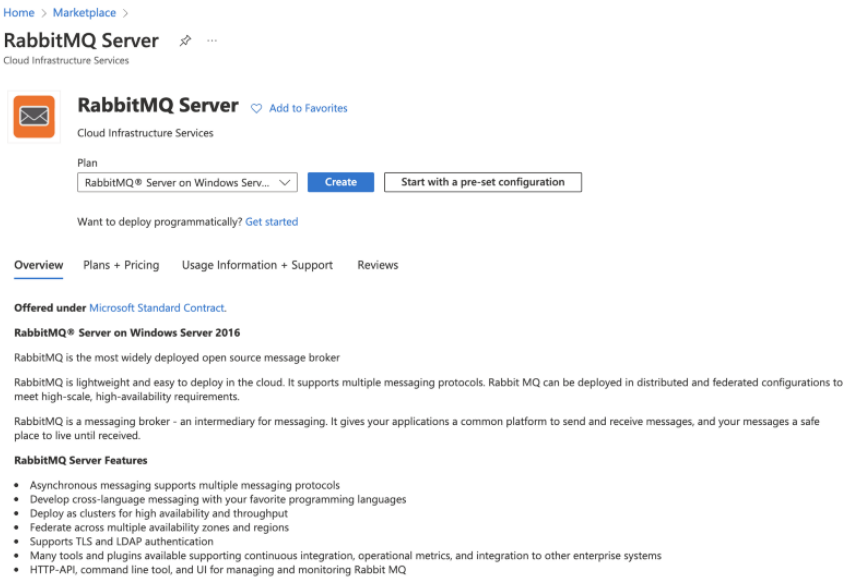

Creating Speech Service

I used the free tier to test out the application and analyse the cost.

Firstly, I installed the cognitive services software development kit using the npm install command. Then I retrieved the required key for the service from the portal to use in my code.




Then I created the RabbitMQ server from the Azure Marketplace




To get the Azure Cognitive Services SDK I used the below command in my Vue.JS application. I used Visual Studio Code to write my application code as it helps with indentation.

```
npm install microsoft-cognitiveservices-speech-sdk
```

Cognitive Service SDK has support across multiple programming languages:

| Programming language | Reference | Platform support |
|---|---|---|
| C# [1] | .NET | Windows, Linux, macOS, Mono, Xamarin.iOS, Xamarin.Mac, Xamarin.Android, UWP, Unity |
| C++ [2] | C++ | Windows, Linux, macOS |
| Go | Go | Linux |
| Java | Java | Android, Windows, Linux, macOS |
| JavaScript | JavaScript | Browser, Node.js |
| Objective-C | Objective-C | iOS, macOS |
| Python | Python | Windows, Linux, macOS |
| Swift | Objective-C [3] | iOS, macOS |

Speech-to-text support

The code for my Vue.JS application looked something like this.

```
const acfg = AudioConfig.fromDefaultMicrophoneInput()

const nse_scfg =
SpeechTranslationConfig.fromSubscription(options.key,
options.region)

speechConfig.speechRecognitionLanguage =
options.fromLanguage

for (const case of options.toLanguages) {
  speechConfig.addTargetLanguage(case)
}

this._recognizer = new TranslationRecognizer(speechConfig,
audioConfig)

this._recognizer.recognizing = this._recognizer.recognized =
recognizerCallback.bind(this)

this._recognizer.startContinuousRecognitionAsync()
```

Additionally, I utilised the below code to establish a connection with the RabbitMQ server.

```
var amqp = require('amqplib/callback_api');
module.exports = rabbitMQMessages;
function rabbitMQMessages(address, callback)
{
amqp.connect(address, function amqpConnectCallback(err,
conn)
{    if(err)
     {
        return callback(err);
     }
     //create a channel
     conn.createChannel(function(err, ch)
     {
        if(err)
        {
           return callback(err);
        }

     }
     ch.bindQueue(q.queue, 'messages', '');
     var altern =
     {
         emitMessage: emitMessage,
         onMessageReceived: onMessageReceived
     };
}
}
```

Once set up, this application was run on the meeting presenter's web browser.

> Working of the application*:*

> *Steps:*

1. The function would execute to pick audio from the presenter's microphone.
2. This audio would be passed to the cognitive service API in real time to convert it to text.
3. The API would convert it to text based on the language selected (It was English in my case.)
4. The users would connect to the RabbitMQ server using the available API.
5. Users could request translation in their desired language. However, in my case, it was only limited to English.

6. These calls would then be filtered based on the request and then be provided with the requested speech-to-text translation
7. If there was an error in receiving the translation, there was a retry logic implemented, which would try until the result was a successful request/response.
8. The end user would see the translated text on their screen while the meeting was going on in real-time.
9. The entire operation was initiated by the meeting presenter and would end with his presentation.
10. At the end the attendees would receive a feedback dialogue, asking them about their experience, which was stored in a google form and then converted to an excel sheet to use for future tweaks and improvements.

> Challenges in implementing the solution:

- Several challenges were faced in designing the application's architecture and establishing its connection with various services.
- It for difficult for me to choose between RabbitMQ and SignalR. However, I was familiar with RabbitMQ as I had used it in previous use cases. Hence, I decided to go ahead with RabbitMQ.
- It took me some time to figure out a latency issue that was being caused in the caption service which was due to RabbitMQ. I fixed it later on with a new build.

> Business Benefit:

- Communication across the team became smooth and efficient with this well-architectured solution.
- The cost of third-party translation services which offer a subscription-based plan of around $2000 per month (for the entire team) was completely eliminated
- The speech-to-text translation was latency-free and effective
- Thanks to the positive feedback from the team members, the same solution was implemented for other teams with similar issues.

> Best practices:

I learnt the hard way and probably you should not.

- Always keep your confidential keys in a secure location. I later used Azure Key Vault to store and retrieve the Cognitive Services Key.
- Use fine-grained roles whenever working with services.
- Never provide owner-level permissions unless there is a compulsion for it.
- System-managed identity and user-managed identity are go-to options whenever possible.

> Where to start?

If you are new to cloud-native applications, you can start exploring with a free Azure subscription and a free code editor.

Azure Account: Get a trial subscription here. It is beginner friendly and has all the capabilities that an organization can use to boost their business.
https://azure.microsoft.com/en-us/free/

Visual Studio Code:
https://code.visualstudio.com/

> References and motivation:

https://github.com/Djangoum/Vue-3-rabbitmq/find/master

https://anthonychu.ca/post/realtime-captioning-translation-cognitive-services-signalr-azure-functions/

https://www.npmjs.com/package/microsoft-cognitiveservices-speech-sdk

https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/speech-sdk

https://azuremarketplace.microsoft.com/en/marketplace/apps/cloud-infrastructure-services.rabbitmq-2019?tab=overview

Blog published by
Mr Piyush Dolai

Follow me on LinkedIN

https://www.linkedin.com/in/piyush-dolai