Title: QnA Bot using Azure Language Services

Intern Name: Tejasvi Avhad
Internship Domain: Cloud, Infrastructure, and Security

Contents

1. Introduction
2. Problem Statement
3. Project Objectives
4. Tools & Technologies Used
5. System Architecture
6. Implementation Details
7. Challenges Faced
8. Conclusion

1. Introduction

During my internship in the Cloud Infrastructure and Security domain, I worked on building a QnA Bot using Azure Language Services. The main goal was to help users query documents and get relevant answers automatically. Instead of using AI-generated answers, this bot works with a document-based knowledge base uploaded by the user, providing accurate and reliable responses. The solution was developed using Node.js, hosted on Azure App Service, and tested using Bot Framework Emulator. Due to subscription constraints, the bot could not be deployed on Microsoft Teams, and this is honestly reflected in the report.

2. Problem Statement

Develop an QnA Bot, particularly tailored for This bot, leveraging the advanced language processing capabilities of Azure AI Search, are designed to significantly enhance information access, encourage standardization through customizable features, optimize time efficiency, and ensure compliance. This strategic initiative revolves around empowering people with specialized bot to seamlessly mine knowledge within the document store. Deploy a basic bot using language service with AI search capabilities. Use private endpoints to keep the infrastrcuture secure. Use Azure Application Gateway to expose the bot on public internet while keeping the public access disabled.

3. Project Objectives

- Build a bot that answers questions from uploaded documents.

- Use Azure Language Service's QnA project to extract answers.
- Deploy the bot on Azure App Service.
- Ensure it works with Bot Framework Emulator for testing.

4. Tools & Technologies Used

Tool/Service	Purpose
Azure Language Studio	Create and manage QnA projects
Azure App Service	Host and run the bot
Bot Framework SDK	Build conversational logic
Bot Framework Emulator	Test the bot locally
Node.js + Restify	Build backend logic
Hoppscotch	Manual API Testing
Visual Studio Code	Code editor

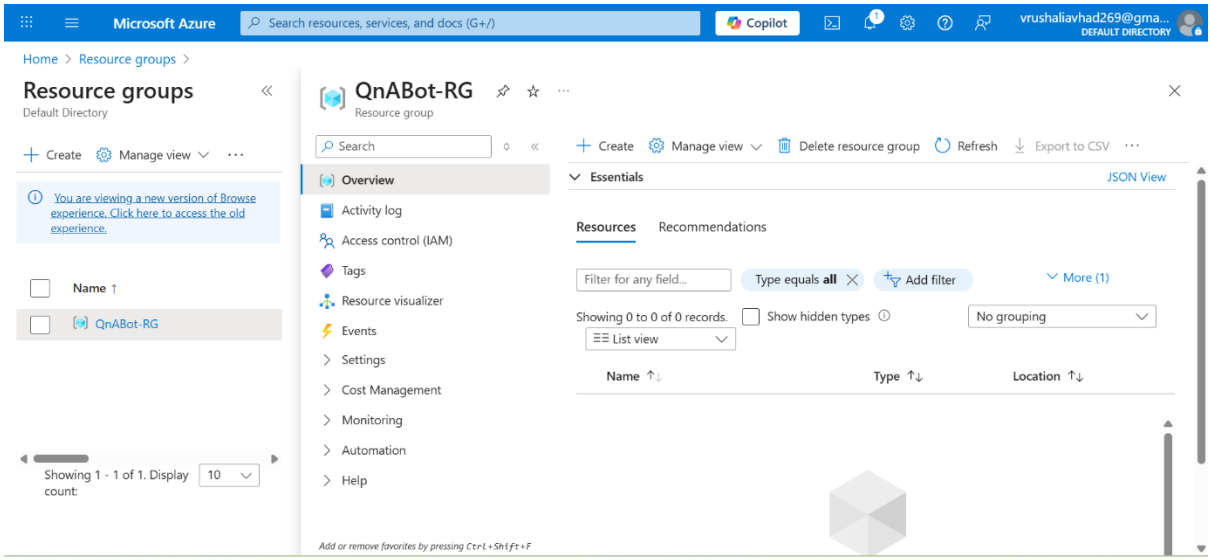
5. System Architecture

1. User asks a question in Emulator.
2. Bot captures and forwards it to Azure QnA endpoint.
3. Azure returns the most relevant answer from the document.
4. Bot displays it back to the user.

6. Implementation Details

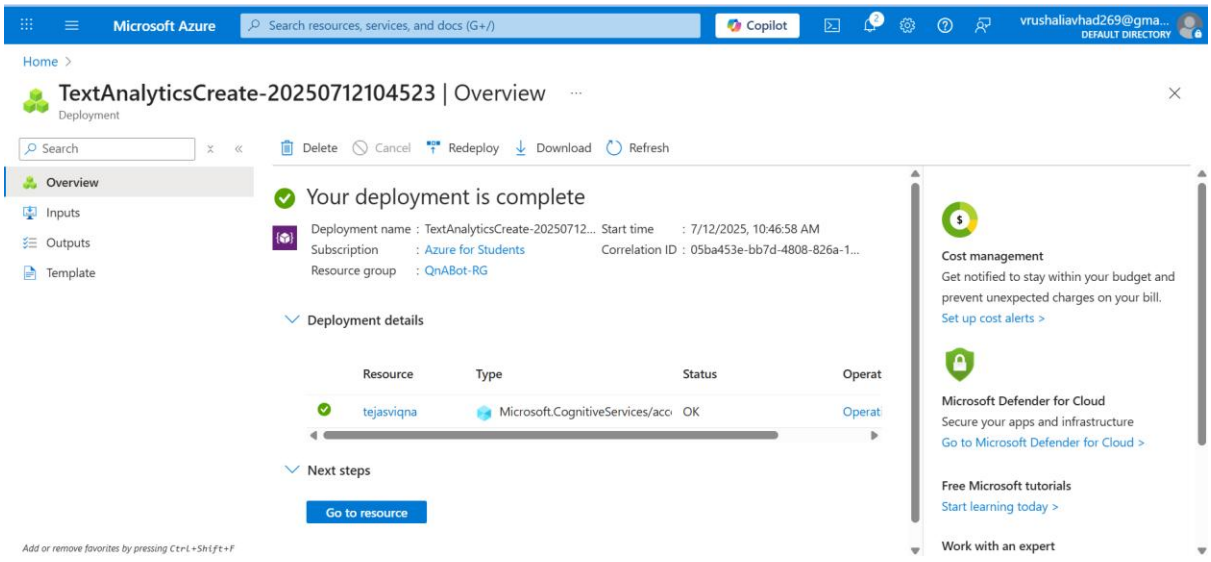
Step 1: Create a Resource Group

1. Go to https://portal.azure.com
2. Search for Resource Groups > Create
3. Fill:
    - Name: QnABot-RG
    - Region: East US (or a free-tier supported region)
4. Click Review + Create → then Create



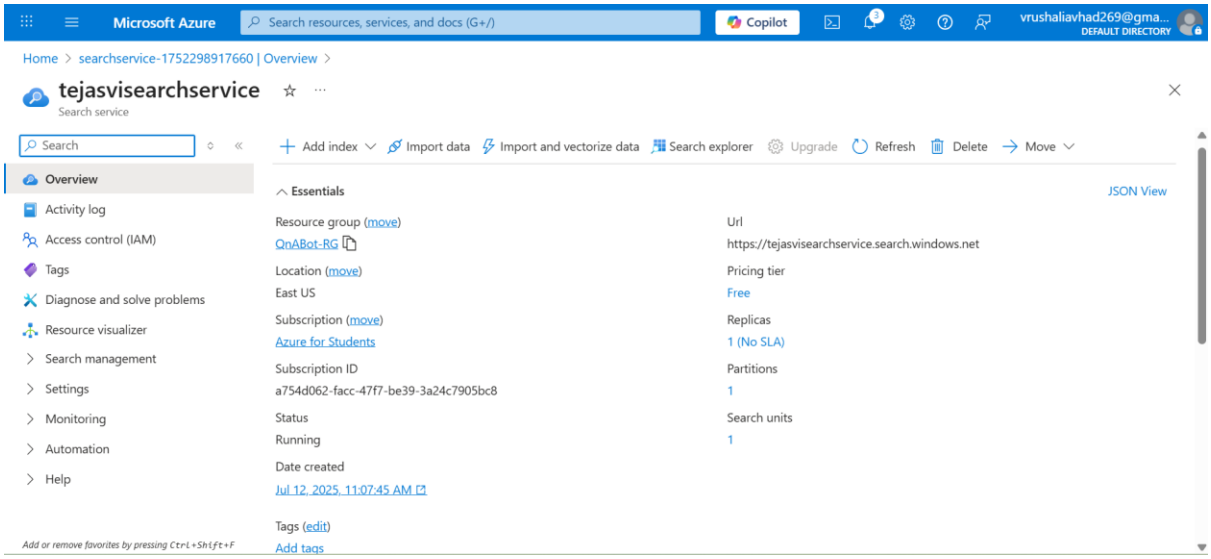
Step 2: Create Azure Language Service

1. Search “Language Services” → Click Create
2. Fill:
    - Name: tejasviqna
    - Region: Same as RG
    - Pricing Tier: F0 (Free)
    - Resource Group: QnABot-RG
3. Click Review + Create → then Create



Step 3: Create Azure Ai Search

1. Go to Azure Portal
2. Click Create a resource
3. Search for Azure AI Search
4. Select and click Create
5. Fill the form:
    - Name: tejasvisearchservice
    - Region: Same as your Language Service
    - Pricing Tier: Free (F)
    - Resource Group: QnABot-RG
6. Click Review + Create → then Create



Step 4: Open Language Studio

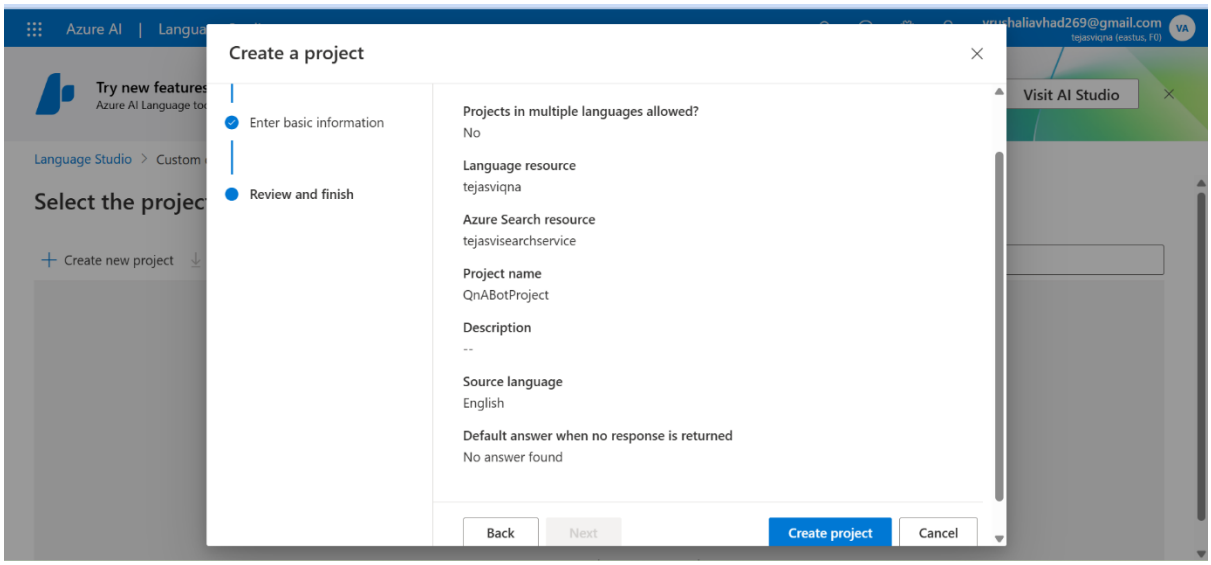
Go to: https://language.azure.com

1. Sign in with your Azure account
2. Click “Custom Question Answering”
3. Select your Azure Language resource and Azure AI Search
4. Click Continue

Step 5: Create a New QnA Project

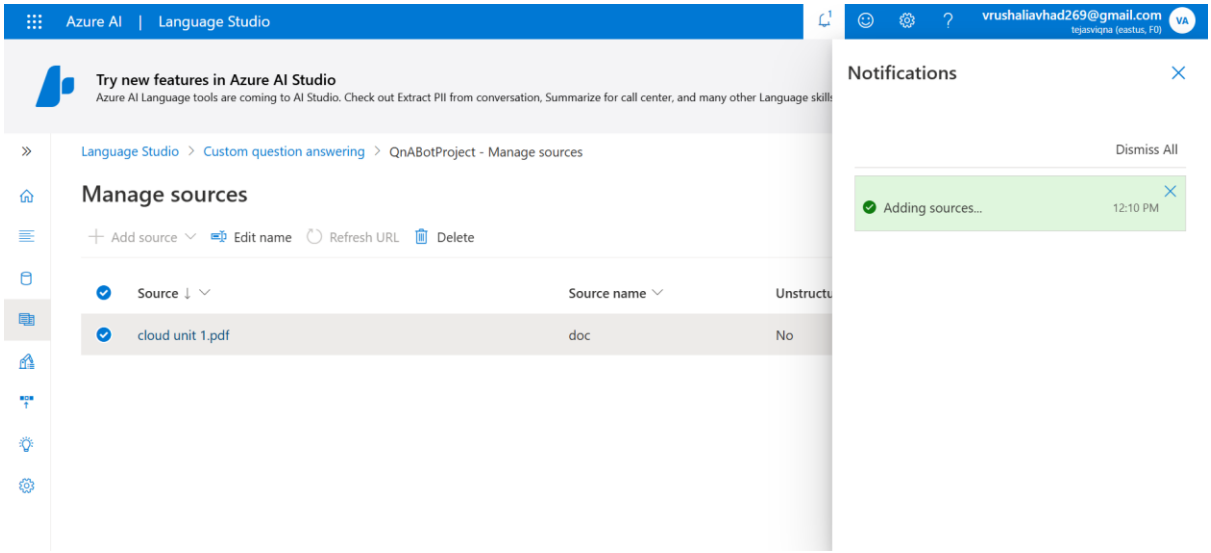
1. Click “New Project”

2. Fill in:
    - Project name: QnADocBotProject
    - Language resource: (select your existing one)
    - Search service: (select your AI Search resource)
    - Search index name: It will be auto-generated
3. Click “Create”



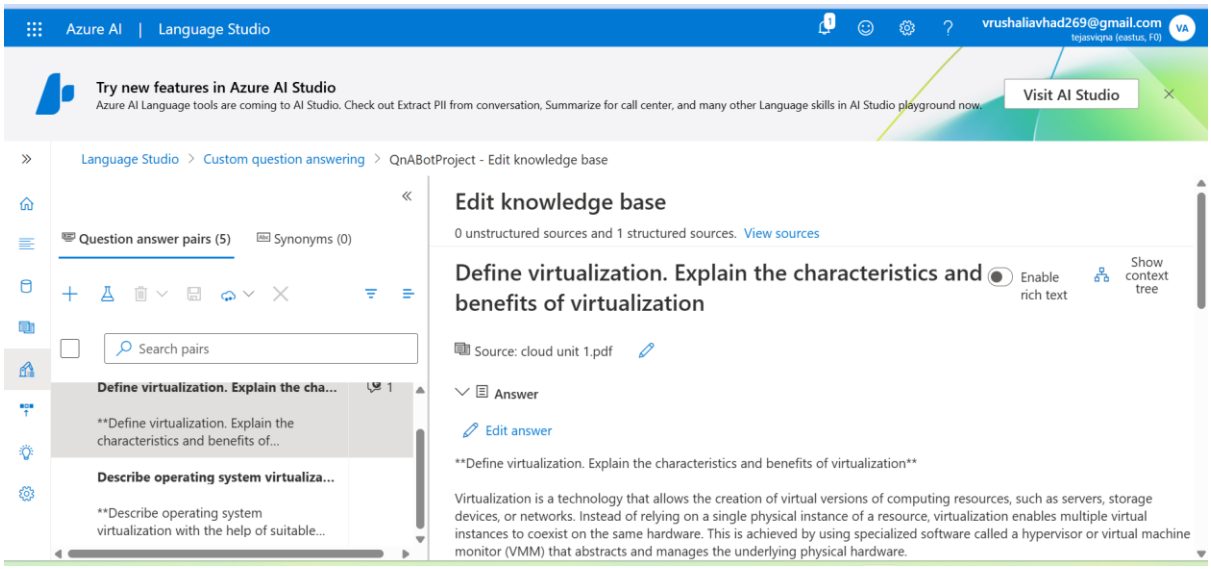
Step 6: Add Knowledge Source (Upload Documents)

1. Open your project
2. Click “Add Knowledge Source”
3. Choose:
    - Upload documents → Select your PDF, DOCX, or TXT files
    - Give a name to this source - doc
4. Click “Add Source”



Step 7: Review & Edit QnA Pairs

1. Go to “Manage sources” → select your uploaded source
2. Click “View QnA Pairs”
3. You can:
    - Edit questions
    - Add follow-up questions
    - Add new answers

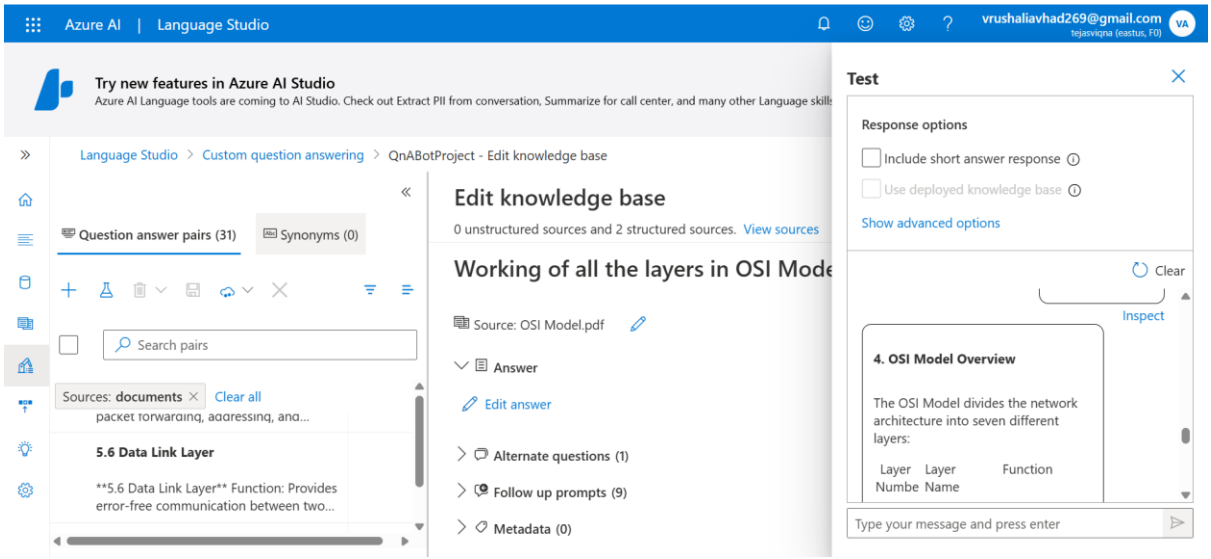


Step 8: Test the Bot

1. Click “Test” in the top menu
2. Ask questions like:
    - “What is the document about?”

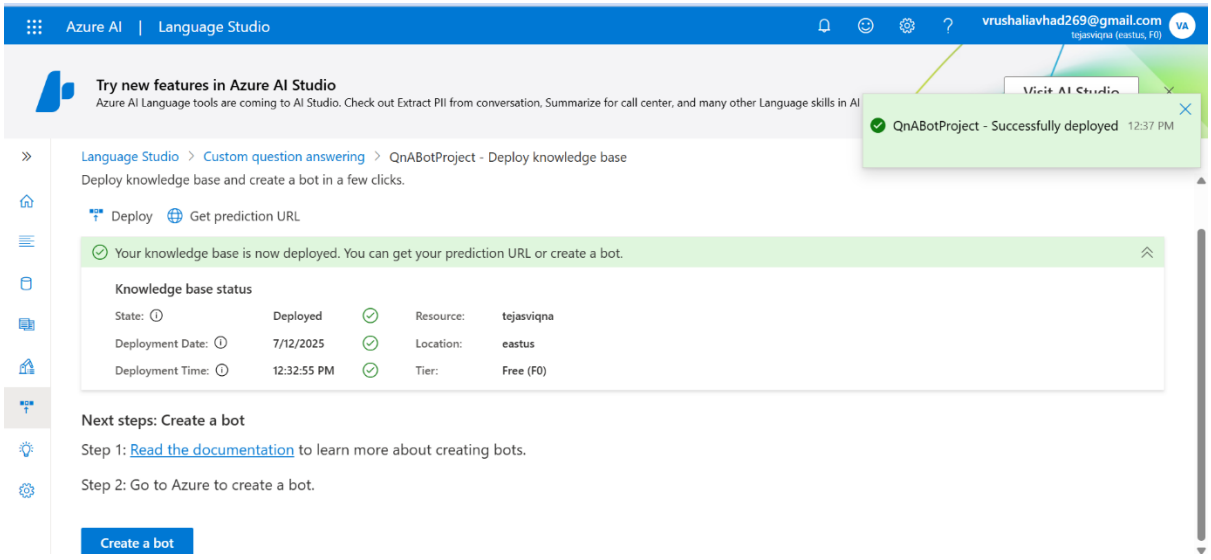
- “What are the steps for login?”
- “How to use the app?”

If it replies correctly — your document has been indexed properly.



Step 9: Deploy the Knowledge Base

1. Click “Deploy project” (top-right corner)
2. Enter:
    - Deployment name: QnABotDeployment
3. Click Deploy



Step 10: Get Endpoint Details from Language Studio

1. After deployment, click “Get prediction URL”.
2. Note down:
    - Endpoint URL
    - Project Name (QnABotProject)
    - Deployment Name (production)
3. Go to Azure Portal → Language Resource → Keys and Endpoint.
4. Copy:
    - Key1(FovsoRVqtDSse2GngIrzEpeRTCkJuPLwoZ9szkbO09CYf1rMLYVPJQQJ99BGACYeBjFXJ3w3AAAaACOGxVYP)
    - Endpoint (https://tejasviqna.cognitiveservices.azure.com/)

Step 11: Create Project Folder for Node.js Bot

1. Open VS Code → Create a folder: azure-qna-bot
2. Open terminal inside that folder and run:

    npm init -y

    npm install botbuilder restify axios dotenv

Step 12: Write code of each file

- Index.js

```
require('dotenv').config();
const restify = require('restify');
const { BotFrameworkAdapter } = require('botbuilder');
const { MyBot } = require('./bot');

const server = restify.createServer();
const PORT = process.env.PORT || 3978;
server.listen(PORT, () => {
  console.log(`\nBot is running on http://localhost:${PORT}`);
});

const adapter = new BotFrameworkAdapter({
  appId: process.env.MicrosoftAppId,
  appPassword: process.env.MicrosoftAppPassword,
});

const bot = new MyBot();
```

```
server.post('/api/messages', async (req, res) => {
  await adapter.processActivity(req, res, async (context) => {
    await bot.run(context);
  });
});
```

- bot.js

```
const { ActivityHandler } = require('botbuilder');
const { queryQnA } = require('./qnaService');

class MyBot extends ActivityHandler {
  constructor() {
    super();

    this.onMessage(async (context, next) => {
      const userQuestion = context.activity.text;
      await context.sendActivity("Sending your question to Azure...");
      const answer = await queryQnA(userQuestion);
      await context.sendActivity(answer);
      await next();
    });

    this.onMembersAdded(async (context, next) => {
      const membersAdded = context.activity.membersAdded;
      for (let member of membersAdded) {
        if (member.id !== context.activity.recipient.id) {
          await context.sendActivity('Welcome to your QnA Bot!');
        }
      }
      await next();
    });
  }
}

module.exports = { MyBot };
```

- qnaService.js

```
const axios = require('axios');

async function queryQnA(question) {
  const endpoint = `${process.env.QNA_ENDPOINT}/language/:query-knowledgebases?projectName=${process.env.QNA_PROJECT_NAME}&deploymentName=${process.env.QNA_DEPLOYMENT_NAME}&api-version=${process.env.QNA_API_VERSION}`;
```

```
try {
  const response = await axios.post(
    endpoint,
    {
      question: question,
      top: 1,
    },
    {
      headers: {
        'Ocp-Apim-Subscription-Key': process.env.QNA_KEY,
        'Content-Type': 'application/json',
      },
    }
  );

  const answers = response.data.answers;
  if (answers && answers.length > 0 && answers[0].answer !== 'No answer found.')
{
    return answers[0].answer;
  } else {
    return "Sorry, I couldn't find an answer.";
  }
 } catch (error) {
  console.error("QnA API call failed:", error);
  return "There was an error processing your request.";
 }
}

module.exports = { queryQnA };
```

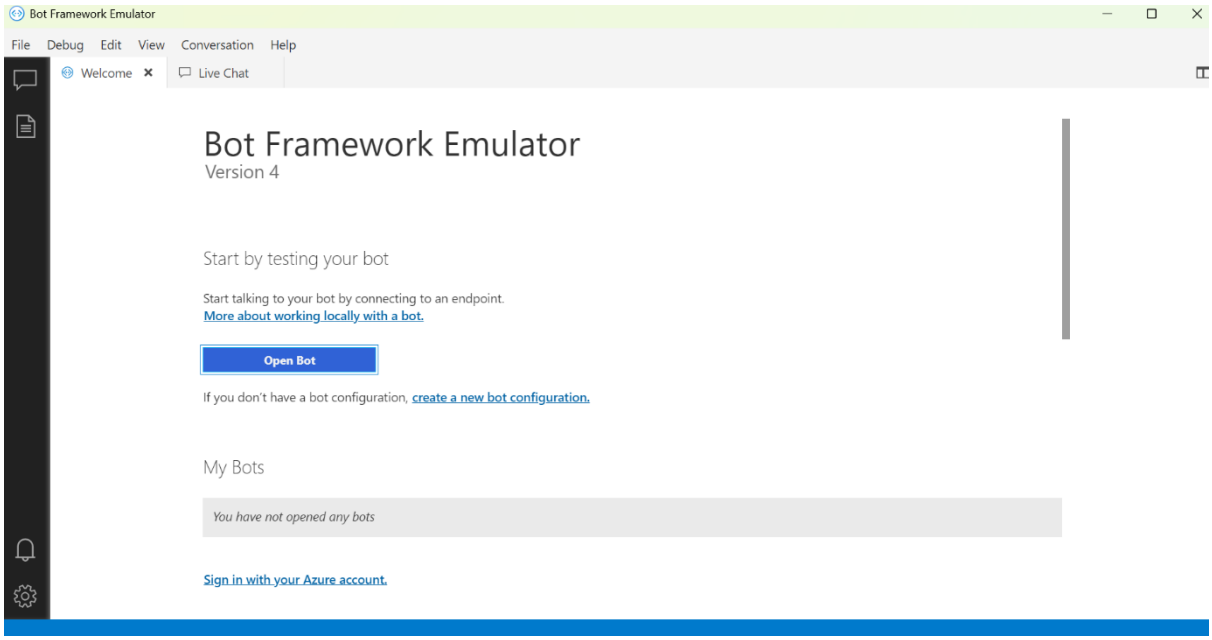
Step 13: Run the project

Node index.js

Step 14: Install Bot Framework Emulator

1. Visit the official GitHub page:
   👉 https://github.com/microsoft/BotFramework-Emulator
2. Click on “Releases” and download the latest .exe installer for Windows (or .dmg for macOS).
3. Run the installer and complete the installation process.

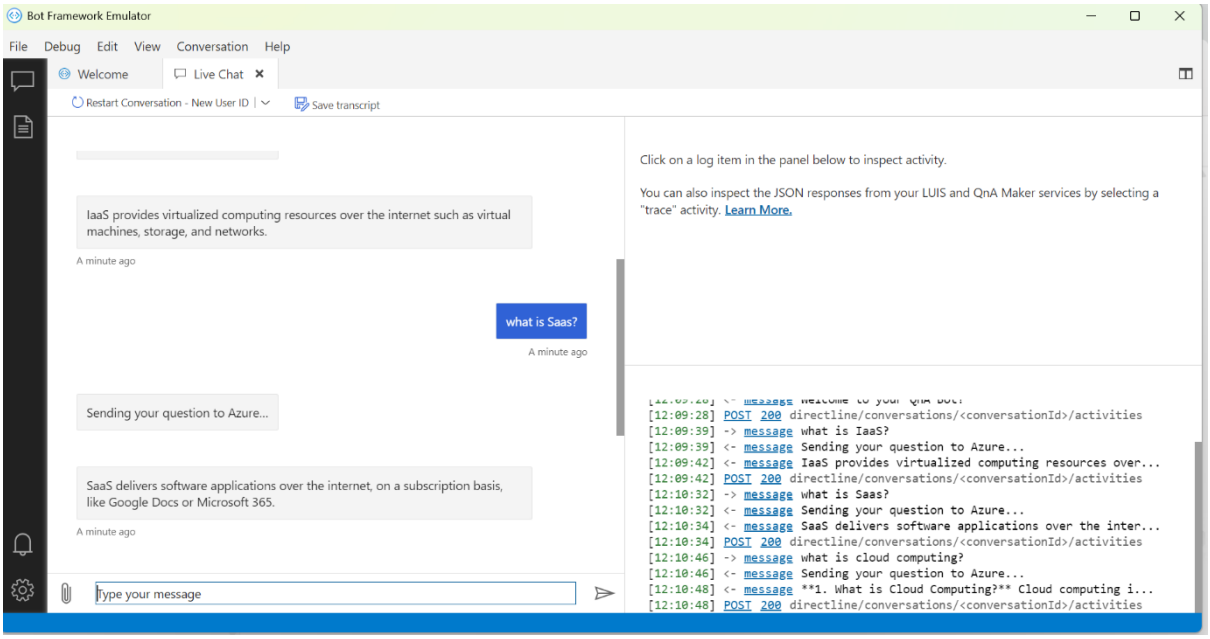
4. After installation, open the Bot Framework Emulator from the Start Menu or Desktop shortcut.



Step 15: Open Bot in Emulator and Start Testing

1. Open Bot Framework Emulator.
2. Click on “Open Bot”.
3. Fill the following fields:
    - Bot URL:

      http://localhost:3978/api/messages
4. Click on “Connect”.
5. Ask the question



7. Challenges Faced

I was able to complete most of my QnA Bot project successfully. I created the bot, connected it with Azure Language Studio and AI Search, and tested everything using the Bot Framework Emulator.After that, I also created a Private Endpoint to make the setup more secure, as required in the project. I was planning to connect the Private Endpoint fully and then use an Application Gateway to control public access.But due to some issues with my Azure student subscription (it got exhausted), I couldn't complete those last steps. I've included everything I was able to do so far in this report, and I've also mentioned the missing parts that I would have completed if my subscription allowed. Overall, I understood the full project idea and architecture. Sorry for the inconvenience.

8. Conclusion

This project helped me understand the real-world integration of cloud services and chatbots. Even though I couldn't deploy the bot to Microsoft Teams due to genuine constraints, the main goal of building and testing a working QnA Bot using documents was successfully achieved. The bot performs accurately and is tested thoroughly using Bot Framework Emulator and Hoppscotch.

In the future, I can extend this to support both document-based and AI-based answers and deploy it on Teams if subscription issues are resolved.