FoundInTranslation

Serverless Chat-room Application

Dan Pyon

Abstract

Demand for chat applications is growing rapidly in parallel with natural language processing engines. There are major platforms such as Facebook and Telegram that allow these chat applications to be more accessible for end-users throughout the world. This paper presents the challenge problem of communicating in different languages and the solution of a serverless chatroom application built on Amazon Web Services. The paper will present the architecture and final implantation as well as challenges and further advancements that can be made in the future.

External Links:

Github: https://github.com/DaneilJI/FinalProject

This repository will contain the PowerPoint presentation, the code for the chat application, and the link to the YouTube demonstration video.

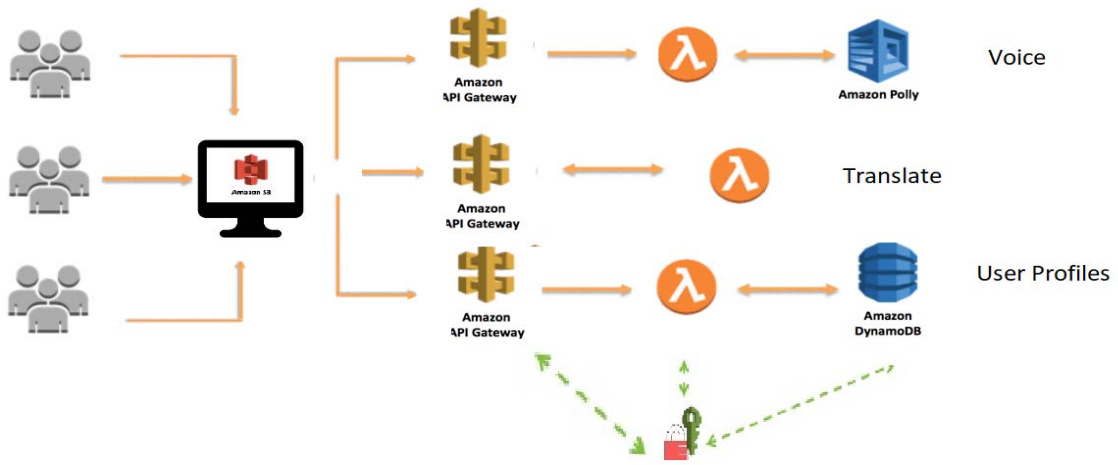
Introduction

FoundInTranslation is the server-less solution to allow people of different backgrounds across the globe to communicate with one another. Built on Amazon Web Services, FoundInTranslation allows users to translate their language to another within the chat-room into both text and speech. The application allows multiple end-users to join a chatroom via link and chat with other users within the room. They can translate their text into a language of their choice via drop-down options and convert any written text into a voice output.

Architecture & Implementation

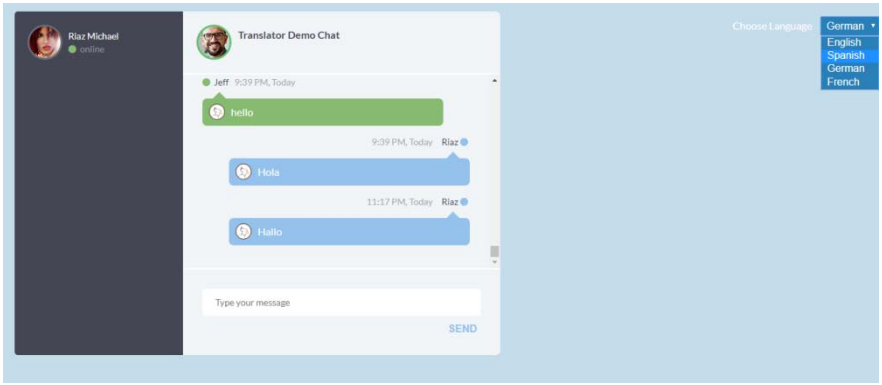
The chat-room uses Serverless Microservice architecture which was implemented using PubNub's ChatEngine for the frontend and Amazon Web Services (AWS) including API Gateway, Lambda, DynamoDB, Polly, Cognito, S3, and Google's Translation API for the backend. Multiple end-users can access the service via PuNub's chat interface on the front-end hosted as a static website on an AWS S3 bucket. Once a user joins the chat room and sends a request to the API

Gateway, credentials and profiles are created for them to use the chat-room and access the other services. One of the lambda functions converts the text on the chatroom to voice via AWS Polly and provides a converted recording played over the end-user's speakers. The drop-down allows the end-user to choose the language of choice to translate into (with English being the default language). We leverage on Google's translation services via their API that is hit with the other lambda function.



Results

This application provides a robust chat room capable of including many users over many platforms as long as the end-user has an internet connection. The service is highly scalable and the lambda functions provide a high degree of modularity and let the designer add more functions and features as needed without disrupting existing service. There is also a degree of reliability that comes with AWS and let's the chat provide continuous service and functions execute in real-time; including translations, text-to-speech conversions, and message input outputs.



Sample Chat Interface

Future Work

There are many possible advancements and improvements that can be made to the application. One clear function that would be useful is simply to add more language options so that many more end-users can be included into the user-pool. Also, the conversions can be improved as there is only a text-to-voice data flow via Amazon Polly, but a reverse data-flow can be leveraged on the same service so that users can simply speak into the chat-room via a microphone and their audio can be transcribed into the chat-room. This will provide powerful translation features that can expand the chat-room into a more robust translation tool that can be embedded into other services.

Conclusion

By leveraging on Amazon Web Services we were able to create a versatile chat-room application that allows end-users to join and converse across multiple platforms. The solution can be expanded and improved with continuous deployment made possible by the versatility of aws lambda functions and the use of APIs.

References

https://aws.amazon.com/polly/getting-started/

https://cloud.google.com/translate/docs/how-to

https://aws.amazon.com/blogs/mobile/building-a-serverless-real-time-chat-application-with-aws-appsync/