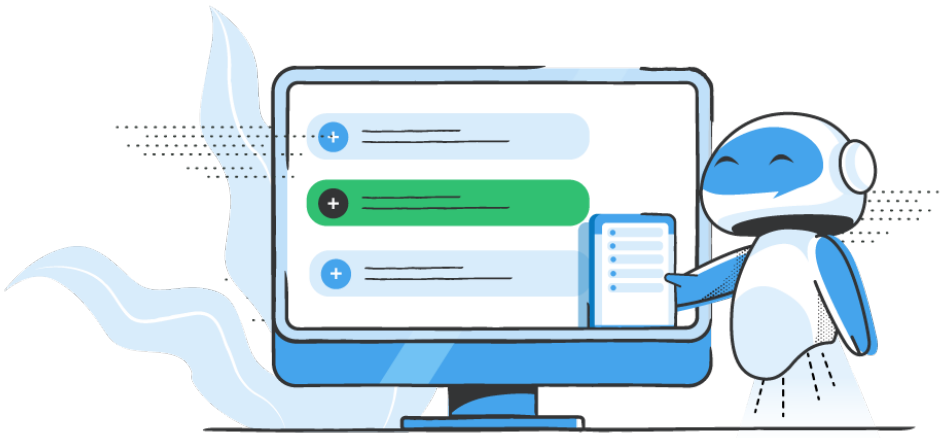
CUBA

CU Boulder Assistant

A chatbot for all Students and Faculties



Team Members:

Aishwarya Satwani

Shreyas Kapoor

TABLE OF CONTENTS

PROJECT GOAL

CUBA is a chatbot that acts like a personal assistant to all members of CU Boulder.

This chatbot is developed using Microsoft Botframework (NodeJS) and Microsoft LUIS for NLP. It is deployed in Microsoft Azure and is accessible to all members of the slack workspace created.

The following would be the features of the bot:

- List out upcoming meetings and show your day schedule based on your google calendar.
- Consists of a knowledge base with the following information to guide the students and faculty
    - University guidelines and essential information.
    - Point of contact for various departments.
    - Events around campus for today and the rest of the week.
- Provide directions to important locations.
- Weather information for the day.
- Buff card balance (Future enhancement, dependent on acquiring the API)

SOFTWARE AND HARDWARE COMPONENTS

SOFTWARE

NLP	Microsoft LUIS
App service	NodeJS Web App Service (along with an appropriate app service plan)
Storage	Cosmos DB (SQL APIs), Redis Cache
Logs	Cosmos DB, Azure Application Insights, Kudo
Cache	Redis Cache
Bot service	MS Bot Service
Frontend, Authentication	Slack
External APIs	OpenweatherAPI, Google Calendar, Google Maps, Slack API for auth
Local Testing	MS Bot Emulator
API Testing	Postman
Code editor	Visual Studio Code
Terminal	iTerm

HARDWARE

Laptops	With support for VS code, MS bot emulator
Internet	Stable internet connection

COMPONENTS DESCRIPTION

Microsoft Azure was chosen as the deployment platform as Azure provides a strong bot integration support as well as numerous bot services.

Web app service

Azure Web Apps is a cloud computing platform for hosting websites. It facilitates publishing Web apps running on various frameworks and developed in different programming languages across the globe. Upon creation the sub-domain azurewebsites.net is provided (custom domain can be assigned based on the app pay-tier). It also provides multiple deployment methods.

Azure app service also provides capability for troubleshooting, tracking logs and creating monitoring alerts. These apps can be scaled up and scaled down based on the usage.

Cosmos DB

Microsoft's Cosmos DB is globally distributed, fully managed NoSQL DB. It has high-availability and said to guarantee response within milliseconds. Following are the features that made us choose this component as our DB:

- Provides SQL APIs to access the stored data (MongoDb and Cassandra available too).
- It can be scaled instantly.
- Multi region writes and data replication capability
- The DB calls and availability can be monitored

In this project, SQL APIs have been used to access the DB. The SQL API lets clients create, update and delete containers and items. Items can be queried with a read-only, JSON-friendly SQL dialect.

Limitations - SQL joins between tables cannot be done. Apart from count, sum, min/max; group by and other aggregations are not possible.

Redis Cache

Redis is used to increase app performance, thus making the app more readily available and reduces the cost of DB calls to Cosmos (i.e. make it cost effective). Redis can handle concurrent users. The throughput increases by 800% upon adding this to the architecture.

MS LUIS

For the NLP of our chatbot we have gone ahead with Language Understanding (LUIS). This is a cloud-based conversational AI service. The model predicts the overall intent of the natural language text along with keywords that would help the bot make decisions accordingly. LUIS is easy to train and integrate.

Bot Service

Azure Bot Service facilitates bot accessibility across various channels. It provides the capability to connect to users via popular channels.

External APIs

Provided additional functionality to our bot in providing real time, user customized data.

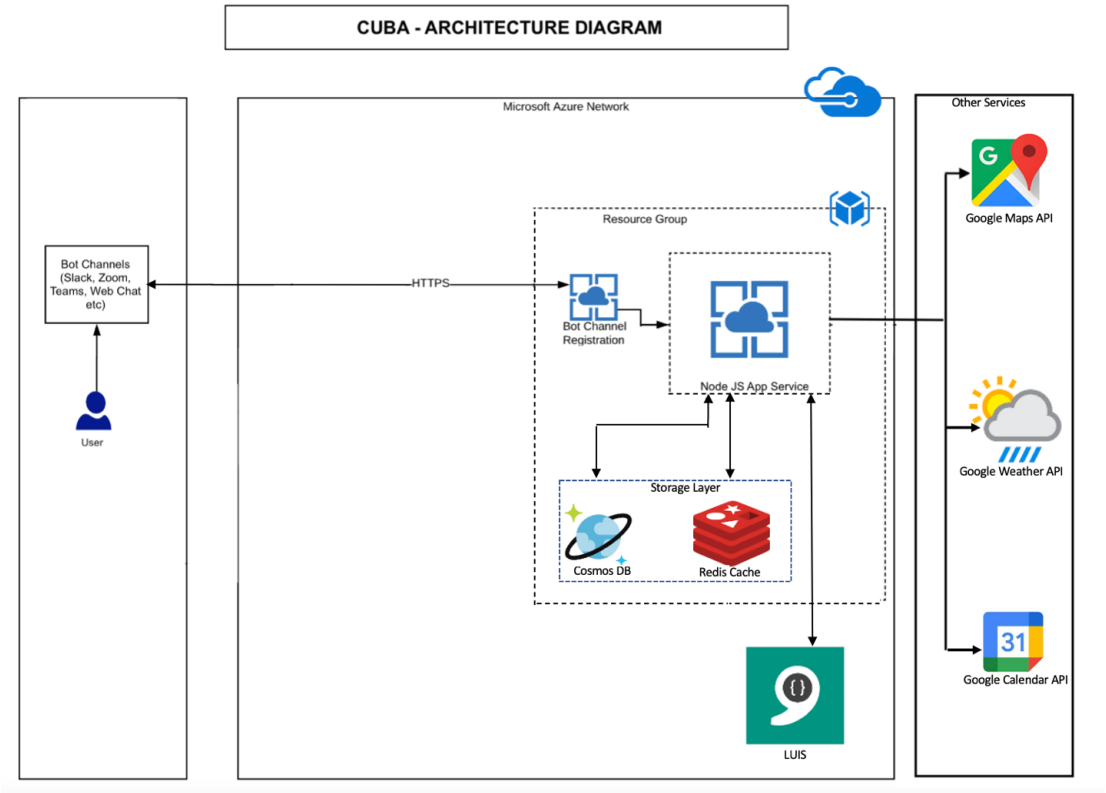
Slack UI

Integration with Slack provided the appropriate UI for our bot backend. It also takes care of user authentication. Slack also provided additional APIs that were used to access user information and created a customized bot chat experience for the user.

App Insights

This is a feature of Azure Monitor. It provides extensible Application Performance Management (APM) service. We used it to monitor our live application. It detected performance anomalies and provided various powerful analytics tools to help diagnose issues.

ARCHITECTURAL DIAGRAM



INTEREACTIONS BETWEEN SYSTEM COMPONENTS

The bot resources used are deployed in MS Azure. The bot is integrated by creating a Slack App on a specific workspace and is accessible to all the members within it. The integration is facilitated by the Bot channel registration provided by Microsoft. This is a service that allows the bot to be available via various channels upon secrets and tokens exchange. The user authentication is taken care of by Slack and the details of the user logged in can be accessed via Slack APIs.

Once the user is logged in, the endpoint of the chatbot is called with the user message and user token. The bot logic is written in Microsoft Botframework (NodeJS) and a user-friendly chat experience is developed.

The user message received via the bot endpoint is then sent as a parameter to MS LUIS endpoint. MS LUIS is an NLP service that categorizes the user input into intents based on the training data. The API response from LUIS is a JSON structure with intents in descending order of the score.

The app then considers the highest score intent and queries the DB (Cosmos DB provides an SQL API) with the intent as the key.

For direct information queries - The response from the DB is formatted and then sent to the user back onto slack, as a bot response.

For queries that require further API calls (such as directions/calendar updates/weather updates etc.) – The DB returns the name of a dialogflow (waterfall model) that requires the bot to make an API call with the appropriate user-details and parameters.

The logs of user and bot conversation are added to the Cosmos DB. The bot responses for future conversation are saved in Redis cache to increase bot efficiency.

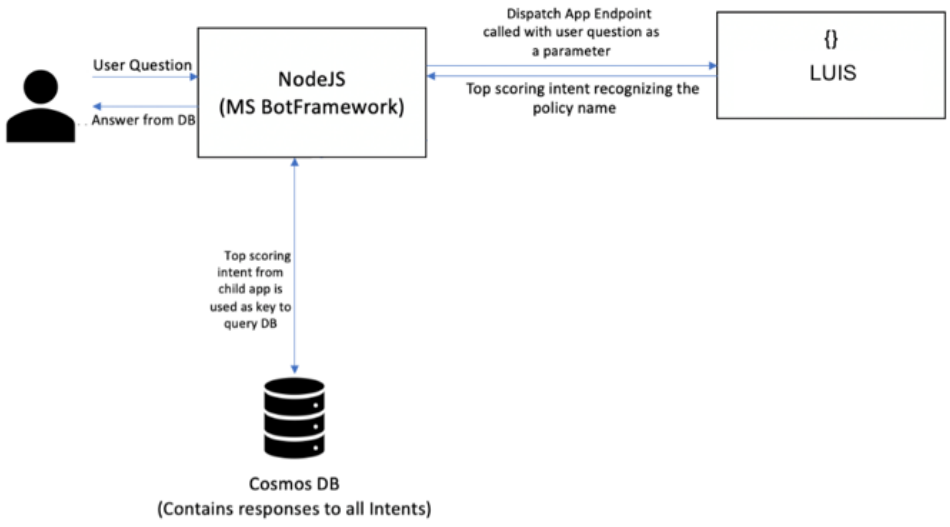
The bot is deployed on Azure Web App by specifying a .yml file. The .yml file contains commands to install the required packages, create a stable build, then create the artifact and finally deploy it.

DEBUGGING

- For the development of the chatbot dialogs, MS bot emulator was used to provide the local frontend for the chatbot and hence a local testing environment. This was used for end-to-end testing of all the features locally.
- To validate the REST APIs, the Postman application was used to test and debug the server.
- To debug the deployed code, Azure Application insights provide the required logs to debug.
- Additionally, kudo (Anytime you create an app, App Service creates a companion app for it that is secured by HTTPS) was also used to debug errors found during the deployment. It additionally provided the following functionality as well:
    - App settings, connection strings, env variables, server variables, HTTP headers.
    - Provides a CLI to run commands.
    - Download IIS diagnostic dumps or Docker logs.
    - Manage IIS processes and site extensions.

TRAINING MECHANISM

- NLP: Training the language model
    - LUIS is trained by firstly creating intents, logical categories on the use-case of the bot.
    - After creating intents, these categories are trained by adding utterances i.e., anticipated user questions for every intent.
    - These utterances can be further modified and trained with entities i.e., keywords in a statement the bot can pick out and perform specific tasks.
    - Once the model is trained, it is published, and the endpoint is created for our application to use.



TESTING MECHANISM

Health check API that periodically validates the alive-ness of the application is used. When the API gives a response apart from 200, then an alert is sent to the owners of the app service.

The health-check API would cover the following services that are essential for the application to run as expected.

- Server check
- LUIS
- Cosmos DB

To test out the NLP, the batch testing mechanism provided from LUIS was utilized to validate the accuracy of the model.

CLOUD TECHNOLOGIES USED

The project will use the following cloud technologies:

- API services: Our project would require API services such as Google calendar, Google Maps, Slack API, Google Weather API and APIs provided by CU to get events updates as well as buff card balance.
- App services: Microsoft app services will be used to deploy the NodeJS backend on Azure.
- Database: The knowledge base is saved in Cosmos DB i.e., the user query responses.
- Cache: Redis Cache will be used to store the frequently queried responses to increase efficiency.

LIMITATIONS

The following are the limitations of the project:

- Azure is expensive and hence deploying the project in large scale for the entire university could use up a substantial chunk of resources.
- Azure also requires platform Expertise. This means that developers will need to be trained beforehand for them to get used to deploying and debugging the resources.
- At this point certain APIs are unavailable to us. Because of this, we were unable to provide the real time buff card balance of the user. Once available, the bot can correctly provide the data.
- Currently, the application runs on the free account, but can be potentially scaled up based on user base.

LEARNINGS

The project was a nice learning curve for us. We learned how to do integration of SLACK with Microsoft app service.

One place where we got stuck was the deployment of this NodeJS application on app service. However, upon modifying and trying various commands, we resolved the deployment issues and then proceeded with our slack integration.

We also got an in-depth understanding of how the tenants and billing work within Azure.