Speech Recognition Engine

Final Report

Bhawneet Singh
bhawneet.singh@colorado.edu

Karan Dhir
karan.dhir@colorado.edu

Tanmay Desai
tanmay.desai@colorado.edu

December 2022

School of Computer Science
University of Colorado Boulder
Boulder
USA

This is the final project report for course CSCI 5253 Data Center Scale Computing.

Professor Dr. Dirk Grunwald

-

Contents

1 Project Goal

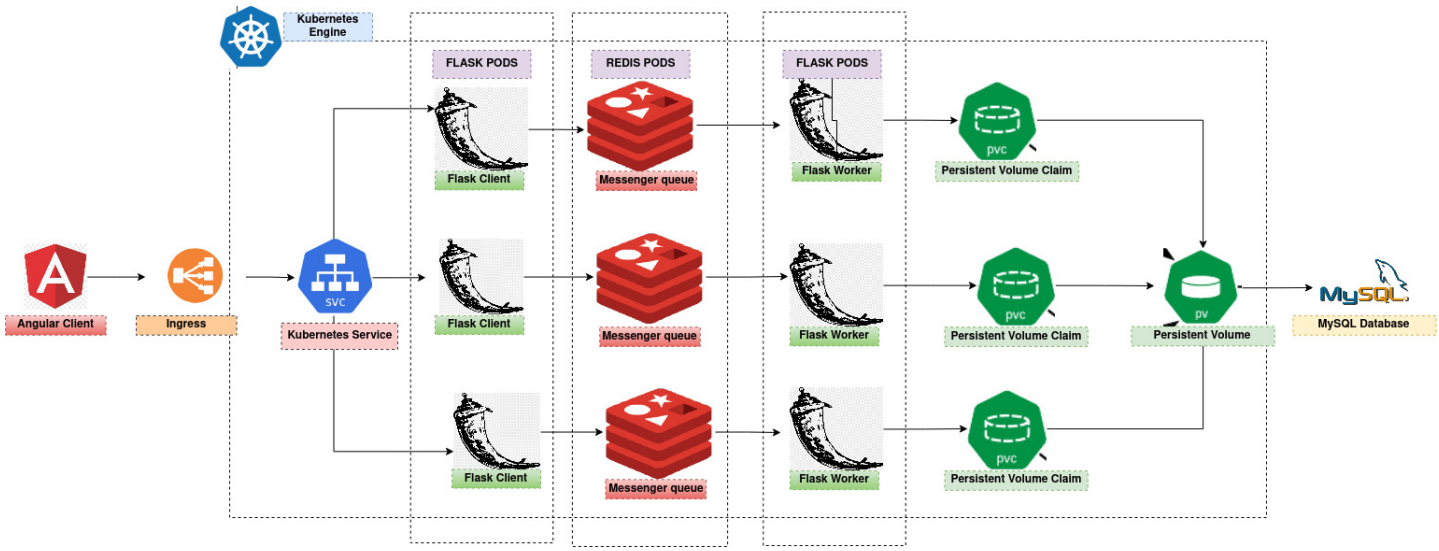
One of the sectors that are expanding most quickly nowadays is the music industry. The number of users on streaming platforms grows yearly. The amount of data, such as preferred genres, singers, songs, etc., is growing as this number does as well.

Data centers are required, and effective cloud computing is required. In this project, we develop a speech recognition engine using angular, which takes user-provided voice input with a.wav extension, converts it to text using speech recognition, and then passes that text through the Spotify database to find all the songs associated with the artist, which will then be displayed on the front end. As a result, we have created a platform for streaming music utilizing Google API and Spotify.

2 List of Hardware and Software Components

Software	Hardware
Rest API	Google Cloud Platform(compute engine)
MySQL	
Redis	
Docker	
Flask	
Kubernetes	
Angular 8	

3 Architectural Diagram

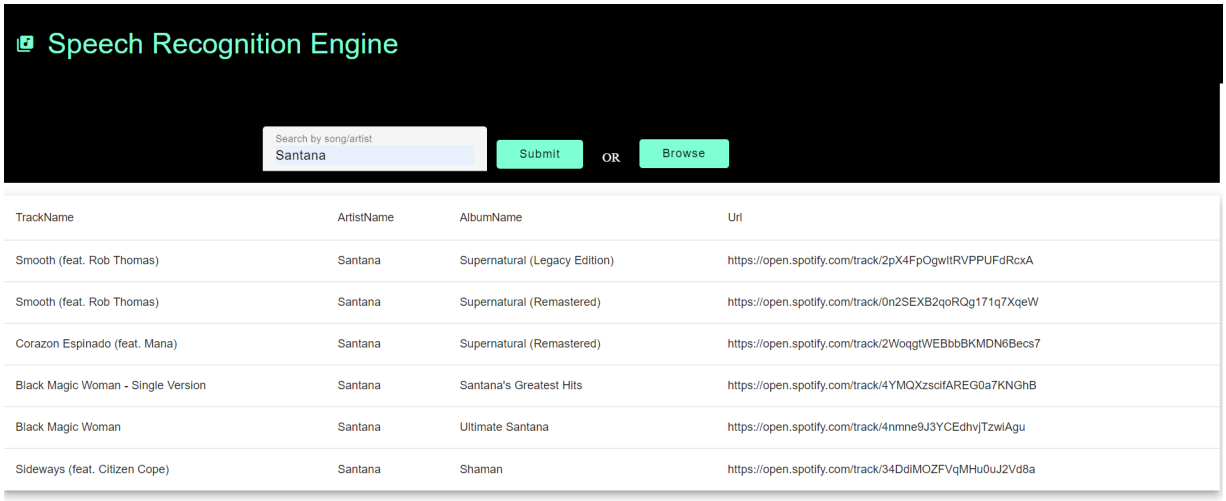


Musify, a speech recognition engine.

4 How do software and hardware components interact with each other?

- The first component that we have is an front-end(built using Angular8) that takes input in the form of ".wav" file format or you can search by artist name for that song if data is present in MySQL database.
- Then we take the input file and pass it to rest server using ingress. Ingress is an API object which provides routing rules to manage external users' access to the services in a Kubernetes cluster.
- In rest server, we have endpoint"/apiv1/voice" that converts voice input into text input using Speech Recognition and calls Spotify API to get response for that text. That response contains 4 values such as Track Name, Artist Name, Album Name and Spotify URL.
- The results are then queued in the worker redis queue which then routes the response to the worker. We used redis because it employs a primary-replica architecture and supports asynchronous replication where data can be replicated to multiple replica servers.
- In worker server, we created a SQL database "spotifydb" and table "tracks". The worker server gets the response from the redis queue and then calls the insert query which inserts all the responses in the table.
- In order to maintain uptime, ensure data consistency, boost performance, and guarantee continuous availability for a better user experience, we employed MySQL, a load balancing program.

- After the responses are stored in the database, we have another endpoint that accepts artist name as string input "/apiv1/find/string: artistname" which will connect to the SQL database and get all the responses to the front-end and display it for the user.
- The main benefit of adopting a Kubernetes cluster is that programs can run efficiently and with little downtime, requiring less assistance when a node or pod fails and needs to be manually repaired.



Front-End Angular

5 Debugging

- Deployed the whole application on Kubernetes locally.
- Used logging with Redis, MYSQL, Rest, and Worker nodes to debug each component/pod.
- It's easier to debug for each component by using kubectl logs 'container-name'.
- Used port forwarding for Redis and MySQL.

6 Working System

Capabilities:

- In a variety of businesses, speech recognition technology can assist decrease errors, increasing customer happiness, and speeding up procedures.
- We can upload multiple input files at the same time and then we can hold their response in the Redis queue to be processed by the worker later on.
- Users can easily process various song URL's(response output) at the same time using our application

Bottlenecks:

- The potential bottleneck for this project is that if the Spotify server goes down, we will not be able to consume its Web API endpoints. Also, the limitations of the cluster are because of high costs.
- We used one replica for each pod(Rest and worker) but we can add more to incorporate the distributed system.
- Currently, we are only accepting voice inputs in the form of wav file format but not other formats.