Spotiguys

Peyton Martin, Nolan Porter, Collin Giess, Yug Patel

Spotiguys is a web based application for Spotify users to make group playlists where songs are chosen depending on a user's tastes. We accomplish this by using Spotify's API to pull user's data, and running analytics to compare it to other user's data using AWS Kinesis.

The main goals of our project were to fix the shortcomings with Spotify's built in group playlist generator called Blend. The things we focused on most were allowing any number of users to create a playlist together, and generating a playlist where the songs fit well together. We were able to accomplish both of these things thanks to AWS technologies. Each Spotify track has a dictionary of 'audio features' associated with it. These are values(usually floats from 0-1.0) for different aspects of the song. We first take all the tracks from a user's profile and find the average values of every song's energy, danceability, and valence(how positive sounding the track is) features. We then compare these averages for each user to each other, to find a total average. Then, from the big pool of songs from all users, we pick out the tracks with audio features within a close enough range to the total average.

Technologies Used

Amplify

Our main goal with Amplify was to host the web application without writing a lot of backend code. We used the Amplify Studio to define the Groups and GroupMembers's Models to create a GraphQL schema, which is being used by our single page React Application through the GraphQL APIs to handle and manage group data.

Api Gateway

We use the API Gateway to define and host our custom API that uses Lambda functions to make calls to the Spotify API. Every interaction with the Spotify API except for the User Login is being handled by the API Gateway.

Callback Lambda

After a user is redirected to login, the Callback Lambda retrieves the user's access token and authenticates them. We then use a Python library called Spotipy to handle making api calls to Spotify. We first get all of the users playlists, and add all the songs from them to a list. For every song, we use Spotipy to get its audio features. All of the user's songs and their audio features are then stored in an S3 bucket.

Producer Lambda

The job of the Producer Lambda is to populate the Kinesis Stream. When a user joins a session, all of their songs and audio features are pulled from an S3 bucket. The songs

and features are then sent to the Kinesis Stream as records. The records are all put in a shard for the user.

Kinesis

A data stream for storing Spotify user data. Each user's data is kept in a shard.

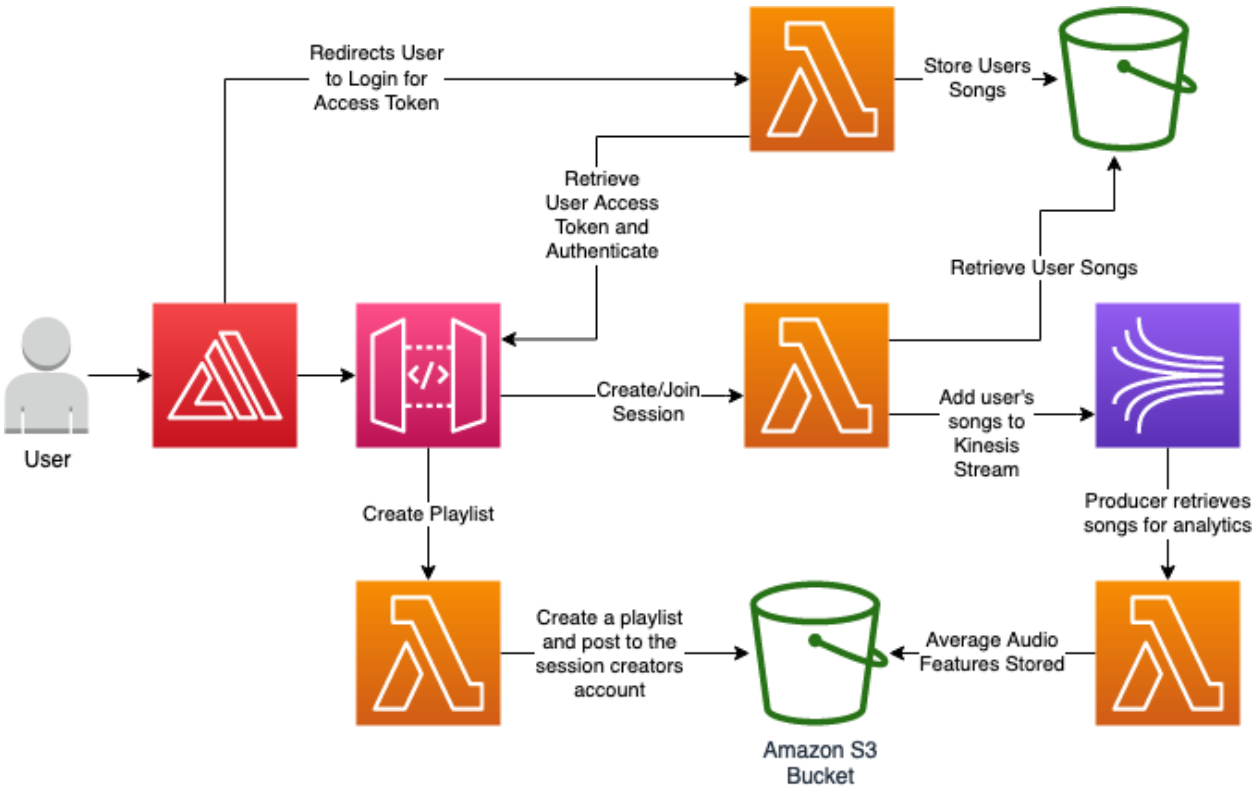
Consumer Lambda

The Consumer Lambda takes in a Kinesis shard which contains all songs and their associated audio features from a user. The average for all the audio features is calculated to find the average taste for a user. These averages, alongside all of the user's songs, are stored in a bucket.

Playlist Post Lambda

The Playlist Post Lambda takes every user's average feature values and songs from an S3 bucket. All the average values are then totalled to find a total average. Every track the group of Spotify users has saved is then compared to these total average values, and if they are within a specific range they get added to a list. If not enough or too many songs(20-100) are added to the list, the range is adjusted and we restart the process. This is done in a loop until we get the desired amount of songs.

Architecture Diagram

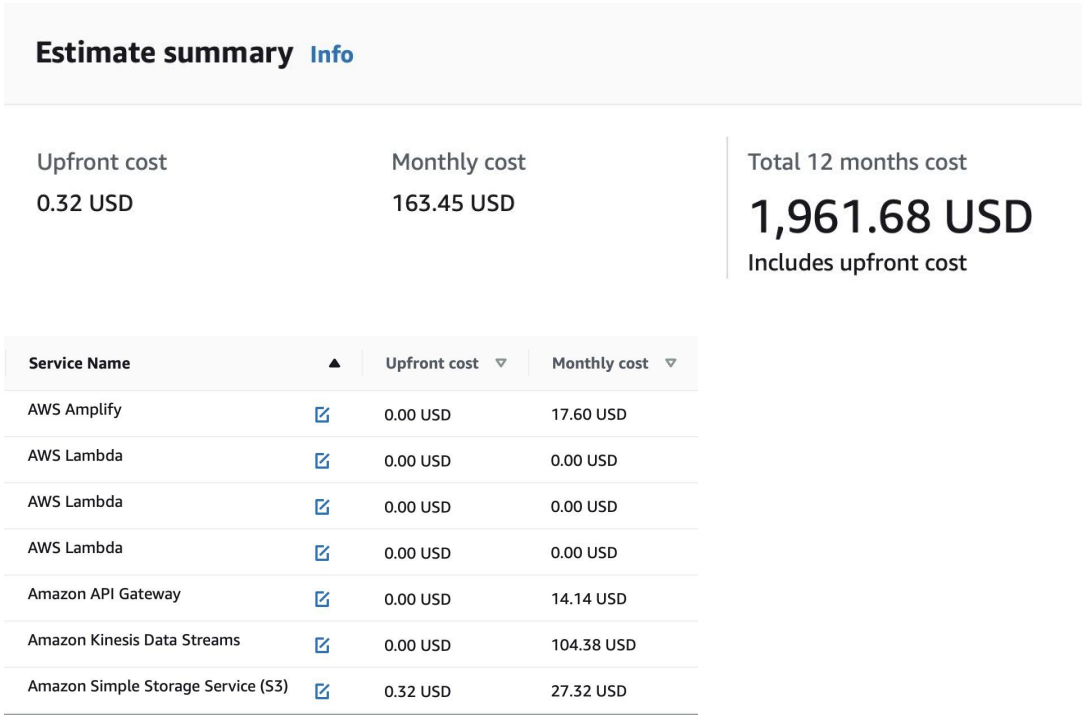


Estimated costs

Our project was very cost efficient and we were able to use technologies that would save us money in the long run. One decision we made was to go serverless. Using serverless technologies like lambdas, allowed us to not have to run EC2 instances. This could save our application money because we would likely have to run EC2 constantly even when the app wasn't being used. Combine this with spinning up multiple EC2 instances in different regions to help with reliability, EC2 would have been a large cost on the application. Using lambdas, we only pay for the amount of times it is used, allowing the price to reflect the demand. To compound this, our lambdas were

extremely cheap and lightweight. So our team using serverless technology definitely made us more cost efficient.

Our Estimate:



Estimate summary Info

Upfront cost	Monthly cost	Total 12 months cost
0.32 USD	163.45 USD	1,961.68 USD Includes upfront cost

Service Name	Upfront cost	Monthly cost
AWS Amplify	0.00 USD	17.60 USD
AWS Lambda	0.00 USD	0.00 USD
AWS Lambda	0.00 USD	0.00 USD
AWS Lambda	0.00 USD	0.00 USD
Amazon API Gateway	0.00 USD	14.14 USD
Amazon Kinesis Data Streams	0.00 USD	104.38 USD
Amazon Simple Storage Service (S3)	0.32 USD	27.32 USD

This is our estimate after using the AWS pricing calculator. This estimate was made using the assumption that we would have roughly 30,000 daily users. Each time a user used the application all lambdas were called and they made at least 8 api requests. This assumed that all of the users' playlists were taken from their spotify account, placed into s3, and processed through Kinesis.

We were able to save a lot of money in data storage because our application only pulls text files about songs and not actual song files. Also after a playlist is created it is stored directly in spotify and not on our application.

Key Questions

- Lessons learned along the way. What would you do differently if you had more time or had to do this over?
    - We found ourselves confused on how to apply Kinesis Data Streams to our application to pull useful analytics from our data sets. It took us a while to figure out the Lambda code to process our data using Kinesis. If we had to do the project over, we would have done more research on Kinesis and its applications to prevent confusion.
    - We should have written our IAC as we were developing our services, and not after the fact. We ran into a lot of issues troubleshooting our Terraform towards the end of the project.
- Why is the cloud better for the project you picked?
    - It is much more cost effective to make use of AWS services like Lambda, so we utilize serverless computing as opposed to hosting ourselves.
    - It is much easier to create applications using managed AWS services rather than from scratch
- Any issues/challenges with the AWS technologies you selected?
    - Amplify was difficult to write IAC for. It was also hard to test because it takes several minutes just to deploy to our website.
    - It was hard to determine which data belonged in our Kinesis stream
- What other features or enhancements or would you like to see added to your project if you had more time?

- We could allow users to filter their playlist creation by genre, or other audio features.
- We could retrieve songs from places besides user playlists, such as top hits playlists