Project ConcertBuddy Final Submission Status Report

E6156 - Topics in SW Engineering Management: Cloud Computing

Focal Point: Yusong Zhao, yz4406
Team Members:

- Fu, Zhengwen, zf2314
- Duan, Junyao, jd4024
- He, Luna, zh2603
- Yusong, Zhao, yz4406
- Yutao, Zhou, yz4359

Introduction

The current situation and gap: finding the right person to go to concert is hard:

- Your existing friends do not necessarily have similar music tastes as you
- Your existing friends may have other plans when the concert happen
- It is inefficient to randomly try to find strangers to go to concert with on social media like reddit / discord, and it's even harder if you want to find someone you can vibe with

What our project is doing: ConcertBuddy uses music preferences of users to match them with other users whom they can go to the concert with. Our application has two assumptions: for time-sensitive events like concerts, we want a efficient way to find people to go with; people with similar music tastes will vibe

How this project fills in the gap: This system makes finding other people to go to concerts with more efficient and simple

Personas and Roles

There are three personas/roles:

- *User* represents a person using ConcertBuddy. A *User* may have one of the following roles:
    - *Interested in attending (some concert)*
    - *Will attending (some concert)*
- *Concert* represents a concert recorded in the ConcertBuddy system that can be attended by users
- *Finder* represents a tool to match a User attending a Concert with other Users attending that Concert based on music tastes

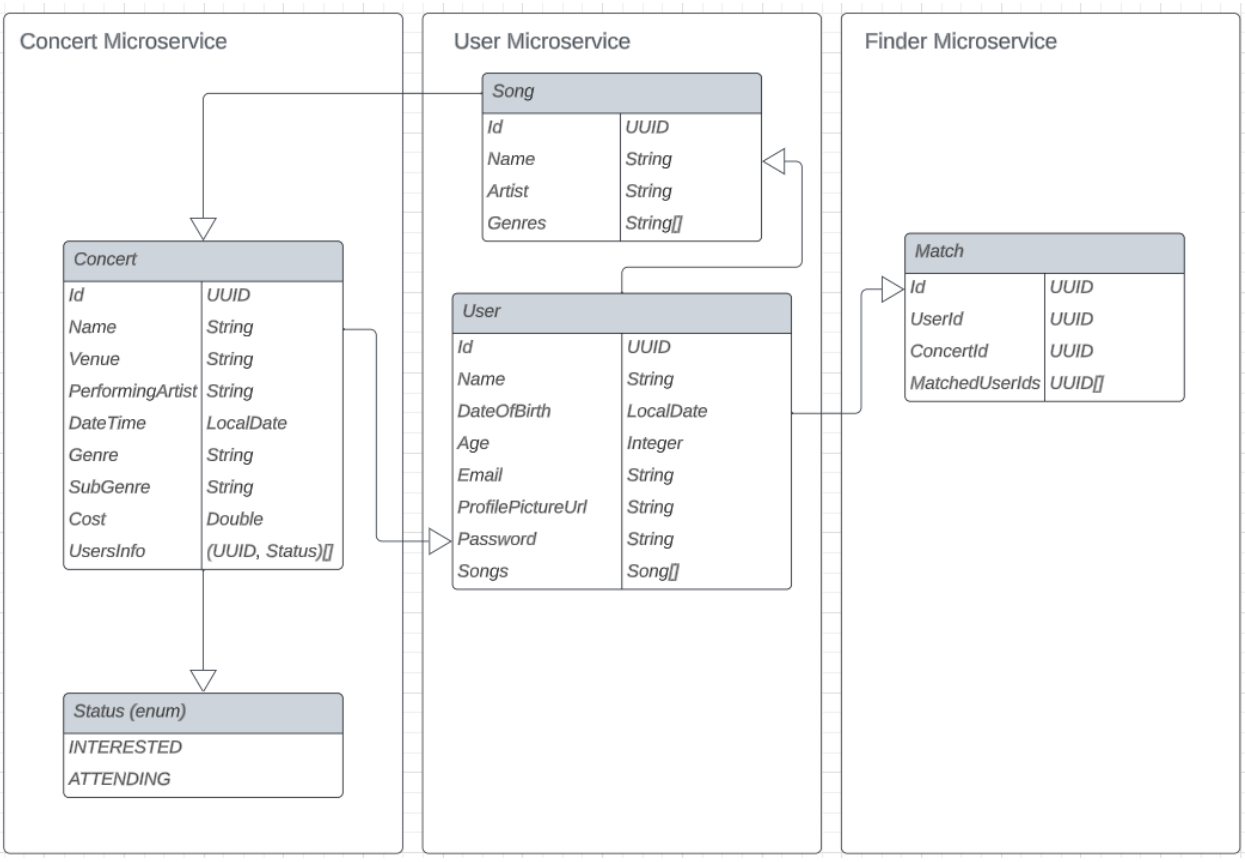
Key Features

- A user can:
    - Sign in to the website using Google Single-Sign On
    - Discover concerts they want to go to
    - Users can add songs they like to their profile using Spotify auto sync
    - Request matching with other users for a concert from the system
    - View detailed profile of the matched users
- The system can:
    - Log users in with Google Single-Sign On
    - Display list of concerts
    - CI/CD concerts microservice (automatically trigger unit test and deployment to Google app engine when there is a new commit to the main branch).
    - Automatically load concert information from Ticketmaster API.
    - Sync user playlist songs from Spotify API.

- For a user that is interested in attending a concert, provide recommendations of other users based on similar music tastes
- Communicate to users with email notifications
- Configure and control access to /users endpoint with API gateway
- Run GraphQL query on users & songs.
- Use logging filter middleware in the finder microservice. For any API calls in the finder microservice, the request url and response type is logged into a file. Additionally, any debug print statements are also logged to the file.
- Retrieve all relevant information (profile, songs, matches) related to a user with an aggregator service, effectively hiding implementation details from individual microservices
- Leveraged infrastructure-as-code technology (Teraform and pulumi) for the aggregator microservice to eliminate the need for manual deployment and update process

Domain Model

The following is an initial overview of ConcertBuddy's domain model.



The User Microservice consists of the User model and Song model. The User model consists of information about a user, and their list of and songs that they like. The Song model consists of

information about a song. The Concert Microservice consists of the Concert model and Status enum. The Concert model consists of the information about a concert, and a list of users who are interested or attending the concert. The Status enum depicts the status of a user going to a concert. The Finder Microservice consists of a BuddyFinder model that stores the user and concert that the finder will run with, and the list of matched users.

Resource Paths

ConcertBuddy exposes a set of REST resources. The following is the initial set of resource paths. The response object samples are located in the data types section after the resource paths:

- /api-docs: OpenAPI documentation.
- Users:
    - /api/v1/users
        - Get
            - Request Body: nameFilter (Optional String)
            - Response Body: User[]
        - Put
            - Request Body: User
            - Response Body: userId (type UUID)
        - Post
            - Request Body: User
            - Response Body: userId (type UUID)
    - /api/v1/users/{userId}
        - Get
            - Path parameters: userId (type UUID)
            - Request Body: None
            - Response Body: User
        - Delete
            - Path parameters: userId (type UUID)
            - Request Body: None
            - Response Body: None
    - /api/v1/users/{userId}/songs
        - Get
            - Path parameters: userId (type UUID)
            - Request Body: None
            - Response Body: SongsWithHATEOASLinks
    - /api/v1/users/{userId}/songs/{songId}
        - Put
            - Path parameters: userId (type UUID), concertId (type UUID)
            - Request Body: Song
            - Response Body: None
        - Delete
            - Path parameters: userId (type UUID), concertId (type UUID)

            - Request Body: None
            - Response Body: None
    - /api/v1/users/{userId}/SpotifySync
        - Put
            - Path parameters: userId (type UUID)
            - Request Body: SpotifyAccountToken (String)
            - Response Body: None
- Songs:
    - /api/v1/songs
        - Get
            - Request Body: None
            - Response Body: Song[]
        - Put
            - Request Body: Song
            - Response Body: songId (type UUID)
        - Post
            - Request Body: Song
            - Response Body: songId (type UUID)
    - /api/v1/songs/{songId}
        - Get
            - Path parameters: songId (type UUID)
            - Request Body: None
            - Response Body: Song
        - Delete
            - Path parameters: songId (type UUID)
            - Request Body: None
            - Response Body: None
- Finder:
    - /api/v1/finder/matches
        - Get
            - Request Body: None
            - Response Body: Match[]
        - Put
            - Request Body: Match
            - Response Body: None
        - Post
            - Request Body: Match
            - Response Body: None
    - /api/v1/finder/matches/{matchId}
        - Get
            - Path parameters:matchId (type UUID)
            - Request Body: None
            - Response Body: Match
        - Delete

            - Path parameters: matchId (type UUID)
            - Request Body: None
            - Response Body: None
    - /api/v1/finder/{userId}/{concertId}
        - Post
            - Path parameters: userId (type UUID), concertId (type UUID)
            - Request body: None
            - Response: MatchWithHATEOASLinks

- Concert:
    - /api/v1/concerts
        - Get
            - Request Body: page (Optional Integer), size (Optional Integer)
            - Response Body: Concert[]
        - Put
            - Request Body: Concert
            - Response Body: None
        - Post
            - Request Body: Concert
            - Response Body: None
    - /api/v1/concerts/{concertId}
            - Get
                - Path parameters: concertId (type UUID)
                - Request Body: None
                - Response Body: Concert
            - Delete
                - Path parameters: concertId (type UUID)
                - Request Body: None
                - Response Body: Concert
    - /api/v1/concerts/{concertId}/usersInfo
        - Get
            - Path parameters: concertId (type UUID)
            - Request Body: None
            - Response Body: UserIdsWithStatuses
    - /api/v1/concerts/{concertId}/usersInfo/{userId}
        - Put
            - Path parameters: concertId (type UUID), userId (type UUID)
            - Request Body: None
            - Response Body: None
        - Delete
            - Path parameters: concertId (type UUID), userId (type UUID)
            - Request Body: None
            - Response Body: None
    - /api/v1/concerts/TicketmasterSync

        - Put
            - Path parameters: None
            - Request Body: None
            - Response Body: None
- Aggregator:
    - /{userId}/{concertId}
        - Get
            - Path parameters: concertId (type UUID), userId (type UUID)
            - Request Body: None
            - Response Body: UserConcertDetails[]

Data types:
User:

```JavaScript
{
  "id": "130bd19b-92b6-4dd7-90ef-c477bca9a824",
  "name": "someUser",
  "dateOfBirth": "1998-10-22",
  "age": 25,
  "email": "u1@gmail.com",
  "password": "pwd",
  "profilePictureUrl": "u1.com"
}
```

Song:

```JavaScript
{
  "id": "0f87a8b4-cc5f-4c36-b116-fd8c50855d08",
  "name": "someSong",
  "artist": "someArtist",
  "genre": [
    "Genre1",
```

```
      "Genre2"
    ]
  }
```

SongsWithHATEOASLinks:

```JavaScript
{
  "_embedded": {
    "songList": [
      {
        "id": "0f87a8b4-cc5f-4c36-b116-fd8c50855d08",
        "name": "someSong",
        "artist": "someArtist",
        "genre": [
          "Genre1",
          "Genre2"
        ],
        "_links": {
          "self": {
            "href":
"http://ec2-3-14-85-109.us-east-2.compute.amazonaws.com:8012/api/
v1/songs/0f87a8b4-cc5f-4c36-b116-fd8c50855d08"
          }
        }
      }
    ]
  }
}
```

Concert:

```JavaScript
  {
    "id": "0f87a8b4-cc5f-4c36-b116-fd8c50855d08",
    "name": "someSong",
```

```
   "performingArtist": "someArtist",
   "DateTime": "2023-12-1"
   "genre": "Genre1",
    "subGenre": "Genre2",
    "venue": "sampleVenue",
    "cost": 123
 }
```

UserIdsWithStatuses:

```JavaScript
[
     {
          "userId": "3e55bd46-7125-11ee-b962-0242ac120002",
          "status": "INTERESTED"
     },
     {
          "userId": "3e55c2c8-7125-11ee-b962-0242ac120002",
          "status": "ATTENDING"
     }
]
```

Match:

```JavaScript
{
      "userId": "3e55bd46-7125-11ee-b962-0242ac120002",
      "concertId": "3e55bd46-7125-11ee-b962-0242ac120002",
      "matchedUserIds": [
          "3e55bd46-7125-11ee-b962-0242ac120002"
     ]
}
```

MatchWithHATEOASLinks:

JavaScript

```
{
  "_embedded": {
    "match":
      {
        "userId": "3e55bd46-7125-11ee-b962-0242ac120002",
       "concertId": "3e55bd46-7125-11ee-b962-0242ac120002",
       "matchedUserIds": [
           "3e55bd46-7125-11ee-b962-0242ac120002"
      ],
        "_links": {
          "self": {
            "href":
"http://ec2-3-14-85-109.us-east-2.compute.amazonaws.com:8012/api/
v1/songs/0f87a8b4-cc5f-4c36-b116-fd8c50855d08"
          }
        }
      }
  }
}
```

UserConcertDetails

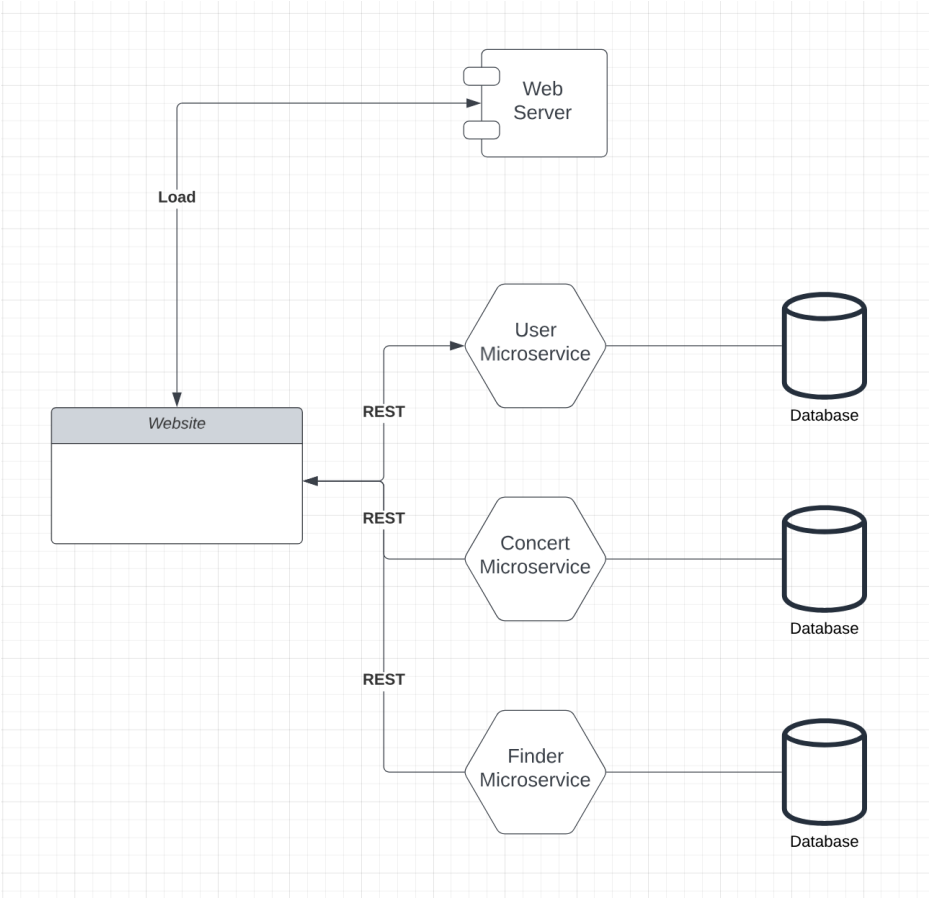
JavaScript

```
{
 "info": {
  "id": "0e21d65c-203a-4ba8-88f6-06cac7a0a2ca",
  "name": "test-user",
  "dateOfBirth": "2000-11-24",
  "age": 23,
  "email": "zh2603columbia.edu",
  "profilePictureUrl": "string"
 },
 "songs": [
  {
   "id": "0697c647-3a4d-4128-8f86-e7a172f9606d",
   "name": "Apoptosis",
```

```
    "artist": "Official Hige Dandism",
    "genre": [
     "Rock",
     "J-Pop"
    ],
    "link":
"http://ec2-13-59-208-208.us-east-2.compute.amazonaws.com:8012/ap
i/v1/songs/0697c647-3a4d-4128-8f86-e7a172f9606d"
  },
  {
    "id": "bd1b8c74-1a77-4cd3-ab4b-09be493cc935",
    "name": "Matsuri",
    "artist": "Fujii Kaze",
    "genre": [
     "J-Pop"
    ],
    "link":
"http://ec2-13-59-208-208.us-east-2.compute.amazonaws.com:8012/ap
i/v1/songs/bd1b8c74-1a77-4cd3-ab4b-09be493cc935"
  },
  {
    "id": "7ea06daa-b0c1-43f3-b84c-553a64628818",
    "name": "From The Start",
    "artist": "Laufey",
    "genre": [
     "Jazz"
    ],
    "link":
"http://ec2-13-59-208-208.us-east-2.compute.amazonaws.com:8012/ap
i/v1/songs/7ea06daa-b0c1-43f3-b84c-553a64628818"
  },
  {
    "id": "dc17081f-9e90-469e-8eba-117f619de656",
    "name": "Congratulations",
    "artist": "Post Malone",
    "genre": [],
```

```
   "link":
"http://ec2-13-59-208-208.us-east-2.compute.amazonaws.com:8012/ap
i/v1/songs/dc17081f-9e90-469e-8eba-117f619de656"
  },
  {
   "id": "c0501637-6968-44ba-8eb7-daf0393ce577",
   "name": "test-song-2",
   "artist": "test-artist-2",
   "genre": [
    "test-genre-2"
   ],
   "link":
"http://ec2-13-59-208-208.us-east-2.compute.amazonaws.com:8012/ap
i/v1/songs/c0501637-6968-44ba-8eb7-daf0393ce577"
  }
 ],
 "concert": {
  "id": "b392da9d-3d11-4d2d-98e1-6219a9a4f056",
  "name": "P!NK: TRUSTFALL TOUR",
  "venue": "Amway Center,400 W Church St.",
  "performingArtist": "P!NK, GROUPLOVE",
  "dateTime": "2023-11-19",
  "genre": "Rock",
  "subGenre": "Pop"
 },
 "matches": {
  "id": "24aa2216-c9e1-4189-818b-8b81d008e1cb",
  "userId": "0e21d65c-203a-4ba8-88f6-06cac7a0a2ca",
  "concertId": "b392da9d-3d11-4d2d-98e1-6219a9a4f056",
  "matchedUserId": [
   "0e21d65c-203a-4ba8-88f6-06cac7a0a2ca",
   "2f549ace-7ce8-466e-b9c4-b973f2bb69bc",
   "059159f4-6c21-41b3-b603-6ce93fac3647"
  ]
 }
}
```

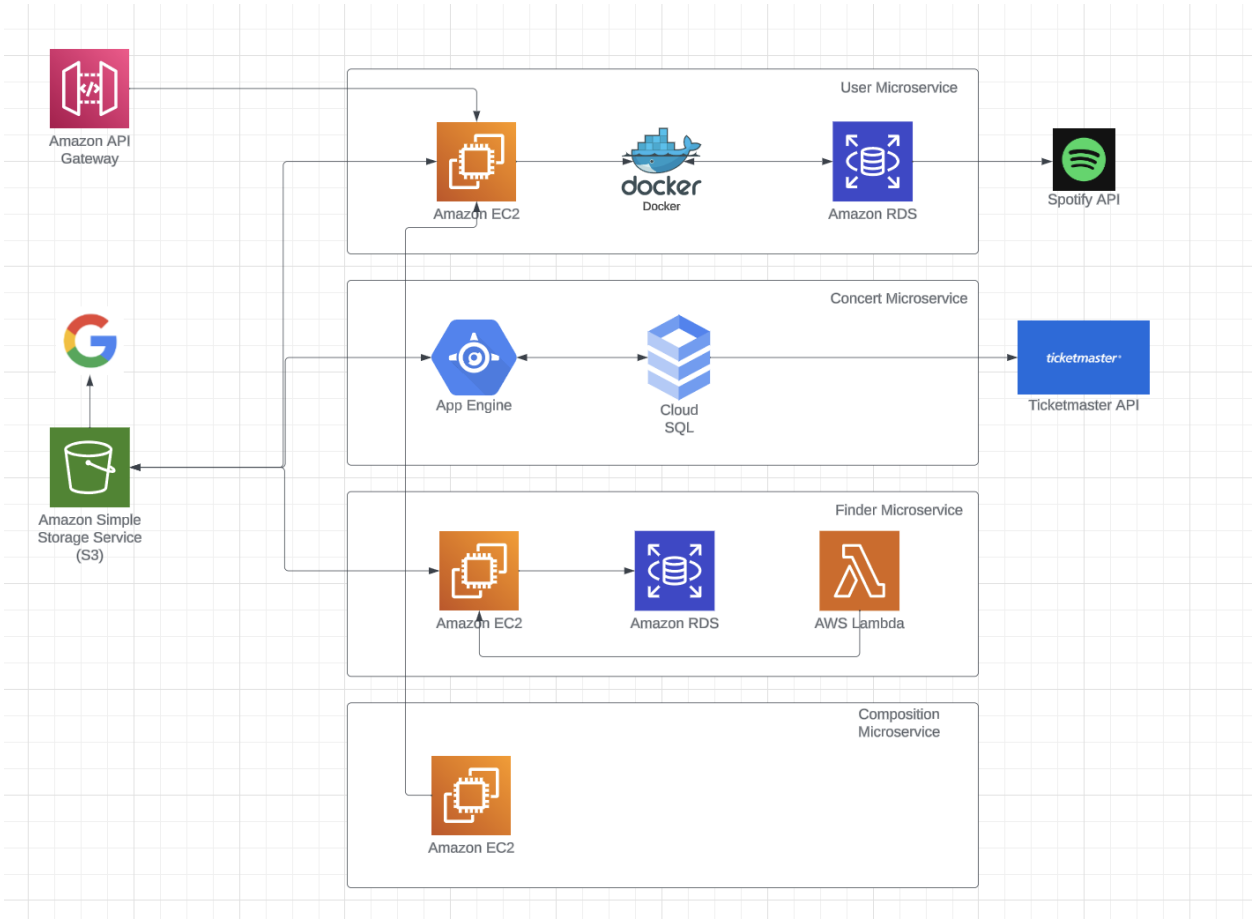
Logical View



The website is loaded by a web server and it communicates with three microservices using REST calls:

1. User Microservice: Interacts with a PostgreSQL database deployed on RDS. This microservice manages user-related information.
2. Concert Microservice: Manages details related to concerts and also interacts with its PostgreSQL database deployed on Google Cloud Datastore.
3. Finder Microservice: Facilitates users in finding suitable matches to attend a concert based on their music interests, storing information using PostgreSQL database deployed on RDS.

Physical View



Amazon S3 Bucket: Store the ConcertBuddy UI website that uses Google Single-Sign On

Amazon API Gateway: Encapsulate the User Microservice API's endpoints

User Microservice:
Hosted on Amazon EC2 with Docker, and stores user data and the users' preferred songs information in Amazon RDS. User and song related actions are executed in this microservice. It can also call third party APIs like Spotify to sync the users' preferred songs.

Concert Microservice:
Hosted on Google App Engine and stores data in the Cloud Datastore. Interactions in the website that relates to concerts are executed by this microservice. It calls third party APIs like TicketMaster to get concert information.

Finder Microservice:
Hosted on Amazon EC2 and stores user matches for a concert in Amazon DynamoDB. The matching algorithm to find other users to go to a concert with is implemented here.

Aggregator Microservice:
Hosted on Amazon EC2 and queries user profile, song list, concert info, and user matches when necessarily. The microservice uses a combination of synchronous and asynchronous calls for max efficiency.

Codebase

ConcertBuddy frontend UI (including SSO): https://github.com/Foo2000/ConcertBuddyUI
User microservice (including Spotify API, filtering, GraphQL, and API Gateway):
https://github.com/Foo2000/concertbuddyuser
User microservice docker image:
https://hub.docker.com/repository/docker/zfmyac/concertbuddyuser-x64/general
Concert microservice (including CI/CD Ticketmaster API):
https://github.com/Yutao-Zhou/concertbuddyconcert
Finder microservice (including Events / Notifications Pub/Sub and Middleware):
https://github.com/fei-uwu/concertbuddyfinder
User profile microservice (composition microservice):
https://github.com/Rootofallevil/Concert-Buddy-aggregator
Infrastructure As Code for composition microservice:
https://github.com/Rootofallevil/Concert-Buddy-aggregator/blob/master/terraform/main.tf
Github Project Board: https://github.com/users/Zhao-YS/projects/4

Links to deployment

ConcertBuddy UI Website (including SSO) hosted on S3:
http://concertbuddyui.s3-website.us-east-2.amazonaws.com/
User service (including Spotify API and filtering) deployed on docker on ec2:
http://ec2-18-224-179-229.us-east-2.compute.amazonaws.com:8012/api
User service GraphQL path deployed on docker on ec2:
http://ec2-18-224-179-229.us-east-2.compute.amazonaws.com:8012/api/v1/graphiql
User service API Gateway protects /users endpoint:
https://m5jadost45.execute-api.us-east-2.amazonaws.com/dev/api/v1/users
Concert service (including CI/CD, TicketMaster API, and pagination) deployed on Google app engine: http://concertbuddyconcert.uc.r.appspot.com/api
Finder service (including Events / Notifications Pub/Sub and Middleware) deployed on ec2:
http://ec2-18-219-191-243.us-east-2.compute.amazonaws.com:8080/api
Composition Microservice (including Infrastructure As Code) deployed on ec2:
http://ec2-107-20-22-151.compute-1.amazonaws.com:8000/docs

Link to Demo Video

https://drive.google.com/file/d/150V5lJMHjW3Ov1D5tQ5pPnTvp1_sn_ZM/view?usp=drive_link

Link to Code File

https://drive.google.com/file/d/16in3iEDrKera2hF9a4mJ1jpUWMTyyzQ4/view?usp=sharing