# Shazam

Trending Movie Trailers and Rating System

CS-GY 9223 : Cloud Computing

Nikhil Nar (ncn251)
Suraj Gaikwad (sgg339)
Chinmay Wyawahare (cnw282)

**Abstract:**

Shazam is a web portal similar to Netflix where users can watch trailers of various movies. The trailers that will be displayed are fetched from youtube dataset. The users can also rate the youtube trailers. Users will be given a choice to watch the trending videos and highly rated movies through a filter. The entire project is deployed on AWS and comprises of various technologies like EC2, DynamoDB, ElasticSearch, Lambdas, SQS, Python, Node.js, PySpark.

**Project Idea and Implementation Plan:**

The idea is to develop topics in SQS for storing the views of the trailers and the rating given to the trailer by various users. Whenever the user watches a trailer, a corresponding entry is pushed into the Queue. Similarly when the user rates a trailer, the corresponding entry is pushed into the Queue. The consumers polling the Queue will store these views (a lambda Function) and ratings in mongoDB. These entries are then further aggregated in Apache Spark and stored in mongoDB. These new entries will then be used to display the highly rated and trending movies trailers. A lambda function will continuously run for multiple times in a day which will execute the aggregation script in Apache Spark. Further, ElasticSearch will be used to index the movie trailers and will be used for searching mechanism on the website. The website will be developed in Node.js for backend and HTML, CSS, and Javascript for UI.

# Contents

# 1. Introduction

It is estimated that there are about 500,000 movies currently in existence. That's more than anyone can watch in a lifetime. Moreover, not all people will like every movie that has ever been made. Most of the times, people tend to watch movies from a few specific genres. Hence, for any given user, most of the movies become irrelevant.

Every once in a while, we all try to look for something new to watch. Many people ask their friends, family and colleagues for suggestions and then watch a movie. But this is not always feasible. Sometimes, we would just want to check the top rated movies in a specific genre or the trending movies around us. This is where Shazam steps in.

Shazam makes it easy for people to look for trending and top rated movies, along with their trailer. It also enables the user to provide a rating to one of the movies that they have watched to help others decide. Shazam is a large-scale distributed system that provides trending movies and top rated movies which is updated twice a day. This keeps the user updated with the latest and greatest movie trailers from all over the world.

# 2. Methodology

## 2.1 Technology Stack

The web server to serve requests is written in Node.js. The user interface was developed by using HTML, CSS and Javascript. To handle multiple concurrent requests, SQS is used as a messaging queue. MongoDB is used for data storage of movies, ratings, views, user profile. Apache Spark is used for aggregation of user views and ratings to provide highly rated and top trending movies. Docker is used to run all the components of Shazam like SQS, MongoDB, Node.js, Apache Spark with AWS serverless application.

## 2.2 Process Flow

Through Shazam, users will be able to watch movie trailers. Users will also be given an option to watch movie trailers of top trending and highly rated movies. Whenever a movie trailer is released, million users starts watching the trailer simultaneously. To handle these million concurrent requests Apache Kafka is used as a messaging queue. In this model, all the users watching the movie trailers will act as the publishers. There are multiple consumers in a consumer group which are subscribed to the topics which further processes the requests and stores the user ratings and user views in MongoDB. Since the total number of user views and corresponding ratings for the movies will be enormous, performing aggregation on MongoDB will require a lot of time. To solve this problem, Apache Spark is used for performing aggregation. A cron job is used to run the Spark job which will fetch the data from MongoDB, perform aggregation and store the aggregated data back in MongoDB. The aggregated data is then displayed to the user on Shazam as top trending and highly rated movies.

### 2.3 Architecture

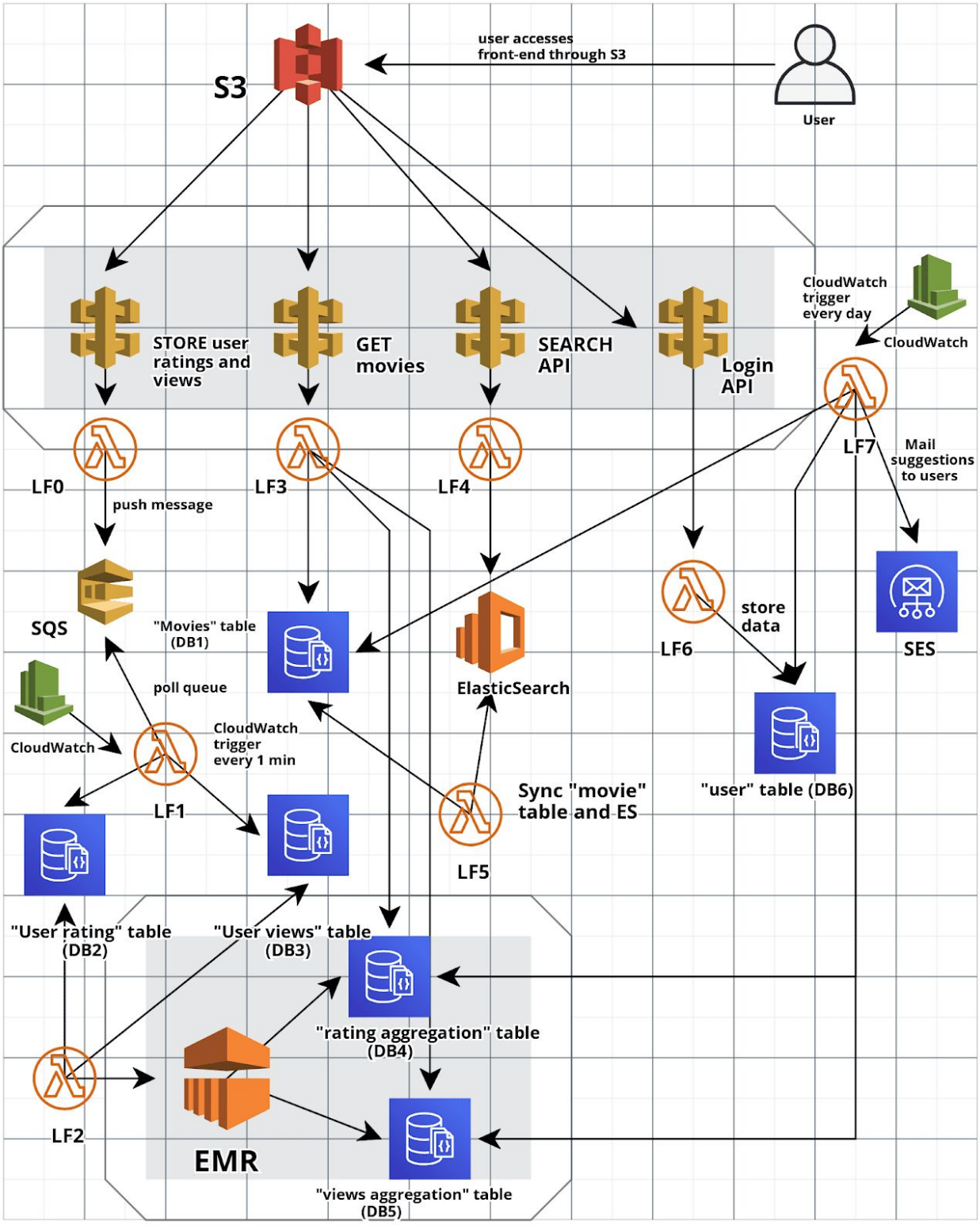


Figure 1: Shazam schema

Services like Apache SQS, Apache Spark, ElasticSearch, etc run under the hood. Whenever a user watches a movie trailer, the information of (movie_id, user_id) is published into movie_users_mapping topic. Similarly, when a user rates the movie trailer then the information of (movie_id, user_id, rating) is published into movie_ratings topics. Multiple consumers in a consumer group are subscribed to both the topics which fetches the stored information in SQS and stores the information in MongoDB collections. Apache Spark will then aggregate this data and will return an array of JSON with schema (movie_id, views) for trending topics and (movie_id, avg_rating) for highly rated movies. This data is stored in trending_movies and top_rated_movies collections in MongoDB. Apache Sparks integrates with MongoDB using Mongo-Spark connector. The activity of aggregating data in Spark is performed two times in a day using a CRON job. The architecture of the system is shown in figure 1.

### 2.4 API Design

1. POST /ratings : To store the ratings given by the user
2. POST /views: To store the views of the user
3. GET /movies: Returns the movie list based upon filters
4. GET/search/movies: Returns the movies from the ElasticSearch
5. GET /login: To get login token
6. POST /register: To register the user

## 2.5 Data Storage Schema

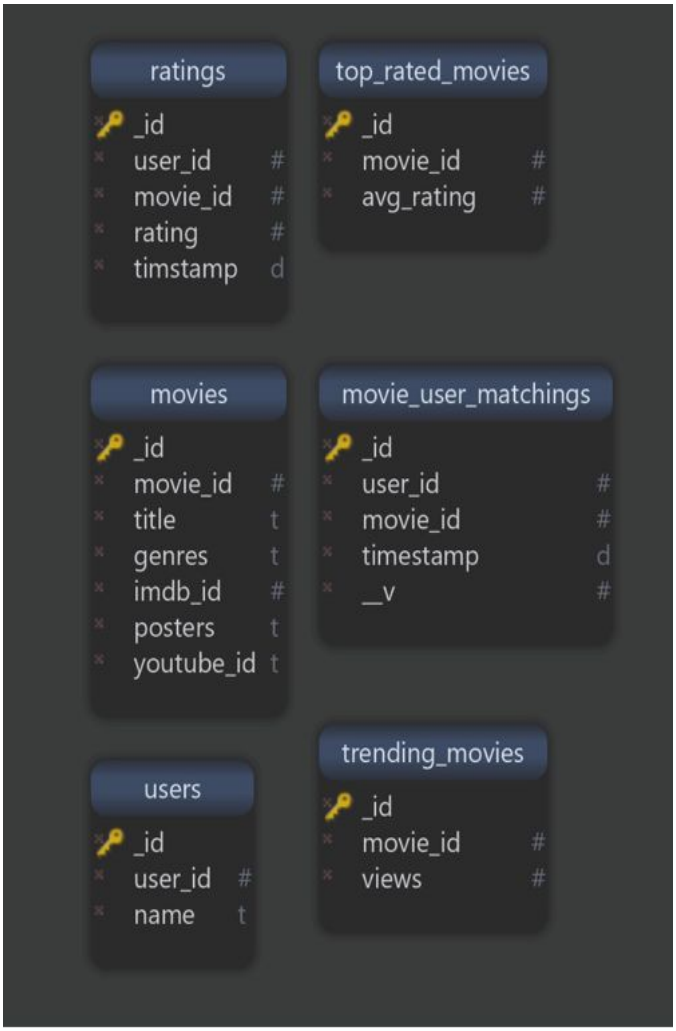


Figure 2: MongoDB schema

A NoSQL database was needed for easy fetching and fast processing of data. Shazam uses MongoDB as the NoSQL database to store the movies, user ratings, youtube videos links, movie posters, and user views. Figure 2 shows the detailed schema of MongoDB with each field name. Consumers reading data from the Kafka topics will store the user views in movie_user_matchings collection and movie ratings in ratings collection. Apache Spark will then read the data from movie_user_matchings collection to find the top trending movies and ratings collection to find the highly rated movies.

### 2.6 Dataset

For movies and ratings, we are using dataset from grouplens.org 'MovieLens 20M' dataset.
For youtube trailers, we will be using the grouplens.org MovieLens 20M Youtube Trailers dataset.
We will use a separate dataset from Kaggle to fetch movie posters.

The links for the datasets are given below:

https://grouplens.org/datasets/movielens/20m
https://grouplens.org/datasets/movielens/20m-youtube
https://www.kaggle.com/neha1703/movie-genre-from-its-poster

### 2.6 AWS Services

1. S3
2. SQS
3. ElasticSearch service
4. DynamoDB
5. EMR
6. API Gateway
7. SES
8. Cognito
9. CloudFormation
10. Lambdas

## 2.7 Screenshots

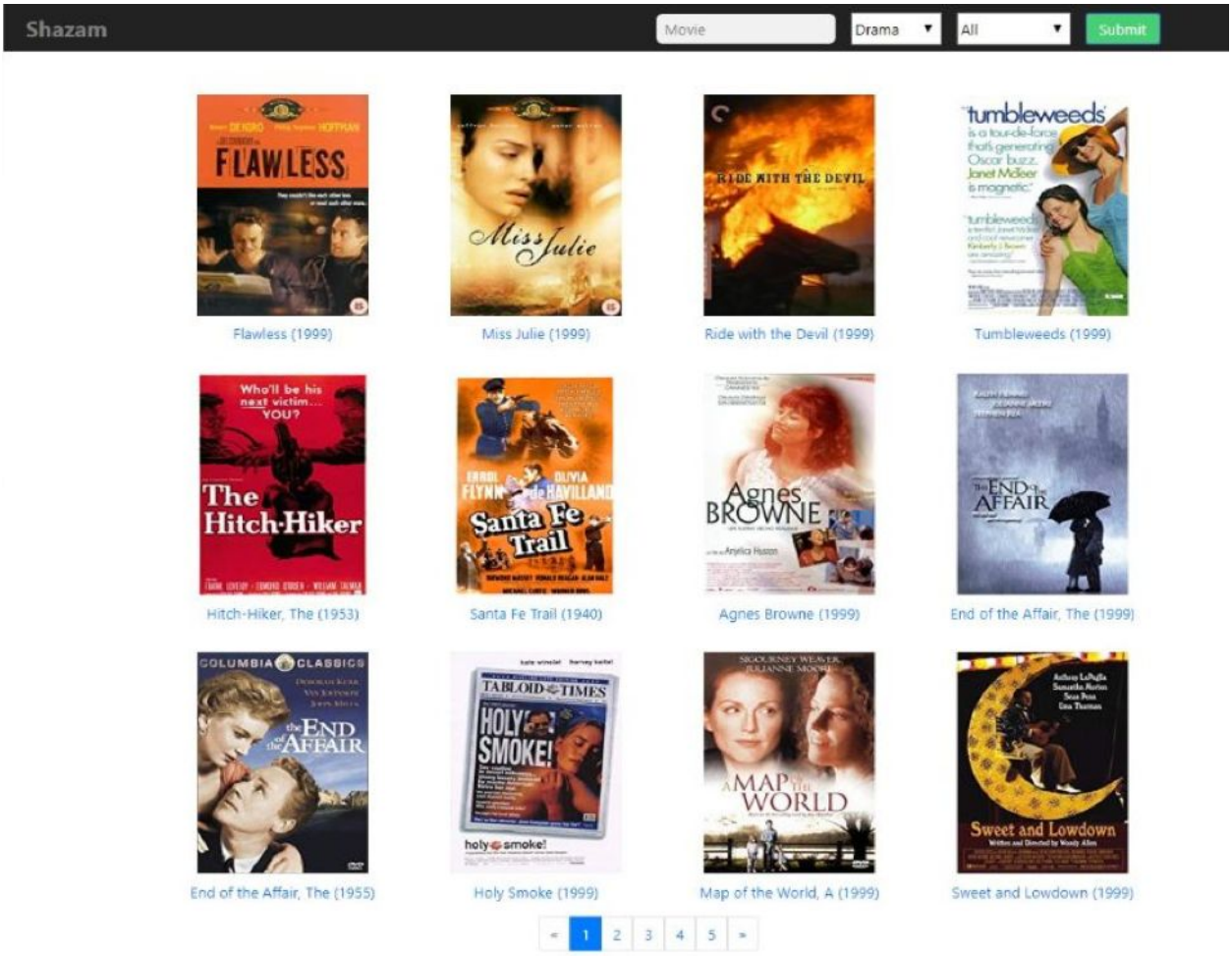


Figure 3: Home page of Shazam

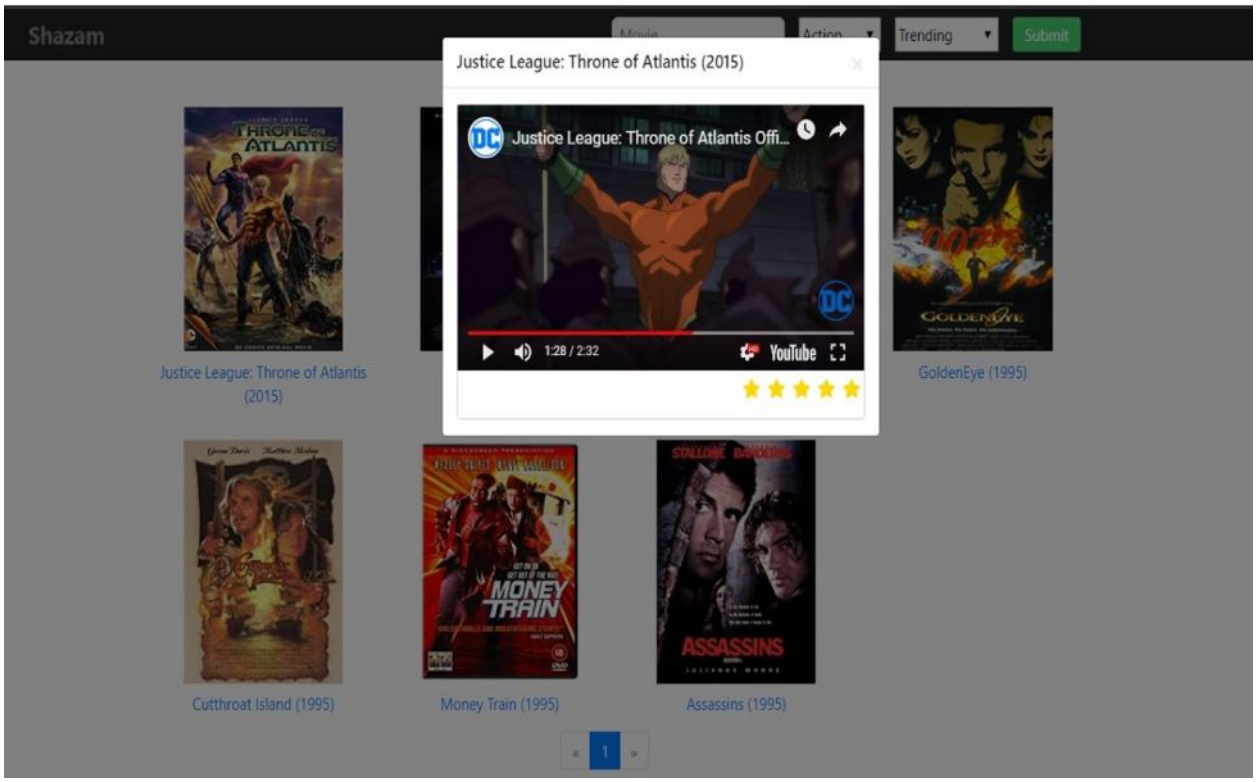


Figure 4: Movie Trailer display in Shazam

## 3. Challenges

There were many challenges we faced during this project. One of the major challenges was to merge multiple datasets to create a new, complete data set that we needed. The main dataset that we first came across only had the ratings of the movies, along with movie and user data. We had to merge this dataset with two other datasets that contained information such as the youtube trailer links and the poster links.

Another challenge that we faced during this project was to decide upon a specific architecture that should be used for this specific project. There were many viable options for this and we finally ended up choosing the architecture that we've discussed before.

## 4. Takeaways

One of the main takeaways of the project was to make the system scalable and run in a distributed environment. This is because there are multiple new movies coming out every single week and the data to this will keep accumulating over time. We have used technologies that are scalable in each part, and does not have any bottleneck anywhere in the architecture. Since the architecture also runs in a distributed manner, processing of huge amounts of data is not much of a hassle. We learned how to design scalable architecture to handle big data.

# 5. Future Work

There are multiple features that can be added to this project in the future. If we are able to get the location data of the users, there's a lot more that we can do to provide a better experience. For one, we can add localized trending and top videos. This will help provide more relevant data to the user, such as local language movies. One more functionality that can be added is to provide showtimes of released or upcoming movies around the user. This will provide an added functionality to help user watch a trailer and also look at upcoming showtimes.

Another improvement that we would like to add is to migrate the data from mongoDB to DocumentDB. This will open up multiple avenues. One of the main thing is that we can provide is search functionality given keywords, names of actors etc.

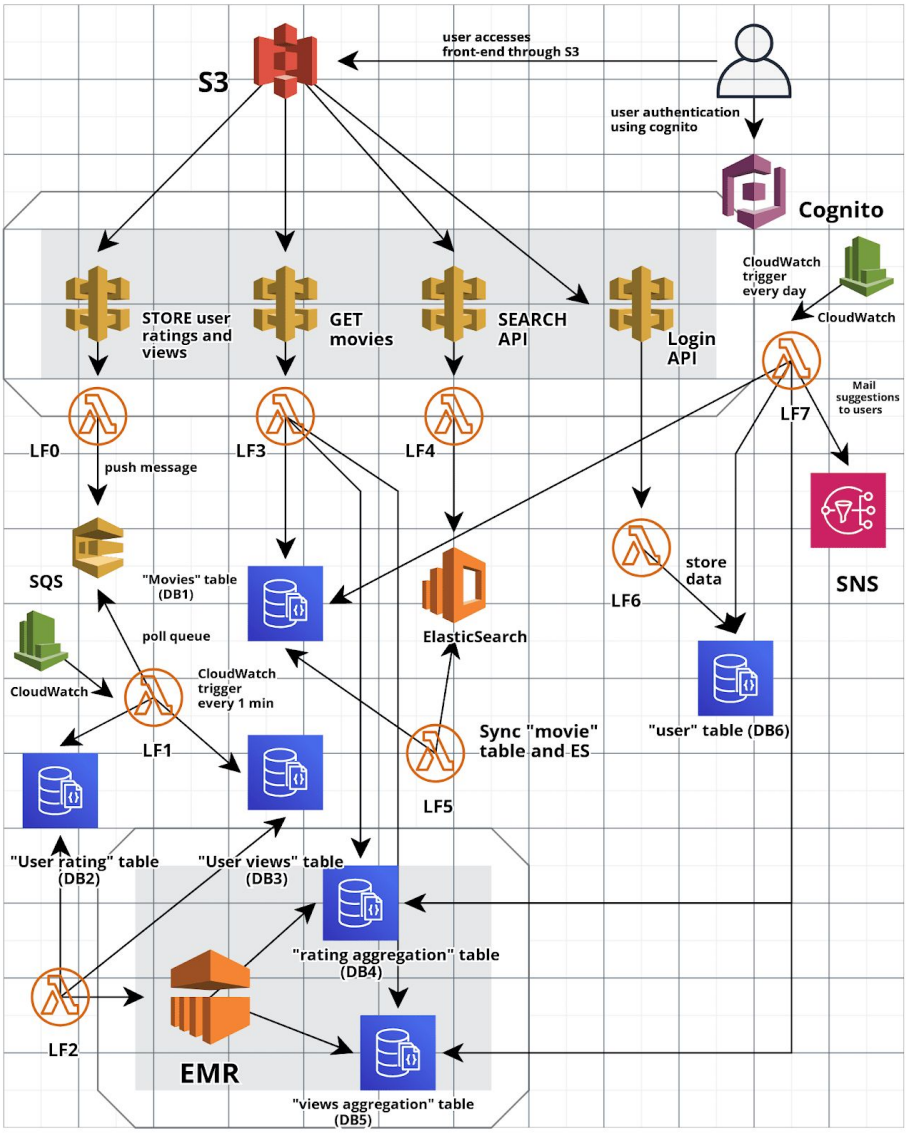


Figure 5: Updated Architecture

## 6. Conclusion

Shazam is successfully providing a scalable, distributed, self updating system that provides the latest data for trending and top rated movies. The data accumulated from the user is stored onto mongoDB with the help of Kafka and multiple consumers. This is then aggregated twice a day with the help of a cronjob that runs pySpark and SparkSQL commands and is written back onto mongoDB. When the user refreshes the page, this provides the latest trending and top rated movies.

## 7. Code

The code can be found here:
https://github.com/NikhilNar/Shazam

## 8. References


1. https://grouplens.org/datasets/movielens/20m/
2. https://grouplens.org/datasets/movielens/20m-youtube/
3. https://www.kaggle.com/neha1703/movie-genre-from-its-poster/downloads/movie-genre-from-its-poster.zip/5
4. https://nodejs.org/en/docs/
5. https://kafka.apache.org/documentation/
6. https://docs.mongodb.com/
7. https://spark.apache.org/docs/latest/
8. https://docs.mongodb.com/spark-connector/master/python-api/