Analytics Service on IMDB Data

Title: Development of API Interface for Analytics Service on IMDB Data

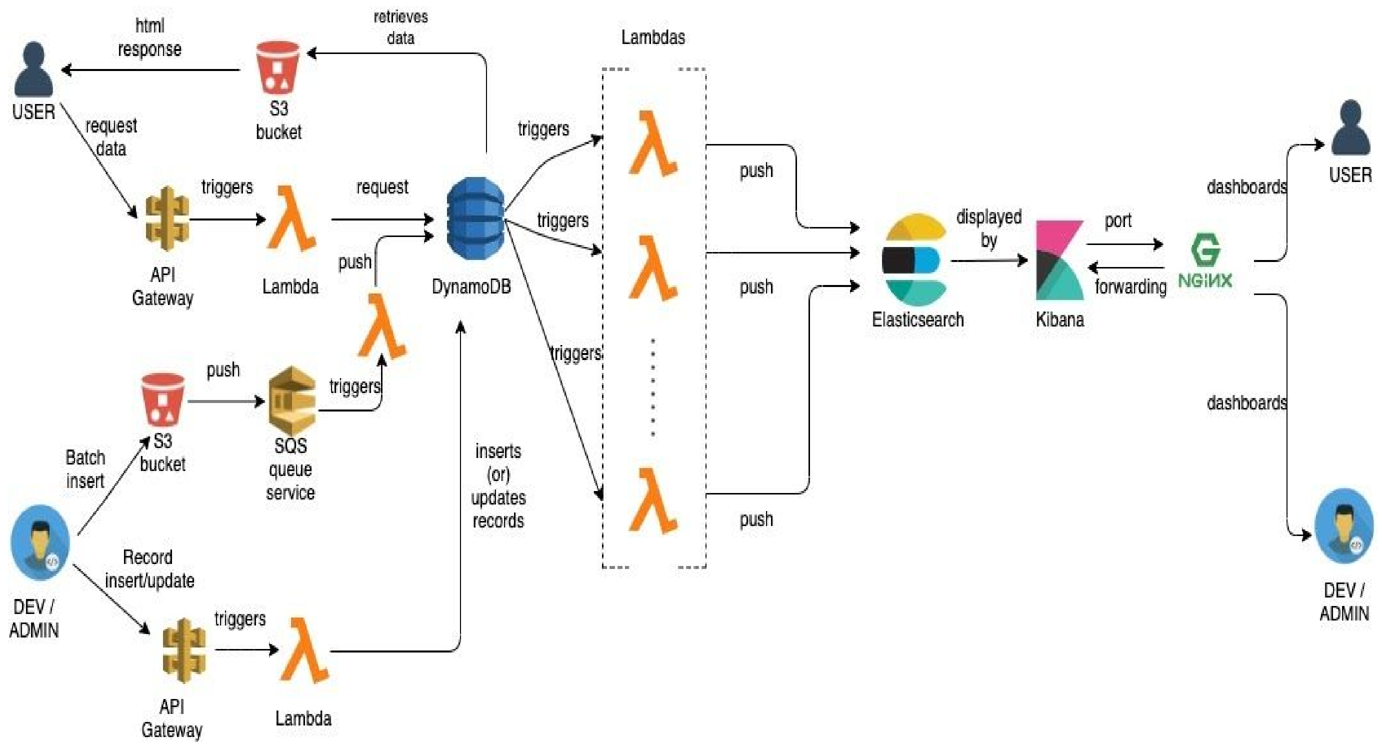
Project Participants: Abhinivesh Palusa, Nikhil Prabhu

Project Goals: IMDB data doesn't provide an official documented API. The main goal of our project is to create an API for describing shows (titles), describing people working in those shows, and providing analytics on already available data (sourced from IMDB interfaces website [1]) and also to update these analytics if either new data is inserted or existing data is updated.

Software and Hardware Components:

- AWS Services:
  - S3 buckets
  - API Gateway
  - Serverless Lambda functions
  - SQS (Simple Queue Service)
  - DynamoDB
  - EC2 instances
  - Cloudwatch logs
- Elasticsearch
- Kibana
- Nginx

Architectural Diagram:



Interaction between Components:

- Developer or DB Admin can perform a batch insert of IMDB data in the form of tsv files available at [1], into a specific S3 bucket where the information is stored (preferably in JSON format). A lambda function is triggered whenever data falls into the S3 data storage. This lambda function helps to push the same data into the SQS queue along with the corresponding table name information as a message. The insertion of this message is an event which triggers a Lambda function. The Lambda function will help transform the message accordingly and transfer the same into one of the DynamoDB tables.
- S3 storage is being used because the data available in the form of tsv files (the entire batch) is pretty huge in file size and such files or data are preferably stored in the S3 storage buckets as any amount of data can be stored and retrieved from S3 buckets, at anytime, from anywhere on the web. AWS S3 gives any developer access to the same highly scalable, reliable, fast and inexpensive data storage that Amazon uses to run its own global network of websites [2].
- AWS SQS is used as it processes tasks asynchronously [3] and our use-case requires that as we upload data in a batch which sends millions of records at a time via the lambda function from S3 to DynamoDB. To process each JSON message/record asynchronously and safely insert into DynamoDB, we store and process data in the intermediary standard AWS SQS.
- A single AWS Lambda function abstracts away the entire client-server architecture between any two components and just asks the developer to fill in what a request looks like and what a response should look like, by automatically managing the underlying compute services. So, for this simplicity feature making it comfortable for the developers to build services, lambda function services are being used in our use-case [4].
- Developer or DB Admin can also insert one record at a time by requesting via API Gateway. This complete request is also an event which triggers another Lambda function. This Lambda function is responsible for transforming the record accordingly and transferring the record (either insertion or updation) into one of the DynamoDB tables.
- Users can request information which is done via API Gateway. This complete request is again an event like the previous flow, but the Lambda function here is responsible for retrieving the data from one of the DynamoDB tables. The alternative is that the data is fetched from one of the DynamoDB tables and the message is stored in the S3 bucket as a HTML response which is publicly available to the requested User.
- An API gateway takes all API calls from clients, then routes them to the appropriate microservice with request routing, composition, and protocol translation. Typically it handles a request by invoking multiple microservices and aggregating the results, to determine the best path. It can translate between web protocols and web‑unfriendly protocols that are used internally [5].

- Whenever an insertion, updation or deletion occurs in one of the DynamoDB tables, which are events again triggering corresponding Lambda functions. These Lambda functions are responsible for transforming the information received and transferring into the respective Elasticsearch indices.
- The information available in various indices of Elasticsearch can be visualized using the Kibana UI. We use two indices for our project, titles and people which store the relevant information about shows and people worked in those shows respectively.
- Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements. This helps us search our requirements on Kibana dashboards and filter what is required to be searched for and what is to be opted out while searching [6].
- NGINX, a proxy server, is responsible for port forwarding making Kibana as the face of our Analytics service. These visualizations available on Kibana UI are viewable by Users and Developers as well. This NGINX service helps us forward any request coming onto port 80 to be redirected to port 5601 where our Kibana dashboards are being served.

Training and Testing/Debugging Mechanisms used:
Firstly, for training, our use-case doesn’t involve any Machine Learning service, but provides only analytics service which is mostly dependent on querying the databases accordingly and so the requirement for any training mechanism would be out of scope for our project. For testing, we created classes representative of each service that will facilitate easier development and better design. We created unit tests centered around these classes and services, including tests for batch uploads by creating a few mini-batches, API-based uploads and requests. In addition we performed integration tests between components to ensure overall working. Our system does not require a training phase. For testing the data in Elasticsearch, we used Kibana Dev tools where we can test data in each document via Insert API, Delete API, Bulk API, Count API, etc. For debugging purposes of each service and the data flow, we used AWS Cloudwatch logs to see whether the data is flowing as we intended and modify accordingly to our requirements.

Cloud Technologies used:
- Storage (AWS S3 service)
- Key-Value Stores (AWS DynamoDB service)
- Databases (Elasticsearch and AWS DynamoDB service)
- API Interfaces (AWS API Gateway service)
- Microservices (AWS Serverless Lambda functions)
- Message Queues (AWS SQS Service)
- Visualizations Display Interface (Kibana)
- Port Forwarding (NGINX)

We believe that our project will cover the project requirements and possibly even more. Our basic infrastructure already includes by necessity the use of Storage, API Interfaces, Databases including a Key-Value Store and Message queues.

Capabilities and Limitations of our Solution:

- Our solution primarily is limited by the AWS Educate service of the AWS where students get only 100 credits for free which limits our usage to almost all AWS services. This also provides a session time limit of 3 hours.
- These limited credits lead us to limited storage on AWS S3 (5 GB), limited number of non-polled messages present in and number of requests (million per month) to AWS SQS , limited number of triggers to lambda functions as our use-case currently uses 4 lambda functions, limited reads and writes to DynamoDB tables.
- The time limit for a session leads us to mini-batch (around 1 GB of records in 3 hours) upload of data instead of the entire batch (around 10 GB) of data from S3 to SQS to DynamoDB.
- Our solution can be extended to more visualizations on the Kibana dashboard where currently it consists of around 15 - 20 visualizations from both the indices.
- This can be extended to performing advanced data transformations while transferring data from S3 to SQS, or from SQS to DynamoDB, or from DynamoDB to Elasticsearch.
- There can be more GET methods to our API interface based on the different set of query parameters provided at the time of request.
- Currently, as we weren't able to store the entire IMDB data on both DynamoDB and Elasticsearch, we pushed only data where the shows were from 'en' language (English). So, we can extend it to the shows from all languages and push all relevant data into our databases.
- We can potentially be adding services involving container architecture as well in the future (Docker and Kubernetes).

References:

[1] https://www.imdb.com/interfaces/
[2] https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html
[3] https://searchaws.techtarget.com/definition/Amazon-Simple-Queue-Service-SQS
[4] https://aws.amazon.com/lambda/features/
[5] https://www.nginx.com/learn/api-gateway/
[6]
https://towardsdatascience.com/an-overview-on-elasticsearch-and-its-usage-e26df1d1d24a