**BIG DATA SYSTEM**

**Anubhav Gulati – 2022OG04004**

**Requirements:**

1/ **Design:** Design the backend services, APIs, data models and the databases you will use for this. The design should support both realtime stream processing and batch analytics use cases. Explain the rationale for your design especially tradeoffs wrt CAP theorem, describe the tech choices for the databases and other components. You must use a NoSQL database as part of your design (MongoDB, Neo4j, DynamoDb, HBase, anything appropriate for your solution).

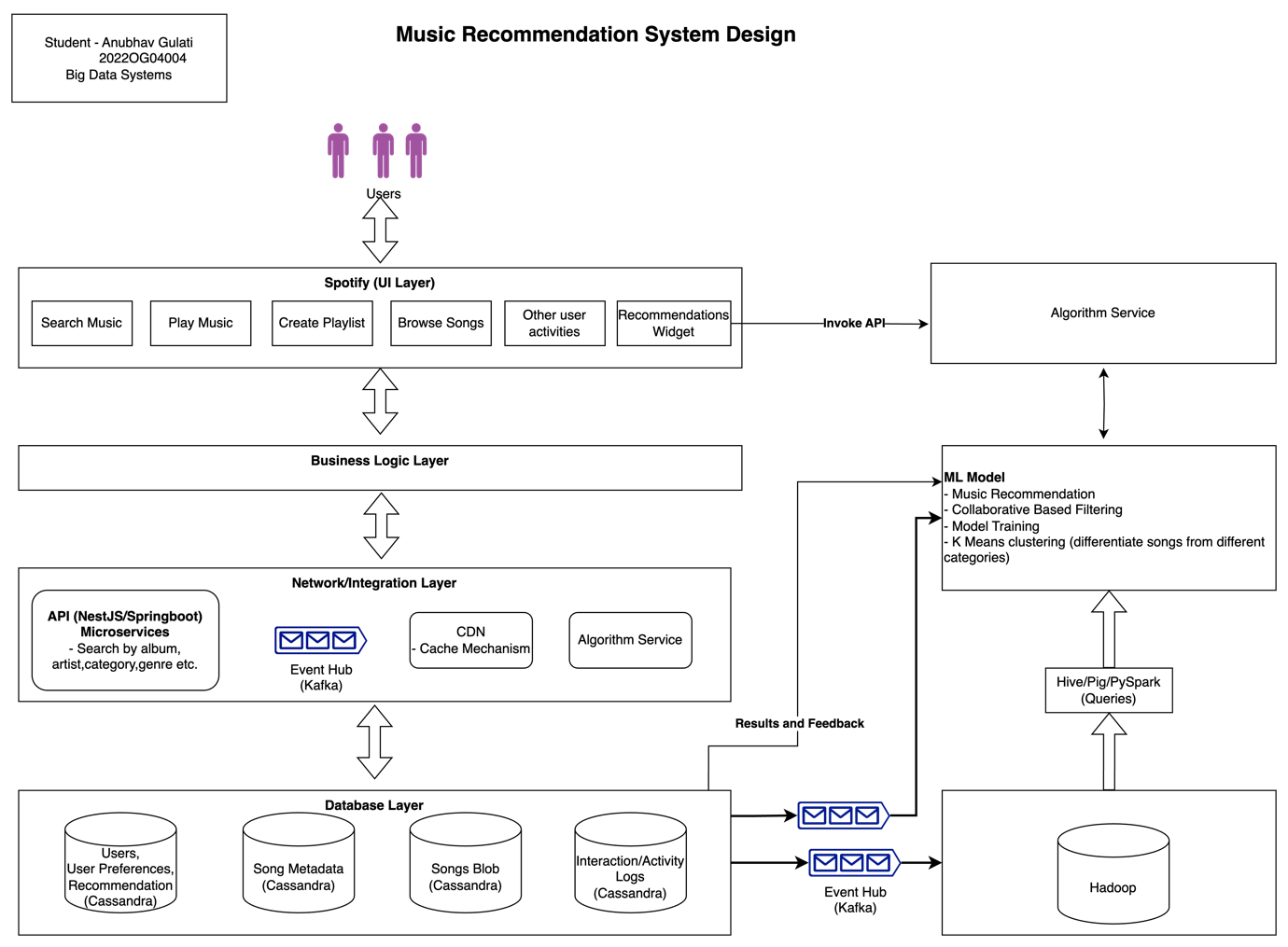
**PART 1 – Answer**

**GitHub link -**

[**https://github.com/anubhavgulatibits/BDSAssignment**](https://github.com/anubhavgulatibits/BDSAssignment)

**Recording link -**

[**https://www.loom.com/share/efffec5a20be44d1b6b933d60a8600fb?sid=92d015cb-e105-4641-997a-b9bfb38ed8eb**](https://www.loom.com/share/efffec5a20be44d1b6b933d60a8600fb?sid=92d015cb-e105-4641-997a-b9bfb38ed8eb)



Music recommendation system design is going to support both the real time stream processing and batch analytics use cases.

**Different architectural layers are shown in the diagram, starting from top – user login.**

* UI Layer – User can login on spotify app and can do multiple operations like search music, play the music, create playlist, browsing of songs and all other user interactions.
  + Also it displays the recommendation songs on the recommendation widget.
* Business Layer – UI and any business logic should be separated. UI should not have any business logic as it can degrade the performances.
* Network/Integration Layer – Just for diagram have shown it in one, but can be considered as 2 different layers.
  + CDN will make sure to help us in latency issues. Information can be cached to improve the latency and system should play next song or search and display the songs fast. That’s one of the key goals.
  + This layer also has all other integration with other systems either by functions developed on nestjs/spring boot and kafka integration too.
  + API functions – all different sort of API’s need to build to query the song information by different songs metadata. This is also deployed on cloud as Serverless functions.
* Database layer – All the information whether its about user, user preferences, what user likes, what user does not like, songs metadata, songs in blob storage, user interaction history gets stored in cassandra.

**Trade off –**

* CAP theorem – Consistency, Availability, Partition Tolerance
  + distributed system can deliver only two of three desired characteristics
* There are several NOSQL Databases available. It’s purely depend on the use case what we need, what’s our focus on.
* Mongo DB is CP system and Cassandara DB is AP system.
* In this case we are going with Cassandra DB, as it can easily store our data and availability is the most thing needed here rather than consistency.
* We are not doing intensive read writes, that’s another reason to go with Cassandra.

**Real Time Recommendation**

* Recommendation can be real time and batch based.
* Based on the user online search, all those key interactions will be logged and it will be fed to ML model, and based on the ML model design, collaborative based filtering, K means clustering, recommendation will be derived which will be shown to user on recommendation widget.
* Users action on recommendation songs whether user plays, or do not plays , all these interactions will be stored and feed to the ML Model to improve the user recommendations.
* Algorithm service – This service is used which will be invoked by UI layer to get the recommendation lists on UI. Behind the scene its invokes model and try to get the recommendations.

**Batch Recommendation**

* All day history and much more info data about the user, can be passed to big data system such as Hadoop via Kafka.
* Data analytics can be done on such large amount of data.
* Same ML model can be used for batch also and model will derive the recommendation which can be stored and based on this recommendation will be displayed to the user.

Users storage, preferences etc using cassandra for that. For scalability, having this solution on cloud env, serverless functions, CDN to avoid latency issues and spark for running analytical queries. All images and this document is attached on github.

**PART 2**

2/ **OLTP Queries:** Execute at least 3 key read/write queries against the NoSQL database as part of the solution with the optimizations and the consistency levels set in your queries.

Answer –

**Loom recording link** –

<https://www.loom.com/share/3fbf0b4fc9a14d3ca161ba38067cd036?t=236&sid=220f83f1-5a8a-411b-981c-d4ecab8b5474>

**GitHub link -**

[**https://github.com/anubhavgulatibits/BDSAssignment**](https://github.com/anubhavgulatibits/BDSAssignment)

**Partition key** by track\_genre should be used for faster access. Partition data based on each field, will be saved in same nodes in the cluster.

If we don’t do any other partioning, and just go by ‘id’ , it will not have any performance improvement. As goes by id will give me single row each time and data is distributed across different nodes too.

In Cassandra, **we should access the data by partition keys.**

In Cassandra, a partition key is explicitly defined as part of the table schema.

* Creating keyspace

CREATE KEYSPACE IF NOT EXISTS final\_recommendation WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': 3};

* Creating table

CREATE TABLE IF NOT EXISTS final\_tracks (id INT, track\_id TEXT, artists TEXT, album\_name TEXT, track\_name TEXT, popularity INT, duration\_ms INT, explicit BOOLEAN, danceability FLOAT, energy FLOAT, key INT, loudness FLOAT, mode INT, speechiness FLOAT, acousticness FLOAT, instrumentalness FLOAT, liveness FLOAT, valence FLOAT, tempo FLOAT, time\_signature INT, track\_genre TEXT,

**PRIMARY KEY ((track\_genre,artists))** );

Track genere is acting as *partition key and artists as clustering key.*

* Copying the data from my local to table Anubhav\_tracks

COPY anubhav\_tracks (id, track\_id, artists, album\_name, track\_name, popularity, duration\_ms, explicit, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time\_signature, track\_genre)

FROM '/Users/anubhav/Documents/3rd Sem/BDS/train.csv’ WITH HEADER = true;

* Inserting new data

insert into anubhav\_tracks (id, track\_id, artists, album\_name, track\_name, popularity, duration\_ms, explicit, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, time\_signature, track\_genre) values (3,'1234','Kishore Kumar','Padosan','Mere Saamne waali khidki mein..', 80, 200000, false, 0.7, 0.9, 2, -5.5, 1, 0.1, 0.2, 0.0, 0.6, 0.8, 120.0, 5,'old songs');

* Selecting the data based on partition key track\_genre

SELECT \* FROM anubhav\_recommendation.anubhav\_tracks WHERE track\_genre = ‘old songs’ ALLOW FILTERING;

* CONSISTENCY

It is set to one

* We can select based on different consistency level too, depending upon use case. We have values like ONE, ALL,ANY,LOCAL\_ONE, TWO,THREE, QUORUM, LOCAL\_QUORUM, EACH\_QUORUM.

3/ **OLAP Analytics Queries:**Within the broader context of this problem (Spotify) you need to perform analytics queries from the dataset to driver better Sales / Growth. For this, setup a big data batch processing environment (Hadoop or a Spark), share the config. Then perform at least 3 queries in the form of Map-Reduce jobs or Pig / HiveQL / SparkSQL queries and share the responses.

**PART 3 - Answer**

Loom video link - <https://www.loom.com/share/8b183004604a4cbaa6ade92d84f2b184?sid=4ccaccc8-cd78-4b33-a781-66005decf423>

GitHub link –

<https://github.com/anubhavgulatibits/BDSAssignment>

Notebook link, Pyspark queries - <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8361694294635267/1406079655828555/8265960166006863/latest.html>

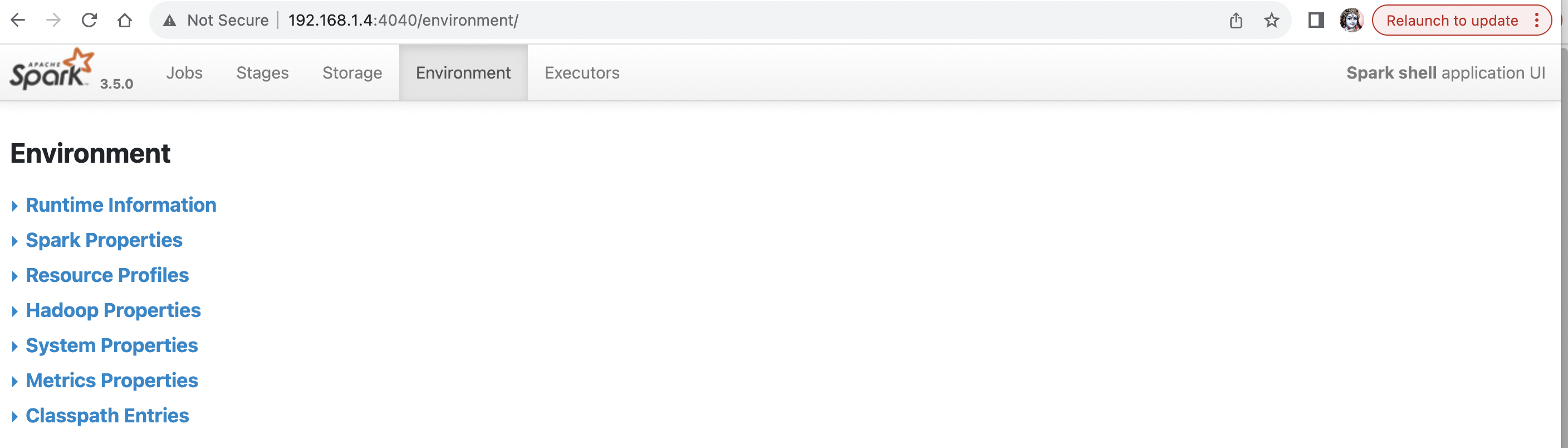
Done Spark set up in local successfully.



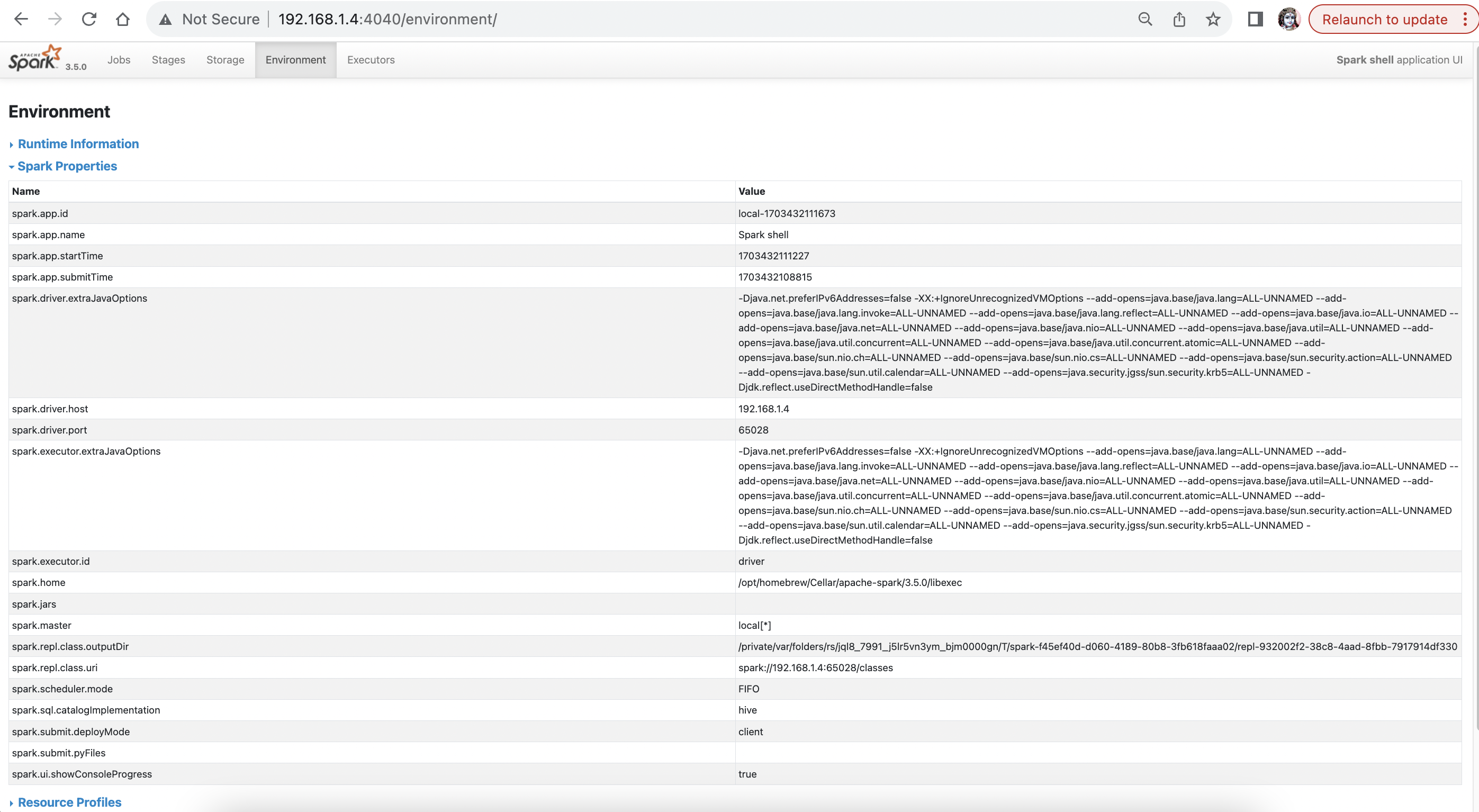
Configuration files

In my local able to run the spark shell application UI…

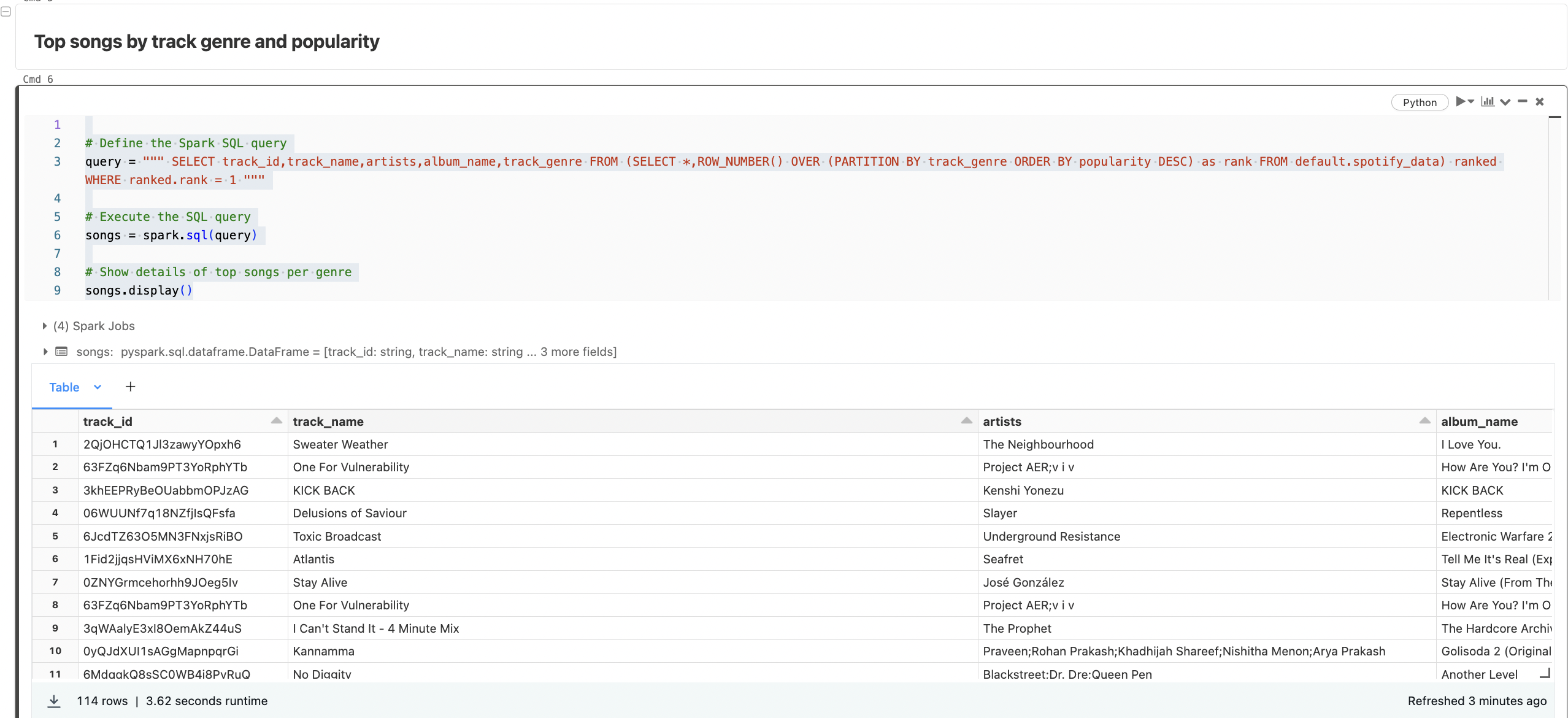
It has all config details



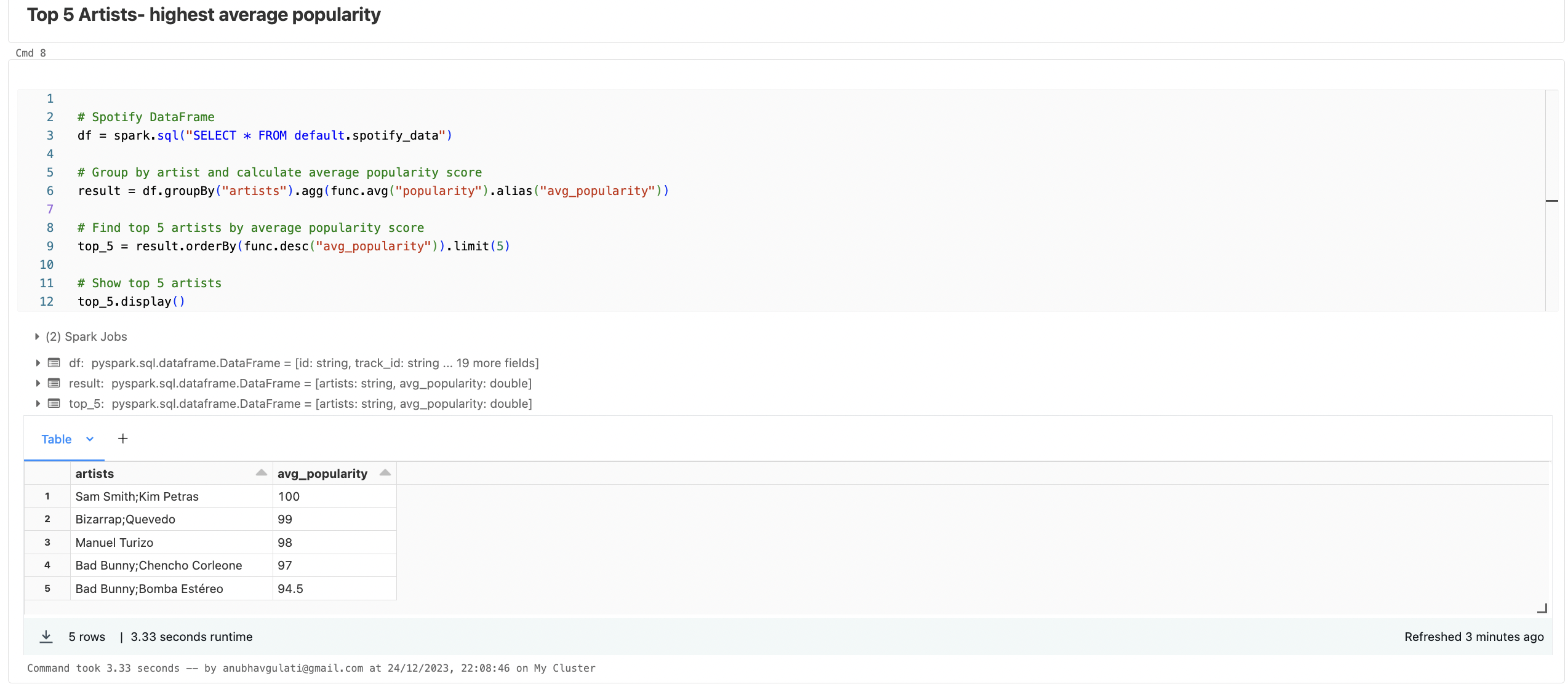
For example spark properties looks like below –



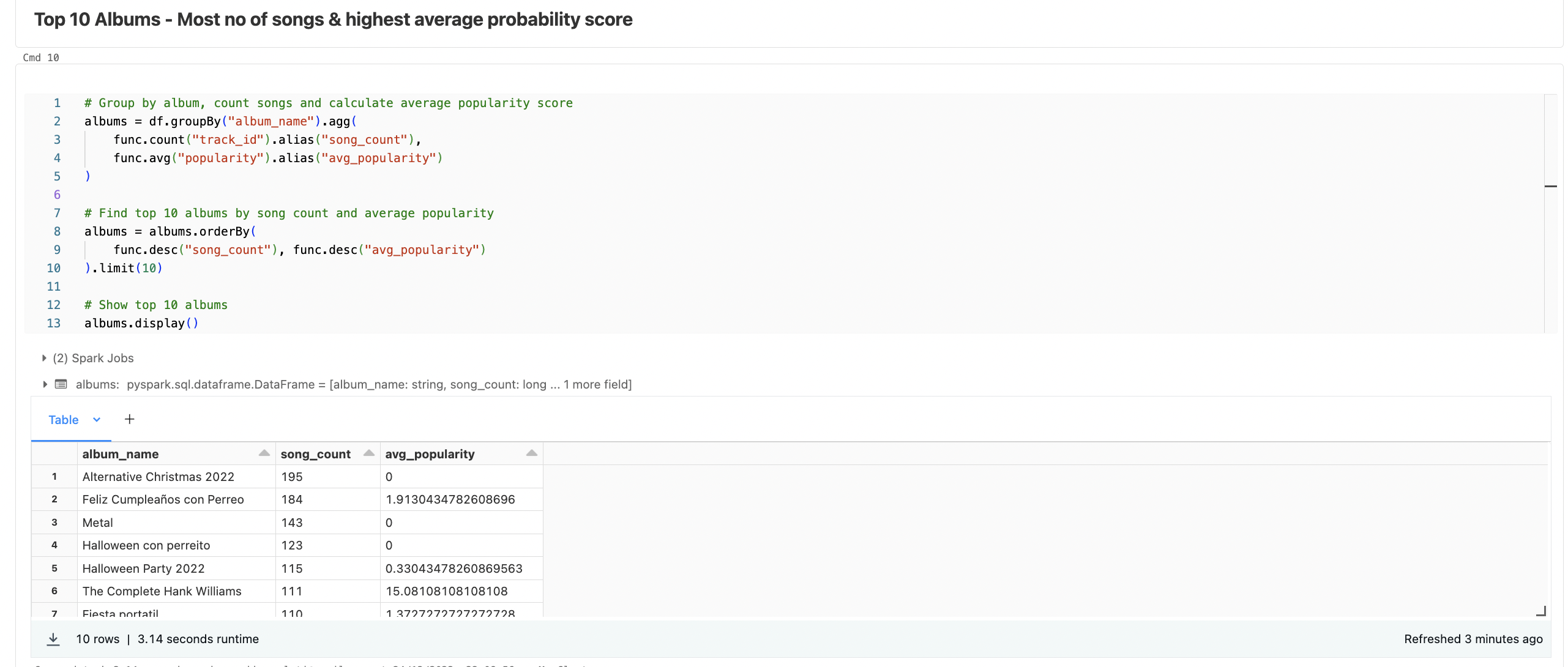
Query 1



Query 2



Query 3



Query 4



Query 5



With all of these analysis, we can easily figure out or it helps which songs are more popular, which artist is more popular etc and an company can generate or do more sales/revenue based on this analysis.