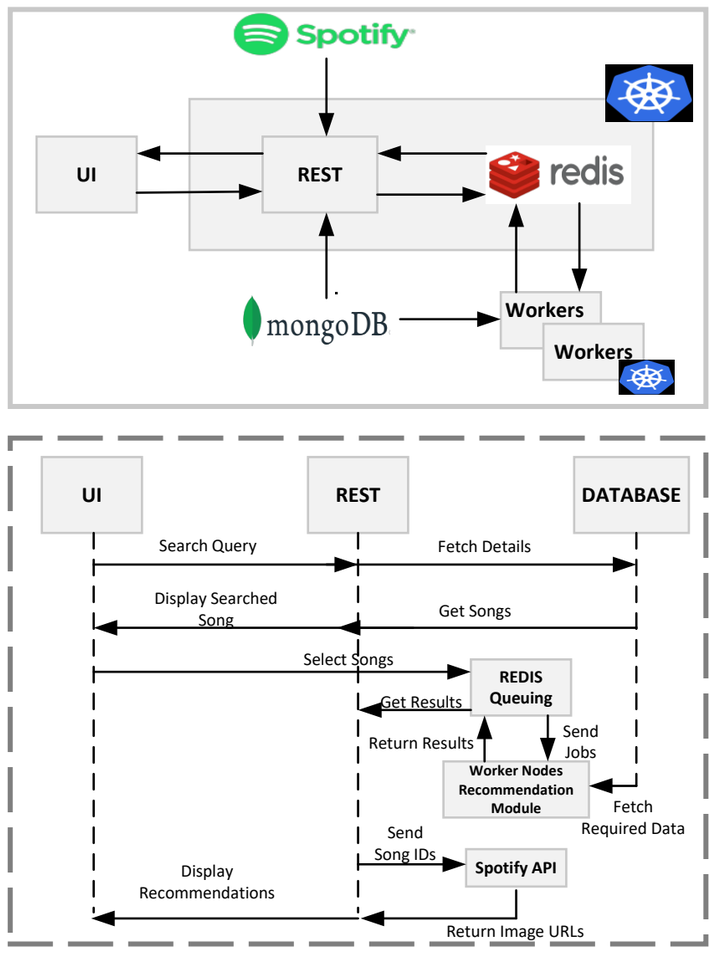
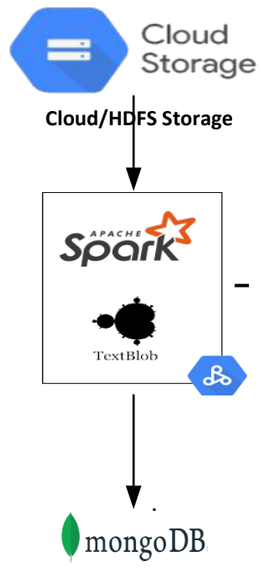
Saransh Rawat (2022OG04027)
Ayush Bansal (2022OG04042)

BDS Assignment - Architecture Diagram : Spotify Music Recommendation

1 High Level Design

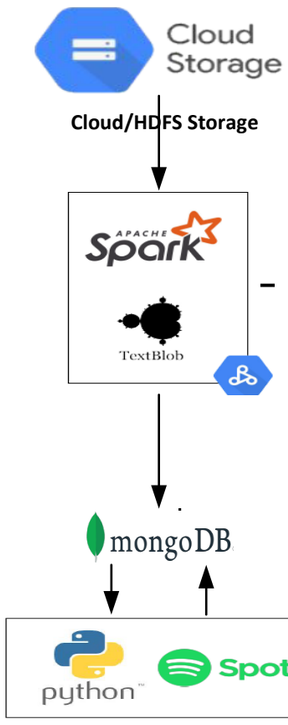


2 Pre-Processing Step 1



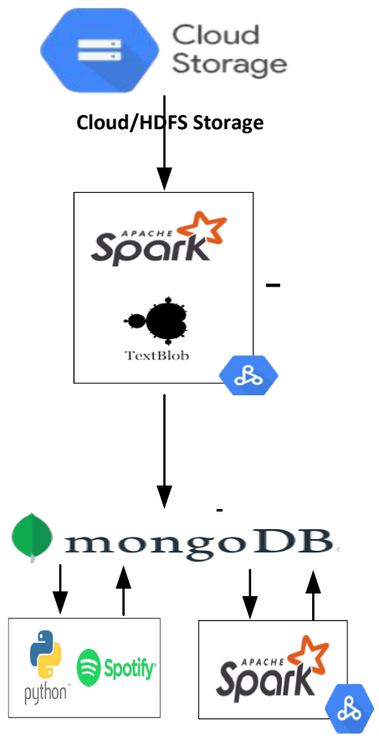
- Data Processing and Data Cleaning.
- Sentiment analysis on track names
- Save the result in MongoDB Database

3 Pre-Processing Step 2



- Data Processing and Data Cleaning.
- Sentiment analysis on track

4 Pre-Processing Step 3



- Use Spark to create feature vectors
- Two Collections :
  #Song Info
  #Song ids & feature vectors

Desired Architecture Components & Tech Choices Summary

- **Flask**: *Framework for developing the REST-Server.* - Flask is a lightweight and powerful web framework for Python. It was chosen for its simplicity and flexibility. It allows you to focus on the core functionality of the application by taking care of the backend details
- **HTML, CSS, and JS**: *For frontend UI* - They allow for a highly customizable user interface and a good user experience.
- **Pyspark:** *For preprocessing the Spotify Playlist Dataset* - This tool was chosen for its ability to handle large datasets. It can process the Spotify Playlist Dataset efficiently, making it a suitable choice for preprocessing.
- **APIs**: *Spotify API for extracting features and cover art images* - The Spotify API is specifically designed to interact with Spotify's music catalog. It provides access to various data, like playlists, albums, and tracks, including their features and cover art images. Using the Spotify API simplifies the process of data extraction.
- **Redis**: *For message queuing* - This in-memory data structure store is used for message queuing. It provides high performance and supports various data structures. It's also capable of persisting on-disk database.
- **MongoDB**: *Storage for extracted features* - MongoDB is a NoSQL database that provides high performance, high availability, and easy scalability. It works well with hierarchical data structures and can handle large volumes of data, making it suitable for storing extracted features.
- **Cloud/HDFS Storage Bucket**: *Used to store raw data* - This is used to store raw data. The choice of a cloud storage bucket is justified by the need for high availability, durability, and scalability. It also allows for easy data access and collaboration.
- **Docker and Kubernetes**: *Deployment of the application* - Docker simplifies deployment by creating a container for the application, which includes all dependencies. Kubernetes is an open-source platform designed to automate deploying, scaling, and managing containerized applications. It helps in managing the application's lifecycle and ensuring its availability and scalability.

CAP Theorem & Correlation

CAP Theorem defines that it's impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees: Consistency, Availability, and Partition tolerance.
In the context of the chosen components:

- **MongoDB:** It provides strong consistency and partition tolerance (CP). However, in the event of a network partition, MongoDB sacrifices availability to ensure data consistency.

This trade-off is acceptable for the Spotify music recommendation system as it prioritizes data consistency over availability.

- **Redis:** It provides consistency and availability (CA) but doesn't guarantee partition tolerance.

This is acceptable for the use case of message queuing where immediate consistency and high availability are more important.

- **Cloud/HDFS Storage:** It offers availability and partition tolerance (AP) but may not always ensure immediate consistency.

This is acceptable in scenarios where the system needs to continue working despite network failures and eventual consistency is sufficient