**Apache Hive**

Hadoop is an open source framework to process huge and voluminous data in parallel environment. Apache Hive, Pig & Sqoop are some of the tools designed on top if Hadoop map-reduce paradigm. The entire purpose of developing these technologies was to allow a user to effectively consume these wonderful, fast and efficient storage and processing technologies without having to worry about skilled resources. Or to deal with configuration, or to redesign their entire application to suit the needs of map-reduce framework.

For our project, Apache Hive is the epicenter of communication or integration to all the technologies involved as a whole starting from Azure ML, Kafka to Stanford API consumption.

We utilized apache hive to save the data consumed to run machine learning algorithms, then we ran some Stanford nlp based sentiment analysis to evaluate a user review and saved that data into tables.

C:\Users\Akash\Downloads\Hive (1).png

Following are the tables we created in hive to support all storage related operations in our project:

1. Expedia\_hotel\_cluster
2. Expedia\_hotel\_review
3. Expedia\_hotel\_search
4. Expedia\_rankings
5. Expedia\_rankings\_competitive\_high
6. Expedia\_rankings\_competitive\_low
7. Expedia\_reco
8. Hotel\_review\_kafka

Setting up configurations is really tricky in hive. Because of lack of documentations or resource examples, the visibility of the way hive can be consumed into industry is really thin.

Essentially, for hive to work on a server, as we had user interface and other utilities utilizing hive as storage. So the connectivity or integration between hive to act as server based and essentially allow java programs to access tables, trying to write sometimes or read.

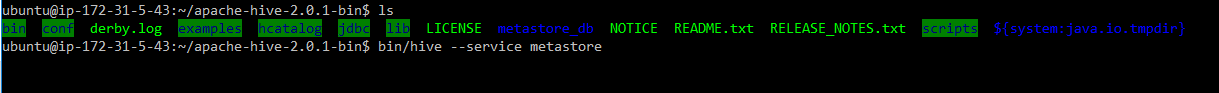
There are three services to be up, before we can consume server services from anywhere:

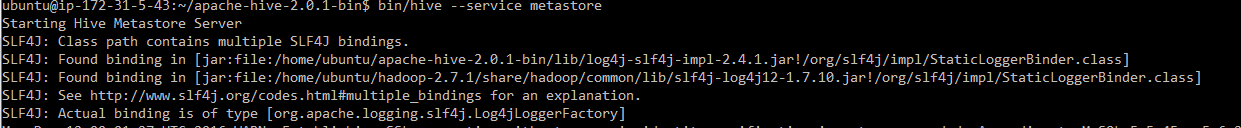
1. Hive Metastore
2. Hive Server2
3. Hive Beeline

There is a configuration file named hive-site.xml which needs to be tweaked before running Hive. And there are close to 100 odd configurations to be set. Most of them will be the default values unless someone trying to implement separate partitioning of data. Allowing access to hdfs & hive as well to specific users using the storage.

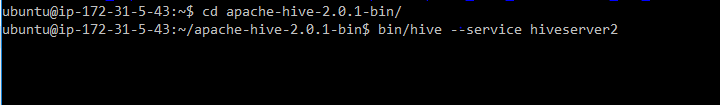
Process to start hive on hdfs:

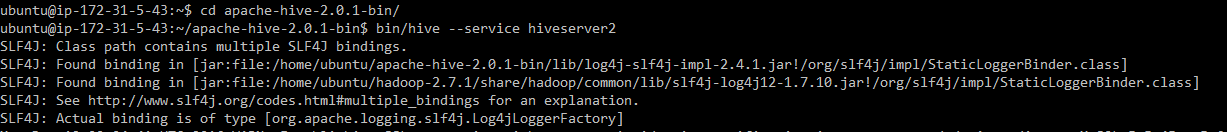
Start metastore



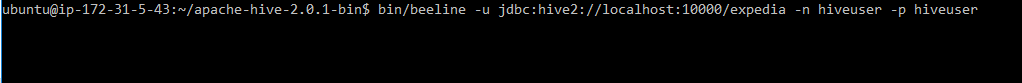


Start hiveserver2

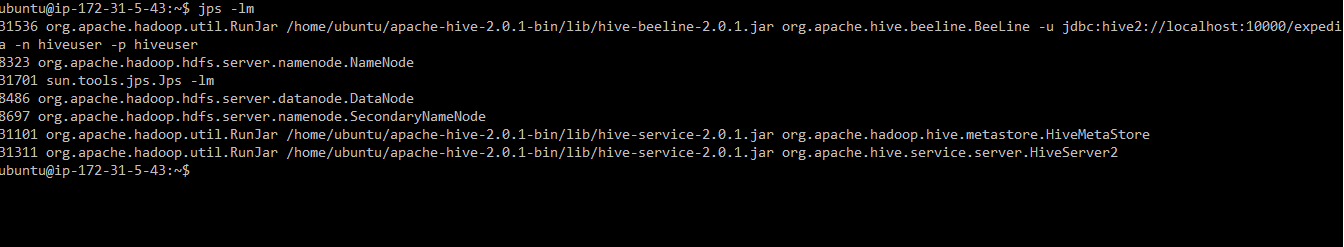




Start beeline



To check whether all three services are running:



Since there were multiple ways in which hive was getting consumed in UI and kafka as well, we created HiveDAO named class to handle all the communication with hive:



Functions:

1. getConnection():

Establish a connection with hive database and provide the connection instance to the calling function.

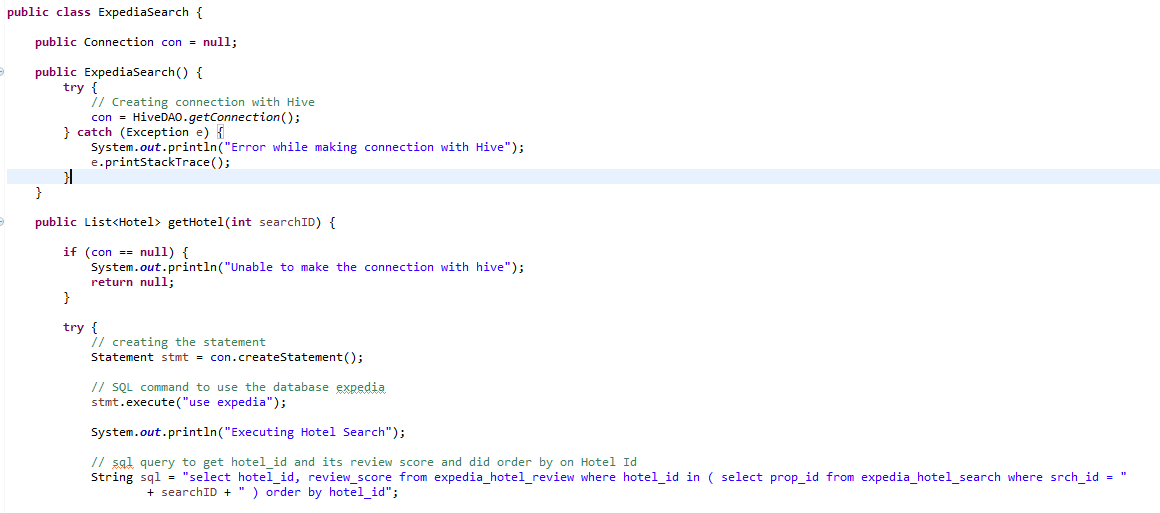
1. updateReviewScore():

For each and every instance of a new review for a hotel, score gets calculated using stanford nlp and the score has to be updated in the hive tables with respect to particular hotel identification.

Since there is already some value available in the tables we use one specific formulae to calculate new review after considering 30 reviews score: 0.8x + 0.2y = review score.

X = old review score; y = new review score

To facilitate web-ui effective retrieval of data from hive, we have created following class:

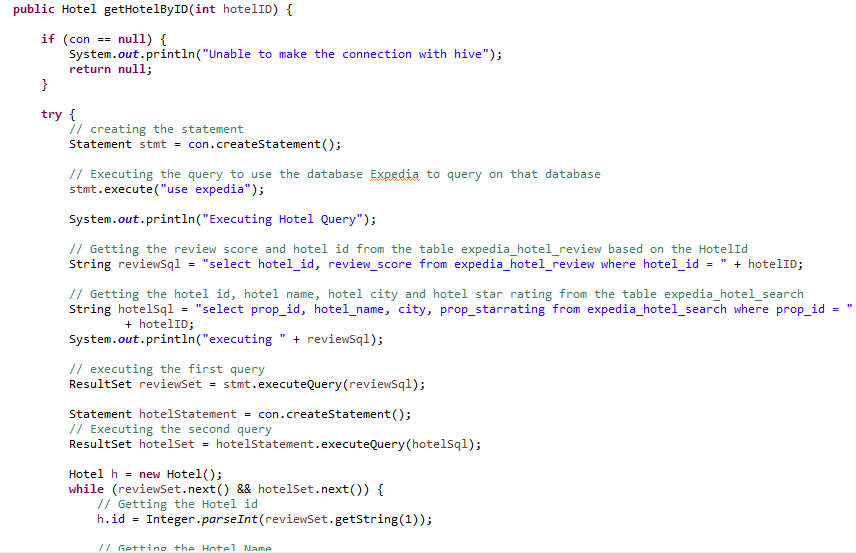


Functions:

1. getHotel(search id)

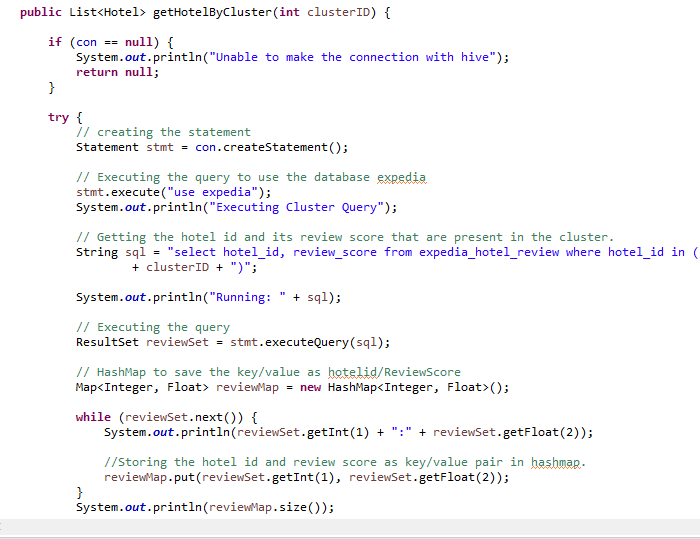
This function triggers a hive query providing search id from UI to facilitate data retrieval. Each hive query essentially runs series of map-reduce jobs to either read or write data.

1. getHotelById(hotelId)



This function helps us with all the details or information available about one particular hotel. One query gets all the details for us using hotel\_id.

1. getHotelByCluster(clusterId)



This function helps us with getting a list of hotel\_ids with based on cluster\_id provided. A query runs at back end to support this transaction with Hive utilizing HiveDAO to get connection.