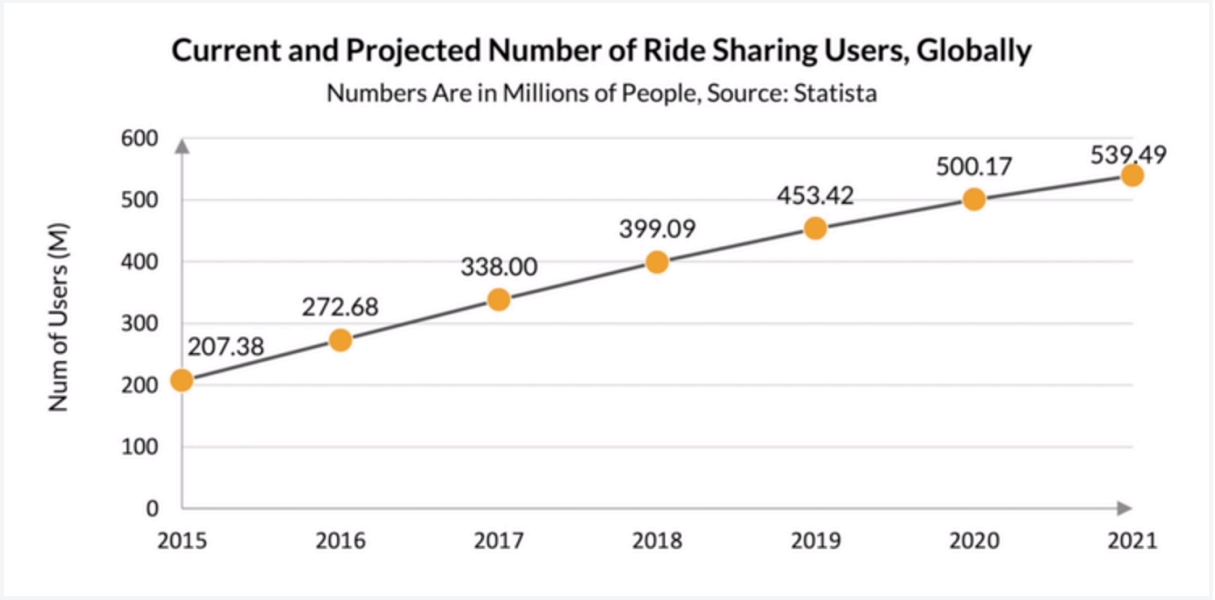
Introduction

In this project, you will ingest data using both real-time and batch frameworks and load them into tables. Once that is complete, you will write queries to derive meaningful insights from them.

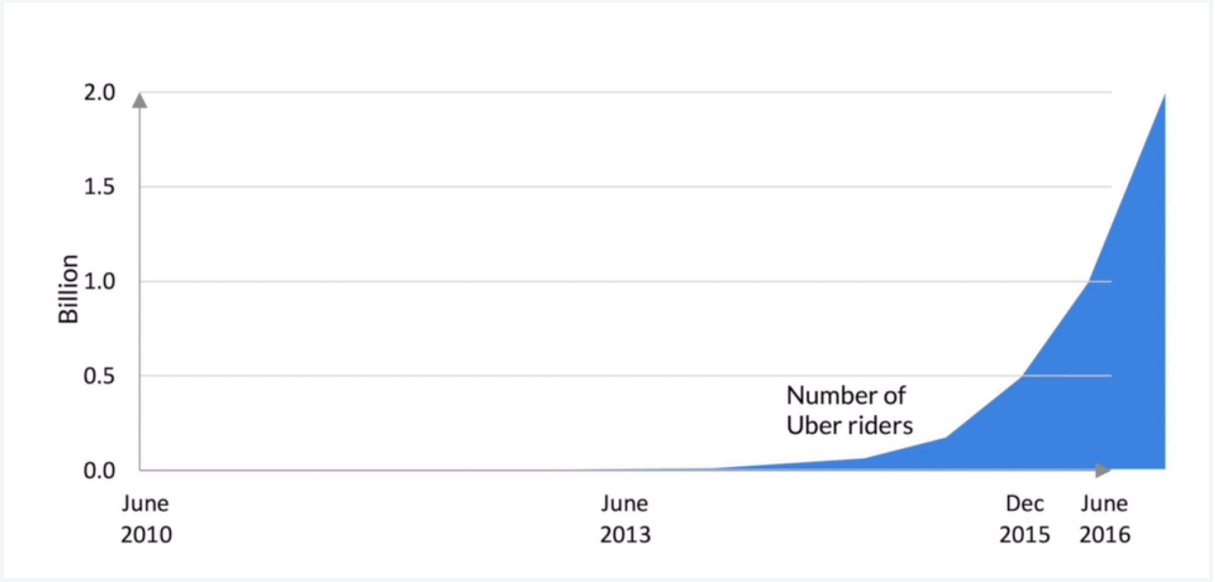
Problem Statement

Suppose you are working at a mobility start-up company called ‘YourOwnCabs’ (YOC). Primarily, it is an online on-demand cab booking service. When you joined the company, it was doing around 100 rides per day. Owing to a successful business model and excellent service, the company’s business is growing rapidly, and these numbers are breaking their own records every day. YOC’s customer base and ride counts are increasing on a day-by-day basis.

It is highly important for business stakeholders to derive quick and on-demand insights regarding the numbers to decide the company’s future strategy. Owing to the massive growth in business, it is getting tough for the company’s management to obtain the business numbers frequently, as backend MySQL is not capable of catering to all types of queries owing to large volume data. The following statistics will help you understand the gravity of the situation:



User Growth Chart



Ride Count Growth Chart

With a rising number of users and rides, the company's major focus is to provide its customers a delightful experience while ensuring zero downtime. Also, business stakeholders should be catered with all the answers that they require that are backed by data. Data-driven decisions play a steroid role for a fast-growing start-up.

You, as a big data engineer, must architect and build a solution to cater to the following requirements:

- **Booking data analytics solution**: This is a feature in which the ride-booking data is stored in a way such that it is available for all types of analytics without hampering business. These analyses mostly include daily, weekly and monthly booking counts as well as booking counts by the mobile operating system, average booking amount, total tip amount, etc.
- **Clickstream data analytics solution**: Clickstream is the application browsing data that is generated on every action taken by the user on the app. This includes link click, button click, screen load, etc. This is relevant to know the user's intent and then take action to engage them more in the app according to their browsing behaviour. Since this is very high-volume data (more than the bookings data), it needs to be architectured quite well and stored in a proper format for analysis.

Additional Reading

Growth in Cab Rides - You can read more from here about the projected growth in ride-sharing.

Data

The following data will be used for this problem:

- **bookings** (The booking data is added to/updated in this table after a booking/ride is successfully completed.)
    - **booking_id**: Booking ID String
    - **customer_id**: Customer ID Number
    - **driver_id**: Driver ID Number
    - **customer_app_version**: Customer App Version String
    - **customer_phone_os_version**: Customer mobile operating system
    - **pickup_lat**: Pickup latitude
    - **pickup_lon**: Pickup longitude
    - **drop_lat**: Dropoff latitude
    - **drop_lon**: Dropoff longitude
    - **pickup_timestamp**: Timestamp at which customer was picked up
    - **drop_timestamp**: Timestamp at which customer was dropped at destination
    - **trip_fare**: Total amount of the trip
    - **tip_amount**: Tip amount given by customer to driver for this ride
    - **currency_code**: Currency Code String for the amount paid by customer
    - **cab_color**: Color of the cab
    - **cab_registration_no**: Registration number string of the vehicle

    - **customer_rating_by_driver**: Rating number given by driver to customer after ride
    - **rating_by_customer**: Rating number given by customer to driver after ride
    - **passenger_count**: Total count of passengers who boarded the cab for ride

Since booking data is updated by the backend team in MySQL, it is stored in a central AWS RDS. You will be given the already loaded bookings table data.

- **clickstream** (All user's activity data such as click and page load):
    - **customer_id**: Customer ID Number
    - **app_version**: Customer App Version String
    - **os_version**: User mobile operating system
    - **lat**: Latitude
    - **lon**: Longitude
    - **page_id**: UUID of the page/screen browsed by a user
    - **button_id**: UUID of the button clicked by a user
    - **is_button_click**: Yes/No depending on whether a user clicked button
    - **is_page_view**: Yes/No depending on whether a user loaded a new screen/page
    - **is_scroll_up**: Yes/No depending on whether a user scrolled up on the current screen

- **is_scroll_down**: Yes/No depending on whether a user scrolled down on the current screen
- **timestamp**: Timestamp at which the user action is captured

This data is the real-time streaming data generated by the application in the JSON format. It is stored in an existing Kafka server. You will be given the already loaded clickstream Kafka Topic.

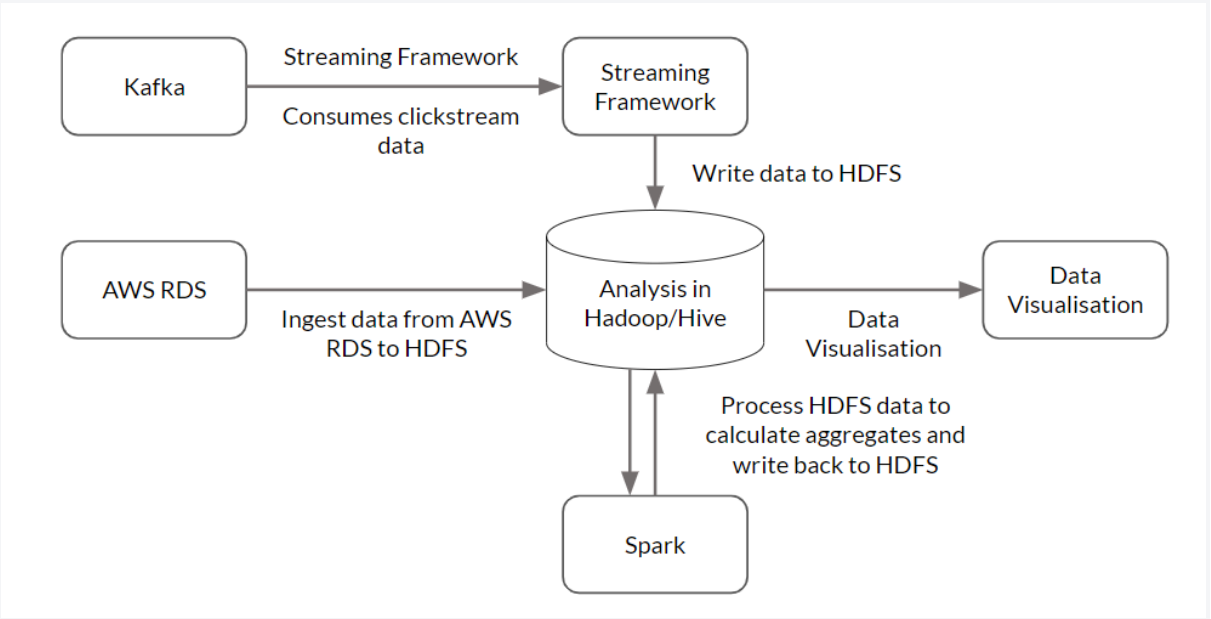
Here is an example of a JSON payload structure that is produced:


```
{
 "customer_id": 342516,
 "app_version": "4.1.34-beta",
 "OS_version": "Android",
 "lat": 14.11,
 "lon": 89.88,
 "page_id": "e7bc5fb2-1231-11eb-adc1-0242ac120002",
 "button_id": "fcba68aa-1231-11eb-adc1-0242ac120002",
 "is_button_click": yes,
 "is_page_view": no,
 "is_scroll_up": no,
 "is_scroll_down": no,
 "timestamp": "2020-10-20TT17:52:24"
}
```


Note: The details to get these data from RDS and Kafka is present in the '**Resources**' segment.

Architecture and Approach

The following diagram will help you understand what the entire architecture should look like.



Architecture

The two types of data are the clickstream data and the batch data. The clickstream data is captured in Kafka. A streaming framework should consume the data from Kafka and load the same to Hadoop. Once the clickstream data enters the Stream processing layer, it is synced to the HDFS directory to process further. For the batch data, which is the bookings data, the data is stored in the RDS and needs to be ingested to Hadoop.

Also, in cases wherein aggregates need to be prepared, data is read from HDFS, processed by a processing framework such as Spark and written back to HDFS to create a Hive table for the aggregated data.

Once both the data are loaded in HDFS, this data is loaded into Hive tables to make it queryable. Hive tables serve as final consumption tables for end-user querying and are eventually consistent.

Having understood the architecture, you will now learn how to approach this.

First, you need to capture the clickstream data from Kafka that needs to be loaded into HDFS using a stream processing framework. Next, you need to ingest the booking data from AWS RDS to Hadoop. These are then loaded into Hive tables to make them queryable. In some cases wherein you need to prepare the aggregates, data is read from HDFS, processed by a framework such as Spark and written back to HDFS to create a Hive table for aggregated data to make it more queryable. These Hive tables serve as the final consumption tables for stakeholders to query and derive meaningful insights from the data.

Tasks

Before we discuss the guidelines and tasks that you need to perform, make sure that you are configuring the EMR cluster properly.

You will need to make sure that you have **Hadoop, Sqoop, Hive and Spark** installed on your cluster with **Hue** as an optional service. Also as an added step, make sure that in the **Hardware configuration step** for the EMR cluster

generation, scroll down to the **EBS Root Volume configuration** and type the **Root device EBS volume size** as **20 GB.**

So, as part of this project you need to do the following tasks:

- **Task 1**: Write a job to consume clickstream data from Kafka and ingest to Hadoop.
- **Task 2**: Write a script to ingest the relevant bookings data from AWS RDS to Hadoop.
- **Task 3**: Create aggregates for finding date-wise total bookings using the Spark script.
- **Task 4**:
    - Create a Hive-managed table for clickstream data.
    - Create a Hive-managed table for bookings data.
    - Create a Hive-managed table for aggregated data in Task 3.
- **Task 5**: Calculate the total number of different drivers for each customer.
- **Task 6**: Calculate the total rides taken by each customer.
- **Task 7**: Find the total visits made by each customer on the booking page and the total ‘Book Now’ button presses. This can show the conversion ratio.
  The booking page id is ‘e7bc5fb2-1231-11eb-adc1-0242ac120002’.
  The Book Now button id is ‘fcba68aa-1231-11eb-adc1-0242ac120002’. You also need to calculate the conversion ratio as part of this task. Conversion ratio can be calculated as **Total 'Book Now' Button Press/Total Visits made by customer on the booking page**.

- **Task 8**: Calculate the count of all trips done on black cabs.
- **Task 9**: Calculate the total amount of tips given date wise to all drivers by customers.
- **Task 10**: Calculate the total count of all the bookings with ratings lower than 2 as given by customers in a particular month.
- **Task 11**: Calculate the count of total iOS users.

Note: You might be getting 10-15 fewer messages from the Kafka topic from what is mentioned in the Validation document. It is not an issue and you can proceed ahead with the project.

Note: Please note that the values present in the validation document might be different than the ones you get after running the project code. If you have roughly the same result as in the validation document, then please proceed further as the marks are awarded for the code that you submit.

Note: At the end of **four weeks** you are required to make submissions for the **first four tasks**. Then at the end of **six weeks**, you are required to make submissions for the **last seven tasks**. The relevant documents related to these submissions are present in the '**Submissions**' segment in the next session.

Note: The details to get these data from RDS and Kafka is present in the '**Resources**' segment.

Resources

The RDS connection string and credentials are as follows:

RDS **Connection String** -
jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase

-
- **Username** - student
- **Password** - STUDENT123
- **Table Name** - bookings

Details of the Kafka Broker are as follows:

- **Bootstrap-server** - 18.211.252.152
- **Port** - 9092
- **Topic -** de-capstone3

Note: Please refer to the Kafka Integration segment in the Industry demo session in the Spark Streaming module to see how to read and write to Kafka with the help of Spark Streaming.

Submission

Note: For the mid submission, you need to complete and submit the files corresponding to the first 5 points and upload the .zip file for the same. The sample submission docs are attached below for your reference.

Submissions required

Upload a zip file containing the following:

Note: Please note that the steps mentioned below are according to the Sample solution where Spark jobs have been run in local mode. If you are not running Spark jobs in local mode then you can skip the parts where you have to move the data to HDFS.

- A python file (**spark_kafka_to_local.py**) containing the code to ingest the relevant data from Kafka and load the data into local file system. A python file (**spark_local_flatten.py**) to clean the loaded Kafka data to a more structured format. A PDF document(**step1.pdf**) which contains the steps to run the python file and the steps to move the data to HDFS.
- A PDF document (**step2.pdf**) containing the command to ingest the data from AWS RDS to Hadoop.
- A python file (**datewise_bookings_aggregates_spark.py**) to create aggregates for finding date-wise total bookings. The date-wise bookings aggregates should be stored in a **csv** file and should be moved to HDFS. A PDF document (**step3.pdf**) containing the steps to run the python file and the steps to move the csv file to HDFS should be provided.

- A PDF document (**step4.pdf**) containing the steps to create a Hive managed table and the steps to load the relevant data in those tables.
- A PDF document (**LogicFirstSubmission.pdf**) containing an explanation of the solution to the problem in detail should be provided properly in a document. This document should also contain screenshots of the code execution.
- A PDF document (**queries.pdf**) containing the queries to complete all the tasks starting from **task 5** till **task 11**. The document should also contain the screenshots of the results obtained after running the queries.
- A PDF document (**LogicFinal.pdf**) containing an explanation of the queries should be provided properly in a document. This document should also contain the screenshots of the execution of the queries and the screenshots of the results obtained after executing the queries.

Note: All the sample documents have been provided to you as a word document at the end of the segment in a zip file. You need to add all the queries and the steps in the same document and then upload the updated documents.

Note: The zip file does not contain any python files. Make sure to include all the necessary python files in your submission.

Please make sure that you are not changing any of the file names that have been provided above in parentheses. The code that you are submitting should run at our end without any modifications in the code.