**Use Case**:

Assignment is about real time streaming data analysis. This real time data is generated out of customers call records. Company would want to analyse this data in real time to get the instantaneous insight which will enable them to make quick business decision which are more relevant in on going situation.

Real time analytics lets users see, analyse and understand data as it arrives in a system.

In other words, users get insights or can draw conclusions immediately (or very rapidly after) the data enters their system.

Real-time analytics allows businesses to react without delay. They can seize opportunities or prevent problems before they happen. By comparison, batch-style analytics may take hours or even days to yield results. Consequently, batch analytical applications often yield only “after the fact” insights (lagging indicators) where as real time analytics can allow businesses to get ahead of the curve.

Call Data Record (CDR) provides data about how a specific phone number or User is utilizing the phone system.

It includes caller number, recipient number, duration, call origination time, source tower and recipient tower location. This data can greatly help companies to scale their infra structure based on volume information in a city or location.

This data is also useful to do analysis on cell phone model being used.

**Architecture Choice**:

Our architecture should take care of below things fundamentally:

1. ability to store and ingest large volume of data.

2. ability to scale horizontally both for ingestion & storage.

3. making data available for analytics in real time-time

4. ability to ingest any new data with minimum development effort.

5. cost effective both in terms of capex & opex

**Choice of Technology stack**:

1. Apache Kafka to stream the data in real time-time

2. Apache Spark as compute or processing engine

3. Apache Hadoop for distributed storage & distributed computing with Apache Spark

4. Apache parquet for storage format

5. Hive, Presto & Pig for query or analytics engine

6. Apache Hudi for fast Insert, Updates & Deletes.

**Workflow**:

1. Apache Kafka will be used to source the raw call record data incrementally to our data platform from various upstreams.

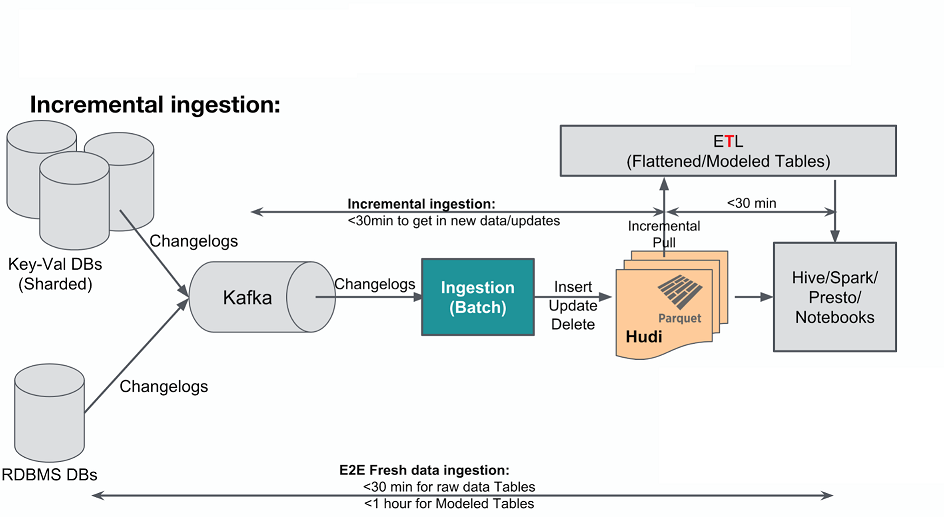
2. Apache Spark streaming reads the raw kafka messages and ingest the data in hadoop in mini batches.

3. Before persisting in hdfs, it passes it through the Apache Hudi to take care of insert, update and deletes. Hudi stores the data in parquet file format.

4. Presto, Hive & Pig can be used to query the data from parquet store.

5. Apart from this, Apache Spark based ETL will model the raw data in the form which various team requires. This will have some time delay and modelled data will not be available in real time.

6. This is a realisation of Kappa Architecture where data is always streamed. How ever architecturally Apache Spark streaming is fundamentally based on concept of micro batches.



**Component Design**:

**Apache Kafka**:

Kafka is used to capture the CDR from various sources. It is persistent and degree of parallelism can be controlled using number of partitions. It is fault tolerant and extremely fast to read and write.

Other options are Amazon Kinesis and Microsoft Events Hubs. But of these options are good choice if we are building the data platform in cloud and want to leverage the tool set available with cloud vendors.

Apache Kafka being open source and having strong community support and is well proven for lot of real time data sourcing make an ideal choice for us.

**Apache Spark**:

Spark streaming provides real time data processing using micro batches. These batches are read with time window to control the batch size. This time window can be increased or decreased based on how frequently the real time data need to be ingested.

Other options is Apache Flink. It is also a streaming processing engine and scale very well, but given apache spark integrates with Hudi which we are using as a tool to take care of IUD, we will be going with Apache Spark. At the moment there is no integration available between Apache Flink & Hudi.

[**Apache Hudi**](https://eng.uber.com/hoodie/):

Hudi backed by columnar storage Apache parquet provides extremely fast insert, updates or deletes on the stored raw data by applying the changes from incoming raw data.

Apache Hudi integrates with Apache Spark as one of the format like avro and parquet and abstract out the IUD as the fundamental functionality which every users dont have to develop.

This will enable users to query the existing data without worrying about how the data is getting changed in the back end.

Hudi will efficiently take care of compacting small files into larger files so the name node is not loaded.

It also partitions the data while writing which makes writing, updating & querying quite fast.

Another option is to use [databricks delta](https://docs.databricks.com/delta/index.html) here, which takes care of IUD. But data bricks delta has its own proprietary format to store the data. This means data cannot be queried using other open source tools like hive and we always have to use databricks delta apis to query the data which makes overall design very much coupled with one specific technology.

**ETL/Data Modeling Layer**:

While raw data is read from kafka through spark and persisted using Hudi, there are also some users who would want to model the certain data for their application.

How ever modelling involves data transformation as well as joining the data from different sources, this layer is not as fast as raw data layer.

Modelled data will have a latency of about 30 mins to 1 hr to be available depending on whether full snapshot is required or only incremental data is required and how large are the joins.

**Data Query Layer**:

There are various query engine option available to users. Given the data is stored in open source format like parquet & avro, users have choices like presto or hive to query the data depending on how whether it is interactive or complex and heavy query.

If the users want interactive query engine for quick exploration of data, then Presto is a good candidate.

How ever if the use case is for analytical queries which joins large fact tables, then Apache Hive is a better choice.

**System Integration**:

1. Users call record data is published to kafka from sensors.

2. Apache spark based stream processing engine(SPE) is continuously listening to the kafka topics.

3. SPE reads the messages from kafka topic in micro batches.

4. each such micro batch is then written in hdfs parquet file using Hudi.

5. given hdfs files are immutable, it means if Hudi need to update or insert a record in file it has to

read the file, append the new records to data read from file and write the full file again.

6. this will become slower and slower as the data grow. Instead Hudi partitions the data to read and

write the smallest chunk of data.

7. Once the data is written through Hudi in hdfs, it becomes available for users to query.

8. Users can use presto or Hive on top of Hudi written parquet file to query the data on hdfs.

9. Another ETL job which model the raw data also process the raw data and produces modelled output which is again written in hdfs using Hudi and is available for users to query.

**Solution Explanation**:

At the start of the assignment we have considered some fundamental functional requirements.

Our data platform has to be built for volume and low latency data availability.

It should be able to scale horizontally.

With these requirements in mind, we have used open source technologies like kafka, spark and hudi which works with hadoop to give a distributed low latency data platform.

Data flows from sources to kafka and apache spark processing engine. Apache spark uses Hudi on top of apache parquet file format on hdfs to provide low latency IUD on data.

While the data is being ingested, users are also able to query the data and as the data gets updated the query returns updated result without being blocked by write process.

Interactive query is done using Presto where as large analytics sql query which requires join from large fact tables can be done using Hive.

At the same time, ETL layer provides users an ability to transform and enrich the raw data to create modelled output.

This modelled output is also written in hdfs file and is availble for users analytics.