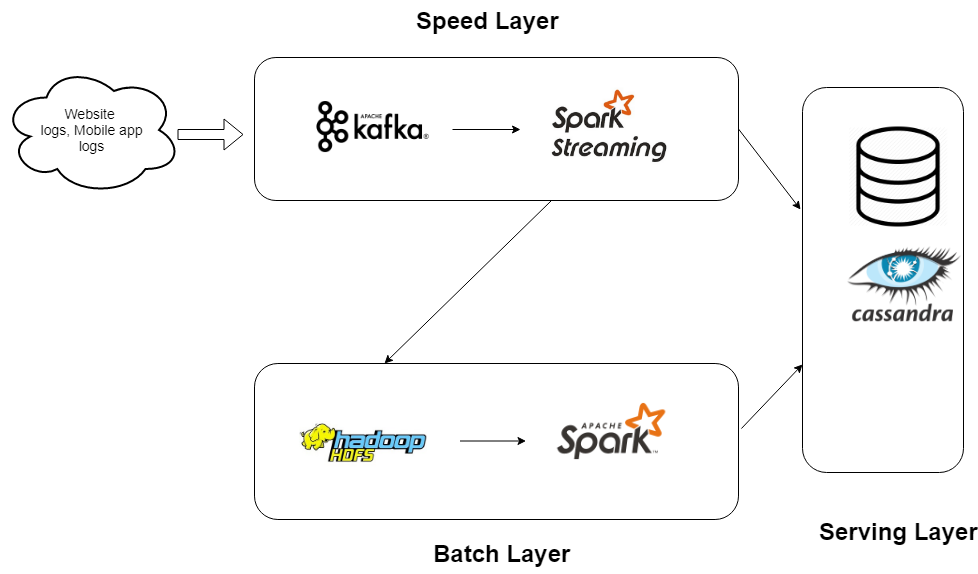
1. Lambda Architecture



Explanations:

1. Apache Kafka

Apache Kafka is a distributed publish-subscribe messaging system that can handle a high volume of data. It supports low latency message delivery and gives guarantee for fault tolerance in the presence of machine failures. It is very fast, can perform 2 million writes/sec. In our use case 300 million raw records needs to be ingested per day and during peak seasons 500 million raw records needs to be ingested. Considering the volume of raw data I think Kafka is the best option. If the data volume groves in the future we just need to add extra Kafka servers to handle the load and it’s very easy. Also, distributed streaming processing engines like Spark and Storm supports Kafka messaging system.

1. Spark Streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Spark Streaming can be used to stream live data and processing can happen in real time. In our use case, Spark can directly read raw data from Kafka topics to perform transformation, data extraction in real time. The output can be persisted in HDFS or Cassandra database.

1. Spark (Batch Processing)

Apache Spark is an in-memory distributed computation engine which is faster than other distributed computation engines. Spark batch application can read data from various sources like HDFS, S3, Local File System and Databases. In our use case, spark can read the data from HDFS location (data persisted by spark streaming) and using ML libraries we can perform predictions on daily basis. Finally the computed data can be persisted in Cassandra ( using datastax Cassandra connector API).

1. Cassandra

Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency. So, it will be suitable for our use case (as we need to ingest 500 million rows per day). It is easily scalable to handle heavy loads.

1. Speed Layer Application
2. Publish data into Kafka topic.

* Start zookeeper server

$KAFKA\_HOME/bin/zookeeper-server-start.sh $KAFKA\_HOME/config/zookeeper.properties

* Start Kafka broker

$KAFKA\_HOME/bin/kafka-server-start.sh $KAFKA\_HOME/config/server.properties

* Create topic to store log data

$KAFKA\_HOME/bin/kafka-topics.sh --create --zookeeper zookeeper\_server:2181 --topic LogData --replication-factor 1 --partitions 1

* Publish data into kafka topic.

$KAFKA\_HOME/bin/kafka-console-producer.sh --broker-list kafka\_server:9092 --topic LogData

(Copy **clickstream-sample.json** content and paste in terminal)

1. Run Spark Streaming application.

* Add **kafka-dependency jars** inside spark/jars
* Update the kafka server details, hdfs location details in LogProcessor file.
* Build the project with command “**sbt package**” (run this command from MS\_Data Engineer folder)
* Run the spark application with below command
* Spark-submit –class LogProcessor E:\MS\_DataEngineer\target\scala-2.11.\ ms-data-engineer\_2.11-0.1.0-SNAPSHOT.jar