**Introduction:**

This PoC is mainly aimed to prove that spark with scala can address the following issues:

* + Two way communication
  + Message out of order

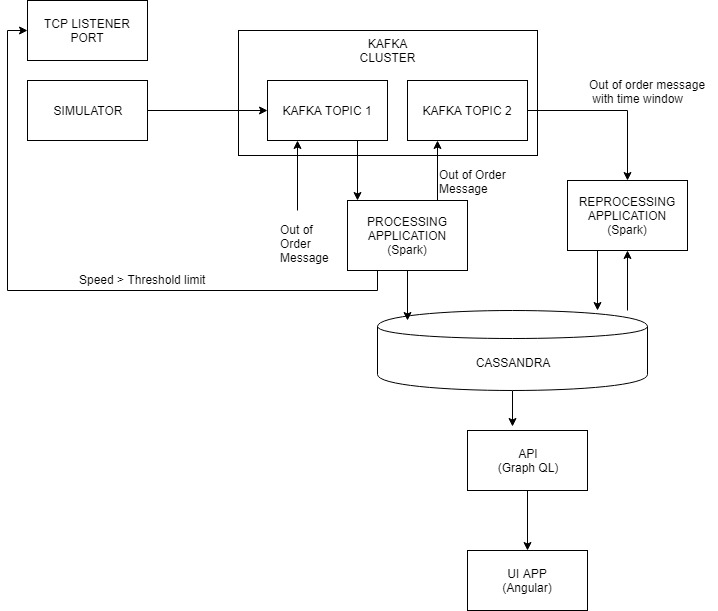
Two way communication is made possible by sending the port number along with the message (this is already a part of the new json format output from the listener). An incoming message is identified as an old message based on the UnprocessedCreatedDate field. These old messages are sent to a kafka topic for reprocessing.

Brief Explanation of Workflow:

We send messages in the new json format using a simulator. Each message has different speed values and different IMEIs and the current time and date. The simulator writes the messages to a kafka topic. We have developed a spark streaming application that reads messages from this kafka topic. From the message, required information like speed, portnumber, UnprocessedCreatedDate and IMEI is parsed . This parsed data is saved into Cassandra table. From the parsed data all the messages that have the speed value greater than a threshold limit is filtered and for each of the filtered messages an alert is send back to the port number in the message using TCP protocol. We have also developed a listener using which we can visualize this alert.

Also another functionality of the PoC is that an average speed of all the messages coming in a time window of 2 minutes is calculated and inserted into Cassandra table.

When a message arrives all the messages that is not part of the current 2 minute time window is filtered out and send to another kafka topic. Another spark application is listening to this kafka topic. This application retrieves every messages belonging to same time window of late message's time window from table,then recalculates the average and updates again in database.



Steps to run PoC:

* Run Cassandra server locally
* Create 2 tables in koreone namespace using the following queries:
  + create table speed(id uuid primary key, imei text, portno int, speed float, actualdate timestamp, unprocessedcreateddate timestamp);
  + CREATE TABLE sample

(

id UUID,

imei TEXT,

avgspeed FLOAT, windowendtime TIMESTAMP,

PRIMARY KEY((imei), windowendtime, id)

);

* Start kafka server locally
* Run portwise process spark application
* Run Recalculation spark application
* Run simulator spark application
* Run poc api (graphql api)
* Run teraflow ui code (ng serve)
* Run all the above applications for some time (a minimum of 2 minutes)
* It is possible to visualize the average speed of different imeis in different windows using the ui (localhost:4200)
* To test the reprocessing capability send a message with an old unprocessed created date (make sure that this time is after the service hast started, you can take any of the time of any previous messages. This is logged to the console) using kafka-console -producer
* After sending this message hit show button in the ui, you can see that the average in that window gets updated
* Attaching sample message:

{"Sensors":[{"Name":"MessageType","Value":"GTFRI","Type":"Other","SensorID":315,"UnitOfMeasurement":28},{"Name":"Col1","Value":"GTFRI","Type":"Other","SensorID":101,"UnitOfMeasurement":28},{"Name":"Event","Value":"GTFRI","Type":"Other","SensorID":2,"UnitOfMeasurement":28},{"Name":"DeviceType","Value":"27","Type":"Other","SensorID":551,"UnitOfMeasurement":28},{"Name":"ProtocolVersion","Value":"0501","Type":"Other","SensorID":558,"UnitOfMeasurement":28},{"Name":"GPSFix","Value":"A","Type":"Other","SensorID":3,"UnitOfMeasurement":28},{"Name":"HDOP","Value":"1","Type":"Other","SensorID":241,"UnitOfMeasurement":28},{"Name":"Speed","Value":"10","Type":"Speed","SensorID":9,"UnitOfMeasurement":24},{"Name":"Azimuth","Value":"9","Type":"Direction","SensorID":7,"UnitOfMeasurement":22},{"Name":"Altitude","Value":"2320.8","Type":"Length","SensorID":8,"UnitOfMeasurement":2},{"Name":"Longitude","Value":"-99.215261","Type":"Other","SensorID":6,"UnitOfMeasurement":21},{"Name":"Latitude","Value":"19.584815","Type":"Other","SensorID":5,"UnitOfMeasurement":21},{"Name":"UTCTime","Value":"01/11/2019 22:24:40","Type":"Other","SensorID":700,"UnitOfMeasurement":28},{"Name":"CellInfo1","Value":"CellID:98502080|LAC:9529|MCC:820|MNC:3","Type":"Other","SensorID":511,"UnitOfMeasurement":28},{"Name":"Odometer","Value":"102755.9","Type":"Length","SensorID":201,"UnitOfMeasurement":0},{"Name":"BatteryLevel","Value":"100","Type":"Percentage","SensorID":10,"UnitOfMeasurement":17},{"Name":"MotionStatus","Value":"17","Type":"Other","SensorID":313,"UnitOfMeasurement":28},{"Name":"IgnitionStatus","Value":"1","Type":"Other","SensorID":133,"UnitOfMeasurement":28},{"Name":"DigitalInputStatus","Value":"00000000","Type":"Other","SensorID":12,"UnitOfMeasurement":28},{"Name":"DigitalOutputStatus","Value":"00000000","Type":"Other","SensorID":13,"UnitOfMeasurement":28},{"Name":"SendTime","Value":"01/11/2019 22:26:57","Type":"Other","SensorID":701,"UnitOfMeasurement":28},{"Name":"SequenceNumber","Value":"2933","Type":"Other","SensorID":15,"UnitOfMeasurement":28},{"Name":"IsGPSValid","Value":"True","Type":"Other","SensorID":4,"UnitOfMeasurement":28}],"Guid":"2977655f-b576-4666-8d20-c4afff094aa2","IMEI":"AA001","ActualDate":"2019-01-11T22:24:40","HardwareName":"QUECLINKGV300","UnprocessedCreated":"2019-03-28T11:14:54","PortNumber":11003,"ServerName":"DALVAREZ-691","TenantName":"itrac","MessageType":"KOREPL"}