SPA ASSIGNMENT#2

❖ MQTT ❖ KAFKA ❖ KSTREAM ❖ KSQL ❖

❖ KAFKA CONNECT ❖ ELK ❖ COCKROACH DB ❖

Group 142

SURESH BABASAHEB NIMBALKAR (2019HC04104)

WAVHAL HEMANT SUDHIR (2019HC04093)

SURAJ KUMAR (2019HC04912)

Requirements

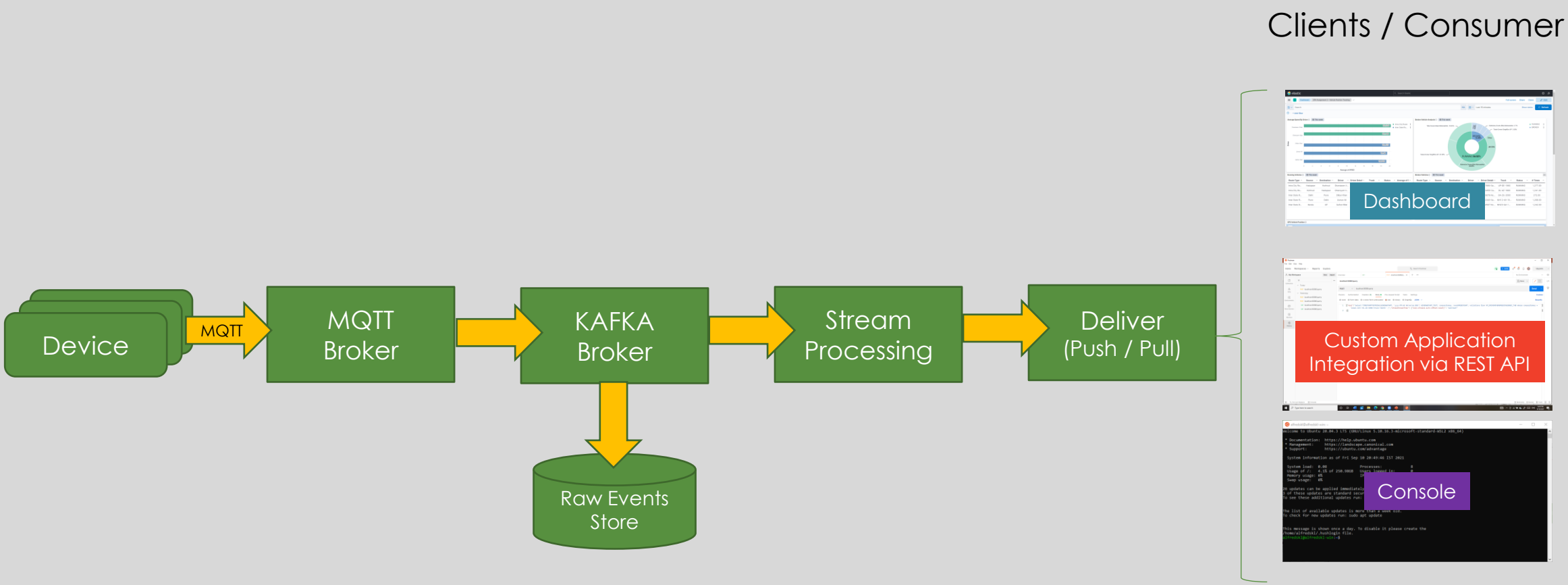
Your group is selected to prepare a working prototype of this IVMS using open-source messaging platform Apache Kafka. A working prototype should mimic the following requirements -

1) Capturing the real time truck movement data from the sensors fitted in the trucks
2) Moving the running truck data over MQTT protocol to a centralized location
3) Moving data from centralized location to messaging store for intermittent storage (may put it in the persistent storage as well)
4) Preprocessing of the data received from the trucks for quality checks and for other required transformations
5) Doing the processing of data to identify the drivers exceeding the speed limits
6) Providing a mechanism to flag out the details of drivers exceeding the speed limits
7) Providing a way to maintain the count of over speeding incidents over the period of time, on particular routes, for particular trucks etc.

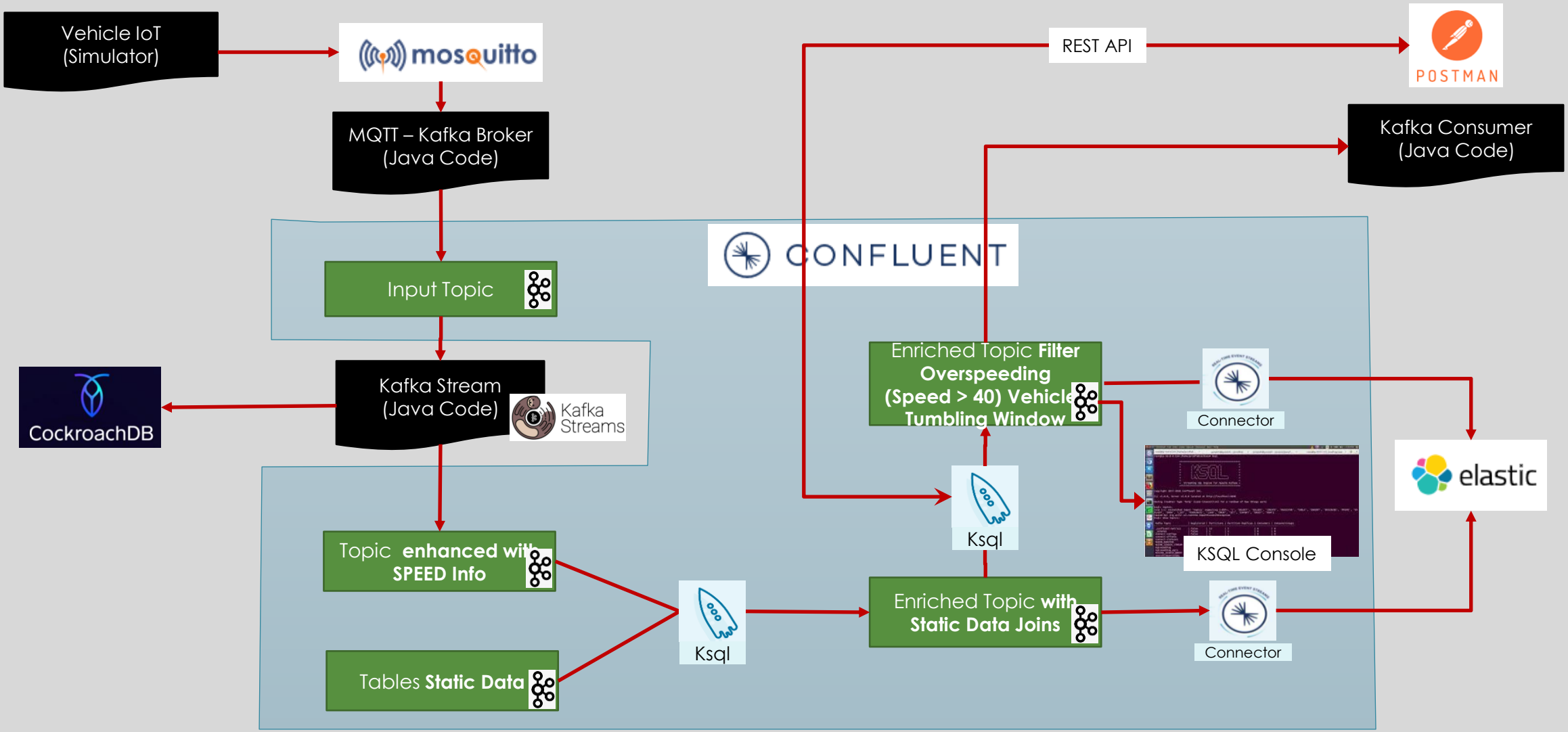
Project Outcomes

Task 1: Architecture diagram for the whole solution
Task 2: Database schema and implementation for Truck driver data storage
Task 3: Simulator program for the truck data movement over the period of time
Task 4: Data Transfer program moving the data from the truck to central server like Mosquito broker through MQTT protocol
Task 5: Data transfer program from Mosquito broker to Kafka Topic and a raw data storage
Task 6: Data preprocessing / filtering program for identifying over speeding cases
Task 7: Program to keep statistics about over speeding cases over the period of time, for different routes, for different trucks etc.
Task 8: A simple interface for showing over speeding statistics to the end consumers

Task 1: Architecture diagram for the whole solution (High Level)



Task 1: Architecture diagram for the whole solution (Detailed)



Technology Stack

Primary Stack

- Confluent Kafka Stack 7.x
- Confluent Connector Elasticsearch
- Mosquitto 2.x
- Cockroach DB 21.x
- Elastic Search 7.x
- Kibana 7.x
- JDK 1.8x
- Windows 10 OS
- Windows Subsystem for Linux (WSL 2)

Alternate Stack Explored

- Apache Kafka 2.x
- Mosquitto 2.x
- Kafka Connector Mosquitto
- Docker with following Images
  - confluentinc/cp-zookeeper:6.1.0
  - confluentinc/cp-kafka:6.1.0
  - confluentinc/cp-schema-registry:6.1.0
  - confluentinc/cp-kafka-connect-base:6.1.0
  - confluentinc/ksqldb-server:0.15.0
  - docker.elastic.co/elasticsearch/elasticsearch:7.11.0
  - docker.elastic.co/kibana/kibana:7.11.0
  - edenhill/kafkacat:1.7.0-PRE1

Reason for selecting Confluent Stack

- ✓ Ease of Installation : Configuring software on WSL is easy, and we get full stack of software required for stream processing in single installation. The only requirement is Disk Space and enabling WSL for Windows 10
- ✓ Ease of Development : With control center (UI) it becomes easy to monitor Topics, KSQL queries, Consumer Groups.

Findings while using alternate stack

- ✓ Getting / assembling the opensource stack for project was challenge especially getting KSQL and Schema Registry. Windows is not supported out of box
- ✓ With Docker, we get entire stack working on machine with single docker command but accessing data or providing port access of these software to host system is tricky. We were able to finally get entire stack working and we kept this as standby option.

Task 2: Database schema and implementation for Truck driver data storage

IOT Message

{"device_id": "9d5abed8530e4d24aa047debe5b04293", "driver_id": 0, "latitude": 25.0, "longitude": 110.95172238349915, "route_id": 10, "status": "RUNNING", "timestamp": 1631217609426, "vehicle_id": 0}

CockroachDB

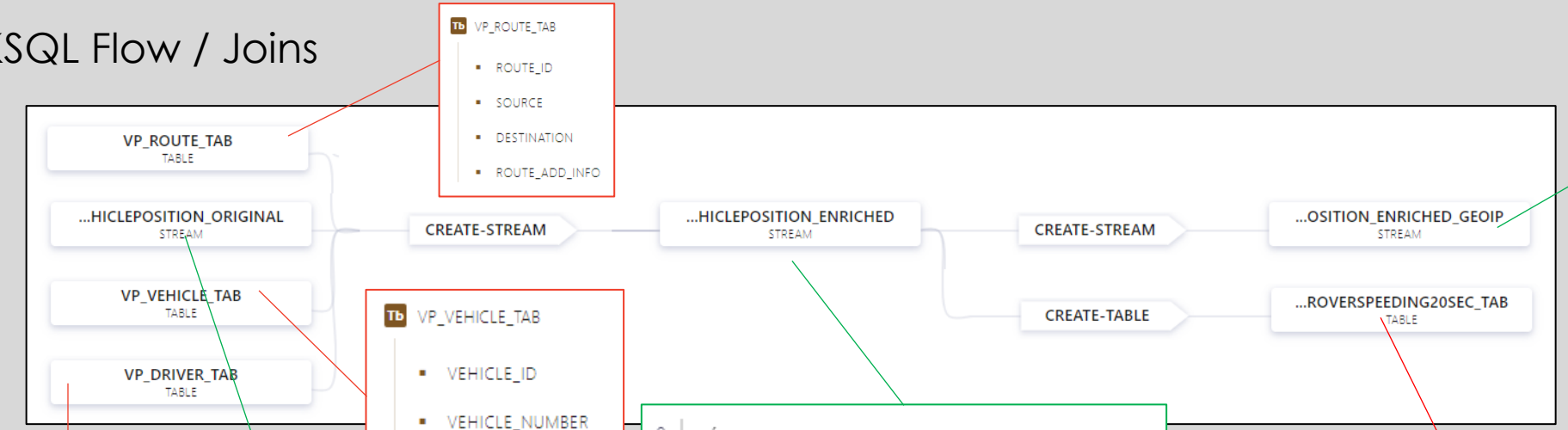
```
root@localhost:26257/defaultdb> show create table spaassignment2.vehicleposition;
              table_name              |                                                    create_statement
--------------------------------------+----------------------------------------------------------------------------------------------------------------------------------------
  spaassignment2.public.vehicleposition | CREATE TABLE public.vehicleposition (
                                      |     id INT8 NULL DEFAULT unique_rowid(),
                                      |     device_id STRING NOT NULL,
                                      |     vehicle_id INT8 NOT NULL,
                                      |     route_id INT8 NOT NULL,
                                      |     driver_id INT8 NOT NULL,
                                      |     latitude DECIMAL NULL,
                                      |     longitude DECIMAL NULL,
                                      |     "timestamp" INT8 NULL,
                                      |     status STRING NULL,
                                      |     message JSONB NULL,
                                      |     last_updated TIMESTAMP NULL DEFAULT now():::TIMESTAMP,
                                      |     rowid INT8 NOT VISIBLE NOT NULL DEFAULT unique_rowid(),
                                      |     CONSTRAINT "primary" PRIMARY KEY (rowid ASC),
                                      |     FAMILY "primary" (id, device_id, vehicle_id, route_id, driver_id, latitude, longitude, "timestamp", status, message, last_updated, rowid)
                                      | )
```

```
root@localhost:26257/defaultdb> SELECT  * FROM spaassignment2.vehicleposition order BY id DESC limit 1;
         id          |             device_id            | vehicle_id | route_id | driver_id | latitude |     longitude      |   timestamp   | status  |
         message                                                                                                          |        last_updated
---------------------+----------------------------------+------------+----------+-----------+----------+--------------------+---------------+---------+---------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------+----------------------------
  691887175890599937 | 9d5abed8530e4d24aa047debe5b04293 |          0 |       10 |         0 |       25 | 110.95172238349915 | 1631217609426 | RUNNING | {"device_id": "9d5abed8530e4d24aa047debe5b04293", "driver_id": 0, "latitude": 25.0, "
longitude": 110.95172238349915, "route_id": 10, "status": "RUNNING", "timestamp": 1631217609426, "vehicle_id": 0} | 2021-09-10 01:30:09.438666
(1 row)

Time: 34ms total (execution 33ms / network 0ms)
```

Task 2: Database schema and implementation for Truck driver data storage

KSQL Flow / Joins



VP_ROUTE_TAB
- ROUTE_ID
- SOURCE
- DESTINATION
- ROUTE_ADD_INFO

VP_VEHICLE_TAB
- VEHICLE_ID
- VEHICLE_NUMBER
- VEHICLE_ADD_INFO

VP_DRIVER_TAB
- DRIVER_ID
- DRIVER_NAME
- DRIVER_ADD_INFO


```
{
  "DEVICE_ID": "5b101deb34f146ba98b1f39005d2
  "VEHICLE_ID": 1,
  "ROUTE_ID": 16,
  "DRIVER_ID": 1,
  "LATITUDE": 66.08551782369614,
  "LONGITUDE": -63.810299813747406,
  "SPEED": 14.982065052458708,
  "TIMESTAMP": 1631034770370,
  "STATUS": "RUNNING"
}
```


original


```
{
  "ROUTE": 16,
  "DEVICE": "5b101deb34f146ba98b1f39005d2447
  "SOURCE": "Kothrud",
  "DESTINATION": "Hadapsar",
  "ROUTEDETAILS": "Intra City Route",
  "TRUCK": 1,
  "TRUCKNUMBER": "MH23-GA-1940",
  "TRUCKDETAILS": "Tata-Owner-Mauli-Maharash
  "DRIVER": 1,
  "DRIVERNAME": "Ghansyam Vaze",
  "DRIVERDETAILS": "ID-34567-Kolhapur",
  "LATITUDE": 48.553178191185,
  "LONGITUDE": -82.76910918951035,
  "SPEED": 0,
  "TIMESTAMP": 1631034685694,
  "STATUS": "BROKEN"
}
```


enriched

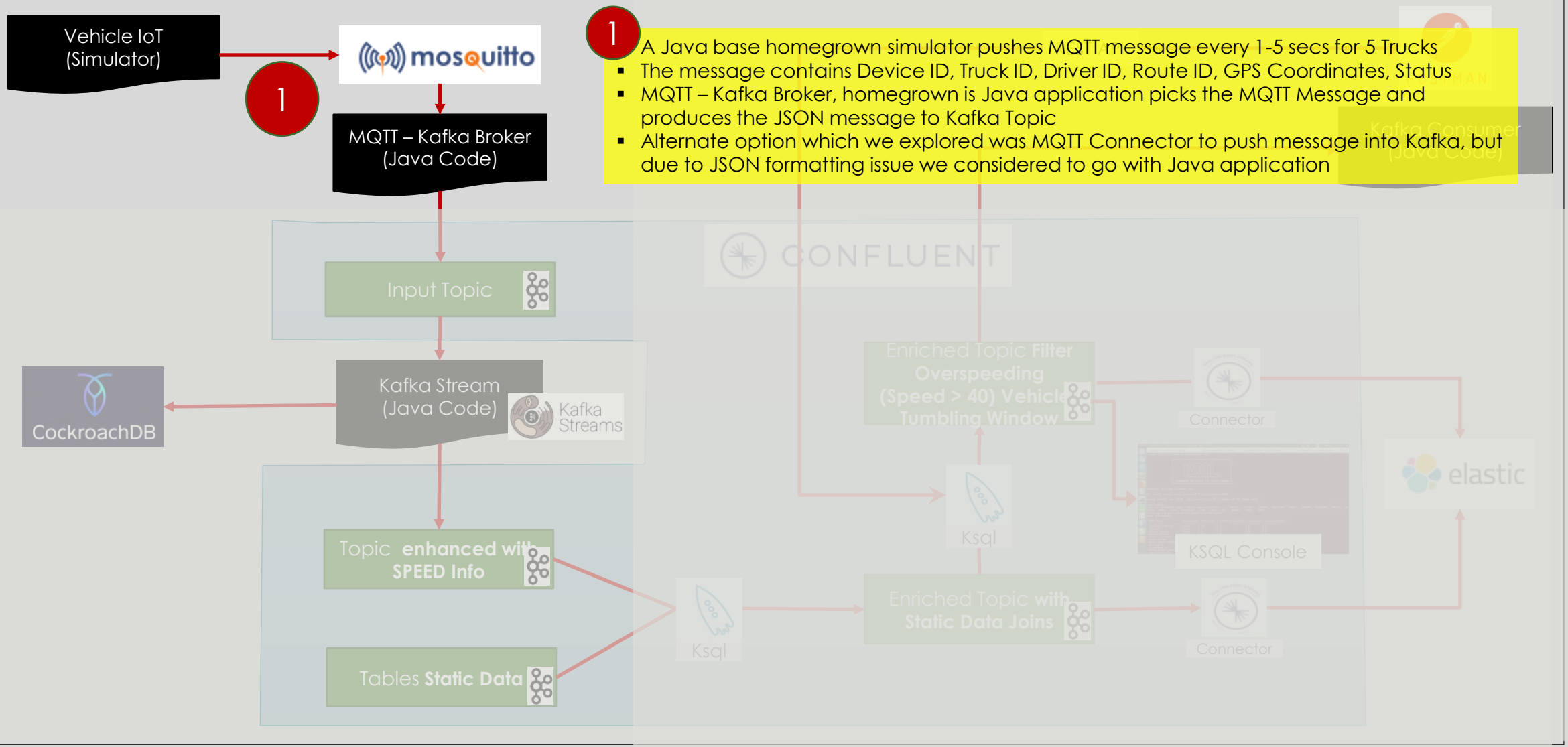

```
[
  {
    "COMPOSITEKEY": "Usman Ali::DL-AZ-1980::Pune::Delhi",
    "WINDOWSTART": 1631217580000,
    "WINDOWEND": 1631217600000,
    "OVERSPEEDCOUNT": 5,
    "VOILATIONS": [
      48.59282605516841,
      48.589209802517225,
      43.71236543053991,
      52.9473995648417,
      56.23102600314864
    ],
    "WINDOWSTART_TEXT": "2021-09-10 01:29:40.000",
    "WINDOWEND_TEXT": "2021-09-10 01:30:00.000",
    "S_TIMESTAMP": 1631217580000,
    "TIMESTAMP": 1631217600000
  }
]
```


Ktable voilations


```
{
  "ROUTE": 22,
  "DEVICE": "b04b421a6d014c12994cccf13d45a1b
  "SOURCE": "Delhi",
  "DESTINATION": "Pune",
  "ROUTEDETAILS": "Inter State Route",
  "TRUCK": 2,
  "TRUCKNUMBER": "UP-DE-1960",
  "TRUCKDETAILS": "Tesla-Owner-SinghBro-UP",
  "DRIVER": 2,
  "DRIVERNAME": "Dilbar Khan",
  "DRIVERDETAILS": "ID-67890-Sajjanpur",
  "LATITUDE": 48.60022222995758,
  "LONGITUDE": -80.85243117809296,
  "SPEED": 18.315001794192746,
  "TIMESTAMP": 1631034678566,
  "STATUS": "RUNNING",
  "location_example_01": {
    "lat": 48.60022222995758,
    "lon": -80.85243117809296
  }
```

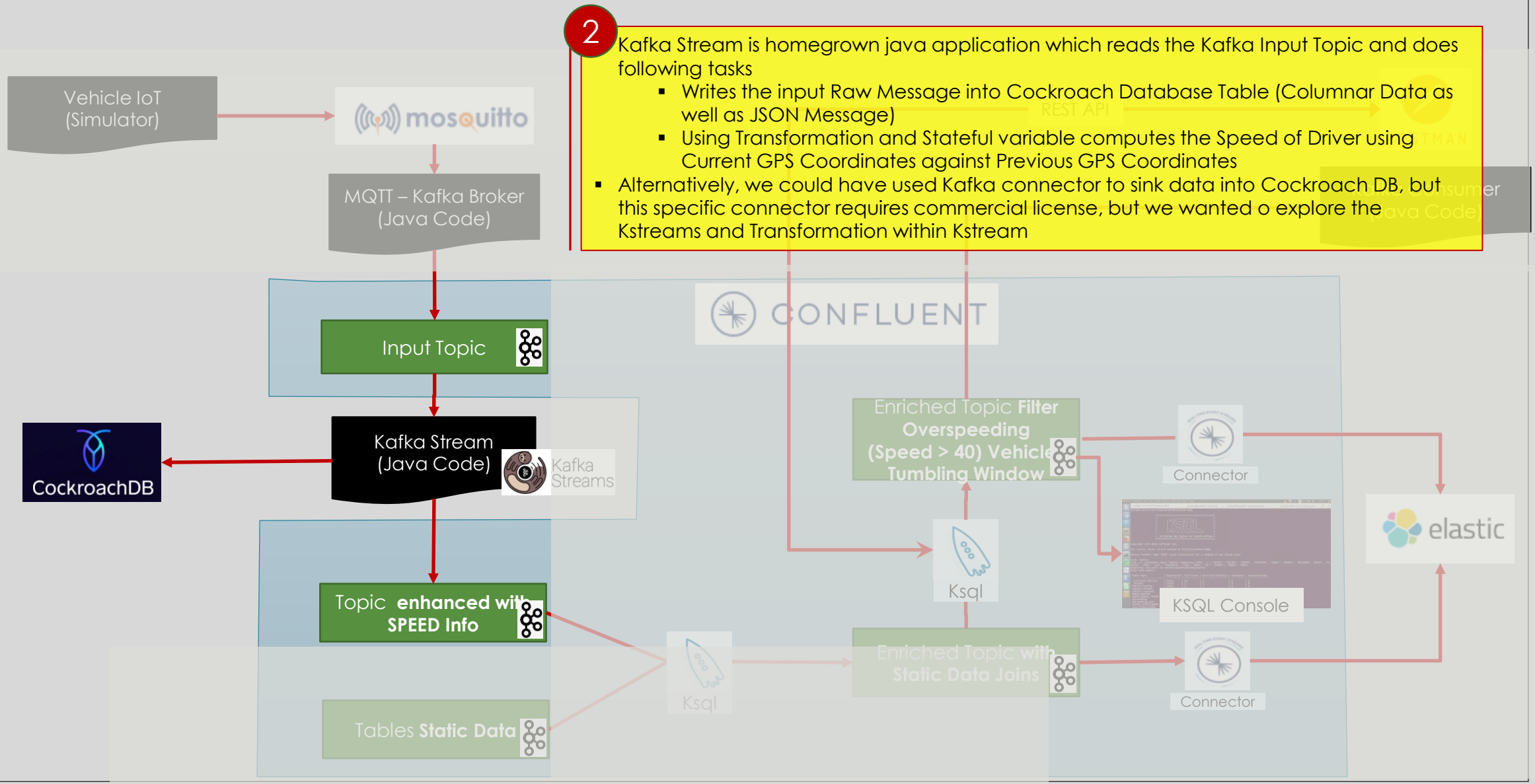

encriched with geolip formatting for elastic

Task 3: Simulator program for the truck data movement over the period of time
Task 4: Data Transfer program moving the data from the truck to central server like Mosquito broker through MQTT protocol

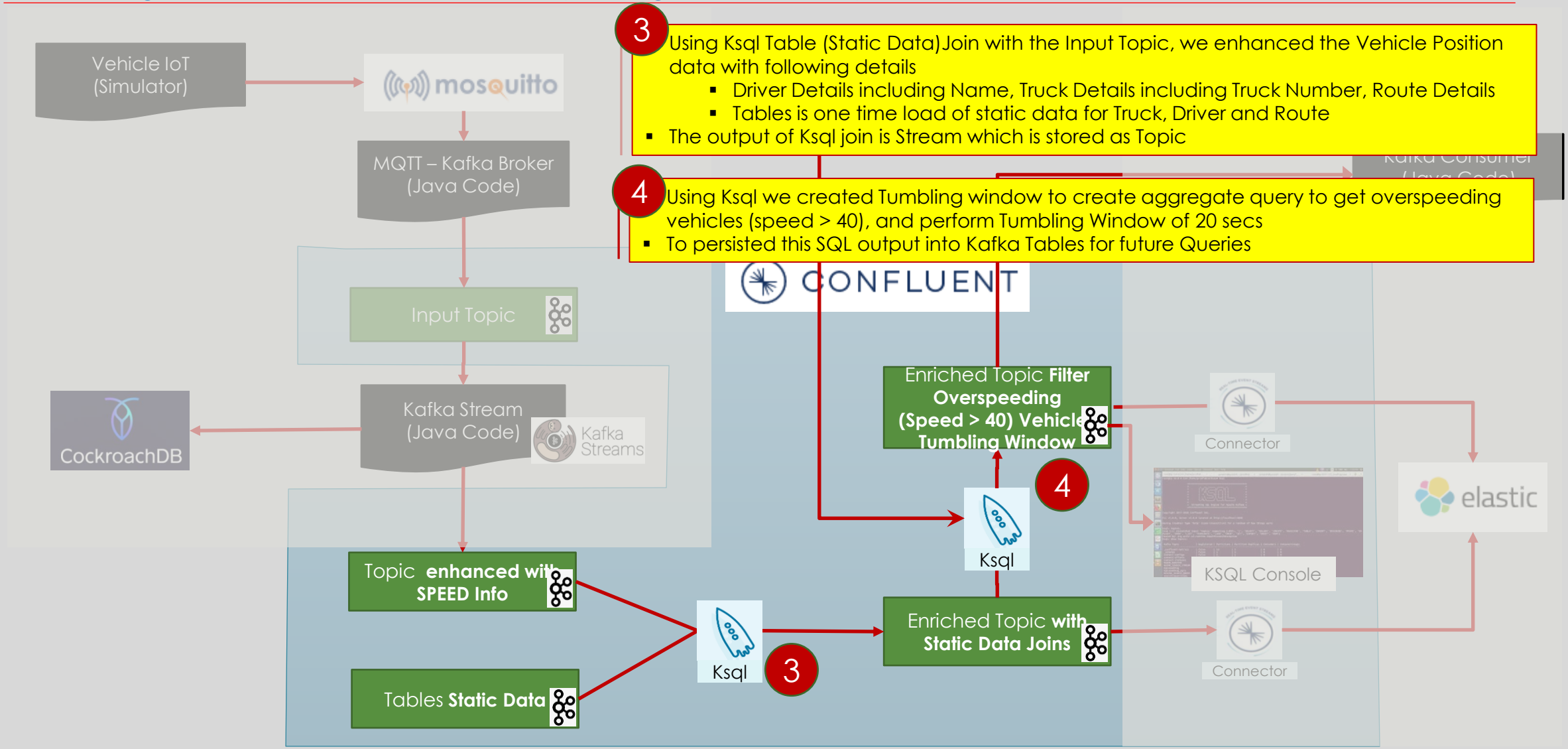


1. A Java base homegrown simulator pushes MQTT message every 1-5 secs for 5 Trucks
   - The message contains Device ID, Truck ID, Driver ID, Route ID, GPS Coordinates, Status
   - MQTT – Kafka Broker, homegrown is Java application picks the MQTT Message and produces the JSON message to Kafka Topic
   - Alternate option which we explored was MQTT Connector to push message into Kafka, but due to JSON formatting issue we considered to go with Java application

Task 5: Data transfer program from Mosquito broker to Kafka Topic and a raw data storage

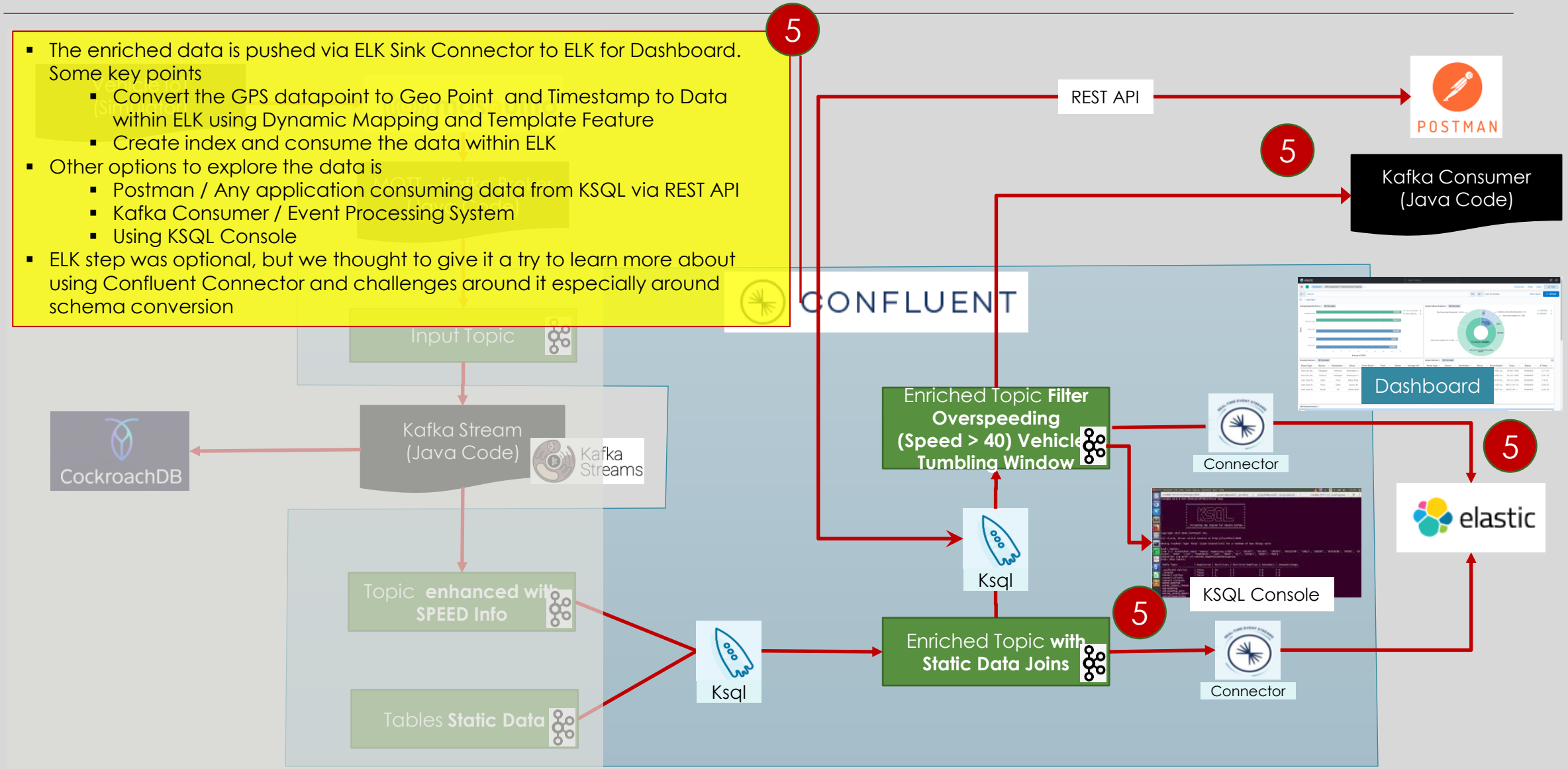


Task 6: Data preprocessing / filtering program for identifying over speeding cases
Task 7: Program to keep statistics about over speeding cases over the period, for different routes, for different trucks etc.



Task 8: A simple interface for showing over speeding statistics to the end consumers

- The enriched data is pushed via ELK Sink Connector to ELK for Dashboard. Some key points
  - Convert the GPS datapoint to Geo Point and Timestamp to Data within ELK using Dynamic Mapping and Template Feature
  - Create index and consume the data within ELK
- Other options to explore the data is
  - Postman / Any application consuming data from KSQL via REST API
  - Kafka Consumer / Event Processing System
  - Using KSQL Console
- ELK step was optional, but we thought to give it a try to learn more about using Confluent Connector and challenges around it especially around schema conversion



Java Code Structure

```
|- TruckDataGenerator
   |- TruckIoTData
   |- MQTTProducer
      |- MessageActionListener
|- MQTTKafkaBridge
|- TruckKafkaStreamProcess
  |- TruckIoTData (re-used)
  |- TruckIoTDataWithSpeed
  |- StreamSerdes
  |- JsonSerializer
  |- JsonDeserializer
  |- TruckGPSSpeedTransformer
  |- VehiclePositionCockroahDBPersister
```

This is Data Generator main class & supporting classes ***to post the message to Mosquitto Broker***

This is MQTT Kafka Bridge Class ***to read message from Mosquitto Broker and write to Kafka Topic***

This is Kafka Stream main class and supporting classes ***to process kafka messages as stream it reads message from Kafka Topic and Writes to Cockroach DB and performs Transformation by Computing Speed of the GPS coordinates against the previous GPS coordinates stored in State***

KSQL Join (to join static information) -> Stream

```
CREATE STREAM ST_MQTTVEHICLEPOSITION_ENRICHED
WITH (KAFKA_TOPIC='ST_MQTTVEHICLEPOSITION_ENRICHED', PARTITIONS=1, REPLICAS=1)
AS SELECT
  ORG.DEVICE_ID DEVICE,
  ORG.ROUTE_ID ROUTE,
  ROUTE.SOURCE SOURCE,
  ROUTE.DESTINATION DESTINATION,
  ROUTE.ROUTE_ADD_INFO ROUTEDETAILS,
  ORG.VEHICLE_ID TRUCK,
  VEHICLE.VEHICLE_NUMBER TRUCKNUMBER,
  VEHICLE.VEHICLE_ADD_INFO TRUCKDETAILS,
  ORG.DRIVER_ID DRIVER,
  DRIVER.DRIVER_NAME DRIVERNAME,
  DRIVER.DRIVER_ADD_INFO DRIVERDETAILS,
  ORG.LATITUDE LATITUDE,
  ORG.LONGITUDE LONGITUDE,
  ORG.SPEED SPEED,
  ORG.TIMESTAMP TIMESTAMP,
  ORG.STATUS STATUS
FROM ST_MQTTVEHICLEPOSITION_ORIGINAL ORG
LEFT OUTER JOIN VP_DRIVER_TAB DRIVER ON ((ORG.DRIVER_ID = DRIVER.DRIVER_ID))
LEFT OUTER JOIN VP_VEHICLE_TAB VEHICLE ON ((ORG.VEHICLE_ID = VEHICLE.VEHICLE_ID))
LEFT OUTER JOIN VP_ROUTE_TAB ROUTE ON ((ORG.ROUTE_ID = ROUTE.ROUTE_ID))
EMIT CHANGES;
```

```
CREATE TABLE VP_DRIVEROVERSPEEDING20SEC_TAB
WITH (KAFKA_TOPIC='VP_DRIVEROVERSPEEDING20SEC_TAB', PARTITIONS=1, REPLICAS=1)
AS SELECT
  ((((((CAST(SRC.DRIVERNAME AS STRING) + '::') +
           CAST(SRC.TRUCKNUMBER AS STRING)) + '::') +
           CAST(SRC.SOURCE AS STRING)) + '::') +
           CAST(SRC.DESTINATION AS STRING)) COMPOSITEKEY,
  COUNT(*) OVERSPEEDCOUNT,
  COLLECT_LIST(SRC.SPEED) VOILATIONS,
  TIMESTAMPTOSTRING(WINDOWSTART, 'yyyy-MM-dd HH:mm:ss.SSS') WINDOWSTART_TEXT,
  TIMESTAMPTOSTRING(WINDOWEND, 'yyyy-MM-dd HH:mm:ss.SSS') WINDOWEND_TEXT,
  WINDOWSTART S_TIMESTAMP,
  WINDOWEND TIMESTAMP
FROM ST_MQTTVEHICLEPOSITION_ENRICHED SRC
WINDOW TUMBLING ( SIZE 20 SECONDS )
WHERE ((SRC.SPEED > 40) AND (SRC.STATUS = 'RUNNING'))
GROUP BY ((((((CAST(SRC.DRIVERNAME AS STRING) + '::') +
                     CAST(SRC.TRUCKNUMBER AS STRING)) + '::') +
                     CAST(SRC.SOURCE AS STRING)) + '::') +
                     CAST(SRC.DESTINATION AS STRING))
EMIT CHANGES;
```

Filtering and Tumbling Window

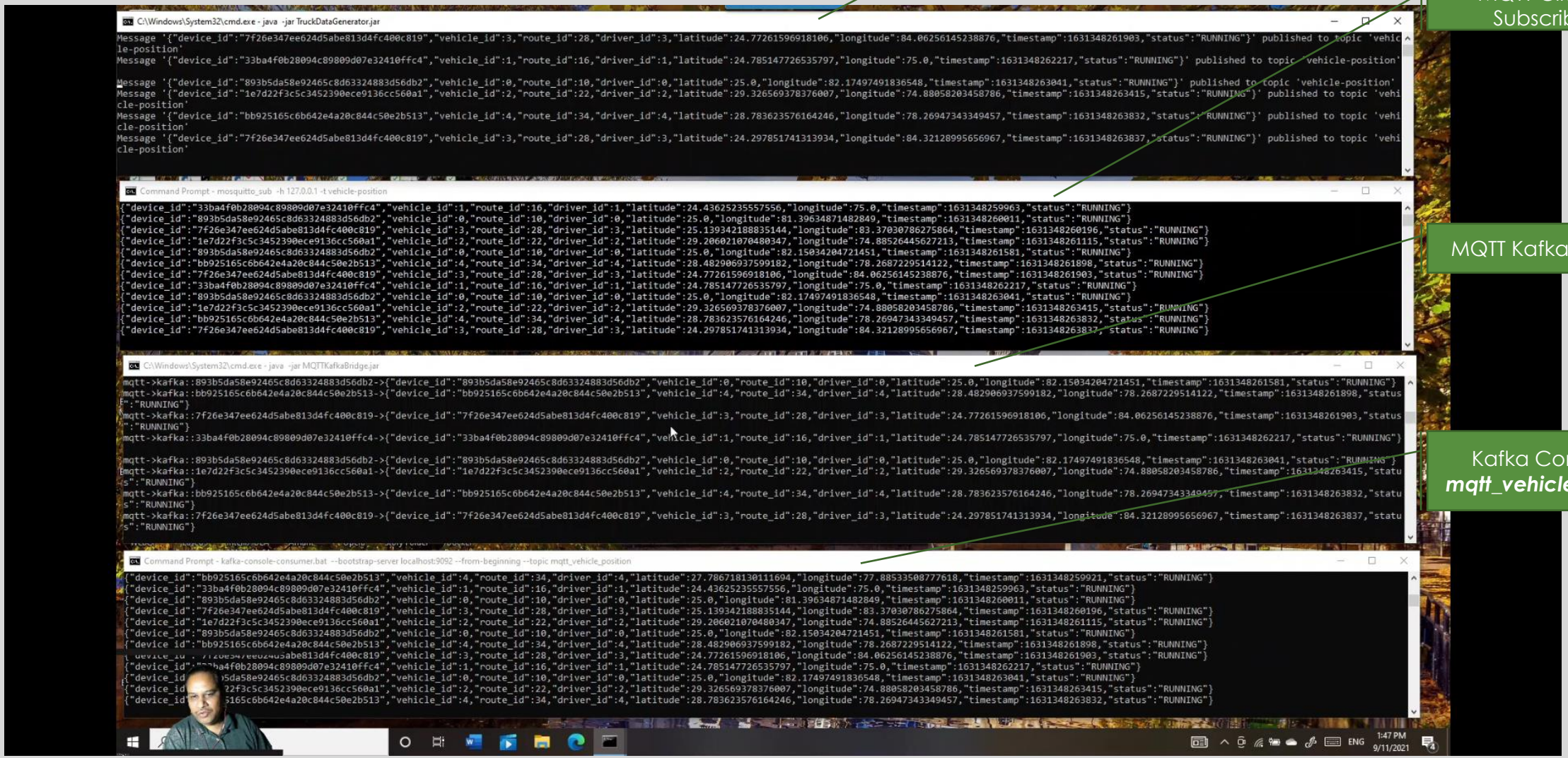
KSQL Aggregate using Tumbling Window -> Table

Elastic Sink Connector Kafka -> ELK

```
CREATE SINK CONNECTOR SINK_ELASTIC_VP_JSON_03 WITH (
'connector.class'                    = 'io.confluent.connect.elasticsearch.ElasticsearchSinkConnector',
'topics'                             = 'VP_DRIVEROVERSPEEDING20SEC_TAB',
'key.converter'                      = 'org.apache.kafka.connect.storage.StringConverter',
'value.converter'                    = 'org.apache.kafka.connect.json.JsonConverter',
'value.converter.schemas.enable'     = 'false',
'connection.url'                     = 'http://localhost:9200',
'type.name'                          = '_doc',
'key.ignore'                         = 'false',
'schema.ignore'                      = 'true',
'errors.tolerance'                   = 'all',
'behavior.on.malformed.documents'    = 'IGNORE'
);
```

Error Handling

Demo Time !


Data Generator
MQTT Client / Subscriber
MQTT Kafka Bridge
Kafka Consumer
mqtt_vehicle_position
C:\Windows\System32\cmd.exe - java -jar TruckDataGenerator.jar
Command Prompt - mosquitto_sub -h 127.0.0.1 -t vehicle-position
C:\Windows\System32\cmd.exe - java -jar MQTTKafkaBridge.jar
Command Prompt - kafka-console-consumer.bat --bootstrap-server localhost:9092 --from-beginning --topic mqtt_vehicle_position

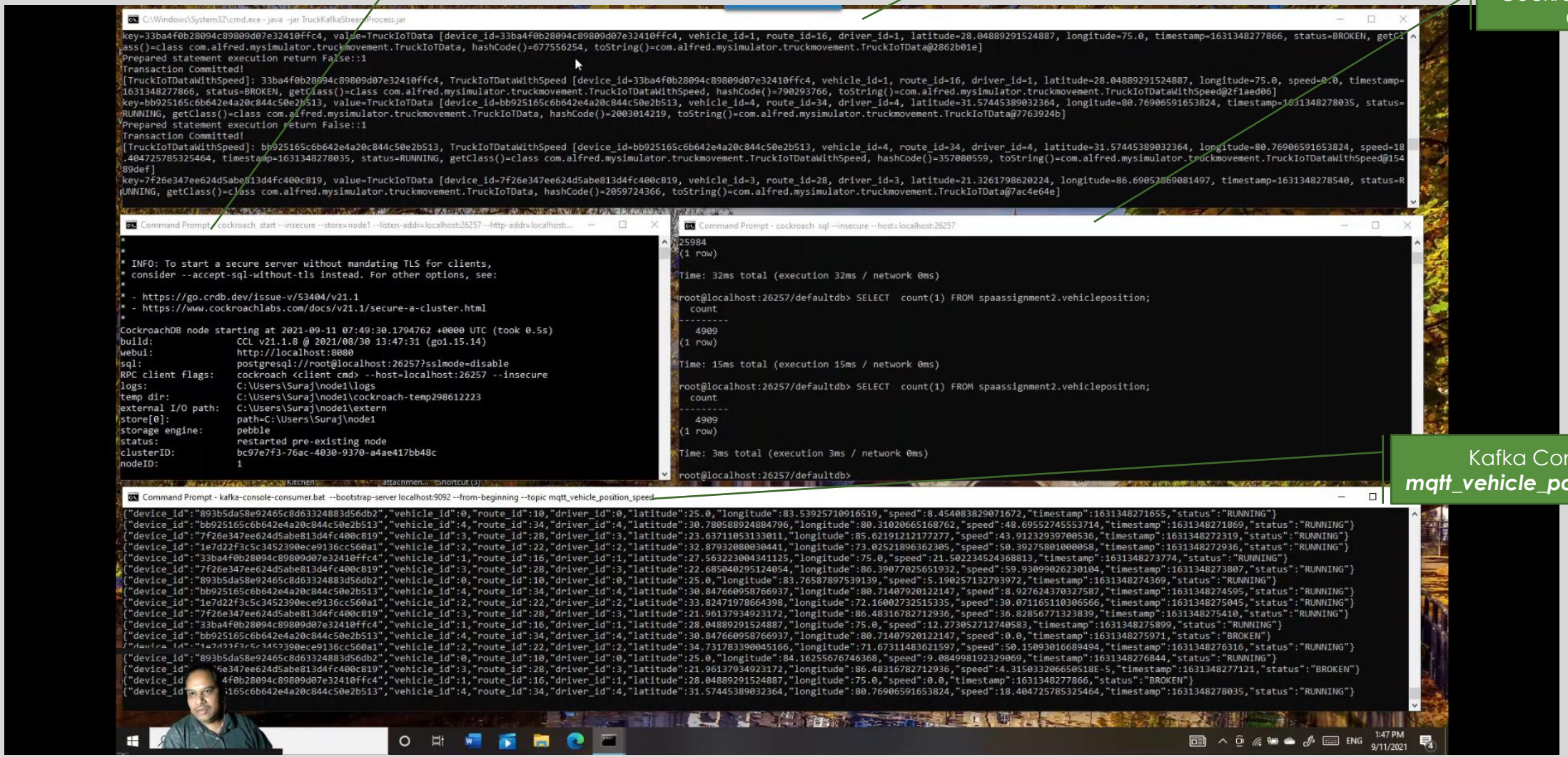
Demo Time !

Coclroach DB Engine

Kafka Stream Processor

Cockroach DB SQL Client

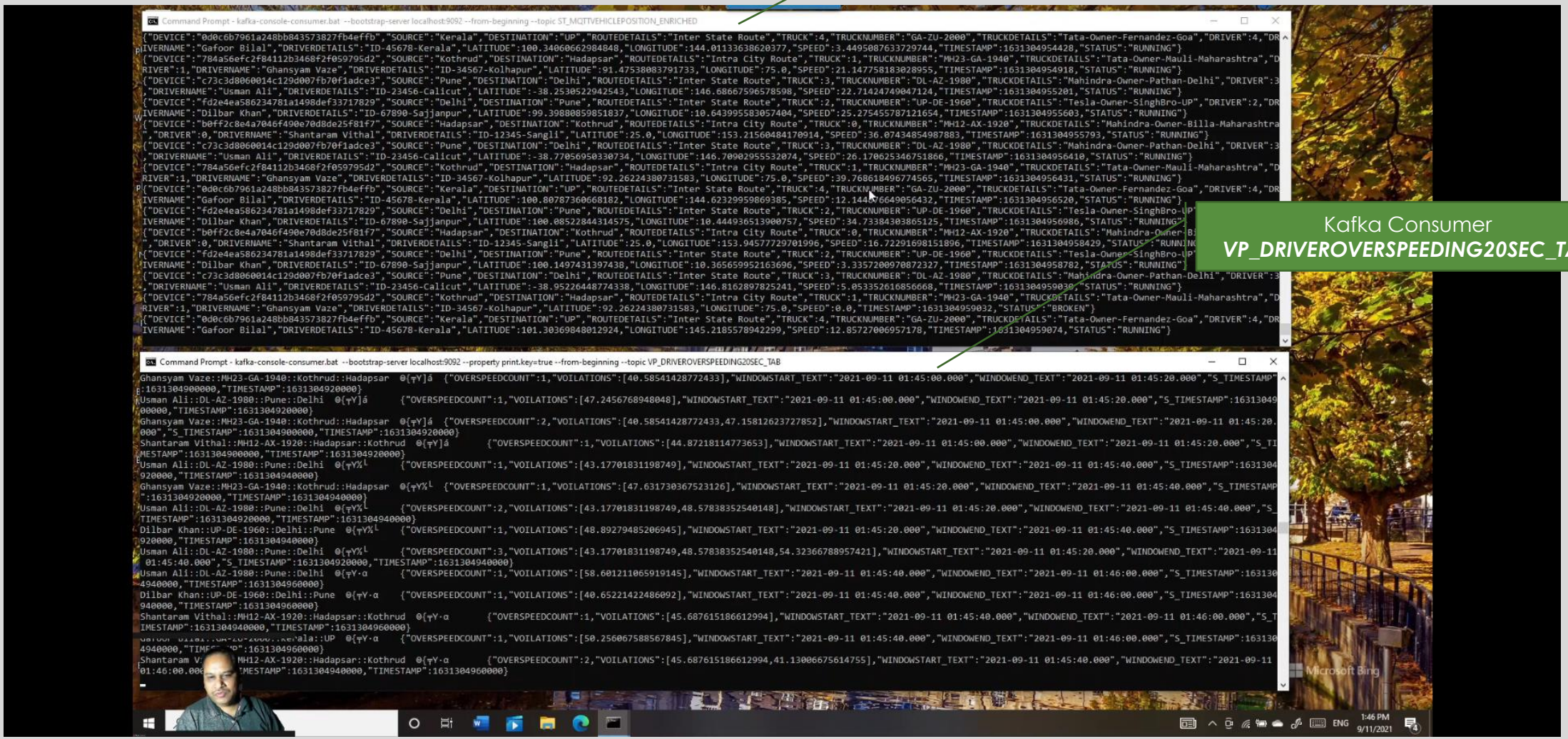
Kafka Consumer
mqtt_vehicle_position_speed



Demo Time !

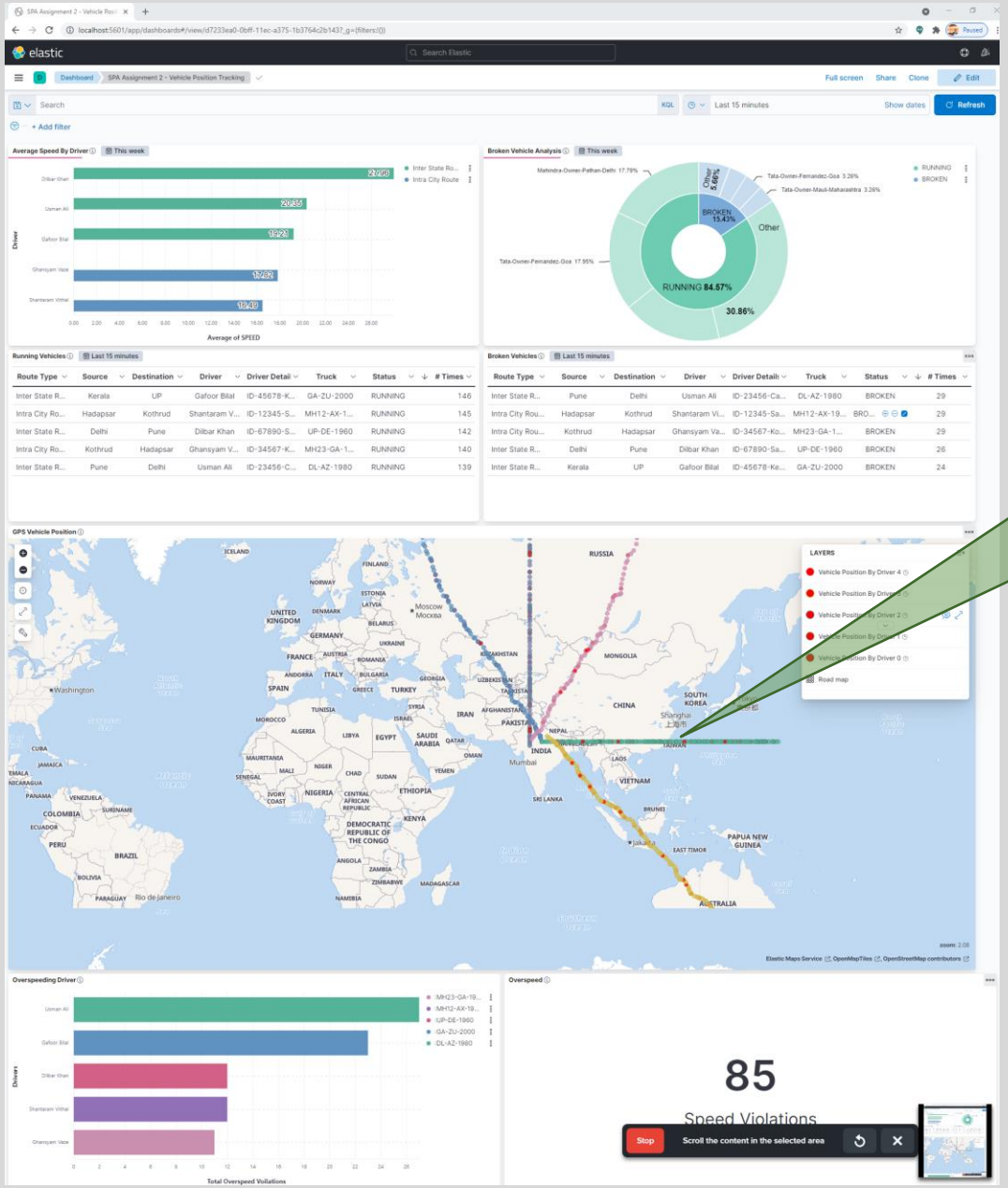
Kafka Consumer
ST_MQTTVEHICLEPOSITION_ENRICHED

Kafka Consumer
VP_DRIVEROVERSPEEDING20SEC_TAB

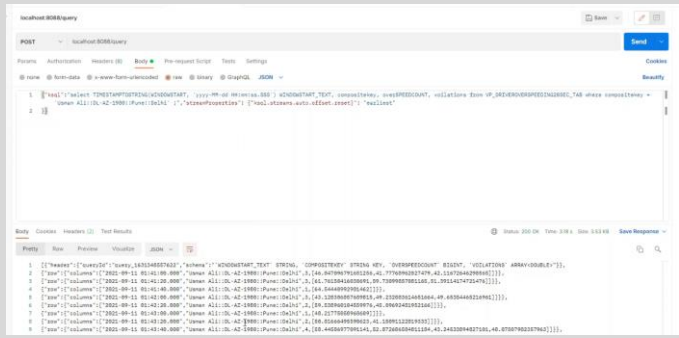


Elastic Kibana Dashboard with Tooltip showing additional details for route

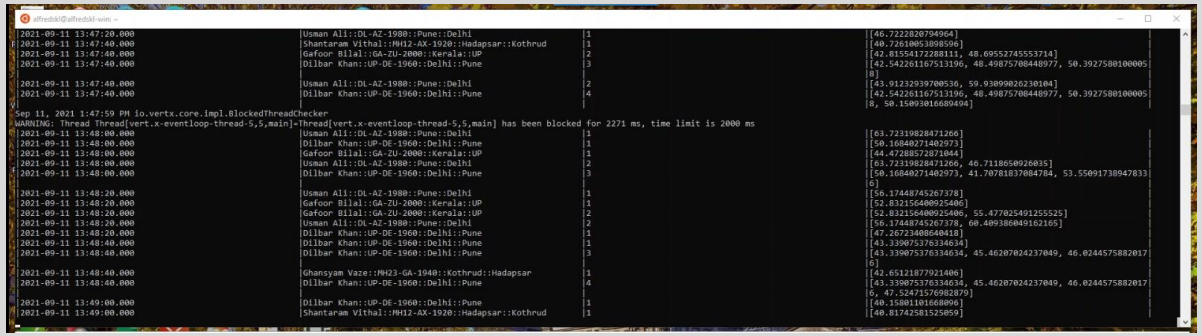
Demo Time !



Postman



KSQL Console - Query



Key Learnings

- MQTT Protocol , and how to read / write MQTT Broker
- Connecting to Kafka Broker and write to Kafka Topic, *this can also be achieved using Confluent Connector, but we choose to write Java Code, as we wanted to set the specific Key in Kafka and moreover, we wanted to understand the protocol ourselves*
- Java Kafka Stream perform pipeline, *these tasks can also be achieved using multiple connectors, but we choose to write the code as we wanted to understand the details of Kafka, and computing Speed was a challenge as we want to compare 2 different events and compare GPS coordinates*
  - Serialize and De-serialize the stream
  - Read the Stream (without moving the offset – Peek) and write it to Cockroach DB
  - Transform the Stream using Storage
    - Store the Previous GPS Coordinates in memory
    - Against current GPS coordinates, compute the SPEED
  - Write the output to Kafka Topic
- Explored various KSQL commands
  - Join Static Table Joins to get Driver, Truck, Route details using Left Join
  - Aggregate Query using Window Function
  - Show / Create / Drop Connectors
  - Print Topic
  - Run Query
- Confluent Connector Configuration for Elastic Sink
  - Stream Error Handling
  - Ignoring / Not Ignoring Key
  - Serialisation / Deseriallise
  - Use REST endpoint to Query Connector
- Elastic & Kibana Index Document Handling and creating Dashboard
  - Index and Index Patterns
  - How to create / use timestamp from data feed
  - Data type conversion for geoip (to use map), timestamp
  - Using REST API to find indexes, and query the document

Thank You!