University of Texas Permian Basin (UTPB)

Odessa, Texas

THE UNIVERSITY OF TEXAS
PERMIAN BASIN

Project Report On
"Real-Time Geospatial Analytics and Visualization"

Prakash Madai (8000257558)

Under the guidance of
Quan Yuan
Assistant Professor, Dept of Computer Science

Contents

Abstract

Inspired by recent success of geospatial analytics in transportation, we proposed an approach to visualize and tracks the real-time location of buses and taxis in Greater Boston, Massachusetts using MBTA API. Along with this we have introduced a complete end to end data pipeline from data extraction to real time data analytics map visualization using streaming data instead of going through the traditional analytics or data warehouse process using distributed system technologies like Kafka, Spark, and MongoDB. This enables immediate action, allowing businesses to be proactive by seizing opportunities or preventing problems before they happen eventually leading to faster decisions and improves business agility. We have achieved a desirable result both in tracking vehicles and obtaining events status of the vehicles using distributed data processing using Spark.

***Keywords**: MBTA, Spark, Kafka, MongoDB*

Introduction

Geospatial data is data that correlates to a geographic location and contains a geographic component. E.g.: an address with a city, longitude, latitude, state, and zip code. We have implemented a distributed ETL pipeline visualizes or tracks the real-time location of vehicles in Greater Boston, Massachusetts using MBTA API. Massachusetts Bay Transportation Authority (MBTA) operates heavy-rail, light-rail, and bus transit services in the Boston metropolitan area.
This multi-component full-stack solution shows the capabilities of what could be done with modern open-source technologies, especially Apache Spark and Kafka. For this application, the other clients could be the Transportation Department or Police department who are interested in tracking the locations of vehicles that are running in public transport area of any location. In our project, we take the real-time data with the help of Kafka and processed with spark and populate the map with the help of ReactJS as front-end framework. For server and client side, we have used Node.JS as our server environment. As a backend database we have used MongoDB which is a document-oriented database which serves perfect for storing big data and manipulating real time data.

Motivation

The traditional data and data warehouse did not make best use of real time data as today with numerous possibilities. Hence, it is difficult to retrieve these data and process them due to huge amount of data generated in short time. Furthermore, compared to present needs, its data analytics is not updated in real time, which may lead to bad and late decisions. So, considering the present needs and situation, this project is developed. This is because this project uses real-time data with the latest technology which can hold large amounts of data faster in manipulating and accessing. Coming to the application of this project, if we want to see the current application, we can find numerous areas where it can solve lots of problems. This application can be used in the field of transportation as it helps the department to track the real time location of vehicles. One more application could be great for the Police, which helps them to track the vehicles which can lead in controlling traffic. We can alert emergency vehicles by providing them the traffics around the area for route diversion. It would play a great role in crime as it can track the vehicle of criminals.

Related Work / Literature Review

(Pradip V Mistary & R H Chile, 2015) **[1].** In this paper, we got an overall idea of a vehicle tracking system that tracks the exact location of a moving or stationary vehicle in real-time. This paper provided the overall design and development of the system in real-time successfully. This system provides positioning and navigational information in terms of several parameters. If required, information regarding the satellites being tracked by the system was also displayed. From this paper, we got an idea about the real-time tracing system for vehicles along with the help of GSM, GPS, and Google Earth. So, this is the survey where we came up with the idea to implement a real-time tracking system with a distributed approach that can work with big data and real-time.

(Nikitha Johnsirani Venkatesan et al., 2017) **[2]** Since our aim is to work with a large, distributed dataset. So, we have come up with the idea of using Spark to work with a large real-time dataset. Here this survey paper gives the overall idea of streaming. Streams, analyzing real-time data become much quicker without latency. Spark streaming with Streams allows users to interact seamlessly with batch and interactive queries. The main idea behind Streams is to treat the streaming data as a series of mini-batch computations on tiny intervals, say in milliseconds. So, from this, we got an idea about the spark and how it works with a real-time large set of data with quick response.

(Kul et al., 2021) **[3]** From this project, we got an idea about the Apache Kafka stream. In this paper, we observed how to leverage Apache Kafka and implement a stream processing system in practice. Kafka performs data ingestion tasks and makes it possible to perform machine learning algorithms in real time. We aim to improve the efficiency of a real-time stream data process. One more major point that we gain from this paper is about our future work. This paper provides us with some information about tracking the vehicle and knowing the vehicle information with the help of Apache Kafka stream data.

Proposed Methodology

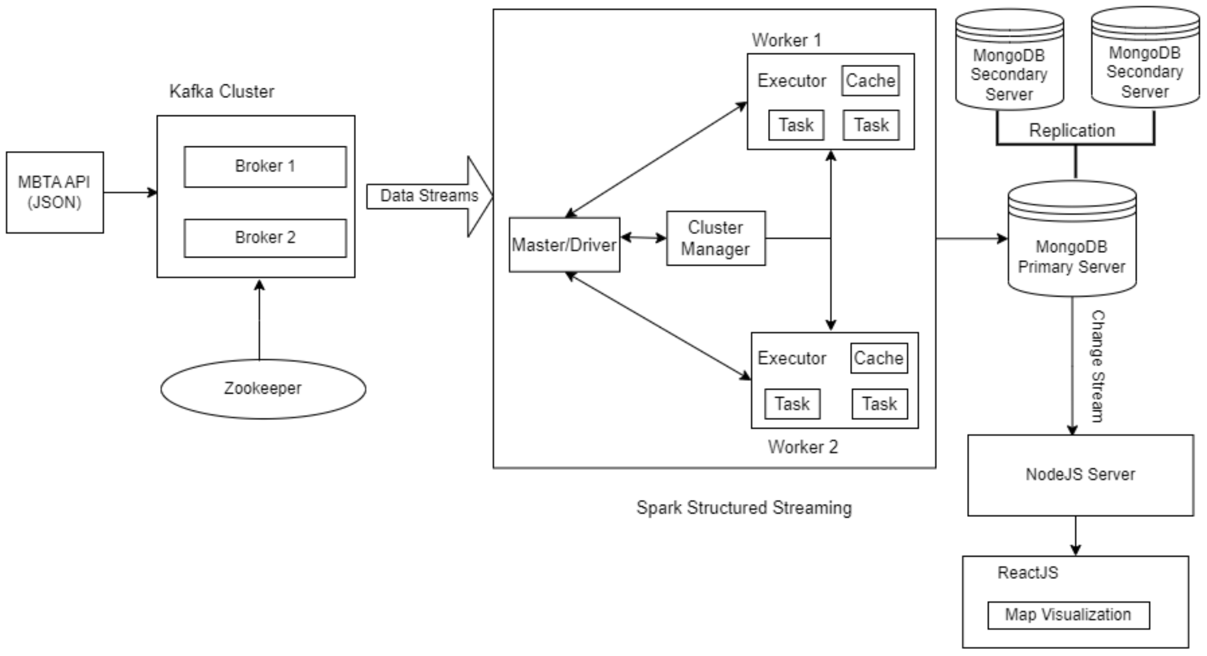


Fig: System Flow Diagram

Steps:

1. **Data Ingestion:**

Data ingestion is the process of obtaining and importing data for immediate use or storage in a database. Data can be streamed in real time or ingested in batches. In real-time data ingestion, each data item is imported as the source emits it. The data was extracted from MBTA API. The data ingestion was done using Apache Kafka which is Open-source distributed, fault tolerant event streaming platform used for scalable high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Components

1. **Producer**: It push messages to a Kafka cluster from any different environment
2. **Broker**: Brokers are stateless server within a Kafka cluster. Having multiple brokers encourage reliability and provides load-balancing. It handles thousands of large messages per second

3. **Topic**: Bucket where messages are published and from where the messages are consumed
4. **Consumer**: Consumers consumes a message from a Kafka topic running on different environment. Consumers within a consumer group can consume messages across multiple topics which offers scalability.

5. **Zookeeper**: It Facilitates configuration, coordination between servers, leader election and related management tasks.

Full API Documentation: https://api-v3.mbta.com/docs/swagger/index.html

API Response JSON:

```
C:\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost: 9092 --topic mbta_msgs
```

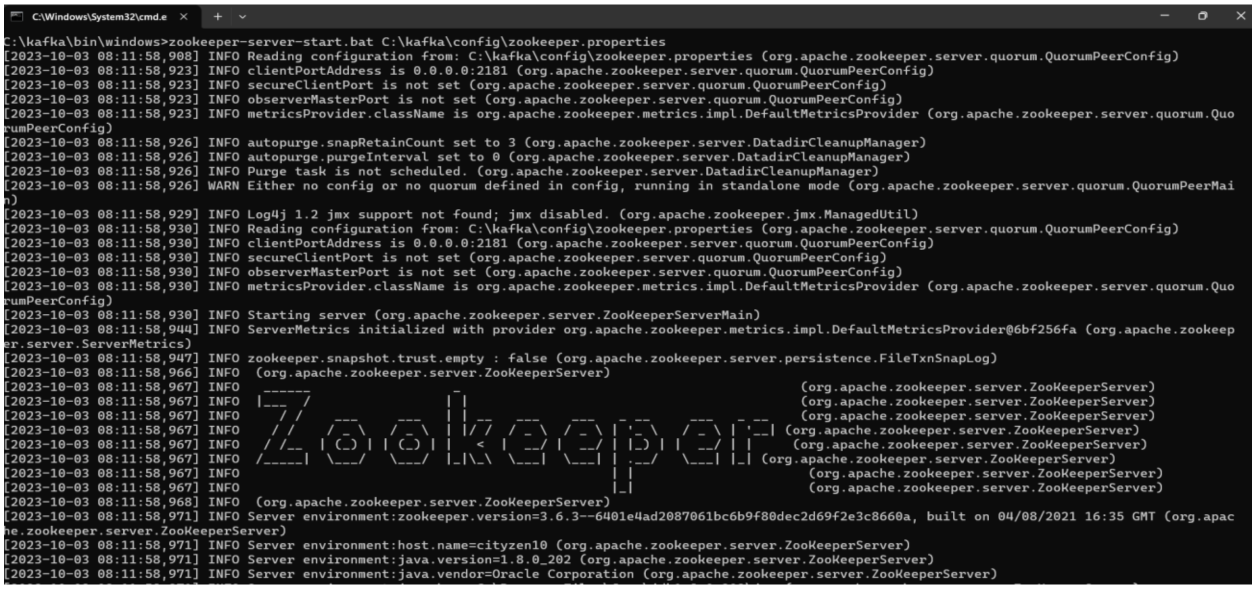
```
{
   "event": "update",
   "data": {
      "attributes": {
         "bearing": 38,
         "carriages": [],
         "current_status": "IN_TRANSIT_TO",
         "current_stop_sequence": 1,
         "direction_id": 1,
         "label": "0820",
         "latitude": 42.27218861,
         "longitude": -70.950601789,
         "occupancy_status": "MANY_SEATS_AVAILABLE",
         "speed": null,
         "updated_at": "2023-10-03T09:19:59-04:00"
      },
      "id": "y0820",
      "links": {
         "self": "/vehicles/y0820"
      },
      "relationships": {
         "route": {
            "data": {
               "id": "216",
               "type": "route"
            }
         },
         "stop": {
            "data": {
               "id": "3265",
                "type": "stop"
            }
        },
        "trip": {
            "data": {
                "id": "58751363",
                "type": "trip"
            }
        }
    },
    "type": "vehicle"
  }
}
```

Field	**Description**
current_stop_sequence	Index of current stop along trip
current_status. 1. ***INCOMING_AT***: The vehicle is just about to arrive at the stop (on a stop display, the vehicle symbol typically flashes) 2. ***STOPPED_AT***: The vehicle is standing at the stop 3. ***IN_TRANSIT_TO***:The vehicle has departed the previous stop and is in transit.	Status of vehicle relative to the stops
bearing	Bearing, in degrees, clockwise from True North, i.e., 0 is North and 90 is East. This can be the compass bearing, or the direction towards the next stop or intermediate location
trip	The trip which the vehicle is currently operating
stop	The vehicle's current (when current_status is STOPPED_AT) or next stop
route	The one route that is designated for that trip
longitude	Longitude of the vehicle's current position. Degrees East, in the WGS-84 coordinate system
latitude	Latitude of the vehicle's current position. Degrees North, in the WGS-84 coordinate system
updated_at	Time at which vehicle information was last updated. Format is ISO8601

label	User visible label, i.e., something that must be shown to the passenger to help identify the correct vehicle

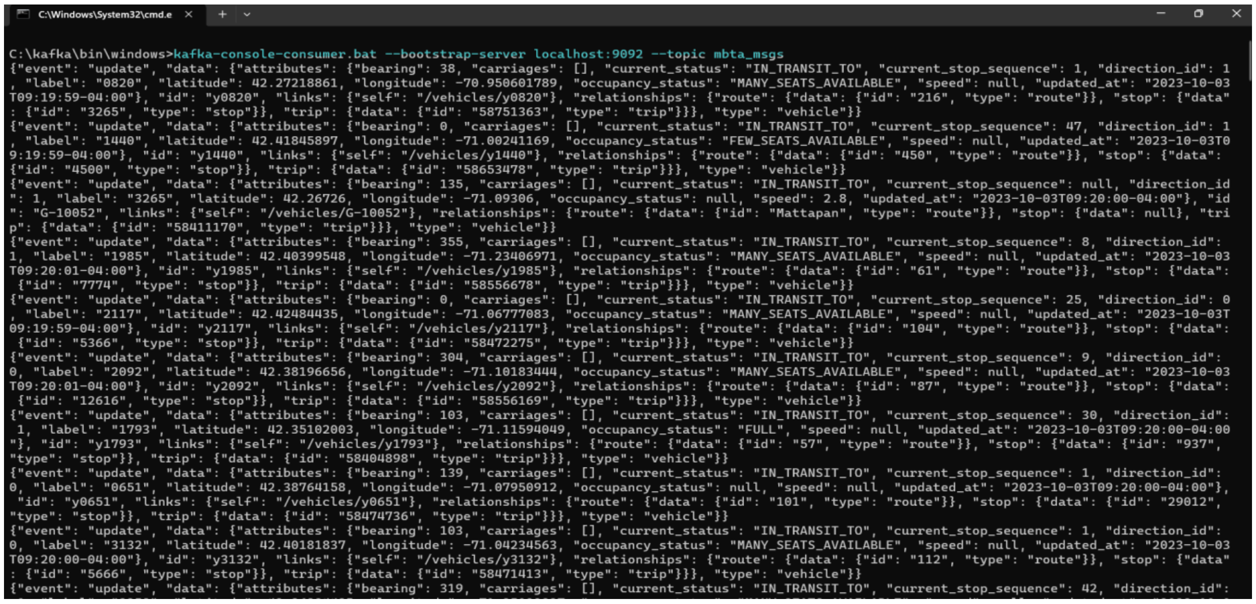
Kafka Setup

1. Zookeeper Setup on port 2181



```
C:\kafka\bin\windows>zookeeper-server-start.bat C:\kafka\config\zookeeper.properties
[2023-10-03 08:11:58,908] INFO Reading configuration from: C:\kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,923] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,923] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,923] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,923] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.Quo
rumPeerConfig)
[2023-10-03 08:11:58,926] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2023-10-03 08:11:58,926] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2023-10-03 08:11:58,926] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2023-10-03 08:11:58,926] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMai
n)
[2023-10-03 08:11:58,929] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2023-10-03 08:11:58,930] INFO Reading configuration from: C:\kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,930] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,930] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,930] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2023-10-03 08:11:58,930] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.Quo
rumPeerConfig)
[2023-10-03 08:11:58,930] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2023-10-03 08:11:58,944] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@6bf256fa (org.apache.zookeep
er.server.ServerMetrics)
[2023-10-03 08:11:58,947] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2023-10-03 08:11:58,966] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO   ______                  _                                   (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO  |___  /                 | |                                  (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO     / /    ___     ___   | | __   ___    ___   _ __     ___   _ __   (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO    / /    / _ \   / _ \  | |/ /  / _ \  / _ \ | '_ \   / _ \ | '__| (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO   / /__  | (_) | | (_) | |   <  |  __/ |  __/ | |_) | |  __/ | |     (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO  /_____|  \___/   \___/  |_|\_\  \___|  \___| | .__/   \___| |_| (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO                                               | |                     (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,967] INFO                                               |_|                     (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,968] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,971] INFO Server environment:zookeeper.version=3.6.3--6401e4ad2087061bc6b9f80dec2d69f2e3c8660a, built on 04/08/2021 16:35 GMT (org.apac
he.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,971] INFO Server environment:host.name=cityzen10 (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,971] INFO Server environment:java.version=1.8.0_202 (org.apache.zookeeper.server.ZooKeeperServer)
[2023-10-03 08:11:58,971] INFO Server environment:java.vendor=Oracle Corporation (org.apache.zookeeper.server.ZooKeeperServer)
```

2. Kafka Server Consuming the message hosted on port 9092



```
C:\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic mbta_msgs
{"event": "update", "data": {"attributes": {"bearing": 38, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 1, "direction_id": 1
, "label": "0820", "latitude": 42.27218861, "longitude": -70.950601789, "occupancy_status": "MANY_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03
T09:19:59-04:00"}, "id": "y0820", "links": {"self": "/vehicles/y0820"}, "relationships": {"route": {"data": {"id": "216", "type": "route"}}, "stop": {"data"
: {"id": "3265", "type": "stop"}}, "trip": {"data": {"id": "58751363", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 0, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 47, "direction_id": 1
, "label": "1440", "latitude": 42.41845897, "longitude": -71.00241169, "occupancy_status": "FEW_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03T0
9:19:59-04:00"}, "id": "y1440", "links": {"self": "/vehicles/y1440"}, "relationships": {"route": {"data": {"id": "450", "type": "route"}}, "stop": {"data":
{"id": "4500", "type": "stop"}}, "trip": {"data": {"id": "58653478", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 135, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": null, "direction_id
": 1, "label": "3265", "latitude": 42.26726, "longitude": -71.09306, "occupancy_status": null, "speed": 2.8, "updated_at": "2023-10-03T09:20:00-04:00"}, "id
": "G-10052", "links": {"self": "/vehicles/G-10052"}, "relationships": {"route": {"data": {"id": "Mattapan", "type": "route"}}, "stop": {"data": null}, "tri
p": {"data": {"id": "58411170", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 355, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 8, "direction_id":
1, "label": "1985", "latitude": 42.40399548, "longitude": -71.23406971, "occupancy_status": "MANY_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03
T09:20:01-04:00"}, "id": "y1985", "links": {"self": "/vehicles/y1985"}, "relationships": {"route": {"data": {"id": "61", "type": "route"}}, "stop": {"data":
 {"id": "7774", "type": "stop"}}, "trip": {"data": {"id": "58556678", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 0, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 25, "direction_id": 0
, "label": "2117", "latitude": 42.42484435, "longitude": -71.06777083, "occupancy_status": "MANY_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03T
09:19:59-04:00"}, "id": "y2117", "links": {"self": "/vehicles/y2117"}, "relationships": {"route": {"data": {"id": "104", "type": "route"}}, "stop": {"data":
 {"id": "5366", "type": "stop"}}, "trip": {"data": {"id": "58472275", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 304, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 9, "direction_id":
0, "label": "2092", "latitude": 42.38196656, "longitude": -71.10183444, "occupancy_status": "MANY_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03
T09:20:01-04:00"}, "id": "y2092", "links": {"self": "/vehicles/y2092"}, "relationships": {"route": {"data": {"id": "87", "type": "route"}}, "stop": {"data":
 {"id": "12616", "type": "stop"}}, "trip": {"data": {"id": "58556169", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 103, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 30, "direction_id":
 1, "label": "1793", "latitude": 42.35102003, "longitude": -71.11594049, "occupancy_status": "FULL", "speed": null, "updated_at": "2023-10-03T09:20:00-04:00
"}, "id": "y1793", "links": {"self": "/vehicles/y1793"}, "relationships": {"route": {"data": {"id": "57", "type": "route"}}, "stop": {"data": {"id": "937",
"type": "stop"}}, "trip": {"data": {"id": "58404898", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 139, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 1, "direction_id":
0, "label": "0651", "latitude": 42.38764158, "longitude": -71.07950912, "occupancy_status": null, "speed": null, "updated_at": "2023-10-03T09:20:00-04:00"},
 "id": "y0651", "links": {"self": "/vehicles/y0651"}, "relationships": {"route": {"data": {"id": "101", "type": "route"}}, "stop": {"data": {"id": "29012",
"type": "stop"}}, "trip": {"data": {"id": "58474736", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 103, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 1, "direction_id":
0, "label": "3132", "latitude": 42.40181837, "longitude": -71.04234563, "occupancy_status": "MANY_SEATS_AVAILABLE", "speed": null, "updated_at": "2023-10-03
T09:20:00-04:00"}, "id": "y3132", "links": {"self": "/vehicles/y3132"}, "relationships": {"route": {"data": {"id": "112", "type": "route"}}, "stop": {"data"
: {"id": "5666", "type": "stop"}}, "trip": {"data": {"id": "58471413", "type": "trip"}}}, "type": "vehicle"}}
{"event": "update", "data": {"attributes": {"bearing": 319, "carriages": [], "current_status": "IN_TRANSIT_TO", "current_stop_sequence": 42, "direction_id":
```

2. **Data Processing:**

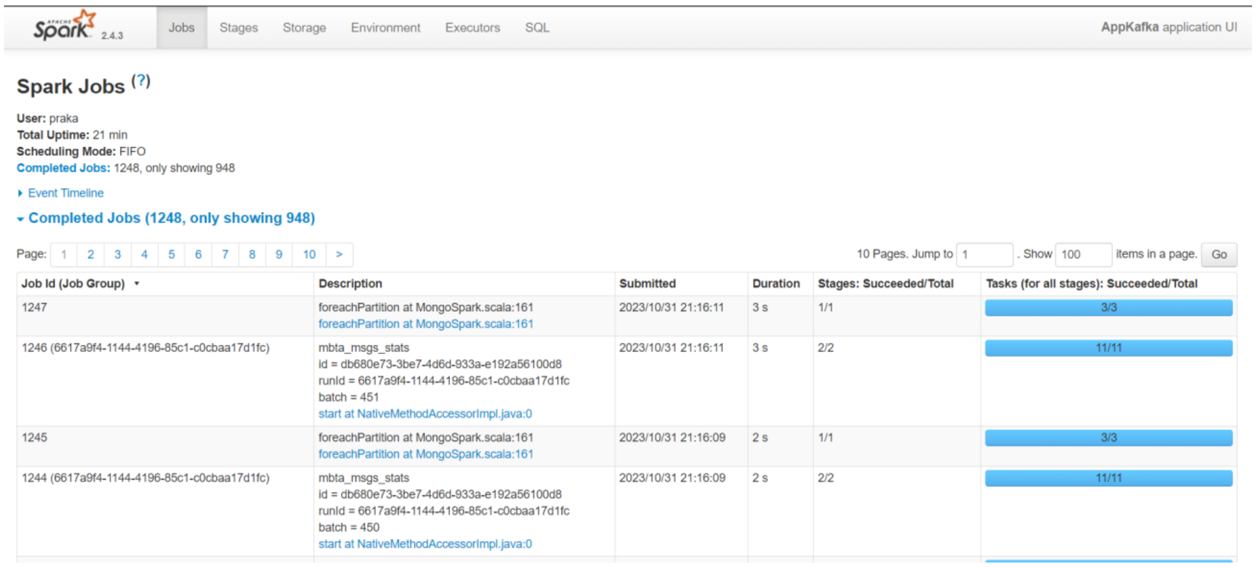
For data processing we used Apache Spark structured streaming. Spark is a data processing framework that can quickly perform processing tasks on very large data sets and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two characteristics are essential to the fields of big data and machine learning, which both call for massive computing resources to be mobilized to process enormous data stores. Also, Apache Spark Structured Streaming is a near real-time processing engine that uses the well-known Spark API to provide end-to-end fault tolerance with guaranteed exactly once processing. Structured Streaming allows you to express computations on streaming data the same way you express batch computations on static data. A structured streaming engine performs computations in stages, continuously updating results as streaming data arrives.

Components

1. **Driver/Master**: The master node (process) of the driver process coordinates workers and monitors tasks. Spark is split into jobs and scheduled to run on executors in the cluster. A Spark context (gateway) is created by the driver to monitor jobs running on a particular cluster and connect to the Spark cluster.
2. **Slave:** Worker node refers to node which runs the application code in the cluster. Worker Node is the Slave Node. Master node assign work and worker node actually perform the assigned tasks. Worker node processes the data stored on the node, they report the resources to the master
3. **Cluster Manager**: Standalone Cluster Manager of Apache Spark provides an effortless method of executing applications on a cluster. It contains one master and several workers, each having a configured size of memory and CPU cores. When one applies, they can decide beforehand what amount of memory the executors will use, and the total number of cores for all executors
4. **Executors**: An executor is a distributed agent responsible for the execution of tasks. Every spark application has its own executor process. Executors usually run for the entire lifetime of a Spark application and this phenomenon is known as “Static Allocation of Executors”. However, users can also opt for dynamic allocations of executors wherein they can add or remove spark executors dynamically to match with the overall workload.

Spark Distributed Processing Summary

1. **Spark Master/Driver node running jobs on port 4041 with executor assigned**



Spark Jobs (?)

User: praka
Total Uptime: 21 min
Scheduling Mode: FIFO
Completed Jobs: 1248, only showing 948

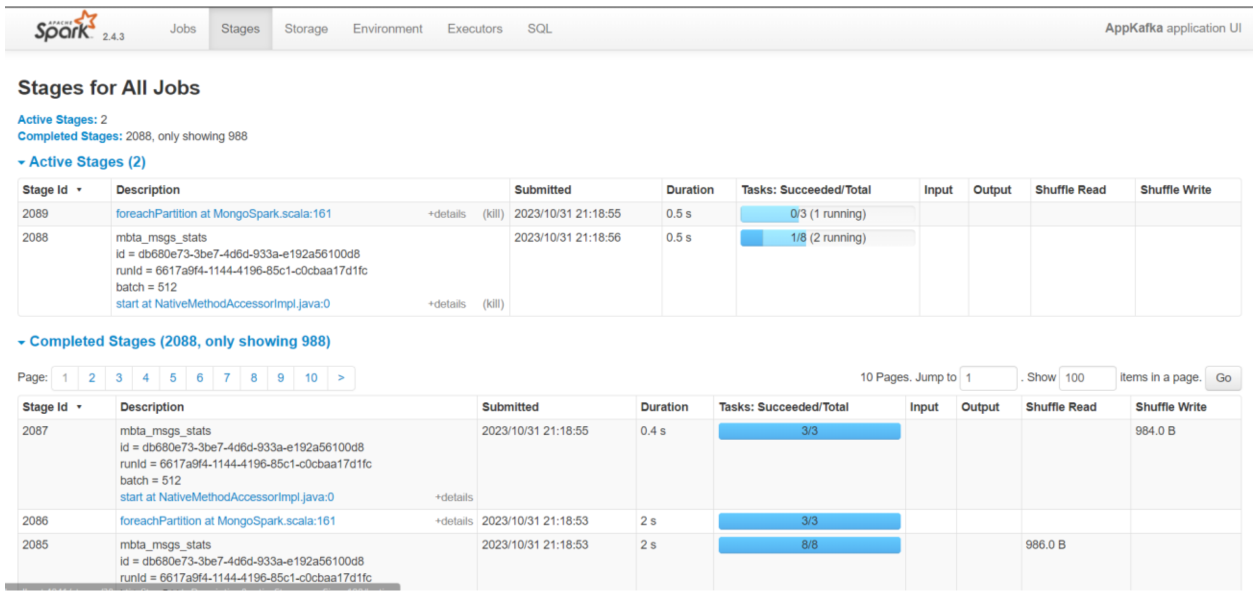
▸ Event Timeline

▾ Completed Jobs (1248, only showing 948)

Page: 1 2 3 4 5 6 7 8 9 10 > 10 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id (Job Group) ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
1247	foreachPartition at MongoSpark.scala:161 foreachPartition at MongoSpark.scala:161	2023/10/31 21:16:11	3 s	1/1	3/3
1246 (6617a9f4-1144-4196-85c1-c0cbaa17d1fc)	mbta_msgs_stats id = db680e73-3be7-4d6d-933a-e192a56100d8 runId = 6617a9f4-1144-4196-85c1-c0cbaa17d1fc batch = 451 start at NativeMethodAccessorImpl.java:0	2023/10/31 21:16:11	3 s	2/2	11/11
1245	foreachPartition at MongoSpark.scala:161 foreachPartition at MongoSpark.scala:161	2023/10/31 21:16:09	2 s	1/1	3/3
1244 (6617a9f4-1144-4196-85c1-c0cbaa17d1fc)	mbta_msgs_stats id = db680e73-3be7-4d6d-933a-e192a56100d8 runId = 6617a9f4-1144-4196-85c1-c0cbaa17d1fc batch = 450 start at NativeMethodAccessorImpl.java:0	2023/10/31 21:16:09	2 s	2/2	11/11

2. **Distributed Processing of Jobs Partitioning in Multiple Stages**



Stages for All Jobs

Active Stages: 2
Completed Stages: 2088, only showing 988

▾ Active Stages (2)

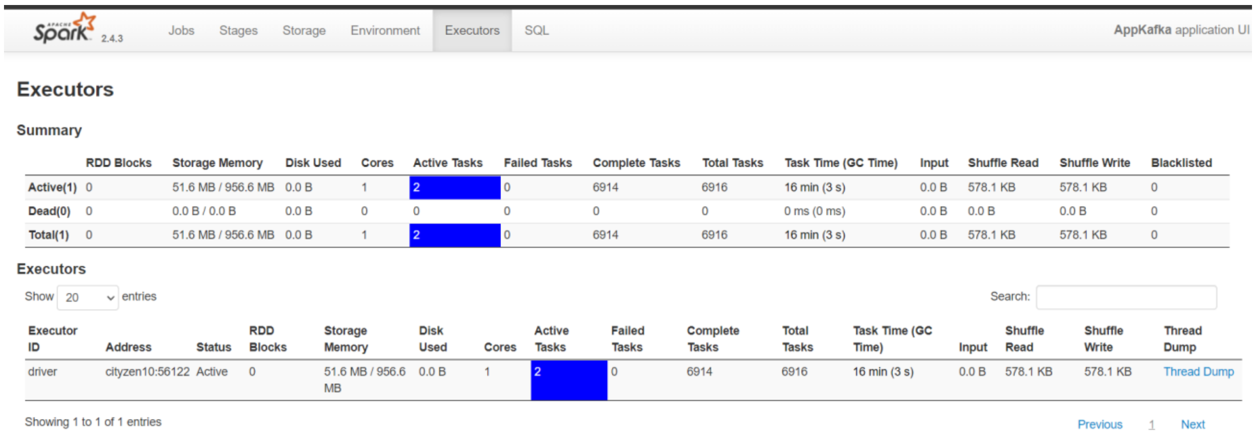
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2089	foreachPartition at MongoSpark.scala:161 +details (kill)	2023/10/31 21:18:55	0.5 s	0/3 (1 running)				
2088	mbta_msgs_stats id = db680e73-3be7-4d6d-933a-e192a56100d8 runId = 6617a9f4-1144-4196-85c1-c0cbaa17d1fc batch = 512 start at NativeMethodAccessorImpl.java:0 +details (kill)	2023/10/31 21:18:56	0.5 s	1/8 (2 running)				

▾ Completed Stages (2088, only showing 988)

Page: 1 2 3 4 5 6 7 8 9 10 > 10 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2087	mbta_msgs_stats id = db680e73-3be7-4d6d-933a-e192a56100d8 runId = 6617a9f4-1144-4196-85c1-c0cbaa17d1fc batch = 512 start at NativeMethodAccessorImpl.java:0 +details	2023/10/31 21:18:55	0.4 s	3/3				984.0 B
2086	foreachPartition at MongoSpark.scala:161 +details	2023/10/31 21:18:53	2 s	3/3				
2085	mbta_msgs_stats id = db680e73-3be7-4d6d-933a-e192a56100d8 runId = 6617a9f4-1144-4196-85c1-c0cbaa17d1fc	2023/10/31 21:18:53	2 s	8/8			986.0 B	

3. Executors' summary with tasks completed plus running



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(1)	0	51.6 MB / 956.6 MB	0.0 B	1	2	0	6914	6916	16 min (3 s)	0.0 B	578.1 KB	578.1 KB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	51.6 MB / 956.6 MB	0.0 B	1	2	0	6914	6916	16 min (3 s)	0.0 B	578.1 KB	578.1 KB	0

Executors

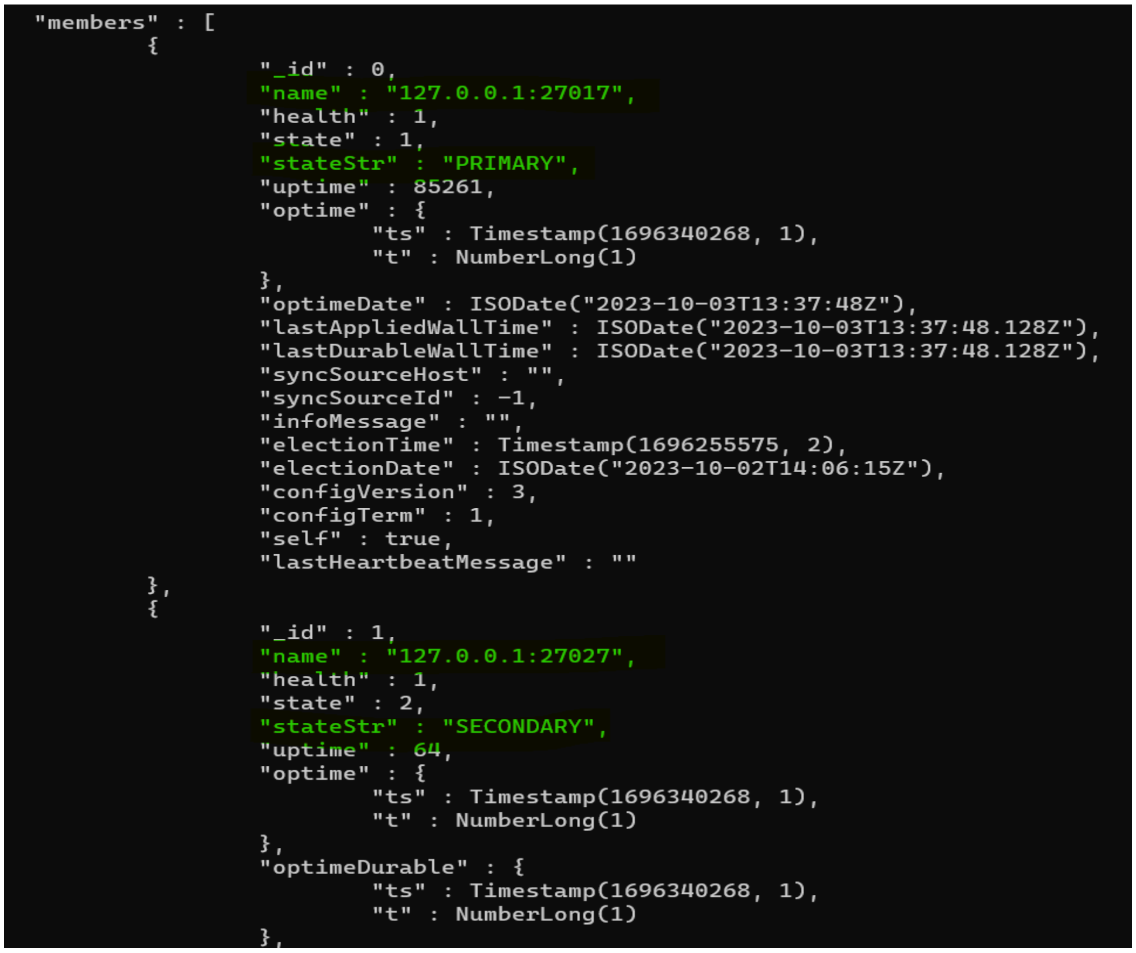
Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	cityzen10:56122	Active	0	51.6 MB / 956.6 MB	0.0 B	1	2	0	6914	6916	16 min (3 s)	0.0 B	578.1 KB	578.1 KB	Thread Dump

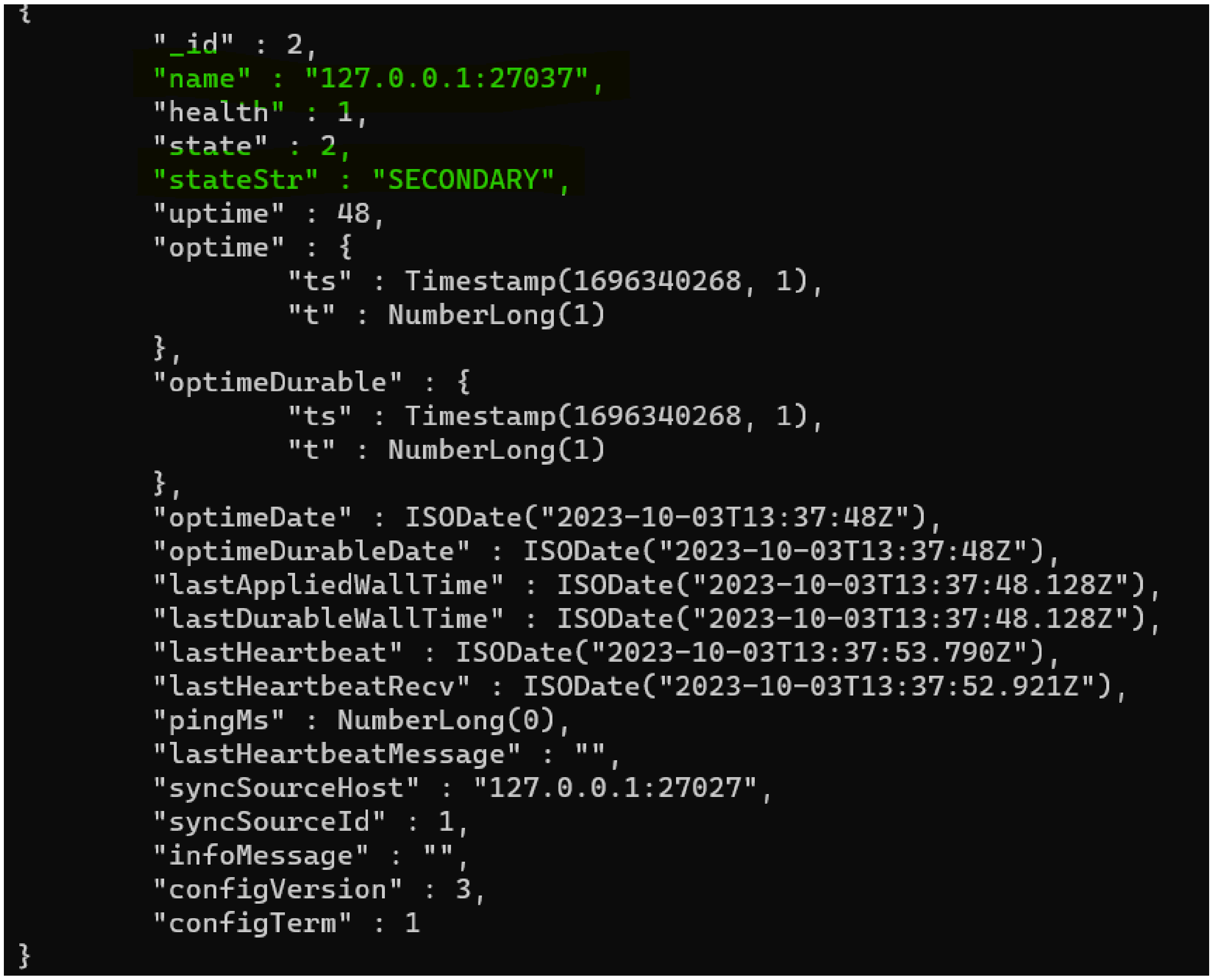
Showing 1 to 1 of 1 entries Previous 1 Next

3. **Storage (Load the data):** We used MongoDB for our data storage. MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need. We have implemented replication concept to provide fault tolerance for our application with one primary server and two secondary servers. If any server goes down, then secondary servers can act as primary server.

Setup of primary server on port 27017 and two secondary server on ports 27027 and 27037

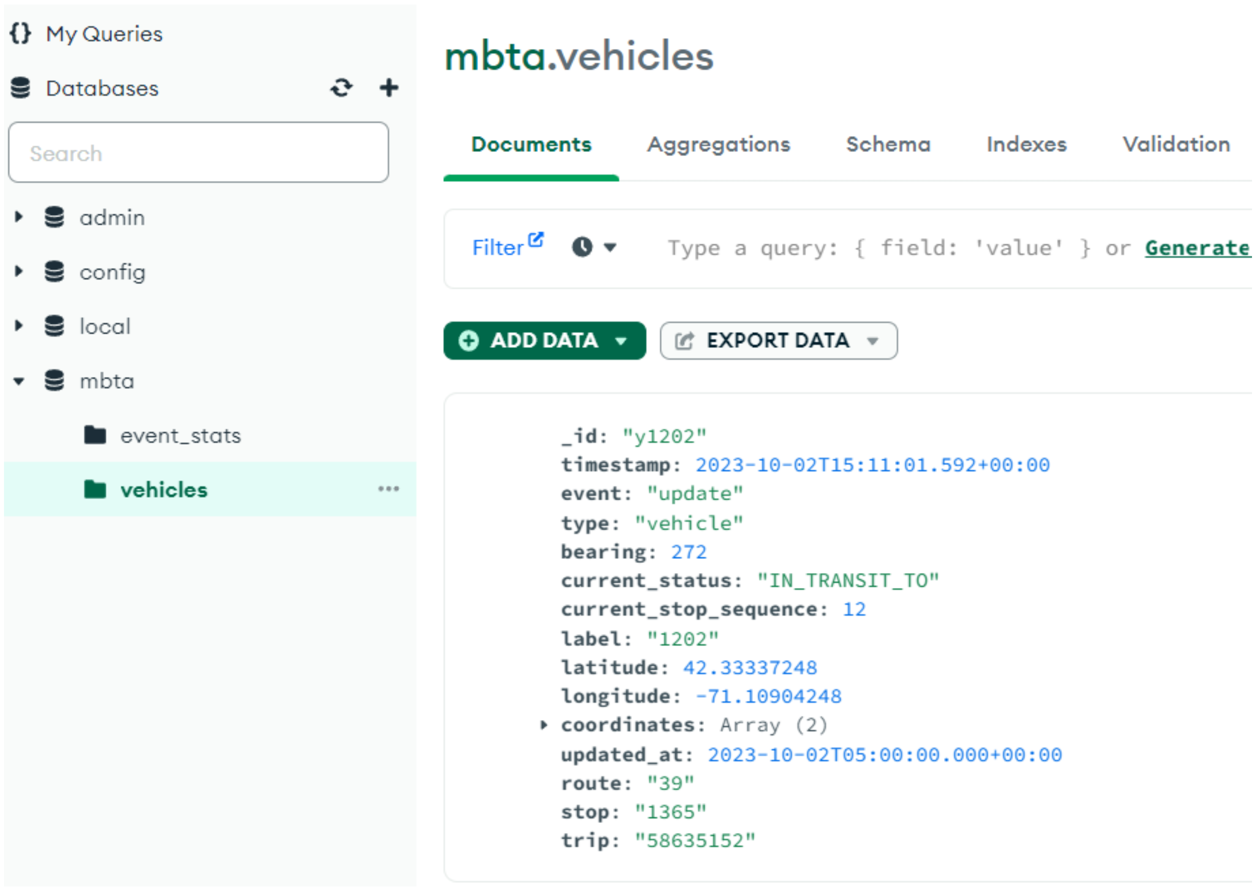


```
    "members" : [
            {
                    "_id" : 0,
                    "name" : "127.0.0.1:27017",
                    "health" : 1,
                    "state" : 1,
                    "stateStr" : "PRIMARY",
                    "uptime" : 85261,
                    "optime" : {
                            "ts" : Timestamp(1696340268, 1),
                            "t" : NumberLong(1)
                    },
                    "optimeDate" : ISODate("2023-10-03T13:37:48Z"),
                    "lastAppliedWallTime" : ISODate("2023-10-03T13:37:48.128Z"),
                    "lastDurableWallTime" : ISODate("2023-10-03T13:37:48.128Z"),
                    "syncSourceHost" : "",
                    "syncSourceId" : -1,
                    "infoMessage" : "",
                    "electionTime" : Timestamp(1696255575, 2),
                    "electionDate" : ISODate("2023-10-02T14:06:15Z"),
                    "configVersion" : 3,
                    "configTerm" : 1,
                    "self" : true,
                    "lastHeartbeatMessage" : ""
            },
            {
                    "_id" : 1,
                    "name" : "127.0.0.1:27027",
                    "health" : 1,
                    "state" : 2,
                    "stateStr" : "SECONDARY",
                    "uptime" : 64,
                    "optime" : {
                            "ts" : Timestamp(1696340268, 1),
                            "t" : NumberLong(1)
                    },
                    "optimeDurable" : {
                            "ts" : Timestamp(1696340268, 1),
                            "t" : NumberLong(1)
                    },
```

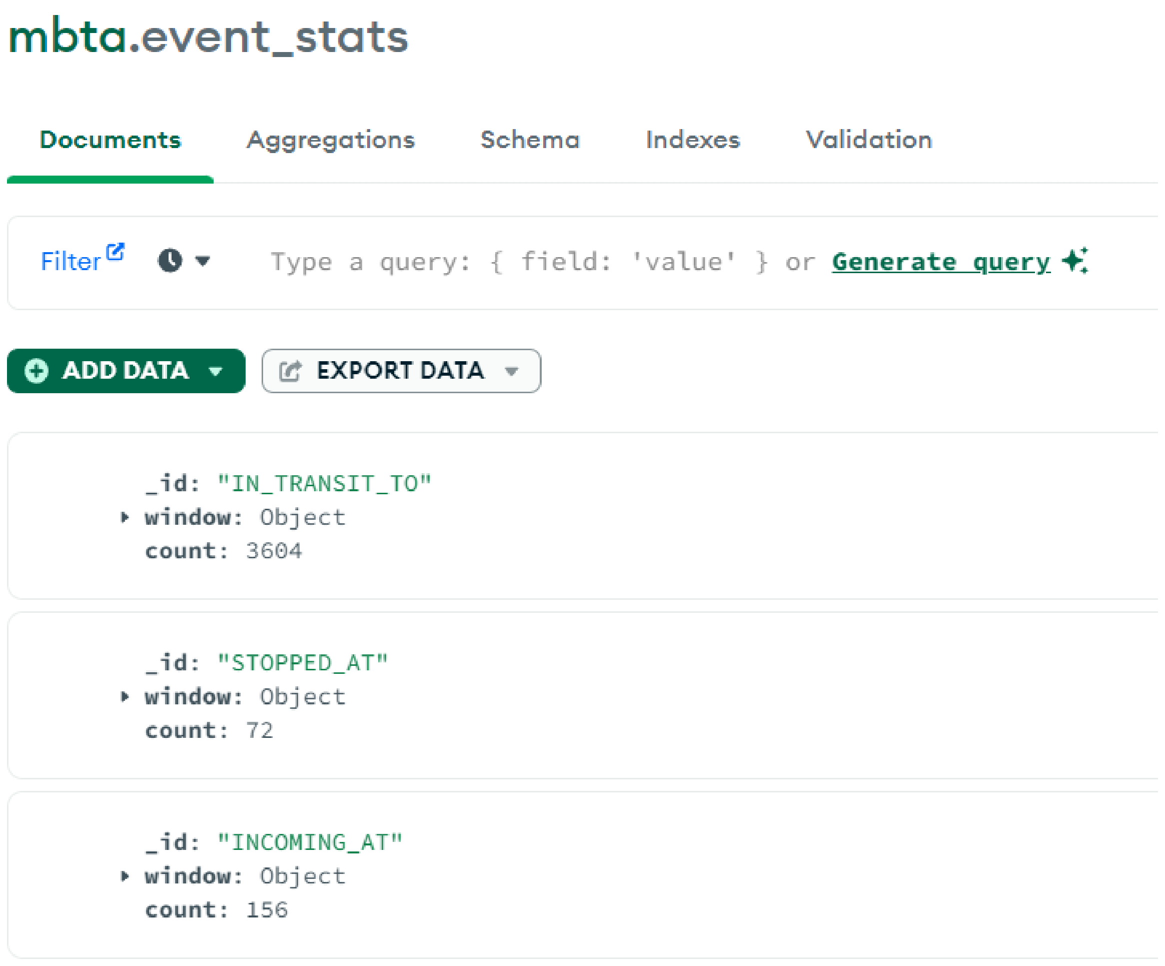


```
{
        "_id" : 2,
        "name" : "127.0.0.1:27037",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 48,
        "optime" : {
                "ts" : Timestamp(1696340268, 1),
                "t" : NumberLong(1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1696340268, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2023-10-03T13:37:48Z"),
        "optimeDurableDate" : ISODate("2023-10-03T13:37:48Z"),
        "lastAppliedWallTime" : ISODate("2023-10-03T13:37:48.128Z"),
        "lastDurableWallTime" : ISODate("2023-10-03T13:37:48.128Z"),
        "lastHeartbeat" : ISODate("2023-10-03T13:37:53.790Z"),
        "lastHeartbeatRecv" : ISODate("2023-10-03T13:37:52.921Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncSourceHost" : "127.0.0.1:27027",
        "syncSourceId" : 1,
        "infoMessage" : "",
        "configVersion" : 3,
        "configTerm" : 1
}
```

Two collections vehicles and event_stats to store the processed and transformed data



```
_id: "y1202"
timestamp: 2023-10-02T15:11:01.592+00:00
event: "update"
type: "vehicle"
bearing: 272
current_status: "IN_TRANSIT_TO"
current_stop_sequence: 12
label: "1202"
latitude: 42.33337248
longitude: -71.10904248
coordinates: Array (2)
updated_at: 2023-10-02T05:00:00.000+00:00
route: "39"
stop: "1365"
trip: "58635152"
```



```
_id: "IN_TRANSIT_TO"
window: Object
count: 3604

_id: "STOPPED_AT"
window: Object
count: 72

_id: "INCOMING_AT"
window: Object
count: 156
```

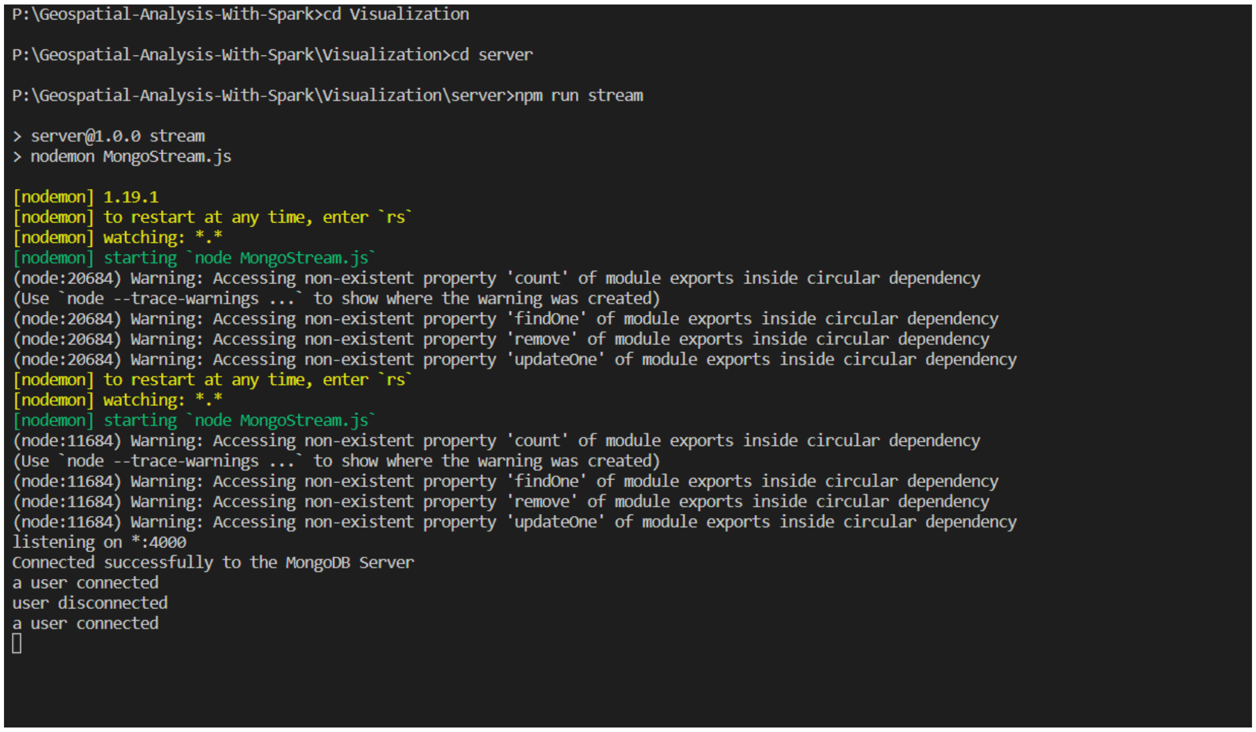
Deployment

This deployment of this application is divided basically in front-end and back-end development

1. **Server Side/Back-End Development:**

As a server environment, Node.js Framework has been used, which is used to build a real-time web application that can run over distributed devices, and Node with WebSocket fits perfectly for tracking websites. In our application, the server and client run on different platforms i.e, the client runs on the browser whereas the server runs on the Node Environment, and the data is real-time. So, there should be real-time communication between the client and server. To achieve this property the socket.io library has been used. In the Server environment, the Express.js framework has been used as a backend to connect the MongoDB database with the socket.io library. The connection between the MongoDB and the server is made with the help of an instance of socket.io. After connecting with MongoDB, data from both collections are fetched from the MongoDB database. One collection with the vehicle details and another with the updated status of real-time data. With the help of stats_event collection, we can get the update on any change in the database data, and this is done with the special watch function that is called when any change stream occurs, and the pipeline provides a matching structure of the document.

NodeJS server listening to MongoDB change stream

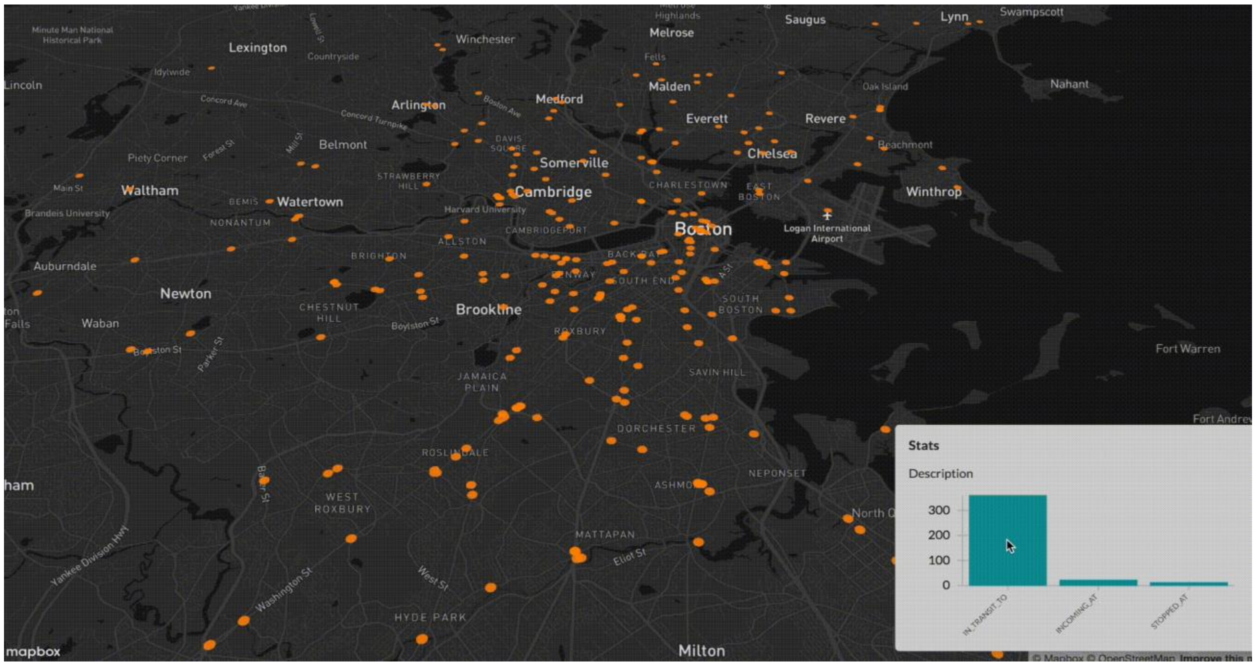


```
P:\Geospatial-Analysis-With-Spark>cd Visualization

P:\Geospatial-Analysis-With-Spark\Visualization>cd server

P:\Geospatial-Analysis-With-Spark\Visualization\server>npm run stream

> server@1.0.0 stream
> nodemon MongoStream.js

[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node MongoStream.js`
(node:20684) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:20684) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:20684) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:20684) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node MongoStream.js`
(node:11684) Warning: Accessing non-existent property 'count' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:11684) Warning: Accessing non-existent property 'findOne' of module exports inside circular dependency
(node:11684) Warning: Accessing non-existent property 'remove' of module exports inside circular dependency
(node:11684) Warning: Accessing non-existent property 'updateOne' of module exports inside circular dependency
listening on *:4000
Connected successfully to the MongoDB Server
a user connected
user disconnected
a user connected
```

2. Client Side/Front-End development:

Here, in the front-end part, we have used React javascript library since it provides an interactive UI interface. As we have used the HTTP socket.io interface on the server side, we have used the socket interface in the front end to connect with the server with HTTP. Basically, we have used two components in the front end to show the data information. One of them is StaticMap. For a better view and perfect efficiency, we have used the StaticMap component to populate the vehicle. Another one is a javascript chart component which is highly customizable. Basically, this chart provides information about Incoming vehicles, Transit information, and Stopped information. After connecting with the server using the HTTP endpoint, the data for both StaticMap and Bar Chart is loaded and rendered by ReactDOM. Since React uses a VDOM abstraction property, each time the data stream changes, it re-renders and shows the newly changed data.

Results



Conclusion & Future Enhancements

We created a complete pipeline to handle the real time data using modern distributed approaches that can help business take decision quickly and efficiently. We have achieved remarkable result that can benefit overall society by contributing to smart city. As part of course we have implemented distributed concepts like scalability, fault tolerance, distributed processing & streaming, resource management in real world environments. Important future work involves regularly track emergency vehicles like ambulances and fire trucks to assist them to choose the best route to travel by analyzing traffic for emergency vehicles. Also, it can also be extended to retrieve vehicle information like vehicle type, vehicle number, and speed of the vehicle which can be help for police department to reduce crimes. It can also alert public users using public transport about vehicles occupancy, wheelchair accessibility, expected arrival time and service disruptions. We can collect similar data of vehicle and extend this project to other public transportation across the world.

References

1. https://www.semanticscholar.org/paper/Real-time-Vehicle-tracking-system-based-on-ARM7-GPS-Mistary-Chile/57641297ec8370e4f59bb755c80130309d1ff53a
2. http://www.tafpublications.com/gip_content/paper/jater-3.4.1.pdf
3. https://ieeexplore.ieee.org/document/9446179
4. https://api-v3.mbta.com/docs/swagger/index.html#/LiveFacility
5. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
6. https://docs.mapbox.com/api/maps/static-images/
7. https://medium.com/javarevisited/how-to-learn-mern-stack-from-scratch-1784618eaffa
8. https://kafka.apache.org/documentation/#consumer_monitoring
9. https://kafka-python.readthedocs.io/en/stable/apidoc/KafkaProducer.html
10. https://kafka.apache.org/documentation/#gettingStarted
11. https://kafka-python.readthedocs.io/en/stable/apidoc/KafkaConsumer.html
12. https://www.youtube.com/watch?v=ExcRbA7fy_A