**Real-Time Analytics on Connected Car IoT Data Streams from Apache Kafka**

### Summary:

In this IoT example, we examine how to enable complex analytic queries on real-time Kafka streams from connected car sensors.

### IoT and Connected Cars:

With an increasing number of data-generating sensors being embedded in all manner of smart devices and objects, there is a clear, growing need to harness and analyze IoT data. Embodying this trend is the burgeoning field of connected cars, where suitably equipped vehicles are able to communicate traffic and operating information, such as speed, location, vehicle diagnostics, and driving behavior, to cloud-based repositories.

### Real-Time Analytics on Connected Car IoT Data

For our example, we have a fleet of connected vehicles that send the sensor data they generate to a Kafka cluster. We will show how this data in Kafka can be operationalized with the use of highly concurrent, low-latency queries on the real-time streams.

The ability to act on sensor readings in real time is useful for a large number of vehicular and traffic applications. Uses include detecting patterns and anomalies in driving behavior, understanding traffic conditions, routing vehicles optimally, and recognizing opportunities for preventive maintenance.

### How the Kafka IoT Example Works

The real-time connected car data will be simulated using a data producer application. Multiple instances of this data producer emit generated sensor metric events into a locally running Kafka instance. This particular Kafka topic is syncing continuously with a collection in Rockset via the [Rockset Kafka Sink connector](https://github.com/rockset/kafka-connect-rockset" \t "_blank). Once the setup is done, we will extract useful insights from this data using SQL queries and visualize them in [Redash](https://redash.io/" \t "_blank).



There are multiple components involved:

1. Apache Kafka
2. Apache Zookeeper
3. Data Producer - Connected vehicles generate IoT messages which are captured by a message broker and sent to the streaming application for processing. In our sample application, the IoT Data Producer is a simulator application for connected vehicles and uses Apache Kafka to store IoT data events.
4. Rockset - We use a real-time database to store data from Kafka and act as an analytics backend to serve fast queries and live dashboards.
5. Rockset Kafka Sink connector
6. Redash - We use Redash to power the IoT live dashboard. Each of the queries we perform on the IoT data is visualized in our dashboard.
7. Query Generator - This is a script for load testing Rockset with the queries of interest.

The code we used for the Data Producer and Query Generator can be found [here](https://github.com/rockset/recipes/tree/master/rockset-connected-cars).

### Kafka and Zookeeper

Kafka uses Zookeeper for service discovery and other housekeeping, and hence Kafka ships with a Zookeeper setup and other helper scripts. After downloading and extracting the Kafka tar, you just need to run the following command to set up the Zookeeper and Kafka server. This assumes that your current working directory is where you extracted the Kafka code.

Zookeeper:

./kafka\_2.11-2.3.0/bin/zookeeper-server-start.sh ../config/zookeeper.properties

Kafka server:

./kafka\_2.11-2.3.0/bin/kafka-server-start.sh ../config/server.properties

For our example, the default configuration should suffice. Make sure ports 9092 and 2181 are unblocked.

### Data Producer

This data producer is a Maven project, which will emit sensor metric events to our local Kafka instance. We simulate data from 1,000 vehicles and hundreds of sensor records per second. The code can be found here. Maven is required to build and run this.

After cloning the code, take a look at iot-kafka-producer/src/main/resources/iot-kafka.properties. Here, you can provide your Kafka and Zookeeper ports (which should be untouched when going with the defaults) and the topic name to which the event messages would be sent. Now, go into the rockset-connected-cars/iot-kafka-producer directory and run the following commands:

**mvn** compile && **mvn** exec:java -Dexec.mainClass="com.iot.app.kafka.producer.IoTDataProducer"

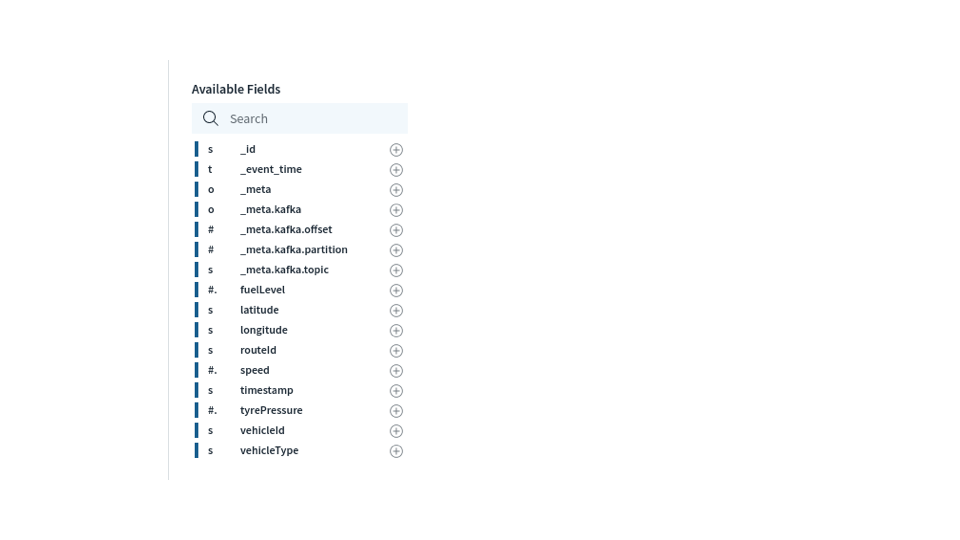
You should see a large number of these events continuously dumped into the Kafka topic name given in the configuration previously.

### Rockset and Rockset Kafka Connector

We would need the Rockset Kafka Sink connector to load these messages from our Kafka topic to a Rockset collection. To get the connector working, we first set up a Kafka integration from [the Rockset console](https://console.rockset.com/integrations). Then, we create a collection using the new Kafka integration. Run the following command to connect your Kafka topic to the Rockset collection.

./kafka\_2.11-2.3.0/bin/connect-standalone.sh ./connect-standalone.properties ./connect-rockset-sink.properties

### Querying the IoT Data

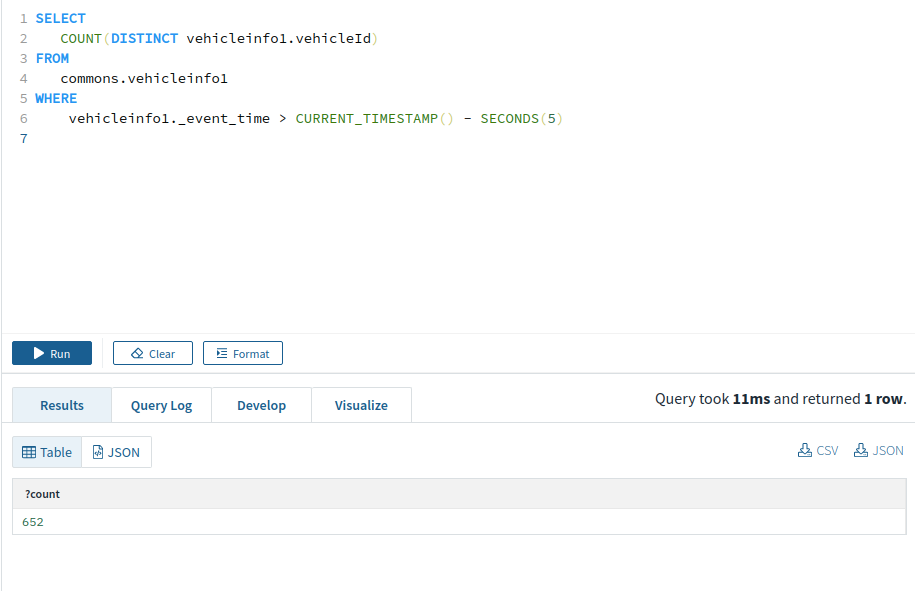


The above shows all the fields available in the collection which is used in the following queries. Note that we did not have to predefine a schema or perform any data preparation to get data in Kafka to be queryable in Rockset.

As our Rockset collection is getting data, we can query using SQL to get some useful insights.

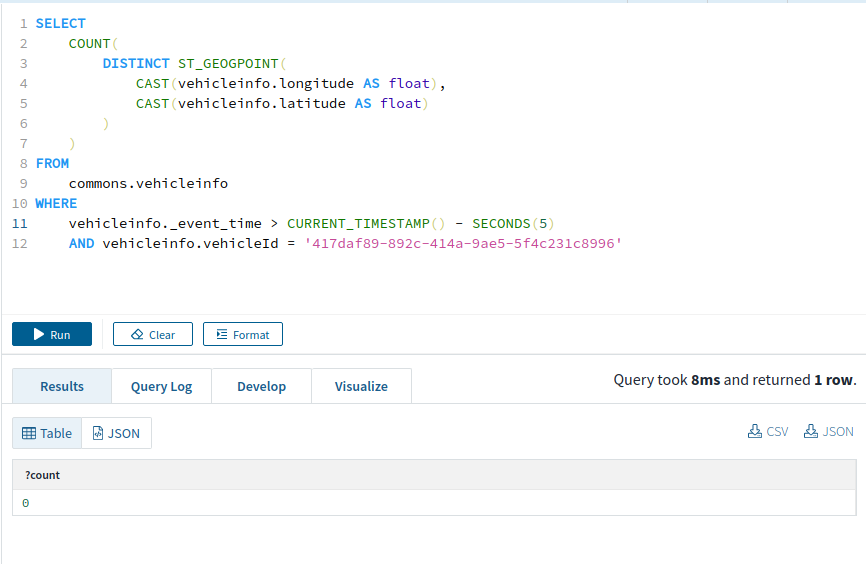
**Count of vehicles that produced a sensor metric in the last 5 seconds**

This helps up know which vehicles are actively emitting data.



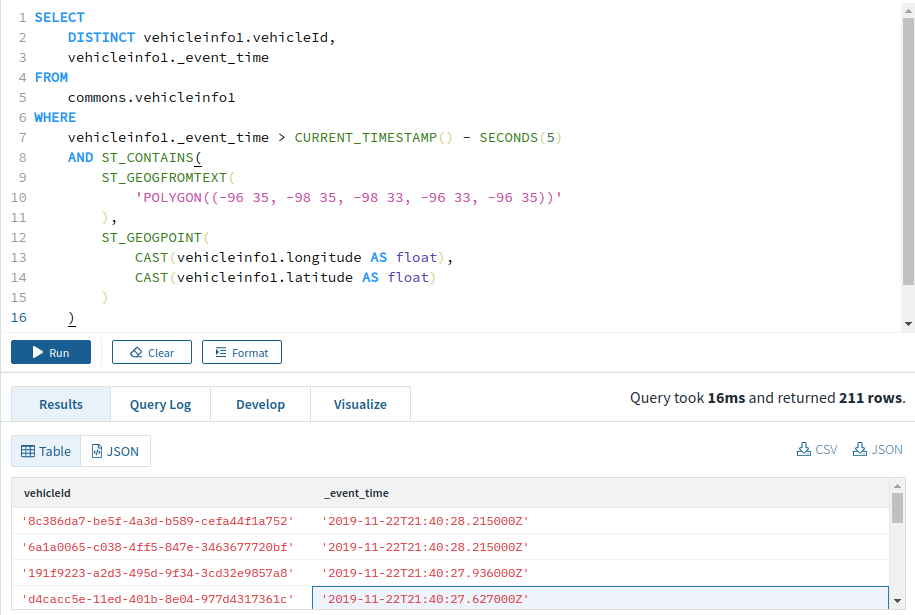
**Check if a vehicle is moving in last 5 seconds**

It can be useful to know if a vehicle is actually moving or is stuck in traffic.



**Vehicles that are within a specified Point of Interest (POI) in the last 5 seconds**

This is a common type of query, especially for a ride-hailing application, to find out which drivers are available in the vicinity of a passenger. Rockset provides CURRENT\_TIMESTAMP and SECONDS functions to perform timestamp-related queries. It also has native support for location-based queries using the functions ST\_GEOPOINT, ST\_GEOGFROMTEXT and ST\_CONTAINS.



**Top 5 vehicles that have moved the maximum distance in the last 5 seconds**

This query shows us the most active vehicles.

*/\* Grouping events emitted in last 5 seconds by vehicleId and getting the time of the oldest event in this group \*/*

**WITH** vehicles\_in\_last\_5\_seconds **AS** (

**SELECT**

vehicleinfo.vehicleId,

vehicleinfo.\_event\_time,

vehicleinfo.latitude,

vehicleinfo.longitude

**from**

commons.vehicleinfo

**WHERE**

vehicleinfo.\_event\_time > **CURRENT\_TIMESTAMP**() - **SECONDS**(5)

),

older\_sample\_time\_for\_vehicles **as** (

**SELECT**

**MIN**(vehicles\_in\_last\_5\_seconds.\_event\_time) **as** min\_time,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

vehicles\_in\_last\_5\_seconds

**GROUP** **BY**

vehicles\_in\_last\_5\_seconds.vehicleId

),

older\_sample\_location\_for\_vehicles **AS** (

**SELECT**

vehicles\_in\_last\_5\_seconds.latitude,

vehicles\_in\_last\_5\_seconds.longitude,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

older\_sample\_time\_for\_vehicles,

vehicles\_in\_last\_5\_seconds

**where**

vehicles\_in\_last\_5\_seconds.\_event\_time = older\_sample\_time\_for\_vehicles.min\_time

**and** vehicles\_in\_last\_5\_seconds.vehicleId = older\_sample\_time\_for\_vehicles.vehicleId

),

latest\_sample\_time\_for\_vehicles **as** (

**SELECT**

**MAX**(vehicles\_in\_last\_5\_seconds.\_event\_time) **as** max\_time,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

vehicles\_in\_last\_5\_seconds

**GROUP** **BY**

vehicles\_in\_last\_5\_seconds.vehicleId

),

latest\_sample\_location\_for\_vehicles **AS** (

**SELECT**

vehicles\_in\_last\_5\_seconds.latitude,

vehicles\_in\_last\_5\_seconds.longitude,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

latest\_sample\_time\_for\_vehicles,

vehicles\_in\_last\_5\_seconds

**where**

vehicles\_in\_last\_5\_seconds.\_event\_time = latest\_sample\_time\_for\_vehicles.max\_time

**and** vehicles\_in\_last\_5\_seconds.vehicleId = latest\_sample\_time\_for\_vehicles.vehicleId

),

distance\_for\_vehicles **AS** (

**SELECT**

ST\_DISTANCE(

ST\_GEOGPOINT(

**CAST**(older\_sample\_location\_for\_vehicles.longitude **AS** float),

**CAST**(older\_sample\_location\_for\_vehicles.latitude **AS** float)

),

ST\_GEOGPOINT(

**CAST**(latest\_sample\_location\_for\_vehicles.longitude **AS** float),

**CAST**(latest\_sample\_location\_for\_vehicles.latitude **AS** float)

)

) **as** distance,

latest\_sample\_location\_for\_vehicles.vehicleId

**FROM**

latest\_sample\_location\_for\_vehicles,

older\_sample\_location\_for\_vehicles

**WHERE**

latest\_sample\_location\_for\_vehicles.vehicleId = older\_sample\_location\_for\_vehicles.vehicleId

)

**SELECT**

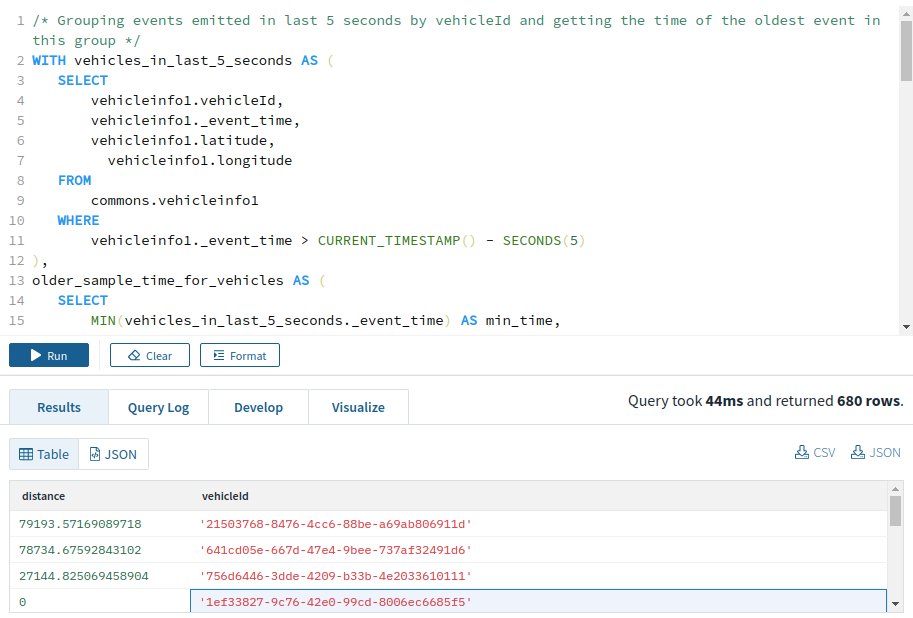
\*

**from**

distance\_for\_vehicles

**ORDER** **BY**

distance\_for\_vehicles.distance **DESC**



**Number of sudden braking events**

This query can be helpful in detecting slow-moving traffic, potential accidents, and more error-prone drivers.

*/\* Grouping events emitted in last 5 seconds by vehicleId and getting the time of the oldest event in this group \*/*

**WITH** vehicles\_in\_last\_5\_seconds **AS** (

**SELECT**

vehicleinfo.vehicleId,

vehicleinfo.\_event\_time,

vehicleinfo.speed

**from**

commons.vehicleinfo

**WHERE**

vehicleinfo.\_event\_time > **CURRENT\_TIMESTAMP**() - **SECONDS**(5)

),

older\_sample\_time\_for\_vehicles **as** (

**SELECT**

**MIN**(vehicles\_in\_last\_5\_seconds.\_event\_time) **as** min\_time,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

vehicles\_in\_last\_5\_seconds

**GROUP** **BY**

vehicles\_in\_last\_5\_seconds.vehicleId

),

older\_sample\_speed\_for\_vehicles **AS** (

**SELECT**

vehicles\_in\_last\_5\_seconds.speed,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

older\_sample\_time\_for\_vehicles,

vehicles\_in\_last\_5\_seconds

**where**

vehicles\_in\_last\_5\_seconds.\_event\_time = older\_sample\_time\_for\_vehicles.min\_time

**and** vehicles\_in\_last\_5\_seconds.vehicleId = older\_sample\_time\_for\_vehicles.vehicleId

),

latest\_sample\_time\_for\_vehicles **as** (

**SELECT**

**MAX**(vehicles\_in\_last\_5\_seconds.\_event\_time) **as** max\_time,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

vehicles\_in\_last\_5\_seconds

**GROUP** **BY**

vehicles\_in\_last\_5\_seconds.vehicleId

),

latest\_sample\_speed\_for\_vehicles **AS** (

**SELECT**

vehicles\_in\_last\_5\_seconds.speed,

vehicles\_in\_last\_5\_seconds.vehicleId

**FROM**

latest\_sample\_time\_for\_vehicles,

vehicles\_in\_last\_5\_seconds

**where**

vehicles\_in\_last\_5\_seconds.\_event\_time = latest\_sample\_time\_for\_vehicles.max\_time

**and** vehicles\_in\_last\_5\_seconds.vehicleId = latest\_sample\_time\_for\_vehicles.vehicleId

)

**SELECT**

latest\_sample\_speed\_for\_vehicles.speed,

older\_sample\_speed\_for\_vehicles.speed,

older\_sample\_speed\_for\_vehicles.vehicleId

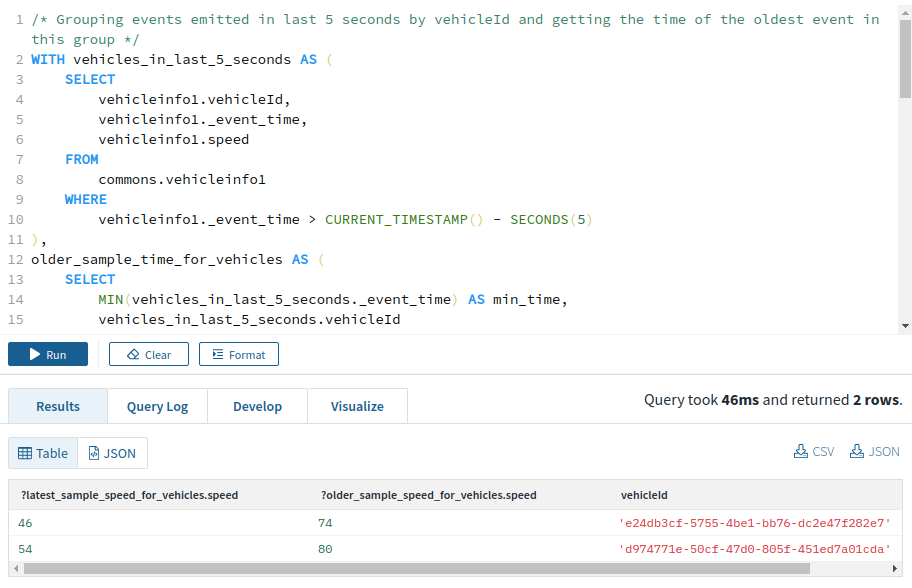
**from**

older\_sample\_speed\_for\_vehicles, latest\_sample\_speed\_for\_vehicles

**WHERE**

older\_sample\_speed\_for\_vehicles.vehicleId = latest\_sample\_speed\_for\_vehicles.vehicleId

**AND** latest\_sample\_speed\_for\_vehicles.speed < older\_sample\_speed\_for\_vehicles.speed - 20



**Number of rapid acceleration events**

This is similar to the query above, just with the speed difference condition changed from

**latest\_sample\_speed\_for\_vehicles**.speed < **older\_sample\_speed\_for\_vehicles**.speed **-** 20

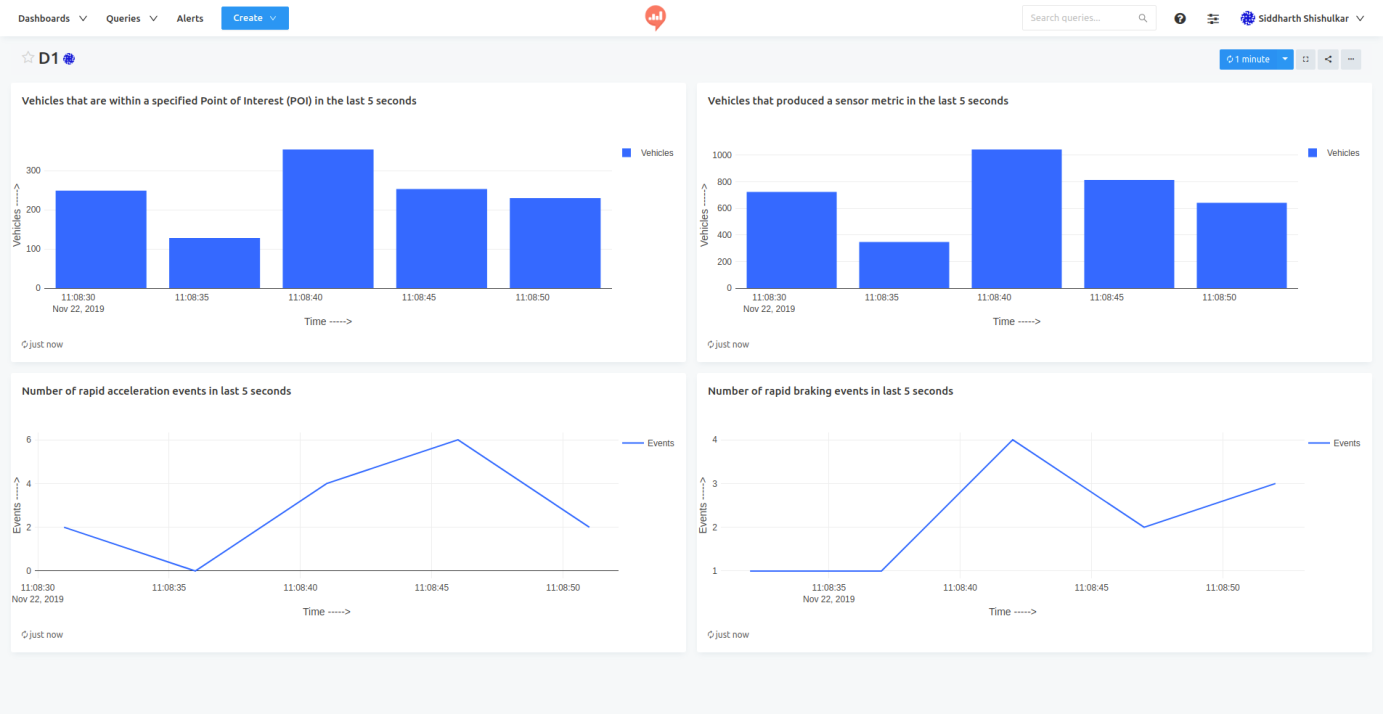
to

**latest\_sample\_speed\_for\_vehicles**.speed **-** 20 > **older\_sample\_speed\_for\_vehicles**.speed



### Live Dashboard with Redash

Redash offers a hosted solution which offers easy integration with Rockset. With a couple of clicks, you can create charts and dashboards, which auto-refresh as new data arrives. The following visualizations were created, based on the above queries.



### Supporting High Concurrency

Rockset is capable of handling a large number of complex queries on large datasets while maintaining query latencies in the hundreds of milliseconds. [This](https://github.com/rockset/recipes/tree/master/rockset-connected-cars/query-generator) provides a small python script for load testing Rockset. It can be configured to run any number of QPS (queries per second) with different queries for a given duration. It will run the specified number of queries for a given amount of time and generate a histogram showing the time generated by each query for different queries.

By default, it will run 4 different queries with queries q1, q2, q3, and q4 having 50%, 40%, 5%, and 5% bandwidth respectively.

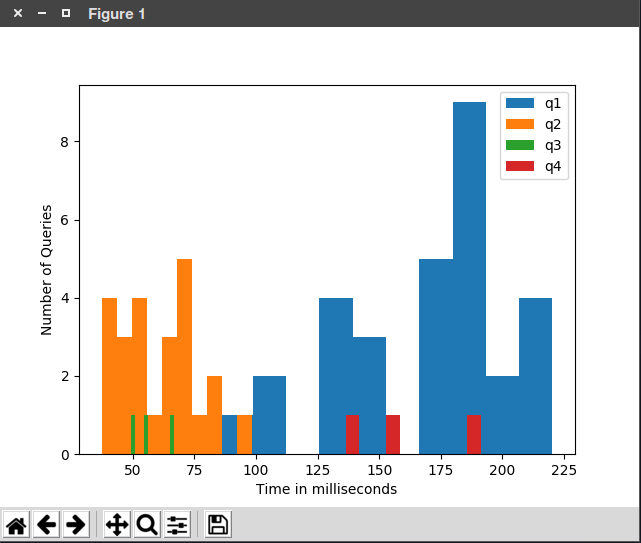
q1. Is a specified given vehicle stationary or in-motion in the last 5 seconds? (point lookup query within a window)

q2. List the vehicles that are within a specified Point of Interest (POI) in the last 5 seconds. (point lookup & short range scan within a window)

q3. List the top 5 vehicles that have moved the maximum distance in the last 5 seconds (global aggregation and topN)

q4. Get the unique count of all vehicles that produced a sensor metric in the last 5 seconds (global aggregation with count distinct)

Below is an example of a 10 second run.



### Real-Time Analytics Stack for IoT

IoT use cases typically involve large streams of sensor data, and Kafka is often used as a streaming platform in these situations. Once the IoT data is collected in Kafka, obtaining real-time insight from the data can prove valuable. In the context of connected car data, real-time analytics can benefit logistics companies in fleet management and routing, ride hailing services matching drivers and riders, and transportation agencies monitoring traffic conditions, just to name a few.

Through the course of this guide, we showed how such a connected car IoT scenario may work. Vehicles emit location and diagnostic data to a Kafka cluster, a reliable and scalable way to centralize this data. We then synced the data in Kafka to Rockset to enable fast, ad hoc queries and live dashboards on the incoming IoT data. Key considerations in this process were:

* Need for low data latency - to query the most recent data
* East of use - no schema needs to be configured
* High QPS - for live applications to query the IoT data
* Live dashboards - integration with tools for visual analytics

Read our other blog, [Where's My Tesla? Creating a Data API Using Kafka, Rockset and Postman to Find Out](https://rockset.com/blog/wheres-my-tesla-data-api-kafka-rockset-postman), to see how we expose real-time Kafka IoT data through the Rockset REST API.

Learn more about how a real-time analytics stack based on Kafka and Rockset works [here](https://rockset.com/sql-on-kafka/).