Building a dockerized functional system for streaming data and gathering coordinates for vehicles location, with Spark, Kafka and MySQL.

This tutorial will include the following parts:

i) Preparing the environment
ii) Simulating Data (coordinates) - Python
iii) Data ingestion - Scala
iv) Creating map and exposing the vehicle's location (SQL, PHP and Leaflet)
v) Finding the nearest restaurant, while knowing the position of the vehicle when the request was made - SQL and PHP

*For a real case scenario, **Percona** and MySQL would improve performance.*

1) Preparing the environment

The choice for this project was a dockerized environment, consisting in:

- 1 Spark and Yarn container - on which there have been installed Apache 2, PHP5, MySQL and Leaflet

- 1 Kafka container

- 1 Zookeeper container

Services

Service	Version
Spark and Yarn	1.6.0
Hadoop	2.6.0
Kafka	0.10.2.1
Zookeeper	3.3.6
MySQL	5.1.73

Creating containers:

The creation of containers was made by using Perl scripts, for a faster deployment:

b) Zookeeper container:

Check link:

https://github.com/LorenvXn/Build-machine-learning-environment-on-dockers-/tree/master/zookeeper

c) Kafka container:

Check link:

https://github.com/LorenvXn/Build-machine-learning-environment-on-dockers-/tree/master/kafka

d)Deploying Spark and Yarn

For this container, the image **sequenceiq/spark** has been used.

Connecting the containers

Once the containers have been created, they must communicate with each other

```
root@host:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS
NAMES
nauseous_kirch
c336c772c5ad        spark               "/etc/bootstrap.sh ba"   4 days ago
Up 27 hours         22/tcp, 4040/tcp, 8030-8033/tcp, 8040/tcp, 8042/tcp, 8088/tcp,
49707/tcp, 50010/tcp, 50020/tcp, 50070/tcp, 50075/tcp, 50090/tcp   spark
e3f53da753fd        kafka               "/bin/bash"              4 days ago
Up 25 hours
kafka
e86cc17e6ee5        zookeeper           "bash"                   4 days ago
Up 25 hours         2181/tcp, 2888/tcp, 3888/tcp
zookeeper
```

a) Create a new network bridge

```
 docker network create --driver=bridge spark-streaming
```

b) Adding each container to new bridge

```
docker network connect spark-streaming c336c772c5ad

docker network connect spark-streaming e3f53da753fd

docker network connect spark-streaming e86cc17e6ee5
```

Real case scenario*: static IP should be attributed to each container as per Dockers documentation*

Starting the services

1) Start Zookeeper service

```
root@host:~# docker exec -ti `docker ps | grep zookeeper | awk {'print $1'}`
/bin/bash
root@e86cc17e6ee5:/opt/zookeeper#
root@e86cc17e6ee5:/opt/zookeeper# cd bin/
root@e86cc17e6ee5:/opt/zookeeper/bin# ./zkServer.sh start
JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

2) Start Kafka service, and create topic CarOneStreaming

```
root@host:/opt# docker exec -ti `docker ps | grep kafka | awk {'print $1'}` /bin/bash
[root@e3f53da753fd /]# cd kafka
[root@e3f53da753fd kafka]#
[root@e3f53da753fd kafka]#bin/kafka-server-start.sh config/server.properties
[root@e3f53da753fd kafka]#
[root@e3f53da753fd kafka]#bin/kafka-topics.sh --create --zookeeper 172.21.0.3:2181
--replication-factor 1 --partitions 1 --topic CarOneStreaming
```

Now, let’s say that at every 5 seconds, the vehicle sends data on the coordinates, speed, etc.

This information will be sent to Kafka producer, and after the data ingestion takes place, through dataframes, it will be written into MySQL (this type of database has been selected for geolocation purposes)

Based on these specifications, our tablespace streams_table_car1 has the below characteristics:

```
bash-4.1# mysql -e "use streams_db; describe streams_table_car1" -uroot -p
Enter password:
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| lat      | float       | YES  |     | NULL    |       |
| lon      | float       | YES  |     | NULL    |       |
| speed    | float       | YES  |     | NULL    |       |
| location | varchar(30) | YES  |     | NULL    |       |
| date     | datetime    | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

ii) Simulating data

To obtain coordinates for a specific geographic area, module **geopy** has been used.

The names of certain addresses were provided to a python script, and as a result they were “translated” as coordinates (latitude and longitude)

Example python script

```
#!/usr/bin/python

from geopy import Nominatim

geolocator = Nominatim()

with open("/root/addresses.txt",'r') as fp:
       for line in fp:
              location = geolocator.geocode(line)
              print (location.latitude, location.longitude)
       fp.close()
```

For an address like “11 5th Avenue NYC”, the script would provide coordinates:
(40.7326282, -73.9958528)

iii) Data ingestion - Spark and Kafka, and stored data into MySQL

Scala code for Spark Streaming and Kafka Integration (Consumer). All data is stored in MySQL tablespace through dataframes.

```
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SQLContext
import org.apache.spark._
import _root_.kafka.serializer.StringDecoder

import org.apache.spark.SparkContext._
import org.apache.spark.streaming._

import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka._

import org.apache.spark.sql.types._
import org.apache.spark.sql.Row;

import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._

import java.util.Properties
import org.apache.spark.sql.SaveMode

import sys.process._

import sqlContext.implicits._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types._


import scala.concurrent.ExecutionContext.Implicits.global
import scala.concurrent.duration._
import scala.concurrent.{Await, Future}
import scala.language.postfixOps

import java.sql.DriverManager
import java.sql.Connection
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd._
import org.apache.spark.SparkContext._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._


import org.apache.spark.sql.types._
import org.apache.spark.sql.Row;

import org.apache.spark.sql.types._

import java.util.Properties



object ScalaStreaming {

case class Dates(lat: Double, lon: Double,
                 speed: Double,
                 location: String,
                 date: String)




  def main(args: Array[String]) {

             val config = new SparkConf()
             val sc = new SparkContext(config)
             val sqlContext = new SQLContext(sc)
             val ssc = new StreamingContext(sc, Seconds(5))

                //kafka set-up - IP 172.21.0.4 belongs to Kafka container

             val brokers = "172.21.0.4:9092"
             val topics = "CarOneStreaming"
             val topicsSet = topics.split(",").toSet

             val kafkaParams = Map[String, String]("metadata.broker.list" -> brokers)

val schema = StructType(Array(
             StructField("lat", DoubleType, true),
             StructField("lon", DoubleType, true),
             StructField("speed", DoubleType, true),
             StructField("location", StringType, true),
             StructField("date", StringType, true)   ))


val prop = new java.util.Properties()
```

```
prop.put("user", "root")
prop.put("password", "M0ns00n!!!")

val driver = "com.mysql.jdbc.Driver"
val url = "jdbc:mysql://localhost:3306/streams_db"



 val linesDStream = KafkaUtils.createDirectStream[String, String, StringDecoder,
StringDecoder](ssc, kafkaParams, topicsSet)


val SQLDStream = linesDStream.map(_._2).map(_.split(",")).map(p =>
Dates(p(0).toDouble,
      p(1).toDouble,
      p(2).toDouble,
      p(3).toString,
      p(4).toString))

SQLDStream.foreachRDD{ rdd =>
if (!rdd.isEmpty) {
               val count = rdd.count
               println("count received " + count)
               val sqlContext = SQLContext.getOrCreate(rdd.sparkContext)

               val df = rdd.toDF()

//extract mysql tablespace existent rows

  val d_test = sqlContext.read.format("jdbc").options(
        Map(
          "url" ->
"jdbc:mysql://localhost:3306/streams_db?user=root&password=M0ns00n!!!",
          "dbtable" -> "streams_table_car1",
          "driver" -> "com.mysql.jdbc.Driver"
        )).load()

//some cache would be nice

        d_test.show()

//write to MySQL

        df.write.mode(SaveMode.Append).jdbc(url,"streams_table_car1",prop)

   //  Thread.sleep(3000)
   }
}
```

Maven dependencies

```
<dependencies>
        <!-- Scala and Spark dependencies -->
        <dependency>
                <groupId>org.scala-lang</groupId>
                <artifactId>scala-library</artifactId>
                <version>${scala.version}</version>
        </dependency>
        <dependency>
                <groupId>org.apache.spark</groupId>
                <artifactId>spark-core_2.10</artifactId>
                <version>1.2.0-cdh5.3.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.kafka</groupId>
                <artifactId>kafka_2.10</artifactId>
                <version>0.8.2.1</version>
        </dependency>
        <dependency>
                <groupId>org.apache.spark</groupId>
                 <artifactId>spark-streaming-kafka_2.10</artifactId>
                <version>1.6.0</version>
        </dependency>
     <dependency>
             <groupId>org.apache.spark</groupId>
             <artifactId>spark-streaming_2.10</artifactId>
             <version>1.6.0</version>
     </dependency>
     <dependency>
            <groupId>com.datastax.spark</groupId>
            <artifactId>spark-cassandra-connector_2.11</artifactId>
            <version>1.6.1</version>
     </dependency>
     <dependency>
            <groupId>org.apache.spark</groupId>
     <artifactId>spark-sql_2.10</artifactId> <!-- matching Scala version -->
            <version>1.6.1</version>
     </dependency>
     <dependency>
             <groupId>org.scala-lang</groupId>
             <artifactId>scala-reflect</artifactId>
             <version>2.10.0-M4</version>
     </dependency>
     <dependency>
             <groupId>joda-time</groupId>
             <artifactId>joda-time</artifactId>
             <version>2.9.1</version>
     </dependency>
     <dependency>
            <groupId>org.codehaus.jsr166-mirror</groupId>
             <artifactId>jsr166</artifactId>
             <version>1.7.0</version>
     </dependency>
      <dependency>
              <groupId>mysql</groupId>
             <artifactId>mysql-connector-java</artifactId>
              <version>5.1.16</version>
       </dependency>
     <dependency>
         <groupId>org.apache.spark</groupId>
          <artifactId>spark-core_2.10</artifactId>
          <version>1.6.0</version>
      </dependency>

</dependencies>
```

Create the fat jar with mvn

```
mvn clean && mvn install && mvn package
```

...and submit it

```
cd target; spark-submit --class ScalaStreaming
spark-scala-maven-project-0.0.1-SNAPSHOT-jar-with-dependencies.jar
--packages org.apache.spark:spark-streaming-kafka_2.10:1.6.1
 --jars /home/mysql-connector-java-5.1.16.jar
```

Now we can send coordinates into Kafka producer:

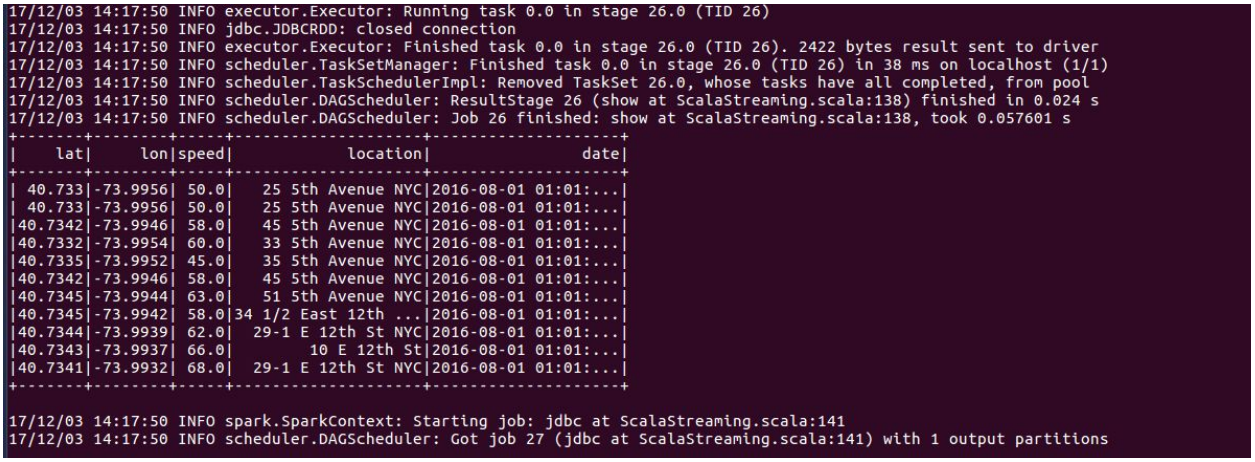
```
[root@e3f53da753fd kafka]# bin/kafka-console-producer.sh --broker-list
172.21.0.4:9092 --topic CarOneStreaming
```

For instance, the vehicle sends messages as below, at every 5 seconds:

```
40.733229, -73.9954147, 60,"33 5th Avenue NYC", 2016-08-01 01:01:15
```

By sending the messages at every 5 seconds, the tablespace **streams_table_car1** is populated with new entries:

Example when reading the table through dataframes(during the spark-submit process)



```
17/12/03 14:17:50 INFO executor.Executor: Running task 0.0 in stage 26.0 (TID 26)
17/12/03 14:17:50 INFO jdbc.JDBCRDD: closed connection
17/12/03 14:17:50 INFO executor.Executor: Finished task 0.0 in stage 26.0 (TID 26). 2422 bytes result sent to driver
17/12/03 14:17:50 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 26.0 (TID 26) in 38 ms on localhost (1/1)
17/12/03 14:17:50 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, from pool
17/12/03 14:17:50 INFO scheduler.DAGScheduler: ResultStage 26 (show at ScalaStreaming.scala:138) finished in 0.024 s
17/12/03 14:17:50 INFO scheduler.DAGScheduler: Job 26 finished: show at ScalaStreaming.scala:138, took 0.057601 s
+-------+--------+-----+--------------------+--------------------+
|    lat|     lon|speed|            location|                date|
+-------+--------+-----+--------------------+--------------------+
| 40.733|-73.9956| 50.0|   25 5th Avenue NYC|2016-08-01 01:01:...|
| 40.733|-73.9956| 50.0|   25 5th Avenue NYC|2016-08-01 01:01:...|
|40.7342|-73.9946| 58.0|   45 5th Avenue NYC|2016-08-01 01:01:...|
|40.7332|-73.9954| 60.0|   33 5th Avenue NYC|2016-08-01 01:01:...|
|40.7335|-73.9952| 45.0|   35 5th Avenue NYC|2016-08-01 01:01:...|
|40.7342|-73.9946| 58.0|   45 5th Avenue NYC|2016-08-01 01:01:...|
|40.7345|-73.9944| 63.0|   51 5th Avenue NYC|2016-08-01 01:01:...|
|40.7345|-73.9942| 58.0|34 1/2 East 12th ...|2016-08-01 01:01:...|
|40.7344|-73.9939| 62.0|  29-1 E 12th St NYC|2016-08-01 01:01:...|
|40.7343|-73.9937| 66.0|        10 E 12th St|2016-08-01 01:01:...|
|40.7341|-73.9932| 68.0|  29-1 E 12th St NYC|2016-08-01 01:01:...|
+-------+--------+-----+--------------------+--------------------+

17/12/03 14:17:50 INFO spark.SparkContext: Starting job: jdbc at ScalaStreaming.scala:141
17/12/03 14:17:50 INFO scheduler.DAGScheduler: Got job 27 (jdbc at ScalaStreaming.scala:141) with 1 output partitions
```

iv) Creating the map and vehicle location

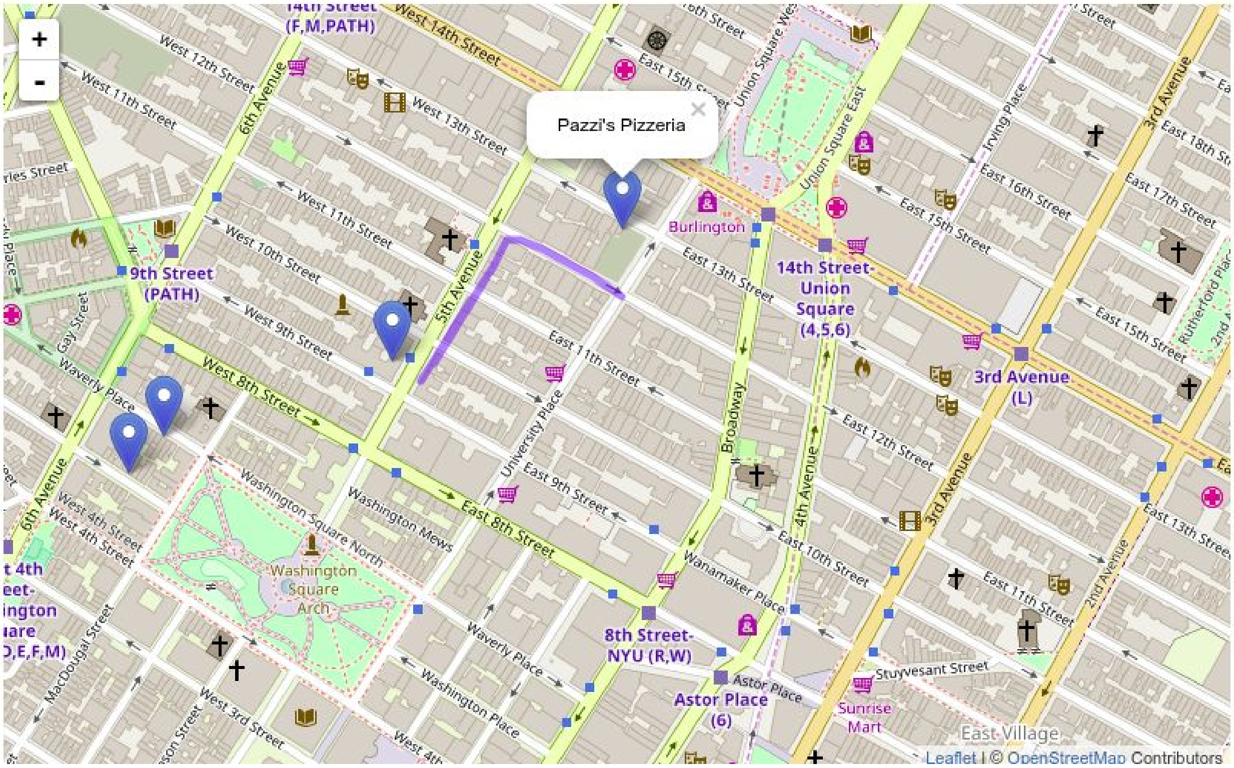
For this section, the implementation will be performed with Leaflet map and extracting data from MySQL with PHP.

Based on the tablespace entries, the vehicle location can be tracked.

At the moment when the map below has been created, tablespace contains the following rows:

```
mysql> select * from streams_table_car1;
+---------+----------+-------+-----------------------------+---------------------+
| lat     | lon      | speed | location                    | date                |
+---------+----------+-------+-----------------------------+---------------------+
|  40.733 | -73.9956 |    50 | 25 5th Avenue NYC           | 2016-08-01 01:01:10 |
| 40.7342 | -73.9946 |    58 | 45 5th Avenue NYC           | 2016-08-01 01:01:25 |
| 40.7332 | -73.9954 |    60 | 33 5th Avenue NYC           | 2016-08-01 01:01:15 |
| 40.7335 | -73.9952 |    45 | 35 5th Avenue NYC           | 2016-08-01 01:01:20 |
| 40.7342 | -73.9946 |    58 | 45 5th Avenue NYC           | 2016-08-01 01:01:25 |
| 40.7345 | -73.9944 |    63 |  51 5th Avenue NYC          | 2016-08-01 01:01:30 |
| 40.7345 | -73.9942 |    58 | 34 1/2 East 12th Street NYC | 2016-08-01 01:01:35 |
| 40.7344 | -73.9939 |    62 | 29-1 E 12th St NYC          | 2016-08-01 01:01:40 |
| 40.7343 | -73.9937 |    66 | 10 E 12th St                | 2016-08-01 01:01:45 |
+---------+----------+-------+-----------------------------+---------------------+
10 rows in set (0.00 sec)
```

The purple lines indicates the vehicle’s trajectory, as per coordinates from MySQL tablespace



(a few restaurants marked down)

PHP scripts for finding the trajectory and drawing map

```
<?php  ?>
<!DOCTYPE html>
<html>
<head>
    <title>Track me down</title>
    <meta charset="utf-8" />
    <link
        rel="stylesheet"
        href="http://cdn.leafletjs.com/leaflet-0.7/leaflet.css"
    />
</head>
<body>

    <div id="map" style="width: 800px; height: 500px"></div>

    <script
        src="http://cdn.leafletjs.com/leaflet-0.7/leaflet.js">
    </script>

    <script>

       <?php include '/var/www/html/trackme.php'; ?>

        var map = L.map('map').setView([40.73, -74.00], 14);
        mapLink =
            '<a href="http://openstreetmap.org">OpenStreetMap</a>';
        L.tileLayer(
            'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '&copy; ' + mapLink + ' Contributors',
            maxZoom: 18,
            }).addTo(map);

        var polyline = L.polyline(trackeme).addTo(map);
 </script>

       <script>
       var places = [
             ["Blue Hill",40.7320, -73.9997],
             ["Babbo Ristorante e Enoteca", 40.7324, -73.9992],
             ["Claudette", 40.7332, -73.9960],
             ["Pazzi's Pizzeria",  40.7346, -73.9928]
                          ];


             for (var i = 0; i < places.length; i++) {
                    marker = new L.marker([places[i][1],places[i][2]])
                           .bindPopup(places[i][0])
                           .addTo(map);
                    }
       </script>
</body>
</html>
```

Script trackme.php to extract coordinates from tablespace

```
<?php
    $username = "root";
    $password = "M0ns00n!!!";
    $host = "localhost";
    $database="streams_db";

    $server = mysql_connect($host, $username, $password);
    $connection = mysql_select_db($database, $server);

    $myquery = "
SELECT  `lat`, `lon` FROM  `streams_table_car1`
WHERE `lat` <> 0
";
    $query = mysql_query($myquery);

    if ( ! $query ) {
        echo mysql_error();
        die;
    }

    $data = array();
       echo "var latlon = [";

    for ($x = 0; $x < mysql_num_rows($query); $x++) {
        $data[] = mysql_fetch_assoc($query);
        echo "[",$data[$x]['lat'],",",$data[$x]['lon'],"]";
        if ($x <= (mysql_num_rows($query)-2) ) {
                    echo ",";
             }
    }

       echo "];";
    mysql_close($server);
?>
```

v) Finding the nearest restaurants

Two steps are implemented:

a) Finding the coordinates of the vehicle when the driver is in search of the nearest restaurants.

This requires to find the last entry in the tablespace. The searching is done after ‘date’ column.

b) Implementing the Great Circle Distance formula

```
r * acos[sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(lon2 - lon1)]
```

PHP scripts for implementing points a) and b)

```
bash-4.1# more morelatlong.php
<?PHP
/**
 * It appliese Haversine Formula
 * on a distance of 5miles
 **/

 $username = "root";
    $password = "M0ns00n!!!";
    $host = "localhost";
    $database="streams_db";

    $server = mysql_connect($host, $username, $password);
    $connection = mysql_select_db($database, $server);


$tableName = "Restaurant";

$Lat=`php lat.php`;

$Lon = `php lon.php`;

$dist = 5; // max distance (in miles) away from $Lat,$Lon
```

```
$mysqlquery = "SELECT location, lat, lon, 3956 * 2 *
          ASIN(SQRT( POWER(SIN(($Lat - lat)*pi()/180/2),2)
          +COS($Lat*pi()/180 )*COS(lat*pi()/180)
          *POWER(SIN(($Lon-lon)*pi()/180/2),2)))
          as distance FROM $tableName WHERE
          lon between ($Lon-$dist/cos(radians($Lat))*69)
          and ($Lon+$dist/cos(radians($Lat))*69)
          and lat between ($Lat-($dist/69))
          and ($Lat+($dist/69))
          having distance < $dist ORDER BY distance limit 10";

$result = mysql_query($mysqlquery) or die(mysql_error());

while($row = mysql_fetch_assoc($result)) {
    echo " | " . $row['location']." | ".$row['distance']."\n";
}
mysql_close($server);
?>
```

Below, the called scripts for finding the latitude and longitude at the searching time (the latest row inserted)

Script for **finding latitude at requesting time**

```
<?PHP
/**
 * lat.php
 **/

 $username = "root";
    $password = "M0ns00n!!!";
    $host = "localhost";
    $database="streams_db";

    $server = mysql_connect($host, $username, $password);
    $connection = mysql_select_db($database, $server);


$string1 = "lat";
$queryLat = "select lat from streams_table_car1 where date=(select max(date) from
streams_table_car1)";

$result=mysql_query($queryLat);

while ($row = mysql_fetch_assoc($result)) {
    echo $row['lat'];
}

mysql_close($server);

?>
```

Script for **finding longitude at requesting time**

```
<?php
/**
 * lon.php
 **/
 $username = "root";
    $password = "M0ns00n!!!";
    $host = "localhost";
    $database="streams_db";

    $server = mysql_connect($host, $username, $password);
    $connection = mysql_select_db($database, $server);


$queryLat = "select lon from streams_table_car1 where date=(select max(date) from
streams_table_car1)";

$result=mysql_query($queryLat);

while ($row = mysql_fetch_assoc($result)) {
    echo $row['lon'];
}

mysql_close($server);

?>
```

Restaurants tablespace is populated as below:

```
+---------+----------+-----------------------------+
| lat     | lon      | location                    |
+---------+----------+-----------------------------+
| 40.7324 | -73.9992 | Babbo Ristorante e Enoteca  |
| 40.732  | -73.9997 | Blue Hill                   |
| 40.7332 |  -73.996 | Claudette                   |
| 40.7346 | -73.9928 | Pazzi's Pizzeria            |
+---------+----------+-----------------------------+
Output of the script, on a distance of 5miles:

bash-4.1# php morelatlong.php
 | Pazzi's Pizzeria            | 0.0483364239446441
 | Claudette                   | 0.174367103438163
 | Babbo Ristorante e Enoteca  | 0.35046220803954
 |  Blue Hill                  | 0.384259956989441
```

Real case scenario:

1) PHP OOP to be applied if we are dealing with a large scale of tracking web application

2) PHP 7 recommended

3) If GPS logs look like XML files, they can be easily manipulated with Perl using XML::LibXML library, and turned into a csv to be sent in Kafka Producer.

Thank you for your attention :)