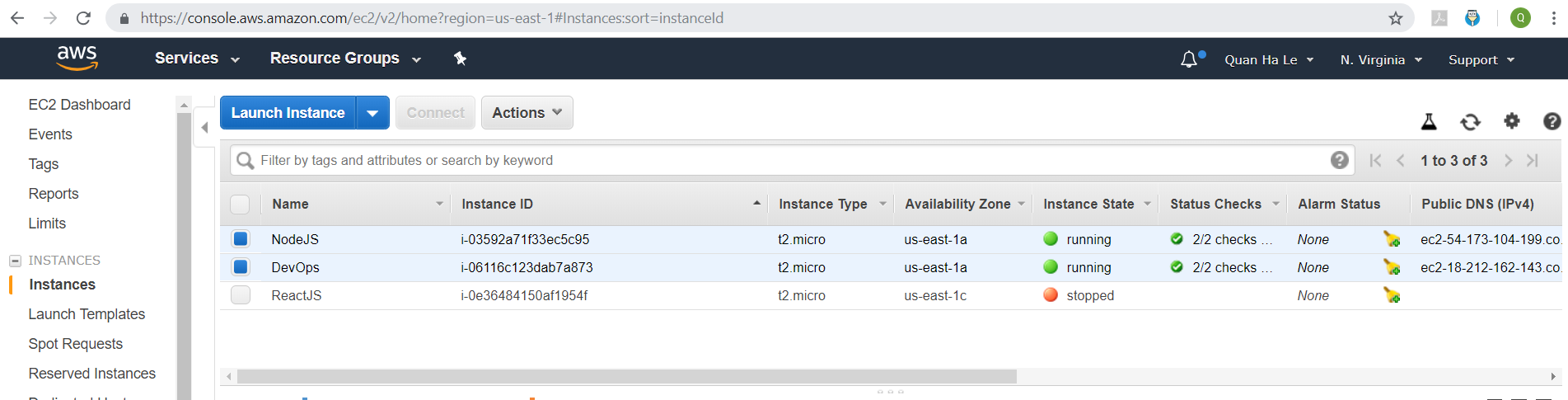
Candidate: QUAN-HA LE

**Installation**

I used 2 ec2 instances on AWS to solve the problem as the screenshot below



- NodeJS ec2 instance:  this one is the NodeJS API to manipulate data into the PostgreSQL with the following routes

router.get('/api/data', db.getAllData);

router.get('/api/clusters', db.getAllClusters);

router.get('/api/sensors', db.getAllSensors);

router.get('/api/data/:id', db.getSingleData);

router.post('/api/data/:clusterid/:sensorid/:data', db.createData);

router.get('/api/cluster/:mean/:deviation', db.getClusterID);

router.post('/api/cluster/:mean/:deviation', db.createCluster);

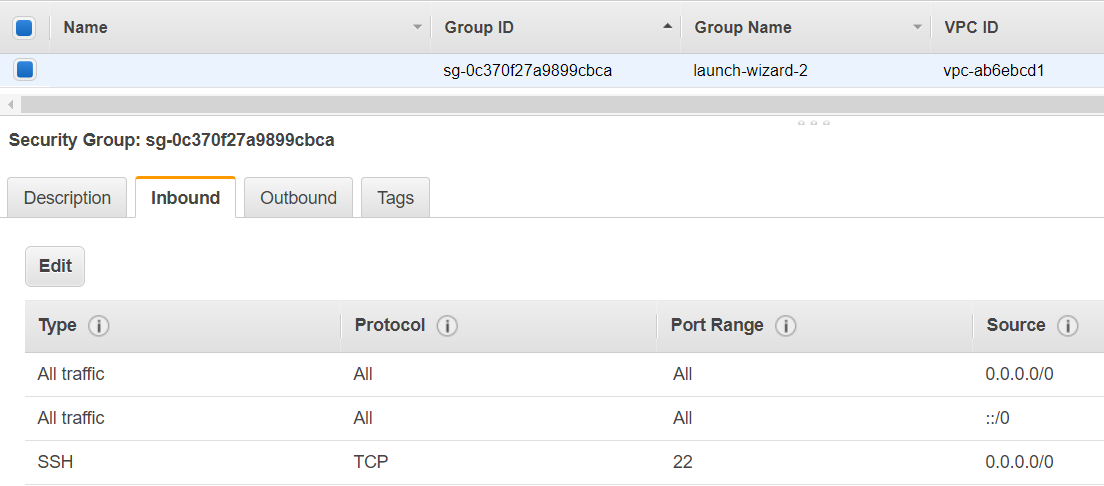
router.delete('/api/cluster/:mean/:deviation', db.removeCluster);

router.post('/api/sensor/:id/:name', db.createSensor);

router.delete('/api/sensor/:name', db.removeSensor);

- DevOps ec2 instance: this one is the Python 3.7 code to generate sensor data. The python code sends its data into NodeJS to update your PostgreSQL RDS

Because we will need setting up Security Groups so that the Python 3.7 and the NodeJS ec2 instances can be accessed properly, at the moment, I only set security groups as All Traffic



However, you can also install both of the NodeJS and Python modules onto one common ec2 instance, in this case the Python module will call the NodeJS server by ‘localhost’ instead of calling the NodeJS ec2 public DNS.

Please use the step-by-step guideline here from the GITHUB to prepare the NodeJS API on CentOS 7.5 (file how-to-prepare-nodejs-environment.txt)

<https://github.com/lequanha/sensors/blob/master/how-to-prepare-nodejs-environment.txt>

Please use the step-by-step guideline here from the GITHUB to prepare the NodeJS API on CentOS 7.5 and Python 3.7 (file how-to-prepare-python-environment.txt)

<https://github.com/lequanha/sensors/blob/master/how-to-prepare-python-environment.txt>

**PostgreSQL database structure**

PostgreSQL RDS Endpoint/Hostname: artesianproject.c0pbwgrsqsdx.us-east-1.rds.amazonaws.com

There are 3 entities inside this problem: sensor, cluster and data.

|  |
| --- |
| SENSOR  CLUSTER  DATA |

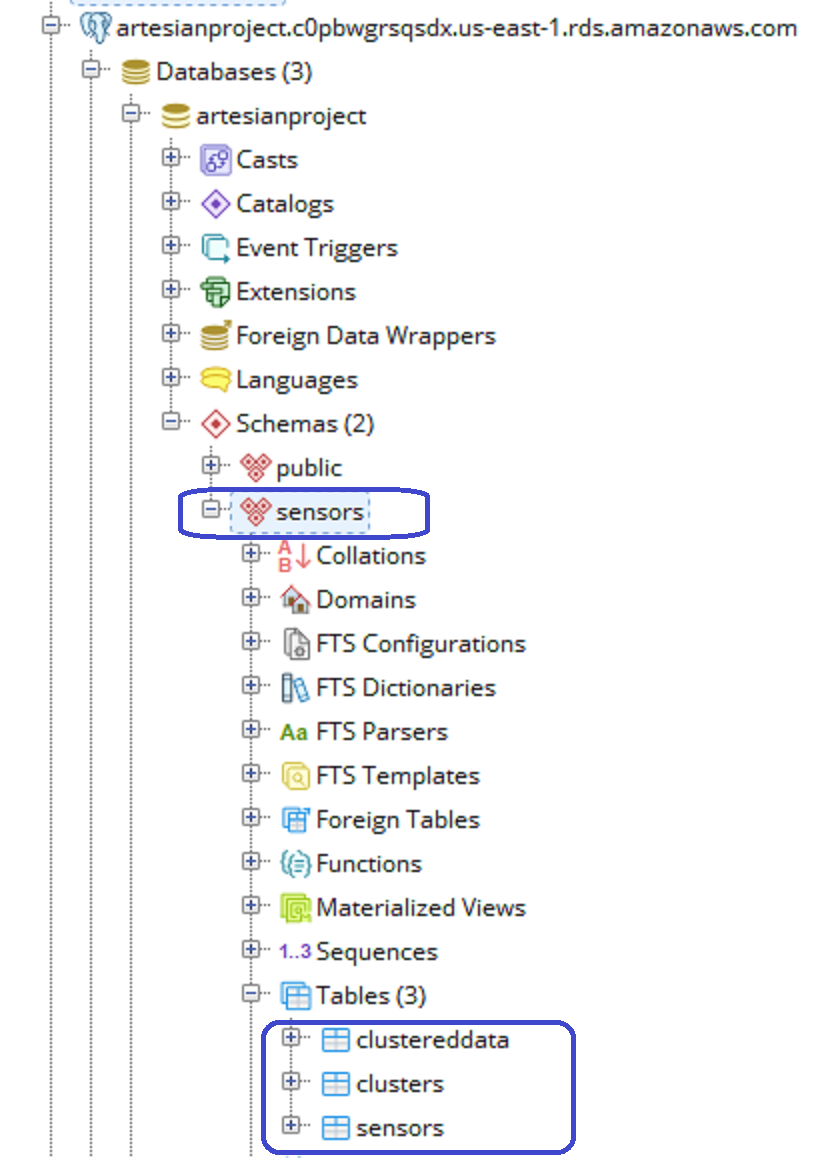
Hence, there are there tables inside the PostgreSQL database

* sensors(sensorid, name): each sensor has a sensor id and a name
* clusters(clusterid, mean, deviation): each cluster includes a mean value and a deviation value
* clustereddata (id, clusterid, sensorid, data): each data record belongs to a cluster, that receives from a sensor

Because the data in the database should be easily sorted and searched, additional to the primary keys, the following database indexes are created

* idx\_data: on data column of clustereddata table
* fki\_sensorid: on sensorid column of clustereddata table
* fki\_clusterid: on clusterid column of clustereddata table
* idx\_mean: on mean column of clusters table
* idx\_deviation: on deviation column of clusters table
* idx\_name: on name column of sensors table

Inside the PostgreSQL RDS, I created all 3 data tables inside a new schema called "sensors".



Please find the RDS SQL backup file postgres\_rds.sql inside GITHUB

<https://github.com/lequanha/sensors/blob/master/postgres_rds.sql>

**Software code**

The Python software uses paho.mqtt.client and threading packages to create X clients and X parallel threads, that is the design for the required X (user inputted/specified) sources.

clients = [mqtt.Client("Sensor"+str(i)) for i in range(X)]

sensors = [Thread(name='sensor'+str(i), target=publishData, args=(clients[i], curlData + str(i+1) + '/', Y, 20)) for i in range(X)]

At first from Python, the input arguments of mean, deviation, X (sources), Y delaying seconds and NodeJS API DNS can be taken from command-line and/or from questions as

if len(sys.argv) > 1:

print('Mean: ', sys.argv[1])

mean = float(sys.argv[1])

else:

mean = float(input("Please enter mean value? "))

if len(sys.argv) > 2:

print('Deviation: ', sys.argv[2])

deviation = float(sys.argv[2])

else:

deviation = float(input("Please enter deviation value? "))

if len(sys.argv) > 3:

print('Number of Sources: ', sys.argv[3])

X = int(sys.argv[3])

else:

X = int(input("Please enter Number of Sources? "))

if len(sys.argv) > 4:

print('Delay: ', sys.argv[4], ' seconds')

Y = int(sys.argv[4])

else:

Y = int(input("Please enter delaying time in seconds? "))

if len(sys.argv) > 5:

print('NodeJS API DNS: ', sys.argv[5])

nodedns = sys.argv[5]

else:

nodedns = input("Please enter the NodeJS API public DNS for your PostgreSQL database (Leave it blank / empty for default value = localhost) ? ")

if len(nodedns) == 0:

nodedns = 'localhost'

If you leave the NodeJS DNS empty, it means that you installed NodeJS and Python 3.7 modules on the same only one ec2 instance, hence that will be set up as localhost NodeJS server.

After that, Python software will create X sensors into the PostgreSQL database by curl statements to the NodeJS server

name = 'Sensor ' + str(si+1)

url = 'http://' + nodedns + ':3000/api/sensor/' + str(si+1) + '/' + name;

res = requests.post(url, data={}, headers={})

if (res.status\_code != requests.codes.ok):

print("Sensor " + name + " already exists in the PostgreSQL database")

this is the POST curl for /api/sensor/:id/:name of NodeJS server

Then the clustered data values are generated based on the mean and the deviation

data = random.normalvariate(mean, deviation)

the following POST curl statements are called from Python 3.7 as /api/data/:clusterid/:sensorid/:data of NodeJS server

curlData = 'http://' + nodedns + ':3000/api/data/' + str(clusterID) + '/'

and then

url = topic + str(data)

res = requests.post(url, data={}, headers={})

in which topic is the curlData parameter, hence this will be

http://nodedns:3000/api/data/:clusterid/:sensorid/:data

The NodeJS API server is made by Express, BlueBird, and pg-promise.

These are examples about curl calling to this NodeJS API ec2 instances (http://ec2-54-226-103-117.compute-1.amazonaws.com)

* if you like to list all the clustered data from Chrome, please use

[http://ec2-54-226-103-117.compute-1.amazonaws.com:3000**/api/data**](http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/data)

* If you like to retrieve a data record id 1055, please use

[http://ec2-54-226-103-117.compute-1.amazonaws.com:3000**/api/data/1055**](http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/data/1055)

* If you like to list all sensors from Chrome, please use

<http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/sensors>

* If you like to list all clusters (means/deviations), please use

<http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/clusters>

* If you like to see if there is any cluster with mean = 1000 and deviation = 2 on Chrome, please use

<http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/cluster/1000/2>

* If you like to insert a new clustereddata with data = 1300.234567, for sensor id = 2 and cluster id = 9, please try in CentOS

[root@ip-172-31-84-126 centos]# curl -X POST <http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/data/9/2/1300.234567>

{"status":"success","message":"Inserted one data"}

[root@ip-172-31-84-126 centos]#

* If you like to insert a new sensor with id = 24 and name = Sensor 24, please try in CentOS

[root@ip-172-31-84-126 centos]# curl -X POST <http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/sensor/24/Sensor%2024>

{"status":"success","message":"Inserted one sensor"}

[root@ip-172-31-84-126 centos]#

* If you like to insert a new cluster with mean = 2224.14 and deviation = 2.7, please try in CentOS

[root@ip-172-31-84-126 centos]# curl -X POST <http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/cluster/2224.14/2.7>

{"status":"success","message":"Inserted one cluster"}

[root@ip-172-31-84-126 centos]#

* If you like to delete the above sensor named Sensor 24, please try in CentOS

[root@ip-172-31-84-126 centos]# curl -X DELETE <http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/sensor/Sensor%2024>

{"status":"success","message":"Removed 1 sensors"}

[root@ip-172-31-84-126 centos]#

* If you like to delete the above cluster with mean = 2224.14 and deviation = 2.7, please try in CentOS

[root@ip-172-31-84-126 centos]# curl -X DELETE <http://ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/cluster/2224.14/2.7>

{"status":"success","message":"Removed 1 clusters"}

[root@ip-172-31-84-126 centos]#

**Software execution**

At first, the NodeJS server needs starting first from the app main folder like this

[root@ip-172-31-84-126 node-api]# npm start

> node-api@0.0.0 start /root/api/node-api

> node ./bin/www

So that we can run the Python module above the NodeJS API module

In order to run Python software, there are two ways to call with the below input values

* Mean = 8888.88
* Deviation = 8.1
* X = 30 sources
* Y = 4 delaying seconds
* NodeJS API DNS = ec2-54-226-103-117.compute-1.amazonaws.com

The first way that we put them all into one command-line call

[root@ip-172-31-95-82 python-generator]# python3.7 SensorEmulator.py 8888.88 8.1 30 4 ec2-54-226-103-117.compute-1.amazonaws.com

Mean: 8888.88

Deviation: 8.1

Number of Sources: 30

Delay: 4 seconds

NodeJS API DNS: ec2-54-226-103-117.compute-1.amazonaws.com

We are initializing 30 sensors into the PostgreSQL database...

Sensor Sensor 1 already exists in the PostgreSQL database

Sensor Sensor 2 already exists in the PostgreSQL database

Sensor Sensor 3 already exists in the PostgreSQL database

Sensor Sensor 4 already exists in the PostgreSQL database

Sensor Sensor 5 already exists in the PostgreSQL database

Sensor Sensor 6 already exists in the PostgreSQL database

Sensor Sensor 7 already exists in the PostgreSQL database

Sensor Sensor 8 already exists in the PostgreSQL database

Sensor Sensor 9 already exists in the PostgreSQL database

Sensor Sensor 10 already exists in the PostgreSQL database

Sensor Sensor 11 already exists in the PostgreSQL database

Sensor Sensor 12 already exists in the PostgreSQL database

Sensor Sensor 13 already exists in the PostgreSQL database

Sensor Sensor 14 already exists in the PostgreSQL database

Sensor Sensor 15 already exists in the PostgreSQL database

value=8901.927312710677

value=8902.094307356943

value=8881.811780775683

value=8890.83576483073

value=8900.969644798448

value=8885.004043324043

value=8879.34712476851

value=8895.801201920549

The second way that we can input each value manually

[root@ip-172-31-95-82 python-generator]# python3.7 SensorEmulator.py

Please enter mean value? 8888.88

Please enter deviation value? 8.1

Please enter Number of Sources? 30

Please enter delaying time in seconds? 4

Please enter the NodeJS API public DNS for your PostgreSQL database (Leave it blank / empty for default value = localhost) ? ec2-54-226-103-117.compute-1.amazonaws.com

We are initializing 30 sensors into the PostgreSQL database...

Sensor Sensor 1 already exists in the PostgreSQL database

Sensor Sensor 2 already exists in the PostgreSQL database

Sensor Sensor 3 already exists in the PostgreSQL database

Sensor Sensor 4 already exists in the PostgreSQL database

Sensor Sensor 5 already exists in the PostgreSQL database

Sensor Sensor 6 already exists in the PostgreSQL database

Sensor Sensor 7 already exists in the PostgreSQL database

Sensor Sensor 8 already exists in the PostgreSQL database

Sensor Sensor 9 already exists in the PostgreSQL database

Sensor Sensor 10 already exists in the PostgreSQL database

Sensor Sensor 11 already exists in the PostgreSQL database

Sensor Sensor 12 already exists in the PostgreSQL database

Sensor Sensor 13 already exists in the PostgreSQL database

Sensor Sensor 14 already exists in the PostgreSQL database

Sensor Sensor 15 already exists in the PostgreSQL database

Sensor Sensor 16 already exists in the PostgreSQL database

Sensor Sensor 17 already exists in the PostgreSQL database

Sensor Sensor 18 already exists in the PostgreSQL database

Sensor Sensor 19 already exists in the PostgreSQL database

Sensor Sensor 20 already exists in the PostgreSQL database

Sensor Sensor 21 already exists in the PostgreSQL database

Sensor Sensor 22 already exists in the PostgreSQL database

Sensor Sensor 23 already exists in the PostgreSQL database

Sensor Sensor 24 already exists in the PostgreSQL database

Sensor Sensor 25 already exists in the PostgreSQL database

Sensor Sensor 26 already exists in the PostgreSQL database

Sensor Sensor 27 already exists in the PostgreSQL database

Sensor Sensor 28 already exists in the PostgreSQL database

Sensor Sensor 29 already exists in the PostgreSQL database

Sensor Sensor 30 already exists in the PostgreSQL database

value=8882.313908101714

value=8884.566982254564

value=8899.537947532364

value=8888.821844533384

value=8896.822485324954

value=8900.194272283712

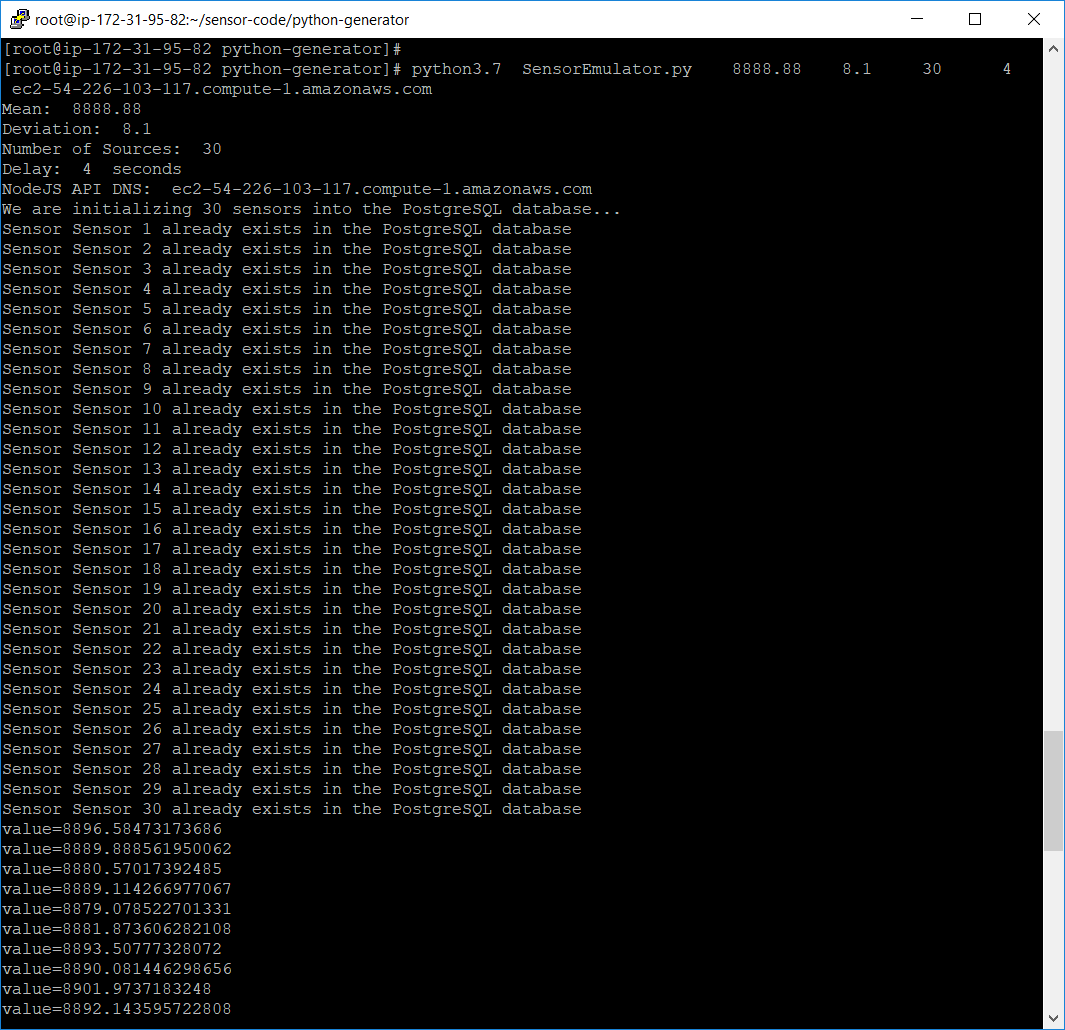
value=8891.66765640433

value=8884.94345408517

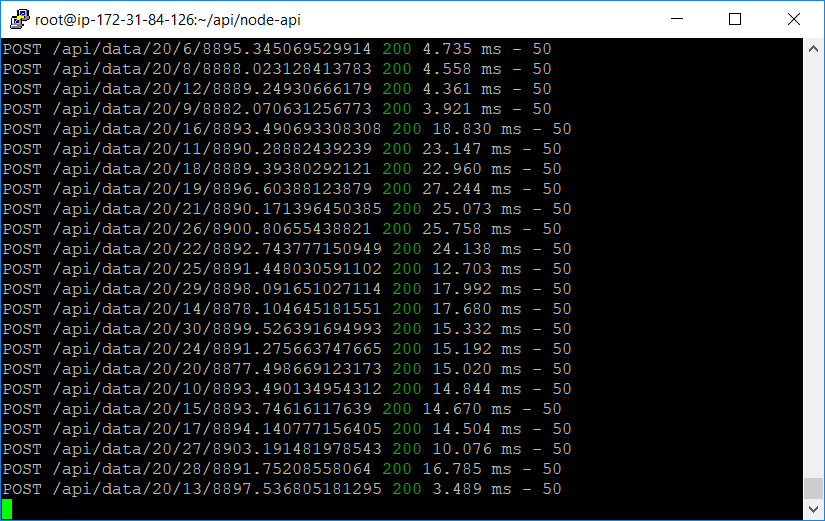
value=8891.320449799463

value=8888.835522469242

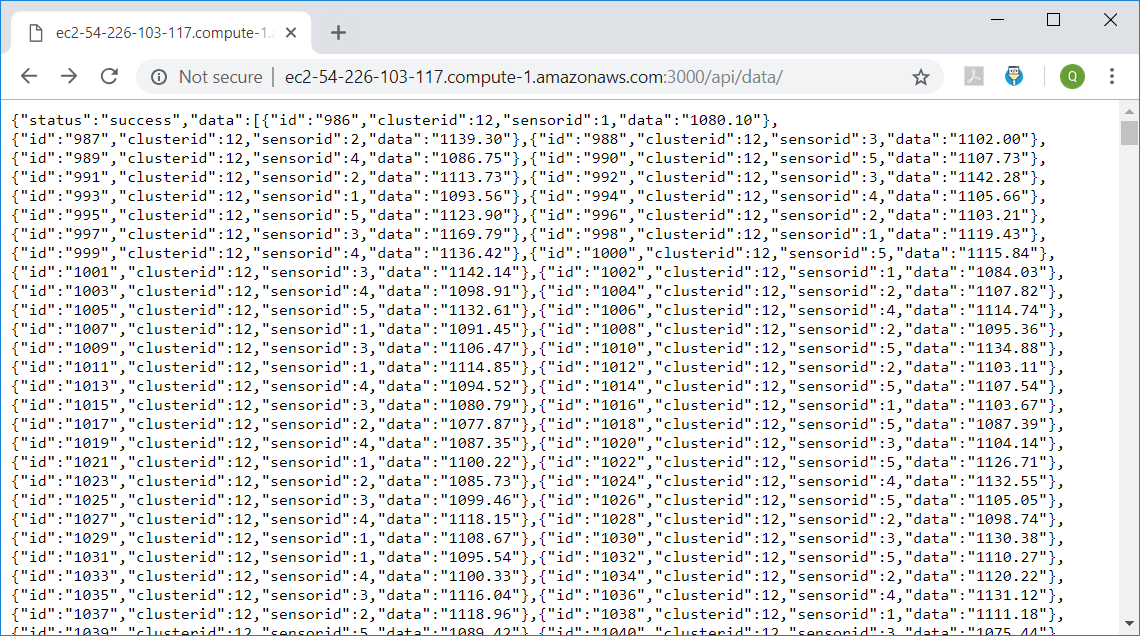
When I run the Python program, this is the Python screenshot



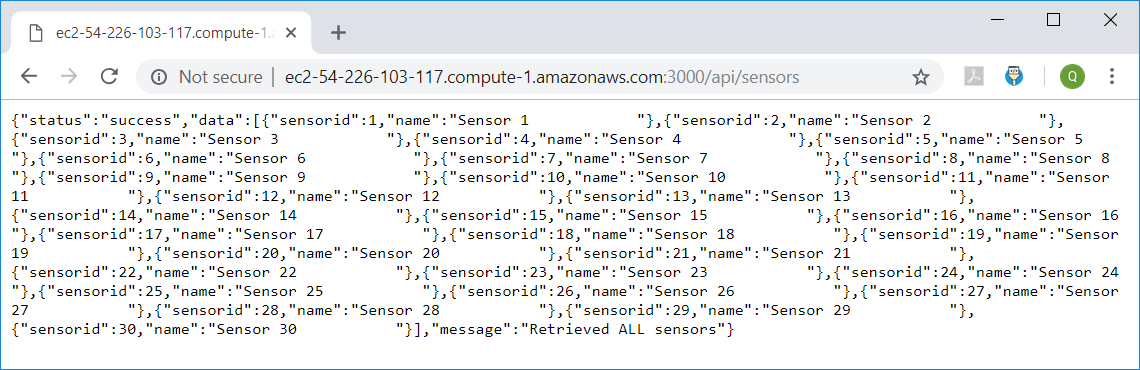
This is the NodeJS screenshot because the above Python ec2 instance sent API curl requests to NodeJS



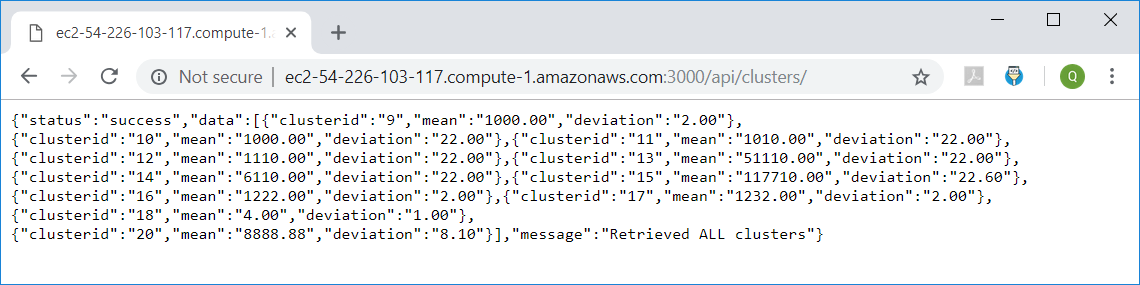
Then you can see the generated data results through Chrome browser



This is the result of sensors created from Python



This is the result of clusters created from Python (each pair of mean/deviation will be 1 cluster)



**More results**

I have done tested as above for 2 ec2 instances, NodeJS API ec2 and Python 3.7 ec2 on CentOS 7.5, however now I would like to show you the localhost NodeJS server test in which I can use only one ec2 instance to install both NodeJS and Python modules together, *in this case we leave the value for NodeDNS empty* so that it will be localhost

