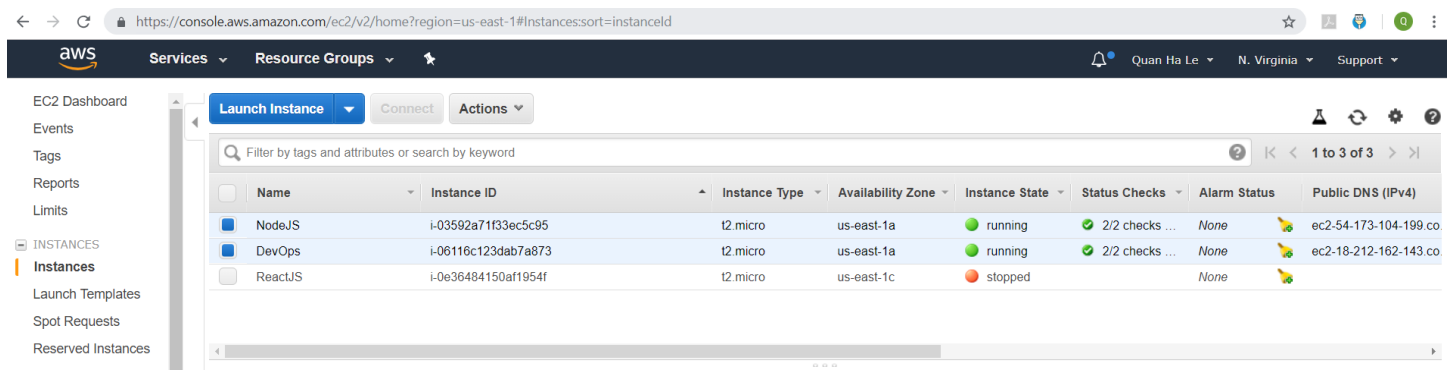


Candidate: QUAN-HA LE

Installation

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

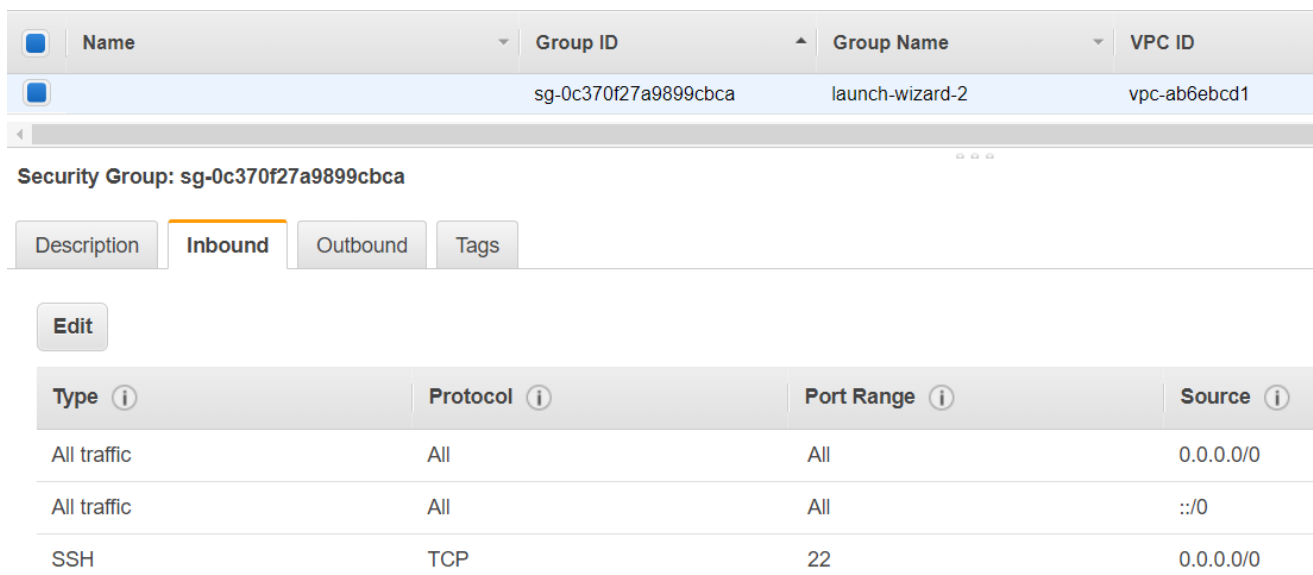


- NodeJS ec2 instance: this one is the NodeJS API to manipulate data into the PostgreSQL with the following routes

```
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.



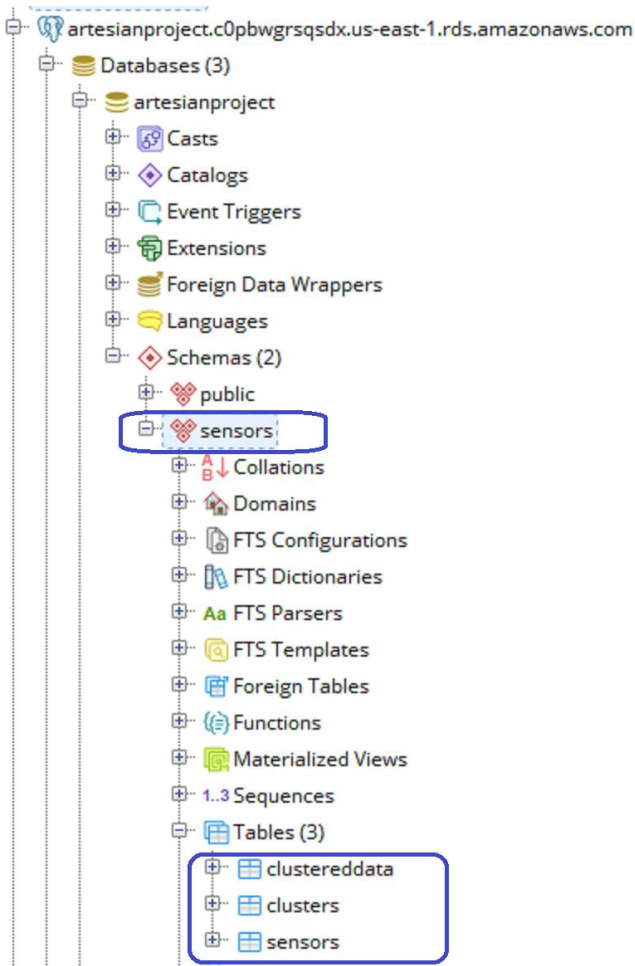
Hence, there are three tables inside the PostgreSQL database

- 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>

- 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>

- 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

```
root@ip-172-31-95-82:~/sensor-code/python-generator
[root@ip-172-31-95-82 python-generator]#
[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
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=8896.58473173686
value=8889.888561950062
value=8880.57017392485
value=8889.114266977067
value=8879.078522701331
value=8881.873606282108
value=8893.50777328072
value=8890.081446298656
value=8901.9737183248
value=8892.143595722808
```

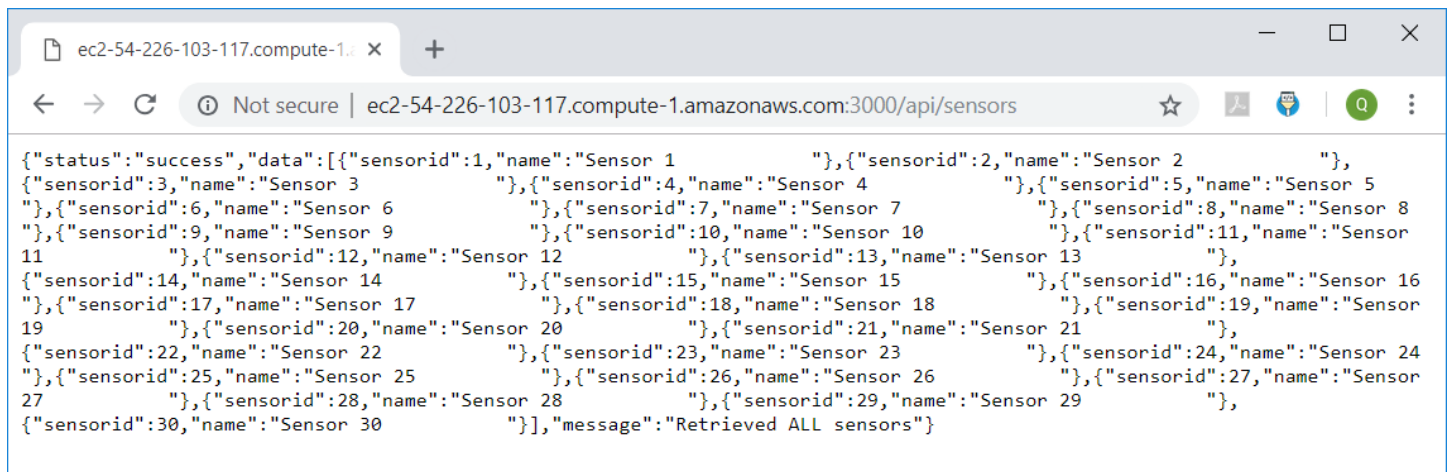

This is the NodeJS screenshot because the above Python ec2 instance sent API curl requests to NodeJS

```
root@ip-172-31-84-126:~/api/node-api
POST /api/data/20/6/8895.345069529914 200 4.735 ms - 50
POST /api/data/20/8/8888.023128413783 200 4.558 ms - 50
POST /api/data/20/12/8889.24930666179 200 4.361 ms - 50
POST /api/data/20/9/8882.070631256773 200 3.921 ms - 50
POST /api/data/20/16/8893.490693308308 200 18.830 ms - 50
POST /api/data/20/11/8890.28882439239 200 23.147 ms - 50
POST /api/data/20/18/8889.39380292121 200 22.960 ms - 50
POST /api/data/20/19/8896.60388123879 200 27.244 ms - 50
POST /api/data/20/21/8890.171396450385 200 25.073 ms - 50
POST /api/data/20/26/8900.80655438821 200 25.758 ms - 50
POST /api/data/20/22/8892.743777150949 200 24.138 ms - 50
POST /api/data/20/25/8891.448030591102 200 12.703 ms - 50
POST /api/data/20/29/8898.091651027114 200 17.992 ms - 50
POST /api/data/20/14/8878.104645181551 200 17.680 ms - 50
POST /api/data/20/30/8899.526391694993 200 15.332 ms - 50
POST /api/data/20/24/8891.275663747665 200 15.192 ms - 50
POST /api/data/20/20/8877.498669123173 200 15.020 ms - 50
POST /api/data/20/10/8893.490134954312 200 14.844 ms - 50
POST /api/data/20/15/8893.74616117639 200 14.670 ms - 50
POST /api/data/20/17/8894.140777156405 200 14.504 ms - 50
POST /api/data/20/27/8903.191481978543 200 10.076 ms - 50
POST /api/data/20/28/8891.75208558064 200 16.785 ms - 50
POST /api/data/20/13/8897.536805181295 200 3.489 ms - 50
```

Then you can see the generated data results through Chrome browser

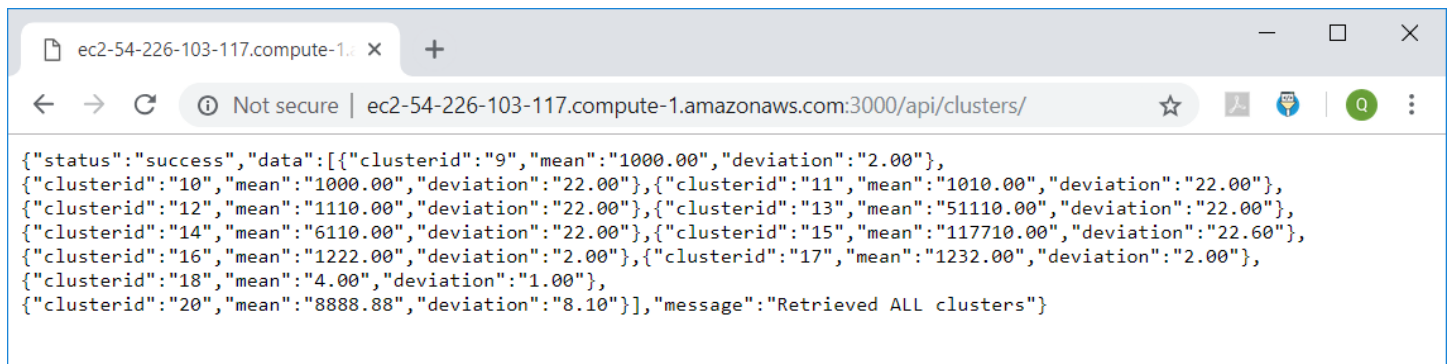
```
ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/data/
Not secure | ec2-54-226-103-117.compute-1.amazonaws.com:3000/api/data/
{"status":"success","data":[{"id":"986","clusterid":12,"sensorid":1,"data":"1080.10"},
{"id":"987","clusterid":12,"sensorid":2,"data":"1139.30"},{"id":"988","clusterid":12,"sensorid":3,"data":"1102.00"},
{"id":"989","clusterid":12,"sensorid":4,"data":"1086.75"},{"id":"990","clusterid":12,"sensorid":5,"data":"1107.73"},
{"id":"991","clusterid":12,"sensorid":2,"data":"1113.73"},{"id":"992","clusterid":12,"sensorid":3,"data":"1142.28"},
{"id":"993","clusterid":12,"sensorid":1,"data":"1093.56"},{"id":"994","clusterid":12,"sensorid":4,"data":"1105.66"},
{"id":"995","clusterid":12,"sensorid":5,"data":"1123.90"},{"id":"996","clusterid":12,"sensorid":2,"data":"1103.21"},
{"id":"997","clusterid":12,"sensorid":3,"data":"1169.79"},{"id":"998","clusterid":12,"sensorid":1,"data":"1119.43"},
{"id":"999","clusterid":12,"sensorid":4,"data":"1136.42"},{"id":"1000","clusterid":12,"sensorid":5,"data":"1115.84"},
{"id":"1001","clusterid":12,"sensorid":3,"data":"1142.14"},{"id":"1002","clusterid":12,"sensorid":1,"data":"1084.03"},
{"id":"1003","clusterid":12,"sensorid":4,"data":"1098.91"},{"id":"1004","clusterid":12,"sensorid":2,"data":"1107.82"},
{"id":"1005","clusterid":12,"sensorid":5,"data":"1132.61"},{"id":"1006","clusterid":12,"sensorid":4,"data":"1114.74"},
{"id":"1007","clusterid":12,"sensorid":1,"data":"1091.45"},{"id":"1008","clusterid":12,"sensorid":2,"data":"1095.36"},
{"id":"1009","clusterid":12,"sensorid":3,"data":"1106.47"},{"id":"1010","clusterid":12,"sensorid":5,"data":"1134.88"},
{"id":"1011","clusterid":12,"sensorid":1,"data":"1114.85"},{"id":"1012","clusterid":12,"sensorid":2,"data":"1103.11"},
{"id":"1013","clusterid":12,"sensorid":4,"data":"1094.52"},{"id":"1014","clusterid":12,"sensorid":5,"data":"1107.54"},
{"id":"1015","clusterid":12,"sensorid":3,"data":"1080.79"},{"id":"1016","clusterid":12,"sensorid":1,"data":"1103.67"},
{"id":"1017","clusterid":12,"sensorid":2,"data":"1077.87"},{"id":"1018","clusterid":12,"sensorid":5,"data":"1087.39"},
{"id":"1019","clusterid":12,"sensorid":4,"data":"1118.15"},{"id":"1020","clusterid":12,"sensorid":3,"data":"1098.74"},
{"id":"1021","clusterid":12,"sensorid":1,"data":"1100.22"},{"id":"1022","clusterid":12,"sensorid":5,"data":"1126.71"},
{"id":"1023","clusterid":12,"sensorid":2,"data":"1085.73"},{"id":"1024","clusterid":12,"sensorid":4,"data":"1132.55"},
{"id":"1025","clusterid":12,"sensorid":3,"data":"1099.46"},{"id":"1026","clusterid":12,"sensorid":5,"data":"1105.05"},
{"id":"1027","clusterid":12,"sensorid":4,"data":"1118.15"},{"id":"1028","clusterid":12,"sensorid":2,"data":"1098.74"},
{"id":"1029","clusterid":12,"sensorid":1,"data":"1108.67"},{"id":"1030","clusterid":12,"sensorid":3,"data":"1130.38"},
{"id":"1031","clusterid":12,"sensorid":1,"data":"1095.54"},{"id":"1032","clusterid":12,"sensorid":5,"data":"1110.27"},
{"id":"1033","clusterid":12,"sensorid":4,"data":"1100.33"},{"id":"1034","clusterid":12,"sensorid":2,"data":"1120.22"},
{"id":"1035","clusterid":12,"sensorid":3,"data":"1116.04"},{"id":"1036","clusterid":12,"sensorid":4,"data":"1131.12"},
{"id":"1037","clusterid":12,"sensorid":2,"data":"1118.96"},{"id":"1038","clusterid":12,"sensorid":1,"data":"1111.18"},
{"id":"1039","clusterid":12,"sensorid":5,"data":"1089.42"},{"id":"1040","clusterid":12,"sensorid":3,"data":"1075.44"}]}
```

This is the result of sensors created from Python



```
{
  "status": "success",
  "data": [
    { "sensorid": 1, "name": "Sensor 1" },
    { "sensorid": 2, "name": "Sensor 2" },
    { "sensorid": 3, "name": "Sensor 3" },
    { "sensorid": 4, "name": "Sensor 4" },
    { "sensorid": 5, "name": "Sensor 5" },
    { "sensorid": 6, "name": "Sensor 6" },
    { "sensorid": 7, "name": "Sensor 7" },
    { "sensorid": 8, "name": "Sensor 8" },
    { "sensorid": 9, "name": "Sensor 9" },
    { "sensorid": 10, "name": "Sensor 10" },
    { "sensorid": 11, "name": "Sensor 11" },
    { "sensorid": 12, "name": "Sensor 12" },
    { "sensorid": 13, "name": "Sensor 13" },
    { "sensorid": 14, "name": "Sensor 14" },
    { "sensorid": 15, "name": "Sensor 15" },
    { "sensorid": 16, "name": "Sensor 16" },
    { "sensorid": 17, "name": "Sensor 17" },
    { "sensorid": 18, "name": "Sensor 18" },
    { "sensorid": 19, "name": "Sensor 19" },
    { "sensorid": 20, "name": "Sensor 20" },
    { "sensorid": 21, "name": "Sensor 21" },
    { "sensorid": 22, "name": "Sensor 22" },
    { "sensorid": 23, "name": "Sensor 23" },
    { "sensorid": 24, "name": "Sensor 24" },
    { "sensorid": 25, "name": "Sensor 25" },
    { "sensorid": 26, "name": "Sensor 26" },
    { "sensorid": 27, "name": "Sensor 27" },
    { "sensorid": 28, "name": "Sensor 28" },
    { "sensorid": 29, "name": "Sensor 29" },
    { "sensorid": 30, "name": "Sensor 30" }
  ],
  "message": "Retrieved ALL sensors"
}
```

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



```
{
  "status": "success",
  "data": [
    { "clusterid": "9", "mean": "1000.00", "deviation": "2.00" },
    { "clusterid": "10", "mean": "1000.00", "deviation": "22.00" },
    { "clusterid": "11", "mean": "1010.00", "deviation": "22.00" },
    { "clusterid": "12", "mean": "1110.00", "deviation": "22.00" },
    { "clusterid": "13", "mean": "51110.00", "deviation": "22.00" },
    { "clusterid": "14", "mean": "6110.00", "deviation": "22.00" },
    { "clusterid": "15", "mean": "117710.00", "deviation": "22.60" },
    { "clusterid": "16", "mean": "1222.00", "deviation": "2.00" },
    { "clusterid": "17", "mean": "1232.00", "deviation": "2.00" },
    { "clusterid": "18", "mean": "4.00", "deviation": "1.00" },
    { "clusterid": "20", "mean": "8888.88", "deviation": "8.10" }
  ],
  "message": "Retrieved ALL clusters"
}
```

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

root@ip-172-31-95-82:~/api/node-api

```
> node-api@0.0.0 start /root/api/node-api
> node ./bin/www
```

```
POST /api/sensor/1/Sensor%201 500 292.674 ms - 834
POST /api/sensor/2/Sensor%202 500 15.329 ms - 834
POST /api/sensor/3/Sensor%203 500 10.976 ms - 834
POST /api/sensor/4/Sensor%204 500 16.126 ms - 834
POST /api/sensor/5/Sensor%205 500 10.427 ms - 834
GET /api/cluster/1000.0/2.0 200 5.921 ms - 76
POST /api/data/9/1/997.9146277162757 200 3.514 ms - 50
POST /api/data/9/2/1001.099916329041 200 2.413 ms - 50
POST /api/data/9/3/998.1165673657694 200 3.475 ms - 50
POST /api/data/9/4/999.0909074006163 200 4.128 ms - 50
POST /api/data/9/5/997.890436176134 200 3.758 ms - 50
POST /api/data/9/3/1000.6556703386916 200 6.897 ms - 50
POST /api/data/9/2/996.4415232690715 200 6.950 ms - 50
POST /api/data/9/1/1003.2169578153931 200 7.033 ms - 50
POST /api/data/9/4/1001.3603068774354 200 2.383 ms - 50
POST /api/data/9/5/1002.7295814394975 200 2.164 ms - 50
POST /api/data/9/2/999.940507592581 200 5.681 ms - 50
POST /api/data/9/3/1002.6089227407924 200 3.922 ms - 50
POST /api/data/9/4/999.2775181787368 200 6.940 ms - 50
POST /api/data/9/1/998.9479606822869 200 6.481 ms - 50
POST /api/data/9/5/999.187753559643 200 2.338 ms - 50
POST /api/data/9/2/998.6417351577426 200 13.490 ms - 50
POST /api/data/9/3/1000.5798757910997 200 14.482 ms - 50
POST /api/data/9/5/1000.4712663566015 200 15.063 ms - 50
POST /api/data/9/1/1000.85564848168866 200 14.865 ms - 50
```

root@ip-172-31-95-82:~/sensor

```
value=998.6417351577426
value=1000.5798757910997
value=1000.4712663566015
value=1002.2456774862235
value=1000.6136426121453
value=998.5028772319652
value=995.4085180333117
value=996.3037768512442
value=1001.0036096524681
value=1000.7761185455596
value=997.5880055250661
value=997.4607071673171
value=999.7364498111538
value=998.722352182712
value=998.7650189726024
value=999.5875882897476
```