TEAM - 4 MODULE DESIGN

PREPARED FOR

IAS PROJECT - GROUP 4 - Team 4

IIIT-Hyderabad

PREPARED BY

TEAM 4

Akshay Choudhary- 2020202013

Shweta Arya -2020201047

Module Covered-

- Sensor Registration
- Sensor Management

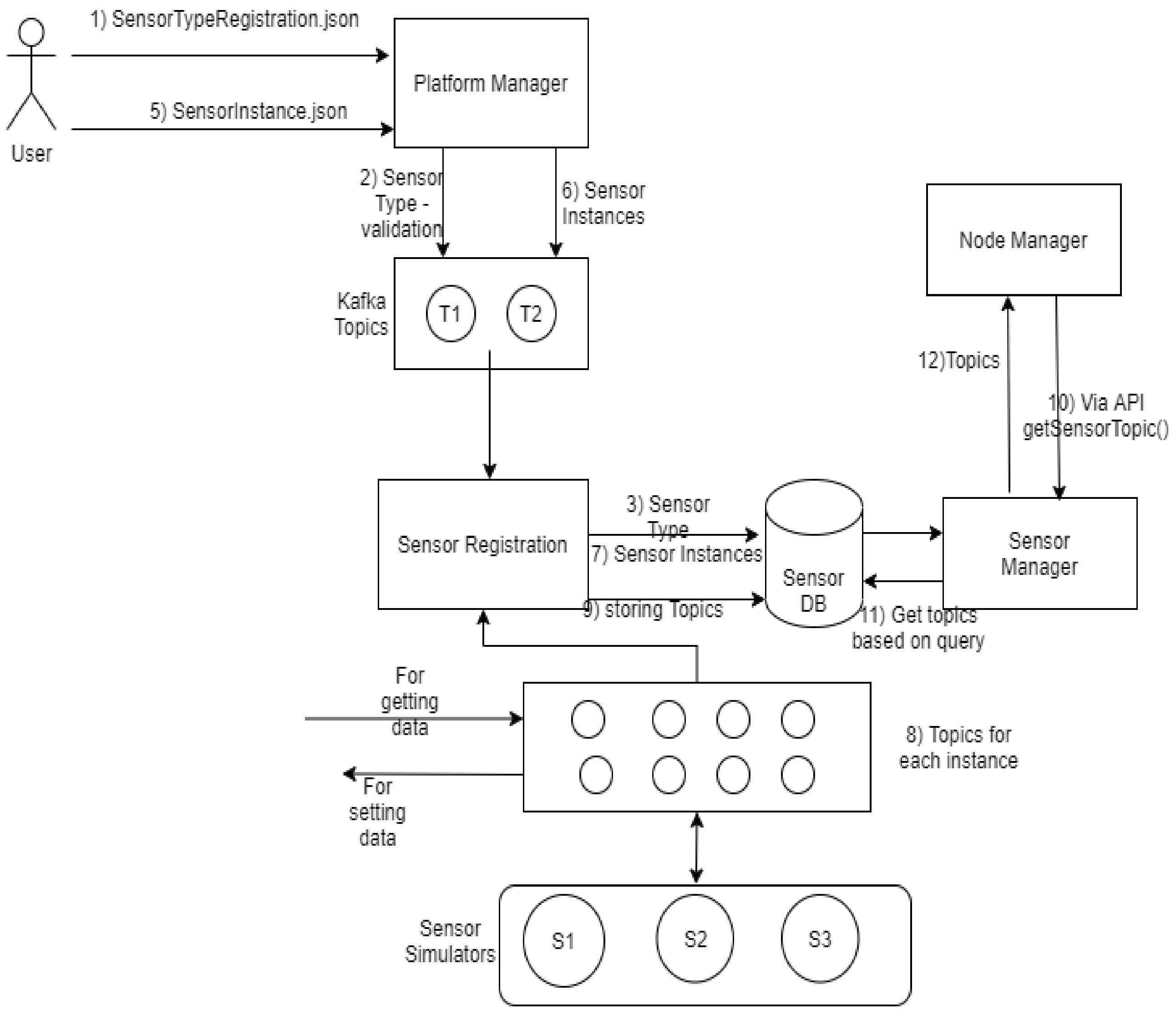
1. Introduction

It is a distributed platform that provides build, development and deployment functionalities of applications. Most data becomes useless just seconds after it is generated, so having the lowest latency possible between the data and the decision is critical. With this platform, we bring IOT capabilities to the field.

This IoT Application platform directly integrates with the IoT devices(sensors) and is capable of performing remote actions on these devices. We need to communicate and get data from sensors to be used by algorithms deployed on the platform and also perform respective actions on the sensors. Our module focuses on this area of the platform.

Our module basically handles Sensor Manager and Sensor Registration . Sensor manager module is responsible for dynamically binding sensor data with running algorithms/instances, and sensor registration for adding a new sensor at platform level, via minimal effort of application developer.

Block diagram



2. Use Cases/Requirements

1. **End to End farm management System App(Primary).**
   To increase the productivity of agricultural and farming processes in order to improve yields and cost-effectiveness with sensor data collection by sensors like soil moisture, temperature sensor,pH sensor,water level sensor.We will use sensor data collection for measurements to make accurate decisions in future. It can be used as a reference for members of the agricultural industry to improve and develop the use of IoT to enhance agricultural production efficiencies.

   We have used two algorithms:
   - watercontent.py
   - airconditions.py

   The sensor types are:
   - **air-condition-sensor** - For measuring the temperature of the soil
   - **soil-moisture-sensor** - For measuring the humidity of the sensor and turning off or on the sprinkler based on the threshold.

2. **Sample Application(Test use case).**
   A covid vaccination drive is going on at IIIT-H & to help people overcome the vaccine hesitancy IIIT-H is providing a transport facility using buses.College wants this transport experience to be comfortable, hassle free & want to span different areas of the city. Sensors used are GPS, biometric, temperature and light sensors.

   Use cases:
   1. Whenever someone boards a bus,he/she will do biometric check-in. Fare should be calculated based on distance multiplied by a fixed rate & should be displayed on the dashboard for that bus application instance (if your group does not have a dashboard, send a SMSnotification to guard) so that guard will collect that amount.
   2. When the bus is not empty, if the temperature is more than a threshold switch on the air conditions or lighting is lower than a lux level switch on the lights.
   3. Since college wants to span a maximum area, if more than two (>=3) buses come in a circle of given radius, except one send buzzer command to the rest. Run this service after every 2 minutes. (For the circle calculation part take suitable assumptions for connectivity).
   4. Also,as such services require proper coordination from administration in covid times, whenever a bus comes closer to a barricade by a threshold distance start/trigger an algorithm which sends an email to administration mentioning unique identifier of the bus& an email body notifying them.

3. Test cases

Sensor Module:

1.During sensor registration it will successfully be able to parse the sensorRegistrationConfig.json and sensor_instance.json file data from Platform manager and store the write information to sensor registry.
2. Topics for the sensor instance are created on registration or not.
3. For a given sensor type it will be able to filter out all valid sensor id based on registered sensors.
3. Will it be able to properly simulate the real time sensors .
4. Will the sensor module be able to provide the sensor data at the required rate and with the data type required by an algorithm.
5. Will it be able to trigger the sensors .

4. Requirements

- **Environment to be used:**
  - 64-bit OS (Linux)
  - Minimum RAM requirement: 4GB
  - Processor : intel Pentium i5 11th generation

- **Technologies to be used:**
  - Python framework is used to develop a platform
  - NFS: Network file sharing
  - Bash shell scripts for automation
  - Sensor simulator
  - MongoDB
  - Kafka, for communication between different modules

- **Interactions between modules:**
  All the modules are interacting with each other with the APIs provided by them. Each module has their respective set of APIs which can be pinged to get the necessary details. If we want any information we can http ping the API of that module.

- **Wire and file formats:**
  Configuration files-
  - SensorTypeRegistration.json
  - SensorInstanceRegistration.json

  API's-
  - getSensorTopic() api

5. Design and flow of Sensor Registration and Management Module

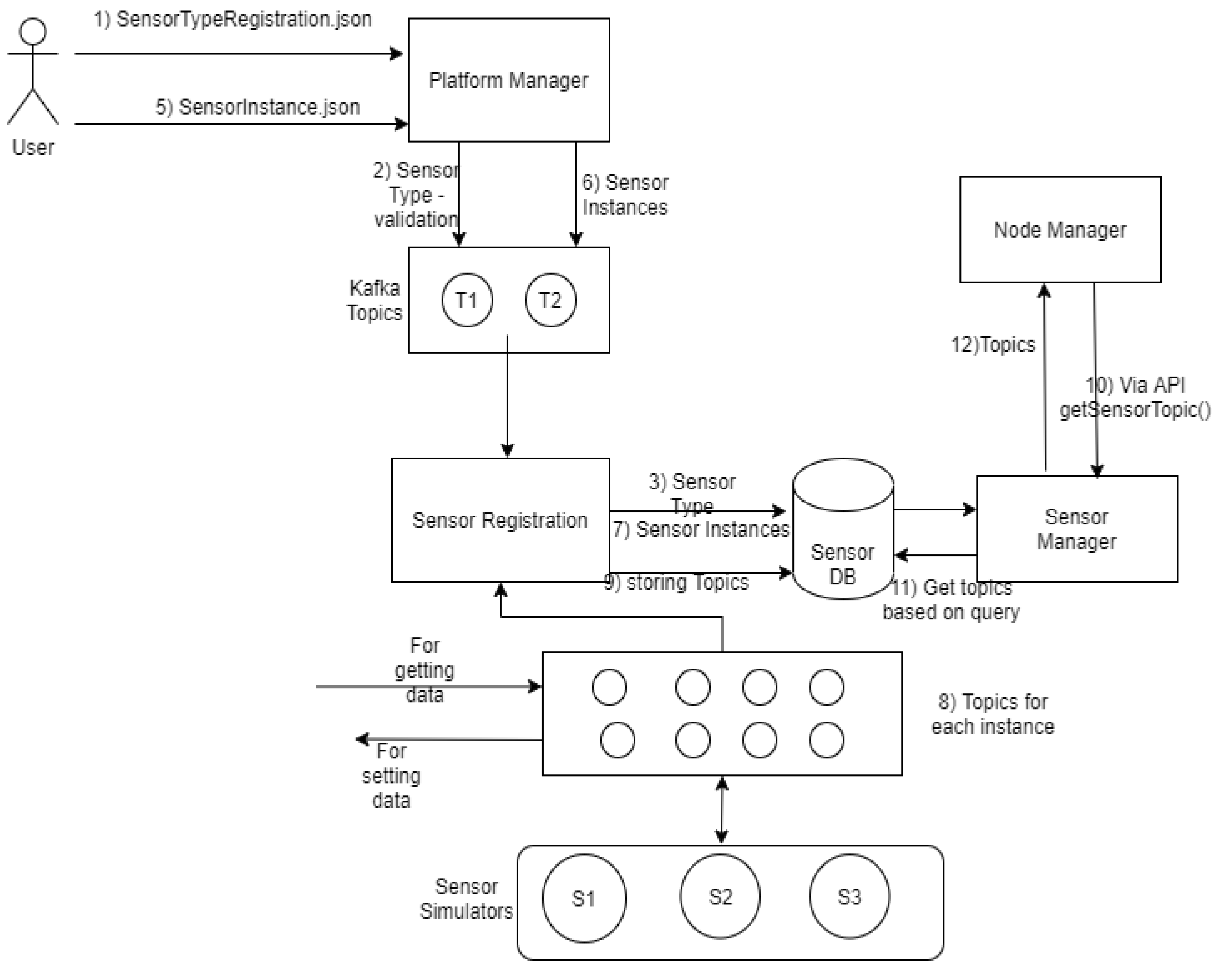
1. Lifecycle of the module

- With the platform initialization service, the modules like sensor registration will connect to platform manager .
- The Sensor manager and registration service will also startup with the initialization of the platform and establish a communication method with sensors.
- The sensor registry will be backed up after a fixed interval of time to monitor the sensor details.

2. List the sub modules-

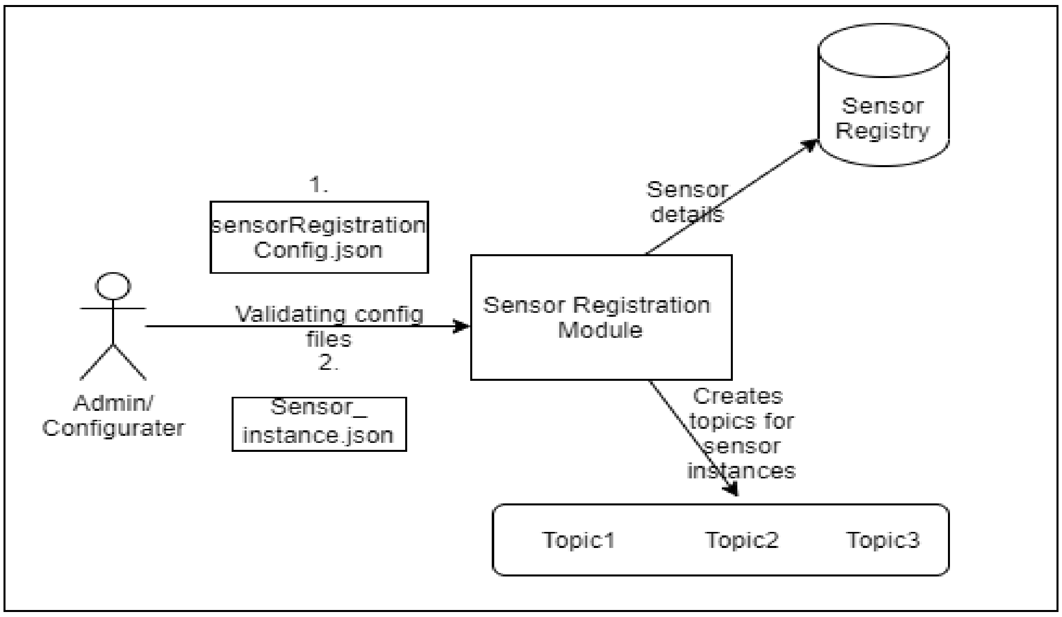
1. **Sensor Manager-**
   This sub-system connects with sensors and gets the raw data of streams. It processes the raw data into proper model input form and starts streaming it via the Message Queue which is consumed by the application on the server.
2. **Sensor Registration**- Whenever a new application is deployed in the platform, it also gives in the details of the sensors it needs. The detail about the sensor is provided in the form of a configuration file. Sensor Registration service parses this config file and stores the details about the sensors in the Sensor database. It further sets up the sensors and their gateways.

A block diagram

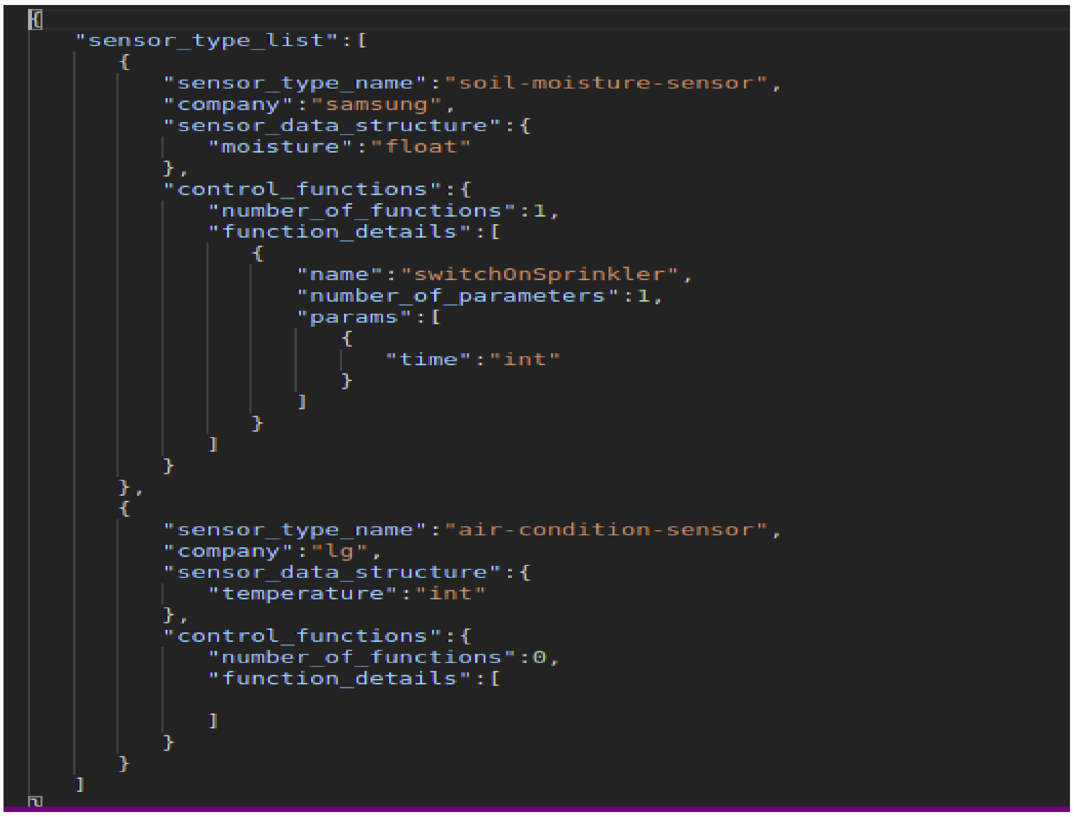


- **Step wise Sensor Registration and Management-**

1) SensorTypeRegistration-



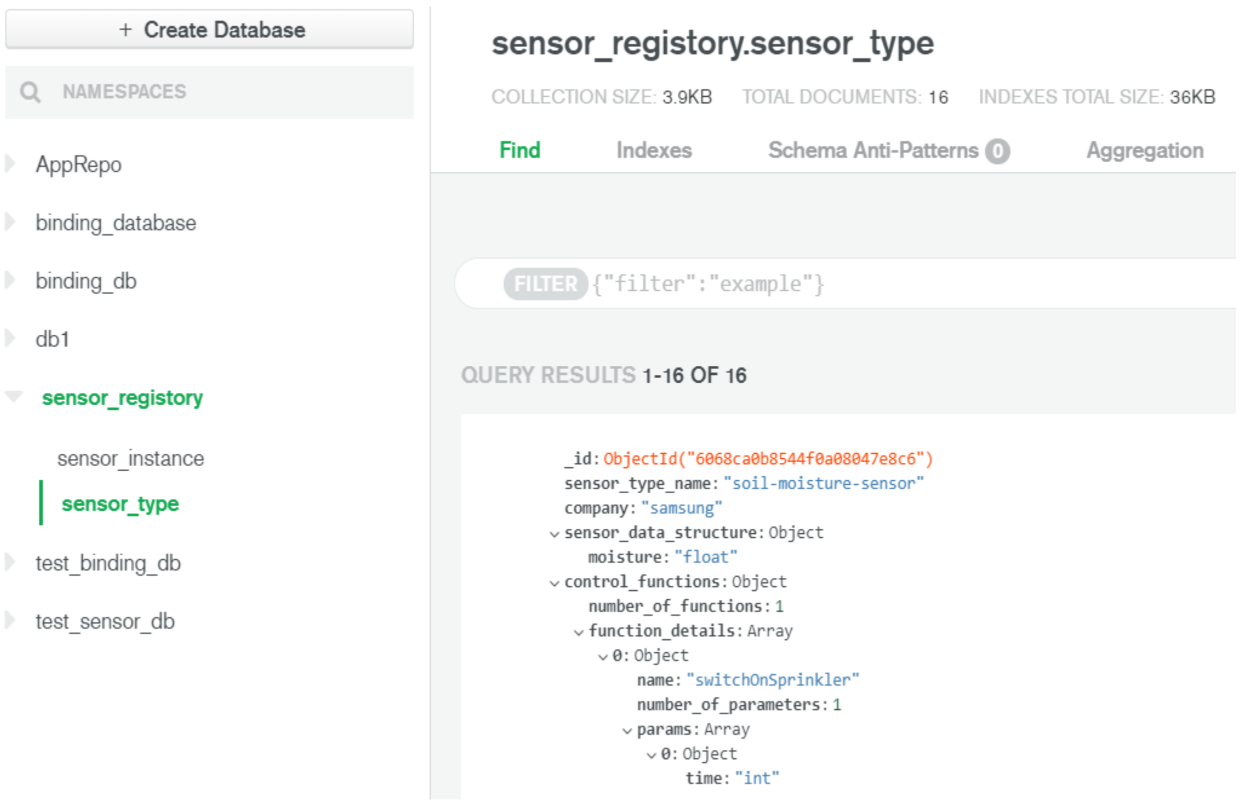
The Application Configurator registers any new sensor type that will be used by the application.He/she uploads a config file- SensorTypeRegistration.json, it is received from platform manager via a kafka topic in the sensor registration module. The config defines details like company,type of data produced, control functions,etc.



```
{
    "sensor_type_list":[
        {
            "sensor_type_name":"soil-moisture-sensor",
            "company":"samsung",
            "sensor_data_structure":{
                "moisture":"float"
            },
            "control_functions":{
                "number_of_functions":1,
                "function_details":[
                    {
                        "name":"switchOnSprinkler",
                        "number_of_parameters":1,
                        "params":[
                            {
                                "time":"int"
                            }
                        ]
                    }
                ]
            }
        },
        {
            "sensor_type_name":"air-condition-sensor",
            "company":"lg",
            "sensor_data_structure":{
                "temperature":"int"
            },
            "control_functions":{
                "number_of_functions":0,
                "function_details":[

                ]
            }
        }
    ]
}
```

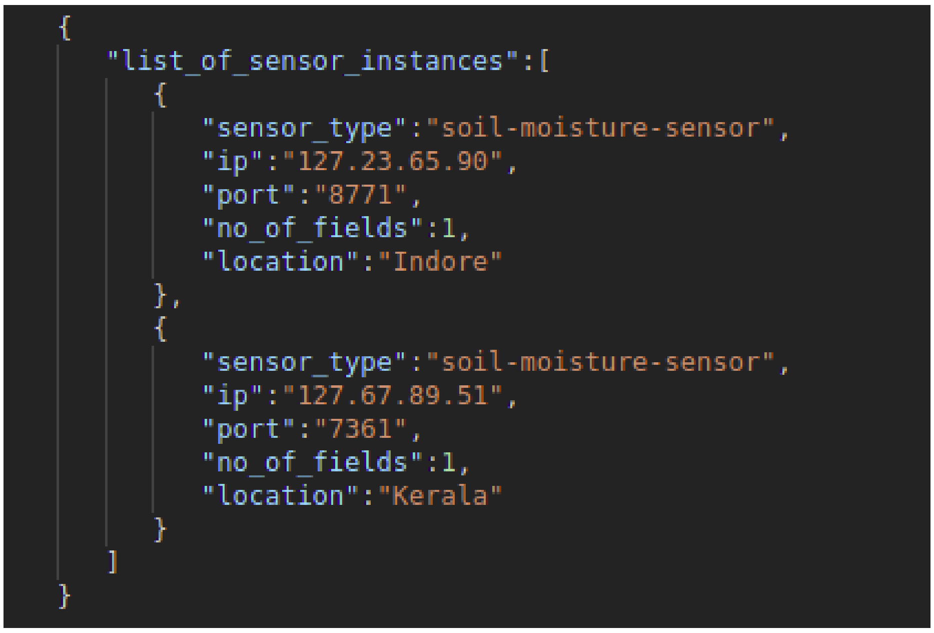
Once the platform manager validates the config file, sensor registration ,registers the type and saves the details in sensor_registory db inside collection sensor_type. This process gets the basic prototype of the sensors of the type.



```
_id: ObjectId("6068ca0b8544f0a08047e8c6")
sensor_type_name: "soil-moisture-sensor"
company: "samsung"
⌄sensor_data_structure: Object
   moisture: "float"
⌄control_functions: Object
   number_of_functions: 1
  ⌄function_details: Array
    ⌄0: Object
       name: "switchOnSprinkler"
       number_of_parameters: 1
      ⌄params: Array
        ⌄0: Object
           time: "int"
```

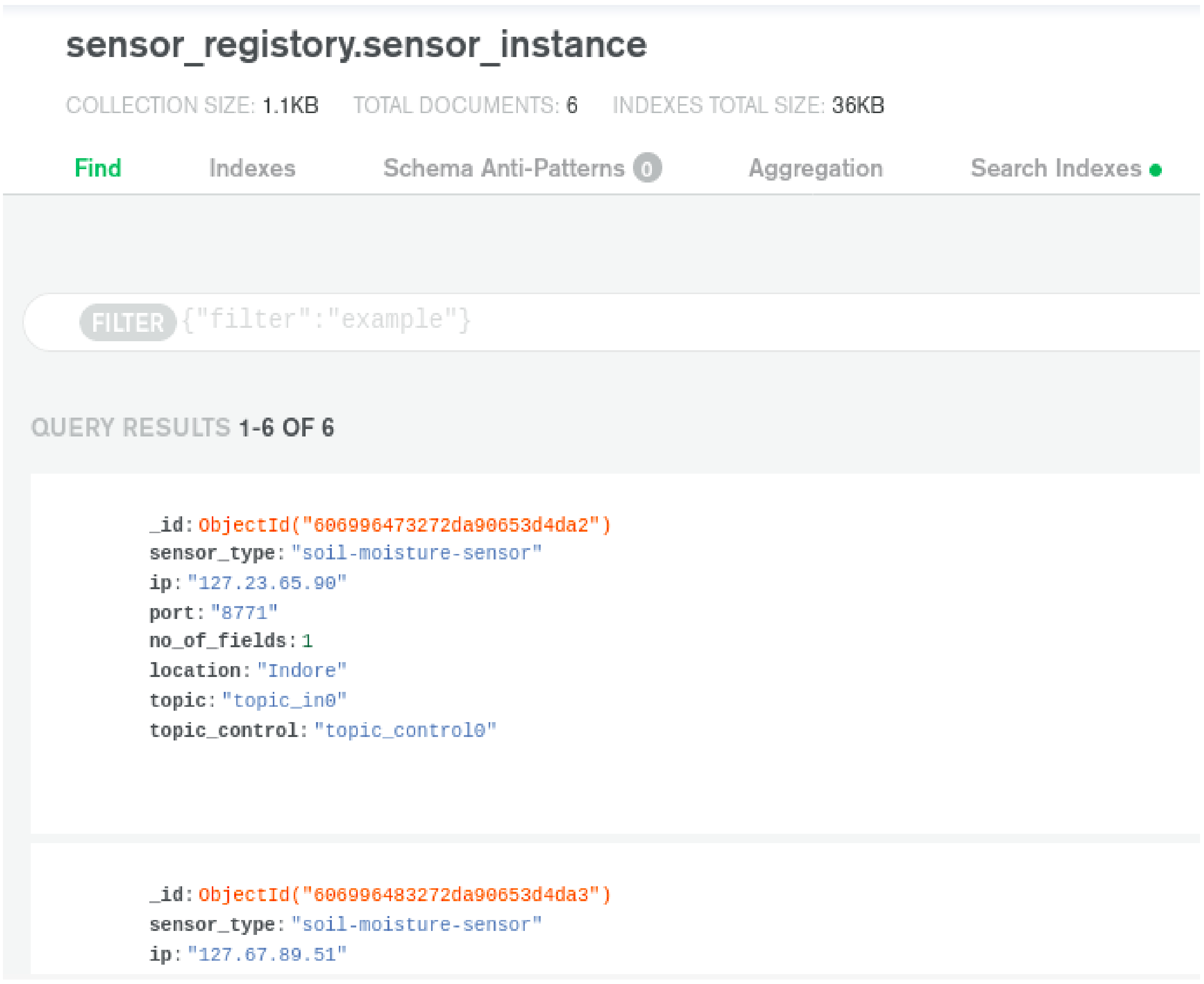
2) SensorInstance Registration-

The Application Configuration registers the instances of the sensors that will be actually used by the application for a particular instance while deploying the application. He/she uploads a config file-SensorInstance.json



```
{
    "list_of_sensor_instances":[
        {
            "sensor_type":"soil-moisture-sensor",
            "ip":"127.23.65.90",
            "port":"8771",
            "no_of_fields":1,
            "location":"Indore"
        },
        {
            "sensor_type":"soil-moisture-sensor",
            "ip":"127.67.89.51",
            "port":"7361",
            "no_of_fields":1,
            "location":"Kerala"
        }
    ]
}
```

The sensor registration saves the details of the instance and registers the instances. Then it gets topics for the instances and stores it in the sensor_instance collection of sensor registry database.


sensor_registory.sensor_instance

COLLECTION SIZE: 1.1KB TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS 1-6 OF 6

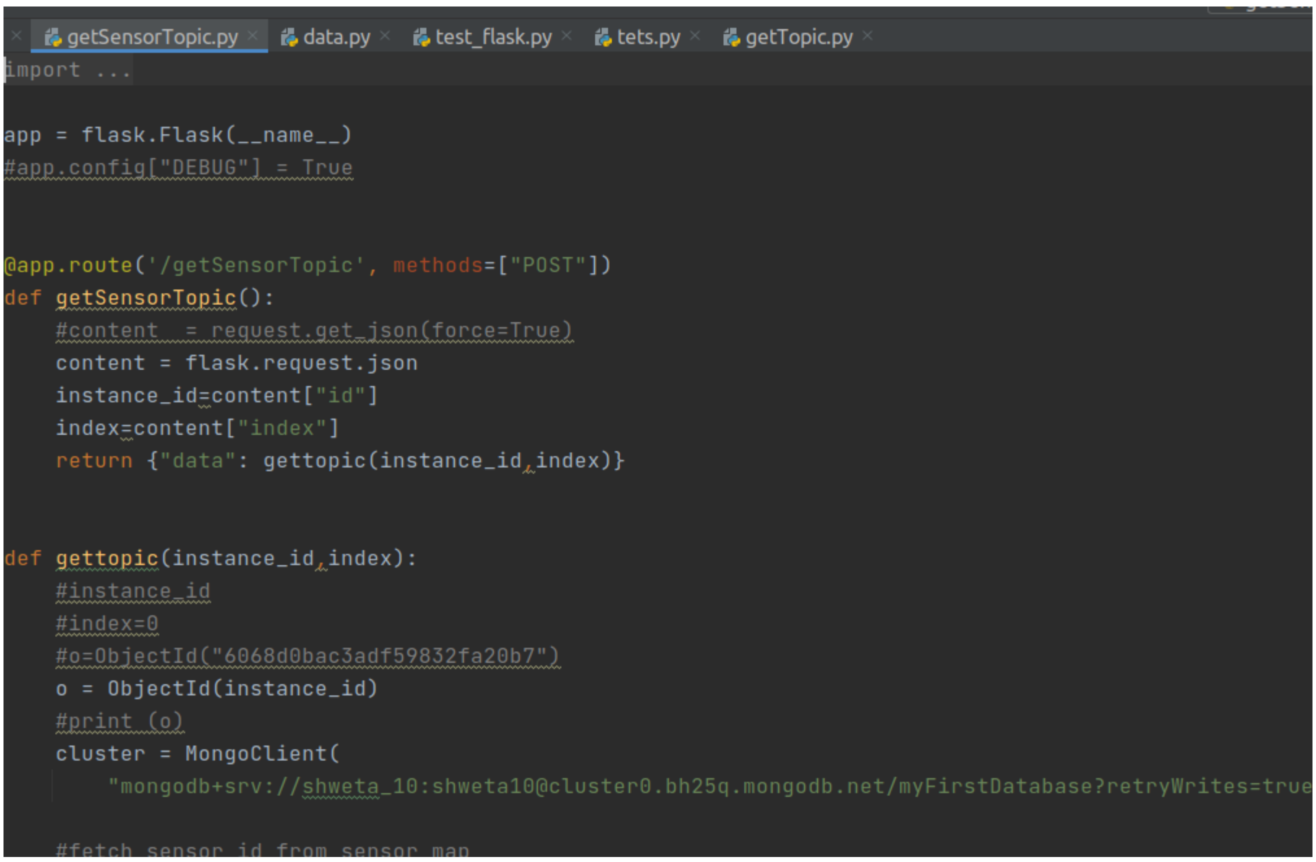
```
_id: ObjectId("606996473272da90653d4da2")
sensor_type: "soil-moisture-sensor"
ip: "127.23.65.90"
port: "8771"
no_of_fields: 1
location: "Indore"
topic: "topic_in0"
topic_control: "topic_control0"
```

```
_id: ObjectId("606996483272da90653d4da3")
sensor_type: "soil-moisture-sensor"
ip: "127.67.89.51"
```

For each instance of the sensor type , 2 topics are created , one for getting the data from that sensor instance and another for controlling the sensor.

3) **getSensorTopic() api** is created for the application developer to get the sensor topic for a particular algorithm and use it . platform_lib.py file indirectly calls this api via the GetKafkaTopic to consume the sensor data created on the topics.

The data binding with the algorithms based on location is done through this module. During the sensor binding process, first based on query of location, room number ,etc the sensors to be used by each algo is mapped in sensor_map collection. Now based on these the particular sensor topics are fetched from sensor_instance collection and returned through this Api.



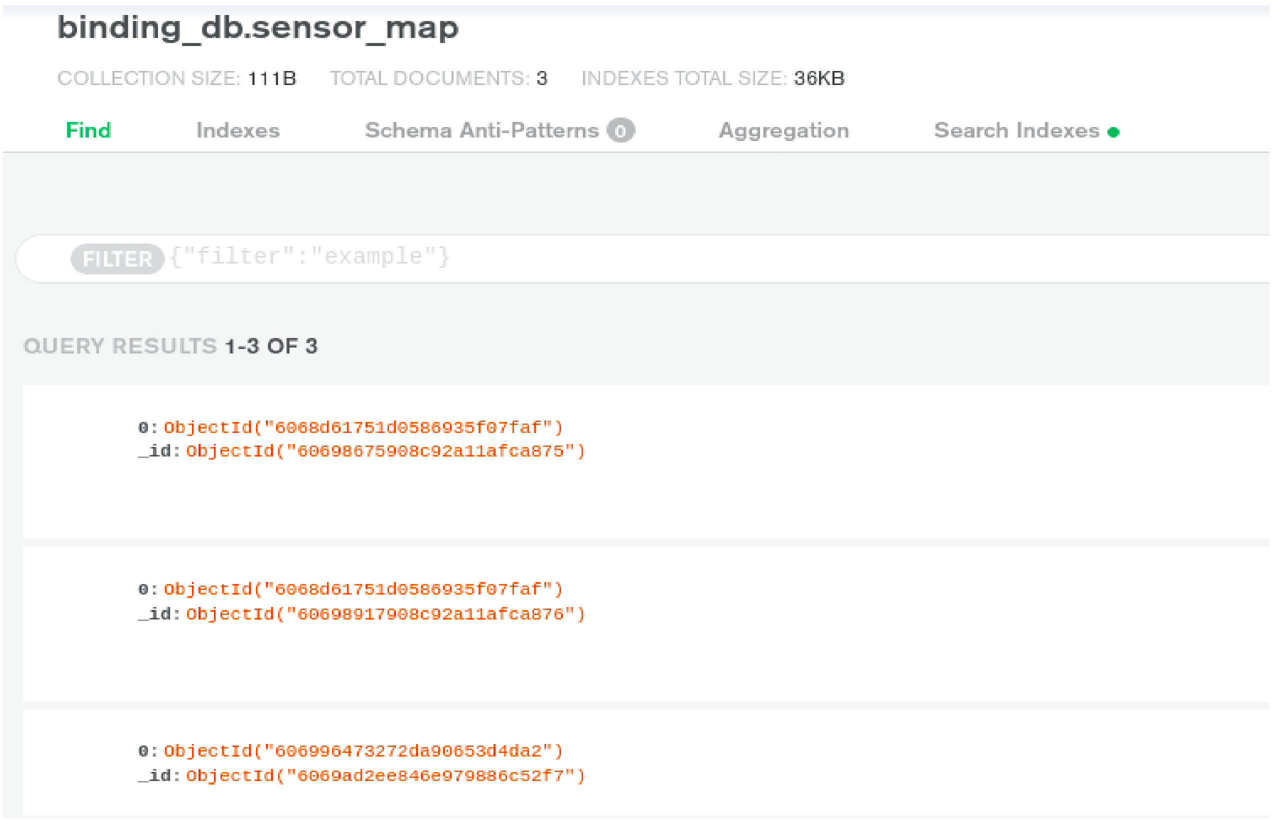
```
import ...

app = flask.Flask(__name__)
#app.config["DEBUG"] = True


@app.route('/getSensorTopic', methods=["POST"])
def getSensorTopic():
    #content  = request.get_json(force=True)
    content = flask.request.json
    instance_id=content["id"]
    index=content["index"]
    return {"data": gettopic(instance_id,index)}


def gettopic(instance_id,index):
    #instance_id
    #index=0
    #o=ObjectId("6068d0bac3adf59832fa20b7")
    o = ObjectId(instance_id)
    #print (o)
    cluster = MongoClient(
        "mongodb+srv://shweta_10:shweta10@cluster0.bh25q.mongodb.net/myFirstDatabase?retryWrites=true

    #fetch sensor id from sensor map
```



binding_db.sensor_map

COLLECTION SIZE: 111B   TOTAL DOCUMENTS: 3   INDEXES TOTAL SIZE: 36KB

{"filter":"example"}

QUERY RESULTS 1-3 OF 3

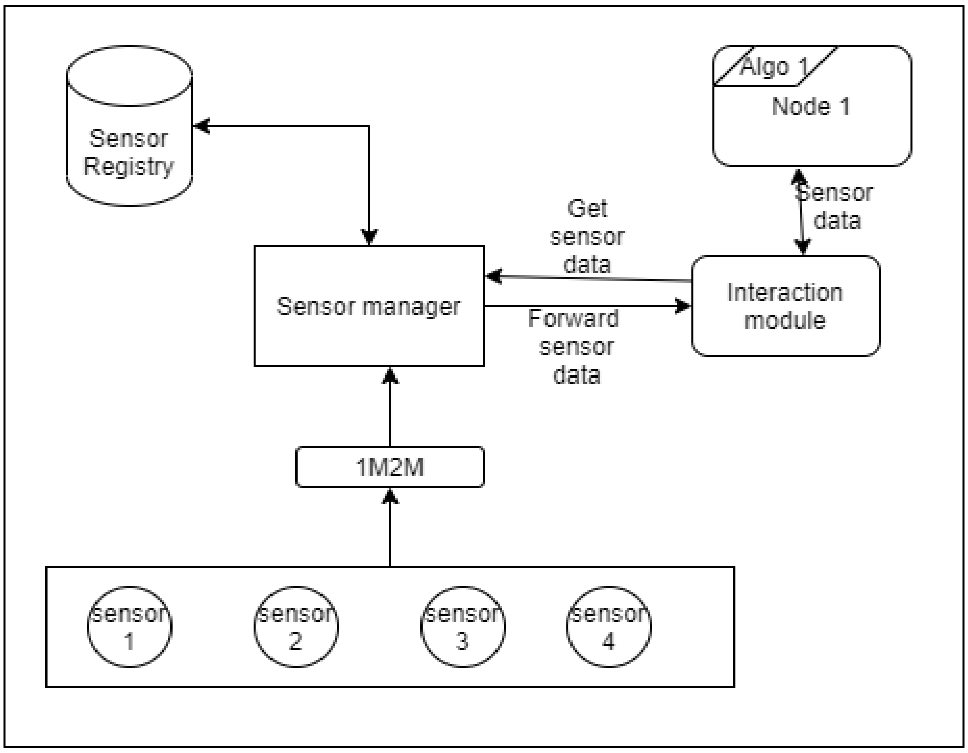
```
0: ObjectId("6068d61751d0586935f07faf")
_id: ObjectId("60698675908c92a11afca875")

0: ObjectId("6068d61751d0586935f07faf")
_id: ObjectId("60698917908c92a11afca876")

0: ObjectId("606996473272da90653d4da2")
_id: ObjectId("6069ad2ee846e979886c52f7")
```

6. Brief overview of each sub module

- **Sensor Manager -**



a) Sensor Data Processing:

- It processes the raw data into proper input form (as required by the application) , like the data unit expected and the data rate at which the application is expecting the sensor data.
- The interaction module takes care of placing processed sensor data on the topic of a particular application instance.

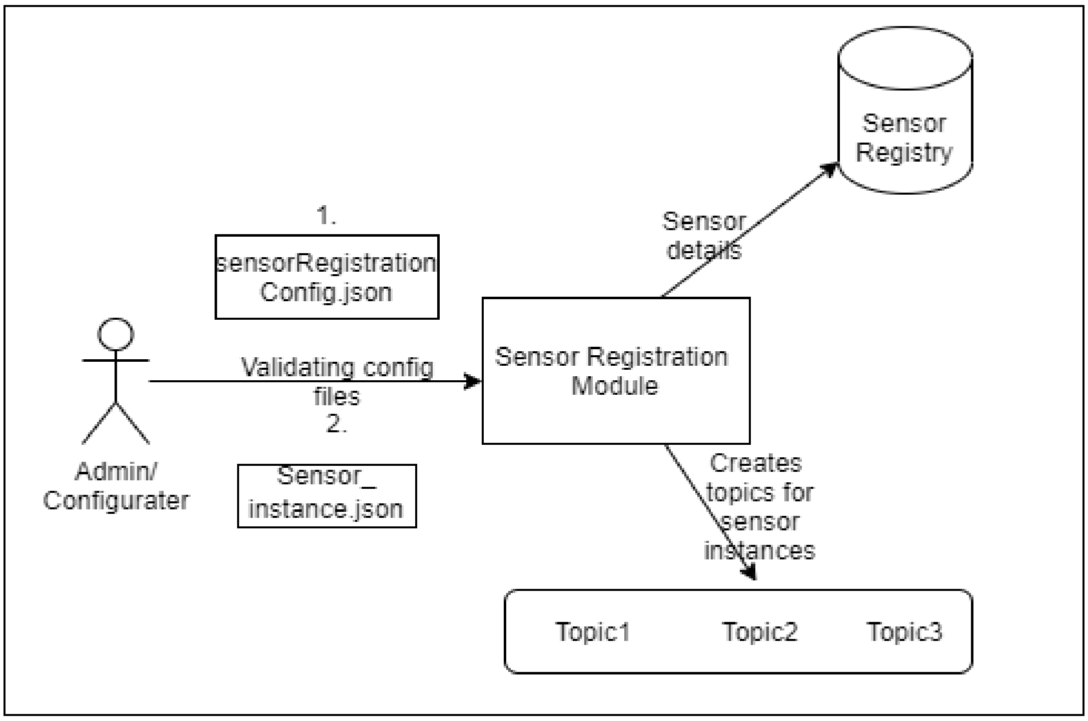
b) Binding Sensor Data:

- The first part is to validate the sensor against the sensor details available. Then identifying the algorithm which requires the sensor data. Then a temporary topic is created for the communication between the application and sensor.

c) Sending Sensor Data:

- Sensor manager upon getting sensor data pushes it to the respected topics. From where the application can pull the sensor data.
- Interaction module of the sensor manager plays an important role in further forwarding the sensor data to a particular instance of an application.

- **Sensor Registration-**



a) Parsing and verifying the Config file- sensorRegistrationConfig.json obtained from the application admin/Configurator to get the details of sensor types.

sensorRegistrationConfig.json

```
{
        “sensorTypeList”:
[
        {
                “Sensor type name”:”temp sensor”,
                “Company”: “samsung”,
                “Sensor_data_structure”:
{
        int temp,
        Int x
}
“controllerList”:
{
        “Func name”: setTemp
        “Input”: “int”
        “Return”: “bool”
}
}
]
}
```

SensoInstanceRegistration.json

```
{
        “Sensor-type”:”temp”,
        “IP” : “x.x.x.x”,
        “Port”: “x”,
        “No of fields of metadata”: “2”,
        “Key1” : “value1”,
        “Key2” : “value2”
}
```

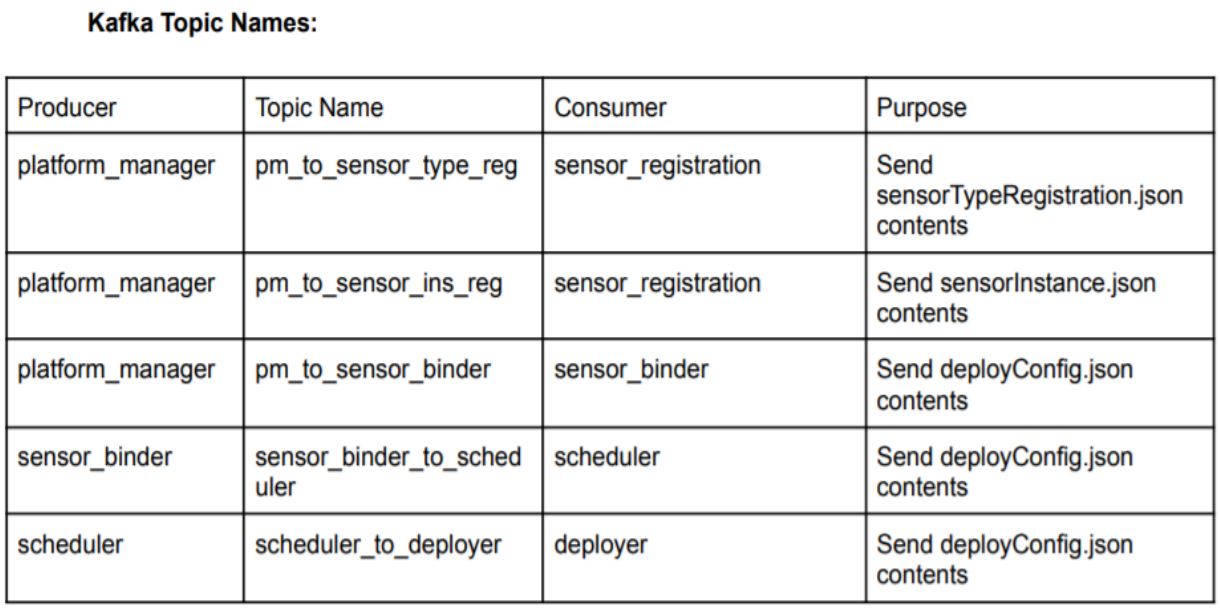
b) Now the sensor instance registration is done through the config file sensor_instance.json by the admin.
c) Registering the details of the sensor in sensor registry.
d) Setting up the new sensors with their gateways and topics.

7. Communication Model (using kafka only)

For communication we are using KAFKA producer-consumer architecture.
In our model we have used this communication at different levels regarding our module-
Between Platform manager and sensor registration-
2 topics are created , one for sending the sensor type registration config and another for instance config.
Platform manager acts as a producer and sensor registration acts as consumer over the topic.

For the sending data from sensors to sensor manager and sending data from topics to the application instance. Here also the kafka producer and consumers are created.

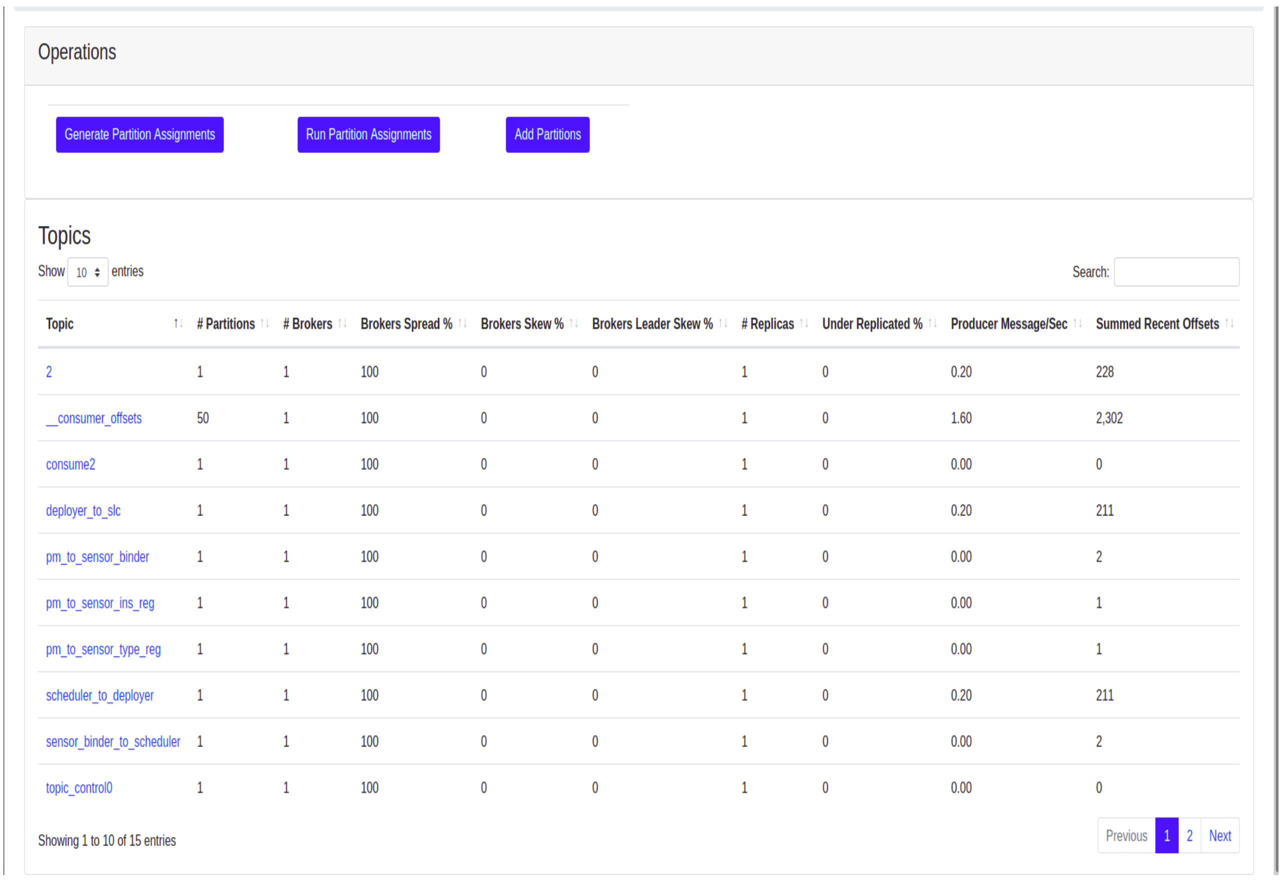
Different topics created-



Kafka Topic Names:

Producer	Topic Name	Consumer	Purpose
platform_manager	pm_to_sensor_type_reg	sensor_registration	Send sensorTypeRegistration.json contents
platform_manager	pm_to_sensor_ins_reg	sensor_registration	Send sensorInstance.json contents
platform_manager	pm_to_sensor_binder	sensor_binder	Send deployConfig.json contents
sensor_binder	sensor_binder_to_scheduler	scheduler	Send deployConfig.json contents
scheduler	scheduler_to_deployer	deployer	Send deployConfig.json contents

View in the kafka manager-



Operations

Topics

Show 10 entries Search:

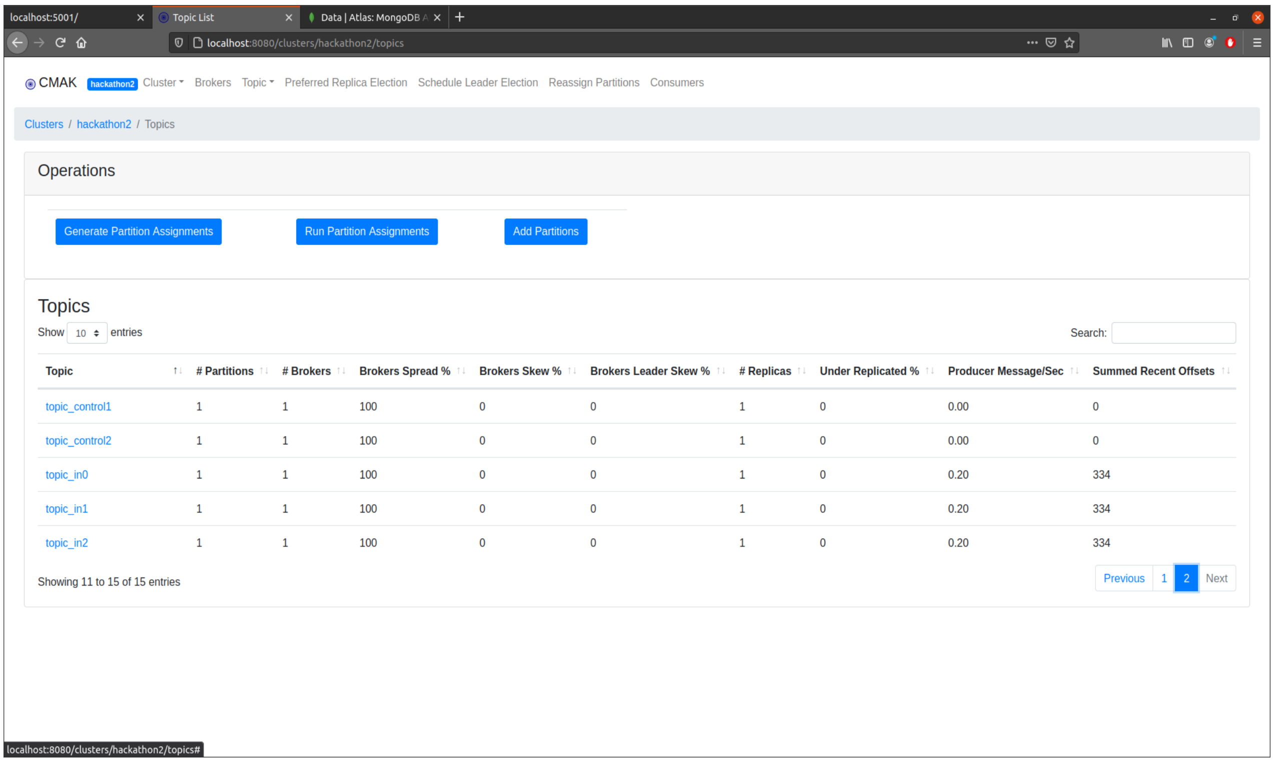
Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas	Under Replicated %	Producer Message/Sec	Summed Recent Offsets
2	1	1	100	0	0	1	0	0.20	228
__consumer_offsets	50	1	100	0	0	1	0	1.60	2,302
consume2	1	1	100	0	0	1	0	0.00	0
deployer_to_slc	1	1	100	0	0	1	0	0.20	211
pm_to_sensor_binder	1	1	100	0	0	1	0	0.00	2
pm_to_sensor_ins_reg	1	1	100	0	0	1	0	0.00	1
pm_to_sensor_type_reg	1	1	100	0	0	1	0	0.00	1
scheduler_to_deployer	1	1	100	0	0	1	0	0.20	211
sensor_binder_to_scheduler	1	1	100	0	0	1	0	0.00	2
topic_control0	1	1	100	0	0	1	0	0.00	0

Showing 1 to 10 of 15 entries

Topics for each sensor instances-

Topic_in0- getting data from the instance

Topic_control0- setting/controlling data from the instance



localhost:5001/
Topic List
Data | Atlas: MongoDB
localhost:8080/clusters/hackathon2/topics
CMAK
hackathon2
Cluster
Brokers
Topic
Preferred Replica Election
Schedule Leader Election
Reassign Partitions
Consumers
Clusters / hackathon2 / Topics

Operations

Generate Partition Assignments
Run Partition Assignments
Add Partitions

Topics

Show 10 entries Search:

Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas	Under Replicated %	Producer Message/Sec	Summed Recent Offsets
topic_control1	1	1	100	0	0	1	0	0.00	0
topic_control2	1	1	100	0	0	1	0	0.00	0
topic_in0	1	1	100	0	0	1	0	0.20	334
topic_in1	1	1	100	0	0	1	0	0.20	334
topic_in2	1	1	100	0	0	1	0	0.20	334

Showing 11 to 15 of 15 entries

8. Interactions between sub modules

1. Sensor registration receives the config files via the platform manager whenever a sensor is registered.
2. The registration process also involves registering the details of the sensors into the sensor registry and creating topics for them.
3. The sensor manager utilizes the data stored in the sensor registry for the real time binding of the sensor to the algorithms.
4. The interaction module is part of a sensor manager which helps in handling requests from an application and providing the sensor data for the algorithm as processed by the sensor manager.
5. The notification module identifies the notification mechanism requested by the application like SMS, email ,etc and reports the end user with the outputs associated with an application instance.
6. The control module searches the application repository for the action algorithm associated with a sensor based on output of an application and runs the algorithm on the sensors to control it.

9. Interactions between other modules

1. Platform manager and Sensor Registration- The config files received from from the admin/configurator is forwarded via api from platform manager to the sensor registration to validate and register the sensor types and instances.
2. Sensor Manager and Node(with an application instance running) , the application forwards a request via api of a sensor data it requires , so the sensor manager gets the sensor data , binds it and forwards the sensor topics to the interaction module and it sends the data to the application.
3. Control/Notification manager and Node(with an application instance running) , the output generated by the application is given to the notification manager to notify the end users or trigger actions on the sensors.