FINAL

PROJECT REQUIREMENT

PREPARED FOR

IAS PROJECT - GROUP 4

IIIT-Hyderabad

PREPARED BY

TEAM 1

Varun Nambigari

Raman Shukla

Akshay M

TEAM 2

Devesh Jha

Amisha Bansal

Parul Ansal

TEAM 3

Dhruv Sachdev

Nisarg Sheth

Pujan Ghelani

TEAM 4

Akshay Choudhary

Shweta Arya

1) Overview

1.1) Introduction

The goal of this distributed IoT platform is to provide all the generic functionality for the application so developers can focus on building features that differentiate products and add value for them.This project constitutes building an IOT platform. Which provides functionalities to develop an application involving interaction with multiple sensors, analyzing the data from the sensors and taking action on the sensors.

1.2) Scope

- Any application which uses the above sensor types and control types can be deployed on this platform.
- Example: Farm Management application (Using Humidity sensor and sprinkler controller)
- Example: control the on or off of the sprinkler.
- Provides application developer easy functions for getting data from each sensor.
- Example. temp.getCurrTemp() (here temp is a sensor type defined in platform)
- It provides the application developer with a UI for uploading and running applications.
- The sensor data is collected by the platform using the sensor interface.
- The platform and the application deployed is fault tolerant, scalable, load balanced.

2) Intended Use

2.1) Intended Use

The platforms intended use is to provide a support to the development and, deployment and running of IOT based applications. The platform provides a application model which can be followed by an application to build and run it on the platform with no worries about the sensor connections.

They can directly register the sensor types needed by the application and add the instances as per the deployment of the application on different scales. The platform also provides API to be directly used in the applications to use the data from the sensors and control the sensors.

2.2) Assumptions and dependencies

- We assume that the application developer and Configurator provide the correct format config files andmin applications as per the model provided by the platform.
- The sensors are simulated in our platform for testing the application.

Dependencies-

Python framework is used to develop a platform

- NFS: Network file sharing
- Bash shell scripts for automation
- MongoDB

- Kafka, for communication between different modules
- Docker Container

3) System Features and Requirements

3.1) Platform Requirements

b) Different actors on the platform

Application Developer:

- Write the algorithms by getting and setting data from the sensor instances provided by the platform.
- Write a proper config file.

Application Configurator:

- Register the sensors(which sensor has what type → predefined by app developer) on which algorithms are supposed to run.
- Provide a config file (deployConfig) to the platform containing when to deploy the algorithms and on which sensors.

End User:

- Interacts with the application UI which is provided by application developers.
- Will receive notification from the action server.
- May trigger a particular action on a sensor.

c) Applications overview on the platform

Sample applications supported-

app.zip file-

- **app**
    - **Algo1.py**
    - **Algo2.py**
- **appConfig.json**

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.

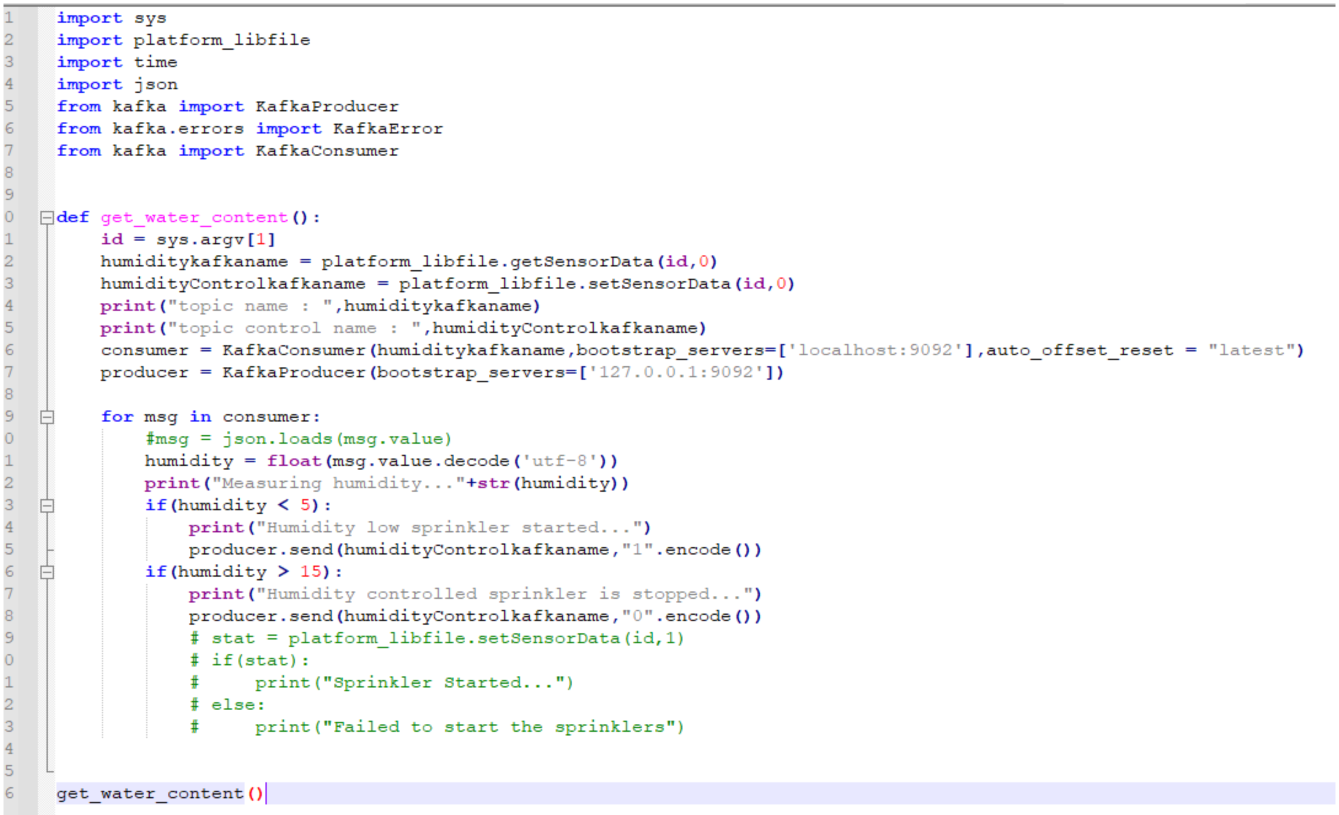
   We have used two algorithms.

   watercontent.py, airconditions.py

   The sensor types -

   air-condition-sensor - measuring the temperature of the soil
   soil-moisture-sensor - measuring the humidity of the sensor and turning off or on the sprinkler based on the threshold.

watercontent.py-



```
import sys
import platform_libfile
import time
import json
from kafka import KafkaProducer
from kafka.errors import KafkaError
from kafka import KafkaConsumer


def get_water_content():
    id = sys.argv[1]
    humiditykafkaname = platform_libfile.getSensorData(id,0)
    humidityControlkafkaname = platform_libfile.setSensorData(id,0)
    print("topic name : ",humiditykafkaname)
    print("topic control name : ",humidityControlkafkaname)
    consumer = KafkaConsumer(humiditykafkaname,bootstrap_servers=['localhost:9092'],auto_offset_reset = "latest")
    producer = KafkaProducer(bootstrap_servers=['127.0.0.1:9092'])

    for msg in consumer:
        #msg = json.loads(msg.value)
        humidity = float(msg.value.decode('utf-8'))
        print("Measuring humidity..."+str(humidity))
        if(humidity < 5):
            print("Humidity low sprinkler started...")
            producer.send(humidityControlkafkaname,"1".encode())
        if(humidity > 15):
            print("Humidity controlled sprinkler is stopped...")
            producer.send(humidityControlkafkaname,"0".encode())
            # stat = platform_libfile.setSensorData(id,1)
            # if(stat):
            #     print("Sprinkler Started...")
            # else:
            #     print("Failed to start the sprinklers")


get_water_content()
```

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 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 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 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 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 requires 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. **Waste Management** : optimising waste pick up trucks routes and giving sanitation workers

insight into the actual fill level of various disposal units.

4. **Smart Home :** Managing home appliances, theft detection using automation of all its embedded technology. It defines a residence that has appliances, lighting, heating, air conditioning, TVs, computers, entertainment systems, big home appliances such as washers/dryers and refrigerators/freezers, security and camera systems capable of communicating with each other and being controlled remotely by a time schedule, phone, mobile or internet. All sensors are connected to a central hub controlled by the user using a wall-mounted terminal or mobile unit connected to internet cloud services.
5. **Traffic Monitoring** : Detecting vehicle number automatically through cameras for e-challan and monitoring traffic.

3.2) Functional Requirements

a) Registering sensors

- Registering the details of the sensor in the platform repository.
- Setting up the new sensors with their gateways.

b) Interaction with IoT sensors

- Creating temporary topics to be used by the Applications for getting sensor data.
- The applications use API to get the topic names from the sensor manager and use it to get and set data from the sensors.
- Processing of the sensor data to be provided in the format required by the applications.

c) Development of application on the platform
d) Identification of sensors for data binding
e) Data Binding to the application

- Data binding to the application is done by the sensor manager.

f) Scheduling on the platform
g) Acceptance of scheduling configuration
h) Starting and Stopping services
i) Communication model
j) Server and service life cycle
k) Deployment of application on the platform
l) Registry & repository) Load Balancing
n) Interactions between modules
o) Packaging details
p) Configuration files details
q) Interaction of different actors with the platform

3.3) Non-Functional Requirements

a) Fault tolerance

a.a) Platform

The platform is fault tolerant. Even if a service stops working, it will immediately be deployed again. So there won't be any discontinuation in any service of the platform.

a.b) Application

If any applications were deployed on the server and server crashes, monitoring module will detect it and after the server is restarted it will re-run all the applications that were running at the time of crash.

c) Accessibility of data

c.a) Application

c.b) Sensors

Sensors are connected to sensor manager using REST API, and all the other services get the sensor data via sensor manager using kafka.

d) Specification about application

e) UI and CLI for interaction

Dashboard can be used for interaction.

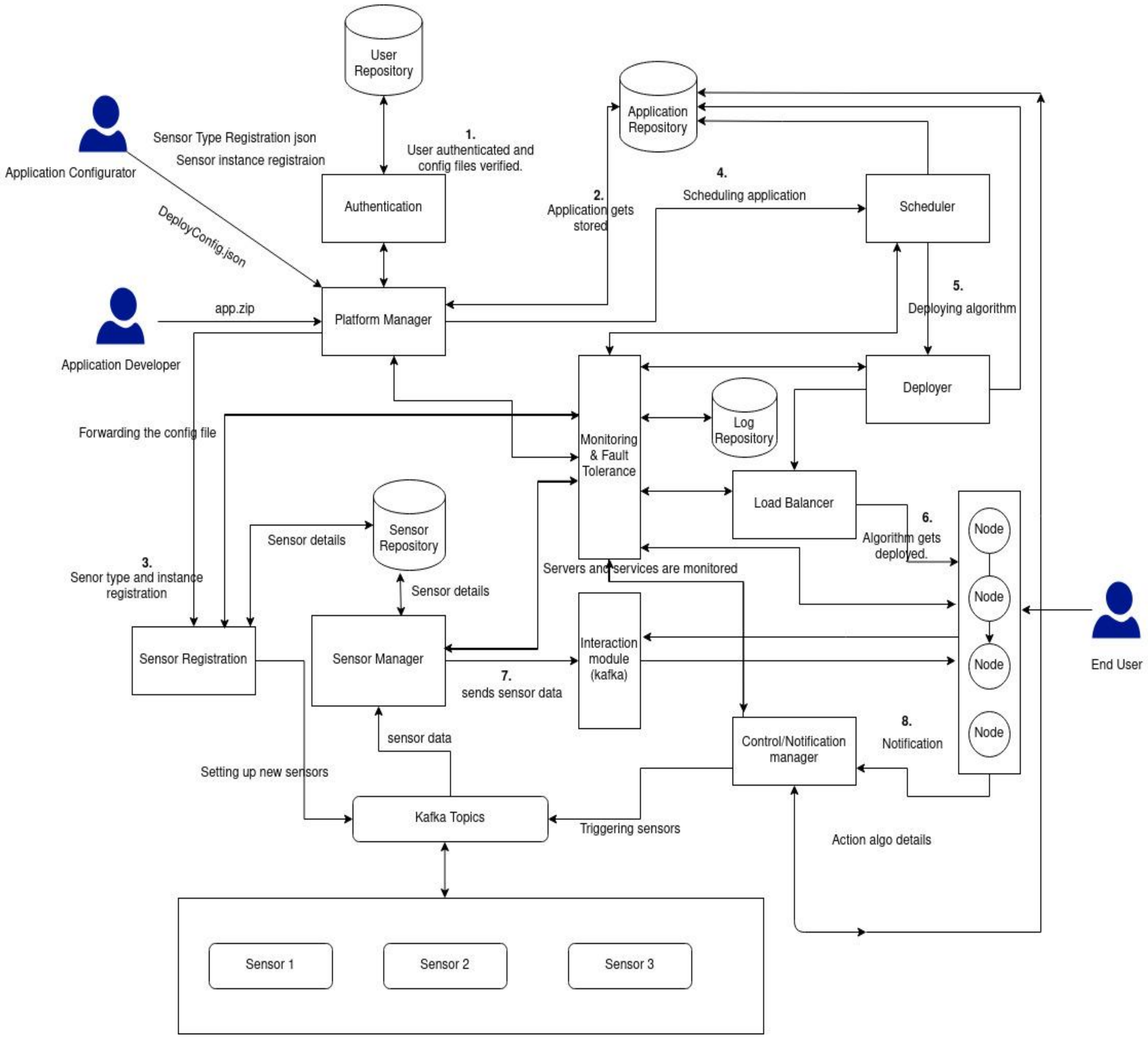
- The platform manager provides APIs for the dashboard.
- The dashboard shows the the following:
  - The logs from the running applications.
  - The status of various containers.
  - The logs of various containers.

g) Persistence

- In case a node crashes, the system brings it back in the last state on some other node.
- In case any system or application service or running model crashes then it will be brought back by monitoring service.

4) List the key functions

a) A block diagram listing all major components (~components)



b) Brief description of each component

Platform repo:

- Contains the docker images of all the services of the platform.
- The deployPlatform.sh script will take the config file containing the details of the nodes on which we need to deploy the platform and run all the docker images.

Platform Manager:

- Provides UI, uploading as well as deployment of applications.

Authentication Service:

- This module is used to authenticate the actor using the user repository. The actor will provide login details through platform manager UI.

Application Repository:

- It will contain code and config files of the application of the user.

Scheduler:

- Scheduler receives the application ID and path to configuration file in the application repository from the platform manager.
- Sends a request start/stop an application to the deployer according to the requests in the configuration file.

Deployer:

- Receives requests to start/stop an application from the scheduler.
- Sends requests to Load Balancer to ask for a certain number of nodes for an application.
- Sets up the environment and packages on the nodes as per the requirements and deploys the application.

Load Balancer:

- Receives information about load on the computing instances from the monitoring module.
- Takes this information as input to the load balancing algorithm and allocates the required number of nodes with minimum load to the deployer.

Server Lifecycle

- Server lifecycle will be responsible for monitoring the status of server's whether they are running or not. And will be responsible for restarting server and apps.

Service Lifecycle

- Service lifecycle will be responsible for monitoring the status of platform services whether they are running or not. And will be responsible for restarting services in case it stopped..

Sensor Manager:

- This module connects with sensors and gets the raw data of streams. It processes the raw data into proper model input form and streams it via the MessageQueue which is consumed by the application on the server.

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.

Sensor Database:

- This database stores the details about all the sensors present in the platform, which is accessed by the sensor manager to bind the sensors to the requesting application running on a node.

Action Server:

- This module is responsible for doing some action or notifying user based on the User's given action condition. User gives the action file while deployment itself as part of the zipped packaged file. The action file is present in application repository from where it can be used to act(on the sensors) and notify accordingly.

c) List the 4 major parts-

1) Platform Deployment and UI(Platform manager)
2) Deployment of the Application
3) Server & Service Lifecycle
4) Sensor and Notification

5) Use cases

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 -
   air-condition-sensor - measuring the temperature of the soil
   soil-moisture-sensor - measuring the humidity of the sensor and turning off or on the sprinkler based on the threshold.

2. **Waste Management** : optimising waste pick up trucks routes and giving sanitation workers insight into the actual fill level of various disposal units.
3. **Smart Home :** Managing home appliances, theft detection using automation of all its embedded technology. It defines a residence that has appliances, lighting, heating, air conditioning, TVs, computers, entertainment systems, big home appliances such as washers/dryers and refrigerators/freezers, security and camera systems capable of communicating with each other and being controlled remotely by a time schedule, phone, mobile or internet. All sensors are connected to a central hub controlled by the user using a wall-mounted terminal or mobile unit connected to internet cloud services.
4. **Traffic Monitoring** : Detecting vehicle number automatically through cameras for e-challan and monitoring traffic.

6) Primary test case for the project (that you will use to test)

a) Name of use case:

An application to facilitate transportation (using buses) needed for COVID vaccination drive.

b) Domain/company/environment where this use case occurs:

Transportation and Logistics

c) Description of the use case (purpose, interactions and what will the users benefit):

- Whenever someone boards a bus, he/she will do biometric check-in. Fare is calculated based on distance multiplied by a fixed rate & is displayed on the dashboard for that bus application instance so that guard can collect that amount.
- When the bus is not empty, if the temperature is more than a threshold air conditioner is switched on, or if lighting is lower than a lux level then lights are switched on.
- If more than two (>=3) buses come in a circle of given radius, except one, buzzer command to all buses except onet. Run this service after every 2 minutes.
- Whenever a bus comes closer to a barricade by a threshold distance, an algorithm is triggered which sends an email to the administration mentioning the unique identifier of the bus.

f) How is location and sensory information used

There are different sensors for different uses:

○ **GPS**: Sends placeholder-id & co-ordinates. Each bus is installed with one such sensor, one sensor is at IIIT-H campus and each police barricade has one such sensor.
○ **Biometric**: Sends placeholder-id & person-id(it will be unique on place-id level).
○ **Temperature**: Sends current temperature of the place
○ **Light Sensors**: Sends Lux Level of the place

g) Processing logic on the backend

- We are making 4 files for the use cases 1,2 and 4.
- For every instance of the buses, these 4 files will also be instantiated.
- The 5th file is a common file which will only be instantiated once for all the buses.
- Thus the 4 files are the local application files and the 5th file is the global application file.
- The local files are Fare Calculation, Light Control,Temperature Control and the Barricade alert.
- The global file is the radius buzzer when more than 2 buses are close enough to be in a boundary of a threshold radius.

h) User's UI view- what is their UI, where and all will they do with this systems

The logs for all the buses are visible on the dashboard.

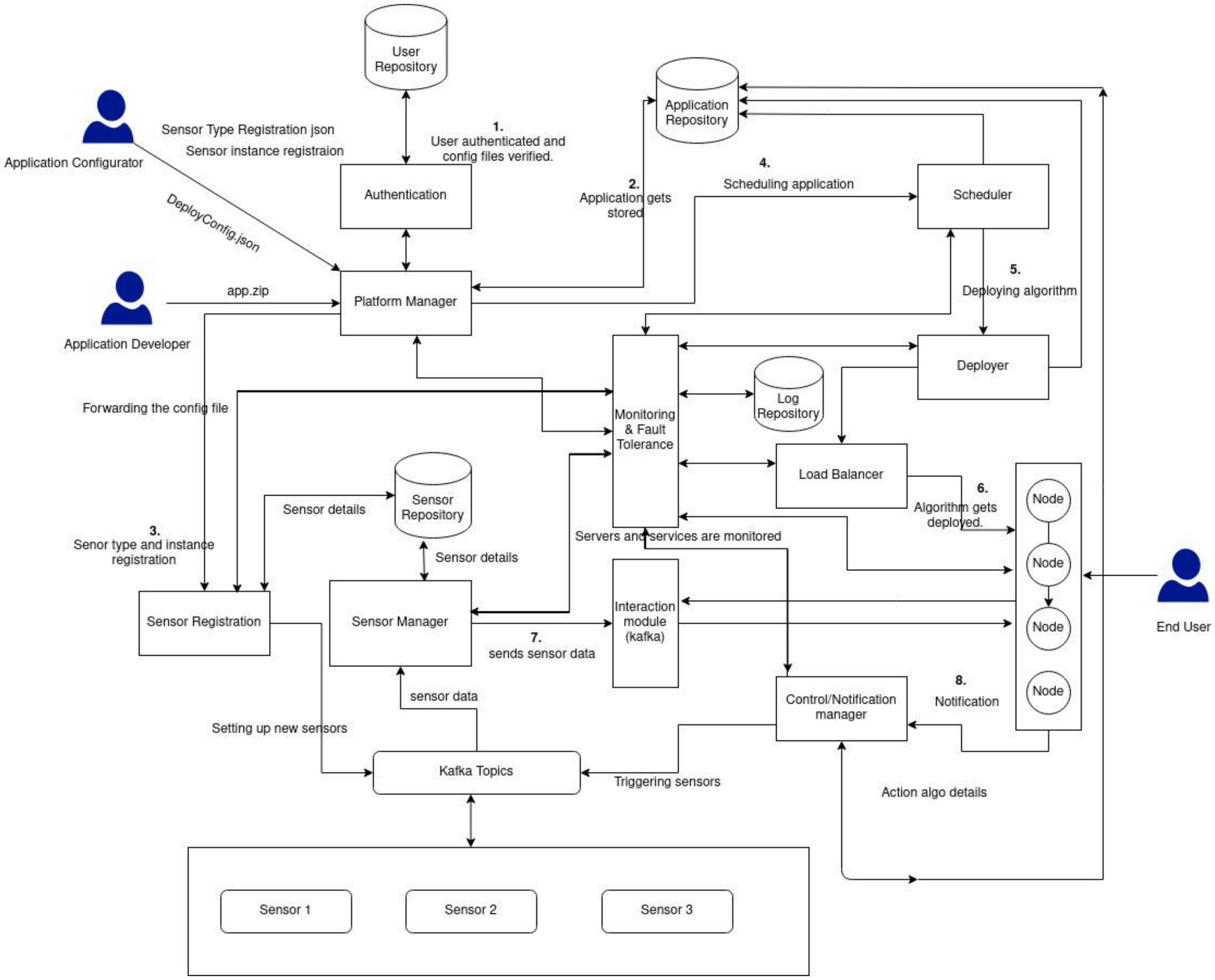
7) Subsystems

a) Key subsystems in the project

- Platform Deployment , UI(Platform manager) and Notification
- Deployment of the Application

- Server & Service Lifecycle
- Sensor Registration and Management

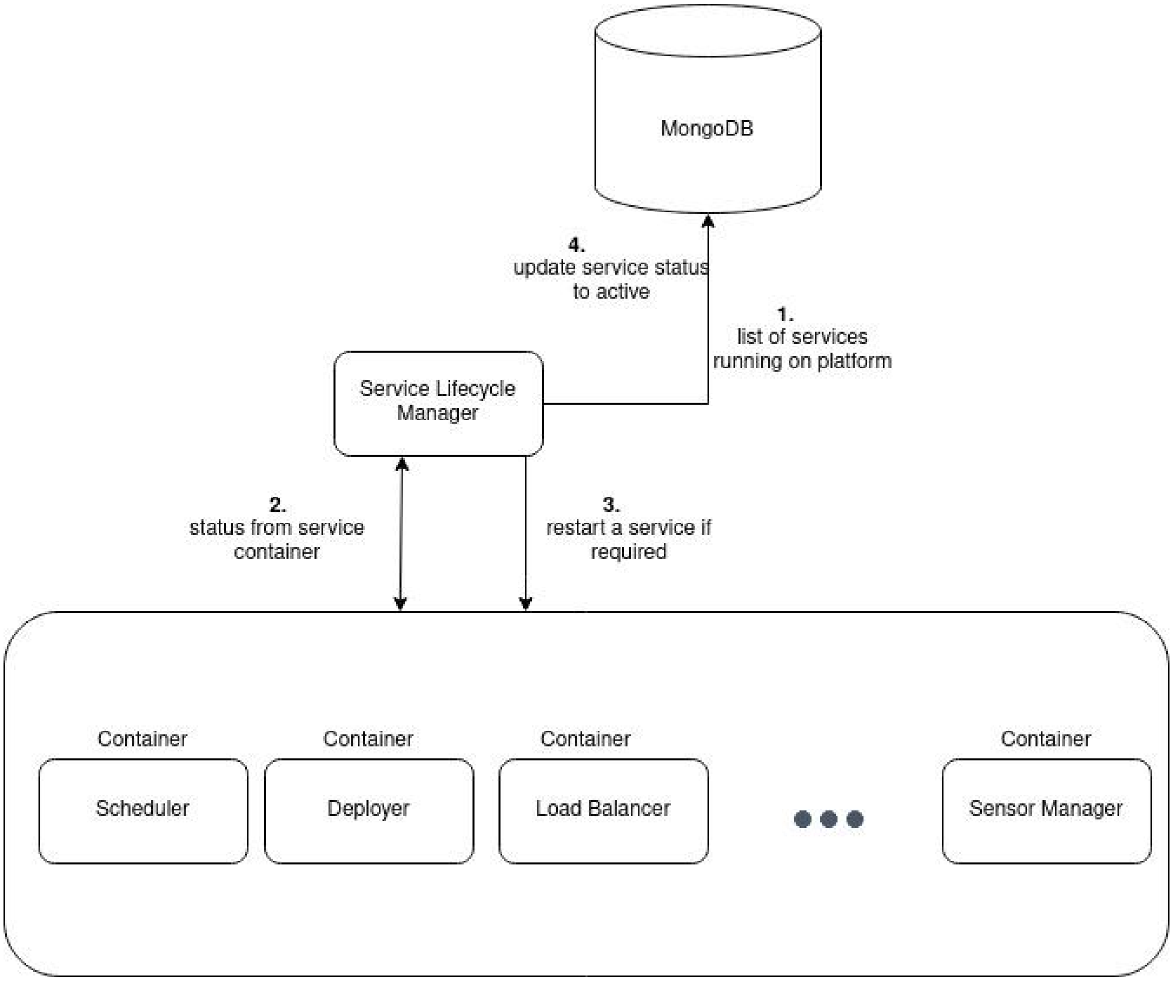
b) A block diagram of all subsystems



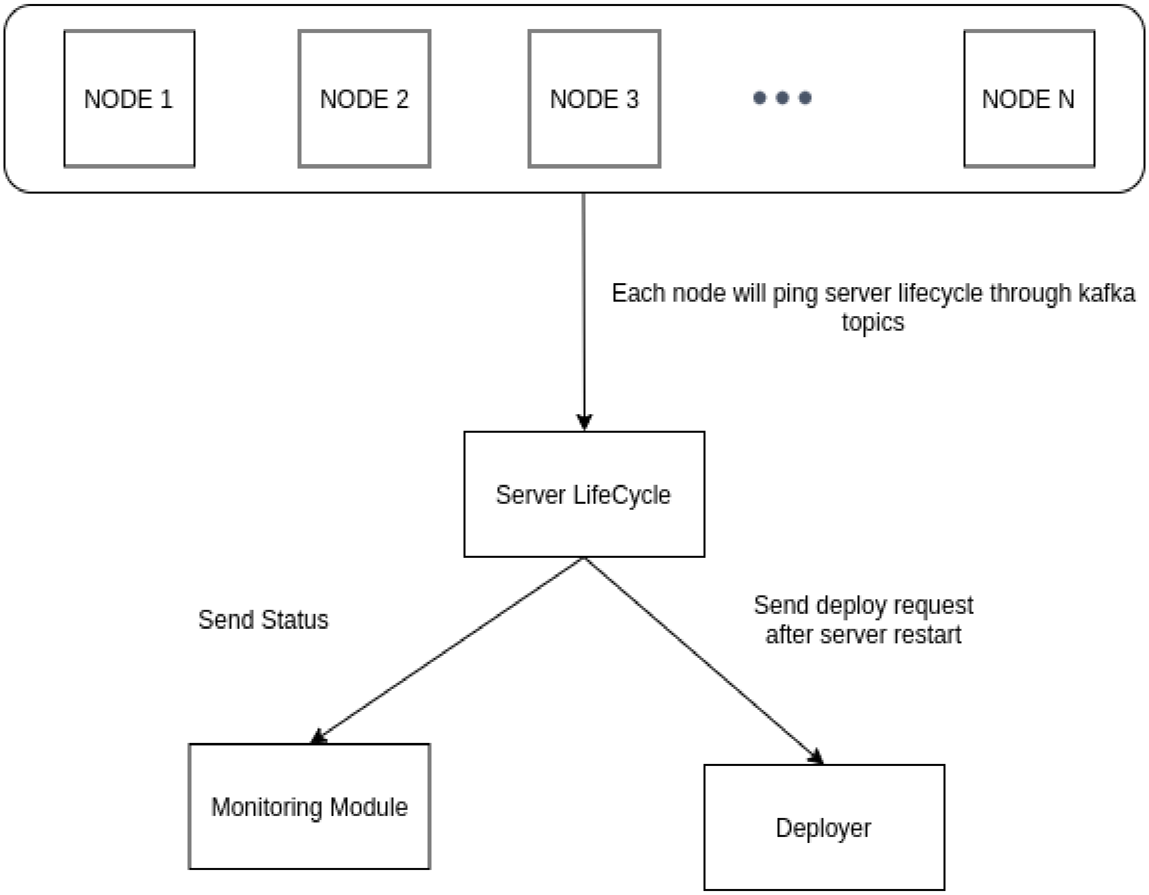
Overall block diagram

Subsystems block diagrams-

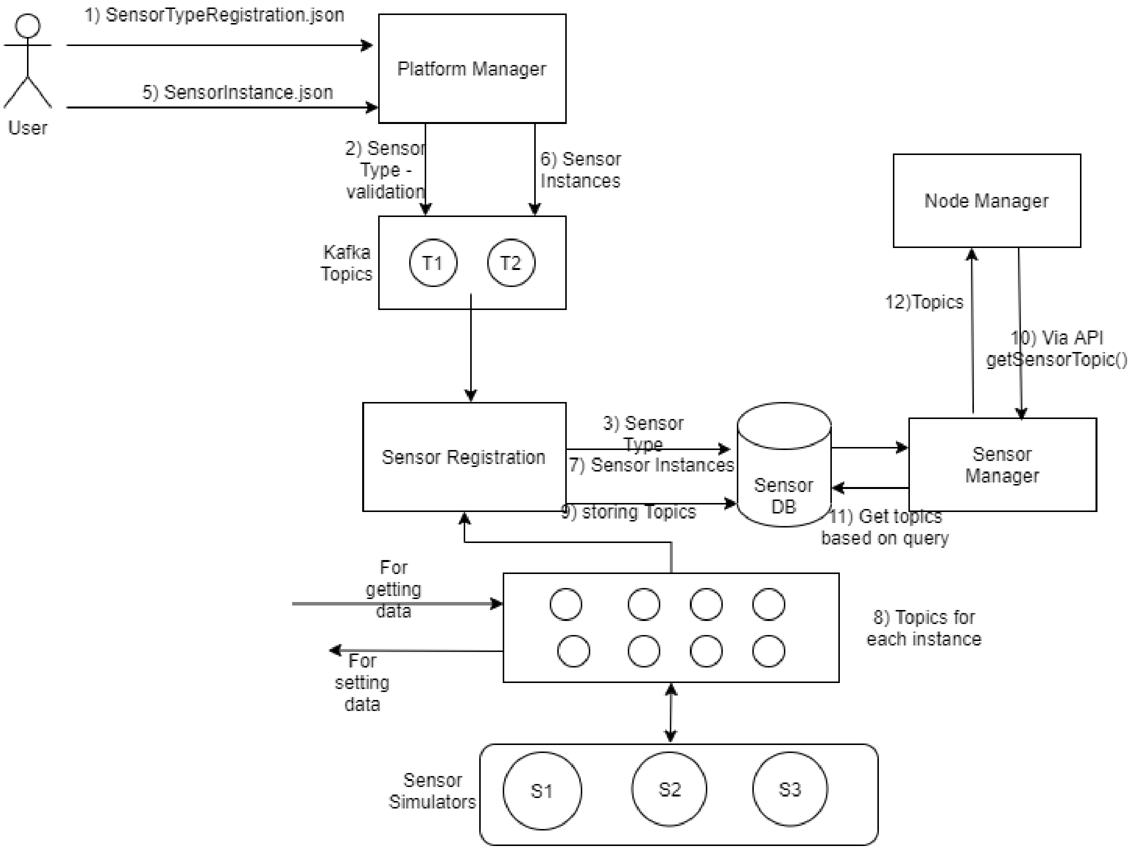
- **Service Lifecycle**



- **Server Lifecycle**


NODE 1
NODE 2
NODE 3
NODE N
Each node will ping server lifecycle through kafka topics
Server LifeCycle
Send Status
Send deploy request after server restart
Monitoring Module
Deployer

Sensor Registration and Manager-



c) Registry & Repository

- Application Repository - For maintaining the different application codes to be deployed on the platform.
- User Repository- For maintaining the information of different users.
- Sensor Repository - Storing the different sensor types details and the topics corresponding to different instances of sensors.
- Log Repository - Storing logs of the system.

d) Understanding requirements of each subsystem-

- Platform Deployment , UI(Platform manager) and Notification
- Deployment of the Application

  1. **Sensor Binder:**

  - Sensor binder receives a request from platform manager in the form of deployConfig.json file.
  - We use the filters given in the config file which are used to decide which sensors should we bind to the application.

- The module looks into the database where the details of sensors are stored to find the ones matching to our criteria.
- It throws an error if it does not find the sensors to bind required by the application.
- Else, it generates an instance id and forwards the deploy config file to scheduler which handles from there on.

2. Scheduler

- Scheduler receives the application id and the location of the configuration file in the application repository from the platform manager.
- Scheduler will fetch the configuration file and it would have to verify if the configuration in the file are in the correct format and according to the constraints of the platform(if any).
- Scheduler will have to store the scheduling details in a list and set some form of cron job which would grab the necessary files and pass it to the deployer at the appropriate time/interval.
- It will also send a request to stop the application to the deployment manager either at the scheduled end time or an interrupt received from the platform manager.
- It will do so by maintaining two priority queues, one queue will be used to handle the start time of the scheduled applications and the other queue will be used for handling the stop times.

3. Deployer:

- Deployment Manager receives the application id and the location of the configuration files in the application repository from the scheduler.
- It will have to locate and identify the sensors which are associated with the application that is to be deployed. It would have to communicate with sensor manager to get the information about the sensors.
- Deployment manager will have a sub system called a load balancer which will receive the number of nodes required for the application from the deployer. It will try to find the required number of nodes with least load and will create nodes if there aren't enough nodes available. It would return the details of the nodes bound with the application to the deployer.

4. Load Balancer:

- Load Balancer keeps receiving information about the load(RAM usage, CPU Usage) on computing instances from the monitoring module so that it can make decisions to balance the load.

- Deployer will setup the environment and packages on the nodes provided by the load balancer and then start the application program on those nodes.

5. **Application level Monitoring**

- It will monitor various application statuses and If it is one of the following 137,143,133 status codes, it shows that the application was stopped/exited abruptly by any reason like shutting down of the machine or any external interrupt. In that case, our module will detect such cases and send the restart request and restart the application on the machine suggested by load balancer.
- Dashboard will ask this module about container logs detail and It will provide required container logs to the dashboard.

6. **Server & Service Lifecycle:**

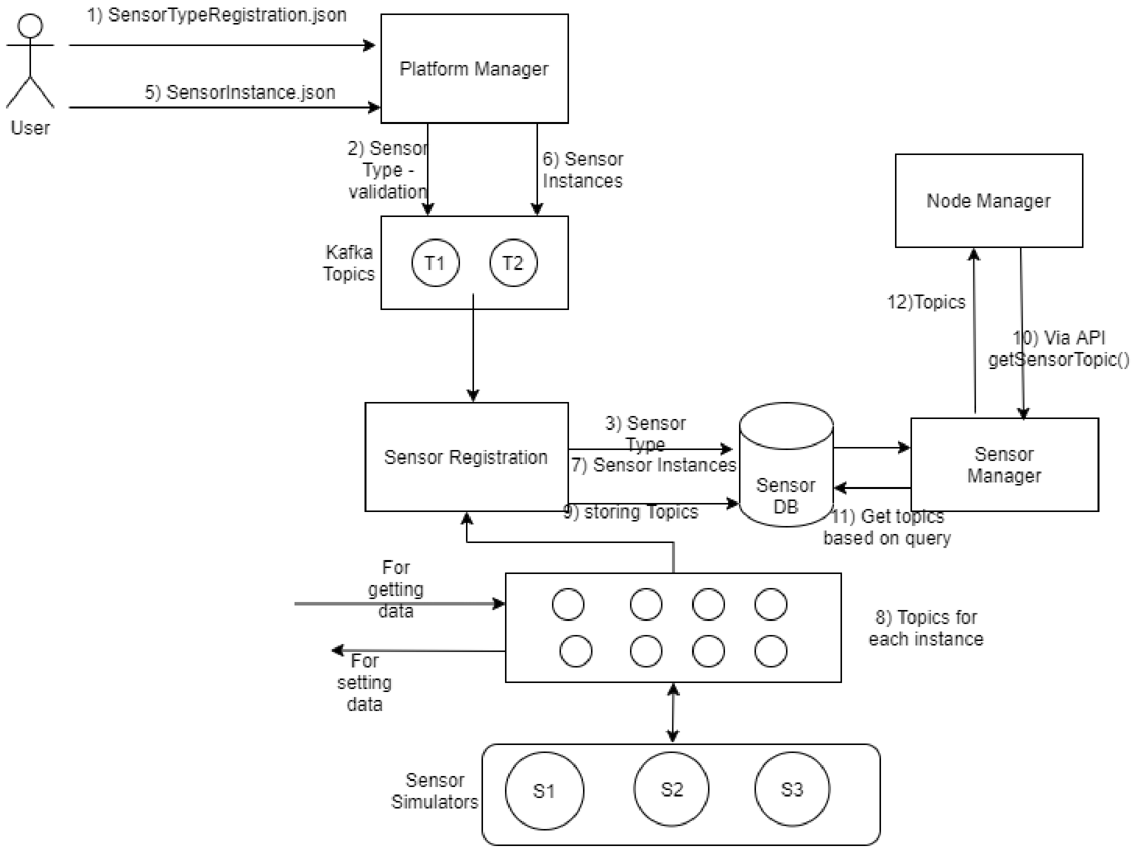
- Each server will have a kafka topic on which it will send its status every 30 seconds.
- The Server lifecycle will consume from each server's topic and if it does not receive anything for 80 seconds, it assumes the server is down and will restart the server.
- If any applications were deployed on the server app monitoring module will detect it and after the server is restarted it will re-run all the applications that were running at the time of crash.
- If any platform services were deployed on the server service lifecycle manager will detect it and after the server is restarted it will re-run all the services that were running at the time of crash.
- Service Lifecycle will fetch a list of all containers running in the platform from MongoDB.
- Every service of the platform is running in a docker container. It will only consider service containers from a list of all containers and will check if the container is running fine using container logs. If it finds out that service has crashed, it will restart a service and update it in MongoDB collection (service_status) with a new container id.

- Sensor Registration and Management

  1) Brief overview of the Module

  a)One block diagram

  Block diagram



b) Functional Overview

Our team basically deals with the sensor registration and management.

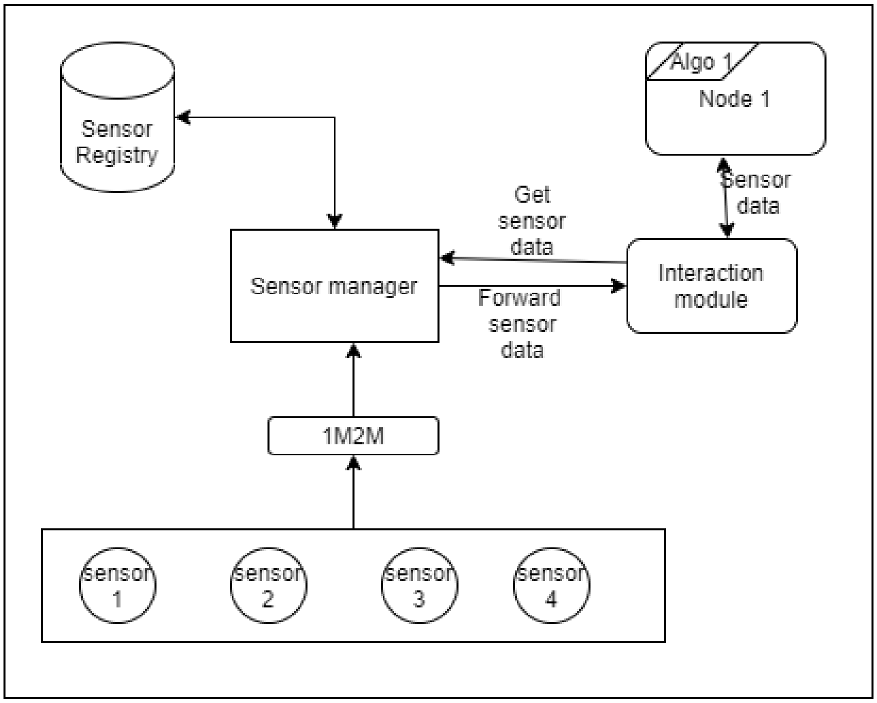
1. **Sensor Manager-**
   This sub-system connects with sensors and gets the raw data of streams. It processes the raw data into proper model input form and starts streams it via the Message Queue which is consumed by the application on the server. This is the module to manage the sensor data and provide it to the algorithm via the API provided by the platform.

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 form of a configuration file. Sensor Registration service parses this config file and stores the details about the sensors in Sensor database. It further sets up the sensors and their gateways.

c) List of sub-systems in this part

a) Brief description of each component

Sensor Manager-

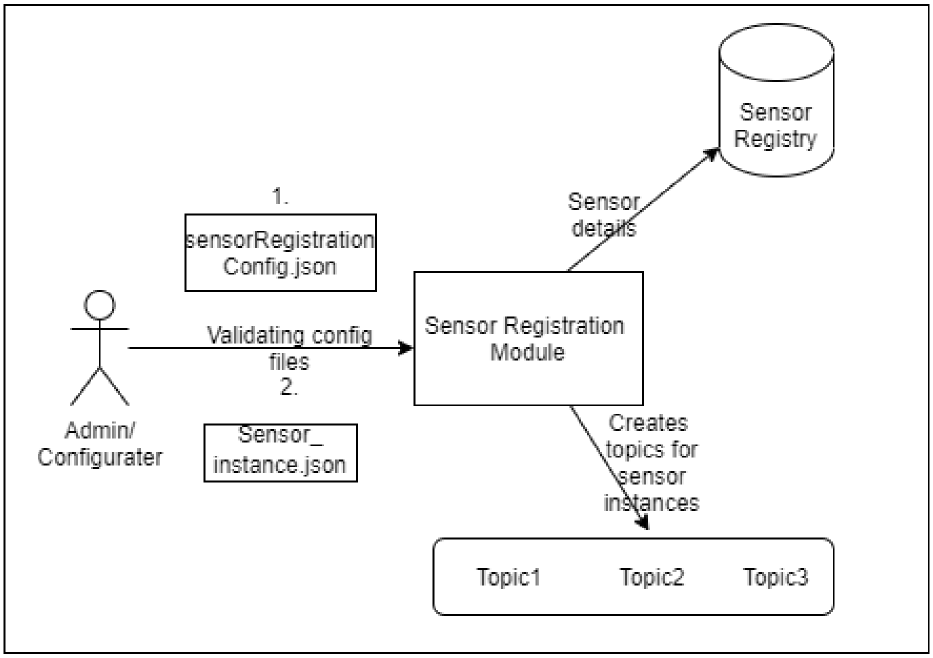


a) **Sensor Data Processing**: It processes the raw data into proper input form (as required by the application) and starts streaming to the Message Queue at a particular data rate which is consumed by the application deployed on the server.

b) **Data Binding:** 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 a sensor data pushes it to the respected topics on the topics. From where application can pull the sensor data.

Sensor Registration-



a) Parsing and verifying the Config file obtained from the application developer to get the details of sensor and its geolocation.

b) Registering the details of the sensor in platform repository.

c) Registration of type and instance of sensors.

d) Setting up the newly registered sensors with their gateways.

d) List of services/capabilities

Sensors

- Simulating the sensor types.
- Generating Topics where the sensors produce data and topics to consume data from so as to trigger actions on the sensors.

Sensor Registration

- Providing a sample config file for sensor type registration and registering the sensor types.
- Providing a sample config file for registering sensor instances and registering the sensor instances to be used by the sample application.
- Storing the sensor details of the sensors in the sensor db to be used by the sensor manager.

Sensor Manager

- Creating temporary topics to be used by the Applications for getting sensor data.
- The applications use API to get the topic names from the sensor manager and use it to get and set data from the sensors.
- Processing of the sensor data to be provided in the format required by the applications.

e) Interactions between this and other parts. Nature/purpose of interactions, likely interchange info/services

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 .

For interaction between the modules, we are using KAFKA producer-consumer architecture.

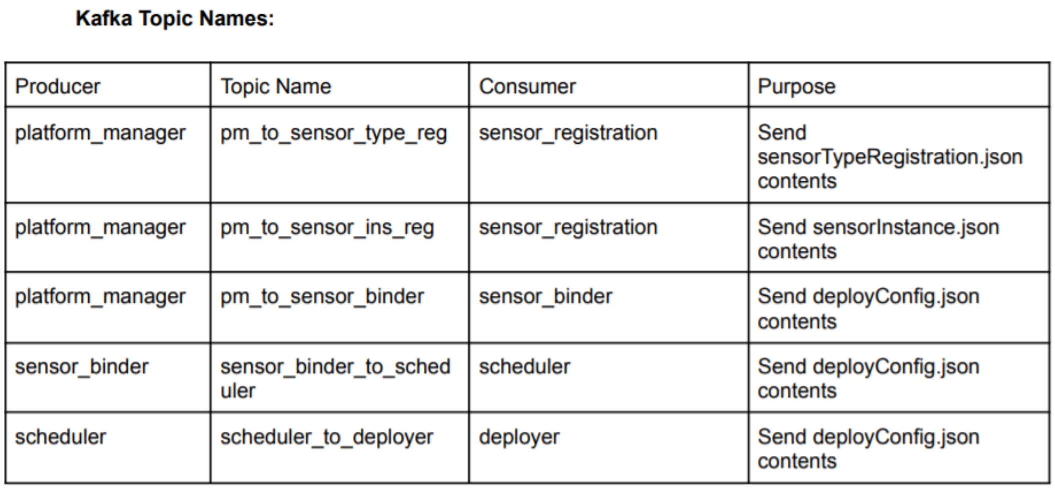
In our model we have used this communication at different levels regarding our module-

Between Platform manager and sensor registration-

2 topics are created , 1 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

4. 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 the sensors be triggered successfully .

Service Lifecycle Manager

1. Checking status of each service
2. If a service is no longer running status in DB should be INACTIVE.
3. If a service is running status in DB should be ACTIVE.
4. After restarting the service, the new container id and the ip of the machine on which it is running should get updated in DB.

Server Lifecycle Manager

1. If we don't get a ping from a particular node for a certain amount of time, the server lifecycle will detect it and should restart the node again.
2. After restart all the applications that were deployed , should be restarted.