FINAL PROJECT DESIGN

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

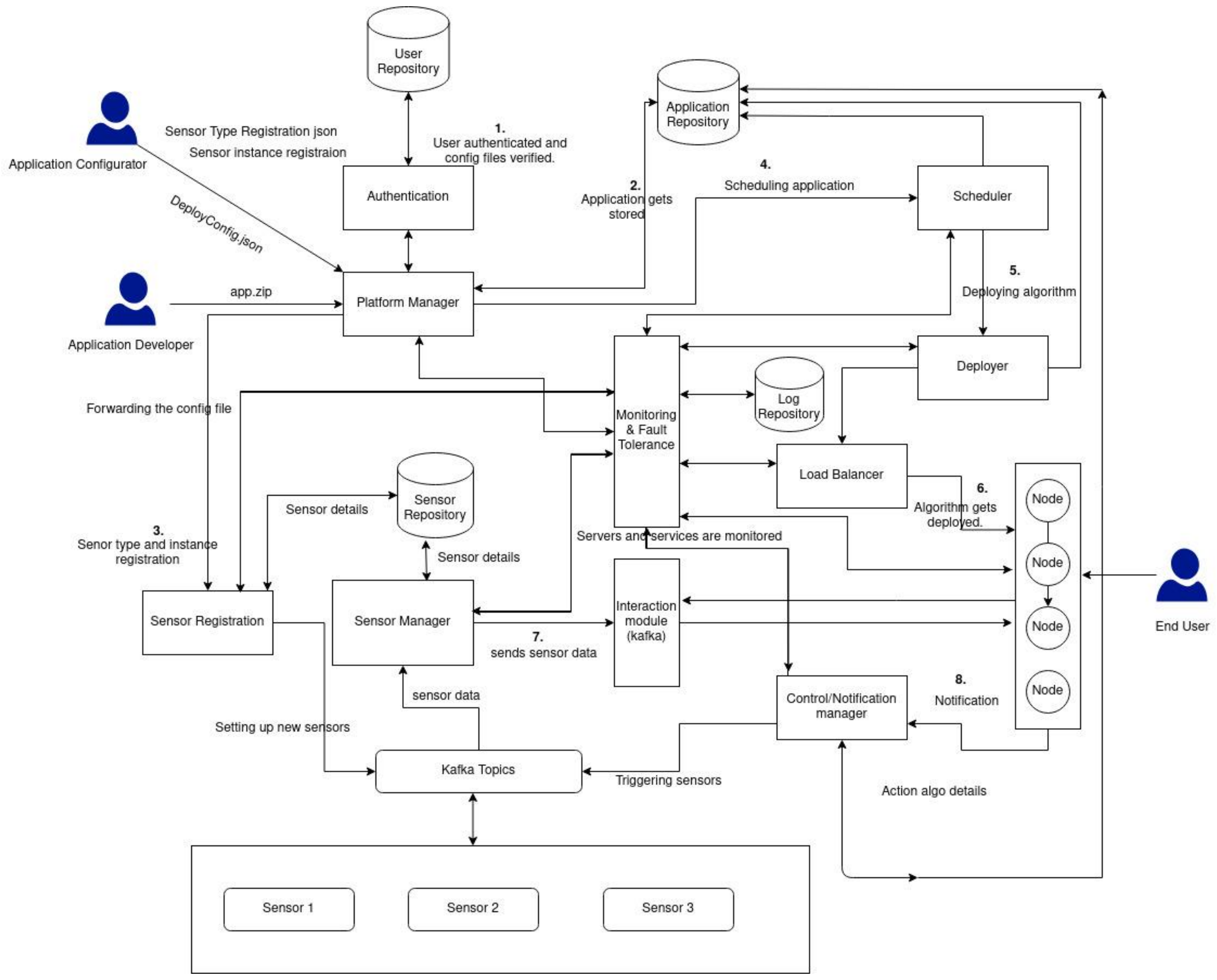
Akshay Choudhary

Shweta Arya

Overview

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 product 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 take action on the sensors.
The application can be developed at the developers side, the sensors needed can be easily registered on the platform ,and then the application can be scheduled and deployed over the platform following the application model defined by our platform.



Overall flow of the Platform

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. **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.

Test Cases :

Authorization Test cases :

1. During signup check whether username is already registered.
2. During login check whether the user exists or not.
3. If a user exists, validate the password is correct or not.

Application manager Test cases:

1. Upon receiving a zip file, check whether there is an application config file.
2. Validate the format of the application config file.
3. Check if the sensor types used in the application(which are mentioned in the app config file) are registered on the platform. Throw an error if there is no such sensor type registered on the platform already.
4. If everything is alright, store the application source files and the app config file in the application repository.

Deployment manager test cases:

1. Before deployment, see if the application and the algorithm asked to be deployed is a valid one
2. If the application and the algorithm is valid, determine the sensor instances which have to be bound with the algorithms. Throw an error if the sensors cannot be determined from the given DeployConfig file.
3. Verify if there are valid environment details that have to be set up on the computing instances.
4. Decide the instances on which the application will be run. This is done with the help of the load balancer. New computing instances have to be created if enough instances are not available.
5. Stop an application if there is a stop request. Throw error if there is no such application running.

Monitoring test cases:

1. If there is disruption in any running service, the module should be able to restart the service from its last stable state.
2. If any machine is down, the module should be able to restart the machine and run the algorithms that were running at time of disruption on that machine.
3. Monitoring should be able to ping the service/machines to check the status.

Sensor Manager test cases:

1. During sensor registration it will successfully be able to parse the SensorTypeRegistration.json and SensorInstance.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.
4. Will it be able to properly simulate the real time sensors .
5. Output is received from the running instance of an application to this module.
6. If any notification is needed to be sent it is received at end user from control/notification manager.
7. Will the control actions be performed on the sensors as expected.

Integration Test Cases for platform-

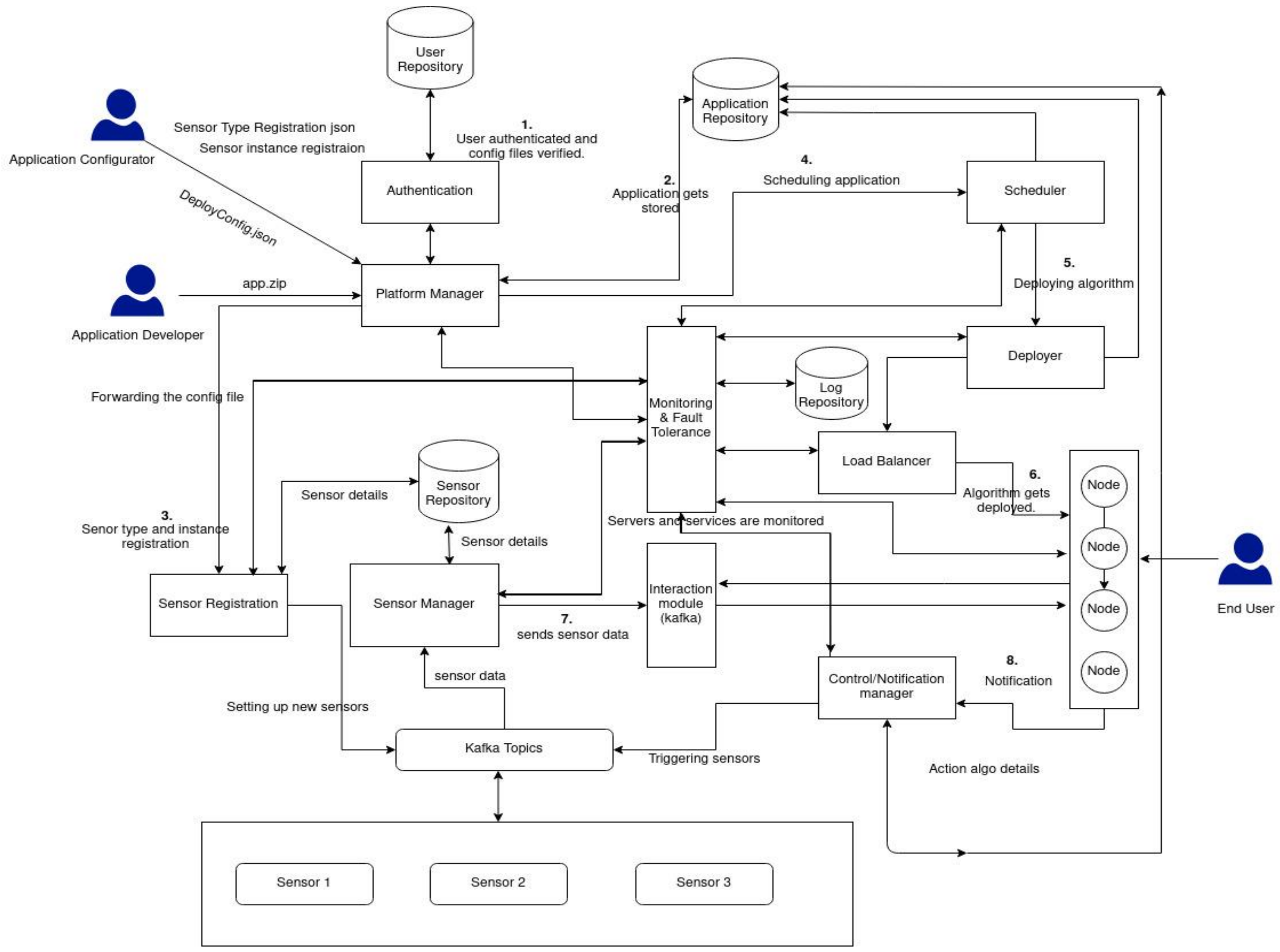
1. Verify Files uploaded in platform manager are sent on the topics used by sensor registration for registering the sensors.
2. Verify After successful registration the sensor details get stored in Mongodb collection sensor_instance.
3. Verify on uploading the deployConfig.json the application is scheduled by the scheduler on the server.
4. Verify the deployer has deployed the application at the scheduled time.
5. Server lifecycle will make sure the application runs smoothly by checking server status periodically.
6. Service lifecycle makes sure every service in the platform runs without any disruption by immediately starting the service back if it crashes.

7. Verify the dashboards are giving out the required outputs and notifications to the User.

Solution design considerations :

- **Design big picture**

  Block Diagram of platform



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 to login then options to upload application folder and all other config files described in detail below.

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 deployConfig.json file. Which contains which algorithm is to be deployed and what time.
- Send a request start/stop an algorithm and IP,Port associated with the algorithm to the deployer according to the requests in the configuration file (deployConfig.json).

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 Manager:
- To make sure application servers are running properly without any interruption or overloading, Server Lifecycle will check the stats of each server regularly. It does so by making sure there is no overload on any server and even if the server crashes due to any reason, it restarts the server immediately by re-running the application.

Service Lifecycle Manager:
- There shouldn’t be any interruption of any kind in the application running due to failure of any platform service. Service Lifecycle manager will manage all services within the platform and restarts the service if required.

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 (sensorTypeRegistration.json). Sensor Registration service parses this config file and stores the details about the sensor types in the Sensor Repository.

- When the application configuration provides sensorInstances.json file, which contains the details of how to communicate with the sensor(IP,port), this service will make a kafka topic and sends all the data received from the sensor to this topic. The topic names for each sponsor are saved in the Sensor Repository.

Sensor Repository:

- Contains details of sensor types
- Contains details of sensor instances and the kafka topic name associated with it.

Control/Notification Manager:

- This module is responsible for doing some action or notifying users 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 the application repository from where it can be used to act(on the sensors) and notify accordingly.

- **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
  - MongoDB
  - Kafka, for communication between different modules
  - Docker Container

- **Approach for communication & connectivity:**

Communication between application developers, configurator with the platform as well as communication between the end user and the platform will happen mainly via config files.

For example, a configurator can register a new sensor type with the platform via giving the sensor details in a config file called
SensorTypeConfig.json.

Similarly when an application developer wants to run his application on the platform then he first informs the platform regarding which sensors to use via a config file called AppConfig.json.

- **Registry & repository:**

  UserRepo: For storing the application developer details for the authentication to the platform.

  SensorRepo: It stores the sensor instances with their kafka topic details, so that they can be easily retrieved when queried.

  AppRepo: This repo stores all the applications provided by the application developer and respective files, like (appConfig.json,interface.json)

- **Server 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:**
  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 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.

- **Scheduler:**
  The application configurator will send a DeployConfig file to the scheduler which has the scheduling and deployment details.
  The format of the scheduling details in the DeployConfig file is validated and a cron job has to be setup according to the request.
  The sensors have also to be bound before scheduling a cronjob because we have to give the kafka topics and ip:port of the sensors in the command line arguments to the algorithm.

- **Load Balancing:**

Load Balancer will receive information about load on each computing instances. It will get information about CPU usage, RAM usage from monitoring module.
From that information Load balancer will take decision about where to schedule application.
After that Deployer will setup the environment and packages on the nodes provided by the load balancer and then start the application program on those nodes.

- **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-

  - appConfig.json
  - SensorTypeRegistration.json
  - SensorInstanceRegistration.json
  - deployConfig.json

  Api-

  - getSensorData()
  - setSensorData()

Application Model and Users view of the system :

API provided by the platform:

getSensorData(ip,port) : kafka topic return.

1. The above function will first find out the kafka topic name from the sensor repo.
2. Then get the data from the topic name and return it to the algorithm which called the function.

getKafkaTopic() → return

Roles :

Application Developer:

- Write the algorithms by getting the sensor data using the api provided by the platform.
- Write appConfig.json and submit a proper app.zip file to platform.

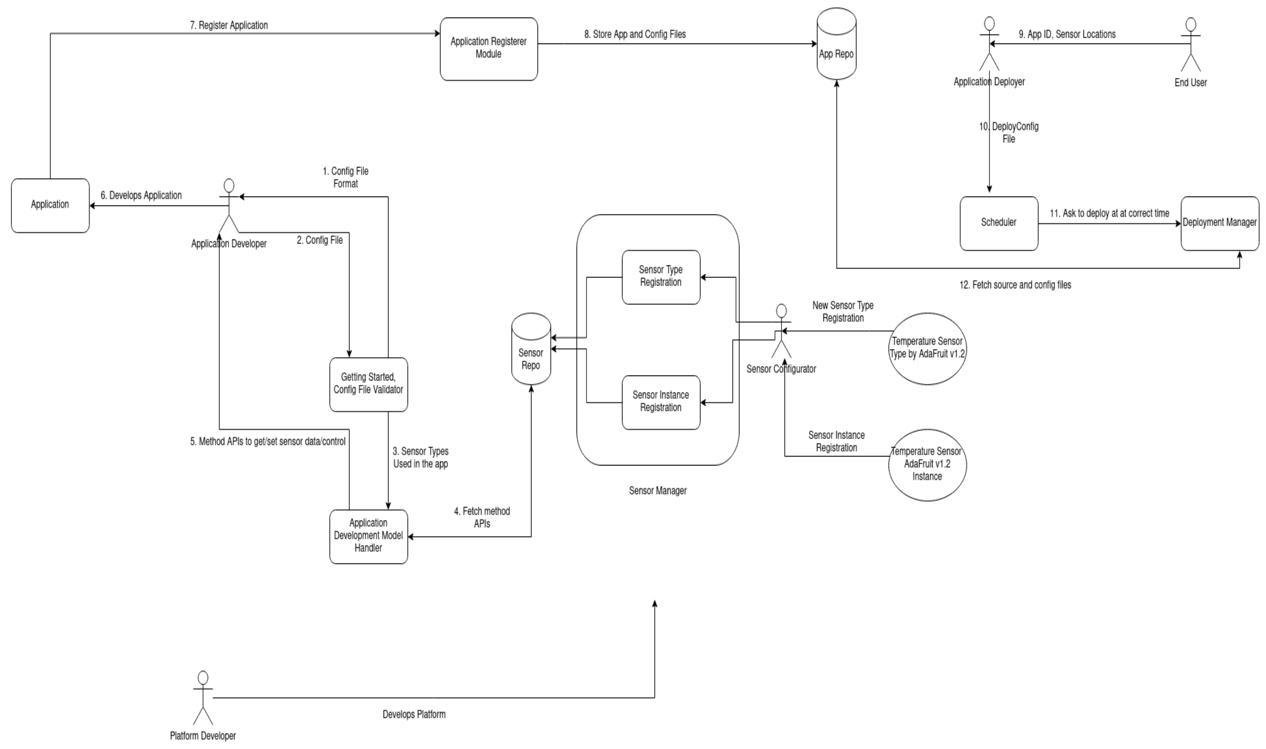
Application Configurator:

- Register the sensor Type and the sensor instances. Using sensorTypeRegistration.json and sensorInstance.json
- Provide a config file (deployConfig.json) 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.

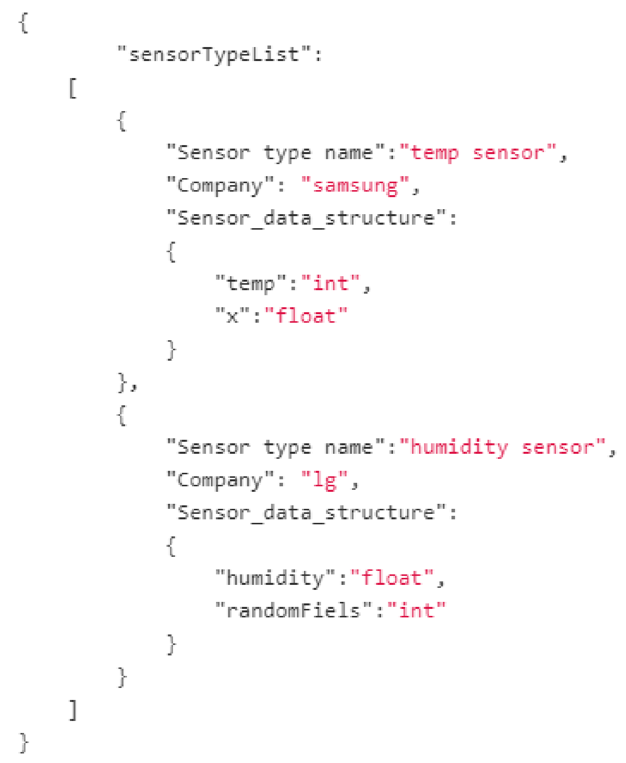
Link to the Application Development Model Flow diagram:
https://drive.google.com/file/d/1kTuKxr758avz_JfU4kjWfnfwnOeFOvL6/view?usp=sharing



7. Register Application
Application Registerer Module
8. Store App and Config Files
App Repo
9. App ID, Sensor Locations
Application Deployer
End User
10. DeployConfig File
Application
6. Develops Application
Application Developer
1. Config File Format
2. Config File
Scheduler
11. Ask to deploy at at correct time
Deployment Manager
12. Fetch source and config files
Sensor Type Registration
New Sensor Type Registration
Temperature Sensor Type by AdaFruit v1.2
Sensor Repo
Sensor Configurator
Getting Started, Config File Validator
Sensor Instance Registration
5. Method APIs to get/set sensor data/control
3. Sensor Types Used in the app
Sensor Instance Registration
Temperature Sensor AdaFruit v1.2 Instance
Sensor Manager
4. Fetch method APIs
Application Development Model Handler
Platform Developer
Develops Platform

Application Model:

Step 1) Application Configurator will define the sensor types and register the sensor type with the platform using sensorTypeRegistration.json



```
{
        "sensorTypeList":
    [
        {
            "Sensor type name":"temp sensor",
            "Company": "samsung",
            "Sensor_data_structure":
            {
                "temp":"int",
                "x":"float"
            }
        },
        {
            "Sensor type name":"humidity sensor",
            "Company": "lg",
            "Sensor_data_structure":
            {
                "humidity":"float",
                "randomFiels":"int"
            }
        }
    ]
}
```

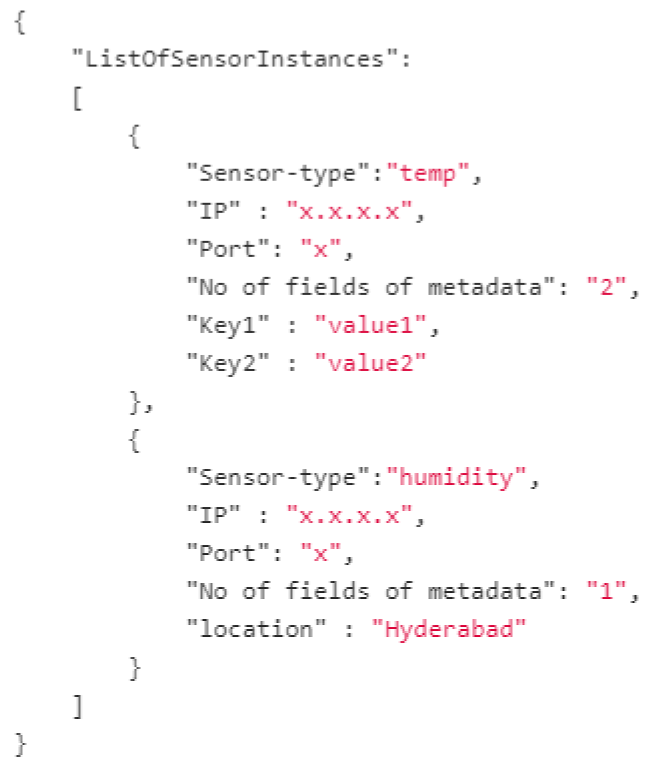
Step 2) Developer needs to write an algorithm by taking the ip and port of the sensor with which it wants to connect. To get data from the sensor will use the api provided by the platform. (getSensorData(string ip,string port)).
Will provide the app.zip file with format as below.

```
app
├──────Src
│   ├──────script1.py
│   └──────script2.py …
├──────appConfig.json
```

appConfig.json:

```
  "application name": "farm_management",
  "application Id": "1",
  "environment": {
    "os": "Linux",
    "modules": [
      "python"
    ],
    "storage": "mongoDB"
  },
  "algorithmList": [{
    "algorithm_name": "soil_profile",
    "developer_id": 867,
    "script": {
      "name": "soil_profile.py",
      "path": "farm_management/soil/soil_profile.py",
      "Dependency": "python 3.0"
    },
    "Interface": {
      "path": "farm_management/configurations/interface.json"
    },
    "sensor_type": {
      "sensor1": {
        "name": "soil moisture sensor",
        "company": "Sony",
        "output": "Moistue content: float",
        "output_rate": "10 kbps"
      },
      "sensor2": {
        "name": "Temperature sensor",
        "company": "Warner Electronics",
        "output": "Temperature: Kelvin",
        "output_rate": "1 kbps"
      },
      "sensor3": {
        "name": "alarm sensor",
        "company": "FIGARO",
        "output": "On/OFF: string",
        "output_rate": "1 kbps"
      }
    }
```

Step 3) Platform will validate appConfig.json file. If it is proper the app folder will be saved in the application Repository.
Step 4) Application configurator will create a sensorInstance.json file. Which contains what sensors to be registered.
sensorInstance.json :



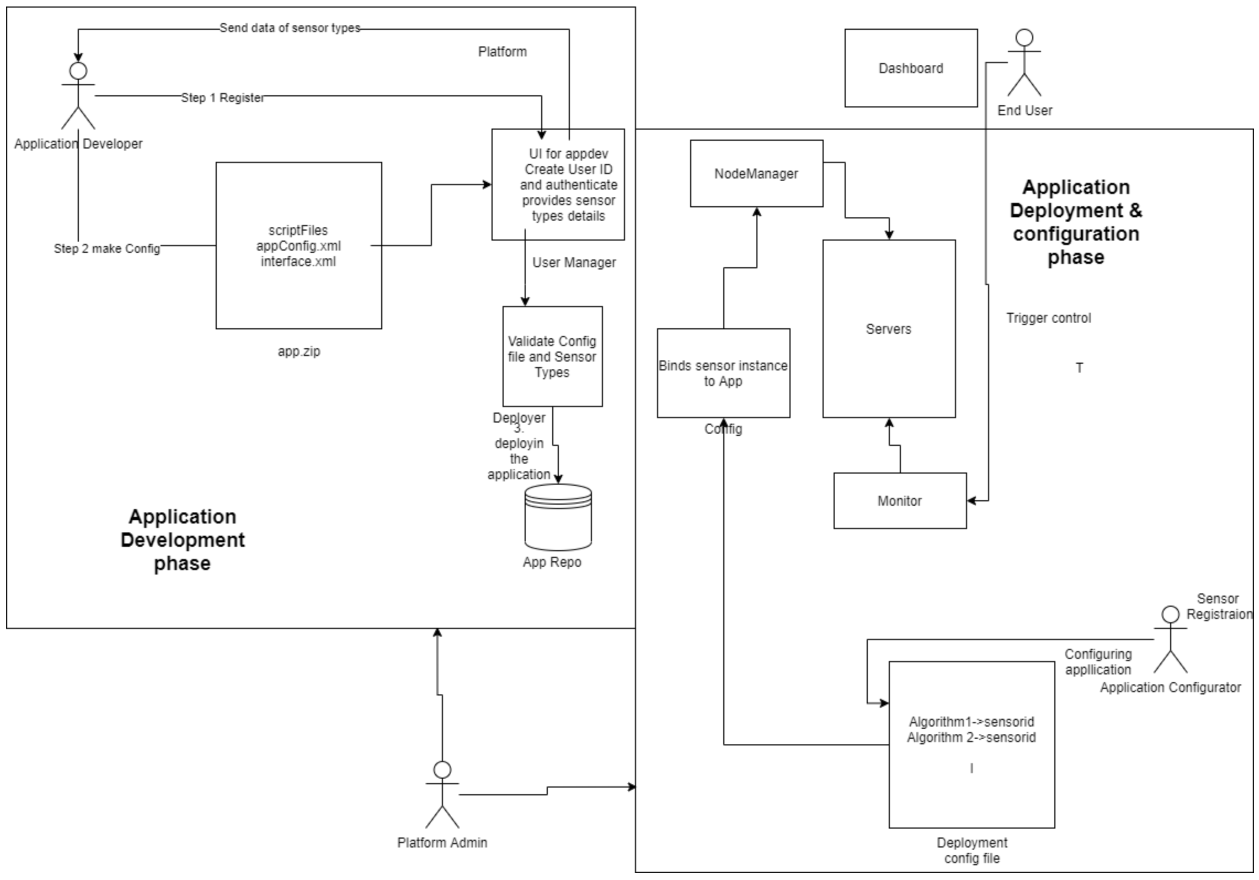
```
{
    "ListOfSensorInstances":
    [
        {
            "Sensor-type":"temp",
            "IP" : "x.x.x.x",
            "Port": "x",
            "No of fields of metadata": "2",
            "Key1" : "value1",
            "Key2" : "value2"
        },
        {
            "Sensor-type":"humidity",
            "IP" : "x.x.x.x",
            "Port": "x",
            "No of fields of metadata": "1",
            "location" : "Hyderabad"
        }
    ]
}
```

Step 5) Platform reads the sensorInstance.json for each sensor a new kafka topic is created and the sensor data received by sensor Manager will dump the data to this topic. The topic names and sensor instance details are saved in sensor Repository.

Step 6) Application configurator will create a deployConfig.json file. Which contains which algorithms are running on which sensor data. Also mentions when to deploy and how often using a cron job expression.

DeployConfig:

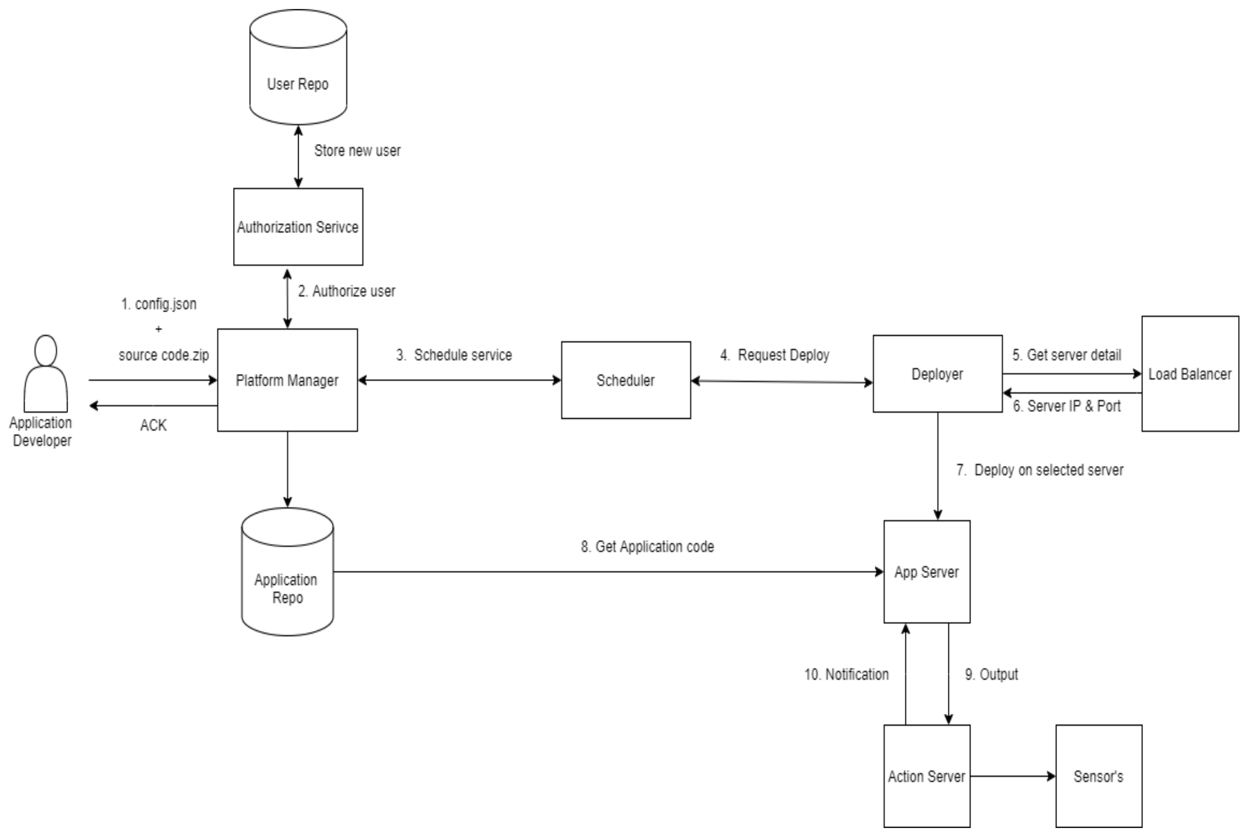
```
{
        "Application ID": "12",
        "Deployables":
    [
        {
            "algorithmName" : "health monitor",
            "SensorInfo":
            [
                {
                    "Sensor type" : "temp",
                    "Sensor IP":"x.x.x.x",
                    "Port": "x"
                },
                {
                    "Sensor type" : "pulse",
                    "Sensor IP":"x.x.x.x",
                    "Port": "x"
                }
            ]
        }
    ]
}
```



Send data of sensor types
Platform
Step 1 Register
Application Developer
UI for appdev Create User ID and authenticate provides sensor types details
Step 2 make Config
scriptFiles appConfig.xml interface.xml
app.zip
User Manager
Validate Config file and Sensor Types
Deployer
3. deployin the application
App Repo
Application Development phase
Dashboard
End User
NodeManager
Servers
Application Deployment & configuration phase
Trigger control
T
Binds sensor instance to App
Config
Monitor
Sensor Registraion
Configuring appllication
Application Configurator
Algorithm1->sensorid Algorithm 2->sensorid
I
Deployment config file
Platform Admin

Application Development and Deployment

Application Deployment Model:



Steps to interact :

1) The application developer signup through platform manager UI dashboard.
2) After successful login, the developer can upload their application source code along with config files to the platform.
3) Config files will be stored in the application repository.
4) Using config files, the scheduler will parse the scheduling information and request the deployer to start or stop the application program(s).
5) In case of starting an application, the deployer asks the load balancer to assign computing instances for the application.
6) Deployer starts or stops the application on the computing instances associated with the application.

Persistence:

- In case a node crashes, the system brings back it 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.

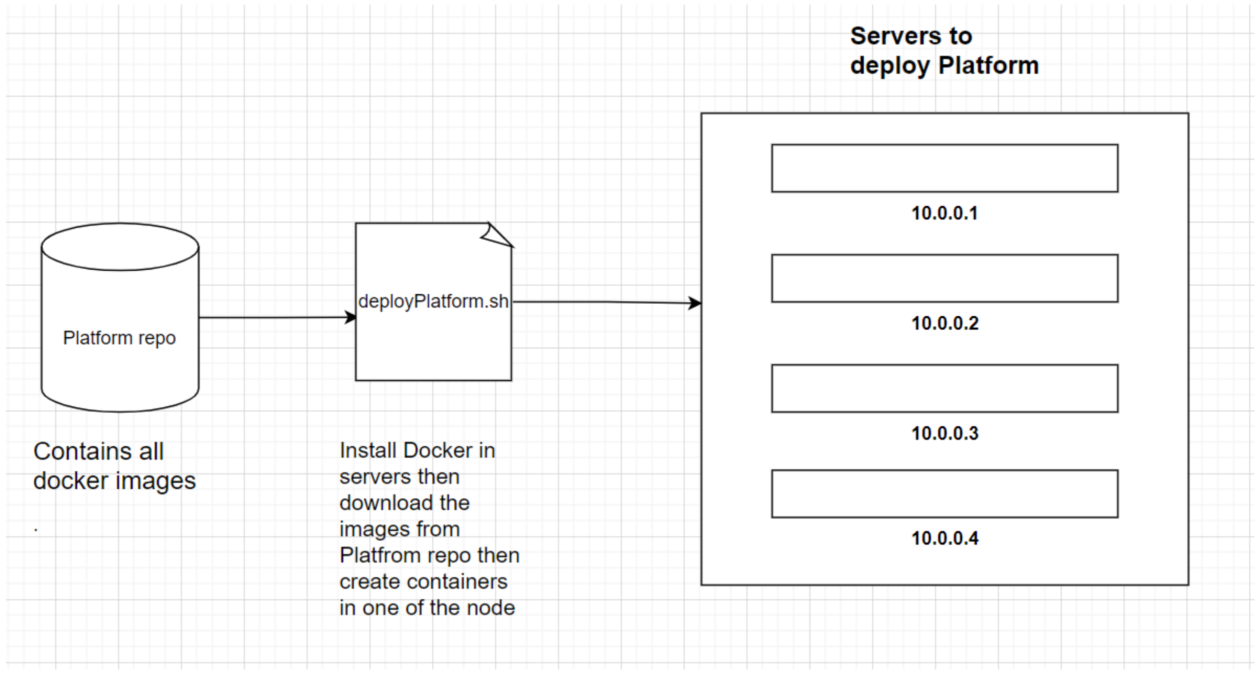
Low Level Design:

List the 4 major parts :

1. Platform Deployment and UI(Platform manager)
2. Deployment of the Application
3. Monitoring and Fault tolerance
4. Sensor and Notification

PLATFORM MANAGER & DEPLOYMENT

Deploying the platform:


Servers to deploy Platform
Platform repo
deployPlatform.sh
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4
Contains all docker images
Install Docker in servers then download the images from Platfrom repo then create containers in one of the node

Platform repo: (Search for docker repo)

Platform repo contains the docker images of all the services which are supposed to run in the platform.
The following docker image files should be present

1) Platform Manager
2) Scheduler
3) Node Manager
4) Deployment Manager
5) Authorization Service
6) Monitoring Services
7) Sensor Manager
8) Registering a Sensor Service
9) Action Manager

Where should we deploy all the above services (which node)?
Fault tolerance if that node fails ? → Deploy all services in Multiple nodes (Research)

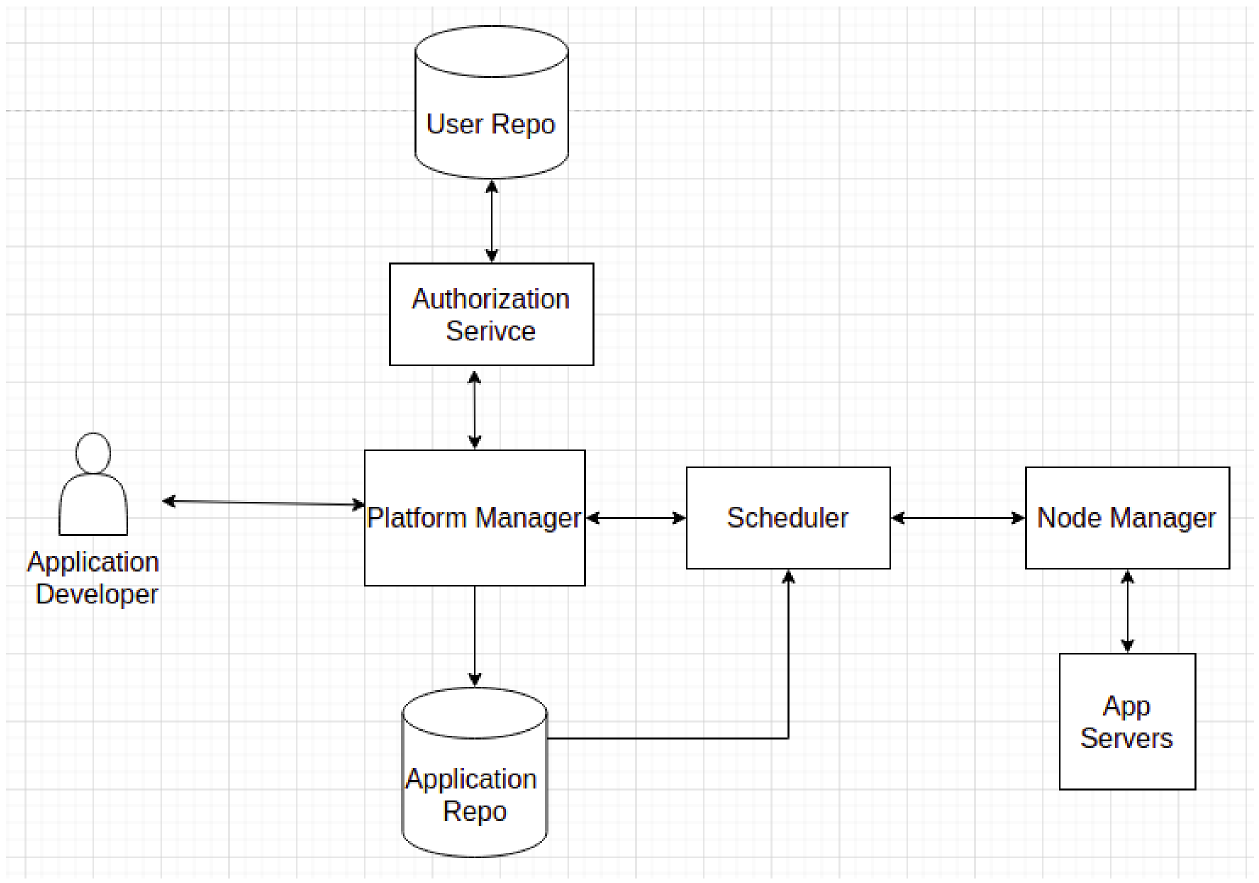
deployPlatform.sh :

Input : List of IP addresses where the platform needs to be deployed, location of the platform repo.
Functionalities:

1) Should install docker in all the servers
2) In one of the servers (say 10.0.0.1) download all the images from the platform repo and start the container.
3) Now we have servers (10.0.0.2, 10.0.0.3, 10.0.0.4) for deploying the user Applications
4) Install Kafka as a messaging queue

Platform Manager:

This is the interface between Application developer and the platform



Roles and Responsibilities :

1) Provides UI for registering and logging in for the user of the platform(Uses Authorization service)
2) Service to upload or update the respective users application inthe Application Repo
3) If the User deploys the application Platform Manager will send the data to the Scheduler to schedule the deployment.

Dashboard:

- 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.

Validator Service

Has the following API:

validate_appConfig(path to app.zip)
validate_appzip()
validate_sensor_type()
validate_sensor_instance()
validate_deployConfig()

This service checks the folder structure of app.zip file.
Also checks the json format of the files given.

Sensor Controller:

API provided by this service

- setSensorData(instance id, sensor index)
  Returns the kafka topic

This service is used to set a value to a particular sensor or to control the sensor.
This is implemented using one more kafka topic. That is each sensor have two types of topics one for data inputs and other if it exists for the control functions.

The platform_libfile.py handles the api call to the platform manager whose job is to fetch the kafka topics related to our query.
We get the kafka topics then we will make a producer and add a signal according to the requirement.

Application Scheduling and Deployment

Introduction

- **Scheduler**: Scheduler takes as an input the time and the intervals at which certain programs are to be run from a configuration file. Job of the scheduler is to start and stop jobs/programs according to the requests received. It sends the requests of starting and stopping jobs to the deployer as it is the deployer module which actually deploys programs on computing instances.

- **Deployment Manager**: Deployment Manager is responsible for deployment of programs on computing instances. It receives these requests from the scheduler. The deployment manager has to recognize the sensors which are associated with the application. It also needs to determine the packages and the environment that needs to be set up for the application. There will be a sub system called load-balancer which would decide which computing instances should it deploy the programs directed by the scheduler.

Functional Overview and Interaction Between Modules

1. **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.

2. **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.
    - Load Balancer also 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 set up the environment and packages on the nodes provided by the load balancer and then start the application program on those nodes.

Roles and Responsibilities

1. **Scheduler:**

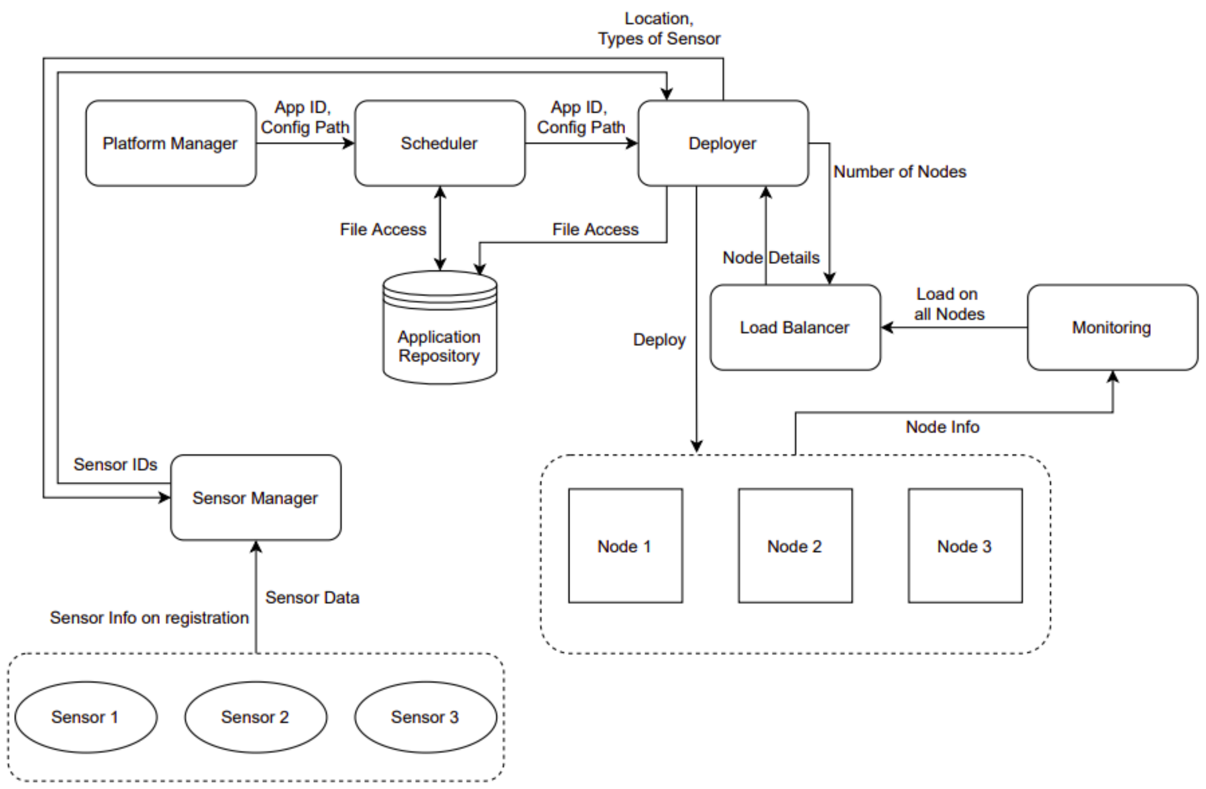
    - It has to start or stop jobs according to the schedule requests.

- It accomplishes these tasks with the help of the deployer.

2. **Deployment Manager:**

   - Deployer has to deploy or stop the programs on the nodes when requested by the scheduler.
   - Load Balancer has to assign the computing instances for an application by using an load balancing algorithm when requested by the Deployer.

Block Diagram



Authentication

1. **Introduction**

   1.1. **Authorization and Authentication:**

   Authentication Factors: Role, User and Password
   Authorization module validates username and password. If validation fails, then it will inform the user about it.

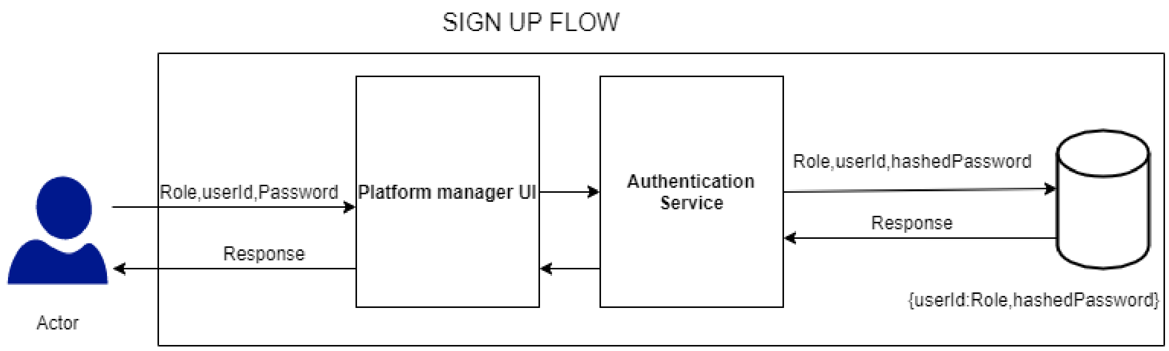
   Authorization: Checking permission for users and validating access.

   1.2. **Monitoring and Fault Tolerance:**

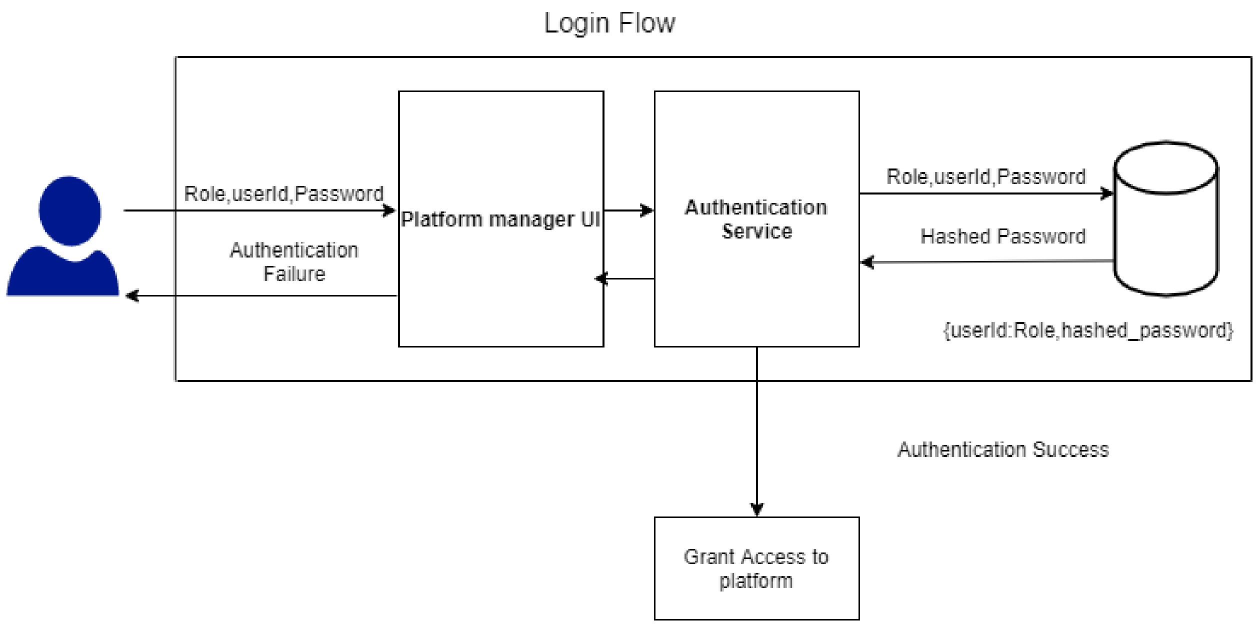
Monitoring and fault tolerance module will monitor all the services such as scheduler, load balancer, deployer,sensor manager etc and will detect any failure in the system. If something in the system breaks, it will detect the fault (what and where is the fault) and help the system recover from it smoothly.

2. Block Diagram

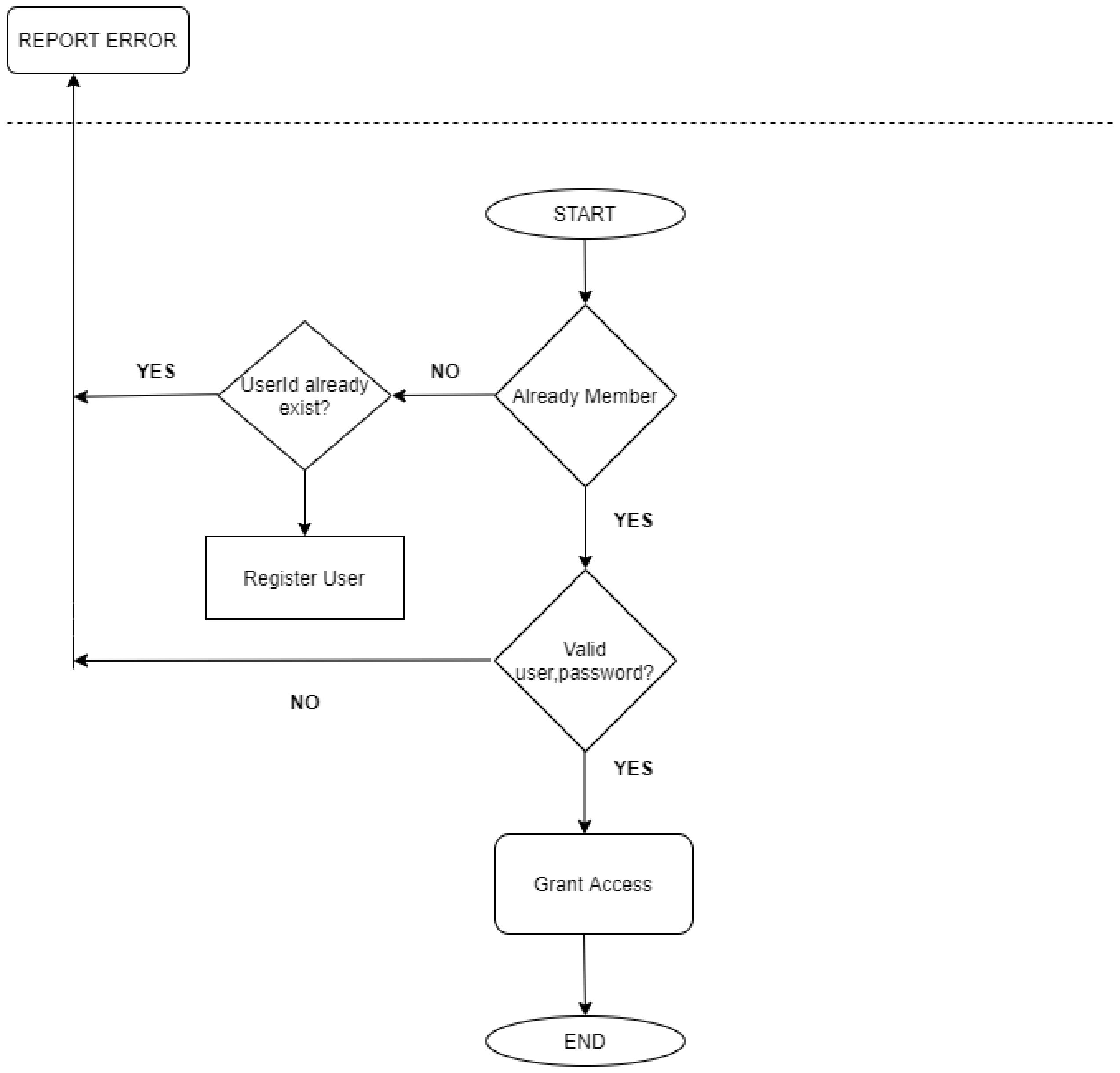
2.1. Authentication



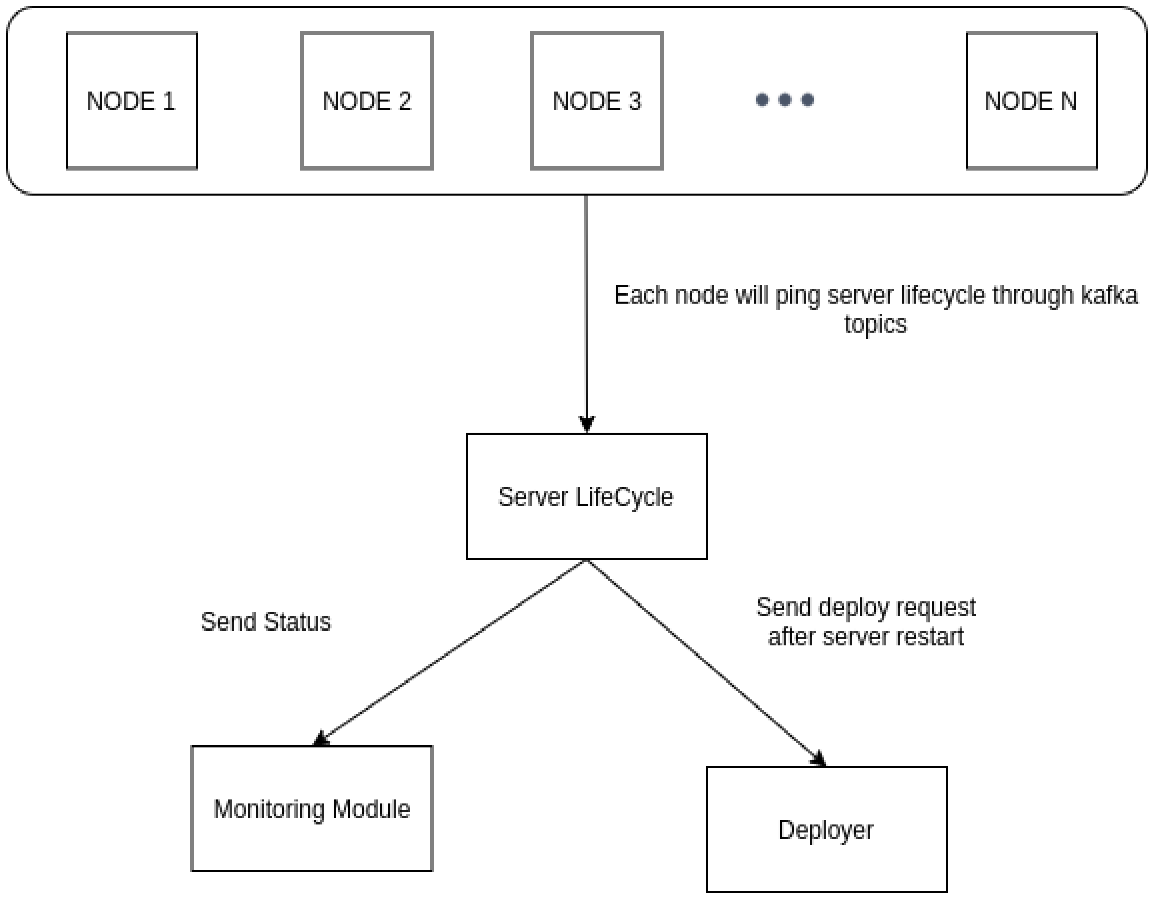
- Users will provide userId(unique) and password for signup.
- If userId already exists, it will return an error response to the user.
- Else password will be hashed and stored in the database.



- User provides userId, password.
- Authentication service will validate the user and on successful validation, the user will be granted access to the platform else it will return login fail error to the user.

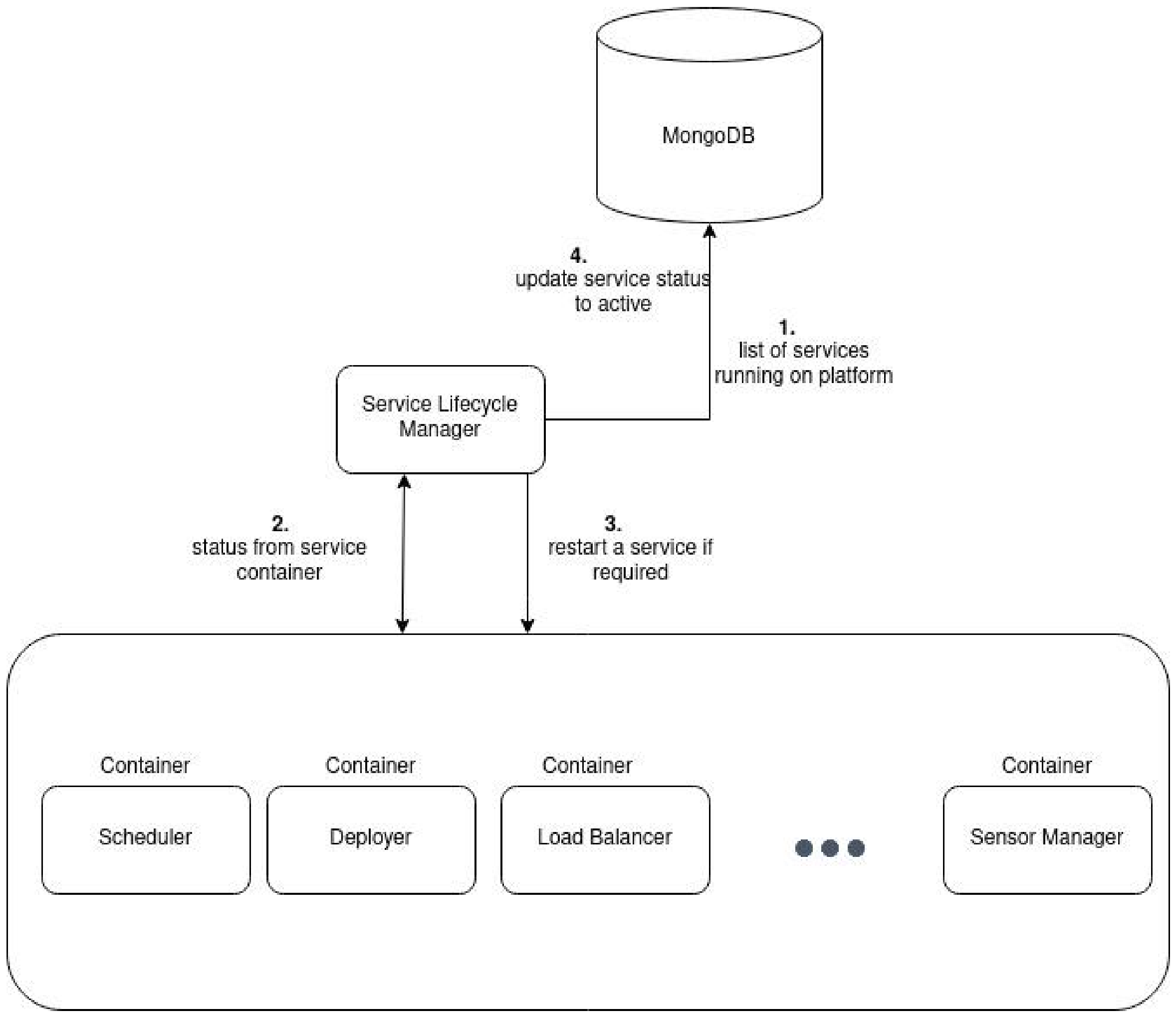


Server Lifecycle Manager


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

- 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.

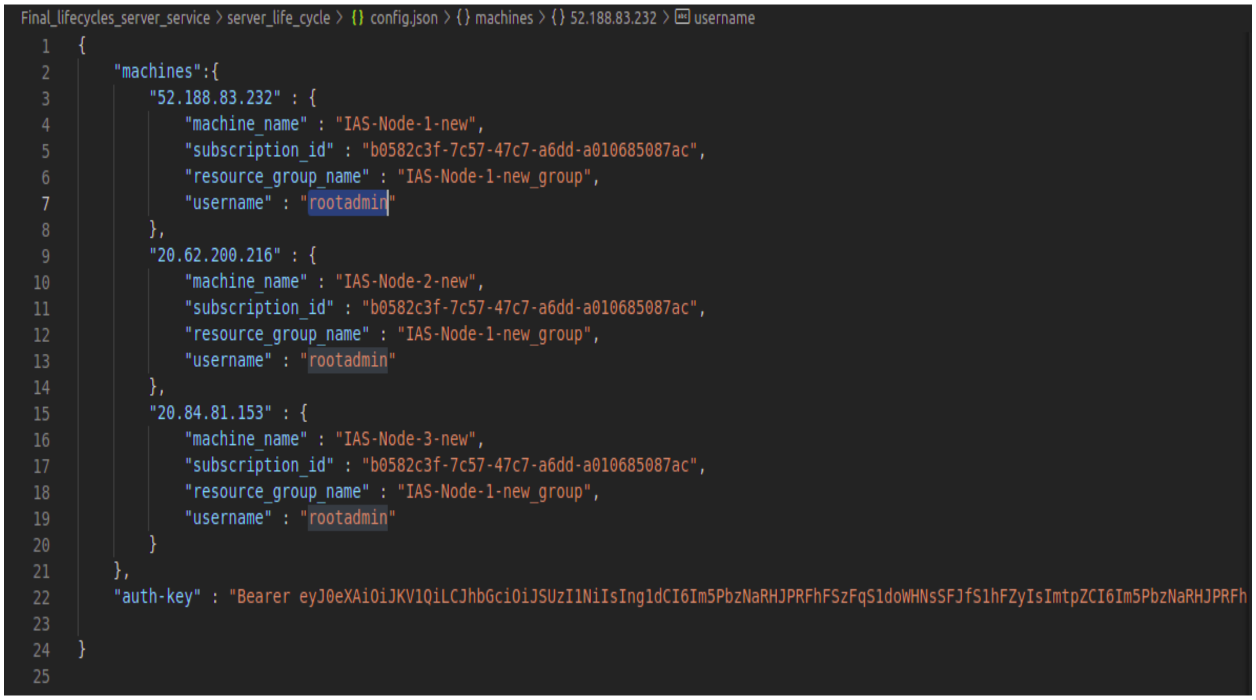
Service Lifecycle Manager



- 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 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.

Configuration Files for service lifecycle manager and server lifecycle manager

config.json



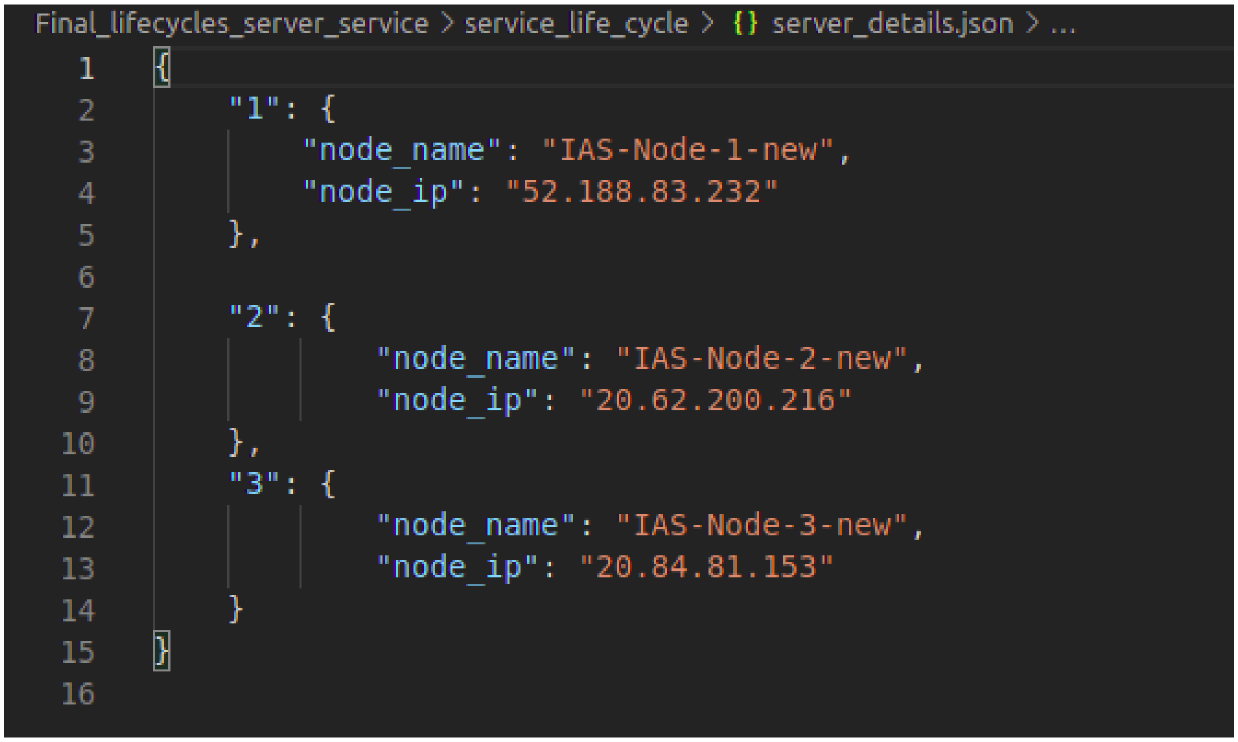
Final_lifecycles_server_service > server_life_cycle > {} config.json > {} machines > {} 52.188.83.232 > username

```json
{
    "machines":{
        "52.188.83.232" : {
            "machine_name" : "IAS-Node-1-new",
            "subscription_id" : "b0582c3f-7c57-47c7-a6dd-a010685087ac",
            "resource_group_name" : "IAS-Node-1-new_group",
            "username" : "rootadmin"
        },
        "20.62.200.216" : {
            "machine_name" : "IAS-Node-2-new",
            "subscription_id" : "b0582c3f-7c57-47c7-a6dd-a010685087ac",
            "resource_group_name" : "IAS-Node-1-new_group",
            "username" : "rootadmin"
        },
        "20.84.81.153" : {
            "machine_name" : "IAS-Node-3-new",
            "subscription_id" : "b0582c3f-7c57-47c7-a6dd-a010685087ac",
            "resource_group_name" : "IAS-Node-1-new_group",
            "username" : "rootadmin"
        }
    },
    "auth-key" : "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNaRHJPRFhFSzFqS1doWHNsSFJfS1hFZyIsImtpZCI6Im5PbzNaRHJPRFh

}

```

Server_details.json (used for load balancing)



Final_lifecycles_server_service > service_life_cycle > {} server_details.json > ...

```json
{
    "1": {
        "node_name": "IAS-Node-1-new",
        "node_ip": "52.188.83.232"
    },

    "2": {
            "node_name": "IAS-Node-2-new",
            "node_ip": "20.62.200.216"
    },
    "3": {
            "node_name": "IAS-Node-3-new",
            "node_ip": "20.84.81.153"
    }
}

```

Sensor Registration and Manager

1. Lifecycle of the module

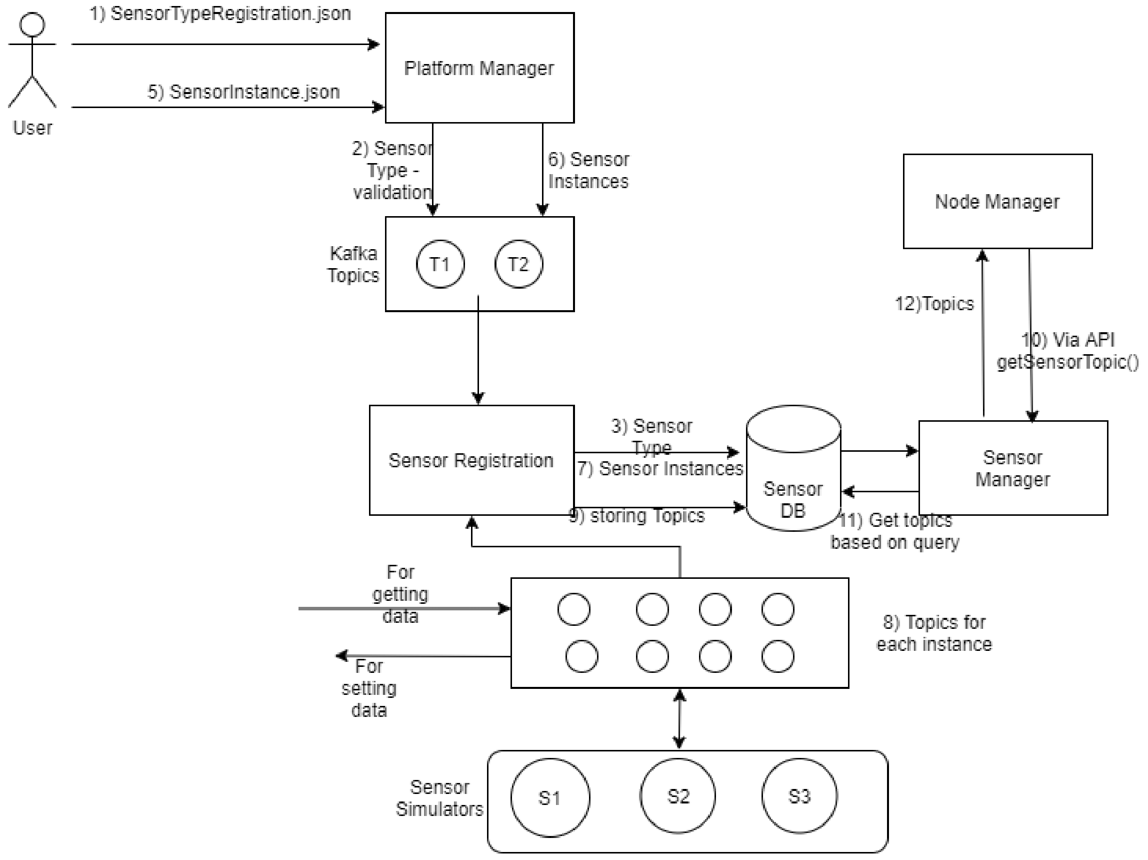
- With the platform initialization service, the modules like sensor registration will connect to platform manager .
- The Sensor manager and registration service will also startup with the initialization of the platform and establish a communication method with sensors.
- The sensor registry will be backed up after a fixed interval of time to monitor the senor 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 streams it via the Message Queue which is consumed by the application on the server.

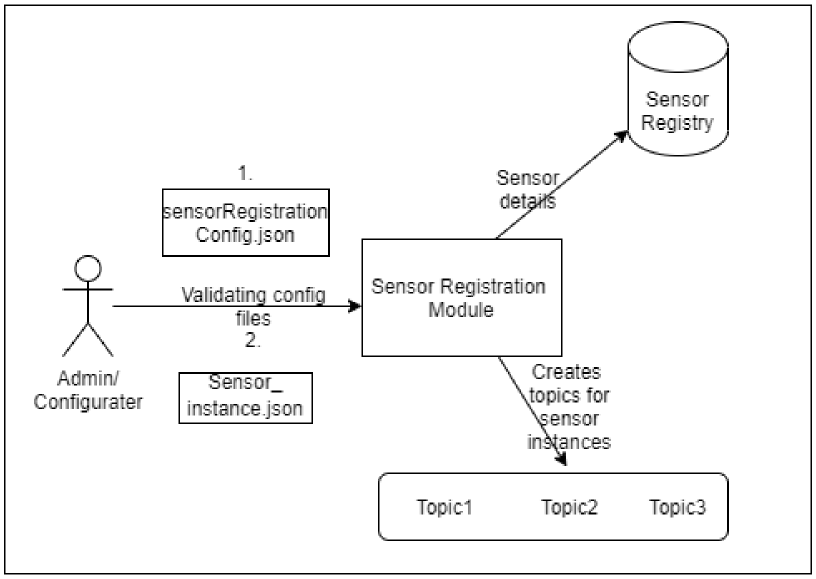
2. **Sensor Registration**- Whenever a new application is deployed in the platform, it also gives in the details of the sensors it needs. The detail about the sensor is provided in form of a configuration file. Sensor Registration service parses this config file and stores the details about the sensors in Sensor database. It further sets up the sensors and their gateways.

A block diagram

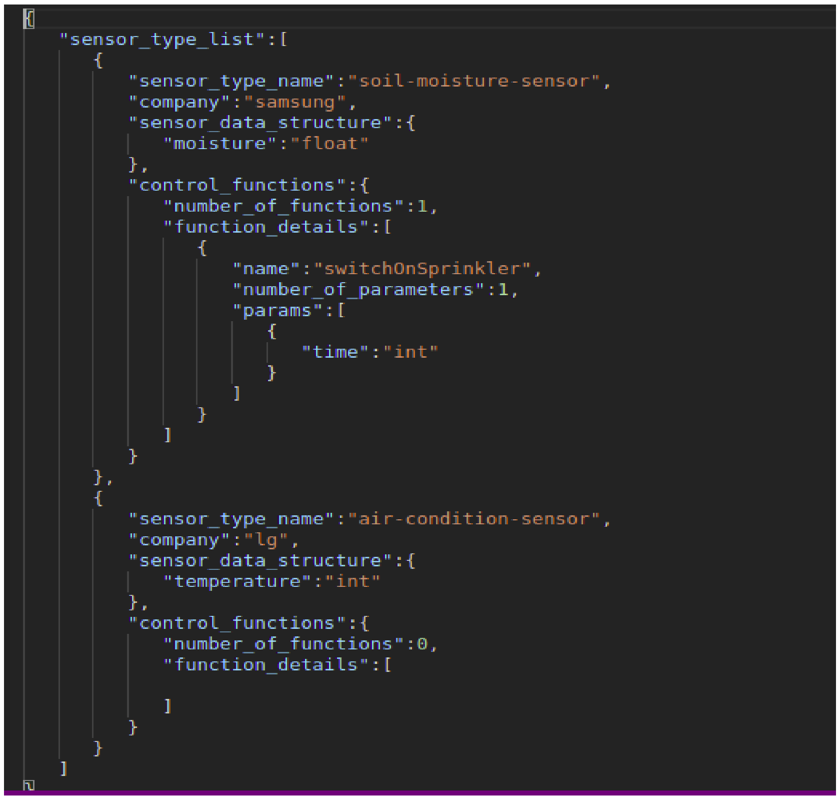


- **Step wise Sensor Registration and Management-**

1) SensorTypeRegistration-



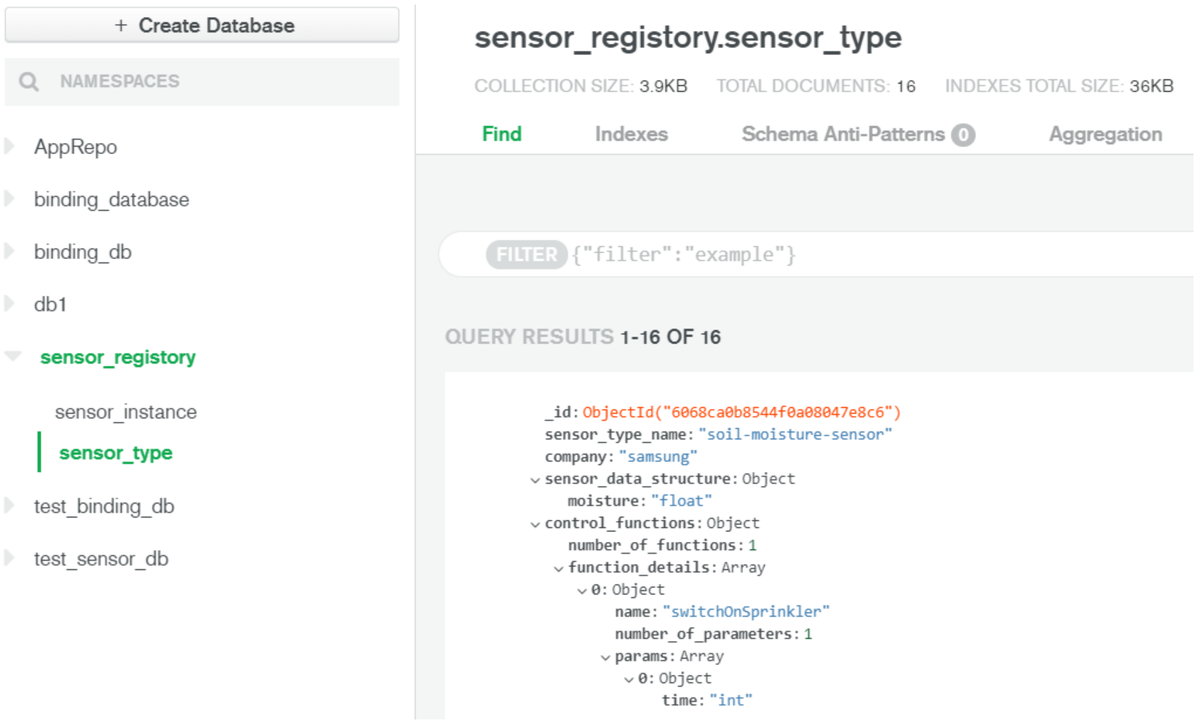
The Application Configurator registers any new sensor type that will be used by the application.He/she uploads a config file- SensorTypeRegistration.json, it is received from platform manager via a kafka topic in the sensor registration module. The config defines details like company,type of data produced,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



sensor_registory.sensor_type

COLLECTION SIZE: 3.9KB TOTAL DOCUMENTS: 16 INDEXES TOTAL SIZE: 36KB

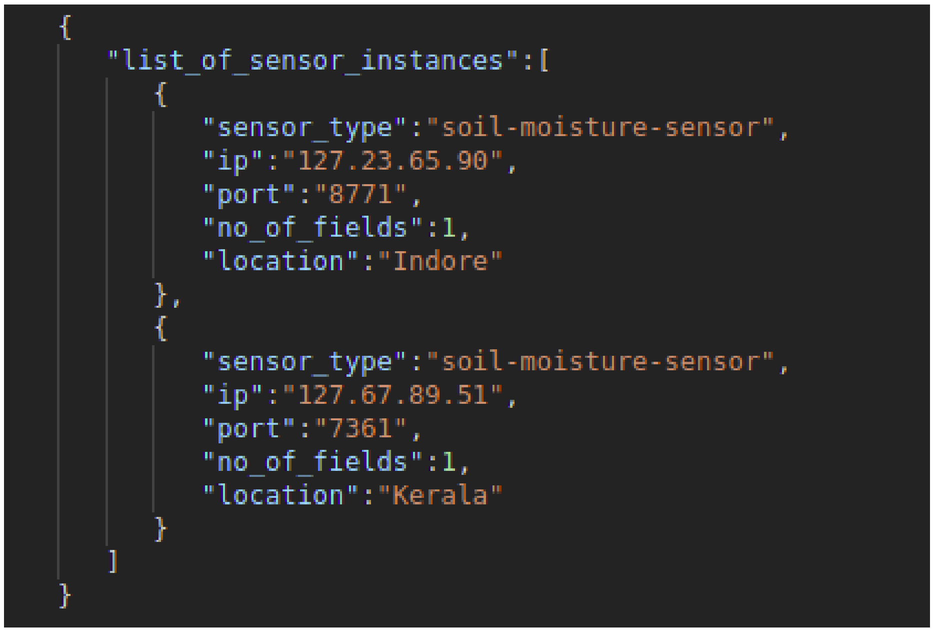
```
{"filter":"example"}
```

QUERY RESULTS 1-16 OF 16

```
_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"
```

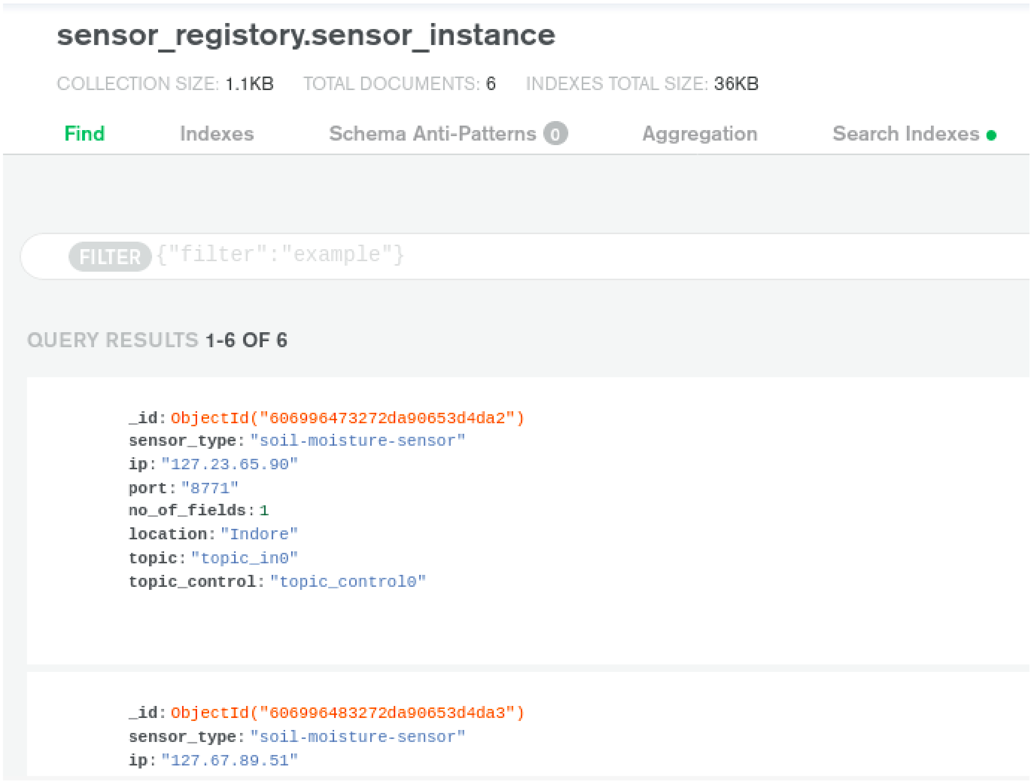
2) SensorInstance Registration-

The Application Configurator registers the instances of the 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
Find
Indexes
Schema Anti-Patterns 0
Aggregation
Search Indexes
FILTER {"filter":"example"}
QUERY RESULTS 1-6 OF 6

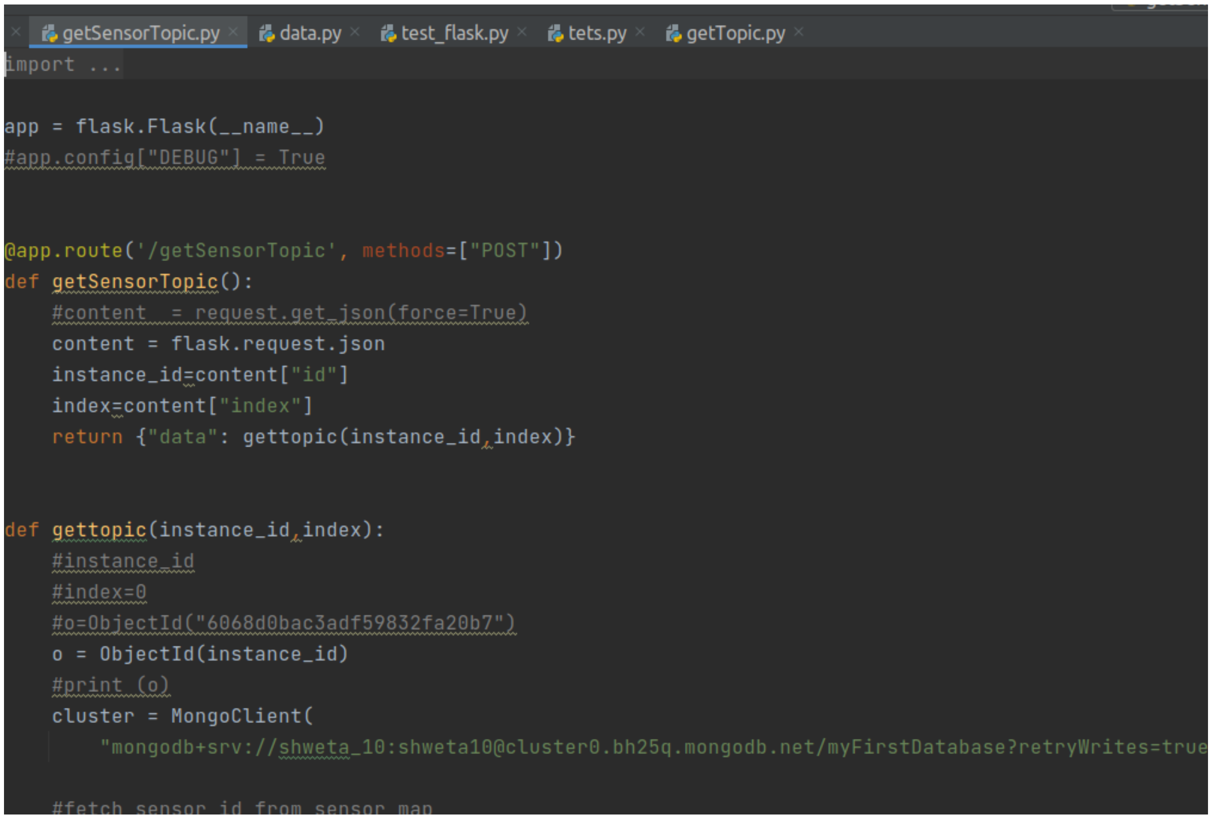
```
_id: ObjectId("606996473272da90653d4da2")
sensor_type: "soil-moisture-sensor"
ip: "127.23.65.90"
port: "8771"
no_of_fields: 1
location: "Indore"
topic: "topic_in0"
topic_control: "topic_control0"
```

```
_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.



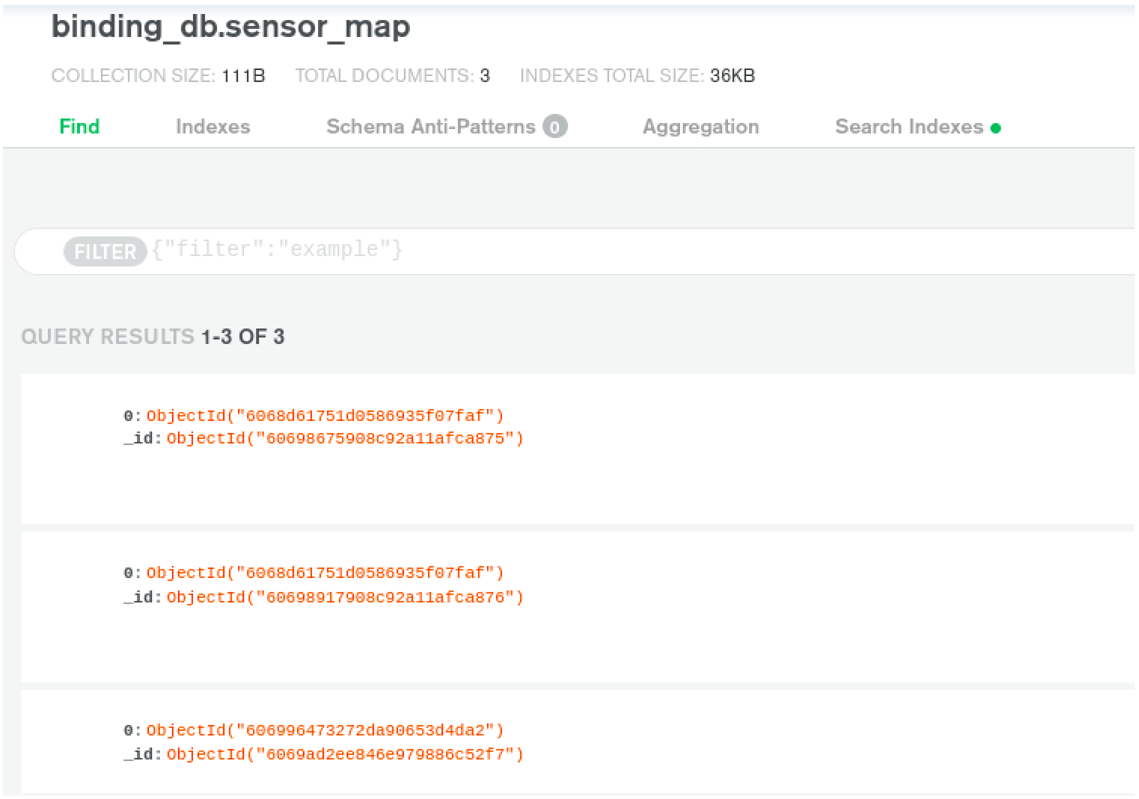
```
import ...

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"}
```

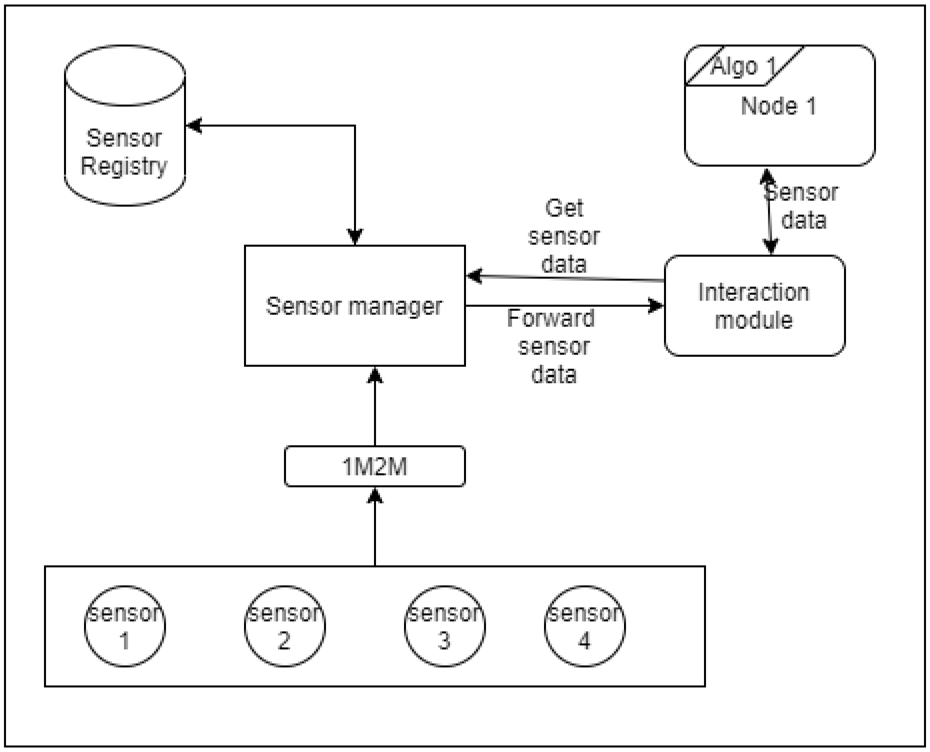
QUERY RESULTS 1-3 OF 3

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

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

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

- **Sensor Manager -**



a) **Sensor Data Processing:**
   - It processes the raw data into proper input form (as required by the application) , like data unit expected and the data rate at which the application is expecting the sensor data.
   - The interaction module takes care of placing processes 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.

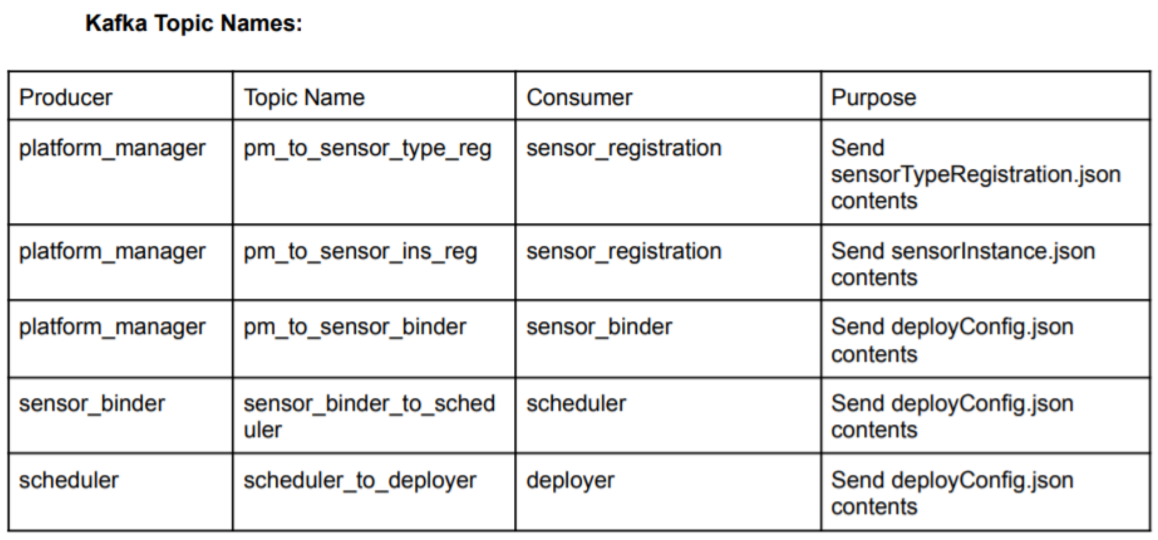
c) **Sending Sensor Data:**
   - Sensor manager upon getting a sensor data pushes it to the respected topics on the topics. From where application can pull the sensor data.
   - Interaction module of the sensor manager plays an important role in further forwarding the sensor data to a particular instance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

3. 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 , 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.

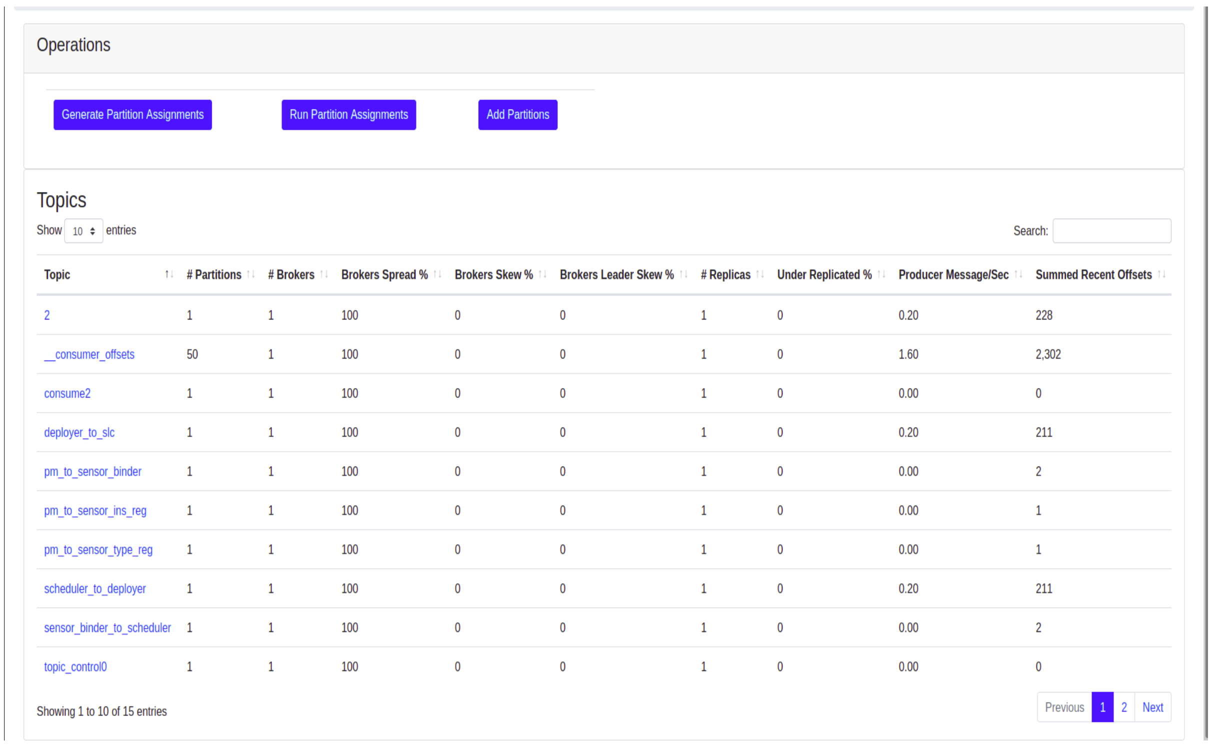
Different topics created-



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

View in the kafka manager-



Operations

Topics

Show 10 entries Search:

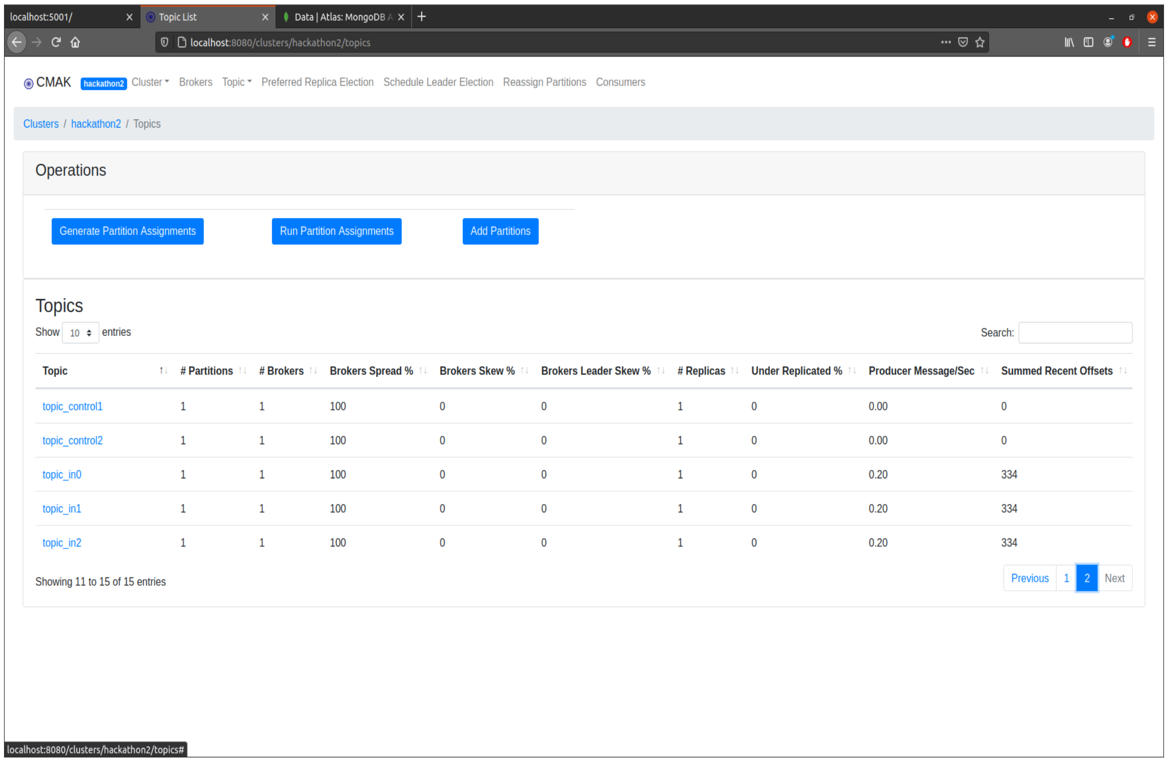
Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas	Under Replicated %	Producer Message/Sec	Summed Recent Offsets
2	1	1	100	0	0	1	0	0.20	228
__consumer_offsets	50	1	100	0	0	1	0	1.60	2,302
consume2	1	1	100	0	0	1	0	0.00	0
deployer_to_slc	1	1	100	0	0	1	0	0.20	211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

4. 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 the request 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 to a sensor based on output of an application and runs the algorithm on the sensors to control it.

5. 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 instance.
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.

Team Contributions:

Team 1:

- Kafka, MangoDB
- Bootstrap
- Parsing(parse all config files) validating and user view(admin view) (Set Api before user view) , registration phase.
- Notification Manager → msg, email
- Action Manager → sets the control of a sensor

Team 2:

- Server Life Cycle
- Service Life Cycle

Team 3:

- Deployment Manager
- Sensor Binder
- Scheduler
- Load Balancer
- Application level Monitoring
- Service Life cycle

Team 4:

- Sensor Simulation.
- Registration - SensorType,SensorInstance.
- Sensor Manager - get sensor data from sensors