IAS - Group - 7
IOT Based Application Platform
Group Design Document

1. Introduction to the Project

The project deals with building an IoT platform that involves putting all the necessary IoT 9 technology stack components required to work on an IoT solution. The stack components include hardware, firmware, IoT data storage, backend services such as monitoring and fault tolerance and communication channels.

This IOT platform will be used to develop, build and deploy the IOT based applications. The platform will provide sensors and an interface to communicate with them to the applications deployed on the platform. User need not worry about the underlying configurations of the sensor, communication mechanisms, load balancing, node management etc and other essential features of application development and deployment. User will be able to do deploy instances of the application he/she wants to use, using minimal effort. User just submits the request to deploy the application, or use an existing instance and the platform will take care of all the processes necessary to make the application usable by the user.

2. Use Cases/Requirements

The developed platform supports IoT related applications. In order to make these applications run on the platform, following scenarios can be considered:

There should a common point of interface where app developer, configurer, user can provide their part of information. This common point is Application Manager

Application Manager, UI, Notification manager and Sensor data binder

- Authentication module will verify credentials of all actors.
- Platform configurer will add sensors and controllers instances to the platform.
- Sensor data binder will bind sensors and controllers to each suitable application.
- Application manager will validates format and structure of different files uploaded.
- Application developer will upload required sensors information, application code to the Platform.
- Application Manager will validate them and stores files on Repository.
- User will enter his/her preferences where and when to deploy the application.
- Application manager will send those details further to deploy application.
- At the end Application will be deployed on nodes and Notifications from application will be sent to notification page, from where user can read them.

The application uploaded need to be scheduled in order to run it. This can be achieved by passing the application to a scheduler.

Scheduler

- The scheduler will receive application instance id from application manager. Based on this id, scheduler will fetch the scheduling info from the app repository like start time, end time, duration, isRepetition required and interval of repetition (if required).
- The application instances are placed in the priority queue as per their start times
- The applications are popped out of priority queue as per their scheduled time and a notification is sent to the deployer of deployment manager to start the application.
- Information sent to deployment manager are: app instance ID, a command to start/kill the app, whether the application is to be run on a standalone machine or shared machine

The scheduled application needs to be run. This provision is provided by deployment manager

Deployment Manager

- The deployment manager receives the request from the scheduler to start/stop the app.
- It then contacts with the node manager to get active nodes.
- Load is balanced among the nodes and the given application gets deployed thereafter.

The deployed application has to run on some machines depending on the configuration provided by configurer. A node manager can be used here.

Node Manager

- This module is used to manage and allocate new nodes (virtual machines) as and when required. It provides a node details along with node id, when a node request is made.If it is unable to satisfy the request a new node is created. It also manages the status of nodes, i.e. active, free or down.

Finally, sensor data has to be fetched and provided to the application in order to run the application. This can be achieved with the help of a Sensor Manager

Sensor Manager

- Sensor manager create topics in kafka for sensor instances and forward the data from sensors to respective partition of topics from where application can consume it accordingly.

Demo Application:

Application is built to provide transport facility using buses.

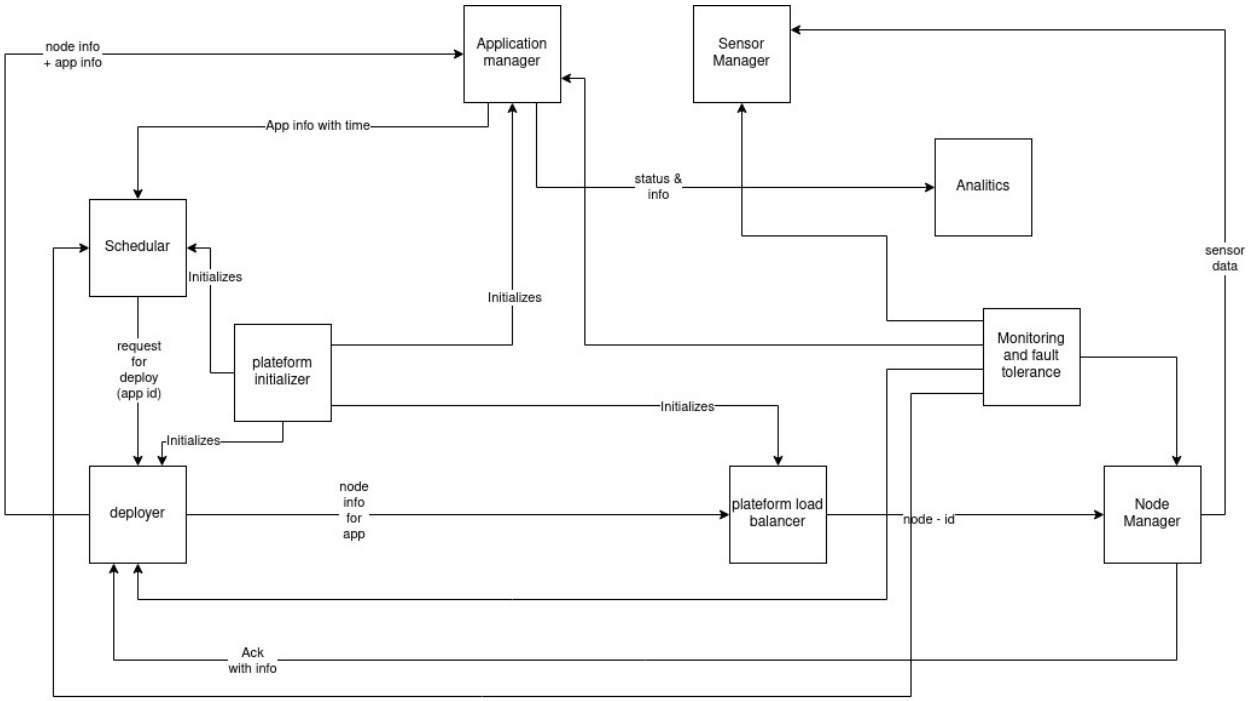
- ➔ Sensors:
  - GPS: Sends placeholder-id coordinates. Each bus is installed with one such sensor, one sensor is at IIIT-H campus and each police barricade has one such sensor.
  - Biometric: Sends placeholder-id person-id (it will be unique on place-id level)
  - Temperature: Sends current temperature of the place
  - Light Sensors: Sends Lux Level of the place
- ➔ Controllers:
  - Lights
  - ACs
  - Buzzers
- ➔ Cases:
  - Whenever someone boards a bus, he/she will do biometric check-in. Fare will be calculated based on distance multiplied by fixed rate that information(Bus-id, Passenger-id and Fare) will be sent to guard.
  - If temperature is more than a threshold switch on the air conditions or lighting is lower than a lux level switch on the lights.
  - 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 1 minute.
  - Also, 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 containing information of bus and notifying them.

3. Project Test Cases:

- Application Package/Zip validation testing
- Application Deployment Testing
- Sensor type configuration testing
- Sensor instance registration testing
- User authentication testing
- API calls testing
- Fault Tolerance testing

4. Solution design considerations

4.1 Design big picture



4.2 Environment to be used

- OS: Linux

4.3 Technologies to be used

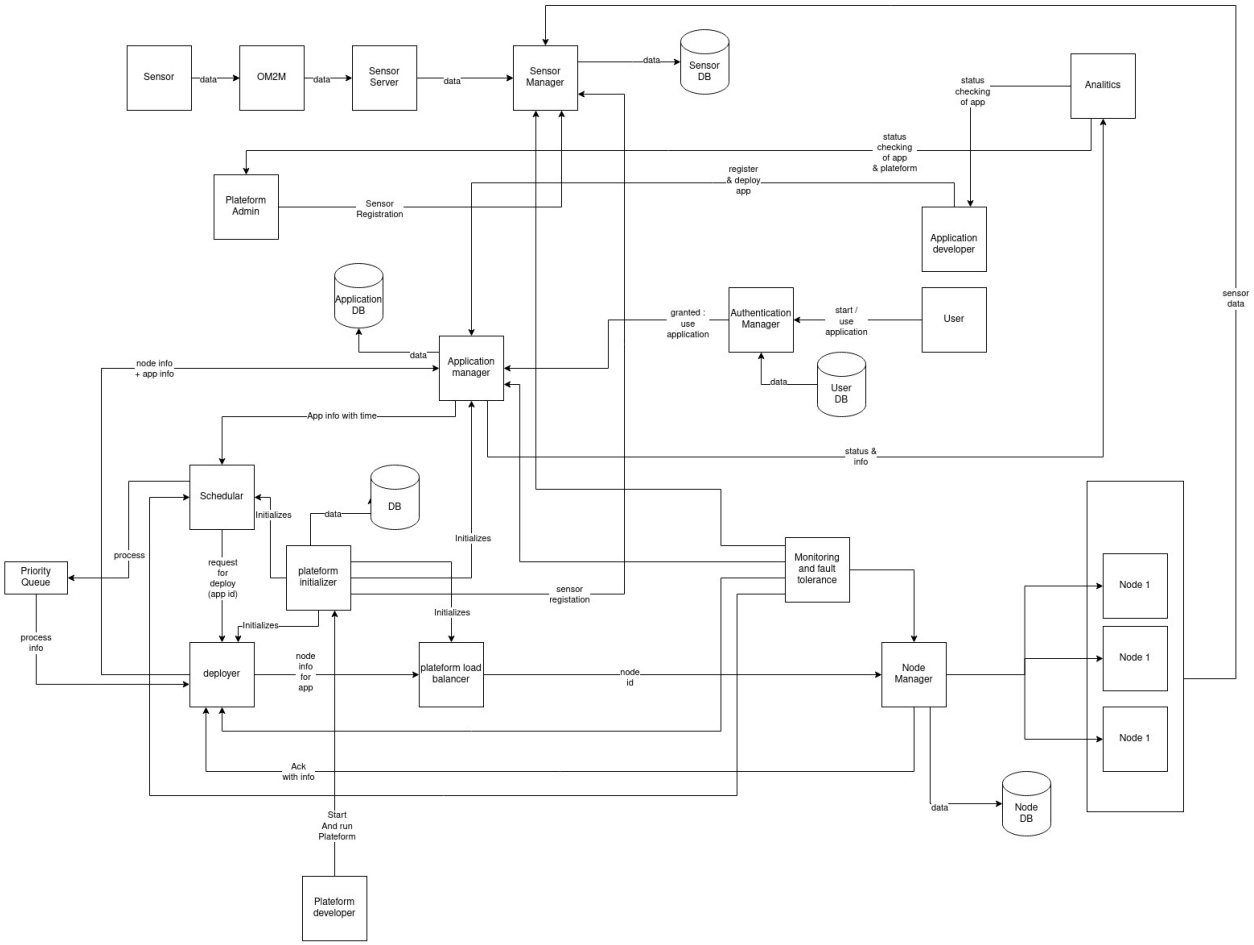
- **Python framework** - overall development
  - Presence of Third Party Modules
  - Extensive Support Libraries
  - User Friendly data structures

- **Apache Kafka** - for communication
  - Scalable
  - Fault Tolerant
  - Publish-Subscribe Model

- **MySQL** - for database
  - Data Security
  - High Performance
  - On demand scalabilty

- **Docker** - for platform independence and portability
    - Consistent and isolated environment
    - Mobility
    - Easy collaboration

4.4 Overall system flow & interactions



4.5 Approach for communication & connectivity

- Kafka as Message Queue
- Publisher and Subscriber Model

4.6 Registry & repository

- Platform Database
- User Database
- Application Database
- Node Database
- Monitoring Database

4.7 Scheduler

- This component is responsible for scheduling of various jobs, and stopping them at the end time specified.
- Schedulers take config files from the repository, as well as from the UI.
- It validates the config files, and then sends the jobs to the deployer at the specified time, to be deployed at a node.

4.8 Load Balancing

- Load Balancer primarily communicates with the Deployer and Node Manager.
- On the request of the deployer, it retrieves a list of active nodes, on each of which it calculates the load, and gives the deployer the node with minimum load.

4.9 Interactions between modules

- **Application Manager and Scheduler**
    - Application Manager will provide Scheduler with the application and the metafile of the application, where information related to scheduling are present (start and end times, job type etc)
- **Application Manager and Sensor Manager**
    - Sensor manager will provide sensor's information to the application manager which will be stored in the application database along with other application information.
- **Application manager and analytic module with Deployer**
    - Deployer will give node status and information to application manager which help to show analytic related information through analytic modules.
- **Scheduler and Deployment Manager**
    - In order to start the application, the scheduler will contact the Deployer in the Deployment Manager. A list of such jobs will be placed in the job queue by the scheduler and retrieved by the Deployer. In turn, the Deployer will provide the Scheduler with a unique identifier(application id) for the running instance of the application.

- **Deployment Manager and Node Manager**
    - The Load Balancer part of the Deployment Manager will contact the Node Manager in order to get a list of active nodes. After getting this list from

the Node Manager, the Load Balancer will calculate the load of each node using Load Balancing Algorithm.

- A node with lowest load will be selected. The Deployer will contact the Node Manager with the selected node on which the application with application id is to be run.

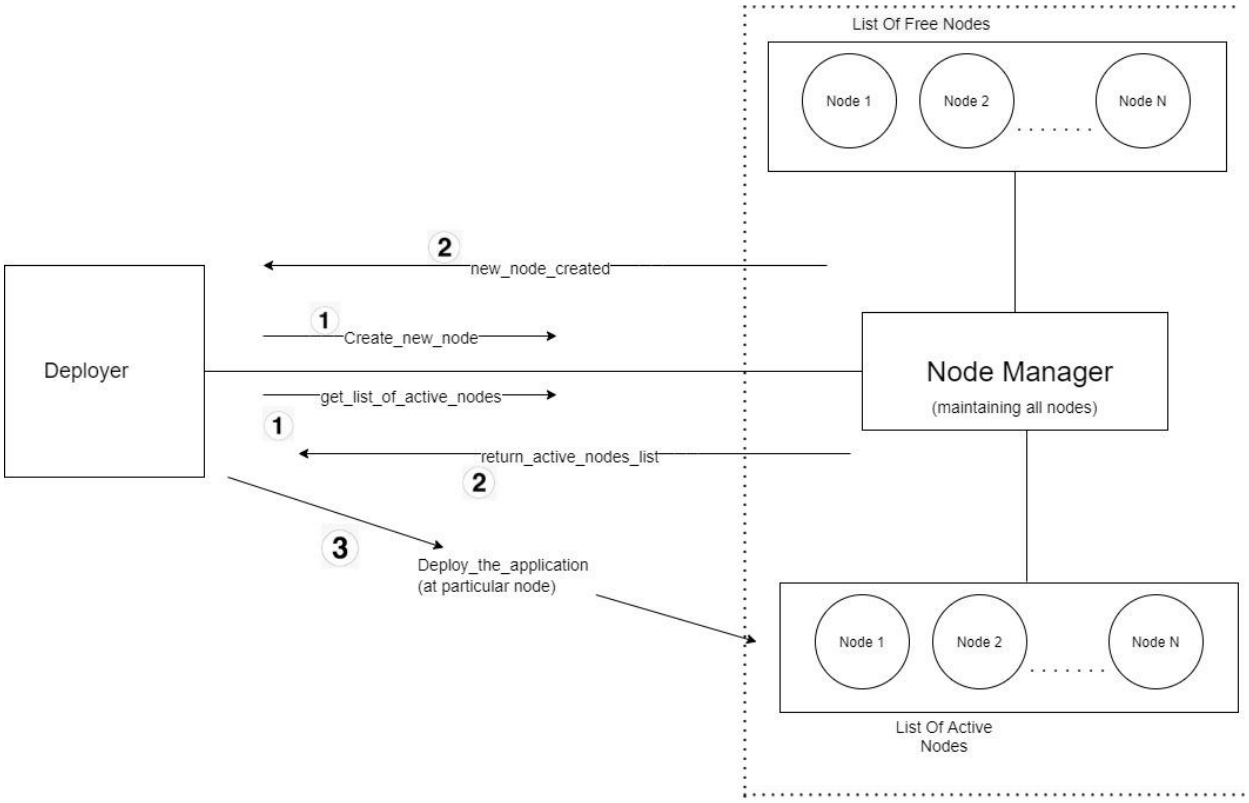


Figure: Interaction between Deployer and Node Manager

- **Sensor Manager and Nodes**
  - An application instance running on a particular node may wish to communicate with the sensor manager to send/receive data streams to/from a particular sensor or multiple sensors.
- **Monitoring and Fault tolerance and other modules**
  - The monitoring and fault tolerance module interacts with all other modules like Scheduler, Sensor Manager etc. For all these modules, periodic monitoring for their proper functioning will be done. If any discrepancy or fault is detected in the working of these modules, the fault tolerance module's instance related to that module gets triggered, and an alternate node is allocated to run that module.

4.10 Server and Service Life Cycle

- **Server Life Cycle Manager** manages the life cycle of the running server. It can be considered as a node manager that keeps the information about available and free nodes at any instance if time. Whenever the service life cycle manager requests for a new node or a list of active nodes, the server life cycle manager provides the same.

- **Service Life Cycle Manager** acts as a deployment manager. It receives details of service from the scheduler. It communicates with server life cycle manager for getting the address of the node/server on which the service is to be run. After getting the address of the node, it then deploys the service on that node address.

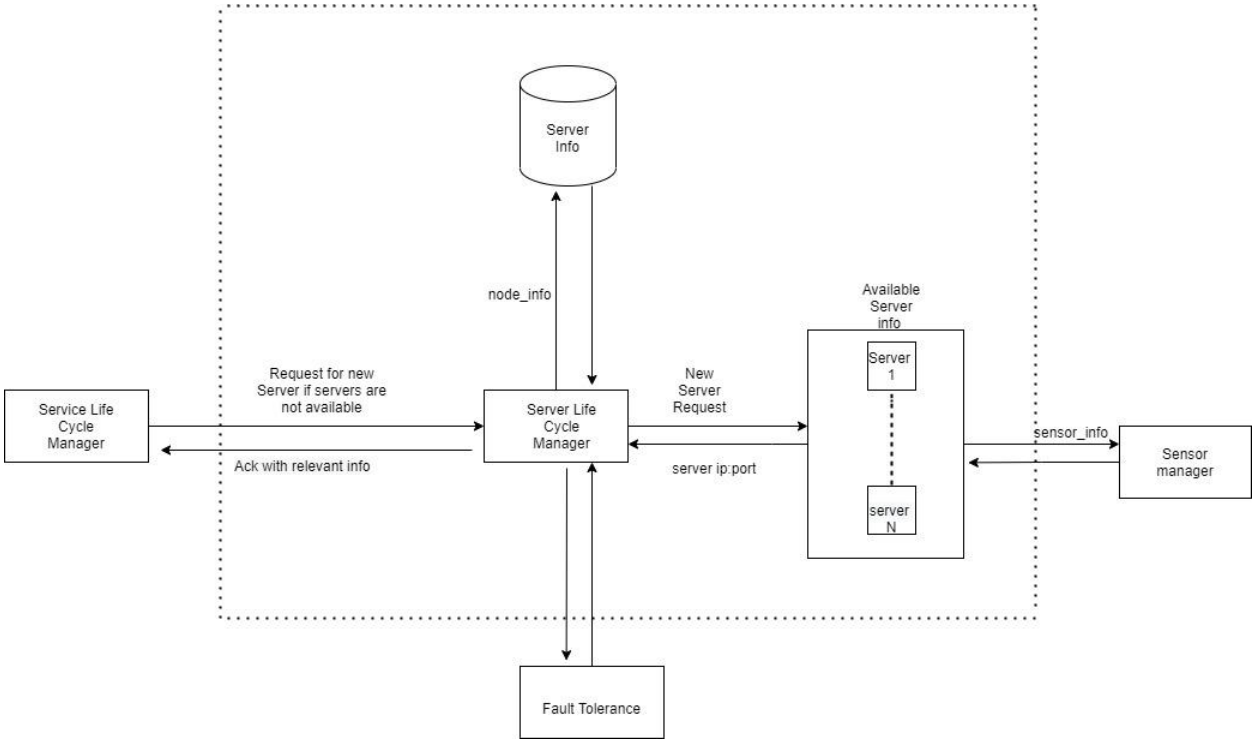
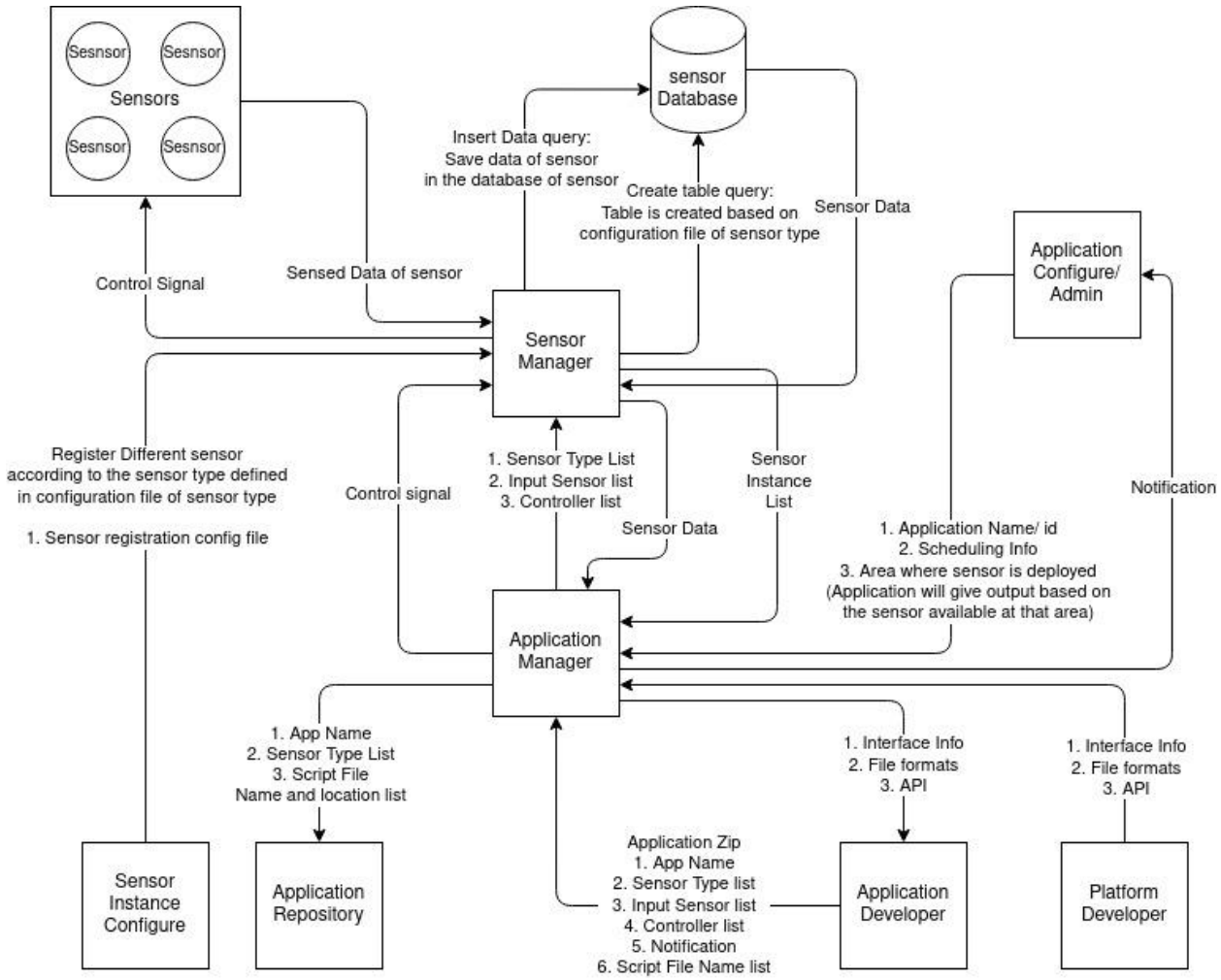


Figure: Server and Service Life Cycle

5. Application Model and User's view of system



Step 1:

Application manager will provide following to Application developer based on which application developer will develop his/her application:

- Interface Info
- File formats
- API

Step 2:

– Application developer tries to upload its application by providing following file

- App Name Configuration File
- Sensor Type Configuration File
- Input Sensor Configuration File
- Controller Configuration File
- Notification Configuration File
- Script File Configuration File

Step 3:

– Application Manager
    * It will receive the request of application developer
    * Save files received in request
        · App Name Configuration File
        · Script File Configuration File
    * Forward the Configuration file of Sensor type to sensor manager to define schema
    for sensor
        · Sensor Type Configuration File
        · Input Sensor Configuration File
        · Controller Configuration File

Step 4:

– Sensor Manager: Defining Sensor type
    * Receives Configuration file of sensor type
    * From that configuration file it will check if there exist any same type sensor schema
        If No
        · It will define the sensor type schema with the help of configuration file of sensor schema
        · Also add sensor type and its information in sensor type information list

Step 5:

– Application Configurer
    * Will deploy/register sensor by uploading
        sensor registration configuration file

Step 6:

– Sensor Manager: Sensor registration
    * Based on the sensor registration file it will create a sensor instance and add it to the respective sensor type.

Step 7:

– Application Admin/ End User
    * Will use application by providing following info
        · Application Name/ id
        · Scheduling Info
        · Area where sensor is deployed(Application will give output based on the sensor available at that area)

Step 8:

– From Area defined in request
    * Application Manager will locate all the sensors which will be used in the application

Step 9:

– After locating sensor following will be done
    * Scheduling
    * Sensor binding

        * Configure according to location
    – For Ex.
        * Locate sensor
        * create instances of sensor
        * Sensor instance binding
        * Setup the run mode
            · Run it continuously
            · Run it on scheduled time
            · Run it on demand

Step 10:

    – Running Process
        * Read from sensor
        * Send control signal to Controller
        * Notify
        * Repeat step i until some exit condition

- Configuration Files

- **Sensor Instance Configurer Configuration.json**

```
{
    "sensor_type": "sensor_type_name",
    "sensor_location": {
        "street": "VC bunglow",
        "city": "Rajkot"
    },
    "sensor_instance_count": 10
}
```

- **Sensor Type Configurer Configuration.json**

```
{
    "name" : "Sensor_type_name",
    "company" : "Samsung",
    "model_no" : "sam_ac",
    "data_rate" : 10,
    "data_rate_unit" : "kbps",
    "data_fiels_count" : 2,
    "data_fiels" : [
        {
            "data_fiels_index" : 0,
            "data_field_name" : "Speed",
            "data_field_type" : "int64"
        },
        {
            "data_fiels_index" : 1,
            "data_fiels_name" : "temprature",
            "data_field_type" : "int64"
        }
    ],
    "methods" : {
        "name" : "get_data",
        "input_parameter_count" : 1,
        "input_parameter" : [
            {
                "data_fiels_index" : 0,
                "parameter_name" : "sensor_instance",
                "parameter_data_type" : "string"
            }
        ],
        "return_parameter_count" : 2,
        "return_parameter" : [
            {
                "return_parameter_index" : 0,
                "return_parameter_name" : "Speed",
                "return_parameter_type" : "int64"
            },
            {
                "return_parameter_index" : 1,
                "return_parameter_name" : "temprature",
                "return_parameter_type" : "int64"
            }
        ]
    }
}
```

- **Controller Type Configurer**

  Configuration.json

```
{
    "name" : "controller_type_name",
    "company" : "Samsung",
    "model_no" : "sam_ac",
    "input_data_fiels_count" : 2,
    "input_data_fiels" : [
        {
            "data_fiels_index" : 0,
            "data_field_name" : "Speed",
            "data_field_type" : "int64"
        },
        {
            "data_fiels_index" : 1,
            "data_fiels_name" : "temprature",
            "data_field_type" : "int64"
        }
    ],
    "methods" : {
        "name" : "give_input_data",
        "input_parameter_count" : 2,
        "input_parameter" : [
            {
                "input_parameter_index" : 0,
                "input_parameter_name" : "Speed",
                "input_parameter_type" : "int64"
            },
            {
                "input_parameter_index" : 1,
                "input_parameter_name" : "temprature",
                "input_parameter_type" : "int64"
            }
        ]
    }
}
```

- **Controller Instance Configurer**

  Configuration.json

```
{
    "controller_type": "controller_type_name",
    "controller_location": {
        "street": "VC bunglow",
        "city": "Rajkot"
    },
    "controller_instance_count": 2
}
```

- **Controller Developer Configurer**

  Appname.json

```
{
    "app_name" : "MyApp"
}
```

- **controller.json**

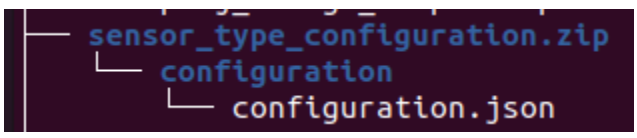
```
{
    "controller_instance_count" : 3,
    "controller_instace_info" : [
        {
            "controller_instace_index" : 0,
            "controller_instace_type" : "controller_type_ac"
        },
        {
            "controller_instace_index" : 1,
            "controller_instace_type" : "controller_type_fan"
        },
        {
            "controller_instace_index" : 2,
            "controller_instace_type" : "controller_type_fan"
        }
    ]
}
```

- **sensor.json**

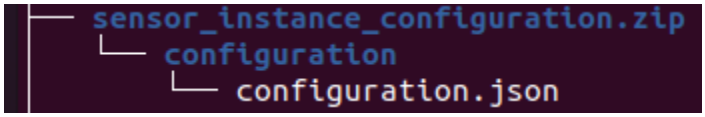
```
{
    "sensor_instance_count" : 3,
    "sensor_instace_info" : [
        {
            "sensor_instace_index" : 0,
            "sensor_instace_type" : "sensor_type_ac"
        },
        {
            "sensor_instace_index" : 1,
            "sensor_instace_type" : "sensor_type_fan"
        },
        {
            "sensor_instace_index" : 2,
            "sensor_instace_type" : "sensor_type_fan"
        }
    ]
}
```

- Structure of the files

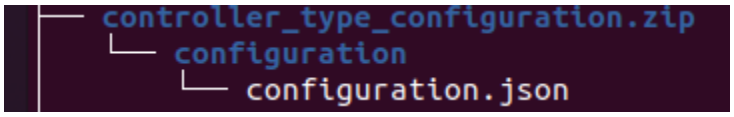
- **Sensor Type Configurer**



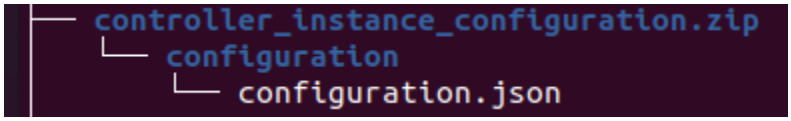
- **Sensor Instance Configurer**



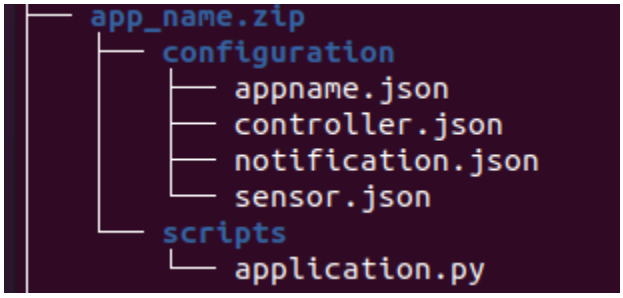
- **Controller Type Configurer**



- **Controller Instance Configurer**



- **Controller Developer Configurer**



- **Application Deployment and Setup**
  - Application developer submits the script files along with the configuration files in a zip file.
  - Then Application developer submits the sensor instance configuration file based on which the application details are configured and stored.
  - Then the application is validated based on the directory/files format specified by the platform.

- Starting and stopping the application

- By calling start app function from script file

```python
import sensor_api
# All the necessory methods will be defiend here
    # methods signature & logic of get sensor data
    # methods signature & logic of give controller data

class myapp(platform_interface):

    # necessor methods which application developer has to make
    def start_app(self, sensor_id_list):
        # sensor_id_list : sensor_ids in sequence stated in configuration/sensor.json file

        # logic of app
            # can take data of sensors using methods signature defined in sensor_type.zip/configuration/
            configuration.json file
                # input parameters info will be defnied there
                # return parameters info will be defnied there


            # can send controll signal to controllers using methods signature defined in controller_type.
            zip/configuration/configuration.json file
                # input parameters info will be defnied there

        pass

    def force_shutdown_app(self):
        pass

```

- Application artifacts

- Service files
- Packaging
- Configuration

- Configuration Files

- App_configuration.json
- Sensor_type_configuration.json
- Sensor_instance_configuration.json
- App_deployment_configuration.json
- application_location_configuration.json
- notification_configuration_file.json

- Application zip file containing the source code - app.zip

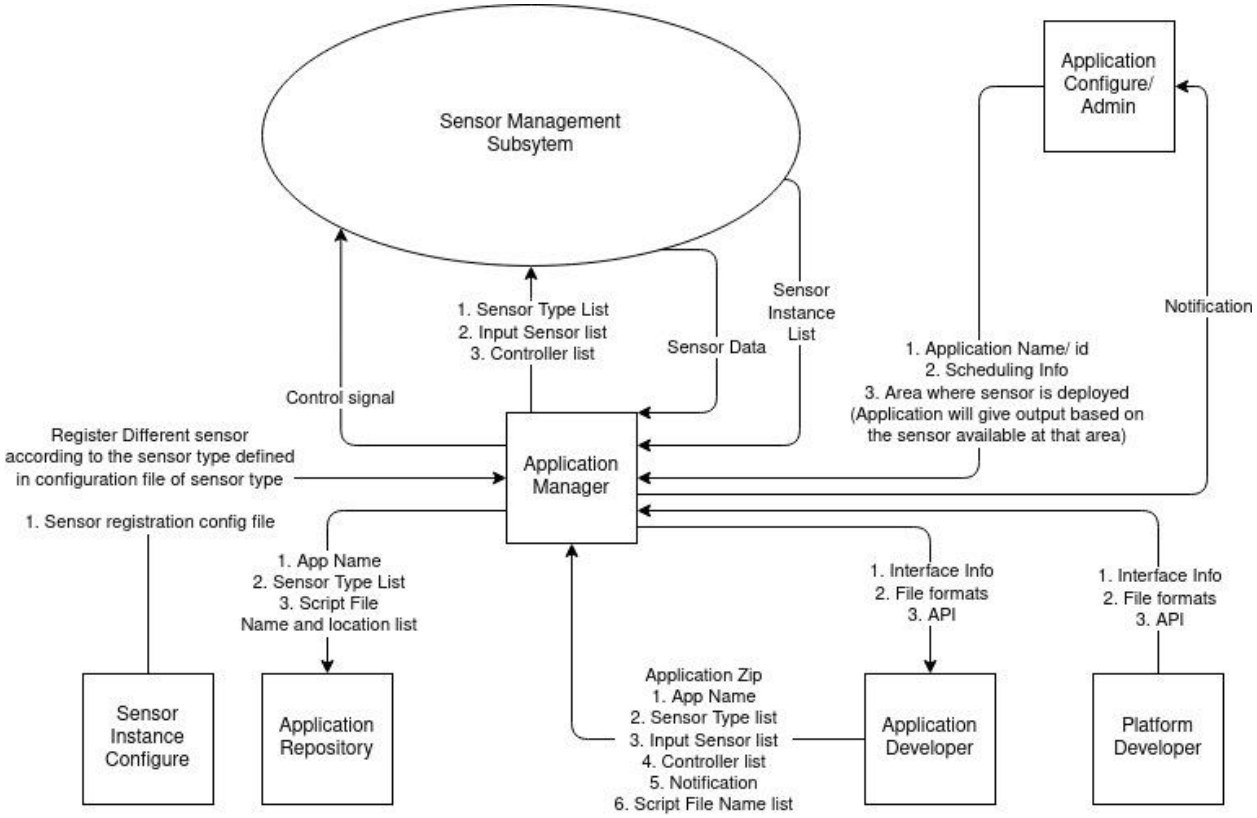
API's

- get_sensor_data(sensor_id, location)
- send_notification_to_user(user_id, notification_type)
- control_sensor(sensor_id,control_type, control_data)

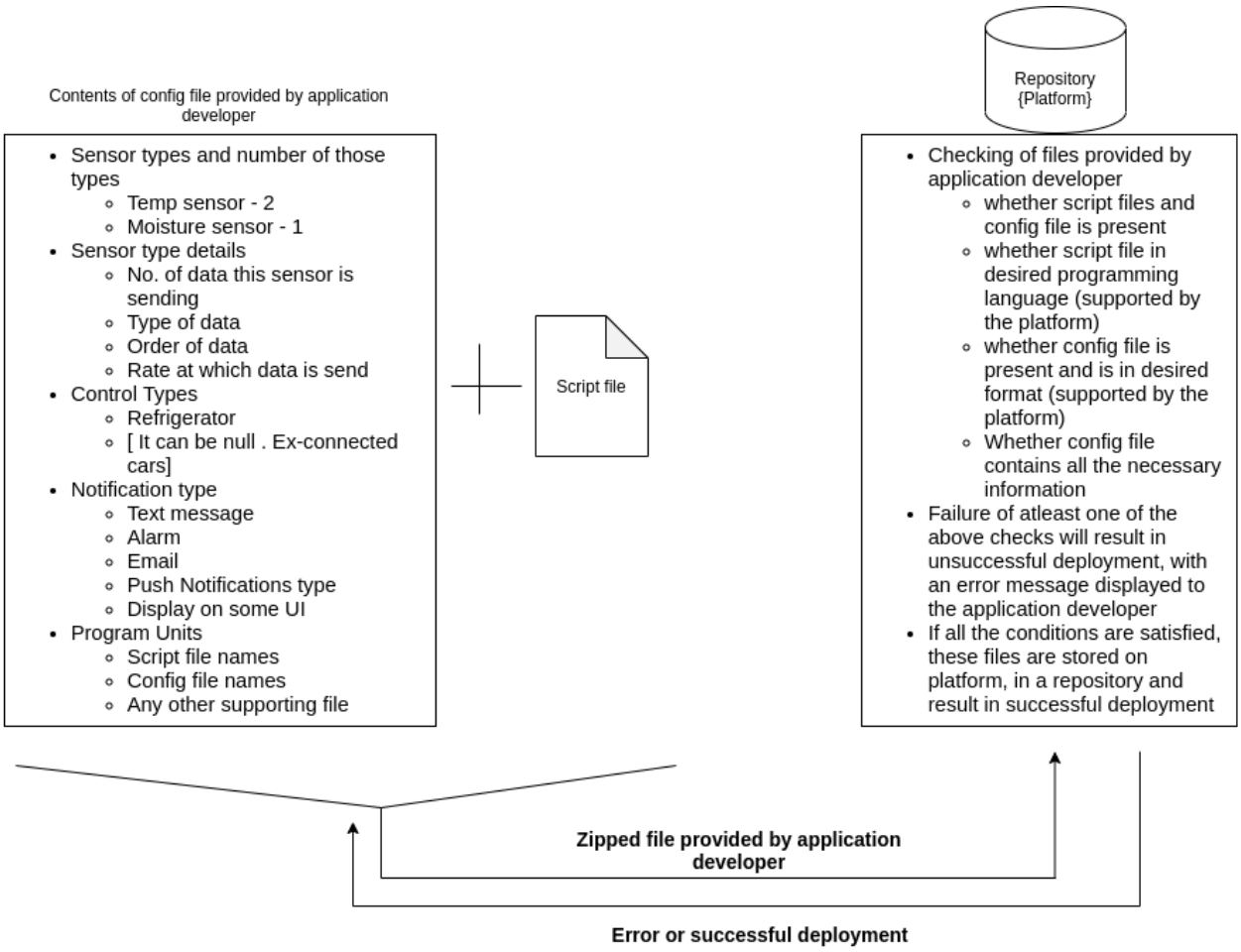
- Structures

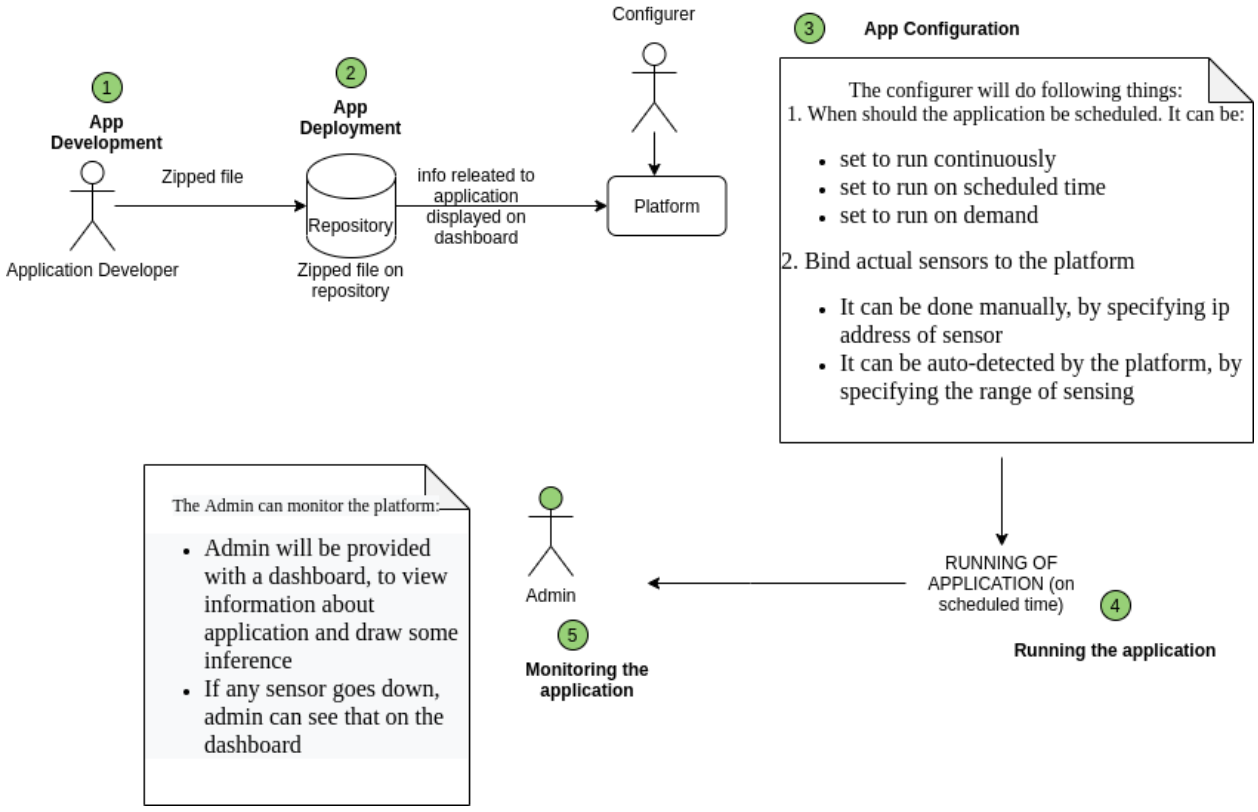
- sensor_type_definition
- sensor_definition

- User admin interactions



6. Interactions & Interfaces



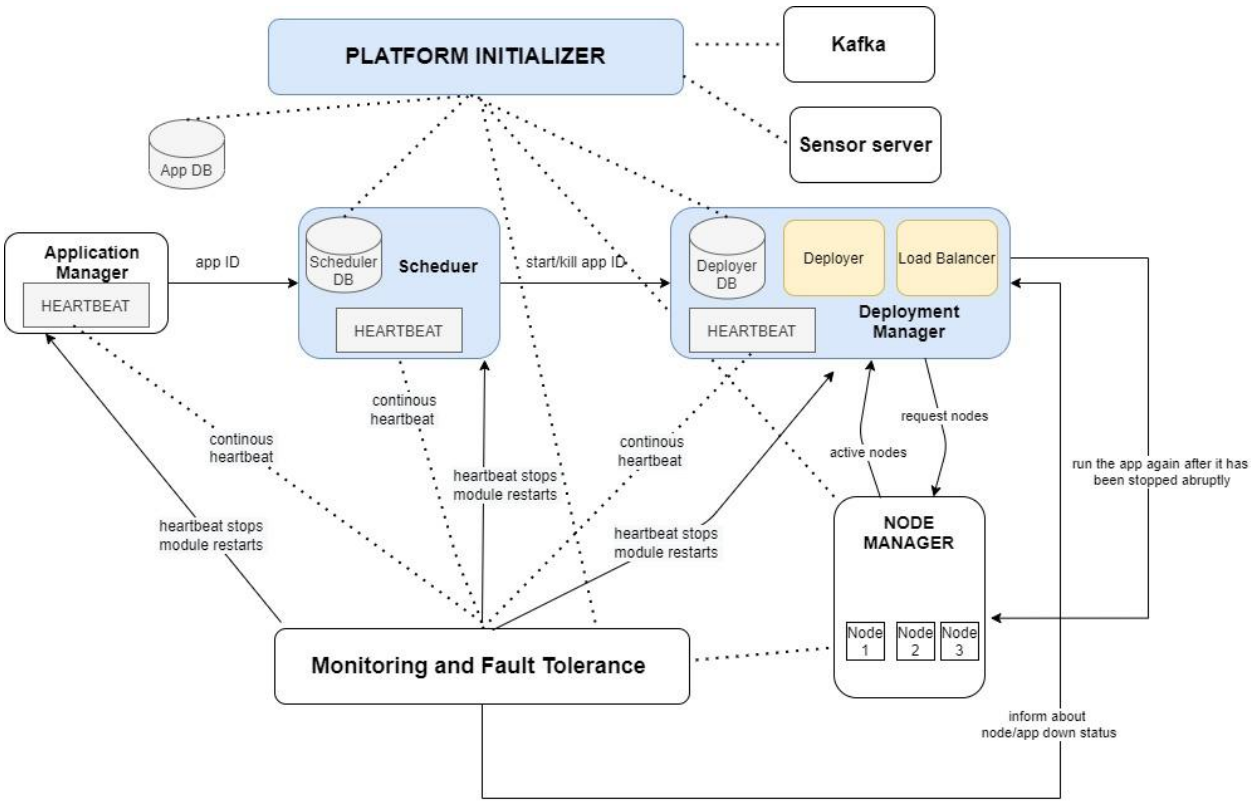


- **APIs**
  - **Get_list_of_active_node**
    - Return the list of active node containing <ip,port> details
      - 127.0.0.1 8082:127.0.0.1 8085:127.0.0.1 8086
  - **create_new_node**
    - Creates a new node and sends its ip and port to the deployer.
      - 127.0.0.1 8089
  - **get_sensor_data()**
    - It takes the **sensor_idx and app_id** as input and resolves the sensor_idx with the physical sensor id. Then it fetches **the current data** from the kafka queue and sends it to the application.
  - **get_stream_data()**
    - It takes the **sensor_idx and app_id and the number of data instances** as input and resolves the sensor_idx with the physical sensor id. Then it fetches the **said number of data** from the kafka queue and sends it to the application in the form of a **list**.
  - **set_controller_data()**
    - This API is designed to update the state of a particular controller that is bound to some application instance.

7. The modules that the four team will work on:

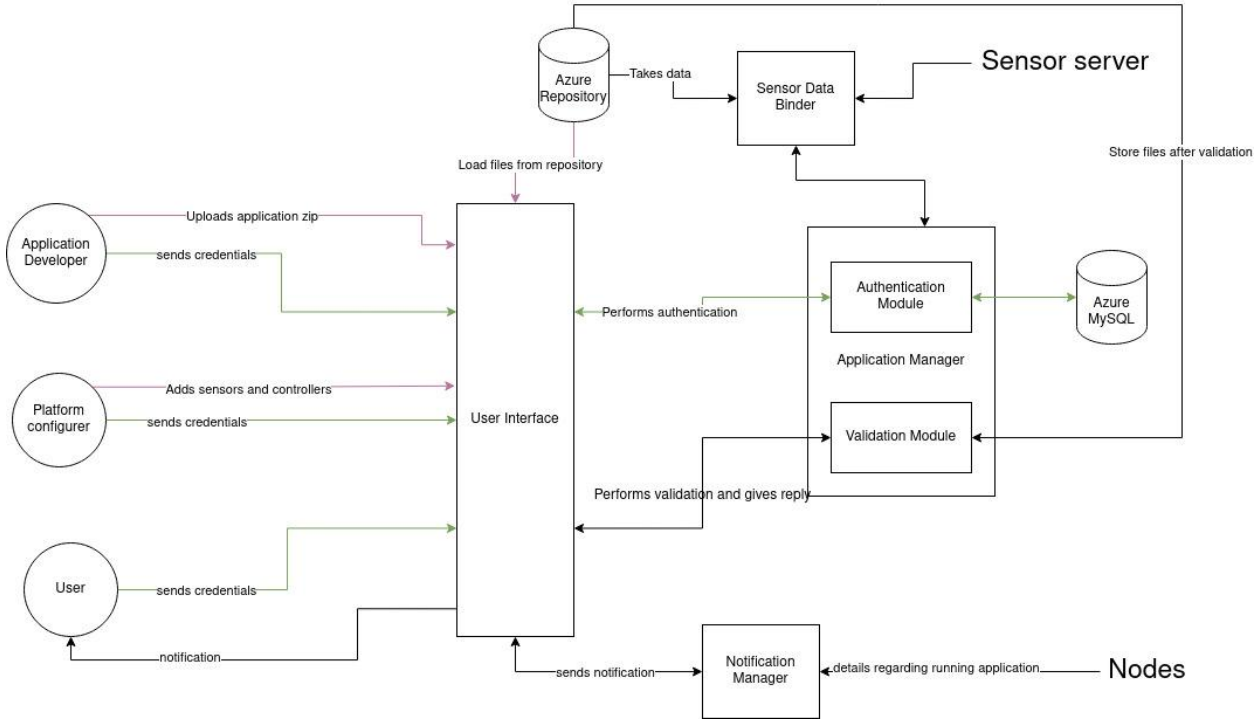
Scheduler, Deployer ,Load Balancer and Platform Initialisation

- This part is Responsible for scheduling the jobs, both from the repository, and from the UI
- This part will validate scheduling config files, and will run and stop jobs at specified times.
- Apart from this, it also keeps the load on all the nodes balanced, with the help of load balancer and scheduler.
- The platform initializer initializes the client machines so that they are able to run the platform components. The platform components are independent of the number of machines the client has, and can run all on one machine, or even if they are distributed.
- After installing the platform clients, the initializer decides which service to run on which machine, and deploys the service on the machine, with the help of bootstrapper.



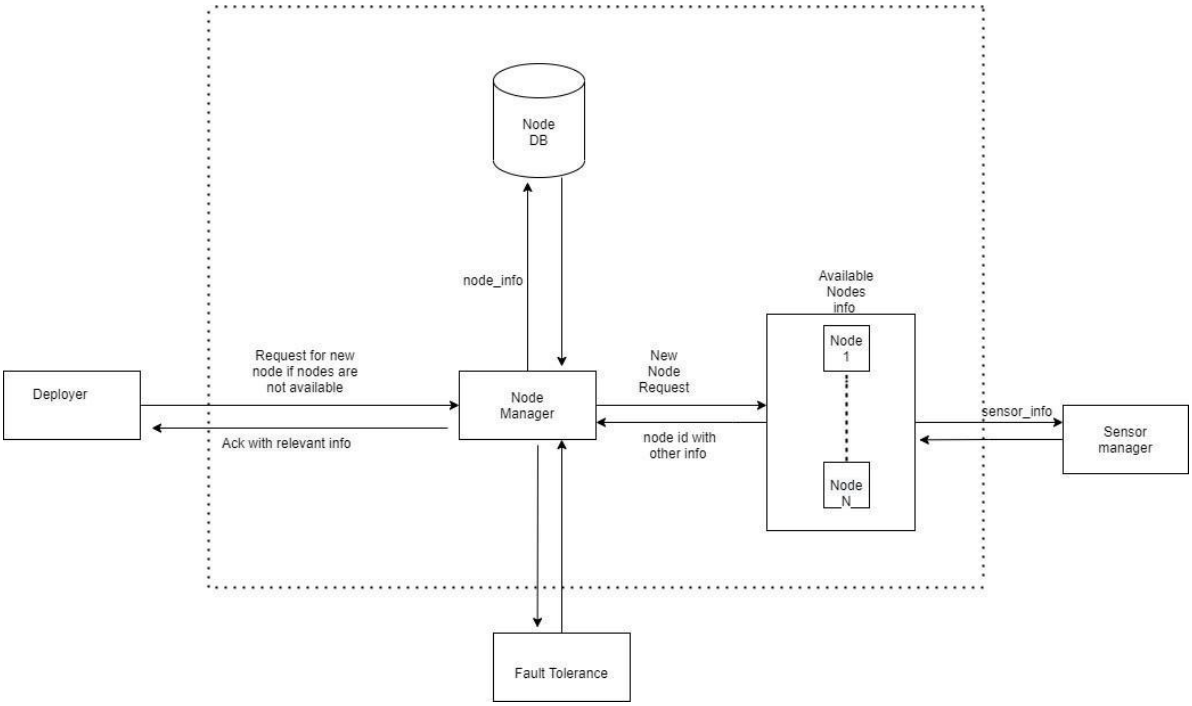
Application Management, UI, Notification manager and Sensor Data Binder

- This part is Responsible for authenticating the users and validate zip whiles which are uploaded by different users.
- Also, all communication will be done through UI.
- Apart from these, we have built notification manager which manage notification for different users.
- Sensor data binder will bind sensor to application according to location and sensor data type which is needed in application.

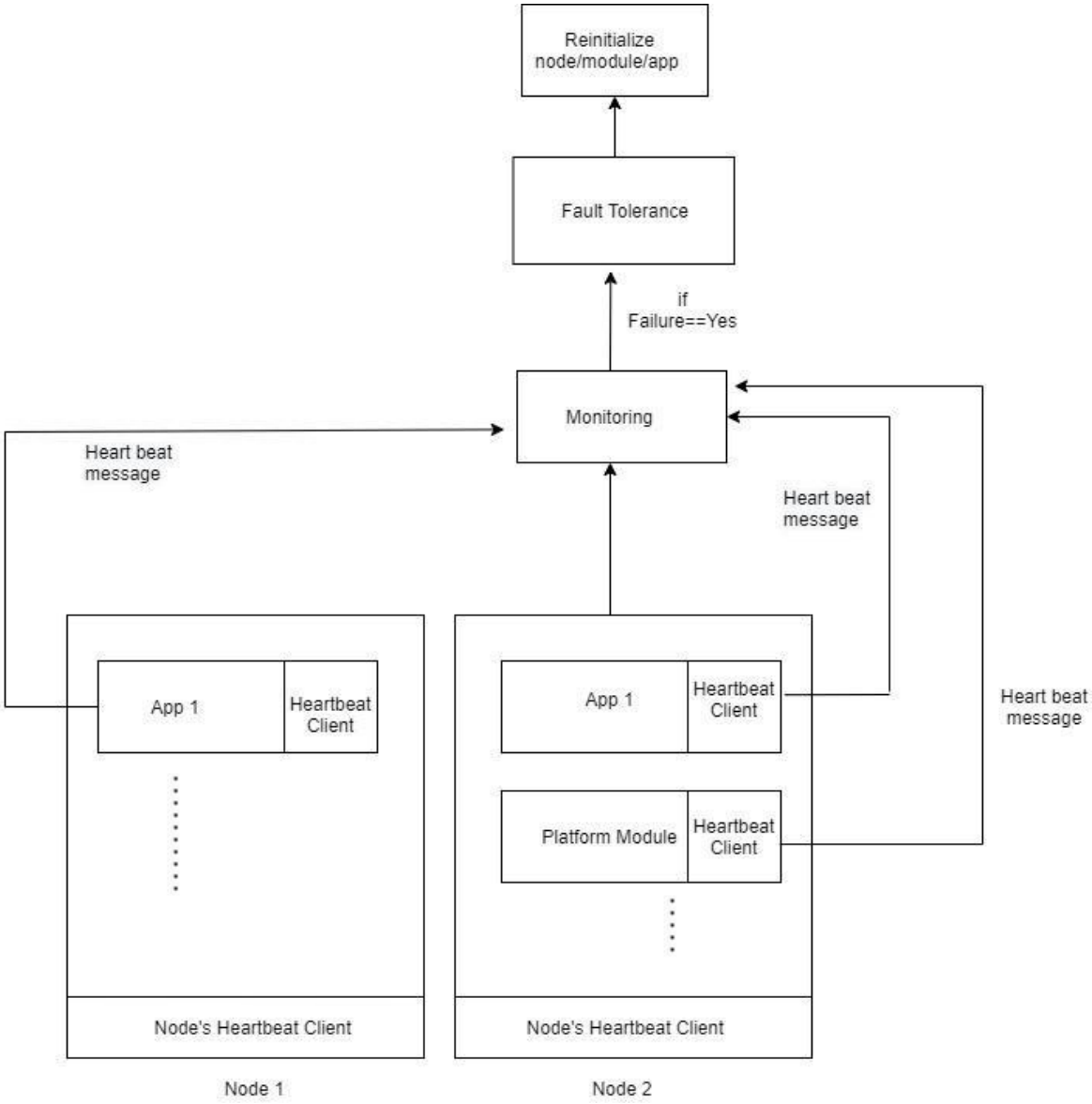


Node Management and Monitoring & Fault Tolerance

- Node Management modules manage all the nodes.
- Monitoring & Fault Tolerance module monitors and reinitializes the nodes/modules/ application on failure.



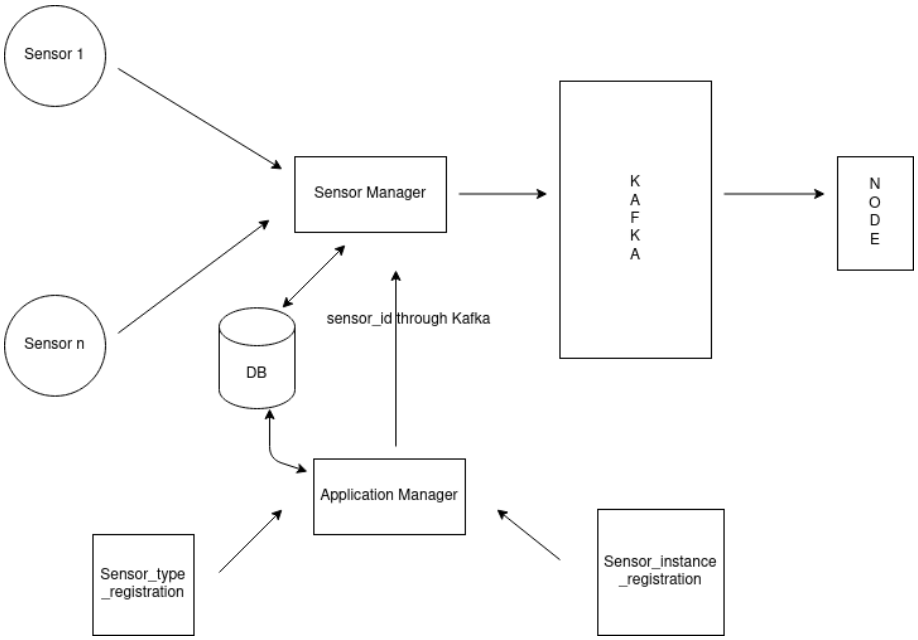
Nodes and Node Manager module



Monitoring and Fault Tolerance

Sensor Management

- Sensor server analyse sensor data and convert to clean and specified format.
- Sensor manager helps in binding sensor data to the specified deployed application.


Sensor 1
Sensor Manager
K
A
F
K
A
N
O
D
E
Sensor n
DB
sensor_id through Kafka
Application Manager
Sensor_type
_registration
Sensor_instance
_registration

8. Persistence:

Data related to application and platform subsystems are maintained in Relational Database. If any discrepancy or fault is detected in the working of any of the modules, the state can be retrieved from the database and fault tolerance module will re-initialize that particular module again.