Project Requirements For IoT Based Application Platform

Internals Of Application Server

Submitted By : Group 6

Contents

1. Introduction

1.1. Project Introduction

Internet of Things is a technological revolution which provides the vision of a connected world of Things. IoT platforms (often referred to as IoT middlewares) are the support software that connects everything in an IoT system.Variety of platforms are now a days available that can support entire development to deployment of IoT applications and systems An IoT platform facilitates communication, data flow, device management, and the functionality of applications. This distributed platform will be able to develop and deploy different functionalities.

IoT platforms help:

- Connect hardware.
- Handle different communication protocols.
- Provide security and authentication for devices and users.
- Collect, visualize, and analyze data.
- Integrate with other web services.

Platform: When you are developing an application, Platform is one which allows you to deploy and run your application. A platform could be a hardware plus software suite upon which other applications can operate. Platform could comprise hardware above which Operating system can reside. This Operating system will allow applications to work above it by providing the necessary execution environment to it.

Middleware: Middleware provides a wide variety of services to the applications from outside. So the application is not bound to use its all services but only uses the necessary set of services. It acts as a middle level agent between the service provider and service consumer. It allows communication among the applications that come from different vendors or runs on different platforms. It acts as a mediator between applications of different forms. IoT middleware is a mechanism that joints different components of IoT systems together and offers smooth communication among devices and components.

1.2. Scope

The platform will provide various services to deploy the applications and connect the applications to sensors scattered across different locations based on oneM2M standard. The services provided by the platform are application manager with building and employment capabilities, logging and monitoring services, scheduler for scheduling algorithms, action manager for sending notifications and carrying out actions based on responses of applications/algorithms, authentication and authorization services and resource management services.

This platform can be used to build an IOT application which performs some action based on collected sensor data. The application will be able to connect to IOT devices remotely and perform actions on them based on computations done by service. The platform stores data of all the sensors present, their location and provides connection for the service to be executed.

2. Intended Use

2.1 Intended Use

Application developers will be able to deploy and run their applications and connect to sensors. The platform will offer services to connect to sensors and get data from sensors. The application developers would be able to schedule their algorithms/applications and choose many types of algorithms from the repository. The platform will also take care of the load balancing, fault tolerance of the applications if needed. The details of underlying communication is abstracted from the application developer or application administrator. The Administrative users of the platform can take advantage of special services like monitoring and logging modules which will help them in understanding the current state of the platform and help them in taking decisions.

2.2 Assumption and Dependencies :

- Algorithms corresponding to sensors should be present.
- All the config files should be properly formatted
- All commands should be properly formatted
- Active network connectivity should be present
- All the required libraries like communication and virtual env. libraries should be present
- All the required 3rd party services should be present on system

3. System Features and Requirements

3.1 Platform Requirement

1. **Deployment of the platform**

- Platform should be easy to deploy and manage.
- Handling of resources should be automatic and customizable.
- Dashboard service or Interface should be provided for checking the performance and health of the platform.

2. **Different Actors on platform**

- **Platform Administrator** : The platform administrator starts the platform and monitors the health of the services provided by the platform. Platform administrator also registers the sensors on the platform.
- **Application Developer** : Application Developer deploys the application on the platform.
- **End User** : The end user is going to use the services provided by the application deployed on the platform.

3. **Applications overview on platform**

- Applications should be mapped to requested sensor correctly
- Applications should be able to carry out actions or send notifications as a part of their result processing.
- Application should be able to safely recover in case of a failure.
- Applications should be able to run freely with an adequate amount of resources provided.

3.2 Functional Requirements

a) **Registering sensors** - The application developer will be able to register their sensors by uploading a config file in specified format. There will be sensor type registration and sensor instance registration done by the admin of the application.
b) **Interaction with sensors** - The application developers will interact with the sensors through the APIs provided by the platform.
c) **Development of application on platform** - The application developer will provide a zipped file containing code and configuration file.
d) **Identification of sensors for data binding** - The sensors will be identified by the type and location and other metadata provided at the time of registration.
e) **Data binding to application** - The sensors will receive the data from sensors using APIs provided by the platform.
f) **Scheduling on platform** - The developer will provide a scheduler file to schedule algorithms.
g) **Communication model** - Communicate with different other modules present in our platform and provide communication between application and sensor.
h) **Server and service lifecycle** - Service life cycle manager will dockerize the applications and provide them an isolated environment for deployment. Service lifecycle manager will monitor the services/applications running on the platform

and handle the fault termination. Service lifecycle will also fetch the logs of the application, stop the applications from executing. Server lifecycle will monitor the servers and provide fault tolerance on them.

i) **Deployment of application on platform** - The containers for deploying applications will start whenever algorithms of an application are to be executed.
j) **Registry and repository** - There will be multiple registries and repositories to store information and files related to application such as sensors registered and code of application.
k) **Load Balancing** - The platform will deploy the applications on the server where the load is less using load balancer.
l) **Interaction between modules** - Communication module is doing this work.
m) **Interaction of different actors with platform** - Platform manager, application developer and application administrator will interact with platform using dashboard and will receive notifications. End-users will interact with the platform through application if any interface is provided by the application developer and may receive notifications.

3.3 Development Model

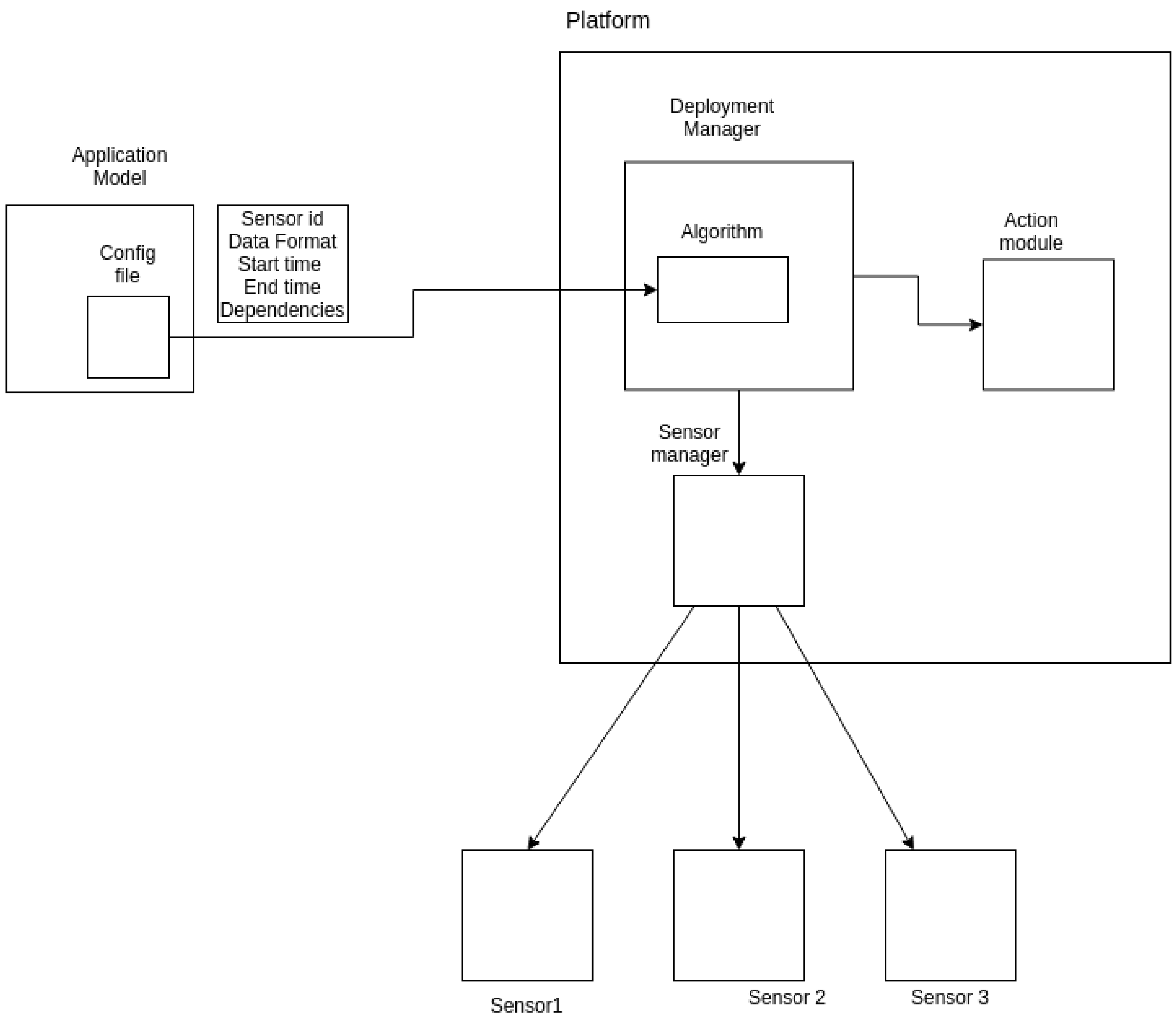
Application Development model

Application Admin Configuration files

1. Sensor Type registration Configuration File:
   Sensor id:
   Sensor Type:
   Sensor Data Type:
   Sensor Data Rate:

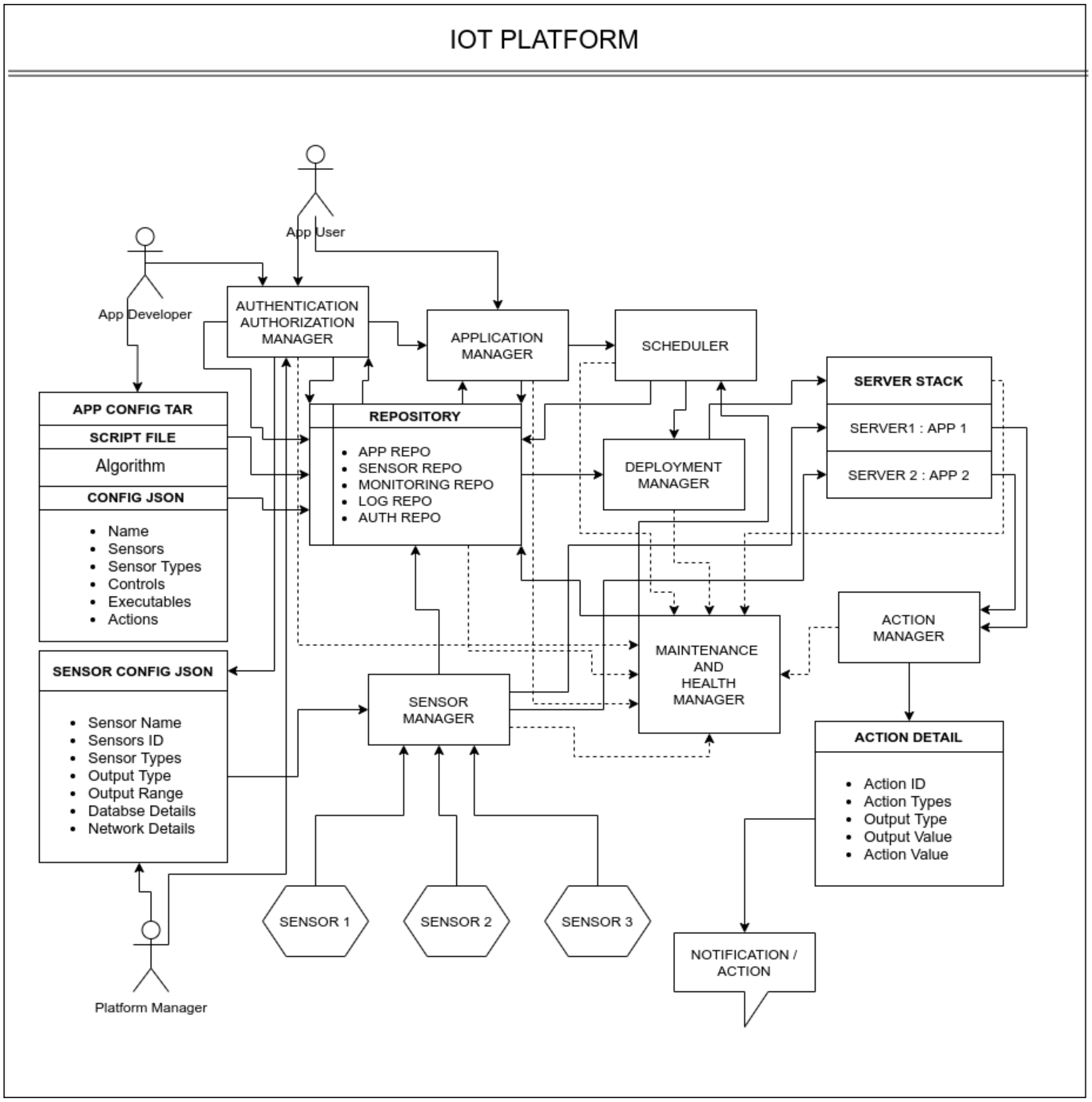
2. Sensor Instance Registration File:
   Sensor Type:
   Sensor Location:
   Sensor Purpose:
   Sensor Status:

3. Application Configuration File:
   Application Name:
   Application Dependencies:
   Algorithms:
      Algorithm Name:
      Sensors Used:
      Action:



Sensor Registration and Information flow:

Platform Administrator will provide a sensor-config which will contain sensor information of sensors which the administrator wants to register on our platform. Now our sensor is continuously in working and communicate with sensor manager and now when application developer request for any sensor detail then sensor manager will give sensor data to deployer to run that data on the specific algorithm and after running the algorithm,output of the algorithm is handled by the action handler and this way our model works.



Develop:

- ○ Understanding the platform and develop application based on a Config file containing -
    - Application Name
    - Application Dependencies
    - List of Algorithms in Applications
        - ○ Algorithm Name
        - ○ Algorithm Id
        - ○ Type and number of Sensors used by algorithm

- Action performed by algorithm

Algorithm Script -

- Contains info about all the sensors to be used based on type.
- What elements are to be controlled.
- What type format of data is needed.
- Processing and interference from the data received.
- Notification to be sent.
- First a <app_name>.zip file must be provided by application developers which contains two pieces of information like app_config.json and scripting files. The app_config.json contains information mentioned above and scripting files contain actual algorithms.
- The developed application must be compatible with our platform.

Deploy and configure:

- Now these details can be used and our sensors should be registered and then details like the running of sensors is given by the developer according to which our sensor will run and now when the app dev wants any sensor data then he/she will request for that .
- And then sensor manager will provide the corresponding data from that sensor via one-m-2-m to sensor manager and sensor manager will pass on data to the deployer which runs that on our algorithm and sends the output to the action handler, which will further process the output in a way that depends on sensor type.

Run and monitor:

- Contain info like when to run a particular sensor or whether it will run always or on schedule and monitoring of the system.
- Application will be executed on a docker instance with the dependencies running.
- Monitoring will be done on platform services and in case of a services failure the application will handle it with the fault tolerant system.

Validation

1. Health Monitoring

Sensors:

Glucometer

Brain activity monitor

Heart rate monitor

Oxy Sensor

Blood Pressure monitor

How will application developers ask for specific sensors?

Sensor location: {“Room”: 23}

Sensor Type: Sends streaming data

Algorithm: predictions for hypoglycemic

Output:
Notification: E-mail/message
Dashboard: Display on screen

Sensor location: {“Room”: 10}
Sensor Type: Sends Streaming data

Algorithm: Predictive analysis of signal using neural networks

Output:
Notification: E-mail/message
Dashboard: Display on screen

How will the sensors be registered in our platform?

1. Sensor Type registration Configuration file:
   Sensor id:
   Sensor Type:
   Sensor Data Type:
   Sensor Data Rate:

2. Sensor Instance Registration File
   Sensor Type:
   Sensor Location:
   Sensor Purpose:
   Sensor Status:

2. Smart Home

Sensors
Camera Sensor
Temperature Sensor
Light sensor
IR security sensor

How will application developers ask for specific sensors?

Sensor location: {“Block”: 23}
Sensor Type: Camera

Algorithm: Face detection algorithm

Output:
Notification: E-mail/message
Dashboard: Display on screen

Sensor location: {“Block”: 10}
Sensor Type: LUX

Algorithm: Efficient Lighting Management

Output:
Notification: E-mail/message
Dashboard: Display on screen

Sensor location: {“Block”: 10}
Sensor Type: Temperature

Algorithm: Temperature Monitoring

Output:
Notification: E-mail/message
Dashboard: Display on screen

How will the sensors be registered in our platform?

1. Sensor Type registration Configuration file:
   Sensor id:
   Sensor Type:
   Sensor Data Type:
   Sensor Data Rate:

2. Sensor Instance Registration File
   Sensor Type:
   Sensor Location:
   Sensor Purpose:

Sensor Status:

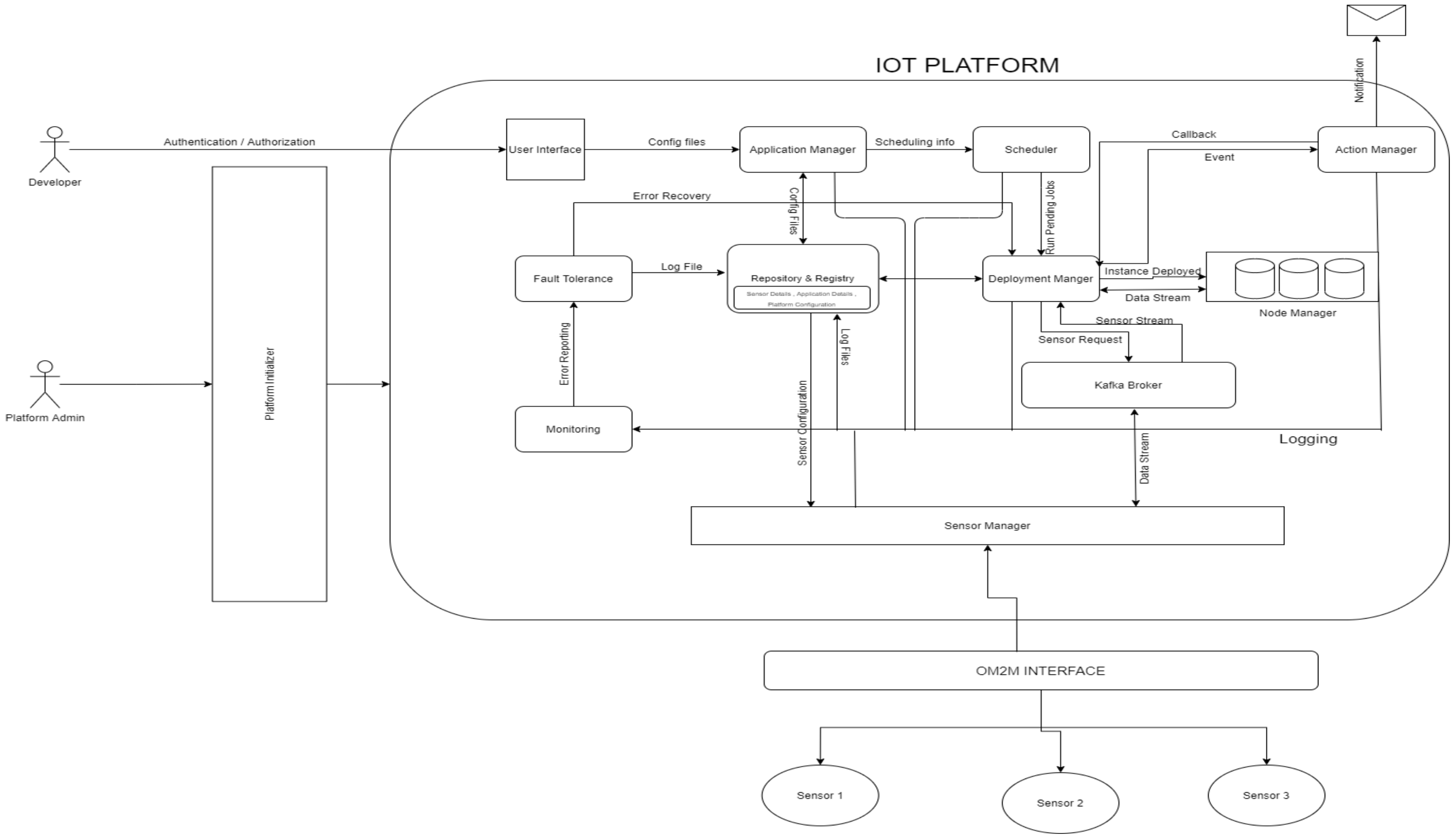
3.4 Non-Functional Requirements

1. Fault Tolerance
    a. Platform Level :- All services of the platform are tracked and if one service becomes unavailable, it is restarted.
    b. Application level :- The servers which host the algorithms are monitored and restarted if they terminate accidentally.
2. Scalability
    a. Application Scalability :- Multiple instances of an application can be executed simultaneously.
3. Accessibility of data
4. Specification of Application: The Platform will support applications written in Python
5. User Interface -
    a. Web GUI - Web Interface for the user to upload applications and monitor the applications and the platform.
6. Security -
    a. Authentication - Sign In for the user which authenticates users to access the platform services.
    b. Authorization - Checks for the access rights of the user to access the platform and applications.
7. Persistence - The applications, schedules and sensors registered on platform persist even after the platform restarts.

4. List the key functions

4.1 A block diagram listing all major components

- Sensor Manager
- Deployment Manager
- Application Manager
- Action Manager
- Monitoring
- Scheduler



4.2 Brief description of each component

1. **Sensor Manager** :- This component will manage sensor registration on the platform and make the data available to the applications hosted on the platform.
2. **Deployment manager** :- This component is responsible for the deployment of the applications that platform supports and executes. Deployer will take care of fault tolerance of applications and servers. It will monitor the servers and applications. It will fetch the logs of the applications and stats of the server. Deployer can stop an application.
3. **Application Manager and Scheduler** :- This component will manage all the interaction for the application developer, platform administrator and end user with the platform.

4. **Action Manager** - This module will take actions based on occurrence of some events on the platform.
5. **Monitoring** - The monitoring module will constantly monitor the platform for any abnormal behaviour and will intimate respective components to take certain actions if any anomaly is detected.
6. **Scheduler** - This component will handle the scheduling of algorithm instances as specified by the end user.

4.3 Major parts

- Deployment Manager
- Application Manager
- Sensor Management
- Platform Initializer

5. Use Cases

5.1 List what the users can do with the solution

1. Users can take decision based on result of the algorithm execution
2. Can generate Callbacks based on sensor data and provide dashboard results.
3. Users can schedule and deploy the applications to the platform and can examine the results.Each application will be isolated and contained from other running applications.
4. Users can bind new sensors to there which are at different locations and are of different types.
5. Users can also get event based message and email alerts from the platform to monitor and get notified after certain events.

5.2 Users of the platform

- **Platform Manager** : The platform manager starts the platform and monitors the health and performance of the services provided by the platform.
- **Application Developer** : Application Developer deploys the application on the platform.
- **Application Administrator** : Application Administrator will be able to monitor the running applications.
- **End User** : The end user user who is going to use the services provided by the application deployed on the platform**.**

5.3 Usage Scenarios:

- Smart home system application -
- Traffic analysis system -
- Smart City -
- Smart Marketing -
- Automated Parking system -
- Health Monitoring - Glucometer , Blood Oxygen level,

6. Primary test case for the project

6.1 Name of use case

Smart Township

6.2 Environment of this use case

Housing Management

6.3 Description of the use case

This will enable us to build smart societies applications that can provide connected solutions to security cameras , power management for the societies.

6.4 What is the "internet: angle around these things"

The main purpose of IoT devices is to generate real-time data that we can then analyze and use to create desired business outcomes.
The sensors are connected to the platform via the internet and notifications actions to different devices are sent via the internet. Communication between different modules takes place via the internet.

6.5 Information model

The sensor will collect data at their ends and send them to the application server instance which will process and send action to the action manager. The action manager will send the notifications to the society people as well as activate the corresponding action sensors like power sensor or alarm sensor. This can be scaled to a variety of smart applications and smart sensors.

6.6 How is location and sensory information used

1. Location wise security camera monitoring.
2. Water level management across the societies.
3. Power Grid Management
4. The location sensors can be used to help find spots in parking spaces for public amenity buildings.

6.7 Processing logic on the backend

1. The application instance will be running on the server instances that will be executed.
2. The server instances will request the sensor data like camera sensor stream or power sensor monitor and then process them accordingly.
3. Finally the end user will be notified through the email , push - notifications.

6.8 User's UI view-

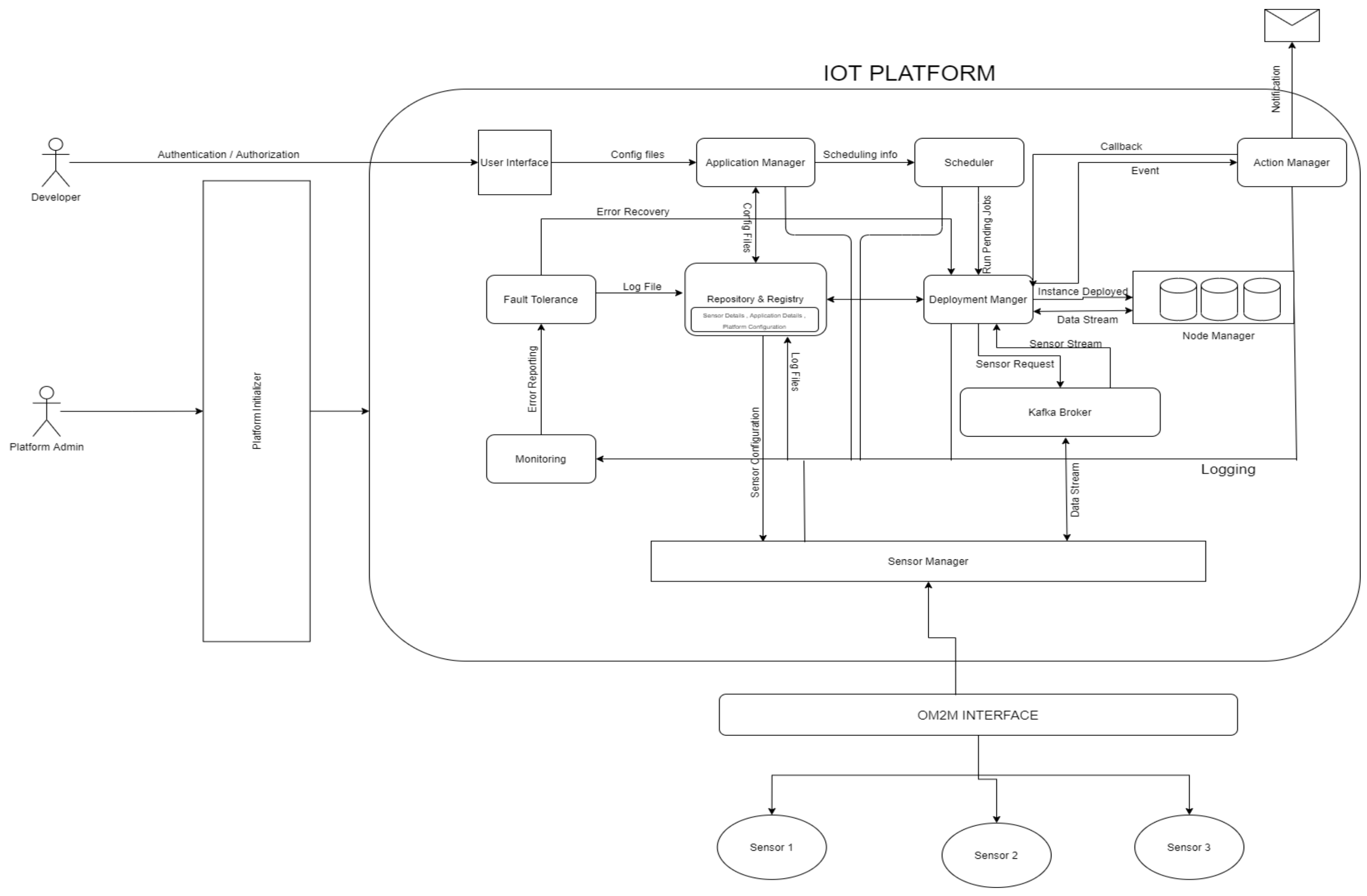
The application developer and administrator will be having a Graphical User Interface for deploying and management of applications.

7. Subsystems

7.1. Key Subsystems :

A. Platform initializer
B. Action Manager
C. Communication module
D. Sensor manager
E. Monitoring and Fault tolerance
F. Deployment manager
G. Application manager
H. Scheduler

7.2. A block diagram of all subsystems:



7.3 Interactions involved across these subsystems

- **Platform Initializer and Others**
  Platform Initializer passes configuration requirements to all the platform services and deploys them on servers after fetching them from the platform repository
- **Application manager and Scheduler**
  Application manager will send application codes from the application repository to the scheduler based on the scheduling requirements given by the user which will be forwarded to the deployment manager
- **Scheduler and Deployment Manager**
  Scheduler passes the application and scheduling details to the deployment manager which uses it to deploy applications on servers
- **Servers, Sensor Manager and Action Manager**
  Server requests the sensor data from the sensor manager based on the application deployed and send the action details to the action manager
- **Monitoring and Fault Tolerance and Others**
  Monitoring module receives status updates, heartbeat pings and logs from all the modules and fault manager uses this information to reschedule failed services, applications and services.

7.4 Protocols. Mechanisms

- Communication mechanism across different subsystems will be handled by KAFKA Module and we will use oneM2M protocol to communicate with different types of sensors in this platform.

7.5 Registry & Repository

1. Application Registry and Repository
2. Sensor Registry
3. Platform Configuration
4. Platform Logs
5. Platform Repository

7.6 The four parts of the project

1. Communication module, Sensor manager
2. Deployment manager
3. Application manager and Scheduler
4. Platform initialization,Monitoring and Action Manager

8. Brief Overview of Four parts

1. Communication module, Sensor manager
2. Deployment manager
3. Application manager and Scheduler
4. Platform initialization, Monitoring and Action Manager

8.1 Communication Module, Sensor Manager

8.1.1 Communication Module

The efficiency and speed of any distributed system depends greatly upon its communication manager which handles how different components of the system will communicate with each other and transfer the requests and responses among them based on some communication standard.

Block Diagram of Communication Module

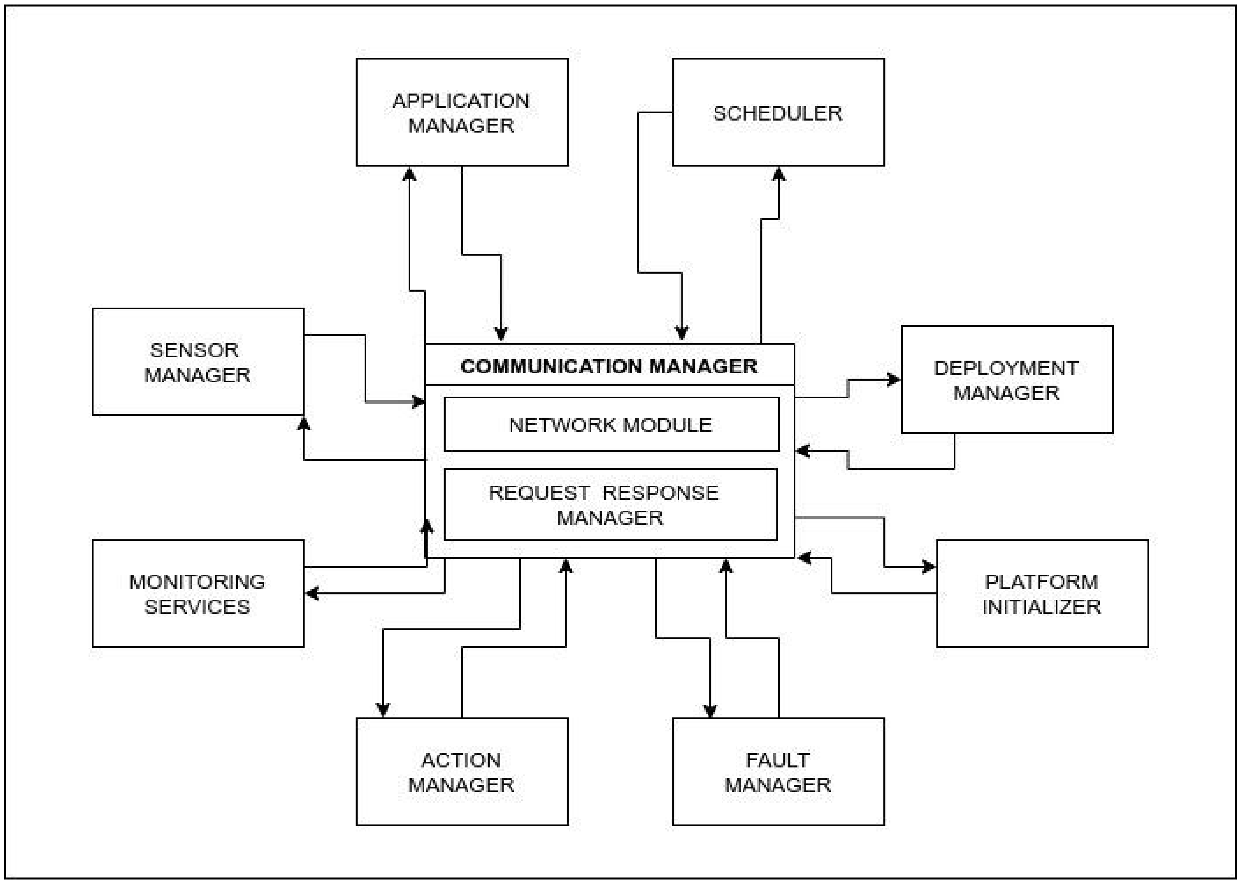


Fig.8.1 Communication Manager

Sub Systems of this Module are:

- **Network Module :** The network module will basically implement our communication standard which can be something like Apache Kafka/ RabbitMQ which will help in the setup of the communication channels between all the modules.
- **Request Response Manager :** This module will deal as a traffic router and will help with the routing of requests and responses to their respective senders and receivers.

Services Provided by this Module are:

➔ Handling the communication channel and communication framework of the whole system.
➔ Handling the transfer of request and response among various modules.
➔ Keeping communications online.
➔ Secure and virtually lag less transfer of data among the various components.

Interactions with other Modules are:
All the modules will be integrated with this module to be able to communicate among each others natures of such interactions are as follows:

- **Action manager** accepts actions to be performed and sends the responses required.
- **Sensor manager** accepts requests for data binding and sends the responses to the application managers and other modules.
- **Fault manager** requests the stat report from Monitoring services and send the response based on the report to the deployment manager and other modules
- **Monitoring services** pings all the modules for their stats and stores it in the log database and responds to requests from fault manager.
- **Deployment Manager** accepts requests for applications to be deployed from scheduler and fault manager.
- **Application manager** sends the request for required data to the sensor manager and application details to deployment manager and scheduler.
- **Scheduler** sends the request to the application manager to fetch the application and sends the request to the deployment manager to deploy an application.
- **Platform Initializer** initializes all the components and sends their respective configurations to them.

8.1.2 Sensor Manager

Collects data from sensors present at different locations and binds it with respective algorithms and will help in registration of sensors, processing sensor data.
The **purpose** of the sensor module is -

- to manage interaction with the sensors present at different geographic locations and collecting different data from the environment
- collecting data from the sensors
- Processing sensor data so that it can be used by different applications implemented on top of our platform
- Storing data from sensors in a database.
- Giving information about sensors being active or not.
- Binding data collected from server to the algorithms used by applications as per their access permissions.

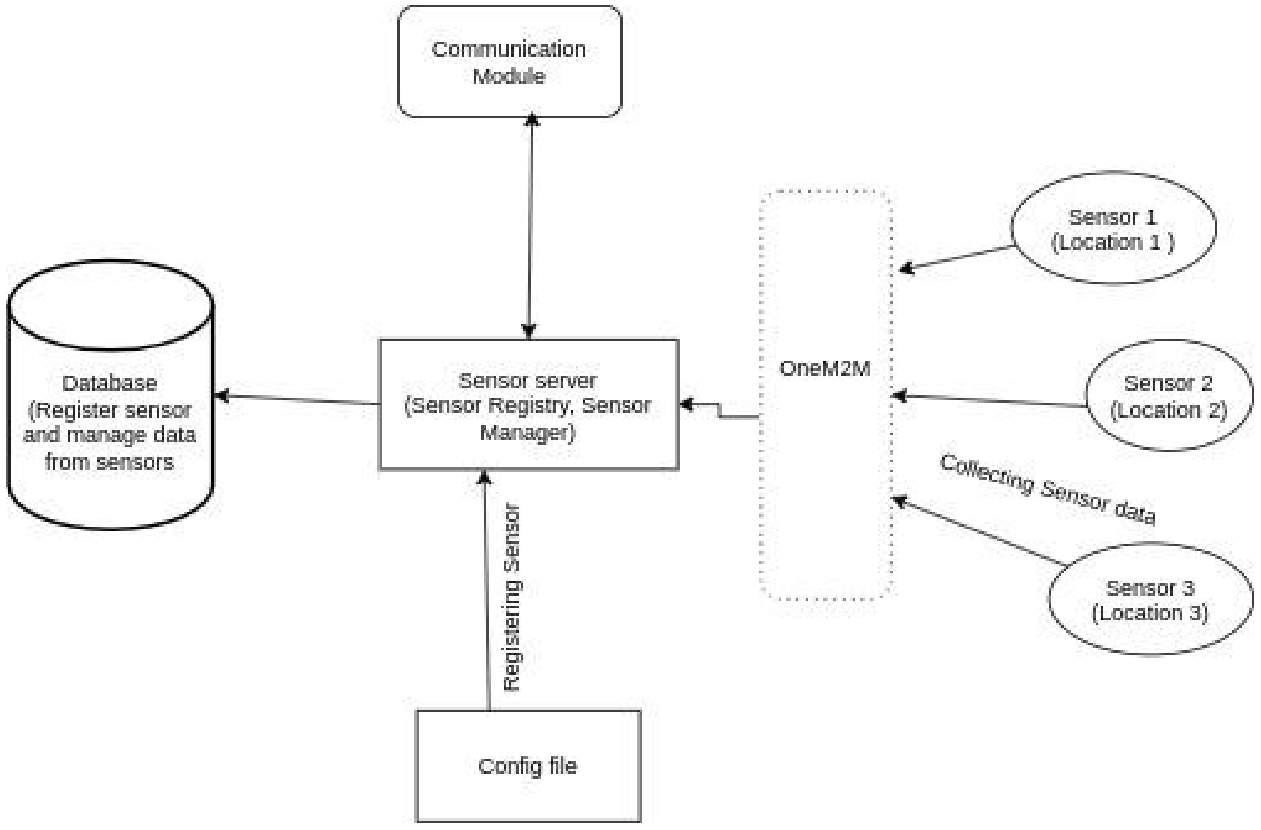


Fig.8.2 Flow diagram of sensor manager

The **subsystems** present are -

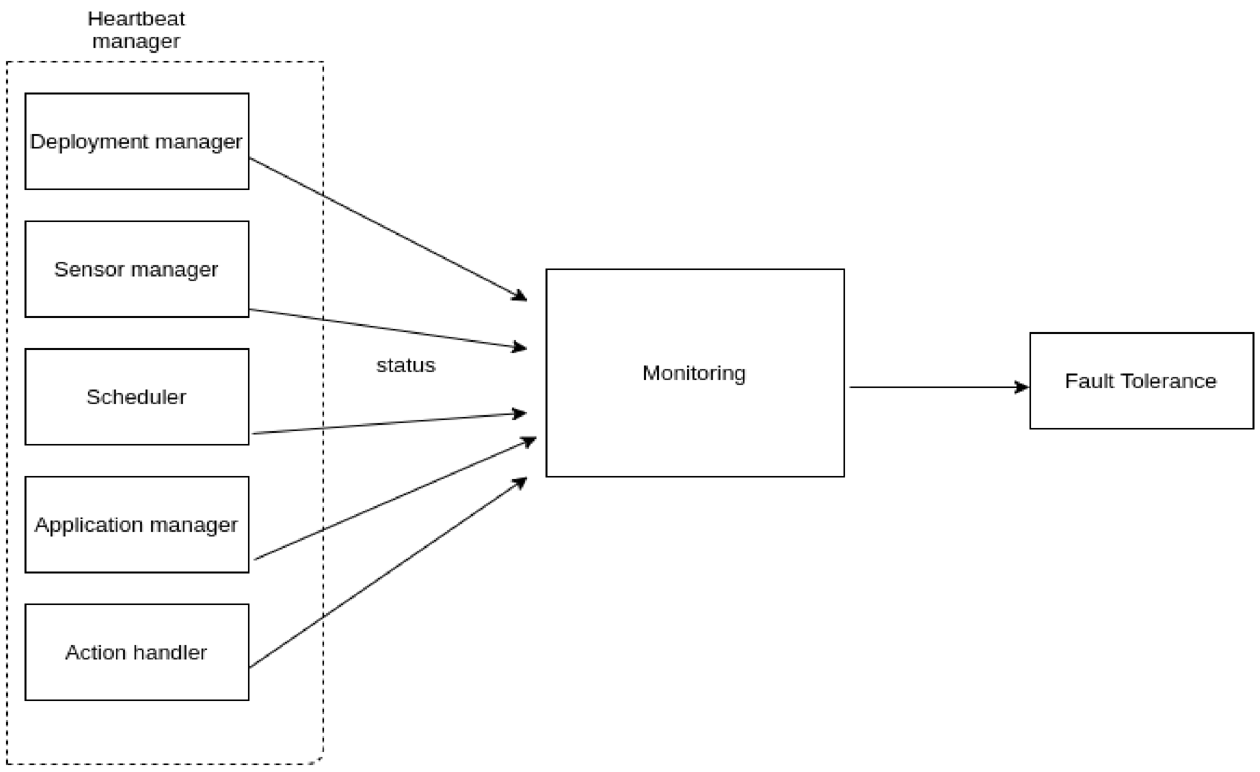
- **Sensor Registration** - takes config file as input from user containing details about sensor and then makes it's entry in the database. And then connect to that sensor to collect its data.
- **OneM2M Interface** - It provides connection between the sensors and the sensor manager. It takes data from the sensors and redirects it to the sensor server.
- **Sensor Server** - Sensor server has all the information about sensors present and is connected with onem2m interface to receive sensor data and database storing sensor data.
- **Data Binding** - It binds collected data to the algorithms which are implemented on top of the platform.

Interactions with Other Modules:

- **Communication Module -** To provide sensor data and other sensor related details requested by Communication Module.

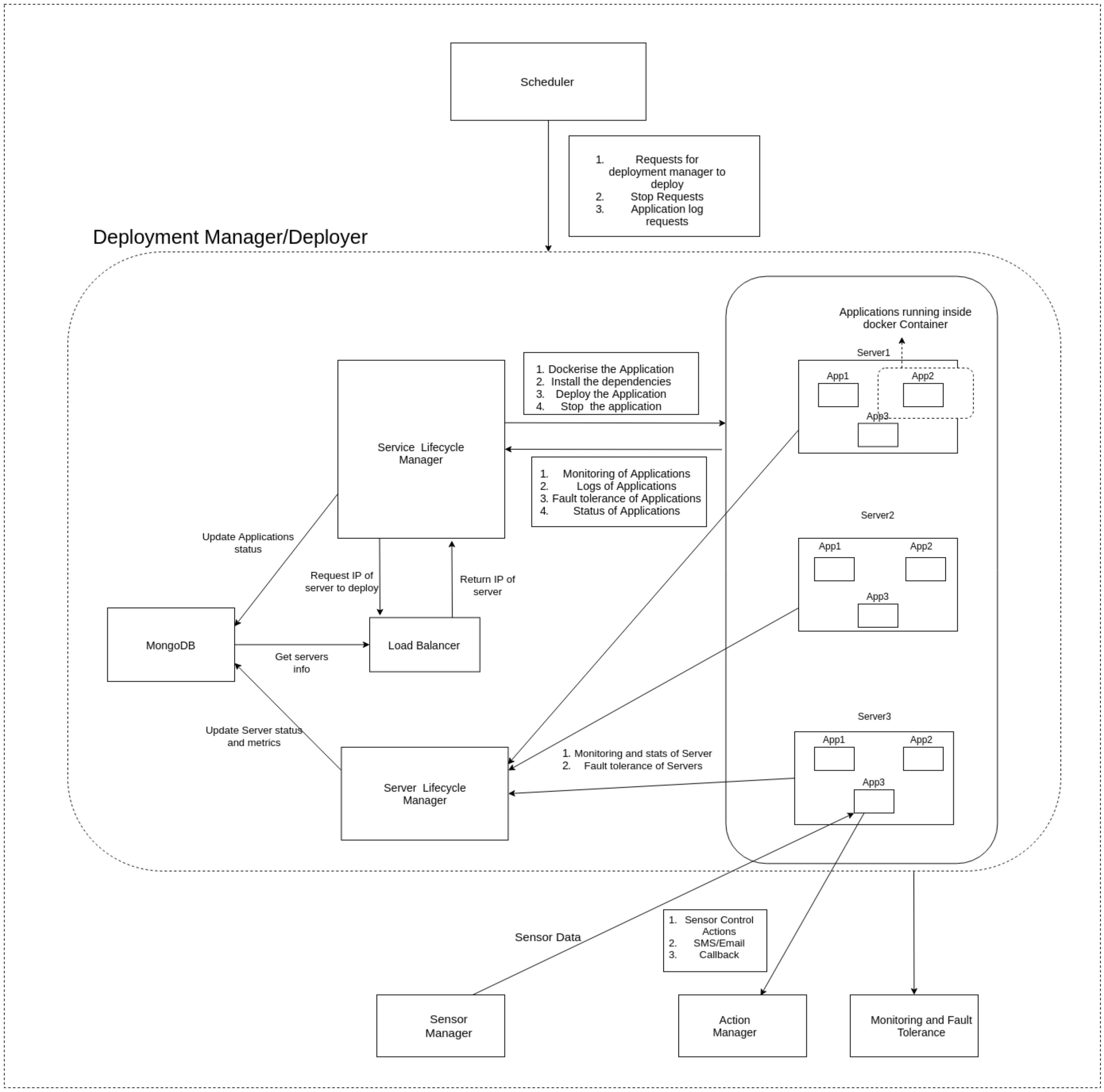
8.1.3 Monitoring

Monitors all the modules present in our platform and checks if they are working properly , detects problems or abnormal behaviour and informs about it to the fault tolerance manager.



- **Subsystem: Heartbeat manager** is the **subsystem** in this part which will keep track of health i.e proper functionality of different components in our platform like of Deployment manager, sensor manager,scheduler,application manager and action handler.
- Now Monitoring keeps status of all the heartbeat from different components and sends the status to fault tolerance and now if there's any problem associated with any components the fault tolerance will take care of that.
- **End points or interaction**: Monitoring takes input as status from different components from heartbeat manager and sends status to fault tolerance if there's any issue.
- **Capability**: Keeps track of the status of proper functioning of all the different components , so that if there is some issue we can resolve and our platform works smoothly.

8.2 Deployment Manager


Scheduler
1. Requests for deployment manager to deploy
2. Stop Requests
3. Application log requests
Deployment Manager/Deployer
Applications running inside docker Container
Server1
App1
App2
App3
Service Lifecycle Manager
1. Dockerise the Application
2. Install the dependencies
3. Deploy the Application
4. Stop the application
1. Monitoring of Applications
2. Logs of Applications
3. Fault tolerance of Applications
4. Status of Applications
Server2
Update Applications status
Request IP of server to deploy
Return IP of server
MongoDB
Get servers info
Load Balancer
Server3
Update Server status and metrics
1. Monitoring and stats of Server
2. Fault tolerance of Servers
Server Lifecycle Manager
Sensor Data
1. Sensor Control Actions
2. SMS/Email
3. Callback
Sensor Manager
Action Manager
Monitoring and Fault Tolerance

8.2.1 Service Lifecycle Manager

The scheduler will send a request to execute an application/algorithm to the ServiceLCM. The request will contain the details of the dependencies, the actual executable. ServiceLCM will request the ip and port of the server to deploy from the Load Balancer.. Once it receives the ip of the server from the Load Balancer, ServiceLCM will first install the dependencies and then deploy the actual application/algorithm.
Service Lifecycle Manager will fetch the logs of the application, monitor the status of the application and restart the failed applications(and update the information in DB also). ServiceLCM will also stop the applications once it to stop requests from the scheduler.

8.2.2 Load Balancer

Load Balancer fetches the stats of the servers from the MongoDB, it chooses the one with the least load and returns its ip to the ServiceLCM once it asks for the server for deployment.

8.2.3 Server Lifecycle manager

Service lifecycle manager is part of the Deployment manager, it checks the health of each of the Node servers. If a server goes down, Then the server lifecycle manager will first restart the server, update its IP and then re-execute the algorithm/applications which were previously running inside the new server.

Interactions with Other Components:

Action Module :- After the application/algorithm gets executed on the server, If the output has to be sent as a message/ notification, then our component will communicate it to the Action Module. The action module then reads the message queue and acts accordingly.

Monitoring and Health Manager :- Monitoring and Health manager will periodically check for the health of our component, If our component (Deployment manager) goes down then it will restart the deployment manager.

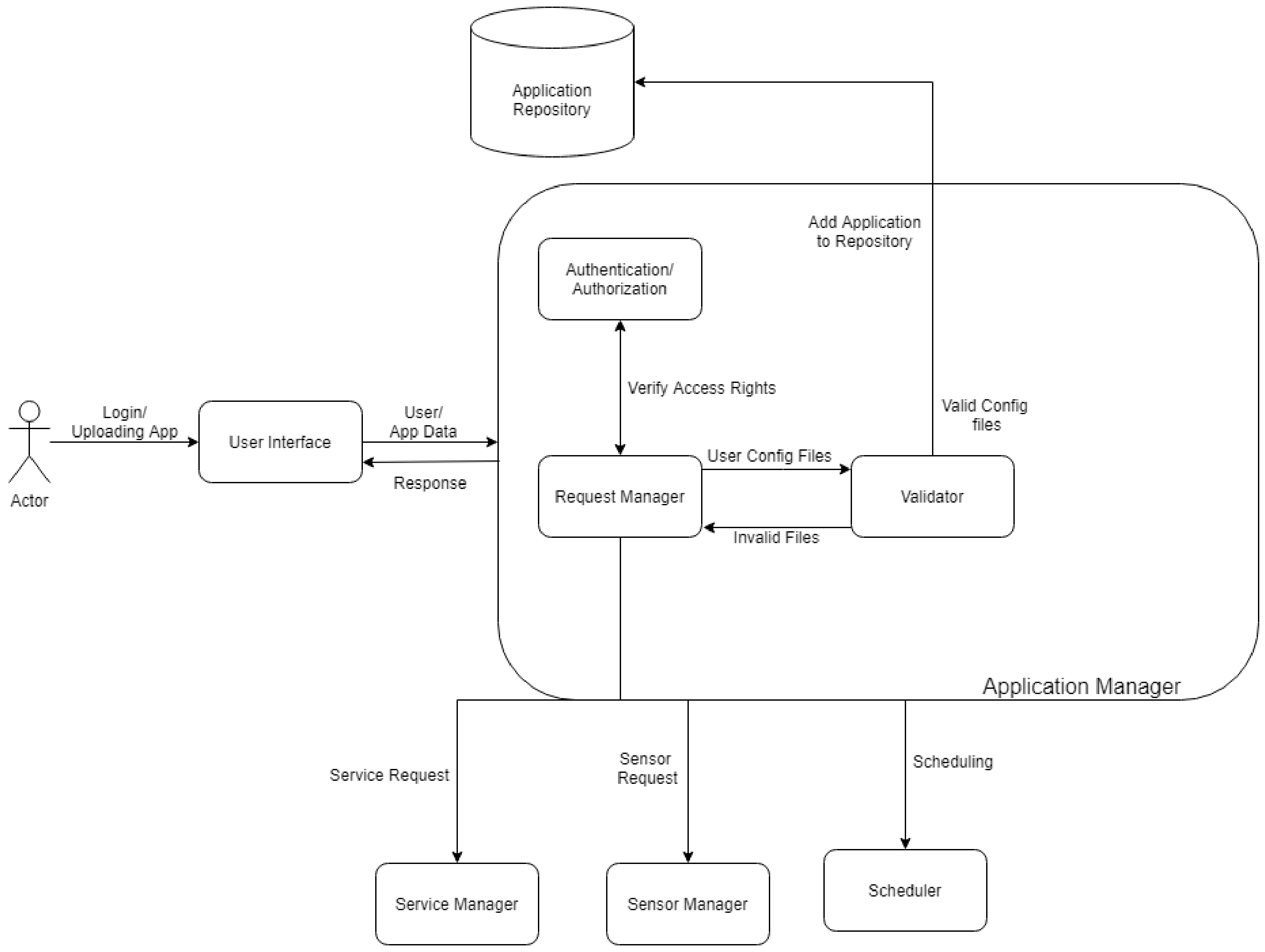
Scheduler :- Scheduler will communicate with the deployer regarding the algorithm/application which has to be executed .Apart from algorithm/application it will also share details regarding the dependencies that have to be installed before the request has to be executed.

Sensor Manager:- Some of the algorithms may need the data streams from the sensors. The data is provided by sensor manager via a communication channel like Kafka.

8.3 Application Manager and Scheduler

8.3.1 Application Manager

This sub-part of the platform will handle interactions with the application developer and administrator. The developer will be able to upload the application code and related files required to execute the application. This also includes the authentication and authorization of the user.



Block diagram of Application Manager

Subsystems -

- Authentication and Authorization Module
- User Interface
- Request Manager

- Validator
- Algorithm Manager

Service provided -

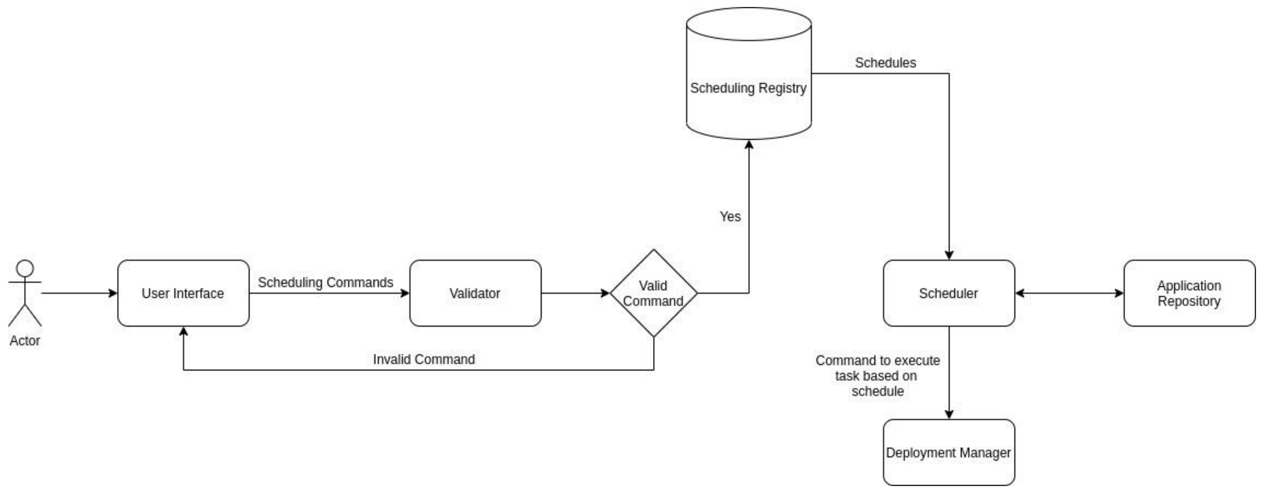
- Access Management
- Dashboard
- Application Repository

Interaction -

This subsystem will interact with scheduler, deployment manager and monitoring service. The scheduler and deployment manager will interact with the application repository to get algorithms of applications. It will interact with monitoring service to get details about the execution of applications. Application Manager will also interact with Sensor Manager for sensor registration.

8.3.2 Scheduler

This sub-part of the platform will handle the scheduling of algorithms of an application as specified by the user.



Service provided

This sub-part provides the feature of scheduling the algorithms by user. It also provides a registry for storing schedules.

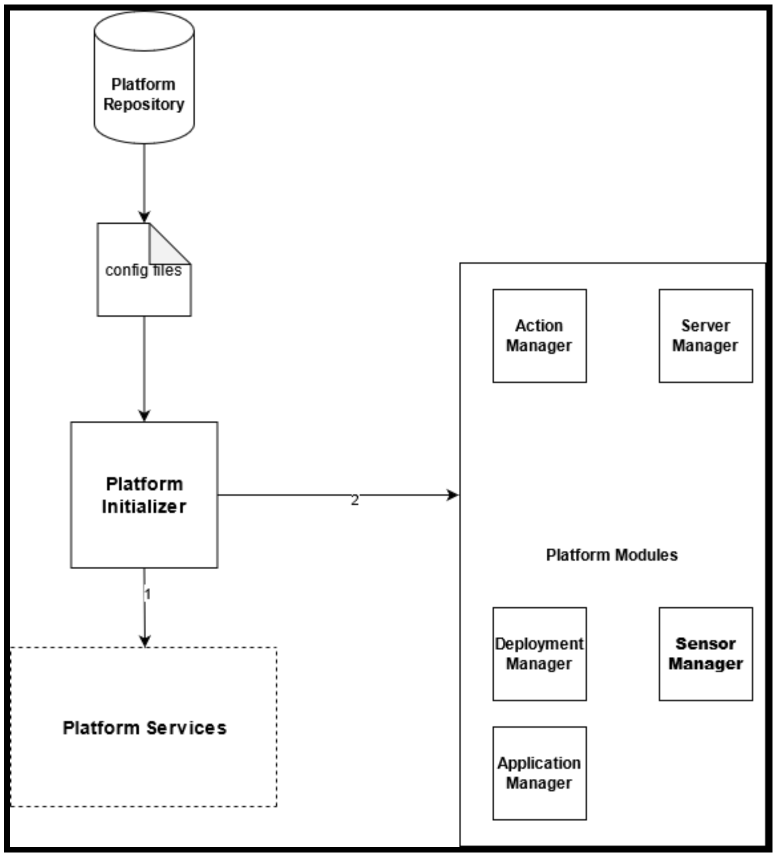
Interaction

Scheduler interacts with the application repository to get algorithms for scheduling and give commands to the deployment manager for execution according to schedules or as input by user.

8.4 Platform Initializer and Action Manager

8.4.1 Platform Initializer

This subpart will act as a Bootstrap Program to initialize all the services available in our platform by loading all the config files and executing all the bash scripts.



Interaction: Platform initializer will interact with the platform repository to fetch the config files which will then initialize the platform services followed by platform modules.

8.4.2 Action Manager

This Module is responsible for collecting the output result given by the assigned server and forwarding it to the intended users via Notifications/SMS/Emails.

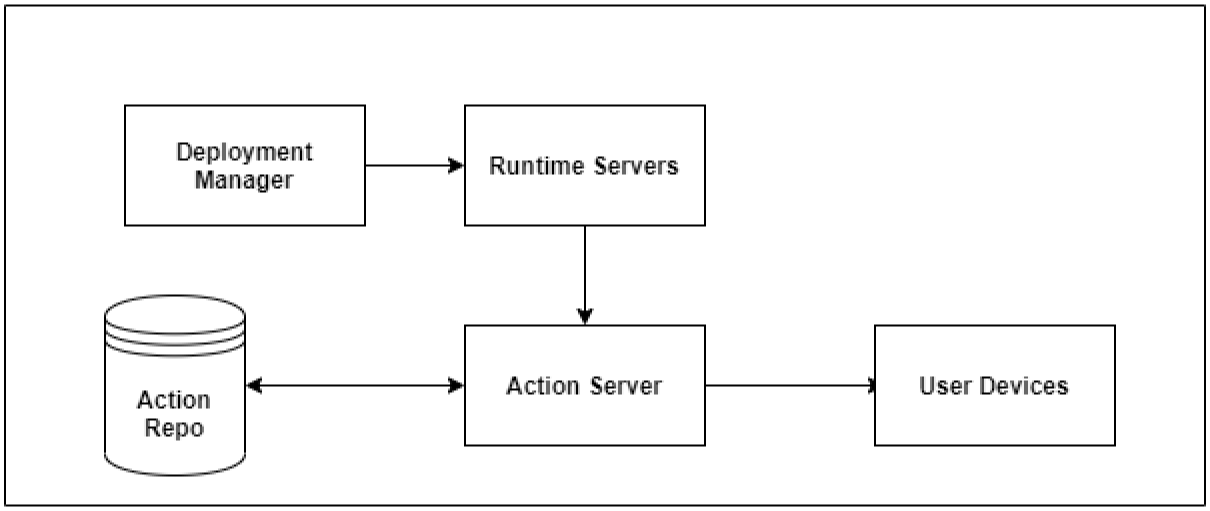


Fig 2. Action Manager