IAS - Group - 7

IOT Based Application Platform

Group Requirements Document

May 2021

Contents

List of Figures

1 Overview

1.1 Introduction

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.

1.2 Scope

Any IOT based application can be deployed on the platform.

2 Intended Use

2.1 Intended Use

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.

2.2 Assumptions and dependencies

- All the sensors are available before application execution.
- User will submit the application configuration file in fixed format.
- User application must be written in python language.

3 System Features and Requirements

3.1 Platform Requirements

3.1.1 Different actors on the platform

- User
  - All the application and its services will be used by this actor in the platform.
- Application developer
  - This actor will deploy the application on the platform.
- Platform admin
  - This actor will do all admin work and management work on platform.
- Platform developer
  - Developing and testing of all major component is done by platform developer.

3.1.2 Applications overview on the platform

- Applications relating to IoT, which involves sensors and/or controllers can be deployed on the platform.
- Different actors involved need to specify their part of information, in order to run the application.
- The sample application which is implemented and tested on the platform can be summarized as follows:
  - There are 5 buses, for covid relief, with below mentioned sensors and controllers and 2 police barricades with one gps sensor each
  - **Sensors involved**: Light sensor, temperature sensor, gps sensor, biometric sensor
  - **Controllers involved**: Light controller, AC, buzzer
  - **Use Cases involved**:
    - When someone boards the bus, the fare information will be sent via sms to the bus guard
    - When temp rises AC should be started, when lux value is low, light should be switched on
    - When more than 2 buses come together, they should be informed by a buzzer, in order to get separated
    - When a bus comes near a police barricade, an email regarding bus info should be sent to the authority.

Hence, such applications with sensor and controller details and algorithm can be implemented on the platform

3.2 Functional Requirements

3.2.1 Registering sensors

- Whenever sensor is added to the platform for the first time or if platform is initialized, sensor registration is done so that sensors's data can be retrieved.
- During registration, details about sensor like sensor id, input/output type, data rate, location etc. is stored in the platform repository.

3.2.2 Interaction with IoT sensors

- Sensors have unique characteristics through which it can be uniquely identified.

- OneM2M is a standard that provides a common M2M service layer that can be embedded within various hardware and software to connect these IoT sensors.
- Horizontal architecture of oneM2M provides common service functions that enables multiple applications, using common framework.

3.2.3 Application Development Model

Development Model Steps:

- 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

Application Development Model Block diagram

- App development Process

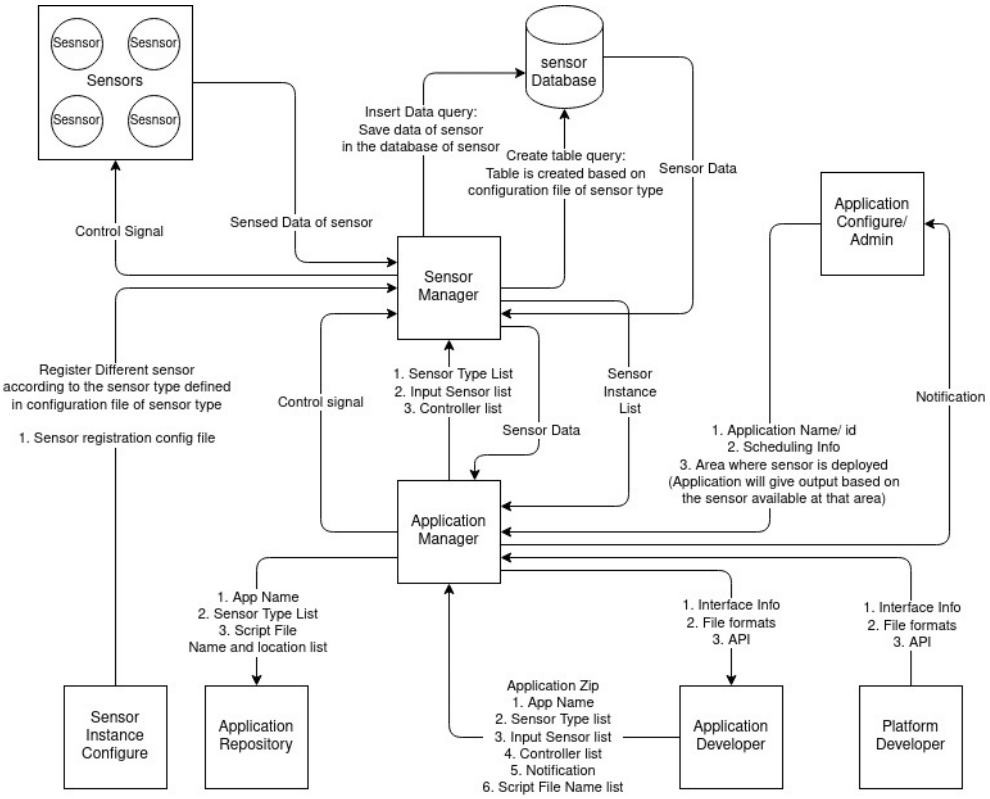
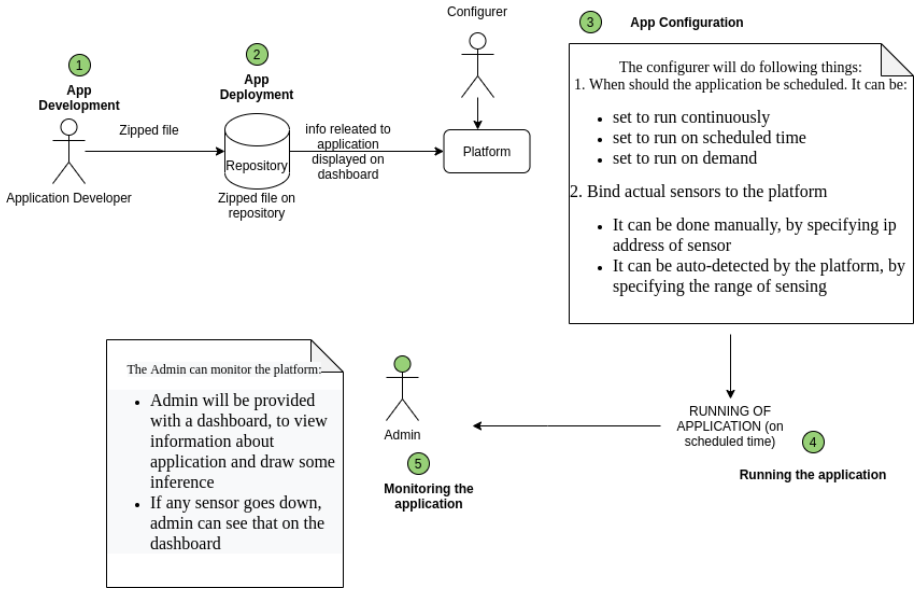
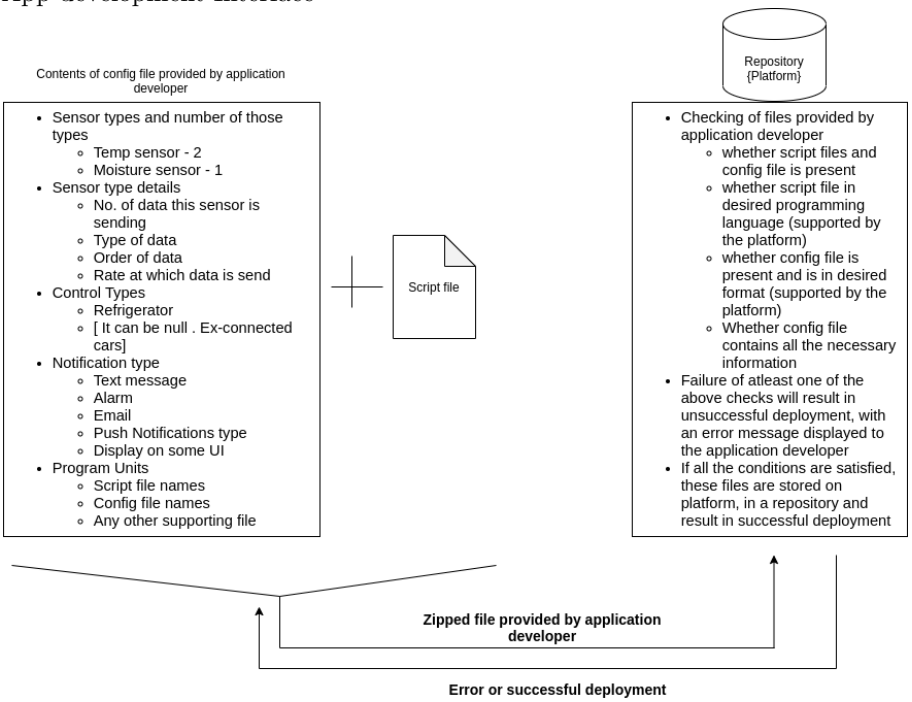


Figure 1: Block diagram of Application Development Model



- App development Interface



Actors

- Application Developer
  Upload Application with following file
  - App Name Configuration File
  - Sensor Type Configuration File
  - Input Sensor Configuration File
  - Controller Configuration File
  - Notification Configuration File

  - Script File Configuration File
- Sensor Instance Configure
  Will deploy/register sensor by uploading
  - sensor registration configuration file
- Application Developer/ Admin
  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)
- Platform Developer/ Admin
  Will provide following info to application manager at the start of platform
  - Interface Info
  - File formats
  - API

Key elements

- App Name Configuration

```
{
    "app_name" : "My app"
}
```

- Input Sensor Configuration File

```
{
    "sensor_type_fan": [
        "sensor_fan_1",
        "sensor_fan_2"
    ],
    "sensor_type_2": [
        "sensor_2_1",
        "sensor_2_2",
        "sensor_2_3"
    ],
    "sensor_type_3": [
        "sensor_3_1",
        "sensor_3_2"
    ],
    "sensor_type_4": [
        "sensor_4_1"
    ]
}
```

- Controller Sensor Configuration file

```
{
    "controller_type_fan": [
        "controller_fan_1",
        "controller_fan_2"
    ],
    "controller_type_2": [
        "controller_3_1"
    ],
    "controller_type_3": [
        "controller_3_1"
    ]
}
```

- Sensor Type Configuration File

```
{
    "new_sensor_type_1": {
        "company" : "Samsung",
        "model" : "tv",
        "data-rate" : "100 mbps",
        "column_count": "4",
        "column": [
            [
                "data_type_1",
                "length"
            ],
            [
                "data_type_2",
                "length"
            ],
            [
                "data_type_3",
                "length"
            ],
            [
                "data_type_4",
                "length"
            ]
        ]
    },
    "new_sensor_type_2": {
        "company" : "LG",
        "model" : "fridge",
        "data-rate" : "2 mbps",
        "column_count": "2",
        "column": [
            [
                "data_type_1",
                "length"
            ],
            [
                "data_type_2",
                "length"
            ]
        ]
    }
}
```

- Script location Configuration File

```
{
    "app_run" : "./location/app_run.py",
    "sensor_data" : "./location/sensor_data.py",
    "contol_sensor" : "./location/contol_sensor.py",
    "notification" : "./location/notification.py"
}
```

  - All Scripts will inherit predefined interface
  - And interface contains method which all scripts has to implement

- Notification Configuration File

```
{
    "notification_high_temp": [
        {
            "controller_type_alert_system": [
                "controller_phone_alert",
                "controller_email_alert"
            ]
        },
        {
            "controller_type_ac": [
                "controller_ac_start"
            ]
        },
        {
            "controller_type_fan": [
                "controller_fan_start"
            ]
        }
    ],
    "notification_type_2": [
        {
            "notification_type_1": [
                "controller_1",
                "controller_2"
            ]
        },
        {
            "notification_type_2": [
                "controller_3",
                "controller_4"
            ]
        }
    ]
}
```

- Sensor Registration zip File

  Sensor Registration Configuration file

```
{
    "sensor type": "Sensor_type_1",
    "location": {
        "ip": "x.x.x.x",
        "port": 50000,
        "geo location": "Ahmedabad"
    },
    "sensor model no": "zzz",
    "subscription cost": "p$"
}
```

- Application Manager
  - Receive request from Application Developer
    - It will receive the request of application developer to upload application on the platform
    - Request will contain
      - App Name Configuration File

            - Sensor Type Configuration File
            - Input Sensor Configuration File
            - Controller Configuration File
            - Notification 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
        * Saves following to repository
            - App Name Configuration File
            - Script File Configuration File
            - Sensor Type Configuration File
    - Sensor identification
        * Application Manager will locate all the sensors which will be used in the application from the area info provide in request of the use application from Application Configurer/ Admin
- 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
            - 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
    - Sensor registration
        * Based on the sensor registration file it will create a sensor instance and add it to the respective sensor type.

3.2.4 Identification of sensors for data binding

- On sensor registration, all important details of sensors are stored in the platform repository. Also, all sensors have unique property thus can be uniquely identified.
- Whenever sensor manager receives data binding request, based upon the parameters it will identify the sensor and sends data for binding in specified format.

3.2.5 Data Binding to the application

- An application instance running on a particular node will communicate with the sensor manager along with the sensor id to send/receive data stream to/from a particular sensor or multiple sensors.
- Based upon the request parameters, sensor manager will uniquely identify the sensor and send data for binding in specified format.

3.2.6 Scheduling on the platform

- The scheduler will retrieve config files of the application or algorithm from the repository and validate them.

- Once the validation has been done, it checks the starting time, ending time and the priority of the job, and adds them to a priority queue.
- In the case of a normal job, the job is added into the queue only once, and in the case of a cron job, it is added repeatedly until its finish time. At the start time of the job, it is sent to the deployer to be run.

3.2.7 Acceptance of scheduling configuration

- The scheduling information(in the form of meta file) is initially provided by the application developer and is stored in the repository
- When the application is to be run, the Application Manager fetches application and the meta file and hands it over to the Scheduler.
- The Scheduler then scans the meta file and infers necessary scheduling information from it and validates this information.

3.2.8 Starting and Stopping services

- Once the acceptance and validation of the scheduling information has been done, it checks the starting time, ending time and the priority of the job, and adds them to a priority queue.
- In the case of a normal job, the job is added into the queue only once, and in the case of a cron job, it is added repeatedly until its finish time. At the start time of the job, it is sent to the deployer to be run.
- Once the ending time of the job is reached, the scheduler sends the signal to the deployer to stop the particular job.

3.2.9 Communication Model

Following communication types are required in our platform:

- Continuous stream of data produced by sensors and consumed by application whenever required.
- Continuous monitoring of nodes, services and application for fault tolerance.
- Inter module communication (eg: Communication between deployer and node manager).
- Intra module communication (eg: Communication between nodes and node manager).

For asyncronous continuous data communication, **Kafka** can be used.
For inter module communication or intra module duplex communication, **socket communication** can be used.

3.2.10 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

3.2.11 Deployment of application on the platform

- The job is handed over to Deployer by the Scheduler
- The deployer checks if the job requires a standalone environment, or a shared environment.
- Accordingly, it either demands a new node from the node manager, or tries to run the job on one of the existing active nodes, list of which it retrieves from the node manager.
- In case of shared environment jobs, the deployer makes use of Load Balancer, to select the node with least load.
- Once the job or algorithm is done running, and the node is no longer being used, it is returned to the node manager.

3.2.12 Registry & repository

- Registry and Repository will be used to store the information for different platform modules like User data , Application data, Node Info etc.

3.2.13 Load Balancing

- The Load balancer 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 and this information will be passed to the Deployer.

3.2.14 Interactions between modules

- Application Manager and Scheduler
  Application Manager will provide Scheduler with the application and the meta file 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 at 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 module.
- 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 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.
- Sensor Manager and Nodes

An application instance running on a particular node may wish to communicate with the sensor manager to send/receive data stream 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.

3.2.15 Packaging details

Application configurer must upload the files in the following way:

Application zip - Contains configuration files and scripts.

Controller instance configuration zip: Contains controller instance configuration files.

Controller type configuration zip: Contains controller type configuration files.

Sensor instance configuration zip: Contains sensor instance configuration files.

Sensor type configuration zip: Contains sensor type configuration files.

3.2.16 Interaction of different actors with the platform

- User
  - All the application and its services will be used by this actor in the platform.
  - This actor will have to register and authenticate himself on the platform to use application & its services.
  - This actor is responsible for starting the respective application which has been deployed by the actor-Application developer.
- Application developer
  - This actor will deploy the application on the platform with which he will have to provide the useful sensor and other required information in the configuration file.
  - This actor can request analytic module to check current status of the application which had been deployed by him.
  - This actor will do required application updation work if needed.
- Platform admin
  - Sensor registration will be done by platform admin.
  - Request to analytic module for current status of the platform can be given by platform admin.
- Platform developer
  - This actor is responsible for starting and stopping all major components in the platform.

3.3 Non-Functional Requirements

3.3.1 Fault tolerance

- Platform
  - To check whether all the components (like scheduler, deployer, application manager etc) of the platform are working properly, and in case of failure reinitialize the component.
- Application

- To check if the application instances are working properly or not. If they are not working then re-initialize another app instance.
  If some active node gets down, then it reinitializes all the applications/modules running on the node on some other node.

3.3.2 Scalability

- Platform
  - Any number of sensor can be registered to the platform.
- Application
  - If an application instance is consuming resources above a specified threshold, then that instance will be shifted to a new node at runtime which exclusively belongs to this application instance.

3.3.3 Accessibility of data

- Application
  - Application will specify what all sensors it will use in the configuration file and it will be able to access the data from the sensors.
- Sensors
  - Based on some events sensors can receive data from the application.

3.3.4 UI for Interaction

- Platform Configurer: He can upload type and instance configuration files for sensors and controllers through UI.
- Application Developer: He can upload application zip file containing application code, required sensor information to the platform using UI.
- User: He can deploy desired application at any place on desired time using UI. He also can see notifications for different applications deployed by him.

3.3.5 Security - Authentication and Authorization

- Authentication
  - Every user would like to connect to the platform has to be authenticate himself to the authentication manager.
  - User can access the application or service on the platform only after authentication.
- Authorization
  - Platform admin has access to check the current status of the platform.
  - Application developer has authority to check all the current status information of the application which had been Deployed by him on the platform.

3.3.6 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.

4 List the key functions

4.1 A block diagram listing all major components

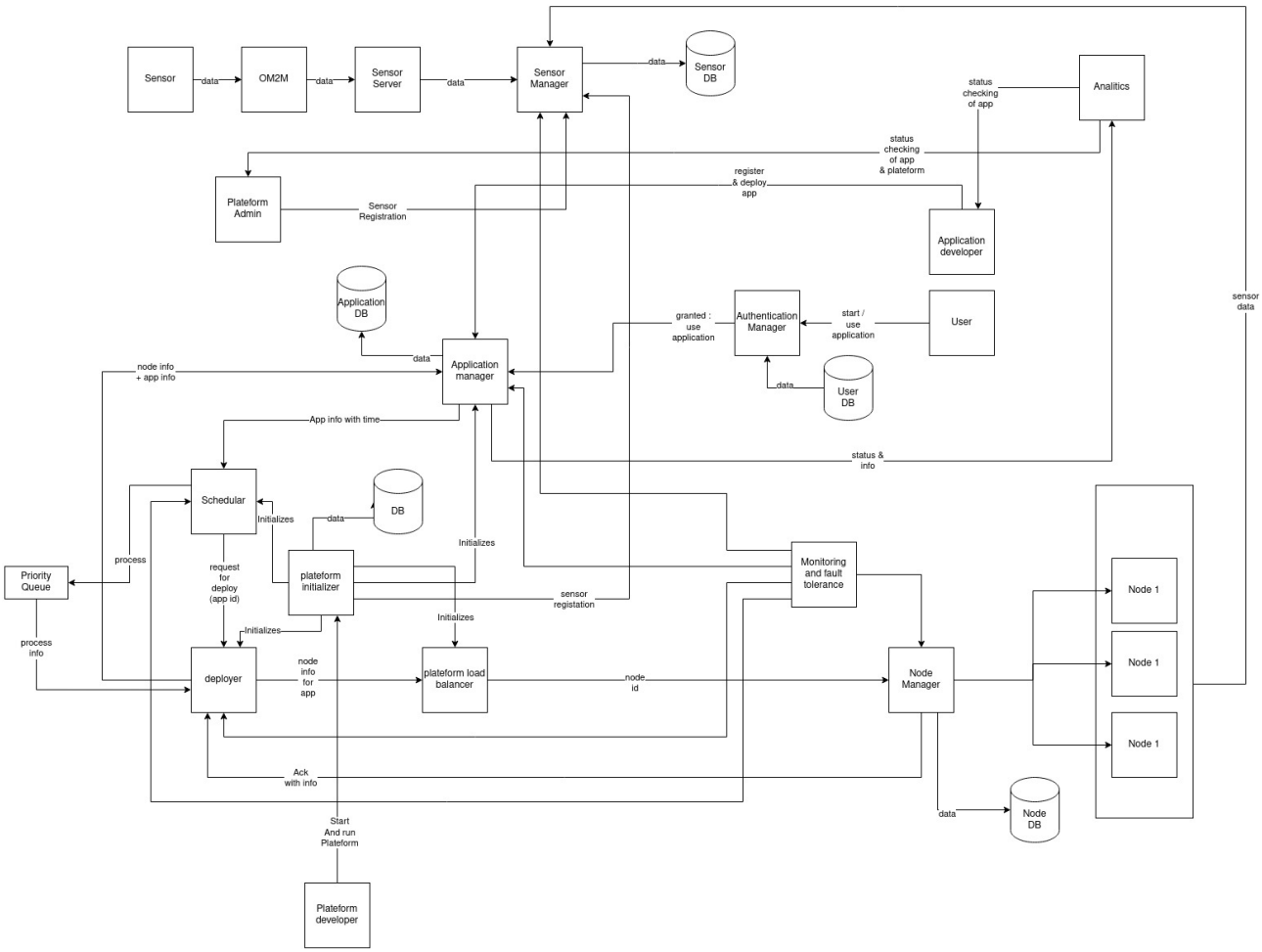


Figure 2: Block diagram of Platform

4.2 Brief description of each component

4.2.1 Platform initializer

- This component is responsible for initializing various platform components on client machines
- It takes in the config of client machines, installs platform clients on them, and deployes the platform components.
- It also builds config files to be used by every component

4.2.2 Scheduler

- This component is responsible for scheduling of various jobs, and stopping them at the end time specified.

- Scheduler take config files from the repository, as well as from the UI.
- It validates the config files, and then send the jobs to the deployer at the specified time, to be deployed at a node.

4.2.3 Deployer

- Deployer is responsible for actually deploying the jobs, algorithm on the node.
- Deployer checks if the job to be run requires shared or standalone environment. Accordingly, it retrieves the nodes from the node manager, or the load balancer, and deploys the job.
- It then conveys the node manager about the node and the application / job running on it.

4.2.4 Load Balancer

- Load Balancer primarily communicates with Deployer and Node Manager.
- On the request of 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.

4.2.5 Application Manager

- This Module is Responsible for authenticating the users and give responses to requests coming from outside and inside of the platform.
- This module is also responsible for requesting event based actions.
- Apart from this, it also helps to provide information to Analytics module.

4.2.6 Authentication Manager

- This Module will authenticate every incoming users.

4.2.7 Nodes and Node Manager

- Nodes are the entities where execution of an application instance or a platform component takes place.
- Node Manager module is responsible for managing the node information and providing the deployer a new node if needed.

4.2.8 Monitoring and Fault Tolerance

- This module continuously monitors the components for failures and allows the system to operate properly, even in case of failure of one or more of its components.

4.2.9 Platform Initialization

- This module is used to initialize all the other modules of the platform, using their configuration.

4.2.10 Sensor Manager

- Sensor manager take care of sending data to application as per request, the rate at which data needs to be send, recieves sensor data from nodes and interact with sensor database for sensor data.

4.3 4 major parts

- Platform initializer, Scheduler, Deployer and Load Balancer
- Application Management
- Node Management and Monitoring & Fault Tolerance
- Sensor manager & server

5 End To End Packaging

5.1 Step 1

- Application developer to application manager(Outer Request handler):
  - Asks for application model
- App manager(Outer Request handler):
  - Reply as Application Model:
    - Sensor info file
    - App info
      - Application type (like python based, java based.. etc)
      - input/ output type of application
    - Environment Info

5.2 Step 2

- Application developer to application manager(Outer Request handler):
  - Application upload
    Things need to be provided by Application developer while uploading:
    - Requirement File:
      Libraries
      commands need to be executed before deploying
    - Application Configuration file:
      Sensor types / Sensors' type
    - Application:
      Source code /Executable /Both
- App Manager:
  - Verification of app
    regarding constraints checking which specified in application model

5.3 Step 3

- User to application manager(Outer Request handler):
  - Asks for applications' info available on platform
- App manager(Outer Request handler):
  - Reply as Application info:
    - App info
      Name/ Id (Uniquely identifiable)
      Types of sensors associated with app

5.4 Step 4

- User to application manager(Outer Request handler):
  - Request to deploy app
    Things need to be provided by User while deploying:
    - Application info
      Name/ Id (Uniquely identifiable)

        * Sensors Info
          Not ids but info
          location
          etc
        * Scheduling Info
          Time
- Application manager(Outer Request handler):
  - to Application manager(Action Manager):
    * forwards the (Request to deploy app)
  - Step 4.1:
    * Application manager(Action Manager) to Sensor Manager:
      · Sensors Info, Not ids but info
        location
        etc
    * Sensor Manager to Application manager(Action Manager):
      · reply with sensors' id
  - Step 4.2:
    * Application manager(Action Manager):
      · to Application manager(DB Handler):
        App id
        sensors' id
      · to schedular:
        app id
        Scheduling Info

5.5 Step 5

- Schedular to deployement Manager(Deployer):
  - App id

5.6 Step 6

- deployement Manager(Deployer) to Node Manager:
  - Asks for Available nodes
- Node Manager to deployement Manager(Deployer) :
  - Node Info List

5.7 Step 7

- deployement Manager(Load Balancer):
  - step 7.1:
    * to Node Manager:
      Instance id
      Node id
  - step 7.1:
    * to application Manager(Inner Request Handler):
      App id
      Instance id
      Node id

    * application Manager(Inner Request Handler) to application Manager(DB handler):
      App id
      Instance id
      Node id

5.8 Step 8

- Node Manager to application Manager(Inner Request Handler):
  - Asks for sensors' id of Instance id
- Application Manager(Inner Request Handler) to Node Manager:
  - sensors' id

5.9 Step 9

- Node To Sensor Manager:
  - Asks for sensor data:
    Sensors' id
    start time
    end time
- Sensor Manager to Node:
  - Reply the sensors' data

5.10 Step 10

- Node will give output
  - as ui output OR
  - as sensor output
  - as new app start
  - as new service call

6 Use cases scenarios

- Smart Home:
  The Smart home app can help users to access multiple devices faster than ever. With this, you can control home IoT devices like a refrigerator, washer, and dryer, dishwasher air conditioner, etc.
  One of the main key features of this app is the ability to remotely control and check the status of devices, group multiple devices together to control them simultaneously, configure the device settings, get notifications about different devices.
  Many smart home devices today are supported. These devices include- smart thermostat, wire-free Pro security camera, security system and many other devices. It means that once you have the Smrat Home devices in your home and the smart home app on your phone, you can access devices with application from anywhere in the home or outside home.
- Smart City
  Environmental Real-time data like temperature,Humidity,etc will be stored continuously in database. Researchers can analyse those data, based on which they can predict events. proximity and location-sensor can be used to find nearby public-wifi and can be used in automated traffic control system.
- Natural Disaster Alarm System
  Natural Disaster Alarm systes basically gives alarm to people/users who might be in danger in future due to natural disasters like earthquakes, flood, draught, volcano etc. There will be many actors, like users, experts, and authorities. There will be sensors deployed at some locations and pre-processed data will be shared across the platform to People and authorities using apps will be notified if experts send emergency red flag if they are in dangerous area.
- Connected Vehicles:
  The IoT ecosystem connects the vehicles, as moving "things", to infrastructure sensors and devices, and to external services and applications (mobility, traffic authorities, autonomous driving services). Here, interoperability is facilitated through the adoption of the oneM2M standard.
- Smart Traffic Management
  This application uses camera, GPS location sensor and RFID sensor to detect the traffic congestion on Traffic signals and roads and suggest alternate routes to the user based on the current location. RFID sensors help for fast and easy payment of toll at the toll plazas.
- Connected Health
  It can allow healthcare professionals to assist patients with prescriptions, medication, and also measure their biometrics using sensors and remote equipment. For instance, patients can connect any wearable or portable device to the cloud and update the data in real-time.
- Smart Industry
  Our platform can be used to deploy applications concerning industry management across locations. Applications in this domain can be used to manage and monitor production, make smart decisions using distributed sensor network, manage deliveries using GPS, vehicle health monitors etc.

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

- **Input** : node failure (did not received node's heartbeat message)
  Output : A new node with all the previous applications running.

- **Input** : Application/module failure (did not receive application/module heartbeat message)
  Output : A new instance of the application running on the same node.

- **Input**: Deployer asking for execution of an algorithm when the available nodes are occupied.
  Output: Return the new node id to the deployer

- **Input**: Application instance requiring data from 4 sensors.
  Output: Pass the application id and corresponding sensor ids from the node (on which that application instance is running) to the sensor manager. Then receive the data from the sensor manager and pass it to that application.

8 Subsystems

8.1 Key subsystems in the project

- Platform initializer
- Scheduler
- Deployer
- Load Manager
- Application Manager
- Authentication Manager
- Node Management
- Monitoring & Fault Tolerance
- Platform Initialization
- Sensor manager

8.2 A block diagram of all subsystems

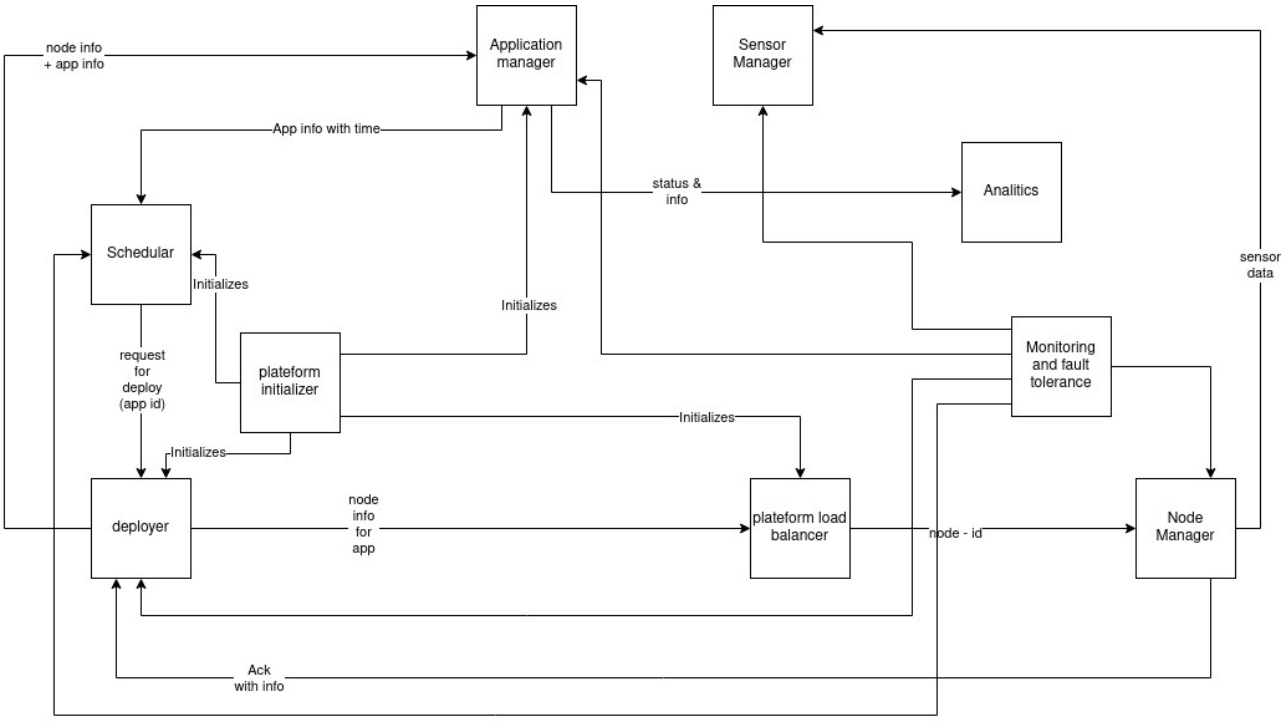


Figure 3: Block diagram of Subsystem

8.3 Registry & Repository

- Platform Database
- User Database

- Application Database
- Node Database
- Monitoring Database
- Sensor Database

8.4 Protocol Mechanisms

- For communication using kafka
  - Kafka server details needs to shared between the communicating entities.
  - Kafka topics need to agreed upon before communication begins.
- Communication using sockets
  - Client should know the server IP.
  - Client should also know the port on which the server is listening.

8.5 The four parts of the project (each will have its own team req. doc to be submitted)

8.5.1 Platform initializer, Scheduler, Deployer and Load Balancer

- 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.

8.5.2 Application, UI & Database Management

- This part is Responsible for authenticating the different actors.
- User-Interface will be provided, so that actors can easily use our platform.
- This part will validates files submitted by applicaion developer and platform configurer.
- We will provide Azure MySQL and Azure Repository service to our platform.
- Sensor instance Binding with Application will be done by us.

8.5.3 Node Management and Monitoring & Fault Tolerance

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

8.5.4 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.