Internals of application server

Project Design for IoT Platform

Prepared by: Group 1

Contents

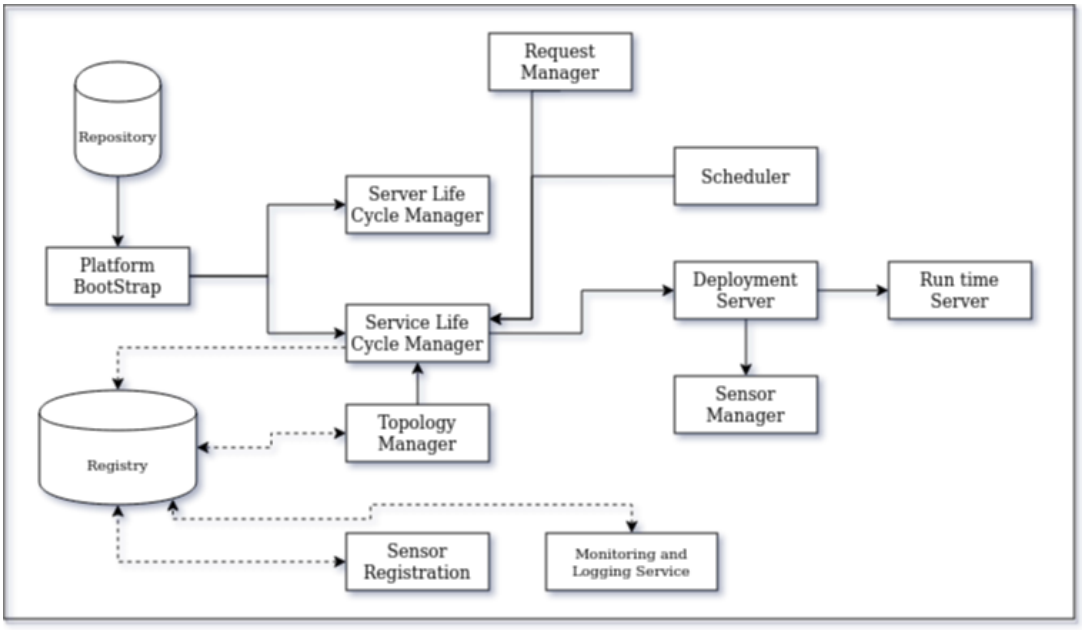
1 Introduction

1.1 Project Introduction

With growing IoT Application respective device data is also growing with similar rate. Each sensor have potential data which can be used to make important decision. Difficulty arises for a developer to keep track of this sensor data and use this data as input in its application.Managing algorithm, binding data and analysing the output are some big task developer has to take care. Aim of this platform is to provide the user with an environment which allows them to create application which can connect to / use location sensitive IoT devices with ease.In addition to this, it also provides the capability to dynamically run user defined algorithms which make use of such devices. This IoT Application platform directly integrates with the IoT devices and is capable of performing remote actions on these devices

1. Providing client with interface where one can upload their application, connect it to suitable IoT devices and run them on required sensor data.
2. Client will be provided with:
    - UI based dashboard, where user can upload and run application.
    - Provide user with set of APIs
    - Algorithm scheduler service to run the application
3. Upon running the application, platform provide user with the following capabilites
    - Binding Service: Binding the sensor with the alogirthm.
    - Deployment Service: Running an executable in response to an event generated by algorithm during its course of execution.
    - Notification Service: Generating notification and pushing it to registered users of the platform.
    - Event Service: In addition to control services over IoT devices the platform provides ability for the app to generate template actions over event generated by algorithm.
    - Scheduling Service: Platform also provides scheduling service which allows the user to periodically run their application.

1.2 Functional Overview:



Platform Initializer

- Responsible to initialize all the available platform modules like Repository,Scheduler,Deployment manager,Sensor Manager etc.

Request Manager

- Dashboard : User Interface to upload/deploy user application
- API: Set of APIs provided to user for deployment
  - This module will serve all request initially given by application developer. Comprises of Login,storing and retrieving algorithms and meta file in .zip format,validating meta file and forwarding the same to scheduler

Server Life Cycle Manager

1. Server Life cycle will accept request from service life cycle manager and provide a running machine with all dependency installed in it required for a service to run.
2. Server Life Cycle will be responsible to add machines to the platform at run time to balance the load of running services.

Service Life Cycle Manager

1. Responsible to Start and Stop service request given by Platform User.

2. Responsible to store all important details of a given service in registry, information like service running on which machine, its start and end time, its container id.

Scheduler

- This module will be responsible to ask developer the scheduling information for algorithms provided.
- Will be responsible to start and kill scheduled algorithms on user specified time.

Deployment Manager

- Responsible for deploying algorithm instances stored in repository to run. Repository contain various configuration files which locate what packages to be deployed. It also take care of system where the algorithm need to be run.
- This module is responsible for initializing the algorithm with configuration data provided using algorithm configuration file.
- Algorithm and Config file : This module fetches the raw algorithm from a repository. Reads the algorithm sensor parameter such as ¡distance,Geolocation,SensorType¿.
- It then queries sensor manager which has information about all the sensor suggested with the platform.

Run Time Server

- Responsible for initiating node instances as requested by deployment manager.
- Run time server will support multiple containers service to run.

Sensor Manager

- This module is responsible for binding correct sensor data stream with corresponding algorithm.
- Sensor Manager,Add sensor are the services provided by communication module
- Will identify IoT devices from the sensor database based upon the parameters.

Sensor Registration

- Sensor Registration will help to initially setup sensors in platform and will store relevant information in run time registry

Topology Manager

- Topology Manager will check load on each machines available in the system and will share appropriate start and stop request to service life cycle manager

Monitoring and Logging Service

- Monitoring service will read data pushed by running service (data like CPU Stats), this service will read data from monitoring topic and will push the same to Run time Registry in correct format.

2 Test Cases

1. Authentication and Authorisation :
   - Check Username and password
     - Input : username and password
     - Output: Success or failure
     - Description : It will verify username and password from DB and return result.
   - Check User access permission
     - Input : Username
     - Output : User's token along with list of services accessible to user
     - Description : It check user access permission from repository and return
2. Request Manager:
   - Request Manager is Down
     - Health check Module will keep on checking the availability of Request Manager
3. Deployment Service
   - Check model is deployed/ Check Model endpoint
     - Input : Given Model API endpoint
     - Output: Got response if model is deployed and running or URL not found if model is not deployed
     - Description : It will check whether model is deployed or not by accessing its end-point. If API is accessible we got response. But if model is not up and running we got 404 page not found error.

4. Scheduling Service :
    - Check model is up between start and end time
        - Input : Start time , Endtime, Model API endpoint.
        - Output: Check Model is schedule and running up between start and end time.
        - Description : It will check whether model is up and running between given time. It will ping model in after every time interval.
5. Binding Service:
    - Check whether model data is packaged properly.
        - Input : Model information,
        - Output: Package model
        - Description : It will check model information in registry and then get the model data and config file from repository. It will create script file and package these files in one zip.
    - Check whether package is received by IoT run machine.
        - Input : Packaged data (Zip file)
        - Output: Send it to un machine.
        - Description : It will send Packaged file to AI run machine where actual model will run.
6. Notification Action Service :
    - Check Action performed or not
        - Input : Model API endpoint action code
        - Output: Check console that action performed and output seen on console or not and notification receive to user registered email/mobile or not.
        - Description : It will check whether user's action code is executed or not after Model had correctly predicted output.
    - Check Notification receive or not
        - Input : Model API endpoint and prediction output.
        - Output: Whether user receive notification on register mobile or email or not.

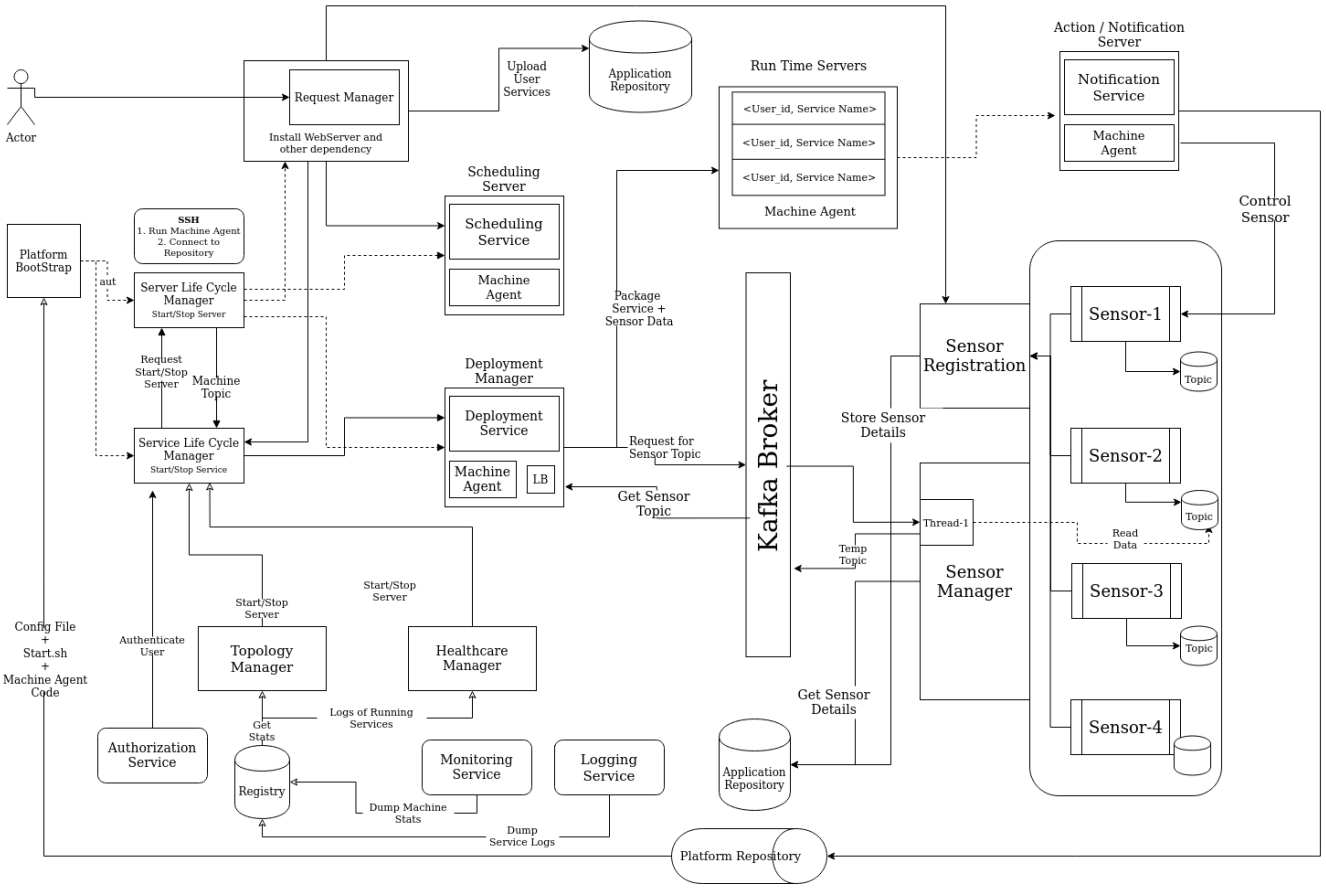
- Description : After model got result and it should notify user about it.Checking whether user receive any notification about it or not.

7. Logging and Monitoring

   - Check log file Content is updated or not
     - Input :Log file path , ServiceName, Logmsg
     - Output: Log File content change or not
     - Description : Check whether logmsg given by service is written on log file or not.

3 System Design

3.1 Design Big Picture



3.2 Technologies/Environment to be used

Flask

Flask is a micro web framework written in Python, python based framework will be used for UI based interfaces and servers.

KAFKA - Messaging Queue

Kafka is used for real-time streams of data, to collect big data, or to do real time analysis (or both). Kafka is used with in-memory micro services to provide durability and it can be used to feed events to CEP (complex event streaming systems) and IoT automation systems.Kafka is often used in real-time streaming data architectures to provide real-time analytic. Since Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system.

Docker

Platform and User Services will be deployed as a docker container, this container will have respective service image running within the container. Docker files will be initially for each services.

MongoDB

MongoDB will be used as a Run time Registry, to store data of platform and user services at any particular instance. MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

Fluentd

Fluentd is an open source data collector for unified logging layer. Fluentd allows you to unify data collection and consumption for a better use and understanding of data.Fluentd treats logs as JSON, a popular machine-readable format.

Bash Shell

Bash shell script will be used for installation and automation of platform services.

Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

3.3 Overall system flow and interactions

1. Starting Platform Bootstrap.
2. Platform initialize will get information about various modules from config files once it is up and running.
3. Using the information from config file, it will initialize all the modules.
4. User will be authenticated using GUI or API to execute his algorithm on the platform.
5. Once the user is authenticated, User will upload a zip format directory containing in-formations like
   - config file
   - Services details
   - Service scheduling information
   - Sensor Details
6. Once user enter this data, there will be a validation module which will check if given input is in correct format.
7. After validation ,on user request service start/stop request will be sent to Service Life Cycle Manager or Scheduler
8. Service LCM will request for a machine to run the service from Server LCM
9. Service LCM will store the information of service in run time registry and will request Deployment manager to run the service at given location
10. Deployment Manager will bind the service with sensor data and will run the wrapped service in Run time Server
11. Once the algorithm is executed on any node/VM, it's result will be sent to action module.
12. Action Module will be communicating with various output sensors and depending on output from the algorithm, it will take the action.
13. When all these modules are communicating with each other, heartbeat monitoring module will check continuously if every module is up and running or not. If any of the node is down or algorithm on any particular node has stopped, heartbeat manager will make sure that the algorithm is either restarted or the algorithm is running on some different node.

3.4 Interaction between modules

Platform Bootstrap and All Other Components

Platform bootstrap will read the config file and performs action based on it. Config file will contain information about which service will run on which machine. Platform Bootstrap will SSH a given machine, then it will copy code from NFS(Repository), will run MachineAgent.py script on it, will then run start.sh script on each machine that will help to run the service on respective machine.

Request Manager and Scheduler and Service Life Cycle

Request Manager interacts with the application developer either through UI or API for the scheduling of algorithms and hence behaves like the interface between scheduler and application developer. Request Manager retrieves the zip file from the developer which contains the algorithms and configuration file , later for scheduling will retrieve meta data file which contains scheduling information and passes it to the scheduler.

Service Life Cycle and Deployment Manager

Service Life Cycle passes the information to the Deployment manager for the execution of the algorithm at the scheduled time and Deployment manager will in return pass the unique id related to the running instance of the algorithm so as to stop the instance in future. As it is responsibility of the scheduler to start and kill the process and hence stores all the id's of the running instance.

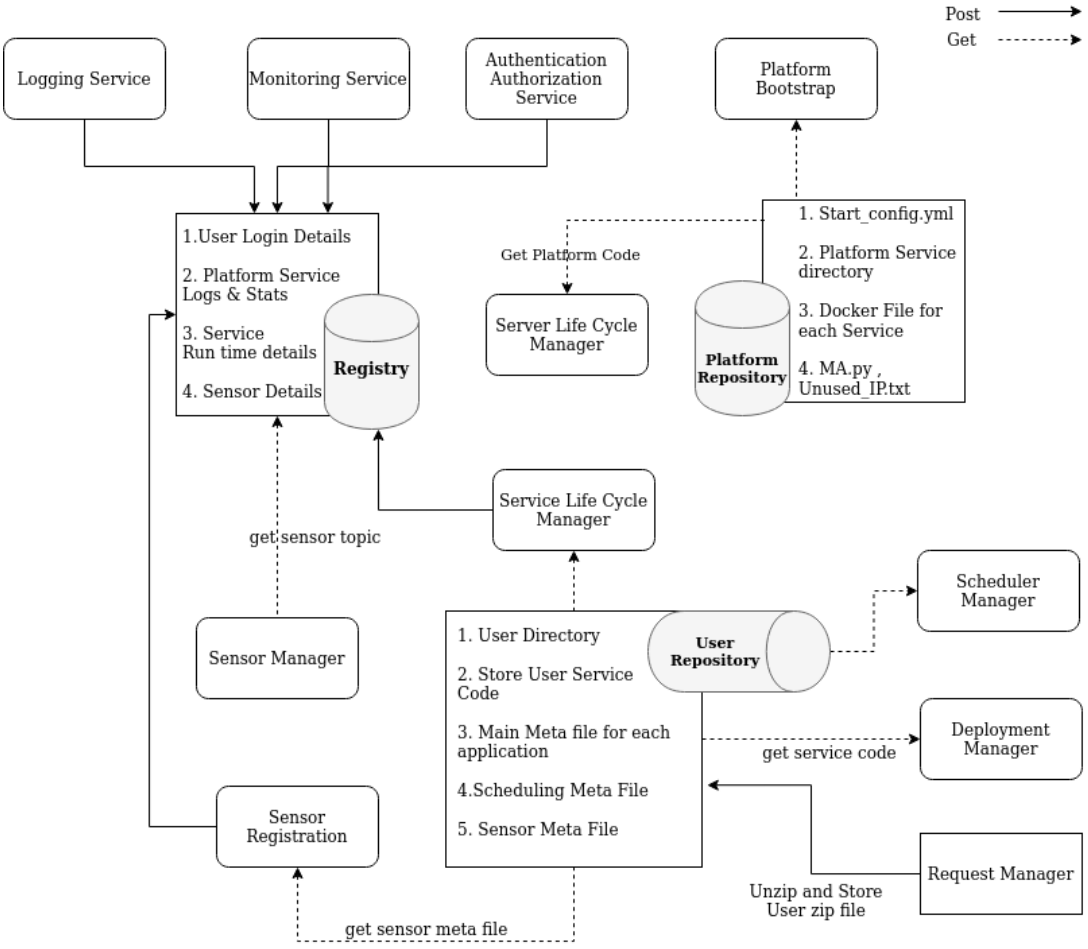
Deployment Manager and Sensor Manager

For the data binding between algorithm and the sensor data ,node which is responsible to run the instance of algorithm will request for the data (query) to the sensor manager .The sensor manager fetches the queries ,creates a topic for communication and dispatches thread which is responsible to control the incoming rate of data input for the algorithm and the thread is also responsible to write the data to the topic at its defined(in query) rate.Hence the sensor manager passes the topic name to the nodes for communication.

Monitoring and Fault Tolerance and All Modules

Monitoring and fault tolerance modules keeps track of all the modules , if it finds any modules is not working , it requests the other node to run that particular module and hence reflects the addresses of the new node in the whole system

3.5 Registry and Repository



Registry

1. Registry will be used to store several run-time related information for applications and platform modules. Mongodb will be used to store such highly unstructured data. In specific mongodb will store following in formations:
    - Login Details
    - Sensor topic
    - Service running details
    - Logs of running services
    - CPU stats of each running servers
    - Url to uniquely identify an algorithm on a node
    - Sensor metadata

2. Major modules interacting with this are:
    - Server Life Cycle Manager
    - Sensor Manager
    - Sensor Registration
    - Deployment Manager

Application Repository

Application repository will store all the static files of applications deployed by all the users with the help of Network file system. This will act as a backing store for all the application codes.
Major modules interacting with this are:

- Scheduler
- Action Module
- Service Life Cycle Manager
- Deployment Manager

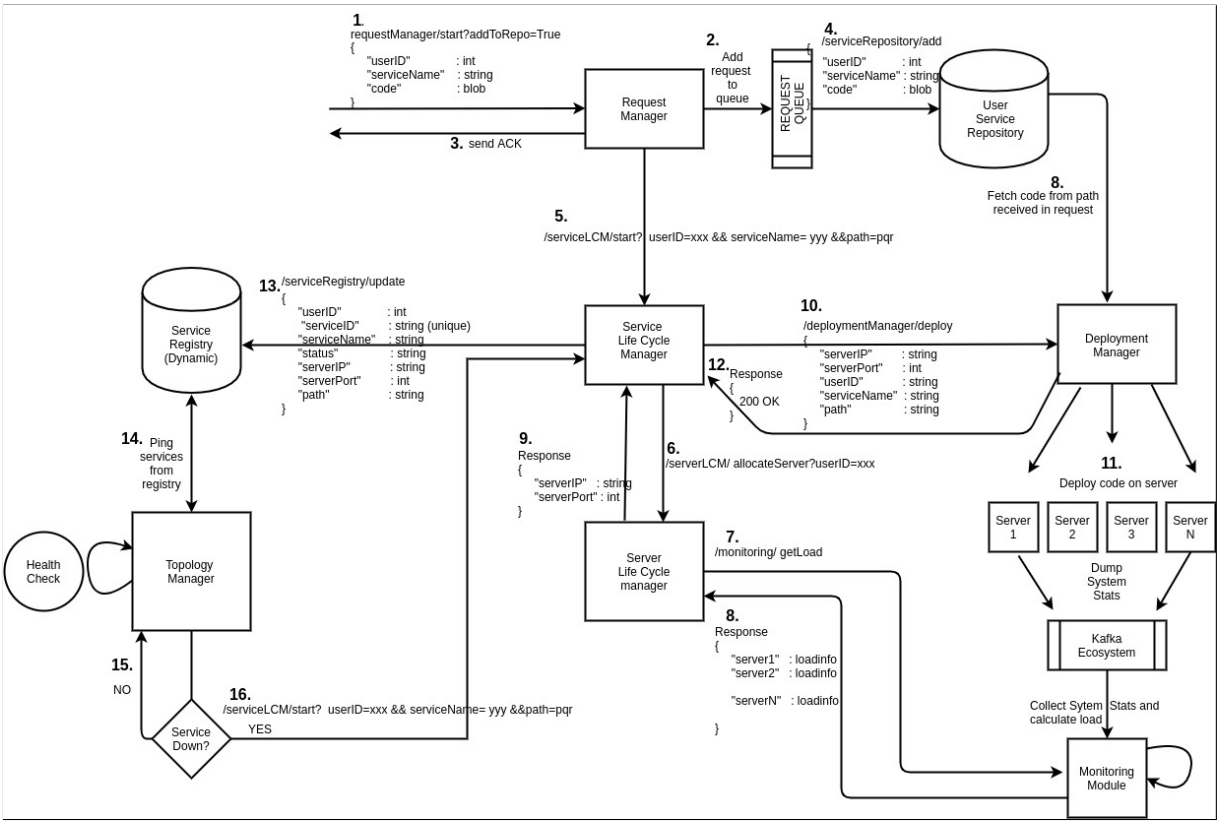
Platform Repository

Platform repository will store all the static files of all the modules of our application server platform via Network file system.
Major modules interacting with this are:

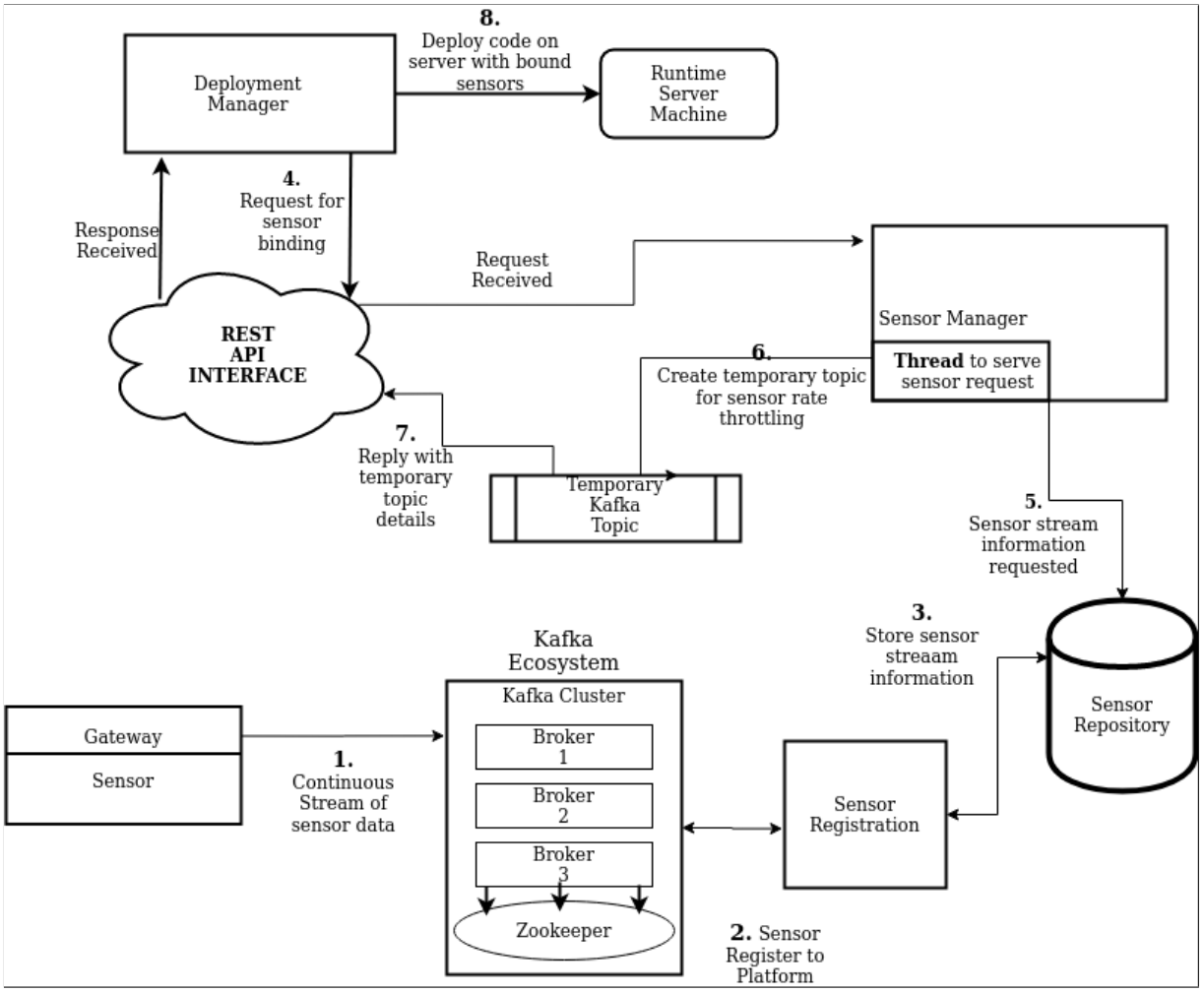
- Platform Bootstrap

3.6 Communication and Connectivity Model

1. Communication Within Modules

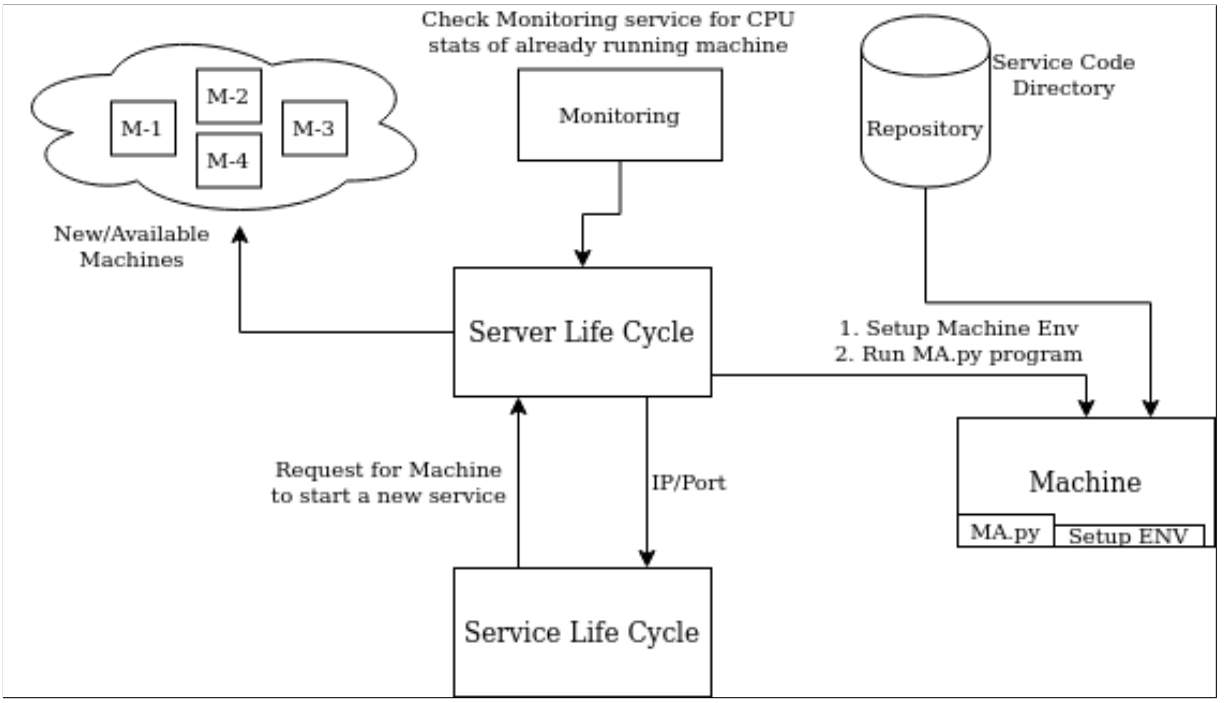


2. Sensor Binding

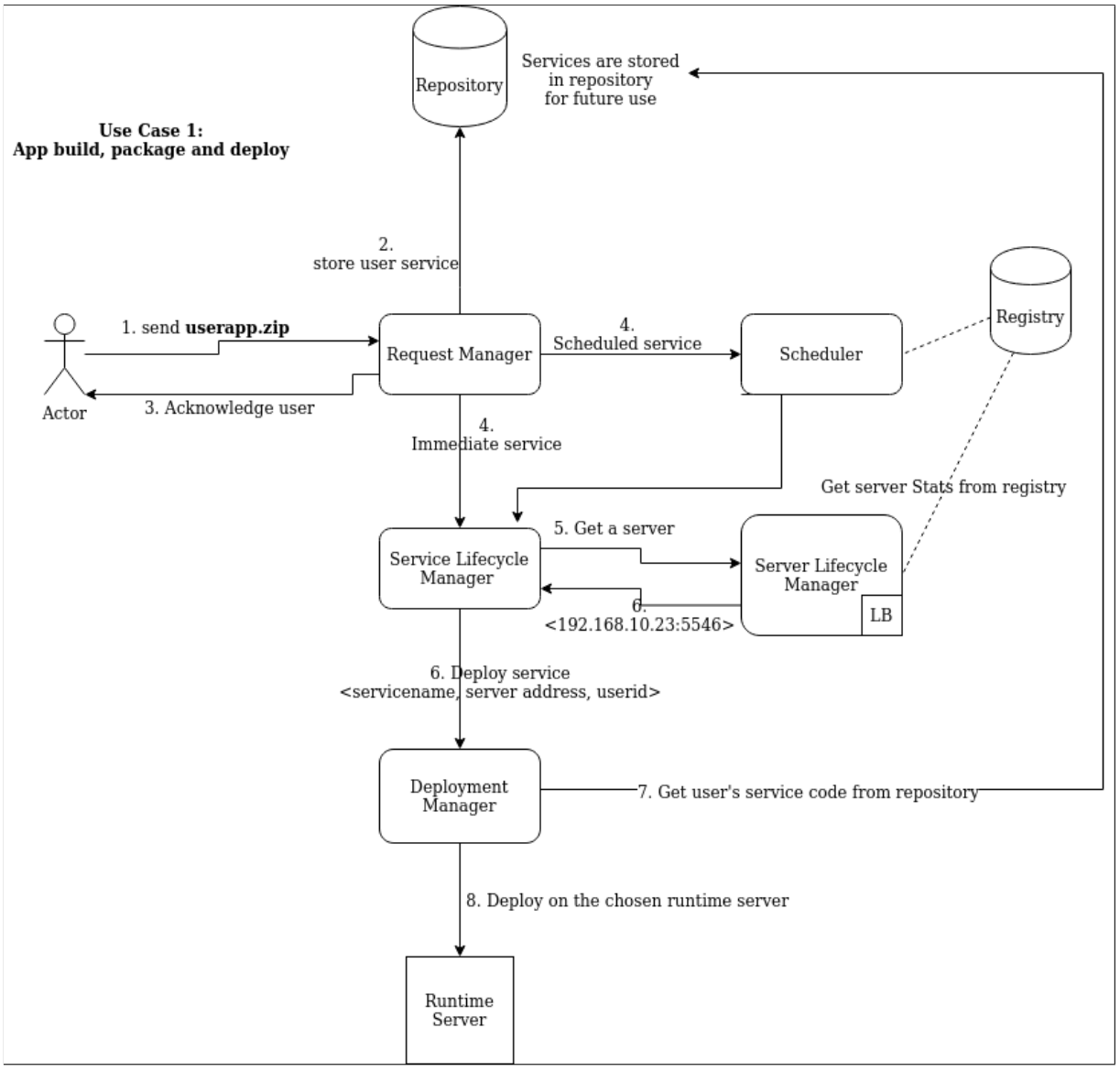


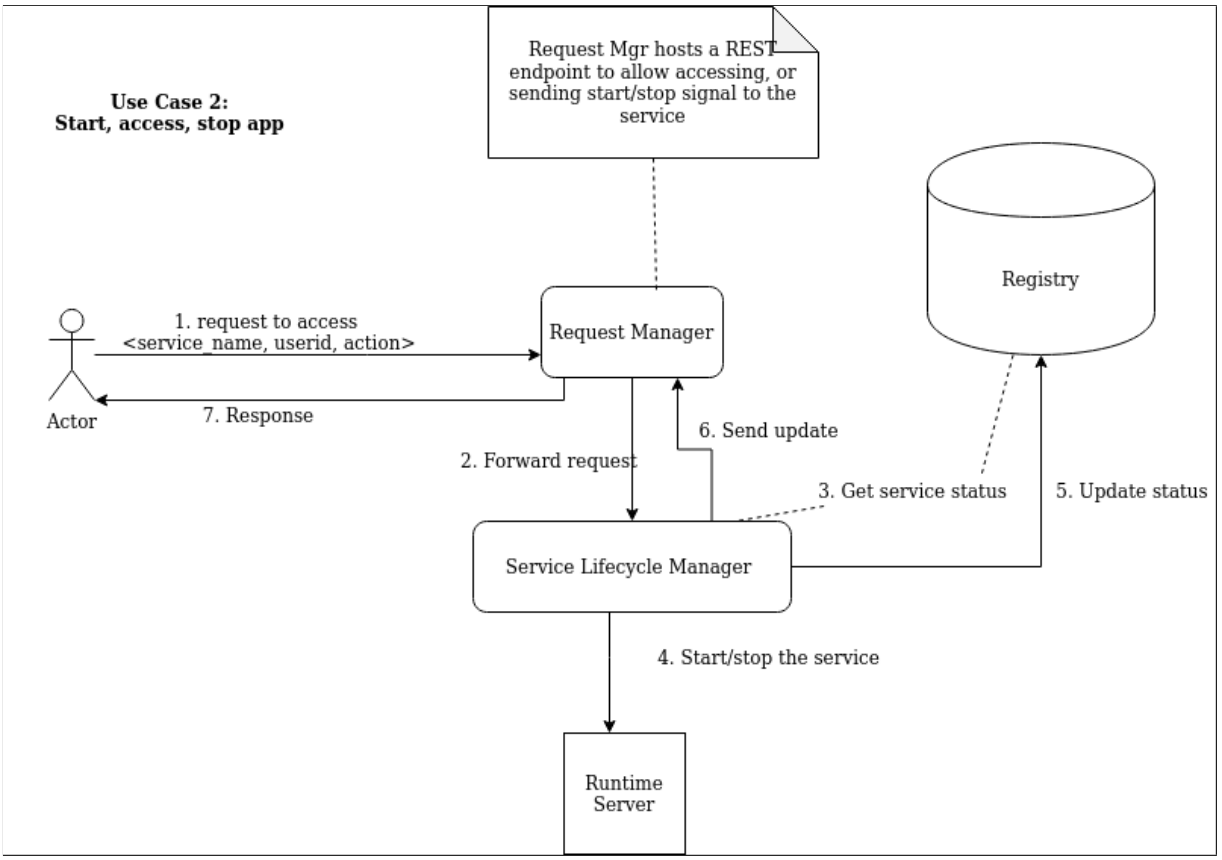
3.7 Server and Service Life Cycle

1. Server Life Cycle



2. Service Life Cycle


Repository
Services are stored
in repository
for future use
Use Case 1:
App build, package and deploy
2.
store user service
1. send userapp.zip
Request Manager
4.
Scheduled service
Scheduler
Registry
Actor
3. Acknowledge user
4.
Immediate service
Get server Stats from registry
5. Get a server
Service Lifecycle
Manager
Server Lifecycle
Manager
LB
6.
<192.168.10.23:5546>
6. Deploy service
<servicename, server address, userid>
Deployment
Manager
7. Get user's service code from repository
8. Deploy on the chosen runtime server
Runtime
Server


Use Case 2:
Start, access, stop app
Request Mgr hosts a REST endpoint to allow accessing, or sending start/stop signal to the service
Registry
Request Manager
1. request to access
<service_name, userid, action>
Actor
7. Response
2. Forward request
6. Send update
3. Get service status
5. Update status
Service Lifecycle Manager
4. Start/stop the service
Runtime Server

3.8 Meta formats

1. Request Manager
    - Meta file provided by User to the platform

```
<Services>
    <service>
        <name>service1</name>
        <filename>service1.py</filename>
        <priority>low</priority>

        <dependencies>
            <dependency>service2</dependency>
            <dependency>service3</dependency>
        </dependencies>
        <sensor_input>
            <input>
                <type>BY_ID</type>
                <id>12345</id>
            </input>
            <input>
                <type>BY_LAT_LON</type>
                <lat>1234.12</lat>
                <lon>24.213</lon>
                <floor>2</floor>
                <radius>12</radius>
                <count>ALL</count>
            </input>
        </sensor_input>
        <scheduling_info>
            <file_name>scheduling1.xml</file_name>
        </scheduling_info>
    </service>
</Services>
```

2. Scheduler Manager:
    - User can upload meta file containing scheduling information

Scheduling File Format

```
<schedules>
    <schedule>
        <!-- Schedule 1 Info -->
    </schedule>
    <schedule>
        <!-- Schedule 2 Info -->
    </schedule>
</schedules>
```

Different Types Of schedules
SINGLE INSTANCE

```
<schedules>
    <schedule>
        <type>SINGLE_INSTANCE</type>
        <start_date>12-12-2020</start_date>
        <time>
            <start>10:00</start>
            <end>12:00</end>
        </time>
    </schedule>
</schedules>
```

DayWise

```
<schedule>
    <type>DAYWISE</type>
    <day>MONDAY</day>
    <time>
        <start>10:00</start>
        <end>12:00</end>
    </time>
</schedule>
```

Periodic

```
<schedule>
    <type>PERIODIC</type>
    <start_date>12-12-2020</start_date>
    <period>10</period>
    <time>
        <start>10:00</start>
        <end>12:00</end>
    </time>
</schedule>
```

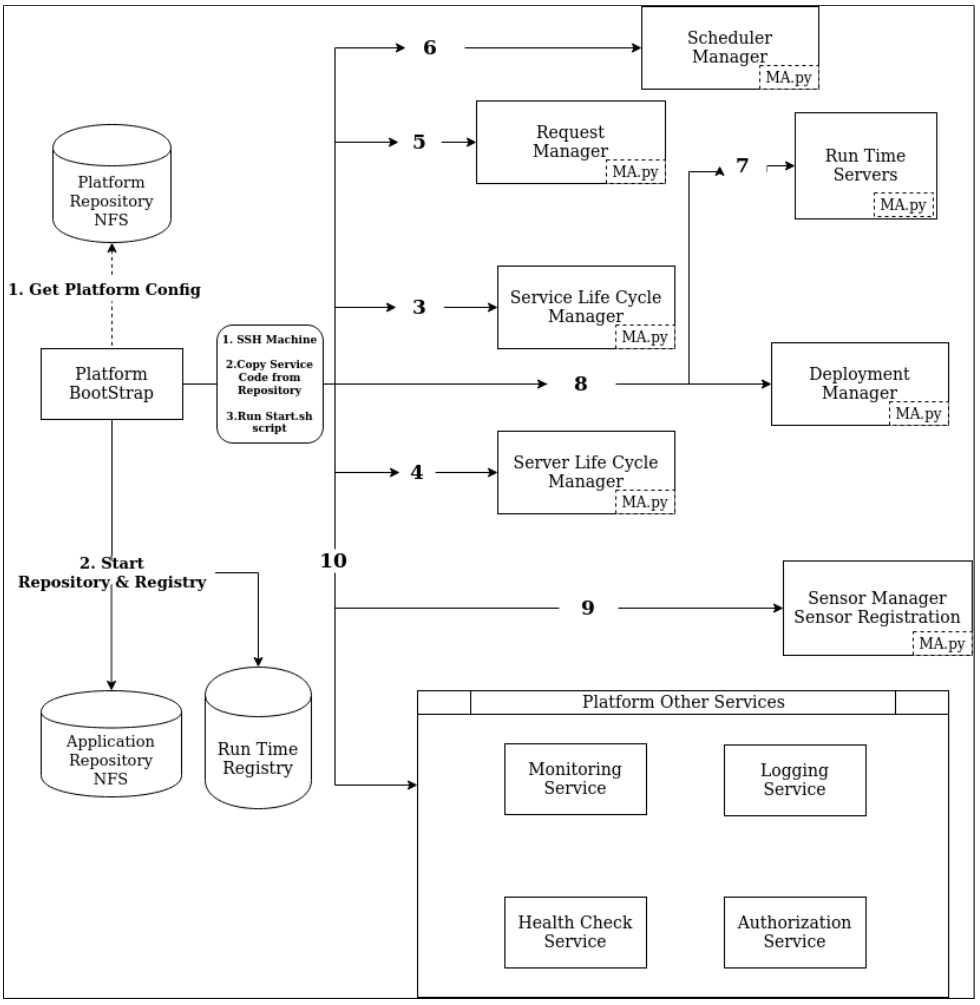
3. Sensor Registration:

   - Admin can upload meta file consiting of initial sensor details required during registration

```
<sensors>
    <sensor>
        <name>BTF-14</name>
        <gateway>ip:port</gateway>
        <sensor_id>SI12</sensor_id>
        <data_type>VECTOR</data_type>
        <size> 20 </size>
        <address>
            <area> iiit </area>
            <building_name>nilgiri</building_name>
            <room_no>D12</room_no>
        </address>
        <geo_location>
            <lat>1234.242</lat>
            <lon>854.21</lon>
            <floor_no>4</floor_no>
        </geo_location>
        <type>INPUT_OUTPUT</type>
    </sensor>
</sensors>
```

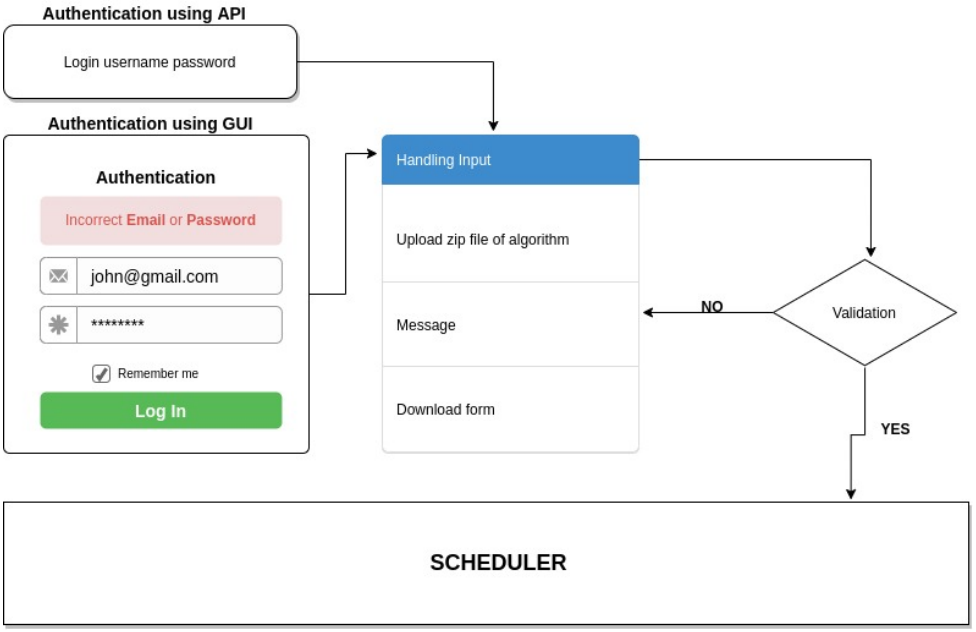
4 Low level design of each module

4.1 Platform Bootstrap



- It reads the configuration file when it is up and running and gets the information of all modules.
- It initializes all the modules using the gathered information after which the algorithm can be executed.

4.2 Application Manager



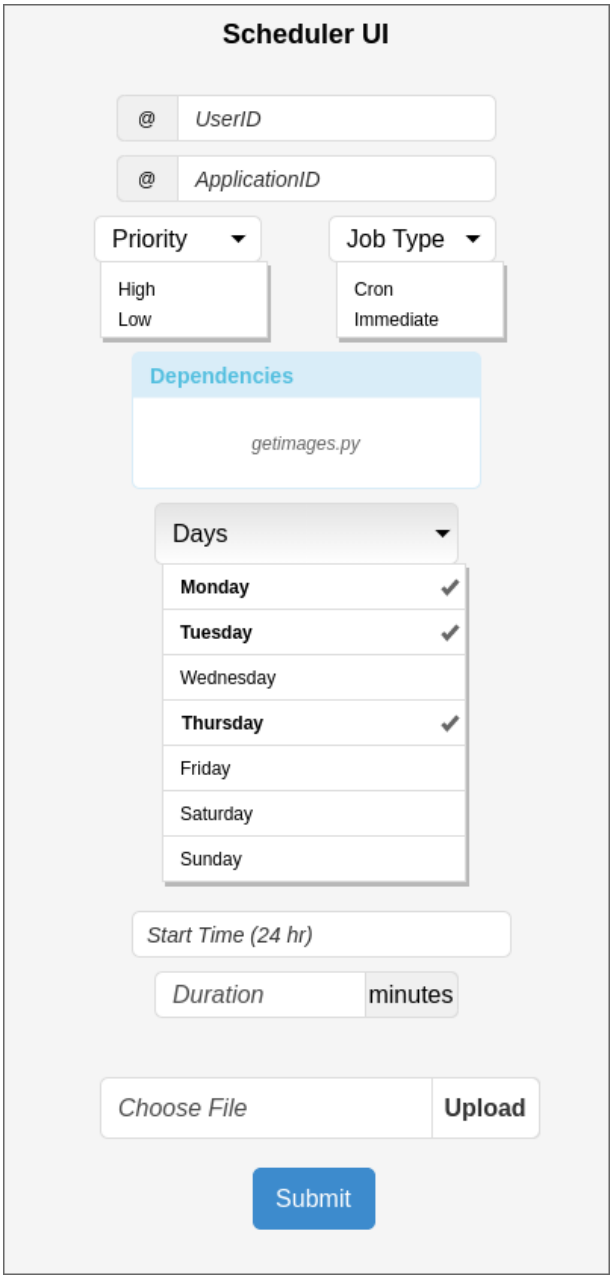
- Sign up : Application developer can use this module for sign up
- Login : Application developer can login through the login module.
- Dashboard : Application developer can upload algorithms and meta files in zip file by dashboard. He can download the format of meta file for the algorithm from Dashboard.
- Validater : The format of meta file uploaded by developer will be verified by Validater if format is not proper error message will be displayed in Dashboard. Else algorithms and meta files will be stored in Repository.

4.3 Scheduler

Job Scheduler module is responsible to run/send jobs to deployment manager periodically at predetermined intervals.The scheduler will receive the scheduling information in json/xml format to get scheduling related metadata after the application is being accepted by Application Manager. For our platform user can specify this through both UI or upload configuration file.

Configuration file should contain following meta-information:

- Userid: unique user id for the user.
- Applicationid: unique id of the application of that user.

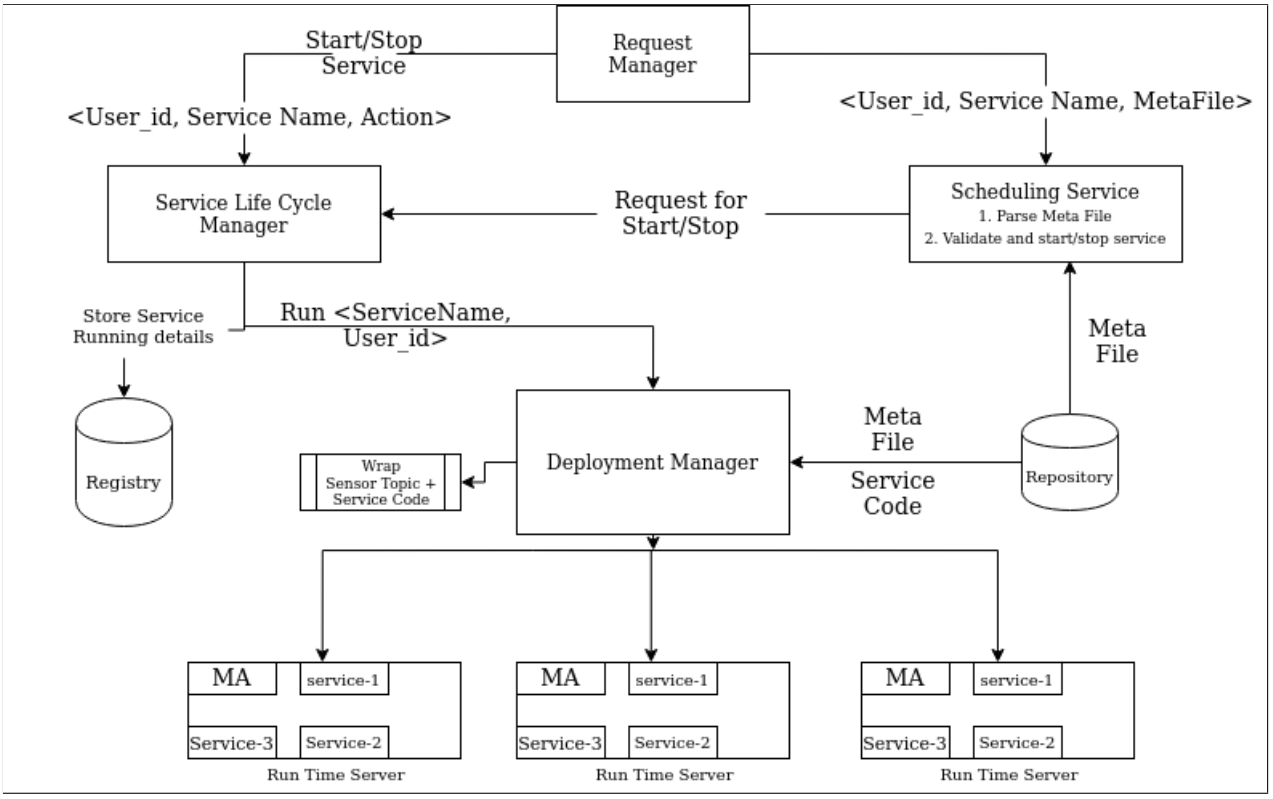


- Priority: This will specify if the algorithm is computation intensive or requires a high amount of resources.
  - If specified "high" the algorithm will run on a single node.
  - If specified "low"(default) the algorithm will run on shared nodes.
- Dependencies: Will specify the dependencies that must be run before running the algorithm strictly in the order they are required in. Scheduler will schedule the dependencies in the same order specified then will schedule the algorithm.

- Jobtype: This will be used by the scheduler to understand how to schedule the algorithm. Two choices are cron single:
  - If specified as a cron job scheduler will schedule it at periodic intervals.
    - Application Deployer can specify day/days at which the algorithm will be scheduled.
    - Also start time(in 24 hours format) and duration for which it should run(in minutes) will be specified.
  - If specified as immediate instance user have two choices:
    - If no start time given the scheduler will immediately schedule it to execution for the time duration specified.
    - If given a start time scheduler will schedule it at the given time for the time duration specified.

4.4 Deployment Manager

- Overview of each sub module and their interaction



Deployer will get an algorithm execution request from the scheduler. Deployer will apply a load balancing algorithm, if a node is available the algorithm will be executed on that node. Otherwise, the Deployer will request for a new node to Node manager and then run the algorithm on the newly allocated node. Deployer may ask for one or more nodes to Node Manager.

Heartbeat Manager is one service of Deployer which keeps checking whether all nodes are alive or not and whether the algorithm inside node is running or not. If the algorithm inside the node is not working, the Deployer will re-execute the algorithm in that node. And if node is not working, the Deployer will run the algorithm at another node.

Load balancing algorithm :
It will store this in a map data structure mapping IP address to the cpu usage and RAM usage of node.

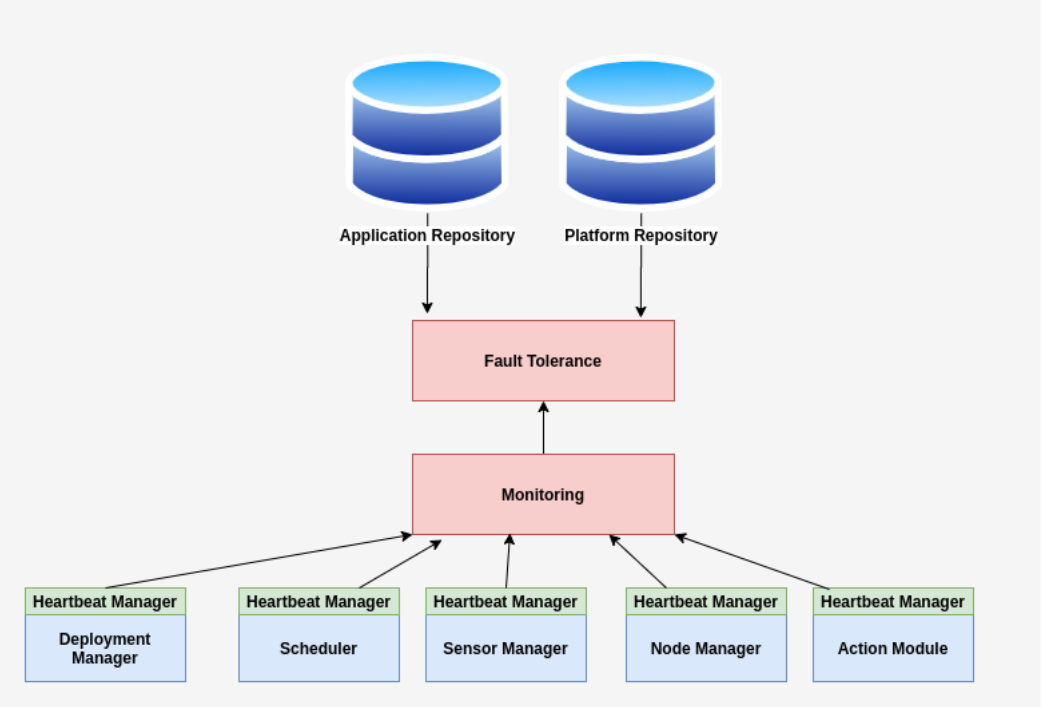
Load of node = 1 / ((1 / cpu usage) + (1 / RAM usage))

Deployer will calculate the load of each active node and execute the algorithm with minimum load.

Load balancer will request for nodes to Node Manager. Node Manager allocates new nodes to Load balancer for execution of an algorithm and sends a list of active nodes to load balancer. It will start the new node and load the init file to node. Init file is responsible for communication between node and load balancer. It can allocate one or more nodes based on requirements of load balancer.

4.5 Monitoring and Fault Tolerance Module

The Monitoring and Fault Tolerance Module is responsible to continuously monitor status of all the modules viz scheduler,deployment manager , sensor manager, application manager, action manager and their corresponding nodes on which they are running.
If monitoring system detects abnormal behaviour of any module, it finds out whether the nodes on which module is running is not working or the instance of module itself is not working.
For the first case it will request fault tolerant manager to run same module on different node and reflect changes if needed everywhere and for the second case it will request the fault tolerant manager to run the instance of the module on the same node.

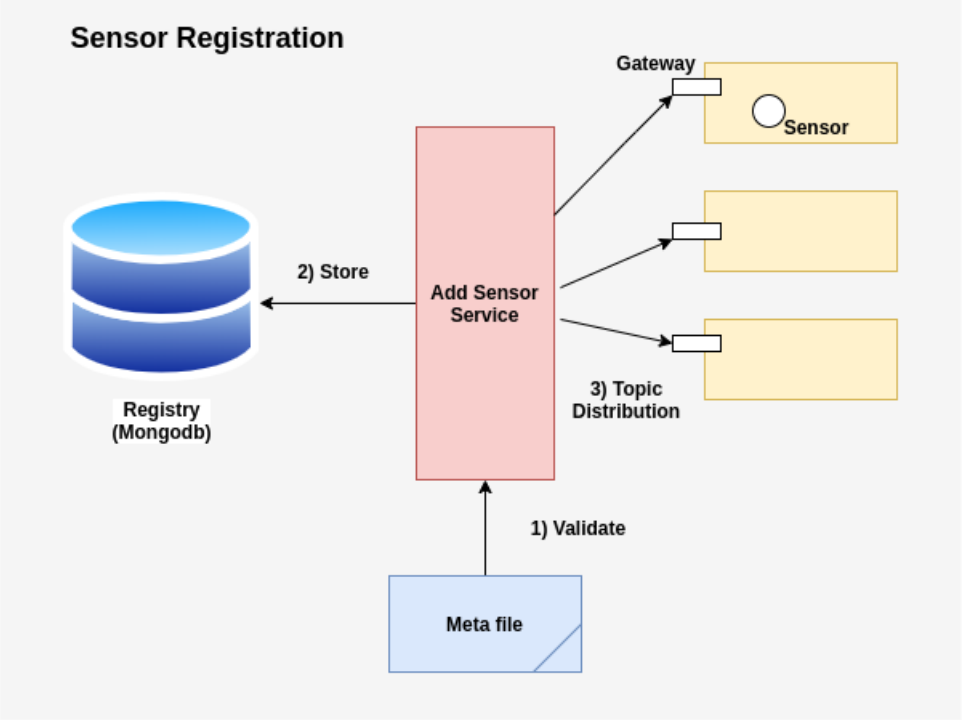


For every node running module ,there will a background service which will continuously write on the communication topic (node and Monitoring Module) which lets the monitoring module know whether the node is live or not and also will write the state of the process(module) whether running or not.
If there is abnormality and the node is still writing on the topic it simply implies the node is fine but the module has stopped working and hence requests fault tolerant manager to run module on same node.
If the topic is not getting new message ,which implies node is not working hence will request fault tolerant manager to run module on different node and reflect all the changes in the system.

4.6 Sensor Manager

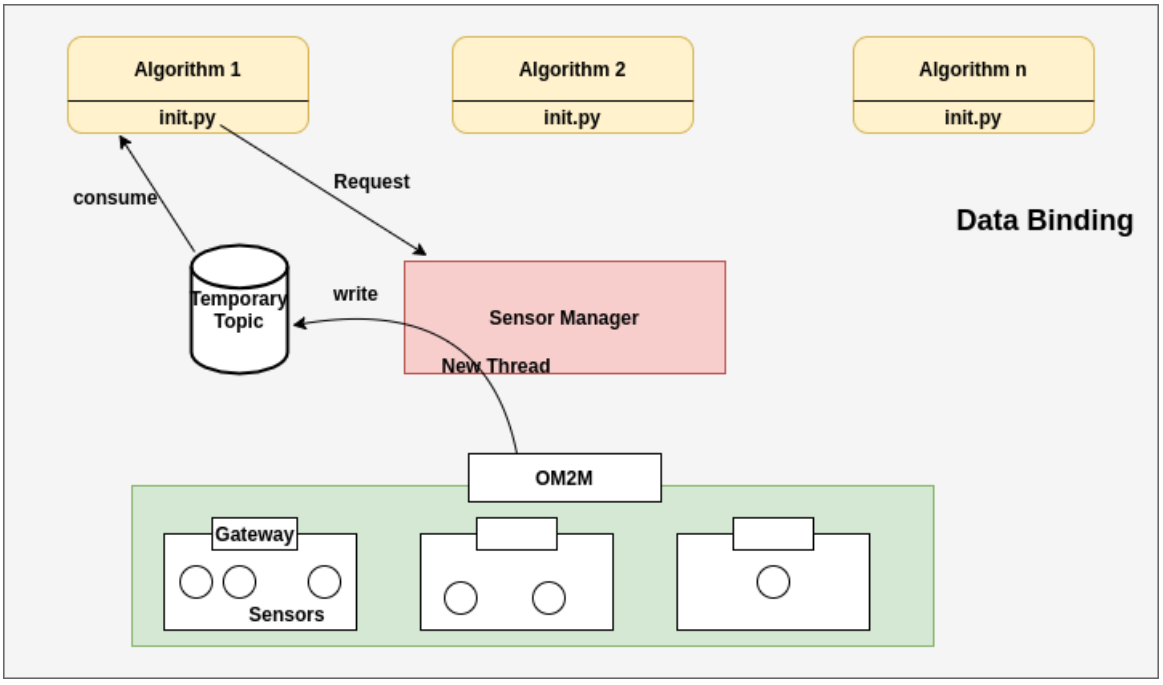
Sensor Registration Service

1. Admin will upload a .json file containing information/mapping of sensor gateway with kafka topic and other sensor details.
2. Module will be responsible to parse the meta file provided by the admin containing sensor information. This service will help register the sensor details and will create a unique topic for data to be dumped for each sensor. Each sensor gateway will receive a topic for data dumping.



Sensor Manager

1. Responsible for validating whether requested sensor details are available in system or not.
2. If available it will get its details from the registry.
3. From the repository it will read the sensor topic over which sensor is dumping the data.
4. Further SM will send this data and appropriately will send the data to Algorithm.



5 Use Cases

5.1 Types of Applications that can be built

This platform can be used to build an IOT application which performs some action based on collected sensor data. The application will have the following characteristics:-
1. The app directly integrates with the IOT devices and is capable of performing remote actions on these devices.
2. In order to achieve this the platform allows the app to specify a config file which contains details about the location of the sensor where the app is to be deployed as well as their number and type.

5.2 List what the users can do with the solution.

1. Control sensor based on the data generated and result analysis
2. Smart decision based during algorithm execution
3. Can generate Callbacks/Over Events based on sensor data.

5.3 Usage scenarios

- Mess Overcrowding Management

- AC Controlling System:
- Smart Parking Solution
- Automated Street Light :
- Classroom Lighting based on Occupancy:

5.4 Test application

The objective of the application is to detect overcrowding in mess in real time during scheduled mess timings and update on common mess portal

Assumption : Admin had already done setup for all the sensors along with their properties viz. geo location , sensor type etc.

System Implementation:

- **Algorithm**
  1. Users of the platform provide the code to be run for mess crowd management. This artefact has the business logic for the use case and is provided to the platform via UI/API. In addition to this a configuration file is supplied along with the algorithm which contains the dependencies and the system environment requirement.
- **Binding Algorithm with Sensors**
  2. Algorithms take into input the number and type of sensors along with their geo-code. The platform then determines the sensors registered with the it within the vicinity of the location. In this case, the sensors are cameras located in the respective mess on which the surveillance is to be done.
- **Determining sensors from their geo location**
  3. Once the cameras are identified, the platform creates a topic for each of them. These act as a channel for the algorithms to access the sensor.
  4. After the topics are created, the platform generates a properties file which has information about the kafka topics created for each camera. The algorithm artefact reads this file to gain metadata required to establish connection with these sensors.
  5. Now, that the sensors have been uniquely identified in the system and bound with their respective application, the platform proceeds with running and deploying the code.
- **Deploying the Application**
  6. The Deployment Module passes the request to run the code to a load balancer, which determines the node on which to run the program or fires up a new instance

to serve the request.Running the Application
7. The Runner Module the checks for the environment required by the application as specified in its configuration file. It reads this file and makes sure that the system is ready before running the program.
8. The application is the run on the node.

- **Scheduler**
  9. The platform provides the user with a module to schedule the crowd management application at various time intervals. This comes in handy when the application needs to be run everyday, thrice a day.
  10. Once the application is scheduled to run, this module invokes the deployment and runner module in order to deploy and run the application.

- **Output and Action Templates**
  11. As the mess crowd management application runs it generates certain callback actions once it identifies/infers that the mess has been crowded.
  12. Platform provides the user application with a predefined template of actions that it can take depending upon the crowd. These include notifying the registered mess users of a crowded mess and suggesting a congestion-free time.

6 Existing systems

	Aws IOT platform	Azure iot platform	Oracle IOT dashboard	Bosch	Salesforce IOT Cloud	Our Project
Sensor discovery	√	X	√	√	√	√
Repository management	X	√	√	X	X	√
Notification Generator	√	√	√	Not mentioned	√	√
Events and Callback	Not mentioned	X	Not mentioned	√	√	√
Scheduler	√	√	X	X	Not mentioned	√
Application deployment	√	√	√	√	√	√

- **AWS IOT Platform :** AWS IoT provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded micro-controllers, or smart appliances and the AWS Cloud.

- **Azure iot platform :** Azure IoT is built on decades of Microsoft enterprise experience and is designed to be accessible for all organizations.

- **Oracle IOT dashboard :** Oracle Internet of Things Cloud Service enables secure and reliable bidirectional communication between IoT devices and the cloud.

- **Bosch :** The Bosch IoT Suite is a flexible IoT platform that comprises an array of cloud-enabled services and software packages and addresses the typical requirements of IoT projects.

- **Salesforce IOT Cloud:** Salesforce IoT loud turns the big data coming from customers' connected things into meaningful insights helping to optimize products.