Design Requirements For IoT Based Application Platform

Internals Of Application Server

Submitted By : Group 6

Index

1. Introduction to the Project

Introduction

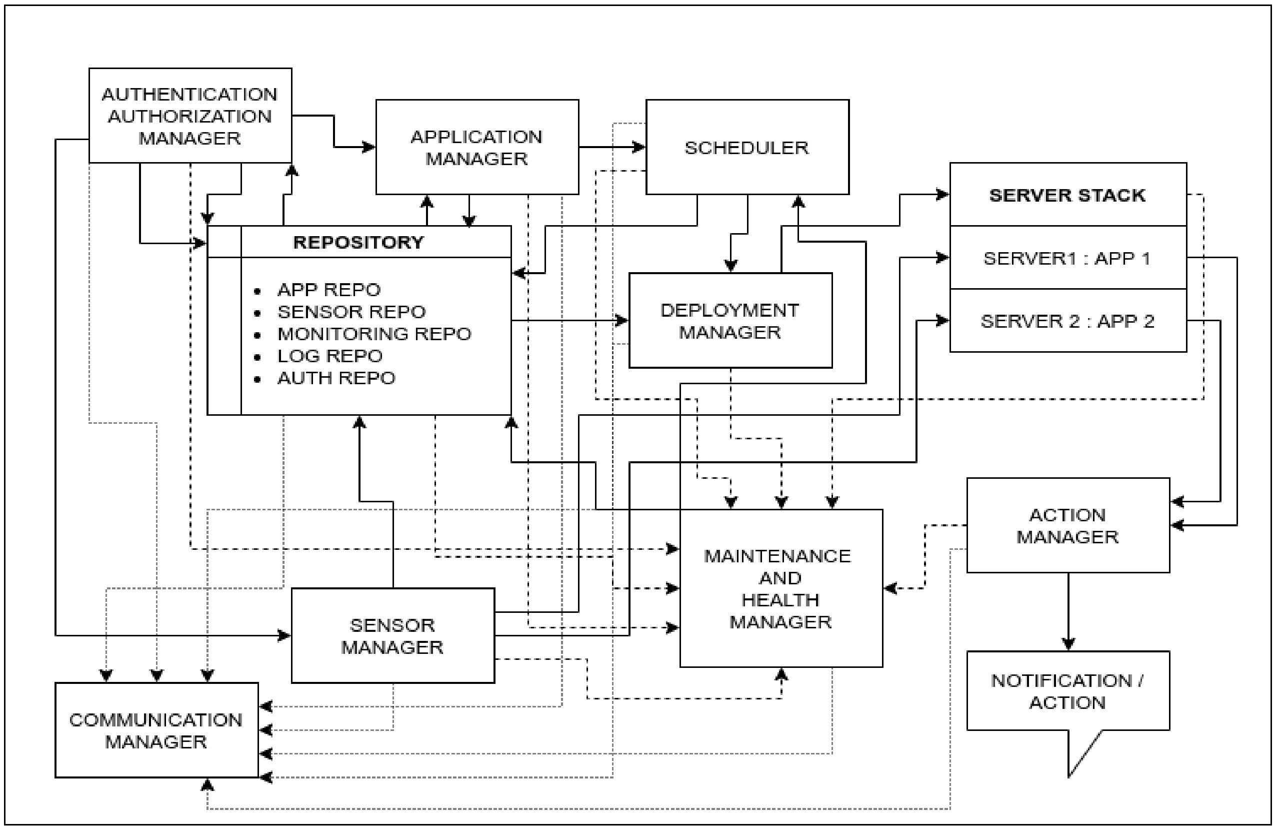
The Internet of Things (IoT) strives to connect devices remotely for seamless functioning and ease of operations. For developers, an IoT platform provides a set of ready-to-use features that speed up development of applications for connected devices as well as take care of scalability and cross-device compatibility and allows developers to spread out the applications, remotely collect data, secure connectivity, and execute sensor management.

Thus, an IoT platform can be wearing different hats depending on how you look at it. It is commonly referred to as middleware when we talk about how it connects remote devices to user applications (or other devices) and manages all the interactions between the hardware and the application layers. It is also known as a cloud enablement platform or IoT enablement platform to pinpoint its major business value, that is empowering standard devices with cloud-based applications and services. Finally, under the name of the IoT application enablement platform, it shifts the focus to being a key tool for IoT developers.

Some Features Provided are:

- ★ Interface for uploading an Application Package to deploy the application.
- ★ Interface for analyzing health of the platform.
- ★ Interface for scheduling various algorithms of applications.
- ★ Interface for managing and uploading details about various sensors deployed.
- ★ Deployer service for Load Balancing and deployment of algorithms.
- ★ Security service for managing user authentication and authorization.
- ★ Scheduler service for scheduling applications and algorithms.
- ★ Heath Manager service for analytics and fault tolerance.
- ★ Sensor manager service for sensor data binding and sensor management and registration.
- ★ Action manager service for getting a variety of outputs and to execute actions based on algorithm requirements.

Functional Overview



Communication Module, Sensor Manager

Communication module: This will provide communication between sensor module and algorithm manager.

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.

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.

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

★ **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.

Application Manager: The application manager will provide a Web GUI to the developer.

The main functions of the application manager are -

★ **Authentication –** The application manager will authenticate the user request by verifying the user id and password.
★ **User Registration –** The User Registers on the platform using the signup link provided in the web GUI.
★ **Validation –** Validating the user file format.
★ **User Interface -** Provide Interface to users for uploading, deploying and running applications.

Scheduler: The scheduler maintains the schedule of each algorithm.

The main functions of scheduler are -

★ **Scheduling –** Scheduling the algorithm and sending the execution command to the deployer.

Deployment Manager:

Services Provided by the module Deployment Manager are:-

- ★ **Dockerizing the application, installing its dependencies**
- ★ **Load Balancing for Deployment**
- ★ **Deployment of the application on the server**
- ★ **Fetching the logs of the application**
- ★ **Monitoring the status of the application and updating in DB**
- ★ **Fault tolerance of the applications**
- ★ **Monitoring the status and stats of the servers**
- ★ **Fault tolerance of the servers**

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.
ServiceLCM updates all its information into DB.

Server LifeCycle Manager

Server LifeCycle Manager monitors the status of the servers and fetches the stats from them, ServerLCM will restart a server if it goes down and restart the applications running on them.
ServerLCM will update this info into the DB.

Load Balancer

Load Balancer will fetch the stats of the server from MongoDB, choose the one with the least load and return its IP once the ServiceLCM asks for it for its deployment.

2. Use Cases

2.1 Usage Scenarios:

★ Smart home system application -

★ Traffic analysis system -

★ Smart City -

★ Smart Marketing -

★ Automated Parking system -

★ Health Monitoring - Glucometer, Blood Oxygen level,

2.2 List of what the users can do with the solution

1. Users can take decisions 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. The platform will automatically scale the Application and nodes depending on the load of the overall applications given to the platform. Each application will be isolated and contained from other running applications.
4. Users can bind new sensors to there which are at various locations and are of several types
5. Users can also get event-based message and email alerts from the platform to monitor and get notified after certain events.

2.3 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 who is going to use the services provided by the application deployed on the platform**.**

2.4 Requirements

The main requirements of the platform are as follows:

- The platform must support a class of applications and the design of the platform must be generalized for a set of IOT use cases.
- It should support various types of sensors with an arbitrary parameter and must support binding them to the platform. It should also provide support for the developers to use the platform and communicate with the sensors.
- The platform must support the deployment of algorithms chosen by the user and must provide a way for the application developers to communicate with the sensors. In other words the data from the sensors should be binded to the algorithm according to the application developer choice.
- Platform must support a set of predefined output formats for algorithms and must provide a way for the application developer to trigger them.
- Platform should be able to recover from any failures automatically.

3. Test Cases

Scheduler:

Test case1: Run the algorithms as per the schedule given by the user.

Test case2: Execute the algorithm if the user wants to run now.

Authentication and Authorization

Test Case 1 Authentication: - Check the user credentials before giving access to the platform / resources inside the platform.

Input: - User credentials.

Output: - Success/Failure message.

Test Case 2 Authorization: - Check whether the user is able to access resources which are authorized to his role.

For example, the algorithm developer should not be able to modify certain platform configs which should be available to platform manager

Input: - Username

Output: - Resources which can be accessed by the user in json format

Application Manager

Test case1: In case of invalid config files, it will reject and display as invalid files on the dashboard.

Test case 2: Show output of currently running algorithm.

Bootstrap:

Test case1: Check if the initializing program can understand the config files and load modules in sequence.

Test case2: Check if it can install all the basic software to make the system runnable.

Action/Notification Module:

Test case1: Check if it can receive the data from the Message queue given by the runtime server.

Test case2: Check if it can interpret the data and decide to send the notification or to perform some action.

Test case3: If action is to be performed, check if the communication between the control unit and Action server is functional or not.

Deployment Manager:

Test case1: Integration with Sensor module: - Check if applications running inside the containers can bind the data with the sensor module.

Input: - Given a sample algorithm

Output: - Check if the data being sent by the sensor is correctly received by the algorithm and the binding of sensors are done properly (kafka topics are being accessed from inside the docker containers).

Test Case2: Integration with Scheduler: - Check if the deployer can deploy the applications which are sent as a request to scheduler.

Input: Deployment request in Json format

Output: Applications have been deployed in the cluster.

Test Case 3: - Integration with the action module: - Check if the algorithm that is running inside the container can trigger specified output by integrating with the Action module Api.

Input: - Given the action endpoint and action parameters

Output: - Check if the action module is being able to trigger appropriate action using the endpoint.

Test case 5: - Kill a particular job that has been deployed

Input: - Algorithm id of the algorithm to be killed.

Output: - Status message of the kill command (either success/ failure / invalid id etc)

Description: -   After the scheduler requests the deployer to deploy, The algorithm developer can request the scheduler to kill a particular algorithm and the deployer will kill the application.

Test case 6:- Fault Tolerance

Input :- manually kill a Container where certain algorithms are running

Output :- The fault tolerance should detect this and reinstate the crashed algorithms

Description :- this test case tests for the robustness of the fault tolerance module and its ability to recover from failed / unresponsive worker nodes.

Test case 7:- Monitoring status of Application

Monitor all the applications running in all the deployment nodes.

Input :- Check if the ServiceLCM is accumulating status of applications running inside docker correctly
Output :- Look at the mongo DB collection for the appropriate updation of the containers.
Test Case 8 :- Get logs of Application
Check if the scheduler is able to properly get the logs of the applications.
Input :- Get logs request from scheduler.
Output :- check if the application logs are proper and are matching with the real application logs.

Sensor Module:

Test Case 1: - Controlling the sensor – checking if the command sent to sensor is working correctly or not.

Input – Action (like switching off AC)

Output – Checking if AC responded or not

Test Case 2: - Checking if registration of new sensor is successful or not?

Input – Sensor information via json file

Output – Sensor simulated or not

Test Case 3: - checking simulation of real time sensor, sending continuous data

Input - sensor type and location of sensor sending Realtime data

Output – stream of data sent to algo and corresponding analysis.

Test Case 4: - extracting data from sensors based on geographical location and type of sensor.

Input - Geographical location and Sensor type

Output – All sensor ids with given location and type

Test Case 5: - Checking if data sent to application via communication module is successful or not.

Input – Data from sensor to communication module

Output - output of corresponding algorithm matches the data that is sent by the sensor

Communication Module:

Test Case1: - If two modules are involved in communication then they are able to send and receive data from each other.

Input: One end will send message

Output: The same message should be received by the other end.

Test Case2: - Parsing is the second case while testing for communication module we should have check for data parsing happening correctly or not.

Input: actual data

Output: parsed data

Test Case3: - How to handle load and manage communication?

Test Case4: - How different types of data to be handled from different sensors in case of sensor data.

Test Case5: - How to handle dynamic behavior of Kafka.

Monitoring & Fault tolerance:

Test Case1: - This service will communicate with different components and get the status of all the components of the system.

Input: Connection with all the modules/components in order to get status.

Output: Will get status of each components and update it in the database.

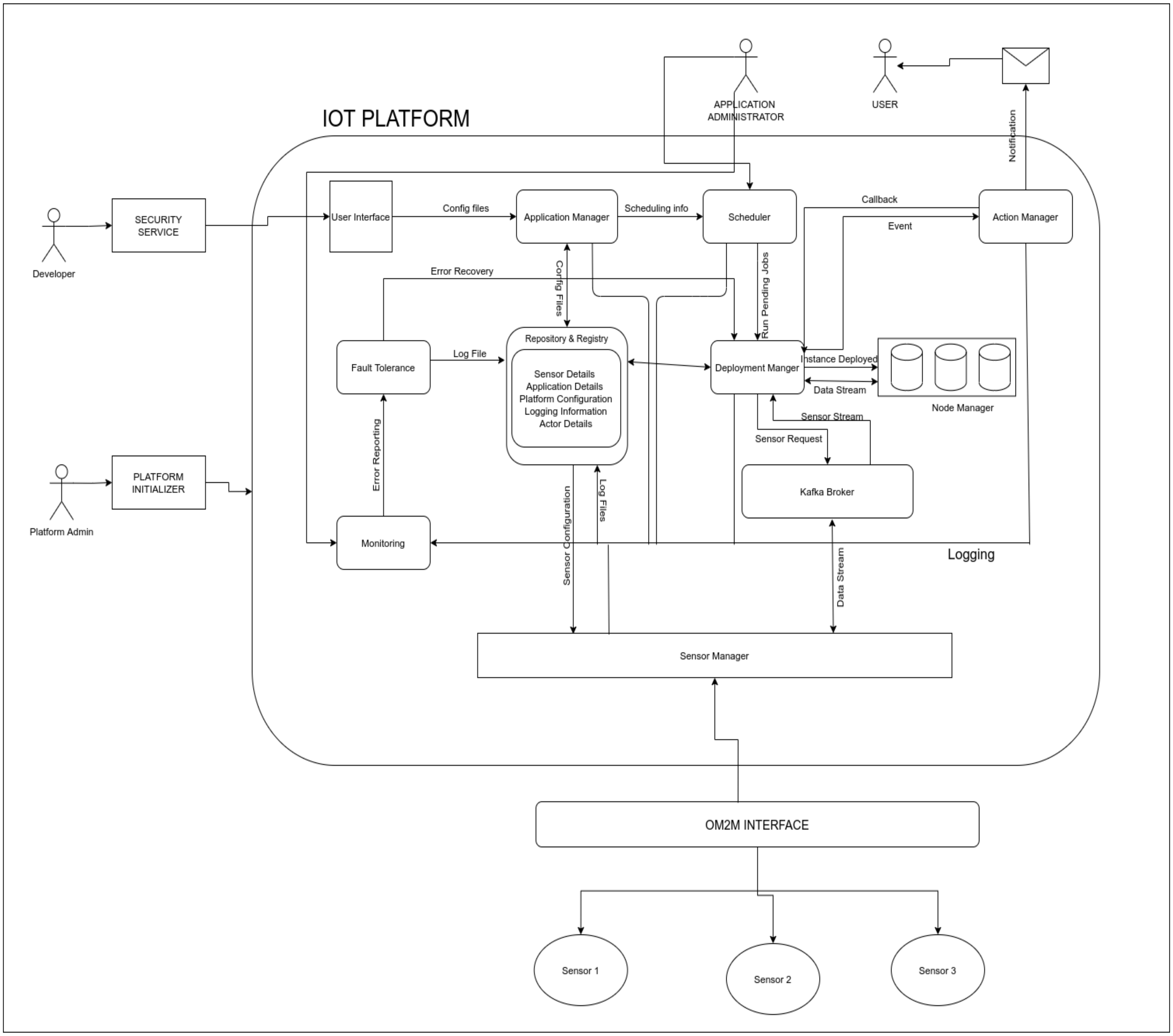
Test Case2: - The Fault tolerance will restore the down component.

Input: Platform Component down.

Output: The component status restored.

4. Solution design considerations

4.1 Design Big picture



4.2 Environments to be used

- 64-bit OS(Linux)
- Minimum RAM requirements: 4GB
- Processor: Intel i3 5th Generation

4.3 Technologies to be used

Amazon AWS
Why: will be used for hosting the platform on clout and making all modules run on different virtual machines.

Docker: Applications will run inside the docker container.
Why: Isolates the environment in which applications will run and allows different applications to run with different libraries and different versions of the libraries and abstracts the underlying OS of our infrastructure.

Flask
Why: Tool provided by it will help to expose Api endpoints for the platform to use and it gives developers a good base framework of request/response management to work with.

Bootstrap
Why: it will be used for UI to build responsive web applications.

Kafka
Why: it will be used for managing communications among various modules

Mongo DB
Why: For use as repository and Database

Python
Why: for deploying and developing platform

Json

Why: for passing json formatted data.

4.4 Overall System flow and Interaction

We will initiate the bootstrap or platform initializer program to initialize our platform. The bootstrap module will make sure that our platform will be up and running.

It will then start the additional external services like Kafka and the database for the repository.

After all the components are running, we will initiate the web interface.

The application manager will start the web interface and accept the request to authenticate it and then will ask for the zipped files.

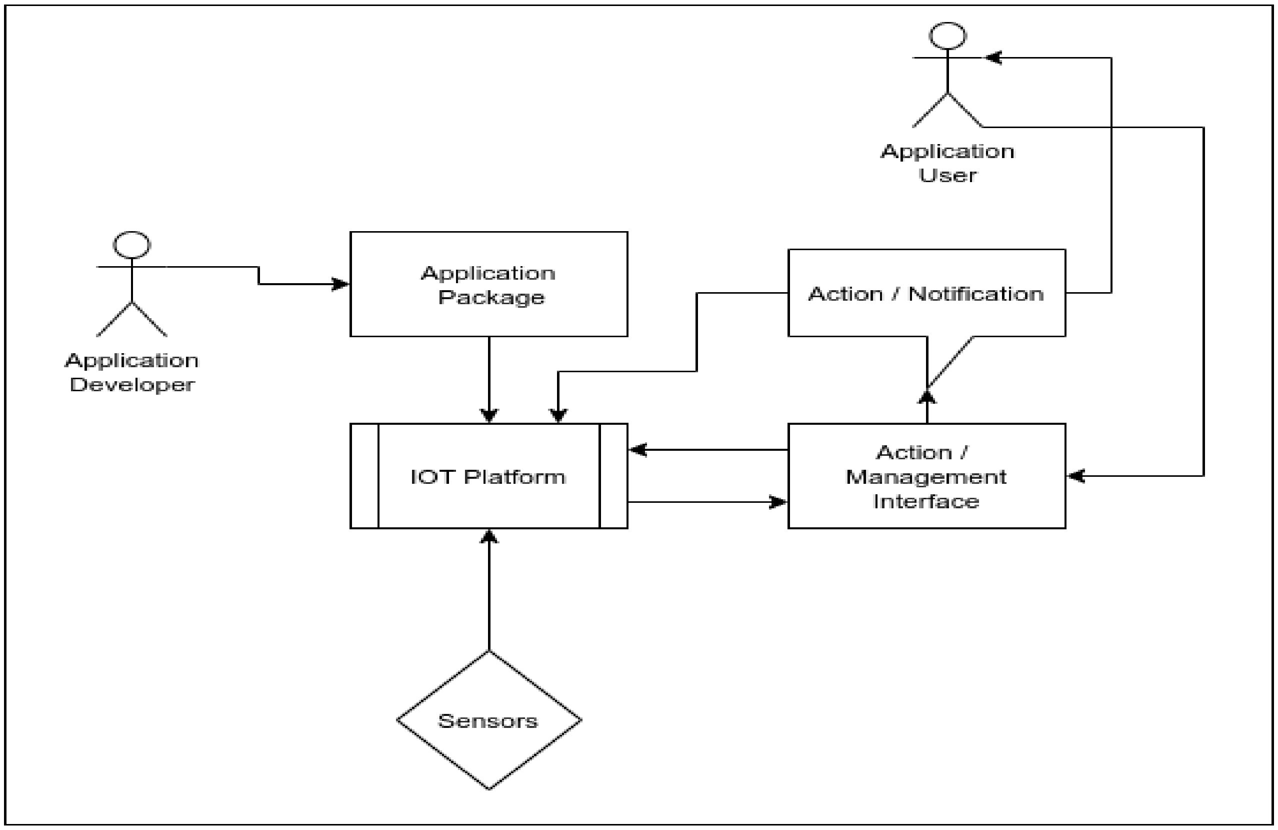
After the zipped files are uploaded the application manager will extract the zipped files and validate them.

The scheduler will read the application configuration file and create a schedule instance. The scheduled instance will have an algoid , start_time , end_time ,job_type and then it will execute accordingly.

Once the scheduler triggers the deployment module at the time a particular algorithm must be run, the deployer will deploy the application, gives some status, and deploys the application.

The algorithm running can also trigger some actions based on certain events and these actions are taken care of by the action Module.

4.5 IOT System Design:



The application developer will develop an algorithm or application which based on user requirements perform some operations on the data provided by the sensors and will send the result back to the users in required response format I.e. either a notification or some action on the sensor itself.

4.6 Server System Design:

The servers where the algorithms are running must be capable of isolating the algorithms from one another and the worker nodes must be easily managed for monitoring and patch updates. To achieve this functionality, we are creating docker containers where the algorithms will be deployed, The deployment manager should be able to manage the worker nodes and manage the infrastructure. The scalability of the servers can be controlled by the platform manager.

4.7 Approach for communication & connectivity:

We are using Kafka to enable communication between different components of the platform. Initially the different platform components will be registered with the Kafka broker and will be sending status/messages/acknowledgements in predefined format.

4.8 Registry & repository:

The Repository will be used to store different files related to different modules and will be separated based on the modules so that each module can access its repository and use them as required.

- **Application Repository** : To store application Configuration and executables.
- **Sensor Repository** : To store sensor configuration and communication details.
- **Credentials Repository** : To store credentials of different actors interacting with the platform.
- **Platform Repository** : To store details and config about the platform.
- **Scheduling Repository :** To store scheduling details of the applications.
- **Logging Registry** : To store logs about various modules and deployed applications for fault tolerance and monitoring purposes.
- **Deployment Repository** : Deployment repository keeps track of which algorithm is running on which node and certain other details like the user who created the job and

some other metadata. This helps in restarting a specific set of algorithms in case when a node goes offline/crashes and helps in fault tolerance. It also provides the status of various algorithms that have been deployed for the application developer.

- **Server Repository :** Server repository will keeps track of the servers and fetches the stats of the servers and other metadata like IP, CPU Utilization.

4.9 Service/Application lifecycle:

Deployment manager will check the health of each of the Node servers and If a container/server goes down, Then the server lifecycle manager will re-execute the application inside a new container and restart a server if it goes down.

4.10 Scheduler:

The scheduler takes the input from the algorithm user when the algorithm should be deployed and some other details like if that job is a recurring job and when to stop it. It creates and sends a request to the deployer when that trigger point has been reached and gets the status that has been given back.

4.11 Load Balancing

Once the deployer receives the algorithm details from the scheduler to deploy. It decides the best node on which the algorithm must be deployed. The load balancing technique can be done either basing on CPU utilization / bandwidth utilization of the worker nodes.

4.12 Interaction with other Components (Internal design overview)

★ **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.

4.13 Wire and file formats

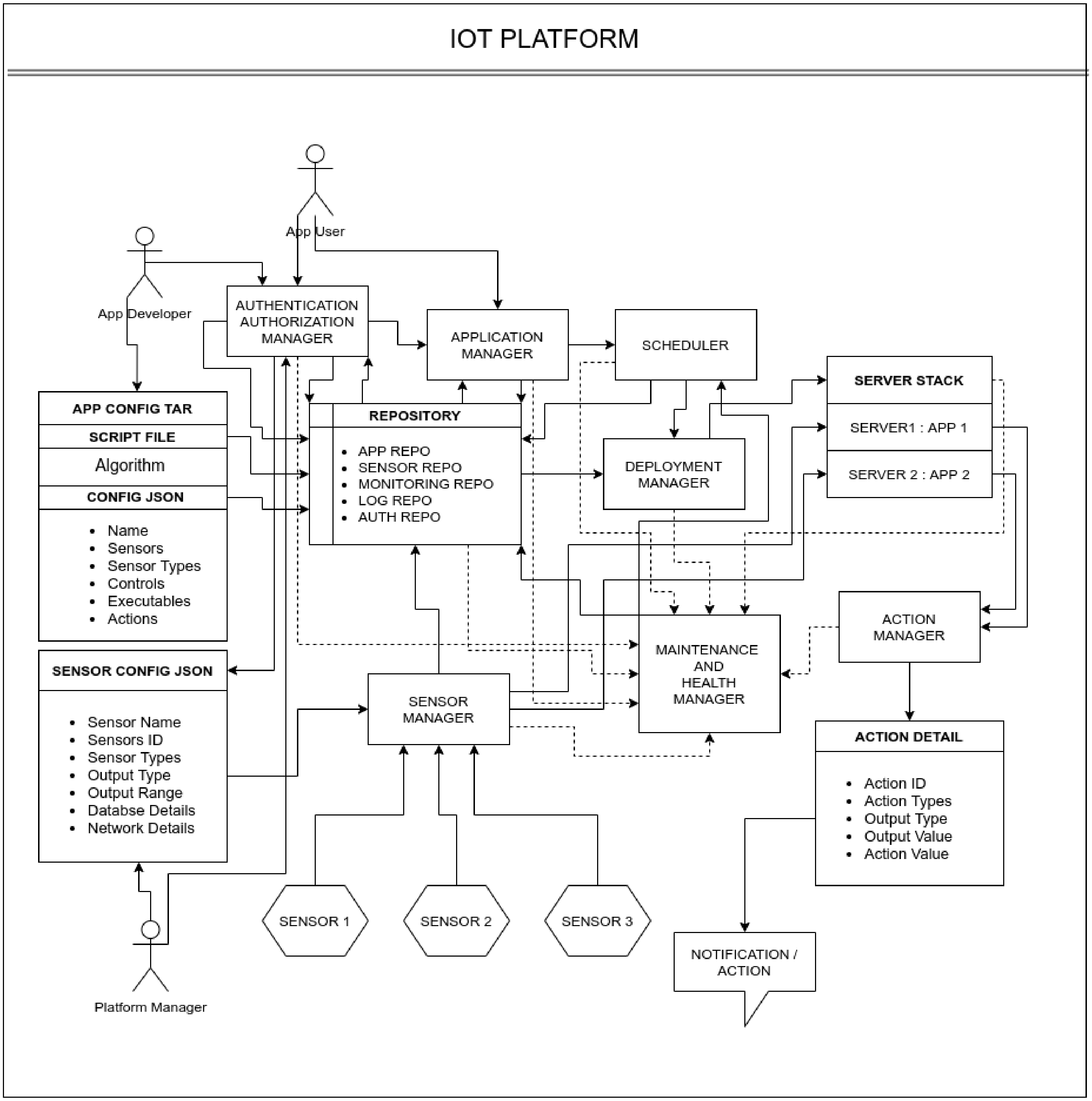
- The main package files are provided in the zipped manner.
- The zipped files consist of the application, application configuration file, sensor configuration file.
- The application file is a python script.
- The application configuration file is json.
- The sensor configuration file is json.

5. Application Model and User's view of system

Key elements

1. **Application Config file** - Contains info about the application used i.e., type of sensor used, name of sensor, controls, executables - all the scripts used in application.
2. **Sensor Config file -** contains details about the sensor used by the platform like sensor name, id, type, output type, output range, network details etc.
3. **Application Platform -** Sends config file to the platform and receives output from the platform.
4. **Sensor Manager -** Uses oneM2M interface to collect data from sensors and store it or preprocess it for algorithms using it.
5. **Sensor Registration -** if the application needs new sensors, then it will request the platform to add it, and the sensor registration module will store all information about the new sensor in the database and register it.

6. **Platform manager -** Manages the flow of the application and when a new application is received, provides it with resources.



Various Users

1. **Application developer** - develops applications, its algorithms and what all applications are going to do.
2. **Platform Admin** - Admin manages the application and monitors it**.**
3. **User -** Users who will be using the application.

Key Steps in whole application development Process -

Develop:

★ Understanding the platform and develop application based on a Config file containing -

```
{
                    Sensor types: [ Temperature Sensor , Camera Sensor, Light Sensor] ,
                    Controller Types: [ Street Light Controller ]
                    Algorithms: {name: id}
}
```

Algorithm Script -

- - contains info about all the sensors to be used, along with their id and location.
- - 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 config.tar file must be provided by application developers which contains two information like application config.xml and scripting files, and now config.json contains algorithm interface template information 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 sensor manager and 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.

6. Key Data structures

Sensor Structure

Sensor Type Registration

- Sensor ID
- Sensor Type
- Sensor Data Type
- Sensor Data Rate

Sensor Instance Registration

- Sensor Type

- Sensor Location
- Sensor Purpose
- Sensor Status

Scheduling Information

- Schedule Type
- Schedule Algo
- Schedule Start Time
- Schedule Stop Time

User Information

- User Role
- User Password
- User e-mail

Algorithm Information

- Algorithm Name
- Algorithm Id
- Algorithm Sensors
- Algorithm Dependencies
- Algorithm Action Type

7. Interactions & Interfaces

APIs

Sensor API – The developer can use the sensor API to interact with the sensors and can request or control the sensors.

Notification API - The developer can use the notification API to set up the notifications like email, push-notifications etc.

Monitoring API - The deployer component will provide an interface/UI to the platform developer to check the health and status of the resources (containers in which applications are running and nodes). The platform developer can access the logs of a particular container and forward it to the application developer if needed. Low Level Design (for each of the modules)

User interactions

The web dashboard provides the user with input files like user-id, password.

The user needs to complete the authentication and then needs to upload the application package.

Module to Module interactions

1. **Action Module: -** After the application/algorithm gets executed on the server, If the output must 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.
2. **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 on a reserve IP and PORT.
3. **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. Deployer will also share the id of the algorithm so that if it decides to kill the job midway it can do so.
4. **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.
5. **Deployment Manager: -** Deployment Manager receives a request for Deployment from the Scheduler. Applications inside docker container will request for data from Sensor Manager. Deployment manager will send its status to monitoring module.

8. Persistence

- The system manages fault tolerance for redeploying and managing faulty modules
- Continuous logging and health monitoring is done and can be reviewed by admins
- Whenever a deployed application crashes it can be redeployed

- Crashed server nodes can also be redeployed along with algorithms deployed on that Node

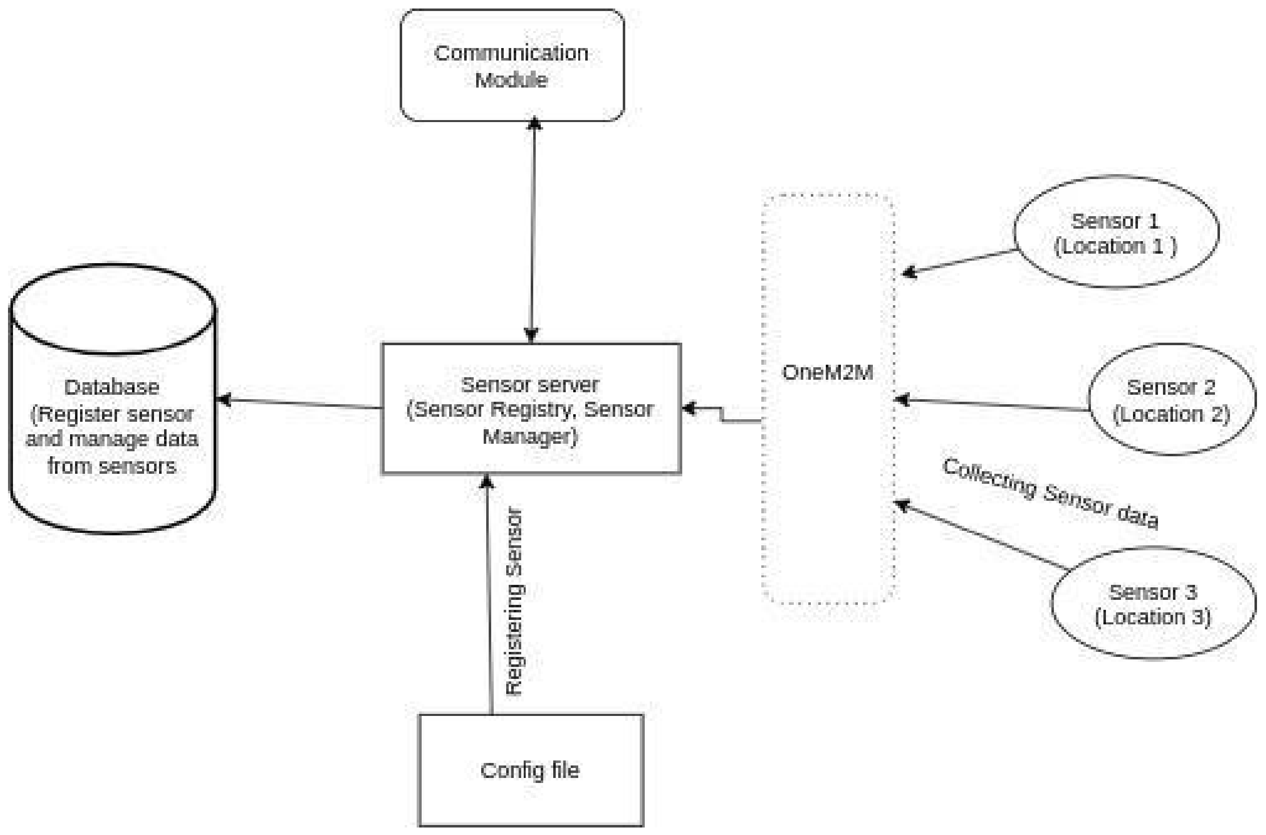
9. Low Level Design (for each of the modules):

- **Communication module, Sensor Manager and Monitoring:**

  Lifecycle of the module:

  In case of Sensor manager, always data from sensor is passed to sensor manager and upon calling of data, the sensor module will pass on data to schedule the algorithm corresponding to that sensor data.

  While in case of Monitoring: This will always be on and keep record of all the components and stop when our platform is down.



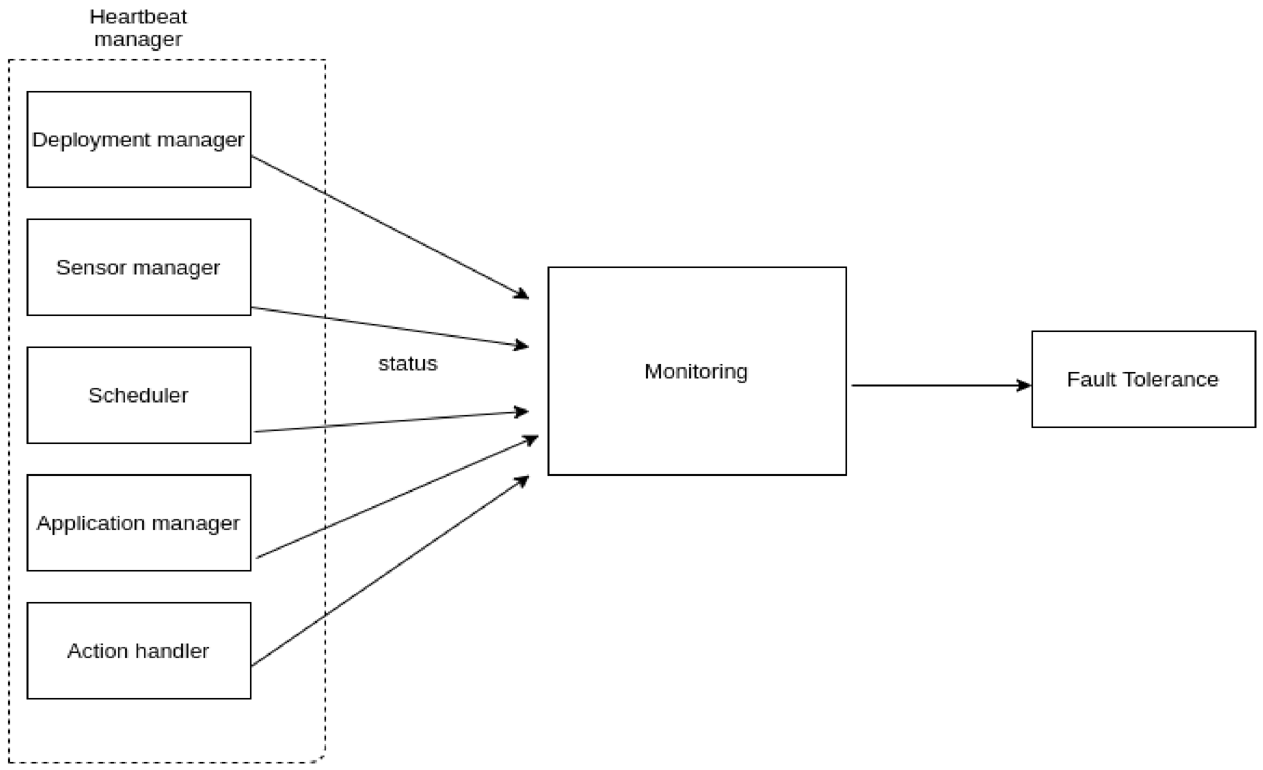
The **submodules** present are -

★ **Sensor Registration** - takes config file as input from user containing details about sensor and then makes its entry in the database. And then connect to that sensor to collect its data.
★ **Sensor Server** - Sensor server has all the information about sensors present and relates to 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.

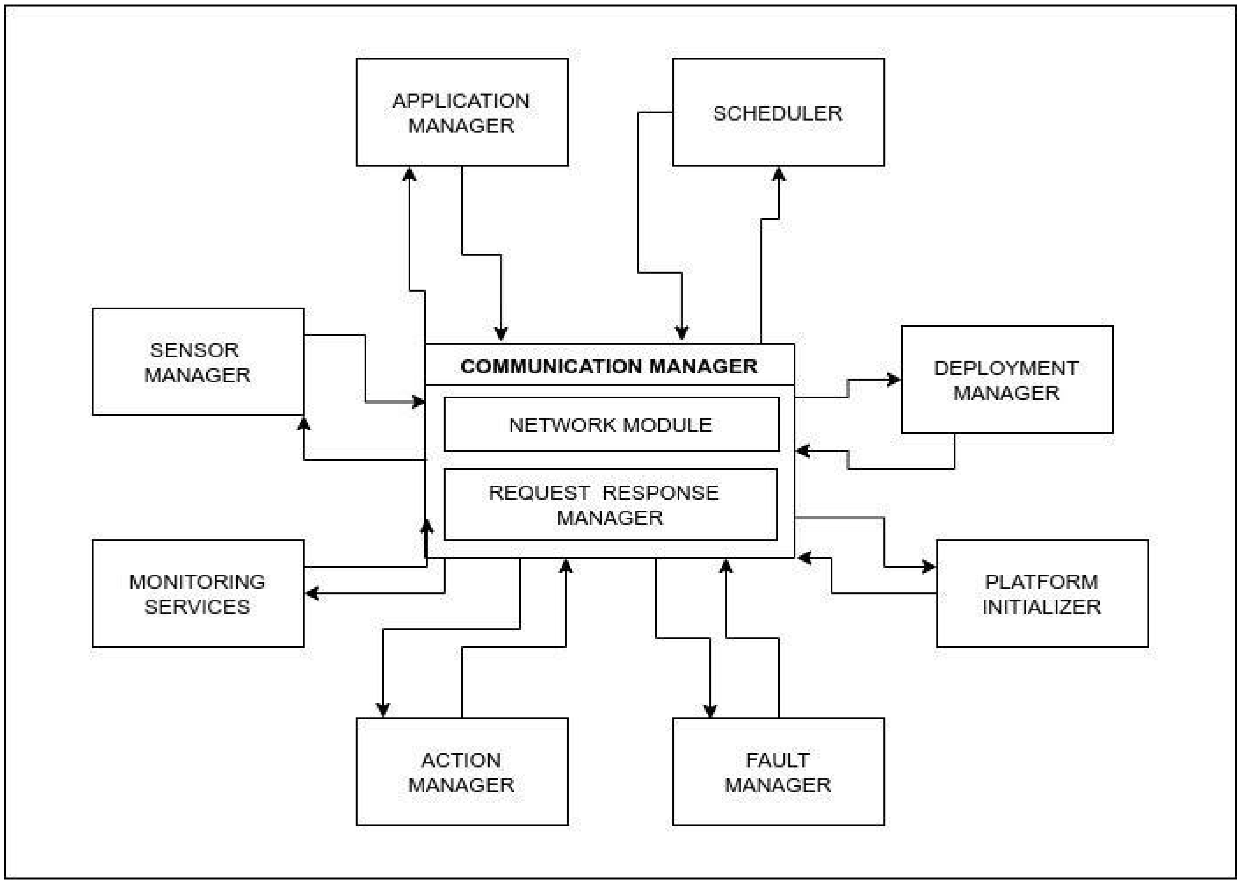
Monitoring -



★ **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.

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.



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.

Interactions with other Modules are:

All the modules will be integrated with this module to be able to communicate among each other 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.

Application Manager and Scheduler

Lifecycle of the module –

Application Manager will start at the time of platform initialization and will continue to run as it is providing the user interface and will be stopped when the platform stops. The user will upload a zip file containing the application code/algorithms and config files using the user interface which will then be validated and saved on in the application repository.

Scheduler will start when the platform starts and will be monitored by the Monitoring service of the platform. It will keep track of all the schedules of the applications on the platform and will execute the algorithms on deployment manager.

Application Manager –

The submodules are –

- ★ Authentication and Authorization Module
- ★ User Interface
- ★ Validator
- ★ Algorithm Manager

Authentication and Authorization Module –

- **Functionality** – This module will verify the users and requests made by user. It will check if the user has appropriate access rights for accessing a resource or making a request.
- **Interactions** – This module will connect with the database to check if the user is registered and will verify the role.
- **APIs** – It will expose an API to receive the username and password and return the verification status.

User Interface –

- **Functionality** – This module will provide an interface for the users to interact with the platform. This will also provide a dashboard to monitor the applications.
- **Interactions** – This module will interact with other services of the platform to receive information to be displayed on dashboard.

Validator –

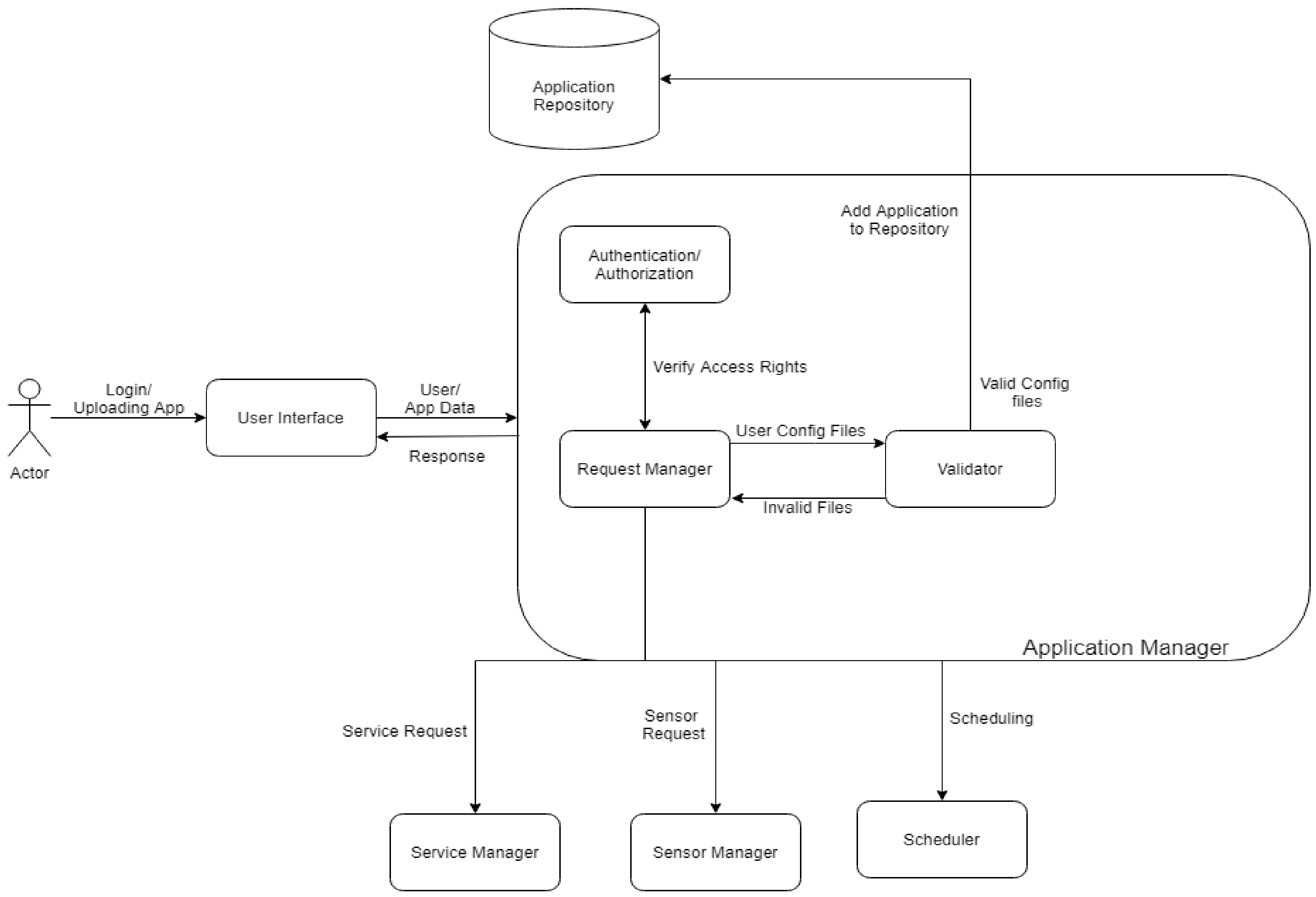
- **Functionality** – This module will verify the files of application uploaded on the platform.
- **API** – An API will be created for validator to which a config file can be passed to check if it is in valid format.

Algorithm Manager –

- **Functionality** – This submodule will manage the algorithms of the applications deployed on the platform.
- **Interactions** – The module will interact with Scheduler to provide the algorithm.

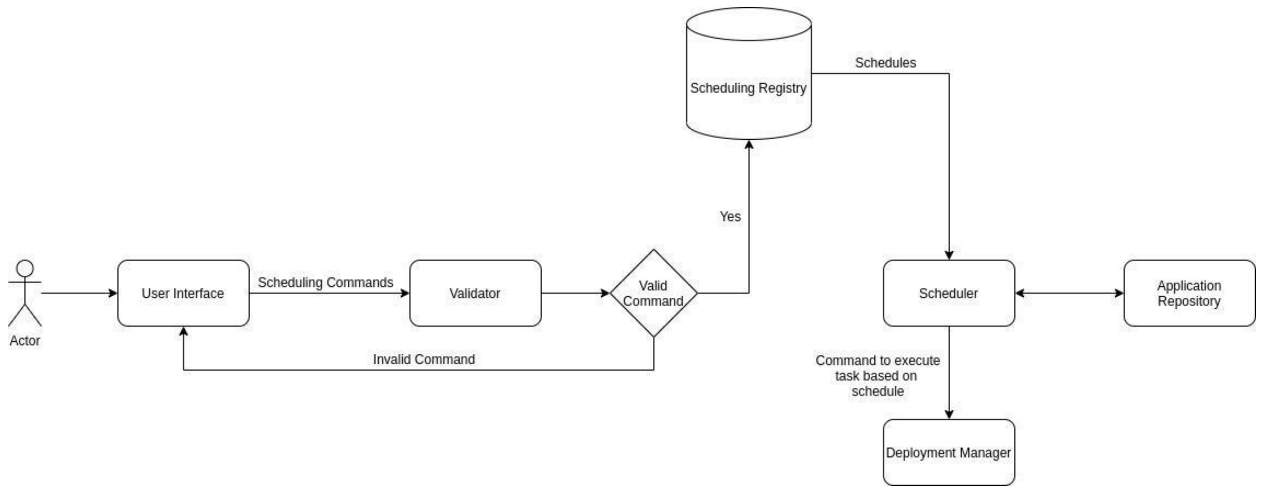
Interaction between submodules –

- User Interface and Authentication and Authorization Module will combine to provide the login functionality to user. The username and password from user will be received by the user interface and will be verified by authentication and authorization module.
- The application uploaded on user interface will be sent to validator to check for the file formats of config files.
- After successful verification, the algorithms of application will be handed over to Algorithm Manager.



Scheduler

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



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.

Deployment Manager

Service Lifecycle Manager

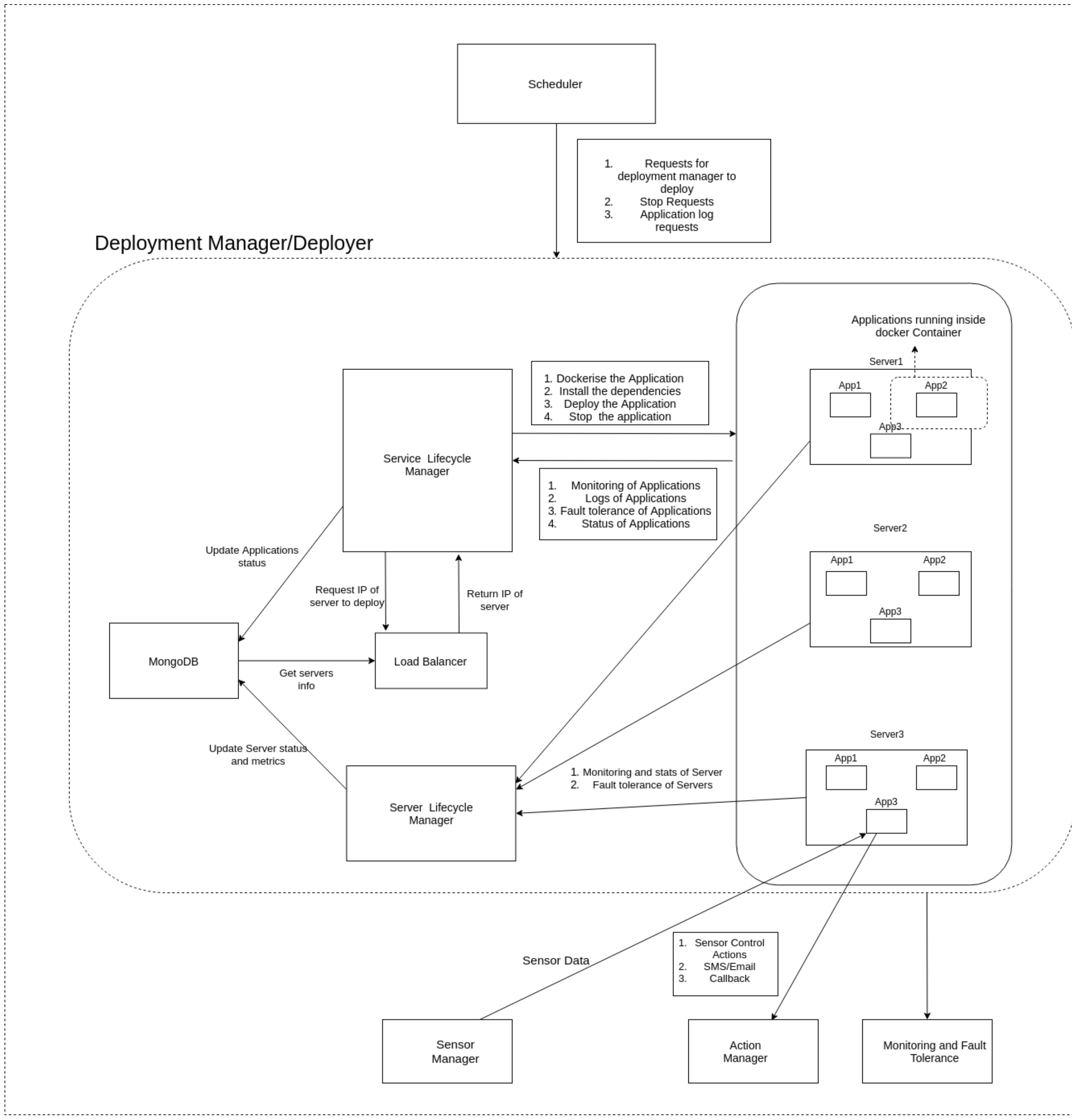
The scheduler will send a request to execute an application/algorithm to the ServiceLCM. The request will contain the details of the dependencies, the actual executable. ServiceLCM will request the ip and port of the server to deploy from the Load Balancer.. Once it receives the ip of the server from 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.

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.

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.


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
1. Dockerise the Application
2. Install the dependencies
3. Deploy the Application
4. Stop the application
Service Lifecycle Manager
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

Deployment Manager

Services Provided

- Dockerizing The Application, installing its dependencies
- Load Balancing
- Deployment of the Application
- Monitoring the Applications
- Logs of the Applications
- Fault Tolerance of the Applications
- Monitoring of the Servers, fetching the metadata and stats of the servers
- Fault tolerance of the servers

Interactions with Other Components

Action Module :- Some the algorithms may execute some actions/control actions which are then executed by the action module.

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. Deployer will also share the id of the algorithm so that if it decides to kill the job midway it can do so.

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.