# IoT Platform For Monitoring End Devices and Controlling Them Through Applications

Group 1

May 14, 2021

Abhishek Gorisaria, Divyansh Shrivastava , Apurva Jadhav, Arushi Agarwal, Aman Nautiyal, Shubhankar Saha, Souptik Mondal, Neerja Gangwar, Jyoti Gambhir, Shreya Vanga, Apurvi Mansinghka, Ramanjaneyulu Payala

**Mentor and Professor :** Prof. Ramesh Loganathan

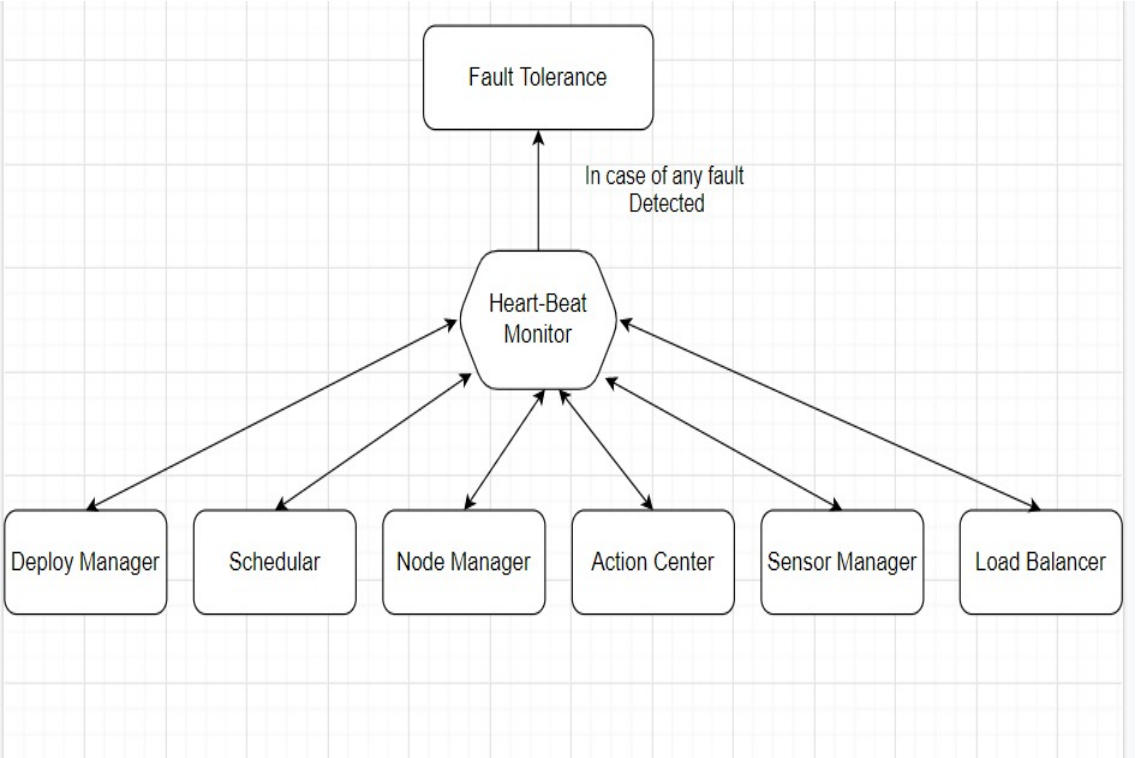
**TA:** Pratik Tiwari, Shubham Agarwal, Jay Krishna

# Contents

# 1 Intro to the Project(Function Overview of Team's Module)

## 1.1 Team1

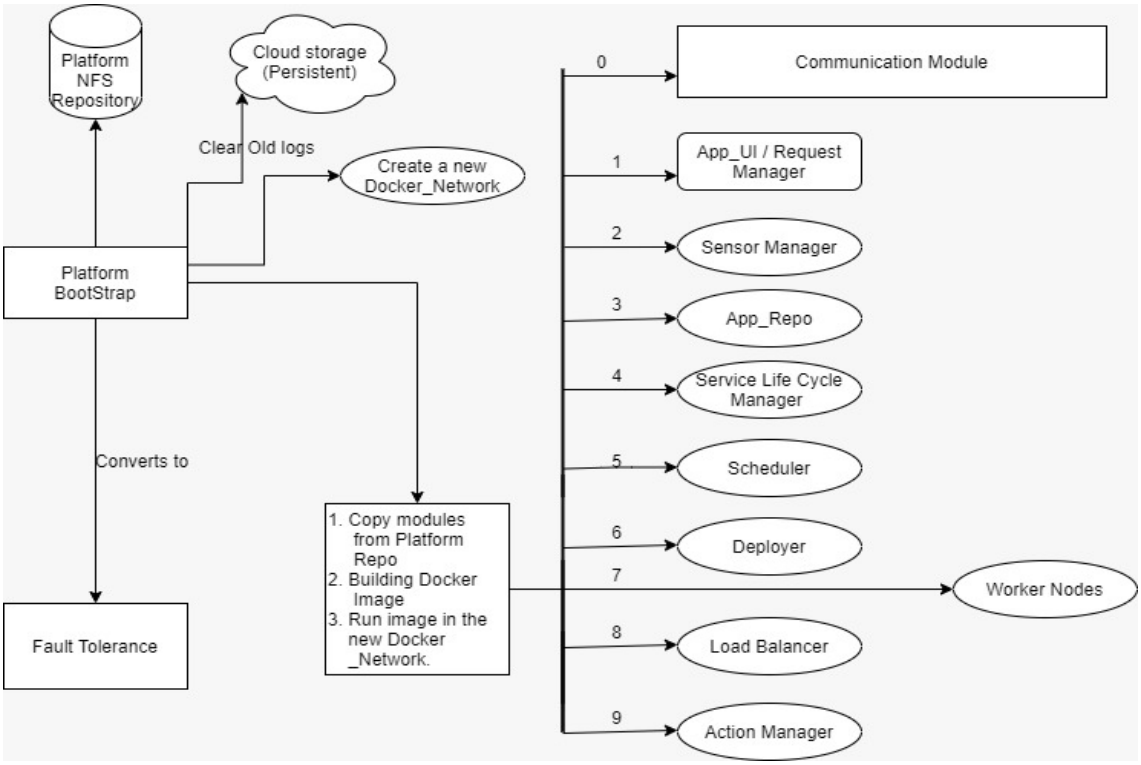
- **Action Manager** : The action manager will receive information from the host machines and performs all the necessary actions.It communicates with the sensor manager using a unique kafka_topic for controlling. It notifies via SMS ,email to a list of users as mentioned in the input j specific controller/sensor.
- **Monitoring** : This module will continuously monitor all the critical components so that the platform can detect if there is any failure of service or node. It can be implemented by using heartbeat signal method. In this method we will continuously check all the components by sending heartbeat signal and analysing the responses form those components.




- **Fault Tolerance** : Suppose there is a situation where any particular node is down for any reason. Then the fault tolerant manager should be able to handle the situation by transferring all the running jobs to another node. Another case if any particular service,

running in a node is down then that service should be restarted in that node itself rather than allocating another node for that service.

- **Platform Initializer** : This module will be responsible for initializing various other modules (like Scheduler, Deployment Manager, Load Balancer, etc) and repositories. It creates a docker network and inside that all the modules can communicate each other by using the module name only. Docker network resolves the module name with the ip addresses. The port for the corresponding modules are exposed in the network. This module gets the platform up and running. Platform initializer performs this task by reading a configuration file at the start. There is a ip.json file that contains all the module names and corresponding ip addresses. This config file has all the relevant details for the modules.




## 1.2 Team2

The modules that are under our Team are :

- **Platform and Application UI** : Platform and Application UI provides a GUI for the

Application developers as well as Application users to interact with our platform. It is more of a luxury than a necessity but it makes the overall UX much smoother.

- **Scheduler** : This module will take input from the application developer or the end user to start and stop the application services that are uploaded on our platform at some scheduled time and date. The output to this module will basically be a signal that will be generated whenever it's than address of the host machine where the service needs to be deployed.
- **Load Balancer and Service Node Manager** : This module is responsible for balancing the load on all of the host machines where the services will be executing.Based on resource usage, the load balancer provides the deployer with the node where the service needs to be deployed. The service node manager keeps track of service nodes that are up and calls fault tolerance module for nodes which go down.

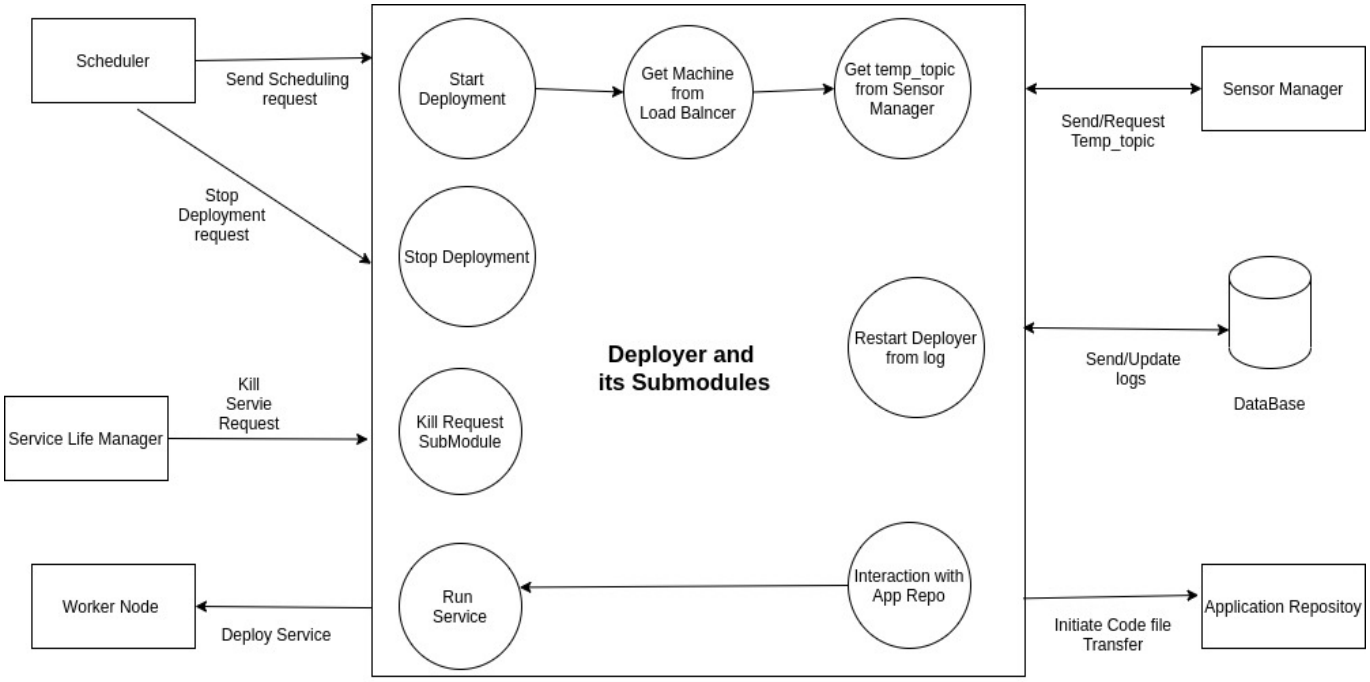
### 1.3 Team3

- **Sensor Manager repo and registry** : 1. This module is responsible for binding correct sensor data stream with corresponding algorithm. 2. Sensor Manager can add or remove sensors. 3. Will identify IoT devices from the sensor database based upon the param- eters.

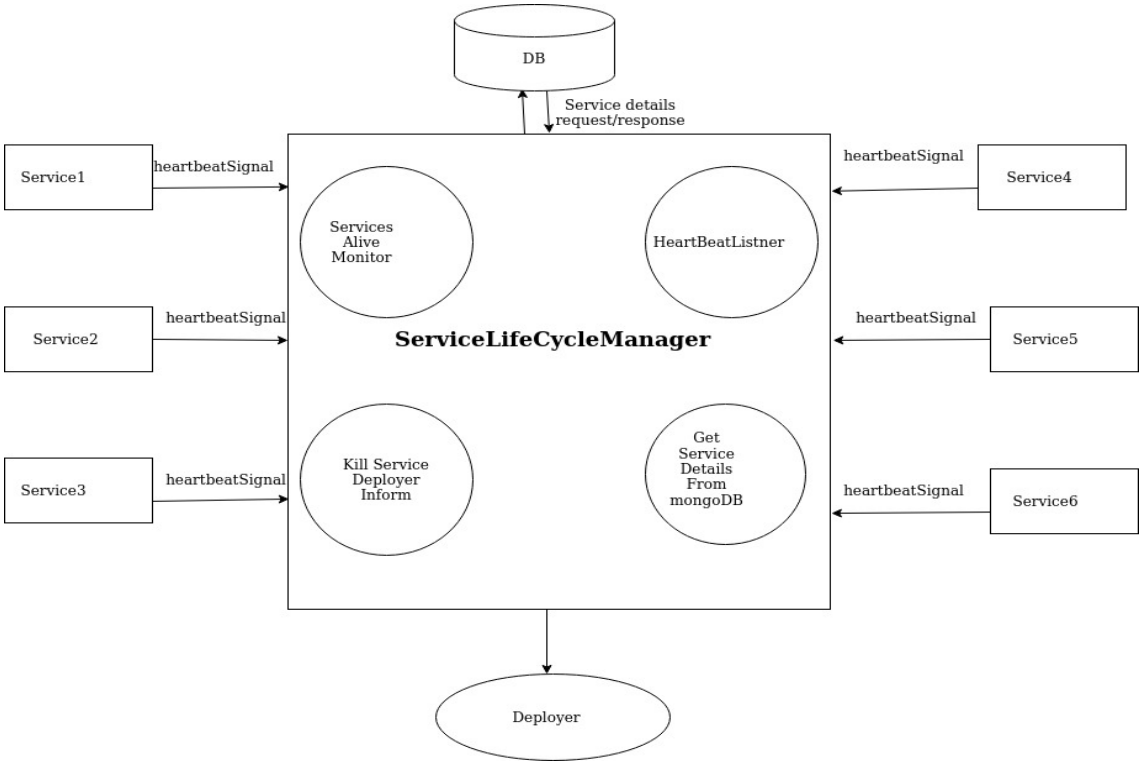
### 1.4 Team4

Modules are as follows:

- **Deployer**
  Deployer is responsible for running the scheduled application service on one of the up and running worker nodes. It interacts with multiple modules such as scheduler, load balancer, Application Repository, service life manager and worker node to setup the enter environment and deploying the service. It also notify the scheduler to reschedule a service in case of service failure.




- **Service Life Manager**
  Service Life Manger is monitoring the working of each of the running application services. It notifies the deployer about the failed services which further interacts with other modules to take necessary action. Heartbeat technique is used at the service life manager to handle fault tolerance of services.




- **Application Repository**
  All the code and configuration files will be uploaded by the application developer on and stored in the application repository. Deployer will fetch the application config file for the application at the time of deploying from the repository.On getting schedule request deployer will instruct this module to send the program/code files to a specific worker node.

## 2 Use Cases/Requirement

- At first the platform initializer is started.
- It will get all the information of the important modules from the master config files once it is up and running. Using the information from the master config file it will start all those important modules.
- User(developer) will login using dashboard or GUI to execute or upload his/her algorithm.

- After successful authentication the user(developer) will be able to upload a zip file which will contain :
    1. Code files(algorithms)
    2. Config file
    3. Scheduling and other service dependency information
    4. Sensor details
- After the file gets uploaded it will be validated to check if the format is correct or not.
- After successful validation it is stored in the application repository.
- If a start/stop request is for a particular service then the request is sent to the scheduler.
- The scheduler will request a host machine from the pool. The scheduler will store the relevant information of that service in a runtime registry(runtime app config file).
- The scheduler will then request the deployer(deployment manager) to run the service at that particular host machine.
- The deployer will bind the requested program/service with its dependencies and the sensor data and will run the wrapped service in the host server.
- Once the algorithm/service is executed on the host machine/VM its result will be sent to the action center.
- The action center will communicate with the various output sensors/controllers, users(who requested the service) and specific users depending on the output from the algorithm.
- Throughout the entire time when all these modules are interacting and working with each other, the heartbeat manager will monitor continuously to check if every module is functioning or not.
- If the heartbeat manager finds that if a particular node is down or the algorithm running in the host server has stopped then the heartbeat manager will make sure that the algorithm is either restarted in the same node or run on a different node.

# 3 Test Cases

## 3.1 Team Module Test Cases

### 3.1.1 Team1

1. **Fault Tolerance**:
   - **I/P** : Forcefully shut down a host machine.
   - **O/P** : The algorithm should start running in a new machine and rest all the machines should remain unaffected.
2. **Monitoring**:
   - **I/P** : Check log before shutting down a module.
   - **O/P** : The change in log will indicate that that particular module is not working.Also, the fault tolerance module will be informed about it, to take necessary actions.
3. **Platform Initializer**:
   - **I/P**: config file to the platform that contains configuration details to initialize different modules of the platform.
   - **O/P**: Different modules will be set to initial parameters. All the modules will be up and running after this.

### 3.1.2 Team2

Following tentative test-cases are decided as of now, further changes may be required.

1. **Platform and Application UI**:
   - **I/P**: Application Developer/User inputs.
   - **O/P**: A request to the scheduler or to the application repository for application upload.
2. **Scheduler**:
   - **I/P**: Request from the user through UI for scheduling an algorithm/service.

- **O/P**: Start and stop signal for the algorithms given to the deployer.

3. **Load Balancer**:
   - **I/P**: Input to the load-balancer will be the service which needs to be deployed.
   - **O/P**: Output will be the service that will be deployed to multiple host machines. Check can done by randomly querying whether a particular machine equal percentage of load compared to other host machines.

#### 3.1.3 Team3

**Sensor manager and repo**:

1. **Test registration of new sensor catalogue to existing**
   - **I/P**: New sensor catalogue json file / user input through form
   - **O/P**: A different type of sensor type created and available for instance creation
2. **Test registration of new sensor instance**
   - **I/P**: New sensor instance json file / user input through form
   - **O/P**: A new sensor instance is created and available for use for selection
3. **Test Sensor selection from manual by the application dev**
   - **I/P**: submit sensors from given manual
   - **O/P**: sensor data is binded with respective service
4. **Test Sensor data that is incoming and the interfaces provided by the platform to access it**
   - **I/P**: The raw sensor data that is incoming through the sensors
   - **O/P**: the data can be accessed using platform interfaces

#### 3.1.4 Team4

1. **Deployer**:
   - **I/P**: Input to this module will be the request from the scheduler to deploy to the service.

- **O/P**: Deployer interacts with multiple modules such as scheduler, load balancer,Application Repository, service life manager and worker node to setup the enter environmentand deploying the service. I

2. **Service Life Manager**:
   - **I/P**: Recieves heartbeat message from all the running application service in every 2 seconds.
   - **O/P**: Monitors status of each running service, in case of failure notify the deployer along with service id.
3. **Application Repository**:
   - **I/P**: Request from deployer to send config files to it and another kind of request to transfer the program files to worker node.
   - **O/P**: App Repo gets the zip folder from database, unzip it, ssh into the worker node and recursively copies all the code files of service and helper files of platform.

### 3.2 Overall project test cases

**Test case I :** Test addition of new user(app dev as well as the end user) profile to the platform

IP : Details for new user to login in terms of form

OP : These details must be persisted in the profiles database under respective actor and should be accessible by the platform init code.

**Test case II :** Test addition of new application by a user to the platform

IP : The application config file along with its code in the required format of directory structure.

OP : The information regarding the applications like types of services available, numbers of services in that application etc must be able to be used.

**Test case III :** Test registration of new sensor catalogue to existing

IP : New sensor catalogue json file / user input through form

OP : A different type of sensor type created and available for instance creation

**Test case IV :** Test registration of new sensor instance

IP : New sensor instance json file / user input through form

OP : A new sensor instance is created and available for use for selection

**Test case V :** Test Scheduling of a service in an application

IP : Developer submits scheduling data using dashboard

OP : The application service is scheduled and runs at the scheduled time

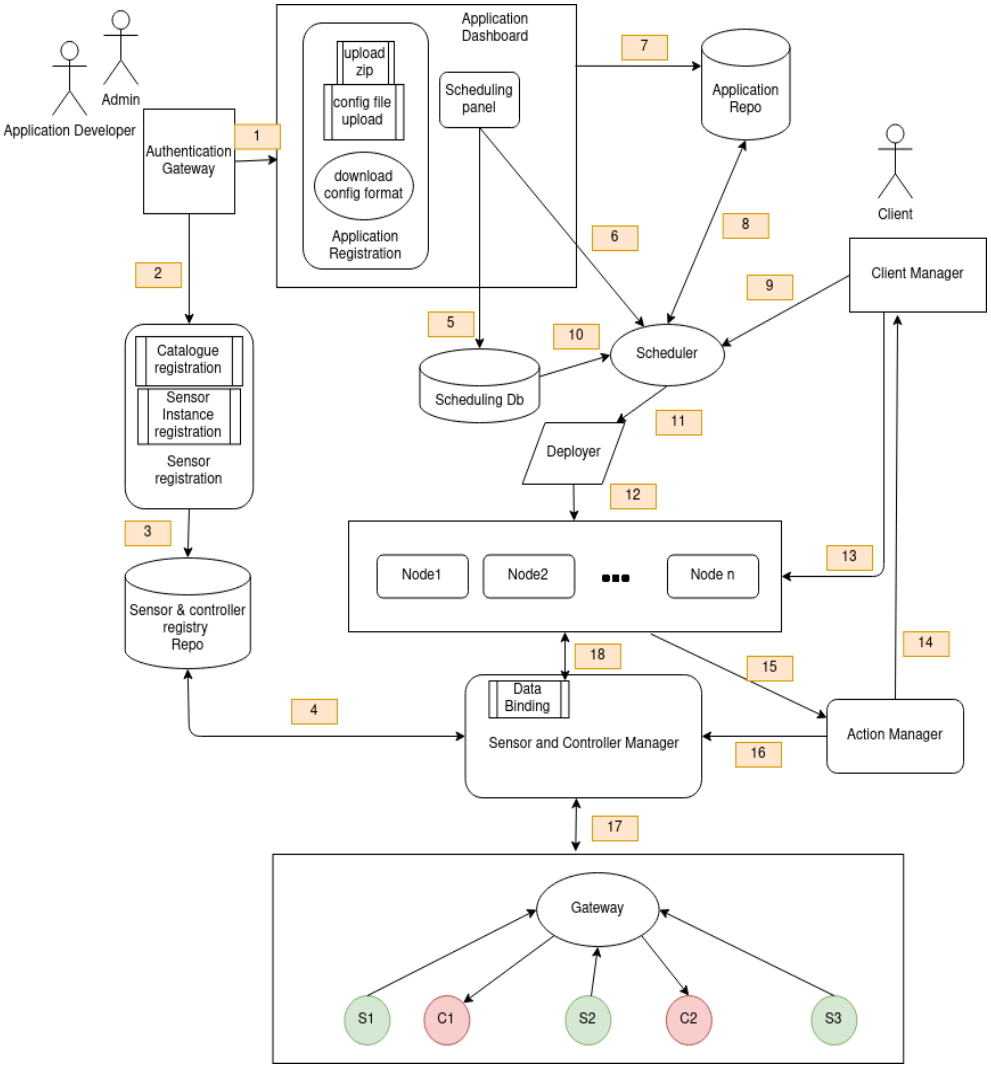
**Test case VI :** Test Sensor selection from manual by the application dev

IP : submit sensors from given manual

OP : sensor data is binded with respective service

# 4 Solution Design Considerations

## 4.1 Design Big Picture




1. After Authentication, developer accesses the dashboard to register his application through zip file and also provides information about the scheduling of these application.
2. Again, after authentication developer can register the sensor, download catalogue(the format in which he has to upload his/her sensor information, however, sensor type and

data type of the sensor will be pre-defined in the platform)

3. Sensor after registration its information will be stored in repository(can be any kind of file-system), also, this module will provide access to controller manager.
4. Sensor controller provides access to controller manager through which controller can be accessed.
5. Dashboard will also provide interface to developer to provide scheduling information of the application which will be stored in scheduling DB.
6. Correspondingly along with scheduling db, the information will be provided in the scheduler.
7. Application source code zip file and the configuration files are stored on the Application repository.
8. Scheduler reads the application's source code and configuration file stored in the Application repository to pass it along for deployment.
9. If the application which the user wants to access it not deployed on any of the hosts, Client Manager will invoke the scheduler to immediately schedule a deployment into one of the hosts.
10. In case of some failure or a platform reboot, the scheduler will read all the scheduling information from the Scheduling database.
11. Scheduler sends the source code and the configuration file of the particular application to the deployer for it's deployment into the host machines.
12. After setting up the Deployment environment on the host machine(s) allocated by the Load balancer, the application's instance will be executed on those host machine(s).
13. Client Accessing the code deployed on hosts.
14. Action manager sending response or notification depending on the type of service expected and mentioned in the configuration file.
15. Hosts sending the response / output to the action manager to perform necessary action.
16. Action manager sending response to control the controller.
17. Sensors sending data to the sensor manager / Controller manager sends signal to manipulate the controller.
18. Sensor manager fetching sensor data (real time as well as stored) and input them to the code running on the nodes.)

### 4.2 Technologies/Environment to be used

- **OS:** Linux
- **Kafka**: Kafka is used for real-time streams of data, to collect big data, or to do real time analysis (or both). We have used it as our primary communication module .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 analytics. Since Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system.
- **MongoDB**: 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).
- **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.
- **Flask**: Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.
- **Docker**: Docker is a software that allows OS-level virtualization and allows deployment of docker containers which are isolated environments containing some program source code and their configurations.

### 4.3 System Flow and Interaction

After the application developer provides the application package to the platform, it is stored in an application repository. The workflow of the application deployment is described below:

1. The end user of the application sends some request.

2. The request is parsed to determine the application which needs to be deployed and when.
3. The scheduler determines when the application will be deployed.
4. At the time of deployment, the deployer fetches the relevant data from the application repository (application code, config, etc.)
5. The load balancer determines the host machine where the application will be deployed.
6. The deployer sets up the environment and takes care of dependencies as provided by the config file, and finally launches the application.
7. The application can then pass the control to the Action centre to perform operations such as send an action to the IoT device, send response to the user, or send a notification to some party.

## 4.4 Approach for communication and connectivity

1. We will use Flask API for communication between the different the modules.
2. The flow of information will be in the form of JSON request-response.
3. We will use messaging-queue as the communication medium.

## 4.5 Registry & Repository

### 4.5.1 Registry

- Sensor Registration
- Application Developer Registration
- Application Upload

### 4.5.2 Repository

- **Application Repository**:
- **Sensor Repository**:
- **Scheduling Database**:
- **Platform Repository**:

### 4.6 Scheduler

Scheduler is component of IoT platform which is responsible for scheduling services which is provided by application developer. The way it works is described in below points:

1. Application developer while uploading application to the platform specifies scheduling information in given format(in our case JSON).
2. The platform has repository which stores this information.
3. There is another module which polls this information from the repository and queries whether any service is there to be scheduled, if any, it will execute that service.

### 4.7 Load Balancer and Service Node Manager

The load balancer will take care of limiting resource usage on service nodes by distributing the load uniformly across them. The load balancer provides the deployer with the node on which the service needs to be deployed. The load balancing is done based on the resource usage statistics at runtime. It is ensures that the resource usage does not go beyond a limit in any service node as well.The service node manager is another function which monitors the service nodes and calls fault tolerance module for the nodes that go down.

## 5 Application Model and User's view of system

1. After Authentication, developer accesses the dashboard to register his application through zip file and also provides information about the scheduling of these application.
2. Zip folder is validated for config file and src folder(which contains code files for the application) and is stored in the Application repository..
3. After validation developer can register the sensor, download catalogue(the format in which he has to upload his/her sensor information, however, sensor type and data type of the sensor will be pre-defined in the platform)
4. Sensor after registration, its information will be stored in repository(can be any kind of file-system), also, this module will provide access to controller manager.Sensor controller provides access to controller manager through which controller can be accessed.

5. Dashboard will also provide interface to developer to provide scheduling information of the application which will be stored in scheduling DB.
6. Correspondingly along with scheduling db, the information will be provided in the scheduler.
7. Scheduler reads the application's source code and config file stored in the Application repository to pass it along for deployment.
8. If the application which the user wants to access it not deployed on any of the hosts, Client Manager will invoke the scheduler to immediately schedule a deployment into one of the hosts.
9. In case of some failure or a platform reboot, the scheduler will read all the scheduling information from the Scheduling database.
10. Scheduler sends the source code and the config file of the particular application to the deployer for it's deployment into the host machines.
11. After setting up the Deployment environment on the host machine(s) allocated by the Load Balancer, the application's instance will be executed on those host machine(s).
12. Client access the application deployed on hosts.
13. Hosts sends the response / output to the action manager which further sends the response to controller to perform the necessary action.
14. Action manager will send response or notification depending on the type of service expected and mentioned in the config file.
15. Sensors sends data to the sensor manager / Controller manager which in turn sends signal to manipulate the controller.
16. Sensor manager fetches sensor data (real time as well as stored) and inputs them to the code running on the nodes.

# 6 Key Data Structure

1. Queue - On load balancer, there will be multiple requests at a time. A queue is used to allocate nodes to the processes on FIFO basis. Requests are popped and allocated.
2. Map - At the application repository, code files and config files are to be stored corresponding to each application Id. Similarly, for storing sensor information corresponding to each sensor id, map can be used.

3. Heap - On scheduler, jobs to be scheduled will be stored in a minheap based on start time. After certain period of time, heap is accessed and the top most process is scheduled. If the current time is greater than or equal to start time of that job then request is made to deploy. Similarly, all the other jobs are scheduled as per their start time.

# 7 Interactions and Interfaces

## 7.1 Interface between Application Developer and Platform

Application Developer will be interacting through dashboard for uploading code files and scheduling the application. A zip file will be uploaded which contains the config file and src folder(scripts required to run the application).

## 7.2 Interaction between the modules

The platform initializer will deploy the codes of all other components like Deployment Manager, Scheduler, Sensor manager on various nodes. This will get our platform up and running.

The service request received by the UI is sent to the scheduler which then passes on the requests to the deployer at appropriate times.

Deployer sends request to the sensor manager when it gets requested by scheduler to schedule an instance of a service and then sensor manager serves the request by sending data of the required sensors (in a service) on a freshly created new topic.

The deployer queries the load balancer on the node on which the service needs to be deployed. The load balancer queries the nodes for resource usage and chooses a node for the service to be deployed.

The working service intakes sensor data being dumped on the new temporay topic created, and sends any output/ feedback/ notification information to the action manager.

The responsibility of the action manager is to notify users on their phones/ email and also send controller information.

### 7.3 Interface between End user and Platform

End user will be accessing the application running on platform through dashboard. User will be uploading the config file which will have the relevant details.

## 8 Persistence

### 8.1 Sensor Repository

The sensor repository is responsible for storing data related to all the sensors that have been registered in the platform for a specific application. The Repository will contain application specific sensors registered, the configuration information stored in Json format files. We will use MongoDb to store these files.

### 8.2 Application Repository

- Application source code zip file and the configuration files are stored on the Application repository.
- Application Manager checks structure of config JSON file(Application config), if correct structure, store it in Application Repository.
- Application Repository will be used by the Deployer at the time of Deployment of Applications, gets the relevant data (application code, config, etc.)

### 8.3 Scheduling Repository

The scheduling repository will store information related to scheduling of the services. The information stored will include service to be scheduled, time of scheduling, frequency and the config file associated with the scheduling request. Again the information can be stored using MongoDB.

### 8.4 Platform Repository

- It contains all necessary code files and configuration files for initializing/running the Platform.
- Platform Repository will also contain the files and information of major components like Schedular, Load Balancer, Sensor Manager, Deployment Manager, Node Manager and Action Centre.
- For ensuring fault tolerance of those major components a heatbeat signals/logs will be stored in the Platform Repository for continuous analysis and reporting accordingly to the platform.

### 8.5 Config-Info

- Platform Config-files are stored in Platform Repository and used for initialization of the platform by the Platform Developer.
- Sensor Registration Config-file is stored in Sensor Repository. Used to register a sensor/instance accordingly.
- Application Config-file is stored in the Application Repository. Used to schedule/run the application. It contains all the details about that application requirements (Dependencies, Environment, Sensors)

### 8.6 Transient State

The transient state refers to the state that the platform is in at a given time. We will have log files for the various actions taken by the modules of the platform. For example, we can have deployment logs, action center logs, etc. We will also maintain the list of currently running services, their request locations, sensor streams attached to them,etc. The platform developer can then determine the state of the platform using these.