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

Abstract

This document defines the requirements of the building an IoT based platform for managing and controlling IoT devices through Applications. Platform will be built in such a way that every kind of Application can be deployed and maintained through desktop/web/mobile based applications.

Contents

1 Overview

1.1 Introduction (Definition of what constitutes in this project)

This is an IoT based platform where multiple types of IoT Applications can be deployed and monitored through web/desktop/mobile based application. The platform is developed so that it eases the work of developers handling each sensors separately, here the sensors management and controlling is kept abstract from users perspective.

1.2 Scope

Scope of the project is to build end to end platform for managing IoT devices where users can deploy their applications, collect data from platform, scale the platform to support high number of concurrent users, build dashboards to get insights from the data.

2 Intended Use

2.1 Intended Use

Consider a use case of taxi in super-app(kind of like everything can be deployed) as an IoT device, in this different types of services/features for taxi can be deployed like taxi-ride, food delivery, grocery pick and drop etc.

2.2 Assumption & dependencies

Assumption: Basic assumption at this point of time, that all the sensor supports REST/HTTP protocol for request and response through the data, for the time being only python codes can be deployed.

Dependencies: Any libraries which makes the work easier are going to be used, however not API's/libraries which uses the entire component of the platform.

3 Use cases

1. Based on some collected sensor data, some callbacks, action and event triggering is possible.
2. Use the sensors depending on the requirements of the application.
3. Automating daily tasks by leveraging the sensor data.
4. Analysis on the collected data (from the sensors).

3.1 Who are the types of users (name, domain/vertical, role, what they are trying to do when they need the system)

3.1.1 Admin

she/he will have the administrative rights to the platform, can add/remove sensors

3.1.2 Platform Developer

will be responsible for developing the various modules of the platform (like scheduler, deployer, load balancer etc) and the platform config files (needed for platform initialization)

3.1.3 Platform/Application User

Uses the applications that are deployed over the platform.

3.1.4 Application Developer

Responsible to upload the actual application codes that run on the platform.

3.2 At least 5 usage scenarios.

3.2.1 Smart Campus

Smart Campus can control various operations like automated AC controlling, smart parking of vehicles in Campus, automated street lightning, smart sprinkler controller, etc in an efficient way. For example, to turn on the AC if classroom temperature reached a certain threshold. Temperature sensor would be sending temperature to the application. Depending on the temperature, required actions like turning AC ON/OFF and increasing/decreasing the temperature can be performed. This solution would minimize power consumption and provide ease of access without the actual user interference.

3.2.2 Smart Vehicle

The vehicle will have sensors that will track the vehicles location, get information about the current trip, check if the fuel tank is empty, or any kind of intrusion to the vehicle (trying to break in or forceful opening) and this kind of data can be sent to the user via the dashboard or if the message requires immediate response, via device notifications so that the car owner can act upon it.

3.2.3 Artificial Traffic Police

Policing the roads by humans may be a problem depending on the location of the roads or the climate of the region. Roads like secluded highways or roads in regions where that are very cold and inhabitable also need surveillance. Any accidents or mishaps on such roads need to be reported. The data collection in this case is mostly in the form of video stream and analyzing this data and sending notifications to the actual authorities can be very useful. Video data after analysis can be used to detect and then report any anomalies or crimes.

3.2.4 Smart stores

Shopping without hassle taken to another level, these stores will have your store account that will contain your wallet information, shopping lists etc and when you enter the store, the sensors in the cart will track the items bought and will charge you automatically without any checkouts and billing lines. For this purpose inventory management and cart item tracking will be done by the sensors. This data can be sent to the server and invoice is sent to the respective payment agent and you can make an online payment accordingly.

3.2.5 Continuous patient monitoring system

form a connected healthcare system, which will enable Healthcare professionals to efficiently manage chronically ill and episodic care patients by monitoring their movement, heart rate, bp etc at defined time periods.

4 Application Model

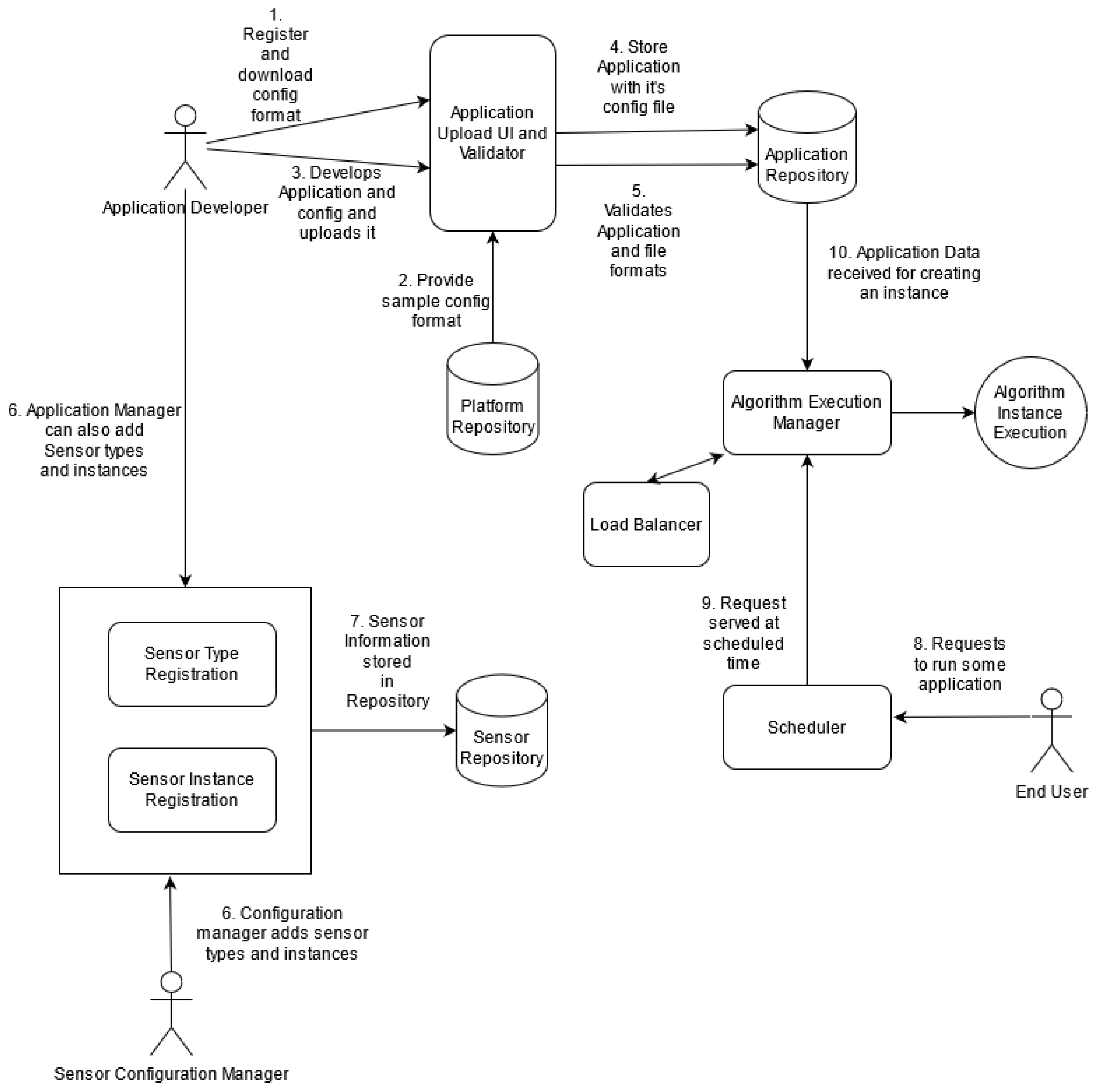
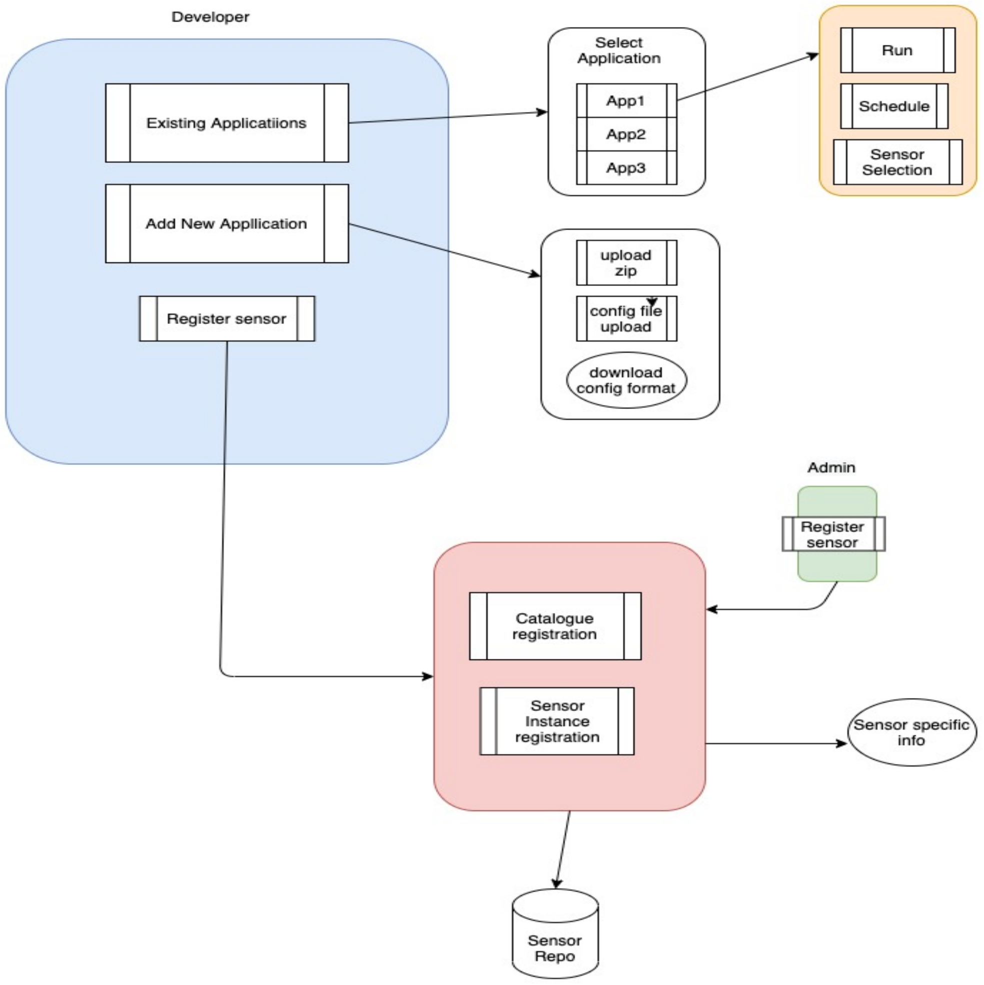

1. Register and download config format
Application Developer
Application Upload UI and Validator
3. Develops Application and config and uploads it
2. Provide sample config format
Platform Repository
4. Store Application with it's config file
Application Repository
5. Validates Application and file formats
10. Application Data received for creating an instance
Algorithm Execution Manager
Algorithm Instance Execution
Load Balancer
6. Application Manager can also add Sensor types and instances
Sensor Type Registration
Sensor Instance Registration
7. Sensor Information stored in Repository
Sensor Repository
9. Request served at scheduled time
Scheduler
8. Requests to run some application
End User
6. Configuration manager adds sensor types and instances
Sensor Configuration Manager

Figure 1: Application Development model

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 config 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 config 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 config file.
15. Hosts sending the response / output to the action manager to perform necessary action.
16. Action manager sending response to control the controller.

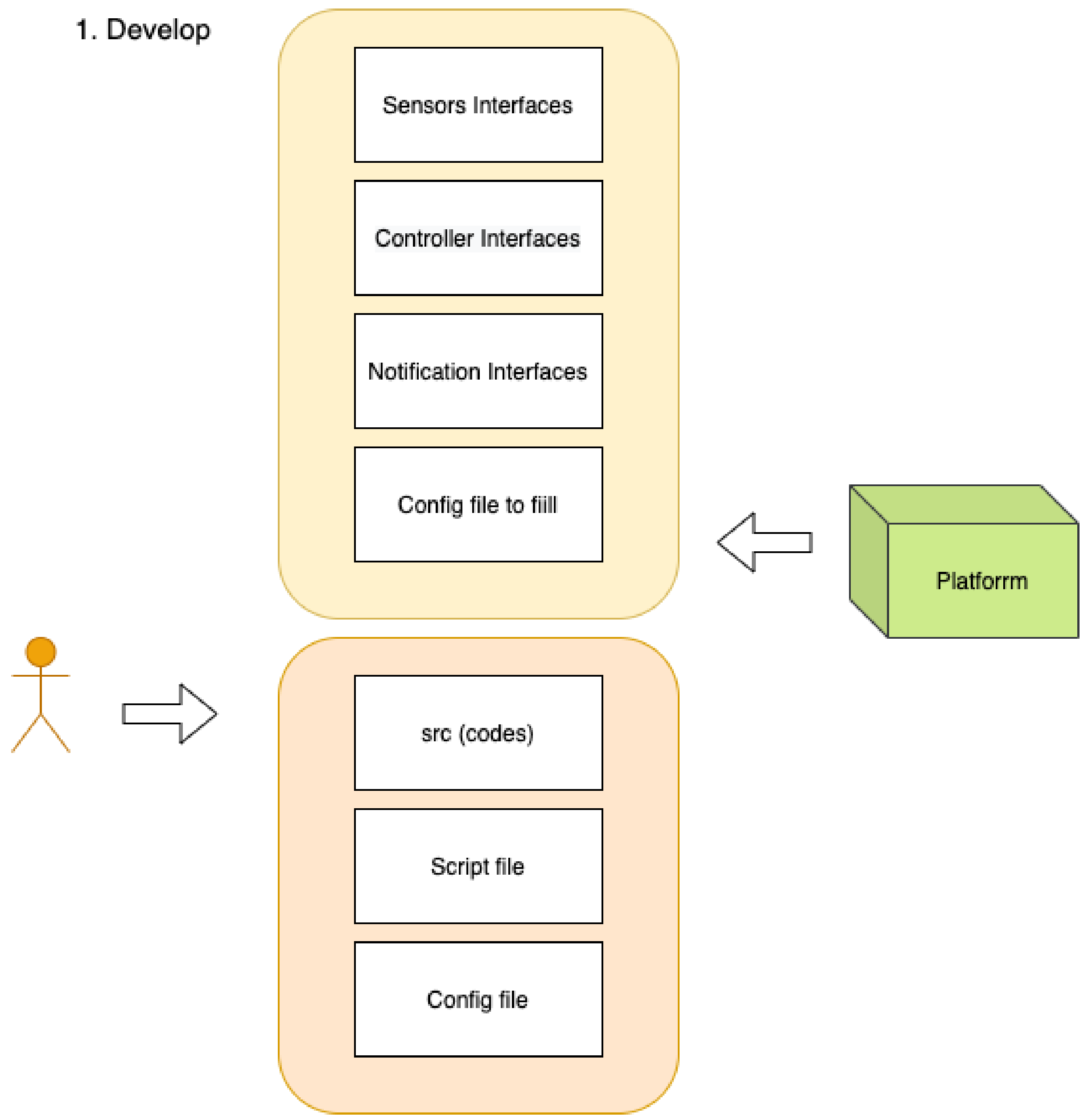
17. Sensors sending data to the sensor manager / Controller manager sends signal to manipulate the controller.
18. Sensor manager fetching sensor data (real time as well as stored) and input them to the code running on the nodes.

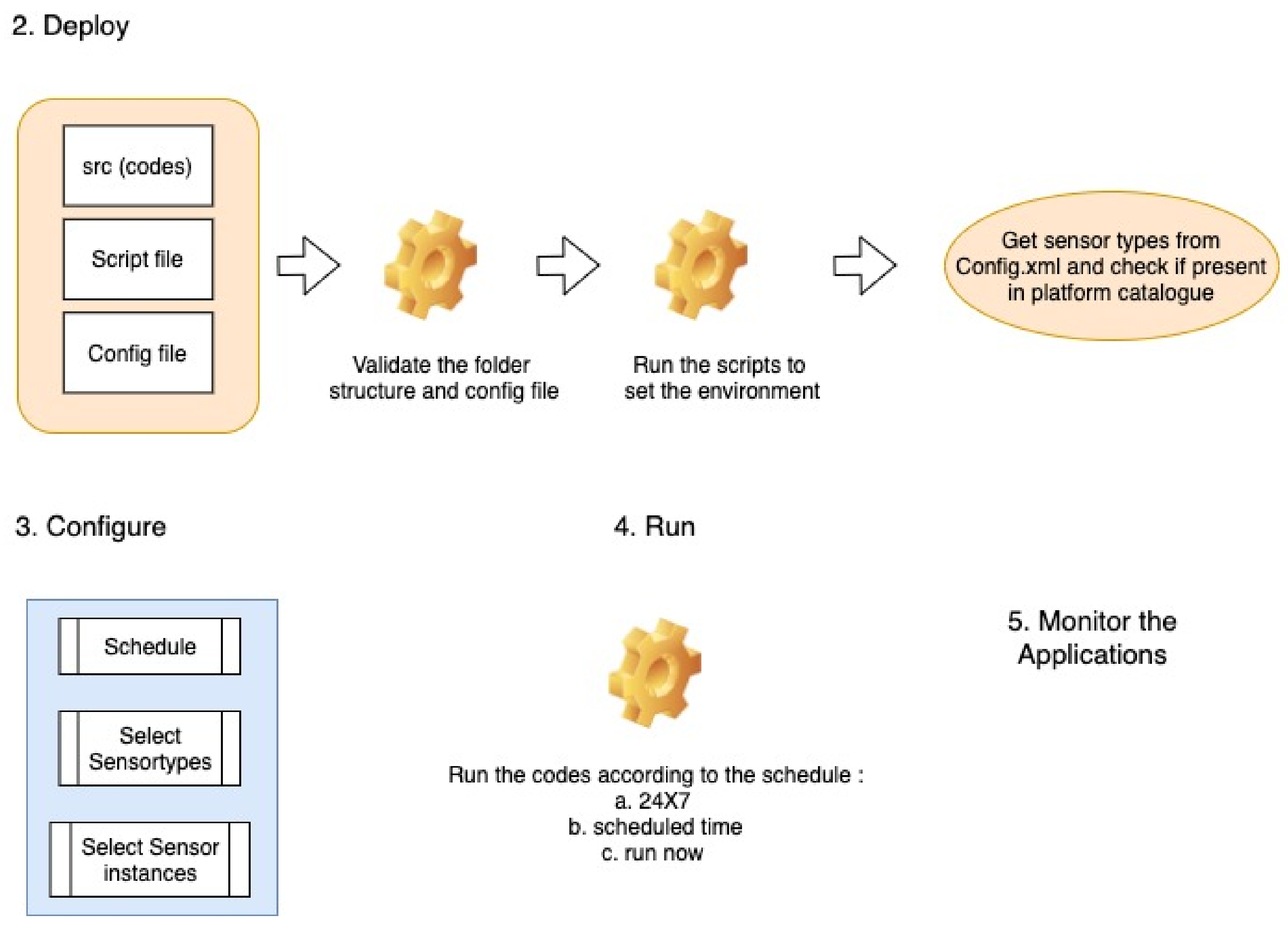
4.1 Dashboard



Dashboard UI Flow

4.2 Application Overview/Workflow




2. Deploy
src (codes)
Script file
Config file
Validate the folder structure and config file
Run the scripts to set the environment
Get sensor types from Config.xml and check if present in platform catalogue
3. Configure
Schedule
Select Sensortypes
Select Sensor instances
4. Run
Run the codes according to the schedule :
a. 24X7
b. scheduled time
c. run now
5. Monitor the Applications

Application Dev Model

4.2.1 Sensor Registration

At first we want the instances/sensors to be registered in our network/platform. The task of registering the sensor details will be performed by the application admin. The admin will take care of the following tasks using the sensor registration program:

1. Verification if the sensor to be registered is a type that is supported by the platform
2. The sensor will be given a sensor_id by the platform after registration.
3. A mapping of the id with the configuration details of the sensor will be there in a file/db.

4. Now this sensor will be identified by its sensor_id uniquely in our system.

This concludes the registration process for a sensor. Now in future , when the sensor will send data to the sensor manager via the gateway , then it will be stored in the db. Any real time data anomaly in the sensor data is monitored/picked up by sensor manager and sent to the action centre.

```
{
    "user_id" : "apurva",
    "sensor_catalogue_config" :
    {
        "sensor_type_1" :{
            "sensor_name": "gps_iiith_bus",
            "sensor_type_data_type": "string",
            "has_controller" : "yes"
        },

        "sensor_type_2" :{
            "sensor_name": "biometric_iiith_bus",
            "sensor_data_type": "string",
            "has_controller" : "yes"
        },

        "sensor_type_3" :{
            "sensor_name": "temperature_iiith_bus",
            "sensor_data_type": "int",
            "has_controller" : "yes"
        },

        "sensor_type_4" :{
            "sensor_name": "lux_iiith_bus",
            "sensor_data_type": "int",
            "has_controller" : "yes"
        }
    }
}
```

Listing 1: Sensor Type Registration Config Format

4.2.2 Application Developer's POV

After logging onto the platform, the application developer is presented with a dashboard which gives the following features:

1. View existing applications and their status.
2. Registered sensor
3. Add new applications :
    - Instructions for adding new applications (package details, config sample).
    - Upload new applications (package containing code and config).
    - schedule services: start,stop services.

4.2.3 Application Deployment

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.

```
{
    "Application1":{
     "algorithms":{
        "algo1":{
           "code":"control_sewing_service.py",
           "dependency":{

           },
           "environment":{
              "flask":"False",
              "python3-alpine":"False",
              "python3-packages":[

              ]
           },
           "sensors":{
              "sensor_type_1":{
                 "sensor_type" : "stitch_setting",
                 "all_sensors" : "no"
              },
              "sensor_type_2":{
                 "sensor_type" : "thread_remaining",
                 "all_sensors" : "no"
              },
              "sensor_type_3":{
                 "sensor_type" : "hook_rotation",
                 "all_sensors" : "no"
              }
           }
        },
        "algo2":{
           "code":"control_dyeing_service.py",
           "dependency":{

           },
           "environment":{
              "flask":"False",
              "python3":"False",
              "python3-packages":[

              ]
           },
           "sensors":{
              "sensor_type_1":{
                 "sensor_type" : "temperature_sensors",
                 "all_sensors" : "no"
              },
              "sensor_type_2":{
                 "sensor_type" : "color_sensor",
                 "all_sensors" : "no"
              },
          "sensor_type_3":{
             "sensor_type" : "camera_sensor",
             "all_sensors" : "no"
          }
        }
      }
    },

             "application_id":"Application1",
    "developer_id":"user_1"
   },
   "Application2":{

    "algorithms":{
      "algo1":{
        "code":"bus_service.py",
        "dependency":{

        },
        "environment":{
          "flask":"False",
          "python3-alpine":"False",
          "python3-packages":[

          ]
        },
        "sensors":{
          "sensor_type_1":{
             "sensor_type" : "temperature_iiith_bus",
             "all_sensors" : "no"
          },
          "sensor_type_2":{
             "sensor_type" : "gps_iiith_bus",
             "all_sensors" : "no"
          },
          "sensor_type_3":{
             "sensor_type" : "lux_iiith_bus",
             "all_sensors" : "no"
          },
          "sensor_type_4":{
             "sensor_type" : "biometric_iiith_bus",
             "all_sensors" : "no"
          }
        }
      },
      "algo2":{
        "code":"all_bus_service.py",
        "dependency":{

        },
         "environment":{
               "flask":"False",
               "python3-alpine":"False",
               "python3-packages":[

               ]
            },
            "sensors":{
               "sensor_type_1":{
                  "sensor_type" : "gps_iiith_bus",
                  "all_sensors" : "yes"
               }
            }
         },

     "algo3":{
            "code":"bus_barricade.py",
            "dependency":{

            },
            "environment":{
               "flask":"False",
               "python3-alpine":"False",
               "python3-packages":[

               ]
            },
            "sensors":{
               "sensor_type_1":{
                  "sensor_type" : "gps_iiith_bus",
                  "all_sensors" : "yes"
               }
            }
         }
      },
      "application_id":"Application2",
      "developer_id":"user_1"
     }
 }
```

Listing 2: Application Configuration File Format

4.3 Application Overview for Use-case : Smart Campus

At first, we want the sensors in the smart campus to be registered in our network/platform. So, here the user will provide the sensor details in his campus and give it to the admin. The task of registering the relevant sensors then falls to the application admin. Let's suppose the application admin has to add noise sensor in our catalog file, we have structures for camera and temperature sensors in the catalog file.

The registration process includes:

1. Verification if noise sensor already exists in our catalog or not.
2. If not, add a structure for the noise sensor in the catalog file.
3. The new sensors will be given a sensor_id by the program.
4. A mapping of the id with the config details of the sensor as given by the user(the one who registers) will be there in a file/db.
5. Now this sensor will be identified by its sensor_id uniquely in our system.

The sensor can now send data to the platform which will be stored in database. In addition, a real-time anomaly monitoring allows for actions to be taken in case of anomalous events. The application developer can now add programs/services that can use the registered sensors. Suppose the developer wants to add a service called service_x which gets temperatures from all classroom temperature sensors and switches on AC's in classrooms which have temperatures over some limit.

The developer uploads application zip+config to the system which will be stored in the repository. The application code will be of the form service_x.py and will be provided to the platform bundled as a zip, along with some other resources as needed. The config file for the application will contain information such as the kind of sensors that the application will use, dependencies and environment requirements. The package which contains the application code, config, etc is now stored on the application repository of the platform.

Now, when a request comes from the client, it is parsed by the system to determine which service needs to be deployed and when. Suppose the request comes for service_x to be run at some time. The deployer determines the time of deployment of the service. At the time of deployment, the relevant code and config are fetched. the load balancer determines the host machine on which the application needs to be deployed. At last, the deployer takes care of the environment and dependencies according to the details provided by the config, and deploys the application. After determining the relevant devices, an action command is sent to the action center for the required devices. Also a response can be sent to the user.

4.4 Scope

4.4.1 Technologies to be used

- **OneM2M**: OneM2M is a standard that provides a common M2M service layer that can be embedded within various hardware and software to connect IoT devices.The layered model of oneM2M comprises 3 layers: Application Layer, Common Services Layer and the under- lying Network Services Layer.
- **Kafka**: 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 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 configs.

4.4.2 Constraints

- Datatype: Sensor datatype is restricted to strings and numbers.
- Sensor catalogue should be registered before creating instance of that sensor.
- Any sensor should be registered before accessing it.
- Source zip file should follow particular format described above.
- Config file should follow particular format described above

5 System Features and Requirements

5.1 Platform Requirements

5.1.1 Deployment of the platform

The platform will need to be deployed on a hosting service. The platform initializer will start the platform modules as per various parameters provided by the platform Configuration file.

5.1.2 Applications overview on the platform

The applications that will be running in the platform will interact will various IoT sensors. Those sensors will be registered in the platform i.e the sensors will collect real world data and provide those data to the sensor server module of the platform. The platform will provide those data to the application and then the application will perform tasks according to the analysis of the data.For example like smart college campus. It contains various sub systems like Smart Parking(Monitoring of parking spaces availability in the campus), Smart Lighting(Intelligent and weather adaptive lighting in campus lights), Temperature control in different labs, classes and server rooms, fire alarm system etc.

5.2 Functional Requirements

5.2.1 Registering sensors

The sensors need to be registered in the platform. Whenever a new sensor is introduced it need to be registered in the platform. An unique id is given to each sensor to identify them. The applications can choose the required sensors from the registered ones to collect data from those sensors.

5.2.2 Interaction with IoT sensors

After registering sensors in the platform there must be proper interaction between applications and the respective IoT sensors. The sensor servers will collect the data from sensors and the sensor server can send those data stream directly to the applications and also can store for future analysis.

5.2.3 Identification of sensors for data binding

Each application will use different set of sensors. So it's required to identify the correct set of sensors using the unique ids for the respective applications and send updated data from sensors to the sensor manager. The sensor manager will provide updated data to that particular application.

5.2.4 Data Binding to the application

Data binding is a general technique that binds data sources from the provider and consumer together and synchronizes them. The applications will need data from sensors through the sensor manager. The applications should get updated data from the sensors immediately.

5.2.5 Scheduling on the platform

The scheduling feature of the platform will enable the facility to run or execute any application at some specific time or can be triggered after specific event. This feature is very useful to automate several actions in our day to day life.

5.2.6 Acceptance of scheduling configuration

A scheduling configuration is a part of the application configuration file that will be provided by the application developer as per his needs. Application Developer can mention the starting and stopping time of the entire application or some particular service. Scheduler will accept this configuration and will act upon it accordingly.

5.2.7 Starting and Stopping services

When a request is received to start a service from scheduler, deployer machine name from load balancer, application files from application repository and sensor topic from sensor manager initiates the application. At the end time, scheduler will hit stop deployement on deployer to stop the service.

5.2.8 Communication model

The communication model between the components is likely to be some kind of message passing model. This can be achieved using some kind of service like Apache Kafka etc. These message queues are very efficient when we want to provide similar data to large number of components. Since, there can be many applications which might need similar kind of data, so this could be a good choice.

5.2.9 Server and service life cycle

All the services send heartbeat to the service life Manager. If heartbeat is not received till 1 min, then deployer is informed about the service.

5.2.10 Deployment of application on the platform

Applications developed by different application developers can be deployed on our platform. These developers can provide the source code of their applications in a zip file along with a configuration file that will direct our platform in configuring this application and starting it on some host machine in our platform.

5.2.11 Registry & repository

Our platform will also support the Registration of new sensors, applications as well as users for the applications. These registrations will be kept in some sort of database Repository that will be central to the platform. This is done so as to ensure safe and secure access to sensor data and prevent unauthorized intruders from accessing the information.

5.2.12 Load Balancing

The load balancer, as the name suggests, provides host machines for various use cases such that the workload is balanced. When the application developer interacts with the platform, the load balancer is involved in assigning a set of host machines to be used. When the deployer required hosts to deploy the applications, the load balancer provides the host in which the app can be deployed.

5.2.13 Packaging details

The application developer provides the application as a compressed file which contains the actual application code. Further, a configuration file is also needed so that the platform can determine the configurations of the environment in which the application needs to be run. The platform will be responsible for deploying the provided code in an environment defined by the configuration file.

5.2.14 Configuration files details

Configuration files are needed to define the initialization conditions in various use cases.

- Platform Configuration file: Provides the configurations for the running of the platform.
- Application Configuration File: Provides configuration of the environment in which an application has to be deployed.
- Sensor Configuration File: Provides details about a new sensor which has to be registered to the platform.

5.3 Non-Functional Requirements

5.3.1 Fault tolerance

Fault Tolerance is the ability of the platform to withstand some technical failures that might occur due to some factors like power cuts, hardware component failure, misbehaving programs etc.

- **Platform** - Fault Tolerance will be inbuilt in the platform in such a way that most of the components are redundant. Whenever we have a failure, we can switch to some other server for the working. This holds true for the platform database as well which will be redundant so as to keep all the registration and sensor data safe.
- **Application** - With reference to the Applications deployed on our platform, they will have some kind of application configuration file which can be used to spawn up additional instances on redundant servers in case of some problem with the current instance.

5.3.2 Scalability

Scalability is a very desirable feature of any System. Scalability refers to the ability of our system to scale with the ever growing demand in computation power due to increase in our user-base or more expensive computing. Here I will mostly refer to the Horizontal aspect of scaling i.e. adding more machines to cope up with the demand.

- **Platform** - Our platform will be scalable in the sense that it can register many different sensors and pull data from them. In order to cope up with this, the sensor server can be made distributed in nature(hypothesizing). The other aspect will be increasing data storage limits, to allow for scalability in this regard, we will be using distributed databases. To allow for different application instances to be run for the users, we will provide and array of different servers that will host these applications. A load balancer will be in place to balance the load on these servers.
- **Application** - The applications that are deployed on our platform are also distributed in nature so they can also be scaled horizontally. We can have different machines hosting the instances of the same application to allow for good scalability. The application configuration file will be used to spin up a new instance of the given application. The choice of load balancer is still unclear but there needs to be some kind of load balancing for requests that are incoming to the application.

5.3.3 Accessibility of Data

Data accessibility is making data accessible to relevant actors. The accessibility to data may be defined through access levels as defined by an actor's role in the platform

- **Platform** - Data specific to platform includes status of applications, status of servers, sensors, etc. This data can be accessed by the means of various dashboards made for the purpose. The platform specific data should only be accessible for Platform developers and Platform Administrators, who are responsible for development and management of the platform respectively.
- **Application** - Application specific data includes the status of application, status of servers on which the application is deployed, status of sensors being used by the application, etc. This again can be accessed via dashboards by application developers. Applications can also access sensor data streams and stored sensor data.

5.3.4 UI and CLI for interaction

Our platform will have some kind of Dashboard for admins as well as application developers to watch out the different activities on the platform and the applications respectively. This Dashboard will most likely be a UI that can be opened in some browser. The CLI aspect of the platform is reserved more for the application devs and the platform devs who will use the CLI to develop and test the platform and applications as per their need.

5.3.5 Security -Authentication and Authorization

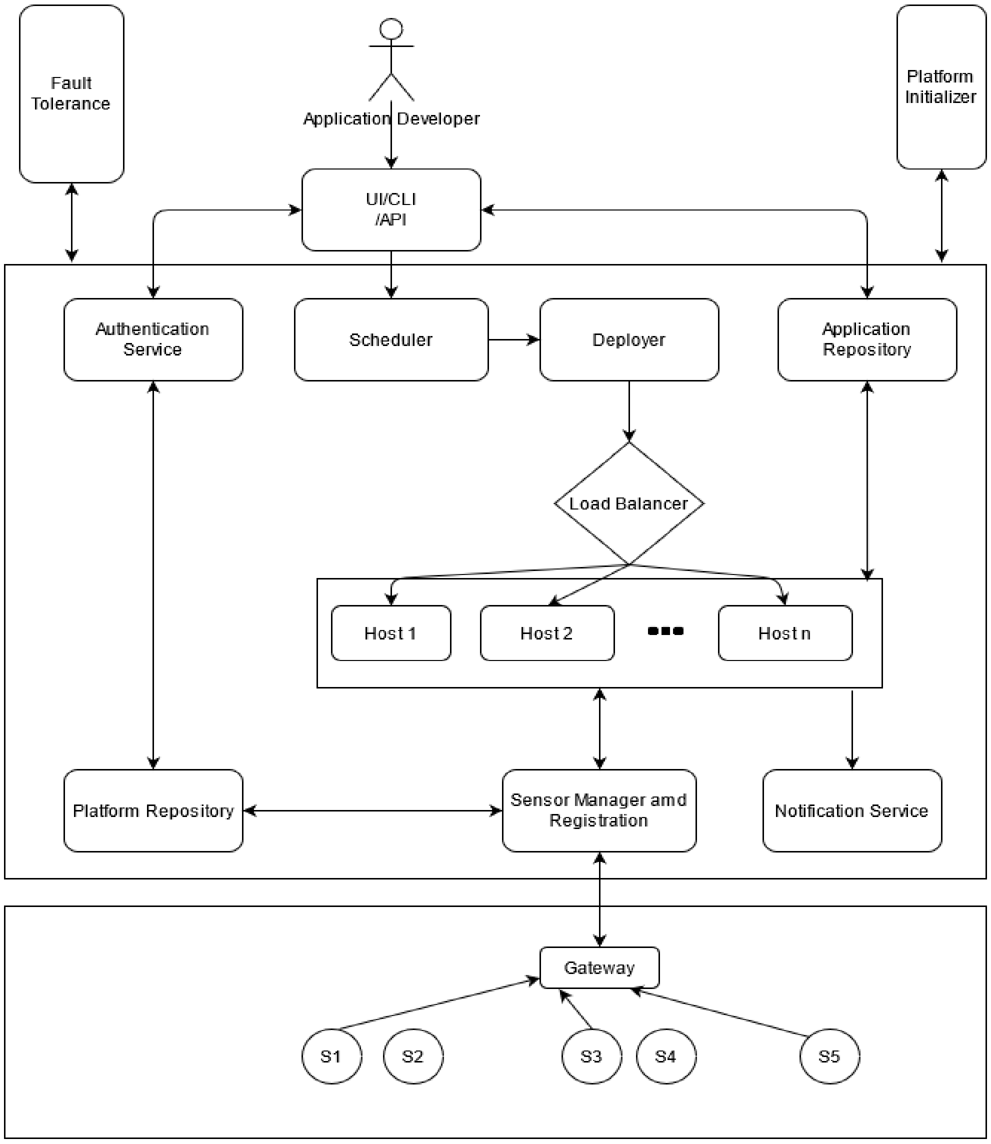
The platform will need authentication with relevant modules to ensure that there are no bad actors involved. The sensors will have to be provided some authentication to ensure that the data being received by the platform is from a trusted and registered sensor. The various actors associated with the platform also need an authentication, possibly in the form of a login password and unique id to determine what level of authority needs to be given regarding the platform. The security is needed to ensure no malicious interference in the functioning of the platform or individual applications.

5.3.6 Persistence

Our platform will provide persistence of data in the form of some kind of database. The sensor data and the user data will be stored inside some kind of SQL or NoSQL database. These databases are by far robust in nature.

6 List the key functions

6.1 A block diagram listing all major components (components)



Block Diagram of platform

6.2 Brief description of each component

There are four basic component is these:

- Platform Initializer: Responsible for Initializing the platform and keep it up and running.
- Platform Interaction Modes: Different modes of interacting with platform are UI/-CLI/API. Through any of these we can interact platform.
- Authentication Service: To get access first we need to authenticated ourselves. Database of users will be stored in Platform repository.
- Platform Repository: It is a repository for storing application developers authentication information, registered sensors data(format of data, type of data etc.), configuration files for code/features of application.
- Scheduler: Start/Stop a particular service, when to run which service is handled through this service.
- Deployer: Setup environment for the application, Import libraries, resolve dependencies.
- Notification Service: Alarms in Application to be redirect to notification centre to display to user
- Sensor manager: Responsible for managing sensors, scaling the sensors if needed, binding of sensors(like i need many sensors in my application then i need to decide flow of different how the different sensors will interact and work in sync)
- Load Balancer and Service Node Manager: The Load Balancer collects resource usage statistics from the service nodes and provides the deployer with the node with appropriate server node to deploy service on. The Service node manager maintains active service nodes and calls fault tolerance module for the nodes that go down.
- Hosting: Suppose we want to deploy more number of services than our current number of servers, we should support scaling of servers without disturbing current service.
- Fault Tolerance: It should be available across our IoT platform, some of them are services should be fault tolerant, our servers where services are running and many more. Some of the way to make fault tolerant service are Heartbeat Mechanism, Log Files, (search more methods of doing it).

6.3 List the 4 major parts (each team will take one part)

- Sensor Management: Registration and management of sensors and the sensor-server(APIs and stuff). (Maybe : storage of data in platform repository)
- Platform Development: Includes platform initialization, platform repository, platform interaction modes.
- Platform Robustness: This includes load-balancer, fault-tolerant, schedulers.
- I/P & O/P modes: This includes notification service, and configuration file format for interaction between services.

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

7.1 Name of use-case

Smart Campus

7.2 Domain/company/environment where this use case occurs

(Smart) Home / Office devices

7.3 Description of the use-case (purpose, interactions and what will the users benefit)

Smart Campus can control various operations like automated AC controlling, smart parking of vehicles in Campus, automated street lightning, smart sprinkler controller, etc in an efficient way. For example, to turn on the AC if classroom temperature reached a certain threshold. Temperature sensor would be sending temperature to the application. Depending on the temperature, required actions like turning AC ON/OFF and increasing/decreasing the temperature can be performed. This solution would minimize power consumption and provide ease of access without the actual user interference.

7.4 How is location and sensory information used

Location of the sensors will be determined by their geocode and the sensory information is : number of sensors

8 Subsystems

8.1 Key subsystems in the project

1. Platform initializer
2. Application and Deployment Manager
3. Sensor Manager
4. Scheduler
5. Load Balancer

8.2 Interactions involved across these subsystems

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 request handler (with load balancer) will relay the user requests to the various nodes on the components are deployed.

The application manager will act as the intermediary between Scheduler and the application developer.

The Scheduler, after getting required information from the application manager, will pass the information to the Deployment manager for execution of the algorithm at the scheduled time

8.3 Registry and Repository

Registry will be used to store several run-time related information for applications and platform modules like,

1. Login Details
2. Config files : Application code, scheduling
3. Address of all nodes
4. Sensor metadata and registering info

Repository will store all the static files. We'll have separate repositories for the Platform and the Application.

8.4 The four parts of the project each will have its own team requirement doc to be submitted

8.4.1 User End

User Authentication and registration and load balancer. Interface to the applications(routing and stuff).

8.4.2 Sensor Manager

Registration and management of sensors and the sensor-server

8.4.3 Platform Development

APIs for Application development and deployment. Also responsible for Platform Init using some config file.

8.4.4 Interconnection (kafka)

connects all the different components using kafka (or any other messaging queue).