Internals of Application Server

IOT Platform (Group 5):

1. Overview

1.1. Introduction

The Internet of Things (IoT) is gaining popularity day-by-day and has attracted the attention of many developers. IoT has brought significant change in the lives of people and in the near future its impact on society will increase more and connected things in IoT will be much more than people. In near future, IoT will transform real world objects into intelligent virtual objects.

The main objective of this project is to develop a platform which helps IOT application developers in developing their applications faster and in an efficient way by providing the common functionalities that are common to IOT applications in general.

1.2. Scope

The purpose of this project is to develop a standalone platform which is capable of hosting, scheduling and running several applications. It provides an interface which makes it easier for them to communicate with sensors and get data from them. It also provides functionalities like notification, logging, monitoring and authentication making life easier for IOT application developers.

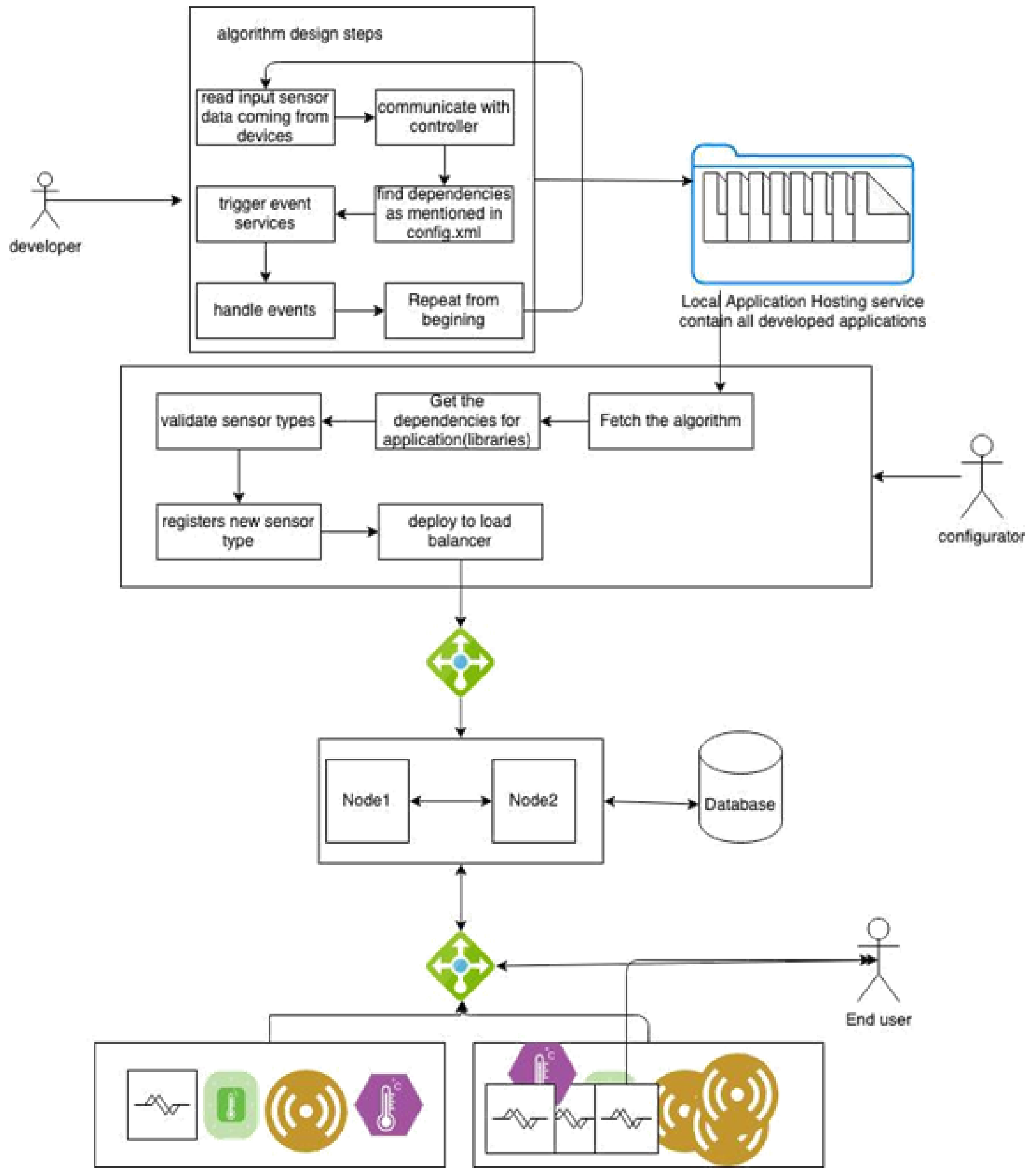
2. Intended Use

2.1. Intended Use

IOT application developers can use the generic functionalities like communication with sensors, notification services in their application development code. Our platform can be used for deploying their applications, hosting their applications, registering new sensors to the application using the format specified in the application development model.

3. Application Model

3.1. Block Diagram



algorithm design steps
read input sensor data coming from devices
communicate with controller
find dependencies as mentioned in config.xml
trigger event services
handle events
Repeat from begining
developer
Local Application Hosting service contain all developed applications
validate sensor types
Get the dependencies for application(libraries)
Fetch the algorithm
registers new sensor type
deploy to load balancer
configurator
Node1
Node2
Database
End user

Fig.: Application Model

3.2. Application Interface:

Let us consider the following example for understanding the interface:-

Pulse Rate Monitor

The purpose of the application is to monitor the pulse rate of ICU patients and alert a doctor or a nurse if uneven pulse rate detects.

Sensor types:

1. optimal heart rate sensor

Algorithm Interface:

1. Name of the Algorithm: Pulse rate alert
2. Sensors:
   a. Pulse Rate
3. Notification: Uneven pulse rate alert
4. Script file names:
   a. algorithm
5. Sensor types details:
   a. Pulse Rate:
      i. attributes: on, off, monitor
      ii. geolocation: xyz hospital, hyderabad
      iii. location: ICU1, ICU2, and etc.,

Algorithm:

```
Pulse_rate_alert()
        1. Read from Pulse Rate
        2. if >140 or <40
        3. notify
        4. continue from step 1 until power off
```

3.3 Key Elements

Algorithm development and integration -

This is one of the important key elements in our application development platform where developers must be able to develop the algorithm scripts and define the set of sensor types and its corresponding controllers based on the template.

Input types of data can be read from the registered driver sensor type. Action and events can be configured with the help of interface classes that are common algorithm types. We can call the action methods defined in classes in the form of dependent libraries while the user is trying to integrate or configure to deploy.

As we are using this algorithm for specific application purposes there must be a large application which has the need to implement multiple algorithms to create big applications. Our application deployment must be able to handle the case of integrating *N* number of to algorithms and study its own **config.xml**.

Scheduler Service -

Scheduling is essentially a decision-making process that enables resource sharing among a number of tasks by determining their execution order on the set of available resources. The process of managing task allocation, to where and to who, is the responsibility of a scheduler. In other words, a *scheduler* is responsible for managing incoming process requests and taking scheduling decisions to determine which process needs to be executed based on process priority, job type etc. The scheduler of a distributed system performs akin to the process scheduler on any operating system.

Application deployer service –

As part of deployment service admin users must be able to fetch the list of hosted applications or libraries. Compiling or building the application to check for any code breakage. Along with this validation we must ensure run-time validation to find the dependent libraries exist. config.xml acts as a main configuration file for application integration with sensor types and remaining data. Sensor checking is also integrated as a part of validation

Sensor type registration service –

During the runtime if an issue of missing sensor type is encountered, we need to have a mechanism to add a new type of sensor. So, the new sensor type must be identified on the platform for future application integration.

Configurator service –

Based on the type of application and need to read the data from sensor live or on demand the data capture varies. We must provide flexibility to provide various types of running modes for the application. Like was discussed in class we can have an option of going demand mode of running the application which means it reads the data only based on given demand. There may be a case where we need to run the application continuously, for instance, like mining where it works all day with change shifts, but every minute monitoring must be implemented.

Configuration of live sensors to integrate with our deployed application and we directly send the data to database runs in background so that it is highly unlikely we might lose data. But data comes every minute so we need to find a way to have an interface between both of us.

Load balancer service –

It has an ability to run applications per environment because earlier versions of applications must be using the default port where new deployed applications must run from the same host but from different ports. At the same time, it is the responsibility of a load balancer which acts as a container to find out the free node where we deploy the app in the necessary node which is free.

3.4. Application Flow:

1. Algorithm development

a. This is the key area of the IOT based application development platform. The developer is responsible for creating algorithms and giving patches to the existing apps.

2. Host all developed apps/algorithms local repository system for configurators/admins.

a. This is an internal local service used to get hold of all the developed applications. While configurators use this service to fetch the required application type and configure the necessary devices.

2. Scheduling

1. Receives a JSON file from application developer.
2. Look for scheduling components like start time/end time/interval/duration

3. Start and end the application at the particular time instant or on the basis of required intervals.
4. Manage the immediate response of starting/ending of the application.

3. Deployment

a. Internal validation steps to check the correctness of code.

b. Validation of config.xml

c. Validation of dependencies from config.xml

d. Locate load balance servers to deploy on the target machines.

4. **Register sensor type if it does not exist**

a. As a part of deployment validation supporting the sensor type is mandatory

b. If sensor type is not found, we need to register the sensor type and may need to ask the developer to modify code changes to the algorithm.

5. Configuration

While the application is working for some stage, we need to make configurations to the same application as scheduling the run, event management as post action for configuration.

6. **End user dashboard/End user alert system – Event manager**

We have a monitoring system to track the application status, sensor and its corresponding controller connections, nodes registered in load balancer.

4. System Features and Requirements

4.1. Platform Requirements

a) Deployment of the platform

Platform deployment will be performed after a self-check which will consist of registering all available sensors, connecting to the oneM2M server, and establishing contact/synchronizing with the total list of applications.

b) Different actors on the platform

The actors will consist of:

1. Admin
2. Application developers

3. End users
4. Platform Developer

c) Applications overview on the platform

Since we are not implementing a specific use-case, any application which will require data from IoT-based sensors, can be developed and integrated with our platform by making use of the various API endpoints provided.

4.2. Functional Requirements

a) Registering sensors

Sensors are assumed to be virtual, which will be registered automatically by the application manager once the platform comes online.

b) Interaction with IoT sensors

Sensors will generate a randomized stream of data based on certain velocity values and number of streams (all configurable). Certain triggers/conditions will be set, which once satisfied will generate an event which can be detected and processed further.

c) Development of application on the platform

There will be a set of configuration files/APIs/parameters provided by the application manager which must be followed for new applications, when they are made by the developers.

d) Identification of sensors for data binding

Sensors will have a unique ID which will be mapped to the list of types of sensors.

e) Data Binding to the application

Communication module between deployment manager and sensor manager will help in binding the streaming sensor data with active algorithms (running instances).

f) Scheduling on the platform

The deployed application can have different scheduling policies either instant or may be having within a certain time interval. There can be dynamic scheduling depending upon the needs of the application and various preemptive features of the server.

g) Acceptance of scheduling configuration

Scheduler module accepts the information from the application manager which validates the information of the user and now the scheduler will handle its scheduling information given by the user.

h) Starting and Stopping services

Service management will be controlled by the deployer based on the commands received from the application manager, the scheduler, and the communication manager.

i) Communication model

Application manager will perform a basic bootup sequence, initialize the platform and connect to the oneM2M server. Communication manager will communicate with sensors and end-user applications and provide requests to the scheduler. The scheduler will assign the requests based on priority and other criteria to the deployment manager. Deployment manager performs load-balancing and distributes the jobs to various server nodes.

j) Server and service life cycle

Server life cycle controls all services and checks where to run a service. The ip/port of the less used server is being provided and then accordingly application runs on the given ip/port.

k) Deployment of application on the platform

Application developers can deploy their applications on the platform following some format specified in the application development model of this platform.

l) Registry & repository

Registry contains sensors that are registered by the admin(or end user in some cases). Registering sensors in the platform requires them to follow some format as specified in the application development model.

Repository contains the code of different applications that are deployed on the platform. It contains information like name of developer, version of code release, date of deployment etc.

m) Load Balancing

The main server may be subdivided into several smaller instances, with different applications getting connected to different instances. Sensor data and configuration files will be shared, and there will be a common database.

n) Interactions between modules

Application manager will handle inter-module communication using JSON/XML configuration files.
Communication module will look after the passing of information between different modules.
Server Lifecycle Manager will look after the monitoring and checkup.

o) Packaging details

Depending on the application requirements developers are required to specify all the packages that the application is dependent on in the format specified by the application development model.

p) Configuration files details

Configuration files will be JSON/XML and will contain server runtime parameters, sensor types, application configuration and saved states.

q) Interaction of different actors with the platform

- Admin will provide initial arguments and configuration parameters to the oneM2M server and will start the platform. At any instant, the platform status can be viewed from the admin dashboard, and certain runtime variables can be changed.

- Developers will have access to the JSON/XML configuration files and all the APIs used for communication with the platform. Their communication with the platform will be in an abstract fashion based on the access-points provided.
- End uses are not concerned with the backend configuration of either the platform or the applications. They will only be able to access the applications they are authorized for, and view the data provided by them.

4.3. Non-Functional Requirements

a) Fault tolerance

i. Platform

1. Deployment manager and application manager will handle fault tolerance, and will restart node servers as and when required

ii. Application

Fault tolerance will be handled by application developers

b) Scalability

i. Platform

Deployers will automatically scale the number of node servers based on the services assigned by the scheduler. In case resources are unavailable, the deployer will signal the scheduler not to assign any more jobs/services.

ii. Application

Currently, our platform has no upper limit for the number of applications.

c) Specification about application

Specifications will be fixed by the platform and will be followed by the application developers when designing applications. Specifications are defined as the endpoints defined by the communication module.

d) UI and CLI for interaction

Admin dashboard will have an UI designed in HTML, and platform and sensor startup pages will have basic CLI started and initialized by using command-line arguments.

e) Security - Authentication and Authorization

Admins and end users will have a password-based authentication protocol to access resources. Since the two types of users will access different protocols/services, roles are not necessary.

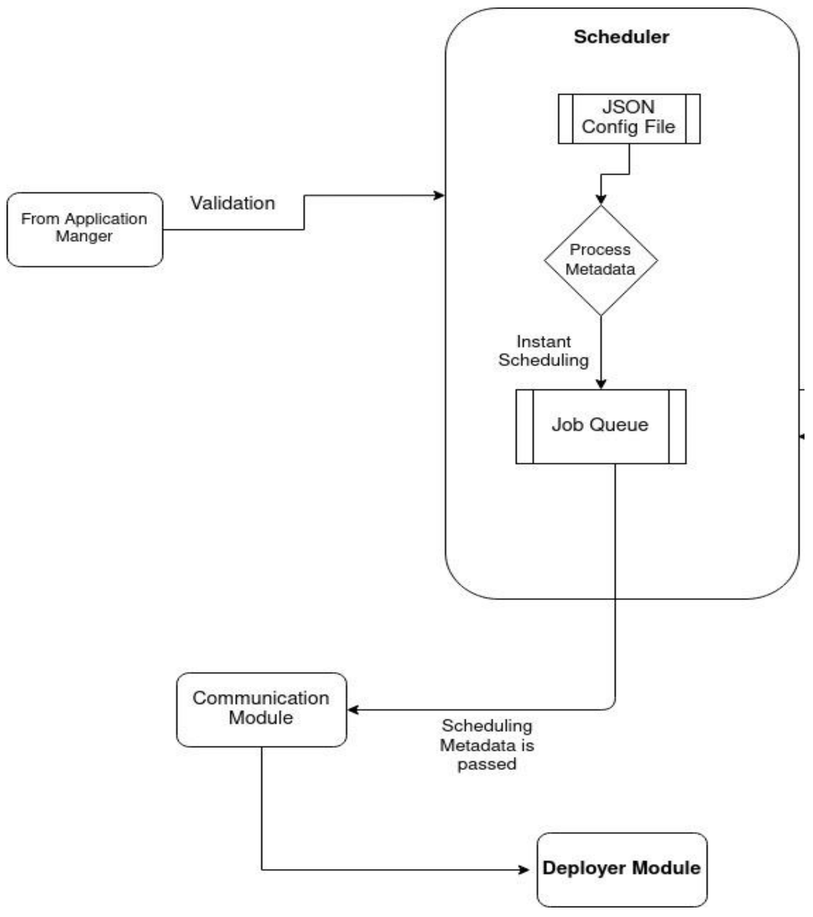
f) Persistence

Data recorded in the sessions will be saved in JSON/XML format.

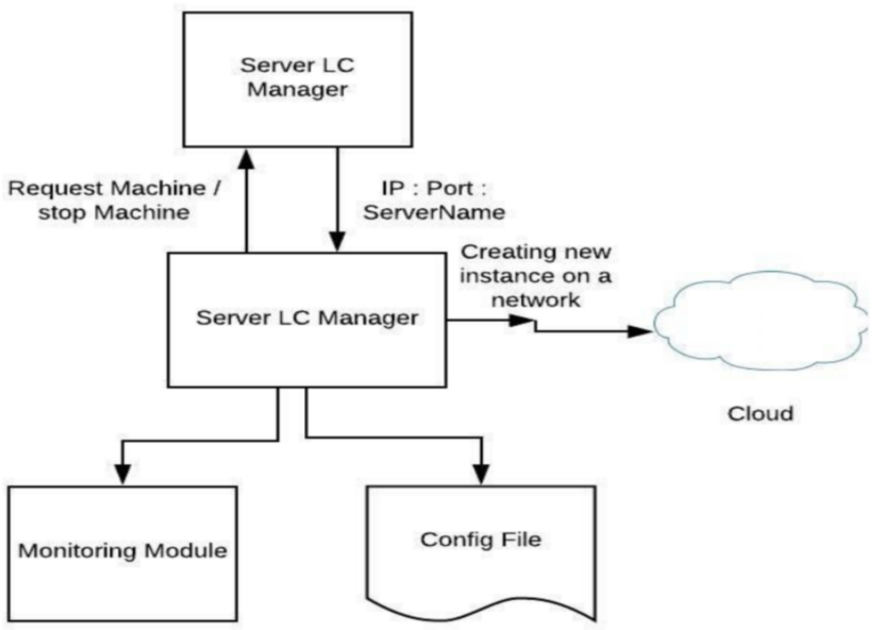
5. List the key functions

5.1. A block diagram listing all major components

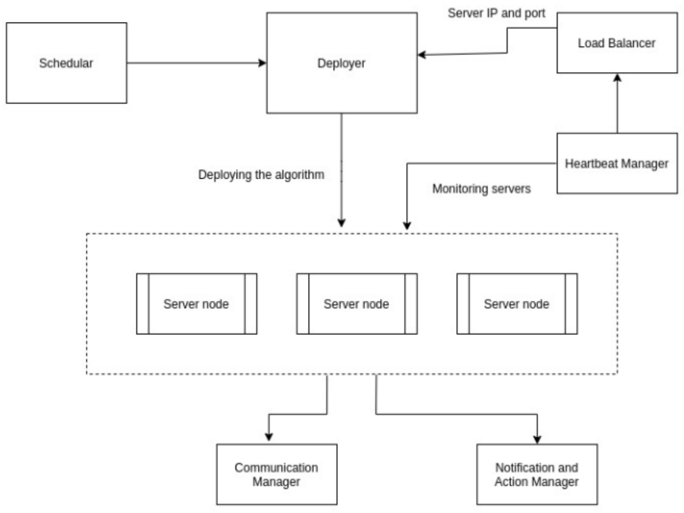
A. Scheduler



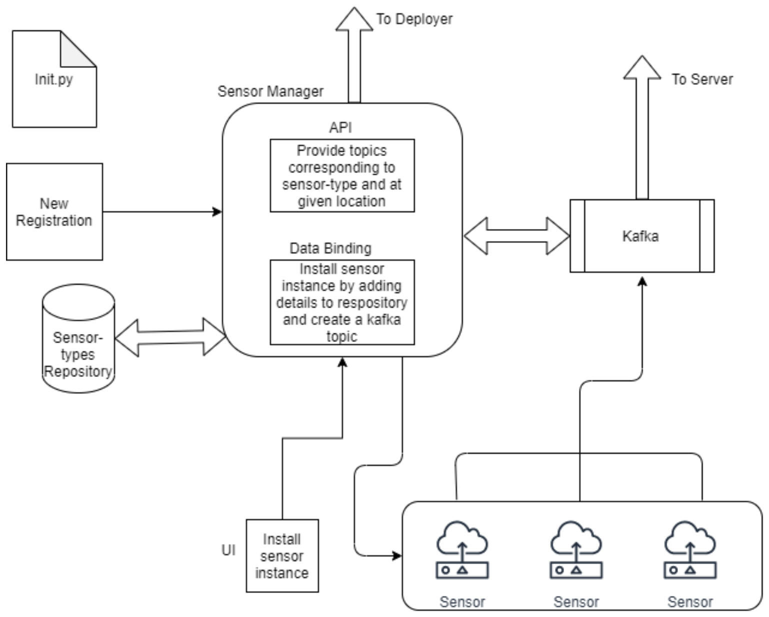
B. Server and Service Management



C. Deployer



D. Sensor Manager & Data Binding


To Deployer
Init.py
Sensor Manager
To Server
API
Provide topics corresponding to sensor-type and at given location
New Registration
Data Binding
Install sensor instance by adding details to respository and create a kafka topic
Kafka
Sensor-types Repository
UI
Install sensor instance
Sensor
Sensor
Sensor

5.2. Brief description of each component

A. Scheduler

The module is responsible to trigger the deployment manager at a specific time given by the user/client. The instant of start and end time according to the needs of the user is provided and the scheduler module takes care of that accordingly. Also if any user wants to end the runtime of the application, just at that instant a call is given to the scheduler to stop the running application prior to the end time. Similar is the case for immediate scheduling of the application.

B. Server and Service Life cycle Management

The series of states through which a Server instance can transition is called the *server life cycle*. At any time, a Server instance is in a particular operating state. Commands—such as start, stop, and suspend—cause specific changes to the operational state of a server instance.

C. Deployer

Deployment is generally defined as publishing an application or resource to a Web server, making it available for use. The deployer will deploy applications/algorithms defined using the constraints specified by the application/communications manager. The scheduler will assign services/jobs to the deployer.

D. Sensor Manager

Sensor Manager will mainly provide various APIs to the application developers for accessing the sensor data directly in their algorithms. The application deployer will be able to register the types of sensor required by the application and then install as many instances of those sensors as required by the application. The data will be entirely managed and scaled by the Sensor Manager thus freeing the developers from hassle of dealing with the sensors and hence provide an easy way for developing and deploying IoT applications with minimal effort.

E. Communication module

Using the Communication module the deployer communicates with the load balancer for assigning server nodes. Deployer communicates with the sensor manager using Kafka for assigning appropriate sensor node for data binding between algorithm and the sensor data. The sensor manager passes the topic name to the nodes for communication to the Deployer using the communication module (Kafka).

6. Use cases

6.1. What can users do with the solution?

- Deploying.
- Monitoring.
- Access to APIs for accessing sensor data.

6.2. Types of users

- Application Developers.
- Platform Developers.
- End users of Applications Deployed.

6.3. At least 5 usage scenarios.

- Patient Health Monitoring System in Hospital.
- Collision Avoidance System in cars.
- Agri Watering Solution.
- Network of wearables for monitoring health.
- Smart cities

7. Primary test case for the project

a) Name of use case.

b) Domain/company/environment where this use case occurs:

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

d) What is the “internet: angel around these things”

e) Information model

f) How is location and sensory information used:

g) Processing logic on the backend

h) User’s UI view- what is their UI, where and all will they do with this system

8. Brief overview of each of the four parts

a) What are the four parts (each team will work on one part)

1. Server and service management module.
2. Deployment Module.
3. Sensor Manager
4. Scheduler.

b) Functional Overview

- Server and service management module: The series of states through which a Server instance can transition is called the server life cycle. At any time, a Server instance is in a particular operating state. Commands—such as start, stop, and suspend—cause specific changes to the operational state of a server instance.

- Deployment: Deployment is generally defined as publishing an application or resource to a Web server, making it available for use.

- Communication Module: Communication module is the core part of the IOT platform. It acts as the central module between deployment manager and sensor manager which will help in binding the streaming sensor data with active algorithms.

- Scheduler: scheduler is responsible for managing incoming process requests and taking scheduling decisions to determine which process needs to be executed based on process priority, job type etc. The scheduler of a distributed system performs akin to the process scheduler on any operating system.