Internet of Things Apps Platform

A Project Design Document
Version-1.0

Submitted by

Team 1 :

Lokesh Walase	201405597
Pankaj Shipte	201405614
Sourabh Dhanotia	201405605

Team 2 :

Abhishek Mungoli	201405577
Swapnil Pawar	201405513
Veerendra Bidare	201405571

Team 3 :

Abhinaba Sarkar	201405616
Aditi Jain	201405549
Atul Rajmane	201405529
Prerna Chauhan	201405544

Of
Group Number - 5

Under the guidance of
Mr. Ramesh Loganathan

For the course
Internals of Application Server
IIIT, Hyderabad

23rd March, 2015

Abstract

This design document gives the overall design of the platform. It discusses approaches taken to implement various modules of the platform, interactions between them, how will end user applications interact with the platform among many other things.

Contents

List of Figures

Chapter 1

Introduction to the Project

The next wave in the era of computing will be outside the realm of the traditional desktop. In the Internet of Things (IoT) paradigm, many of the objects that surround us will be on the network in one form or another. Thus environment around us will be an invisible mesh of information and communication systems constantly exchanging data, talking to each other. This results in the generation of enormous amounts of data.

This data have to be dealt with at three different levels -

- Storage - Store the relevant data in required format in a database.
- Processing - Process/filter the data as per systems' requirements.
- Delivery - Deliver the required data to the end-user in a specific format.

IoT is heterogeneous in nature as it consists of a variety of hardware devices, interacting with software-systems, each system having different communication protocols. Hence creating an app for a new IoT system is not an easy task. To ease this problem of heterogeneity, we propose to create

- IoT Apps Platform. The primary objective of the IoT Apps Platform is to provide a generic (protocol and hardware independent) platform that will make the data from IoT available to the mobile app. It will majorly speedup the process of creating different mobile apps since the intrinsic details of handling the IoT data are taken care of, thanks to the platform. Thus our platform will seamlessly handle the storage, processing and delivery of the data that is generated by the Sensors and make it available to the app.

Chapter 2

Test Cases

2.1 Test cases - used to test the team's module

- In guided Parking system, The user will connect with the platform and ask for a free slot. If 2 users requests at the same time, the platform should not assign the same parking slot to them.
- In wildlife sanctuary, when an animal comes in the area of the sensor twice, He should be counted only once.
- In Gas leakage detection system, the sensors should not report the normal gas pressures as a leakage.
- In vehicle type temperature monitoring, The application tier should notify the application irrespective of the active or inactive status (emergency situation)
- In smart garden, it should be monitored properly as which plant needs water. (Do not overwater a plant)

2.2 Overall project test cases (relevant to the module)

For the use case - "Guided Parking System", the following scenarios are possible :

- The user at the parking building will turn on his app send request to the server. The server responds with exact free parking slot details(for eg - floor number, section number) on his app. This slot is the nearest one to the user and he is given full directions details to reach the destination. The user is expected to reach his destination within a time period.
- If a free parking cell is identified and the parking cell is allocated to a user, a session is maintained for certain period of time within which the user is supposed to reach the parking cell. On failure to do so in that period of time, that cell is deallocated from that user and reused to allocate to other users.

Chapter 3

Solution Design Considerations

3.1 Design Big Picture

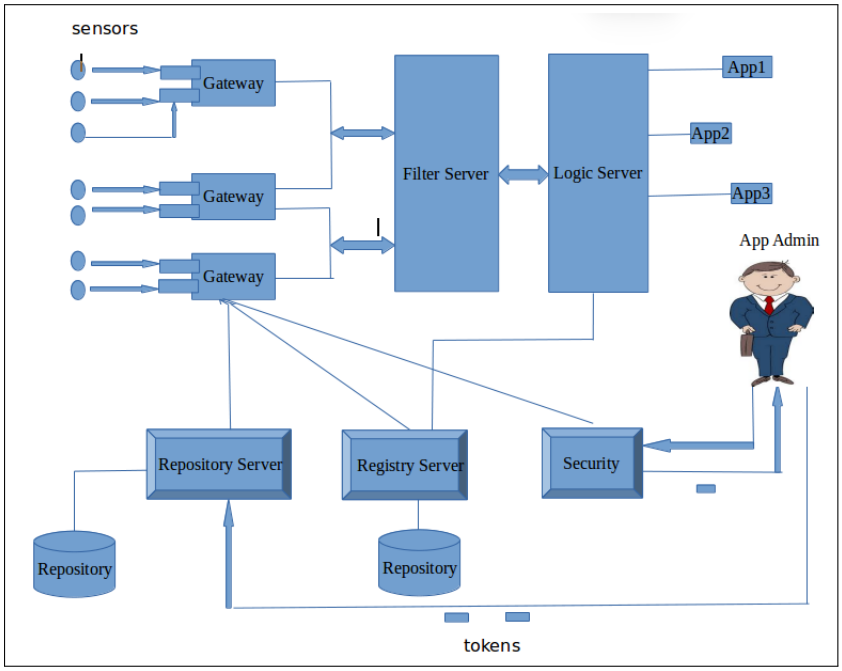

Figure 3.1: *Block Diagram of IoT Apps Platform*

3.2 Environment To Be Used

- **NPM -** NPM provides an execution environment for Node.js on all of the platform server - Filter server, repository server, registry server.

- **Chrome V8 -** Chrome V8 provides an execution environment for JavaScript on all the gateways.

3.3 Technologies To Be Used (Why, Where, How?)

3.3.1 Node.js

Node.js is a platform built on Chrome's JavaScript run-time for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Registry and Repository servers are implemented in Node.js and since they use json files for configuration, this provides very good integration and responsiveness. Also, Node.js provides very good support for RESTful APIs.

3.3.2 REST

RESTful APIs run on top of HTTP protocol and since HTTP protocol is supported by all the environments, it makes the communication in heterogeneous environment easy.

REST is implemented using Node.js in Registry and Repository and is used to support query API.

REST uses JSON objects to communicate, which is supported by all technology stack involved in different components of our platform.

3.3.3 JavaScript

JavaScript runs on all the Gateways. JavaScript is a better choice because it being a client side scripting is fast. It is easy to learn and implementation becomes simpler with JavaScript.

3.3.4 JSON

SON is simple and easy to use compared to XML. Moreover, the platform is exposing RESTful APIs. Therefore, JSON looks like a better choice for the data representation.

Both the repository and registry files are maintained in JSON format. All the RESTful APIs of the platform use JSON for the upstream and downstream data representation.

3.4 Approach For Device Gateways

- In our architecture, different types of registered sensors (viz, motion temperature and proximity, etc) send raw data to the gateways and gateways are simulated with Android devices.
- Gateways are registered in repository server . At boot time the type handlers are loaded for a particular gateway from the jars present in the repository server. Protocols for a particular type are defined in the type handlers.
- When data is received from the sensors it is first checked if protocol

for that type is defined. If it is present the data is then passed to the gateways.

- Gateways health ping sends the health of sensors connected to it and this data is then updated in the registry.
- Sensors can send data to the gateway only if gateway is up. For any broken down gateway the platform won't be able to receive data from the sensors connected to it until a new one is put in place.
- The gateway then converts the data to the required format as per the communication protocol.
- Since there is a mesh of gateways some routing algorithm will also implemented for smooth interactions between gateways.
- To handle heterogeneity Macro Programming will be implemented. There will be an external layer which will automatically decompose the program depending on the target deployment.
- The gateway then sends the formatted data to the central filter server through TCP/IP.

3.5 Device Type/Interface And Information Structure

- Sensors communicate with Gateways over BLE or WiFi. And the data that they send is in raw format. The interval at which the sensors generate data, the type of data is specified in the sensor configuration dictionary when they are introduced in the system.

- For each type of data different type handlers are defined and the data is received based on the protocol defined in the type handlers. If the type handler is not defined during bootup data from that sensor will not be received by the gateways.

- In case the status of the gateways defined in the registry server states that the gateway is down then the data from the sensors connected to that particular gateway will not be received

3.6 Devices Registry And Repository

3.6.1 Repository Server

- All the gateways that will be communicating with the filter server to send data needs to be pre registered in the repository server and gateways can only be added before bootup.

- Type handlers which define the protocols for various type of sensors are to be defined in the repository and will be loaded during boot time.

- It is mandatory for the sensors to be registered with the platform.Their entry describes key attributes of the sensor viz ID, gateway which it is a part of, family that it belongs to.

- New gateways and sensors can be added only by the app admin interface.

3.6.2 Registry Server

- Timestamps of each gateways which indicated when was the last time data was received from the gateways.
- Gateways health ping sends the health of sensors connected to it to maintain the status information of the sensors.

3.7 Approach To Get Device Information (Event Streams)

Devices/Sensors are periodically sending the data to the interfaces of the Gateways. On receiving the data the Gateway knows which type is the data of and it executes the pertinent type handler. The intervals at which the devices are sending the data varies from device to device and those intervals are governed by the configuration in the repository. Devices send not only the data that they produce but their health status information as well. These health pings too are periodic.

Gateways keep collecting these data and convert them to agreed upon format in order to pass them on to the Filter Server. As and when the data is required Filter Server sends requests for the device information. These requests are in fact simulated pushes. That is, Filter Server keeps en-queueing the commands at its end and whenever Gateway asks it for the commands it returns the content of this command queue. These simulated pushes happen periodically.

Typical command to get the device information is a query. It can be of one of the two types. One is, location based which returns the information

of the devices in the close proximity of the the location requested. And the other is type based which returns the information of the devices of the specified type/family. To answer such queries, Gateways have to communicate with Repository server or use the device Repository directly for the required information.

3.8 Device Information Proxy Server

3.8.1 Repository Server

This server facilitates the maintenance of the device repository. Device repository is a JSON file that has the information of all the devices, gateways registered with the platform. In addition to maintaining the repository, this server also helps gateways in answering the data Queries that ask for the data coming from a certain set of sensors.

Whenever a new device or a Gateway is added to the system, application administrators update the repository through an interface provided to them. This interface is directly coupled with the Repository Server. This interface also requires the application administrators to provide the required executable files to handle the data sent by the newly added devices. These executable files are later used by the type handlers at the Gateways.

The Repository Server is communicated with by only the Gateways and the application administrator interface.

3.8.2 Registry Server

This server facilitates the maintenance of the device registry as well as helps gateways/filter server in answering the data Queries that ask for the data coming from a certain set of sensors.

Device registry file has all the status information of the devices. The maintenance happens through periodic health pings received from the Gateways. A Gateway sends health ping not only for itself but for all the devices connected to it.

Both the filter server and the gateways can communicate with this server.

3.8.3 Security Server

This server makes sure that the resources of the platform are accessed by only the authentic users. It issues security token for the authenticated user and that token is sent along with subsequent requests.

3.9 Communication Overview

- **Devices and Gateways -** This one way communication from Devices to Gateways happens across heterogeneous protocols. The transmission of the data is periodic.

- **Gateways and Filter Server -** This communication happens over TCP. The data format used is JSON. The filter server pushes the commands to Gateways which in turn execute them for the data sent by the devices connected to it. These pushes are simulated pushes. That

is, Filter Server keeps en-queueing the commands at its end and whenever Gateway asks it for the commands it returns the content of this command queue. These simulated pushes happen periodically.

- **Logic Server and Platform -** This two way communication happens over TCP and is through RESTful APIs. The communication from Logic server to Platform is on demand whereas the one in reverse direction on request as well as of the callback form. Callback means notifying the logic server when some condition is met at the Filter server for the current batch of data.

- **Application Administrator Interface and Platform -** This communication happens happens over TCP. This too uses RESTful APIs.

3.10 Filter Server

This is the contact point of the platform to the application tier. Filter server runs Node.is and uses Mongo DB to store the incoming stream of data from the Gateways.

It exposes RESTful APIs to the application tier and that enables the heterogeneity of the applications. Filter server refers to the registry server to get the list of sensors whose data is requested in the query. Once it has the list it adds the corresponding command that is to be sent to gateways in a periodic manner when requested.

Only the registered gateways should be able to talk to Filter server. This is made sure using a secure token that is issued to Gateway at the time of

its boot up by the Security Server.

3.11 Logic Server

This is the contact point of the end user mobile application with the platform. This runs Node.js. All the end user mobile applications communicate with the platform through this server only.

This talks with the Filter server on receiving requests from the end user application and in turn talks with Filter server to get the results for a certain query. On receiving response to its queries the logic server returns it to the requesting mobile application.

3.12 Interactions Between Modules

3.12.1 Gateways and Filter Server Interaction

Gateways send execute the commands sent by the Filter server. These commands actually are not immediately sent by the Filter server to Gateways but are en-queued at its end. Gateway periodically pulls these commands and returns the results for them. To make sure that only the registered gateways talk with the filter server security tokens are used.

3.12.2 Filter Server and Logic Server Interaction

For the logic server the filter server is point of contact with the platform. Filter server answers the data queries of the logic server and also notifies whenever a certain threshold is reached. This notification can in turn be

interpreted by the logic server to trigger something on the userâĂŹs mobile application.

3.12.3 Application Administrator Interface and Repository Server

Application administrator, after authenticating him/herself, can register any new devices with the platform through the interface provided to him/her. This interface is directly coupled with the Repository server. This communication happens as simple client server communication.

3.13 Wire And File Formats

- **Wire Format -** All the communications but the one from Devices to the Gateways use JSON as the data format. The latter uses the format that is natural to the devices transmitting.
- **File Format -** Repository and Registry files are stored in JSON format as flat files.

3.14 User's View

This UI is in fact the one decided by the application developer using this platform.

3.14.1 Smart Garden

Let's consider a Smart Garden solution using our platform. There will be lux meters measuring the intensity of light falling on the plants. And when

the light is too to bright, user gets a notification on his mobile application and he/she in turn can push a button to pull down a curtain that prevents the light from directly falling on the plant.

3.14.2 Smart Garden Application UI

On the mobile application user can request the light intensity of certain lux meter(s) in the garden by their location or even all of them by clicking on the "Check light intensity" button. On successful retrieval of the data from the Filter server user can browse through the data.

He/she will get an alert whenever a certain rule is matched on the Filter server to let the user know that the light is too bright. He/she needs to push a button that pulls down the curtains automatically.

Chapter 4

User's view of system

4.1 Application

An application is a software program designed to meet user's needs and requirements.An app may belong to various categories like banking, business, manufacturing or medical categories.In our case, app will give the processed information based on one of the category mentioned above. The app platform we are developing, helps developer quickly develop and deploy app.

4.2 Files involved

Main files involved are sensor registry, sensor config dictionary and application rules dictionary.

```
{
    "SensorOneID" : {
        "app" : "App1",
        "family" : "typeX"
    },
    "SensorTwoID" : {
        "app" : "App2",
        "family" : "typeY"
    },
    "SensorThreeID" : {
        "app" : "App2",
        "family" : "typeX"
    },
    ...
    ...

    // new entry goes here
}
```

Figure 4.1: *Snapshot of the registry file at any given time*

4.3

4.3.1 Sensor configuration Dictionary

Sensor configuration Dictionary holds the specifications of the data generation attributes of sensors. They include

- Protocol the sensor uses to communicate with gateways. It can be one of BLE, Blue tooth, WiFi, etc.
- Payload the sensor generates. For each data key its length and any additional setting can be provided.
- Payload length in bytes.
- Interval at which the sensor generates and pushes the payload. The interval is in Milli seconds.

- Family the sensor belongs to.
- Application the sensor is a part of.

```
{
    "appOne_FamilyOne" : {
        "payloadLength" : "someInteger",
        "protocol" : "XXX",
        "pushInterval" : "someInteger",
        "payload": {
            "dataKey1" : {
                "length" : "someInteger",
                // less than or equal to payloadLength
            }
        }
    },
    ...,
    ...,
    // new entry goes here.
}
```

Figure 4.2: *Snapshot of the sensor configuration file*

4.3.2 Callback API

Logic server before performs some checks on the request from App-tier and then registers a callback with FILTER SERVER through an API to get the required data when the rules are met. Filter server then pushes the data in the following scenario

- Minimum threshold of the physical quantity measured by that sensor family is reached
- Maximum threshold of the physical quantity measured by that sensor family is reached
- The time interval in seconds registered by logic server is reached

4.4 Analytic Rules

Analytic Rules are defined by summarizing the log file details captured over time. These rules are an important part of logic server.

4.5 Structure of the files

All these files are maintained in JSON format. JSON(JavaScript Object Notation) is a text based data format which uses key-value pairs to structure data.

4.6 Deployment and Setup

The module is deployed in cloud and accessed through HTTP Rest API's.The module setup is done using information from the configuration files.

4.7 Module Start and Stop

The module consists of REST APIs on top of node.js platform which uses MongoDB internally for scalable and flexible Database performance. The module comes into action when the application is first launched, and stopped only when the application is stopped end to end. During its functioning, its monitored for two sub-processes.

4.8 Access/Invoke

Gateway module invokes sensor details using Rest API's and similarly Application Tier module invokes details from filter server module using Rest API's.

User interactions

i. How will the user use this system

ii. What and how to configure (processes, config files)

iii. What and how to deploy

Chapter 5

Key Data structures

5.1 Definition and other details of device types

The platform consists of following device types

- **Sensor:** This part of platform generates data which is processed by the gateway. Examples of types of sensors supported are - Temperature, Location, Humidity etc

    a. Temperature sensor: Collects temperature from specific locations for a given time frame.

    b. Location sensor: Detects location as per the app requirements and forwards data to gateway

- **Gateway:** This part of platform receives data from sensors connected to it. It processes the data as per the application requirements and forwards data to server. The data can either be dumped at this stage or can be handled by the filter server.

- **Filter Server:** This part filters the data received from gateway as per the requirements of the underlying app.

5.2 Wire formats

- This represents the data - type received/sent from the gateway.
- The data generated from sensor is dumped to the MongoDb powered database.
- The data is received in form of JSON.
- The server (filter server) queries database is javascript and receives JS objects.

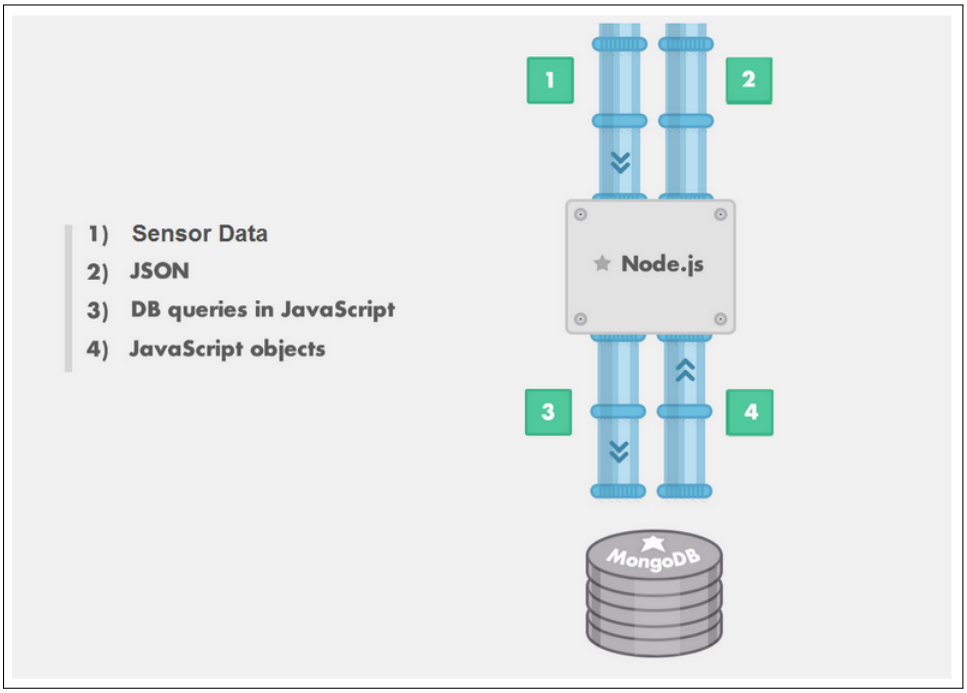

Figure 5.1: *Communication of Filter Server with other system components*

5.3 Persisted data (registry, rules, et al)

Registry

- This file holds status of all the sensors registered with the platform. It is dynamic file and data is valid only for specific time frame.
- Each entry associates status of the sensor registered in the following format { Sensor family, Time interval, status }

Repository

- As the name suggests this file holds all the sensors types supported by the platform. It is a static collection in form of key-value pairs of sensor id - sensor type registered with the platform
- Repository entry describes key attributes of the sensor viz ID, application which it is a part of, family that it belongs to. An entry should be added to this file for every new sensor being plugged in to the platform.
- The incoming data from only the registered sensors are allowed to make it past the Gateways. Any number of sensors belonging to the same or different family/ies can be registered with the platform.

5.4 APIs - interaction data objects

- The sensor generated data is received by gateway and forwarded to server in form of **JSON object**

- The data saved to DB is retrieved by shooting queries in javascript. The data received is again a **Javascript Object**

Chapter 6

User Interactions

- User will launch the app and as he/she interacts with it, according to the app logic a RESTful service is called with the required parameters sent in a JSON format.
- Filter server processes the data received as to deduce the requested action before executing the target service.
- Service generates the appropriate response and sends it back to the requesting app.

6.1 Module to Module Interactions

- **End users** interact with the system through the apps running on hand-held devices.
- **Rules Server:** This is the server which will be responsible to process all the users requests. It will interact with other subsystems to achieve desired results, such as in case of Smart Parking System it will interact with the systems like, Location and Maps etc. It is also responsible

for processing data from various gateways and then storing it to the Sensory Data Repository , processing of data includes Unmarshalling of data received from the gateways , and applying some checks on data and retrieving information from raw data received.

- **Sensory data Repository:** It has the information related to various parking lots, which is used by Rules Server to answer the queries generated by the controller of App Server. This repository is updated by the rules server periodically.

- **Gateways:** These are responsible for collecting data from various sensors and forwarding it to the rules server. There can be any number of gateways between the sensors and the rules server, they are basically used to forward the sensory data to the rules server.

6.2 File/Wire Format Definitions

Sensor Packet: Packets Transmitted by sensors.

- SensorID: It's a unique ID given to each Sensor.

- Type: It describes the type of data that sensor senses in our case it is Heat, Proximity and motion sensor.

- Data: It signifies the data associated with the sensor, like its position , or data generated by it.

- Status: It will show whether the corresponding parking spot is free or reserved.

- Timestamp: It will contain the time at which the packet is generated.
- CRC: It will contain checksum for error detection.

User Packet: It will be generated by the user, when he/she launches the application.

- DeviceID: Its a unique ID of the user.
- Type: It will show the type of packet, ie. whether its request or response packet.
- Data: It will contain the data like users location from where it has generated the request, or other parameters needed by the Rules server.
- Timestamp: It will contain the time at which the packet is generated.
- CRC: It will contain checksum for error detection.

Chapter 7

Persistence of the Platform

7.1 Configuration Information what needs to be saved

- Configurations of the sensors (Location , type , id , data format etc)
- Details of the registered mobile apps
- The history and information of the data received from various sensors at various point of times (to be stored in database attached with filter server)

7.2 Transient state (services location status , et al)

- (Application Tier , Filter server) and (Gateways and sensors) will be located in remote locations communicating through some media.
- A sensor will be in active state when it gets registered with the filter server (unless that it will be in inactive state)
- An application can send requests only when it gets registered with the platform (until then its requests will be overlooked by the application

tier considering them as invalid)