ConnectX IoT platform

Skyline labs

www.skylinelabs.in

Abstract

Internet of Things is a fast developing concept - collaboration of various Computer science fields resulting to Internet connected and intelligent devices. The Internet connectivity allows devices to act as source of data - in order to monitor and predict their performance, and remote control of devices. Current projection for IoT markets suggests that IoT devices will be the largest data creators and Internet consumers by 2025. The project (ConnectX platform) is an end to end IoT platform ready to meet the growing amount of data and security threats. The platform enables users to deploy complete IoT projects, with no coding, all within minutes. The system deployment is completely automated, easily scalable and has easy integration support for any 3rd party APIs. The platform has real time data analytics running to analyze the data streams. This platform is a big data system, and it supports infinite horizontal scalability as it is based on clustered distributed back-end providing best in class latency, throughput, and response parameters. Connect-X : the platform developed has a graphical interface where users can design their IoT systems using drag-drop/drop-down lists based tools. The platform will then auto-generate codes for hardware, IoT hubs and cloud; and deploy them on the servers. The key feature for this platform is that its every component ensures scalability and security. It supports every IoT data communication protocols available (MQTT, CoAP, DDS, HTTP2.0, XMPP). The real time data analytics provides easy interfacing with 3rd party databases and services thus enabling users to monitor their IoT devices and trigger actions they want when certain conditions are set. Light-weight, secure and reliable M2M communication support is an inbuilt feature of the platform.

This platform enables users to easily create web-dashboards, android applications to monitor their IoT enabled devices. Through the web-UI, users can decide Rules for the data streaming to the cloud. These rules will enable real time analysis of data streaming, to perform actions as decided by the users. Connect-X enables a code-less deployment of IoT systems. The users need not code any part of the back-end, and every component is auto-configured to scale up as per incoming data traffic. IoT platforms are an upcoming technology concept with the current market leaders having launched their platforms not more than 5 months ago.

Connect-X platform already meets the standards and benchmarks of industry leader IoT platforms like ThingsWorks, IBM IoT, AWS IoT, etc. With additional features incorporated in the platform, it has a potential to be the leading IoT platform in the industry.

As the demonstration of the capabilities of the platform, the project includes a home-automation use case based on Connect-X platform. Users can monitor the state of electronic devices in their home and control them via an android application. We also implemented a contextual personal assistant (Chat bot) to control various devices at home. Users can automate the home applications by commanding the chat-bot to set time and conditions based triggers like Start AC on 8am or

Start AC when temperature is greater than 40 degree all via voice commands.

Thus the project is a complete end to end implementation of an IoT Platform using Open Source Technologies allowing the users of the Platform to deploy IoT Solutions easily.

Contents

List of Figures

Chapter 1

Introduction

IoT or Internet of things is a trending technology paradigm that puts computing powers and connectivity into all sorts of non-living appliances. The internet connectivity available with such appliances enables remote control and monitoring leading to better operation and reduced maintenance costs. IoT devices typically have a low power low footprint processor that is responsible for sending and receiving control data from the servers. Since the onset of Internet of Things, devices have become more intelligent thanks to cloud based and edge data analytics. This has lead to a new era of Machine learning and artificial intelligence that operates on the data captured by Internet enabled devices to make decisions critical to device operation and human life. Typically, IoT systems involve following components:

1. Hardware appliance : Existing or new systems that perform tasks of controlling actions or collection of data (Example Windmills, Cars, etc).

2. Sensors : The data collection modules that capture Digital/Analog data on the Hardware appliances (Example Temperature sensor, pressure sensor, gas sensor, etc).

3. Microprocessors : The control units on the hardware appliances that are responsible to handle data coming from the sensors, analysis of data, internet communications with the servers (Example x86 processors, ARM, etc).

4. IoT hubs : The internet connection points connected to multiple microprocessors locally and to the cloud via internet. As the data handed by such hubs is larger than that handled by microprocessors, the IoT hubs are typically stronger processors with higher computing powers, more RAM and memory. IoT hubs are optional components. The individual microprocessors may be internet enabled, removing the need of IoT hubs.

5. Communication protocols : The IoT hubs or the microprocessors have to send data to the servers in real time. Internet connectivity is costly for most IoT applications as the devices are in remote areas with minimal connectivity. Also, the huge amount of data being streamed

to the servers makes the total internet consumption huge. Hence, there are a special set of protocols implemented in IoT application that ensure light weight, secure and reliable data transfer to the servers. Protocols like MQTT, XMPP, DDS, CoAP, HTTP2.0 are examples of such IoT friendly protocols.

6. Message brokers : Millions of IoT devices stream huge amounts of data to servers every second. Hence, the servers must have a scalable message (data) broker (handler) that can reliably handle the data and send it to its intended service be it data analytics backend, databases or other IoT devices in-case of M2M communication. Message brokers also act as interface between dat coming from multiple protocols. All the devices connecting to cloud may not support a particular protocol. Hence, there needs to be an agent that aggregates data coming from multiple protocols Examples of such message brokers include RabbitMQ, eMQTTD, Apache Kafka, etc.

7. Databases : The data coming from IoT devices is tremendous. The databases storing this data hence need to be scalable, ready to handle big data and they should be reliable to ensure that they dont lose any data and fetch it in minimum time when some application queries the data. Examples of such database is Apache Casandra.

8. Analytics engine : Real time analytics engine is a basis for Rules engines of IoT solutions. Rules engines are responsible for the data pipeline and analysis function where data coming from the IoT devices is analysed, filtered and forwarded to appropriate end applications. The data analysis should be real time, so as to ensure that the data triggers actions in real time, so as to support time critical applications. The analytics engine should be able to handle big data operations. Examples of frameworks for such Analytics engines are Apache Storm, Apache Spark streaming, Samza, etc.

9. End application interfaces : The IoT solutions systems should be able to trigger alerts, notifications, and control actions based on the data captured by the devices. Such end applications include sending data to custom databases, Emails, SMS, Smartphone application notifications, triggering some other IoT device, calling Web-APIs and web-hooks, etc.

10. Security : The IoT devices collect and stream data that is confidential and critical in many cases. There must be encryption and credential mapping in IoT systems to ensure that the data remains safe in the backend.

11. Access control : IoT devices often are located in remote locations, where they can be stolen. Such stolen devices can lead to breach of server IP and credentials. A rouge device may try to mask itself as some other device and send malicious data to servers. Hence, there needs

to be control on the type and kind of data the device sends to the servers. The cloud should be able to recognize malicious data coming from devices and immediately deny connection to that device to ensure security.

12. Real time data visualization : The IoT solutions contain a visual interface that show graphically the data captured from various devices. The connection status of all the devices, status of the systems should all be available on such interfaces.

Chapter 2

Problem Statement

2.1 Need of an IoT Platform

The components mentioned in the previous section are more or less the same for all IoT systems. Every time a new IoT solution is deployed, the software stack constructed is very similar in nature to that of other IoT solutions. This means that deploying a new IoT system often is a repetitive work, where same software is developed again and again resulting to waste of time and money. There needs to be a way that avoids these unnecessary repetitive actions, a way that will lead to quick deployment of systems at minimum investment. Internet of Things platform is a concept that solves all these problems. IoT platform is still in the Innovation trigger stage of the Gartner hype cycle, which makes it technology that will become mainstream after 10 years. IoT platform is a concept where users can deploy end to end scalable IoT systems without coding, removing the repetitive nature involved in deploying IoT solutions. An IoT platform enables non-programmers to develop IoT solutions within minutes, without know how the backend works. The entire system is configurable through simple UI based dashboards, with SDKs for hardware handling the hardware side code. In short, IoT platforms AUTOOMATE creating IoT solutions. Deploying IoT solutions becomes a task involving 3 simple steps:

1. Register an IoT device through the dashboard
2. Configure the Rules, actions to be triggered, data pipeline all through the UI.
3. Download the SDKs for the hardware in use, put the code in the hardware

The system will come online once the hardware connects to internet! Thus IoT platform provides the following benefits:

- Easy, quick deployment of IoT systems.
- No need to code the system.

- All features including security, scalability, data pipeline, etc are auto-configured.
- Minimum costs involved as software is already developed for any kind of IoT system.
- Supports all hardware, communication protocols, databases and end applications.

2.2 Objective

Develop a system that enables users to develop IoT systems through a UI based dashboard with following features:

1. Easy and quick deployment of IoT solutions
2. Eliminating need to code while deploying IoT
3. Support to all popular hardware platforms
4. Support to all IoT communication protocols
5. Scalable message broker
6. Real time data analytics Rules engine to analyse, filter data and trigger actions
7. Security and access control for devices registered
8. Devices monitoring
9. End applications interfaces
10. Scalable database
11. Data visualization
12. M2M support

Chapter 3

Proposed Solution

To implement the IoT platform with features mentioned in the previous sections, the solution to be implemented requires a big data system architecture.

The key of the approach to ensure dynamic configurations is to maintain the metadata and users configurations in a database, which will later be fetched by all the big data components to perform actions accordingly. The web-dashboard will save data to the configuration database - which will then be fetched by the backend when data arrives. To support multiple hardware platforms, the protocols implemented for data transfer should have hardware client support for most the hardware. This can be achieved by choosing an application layer protocol over TCP, one thats not complex and still retains features like being light weight, secure and ensures reliable data transfer with high quality of service.

The IoT platform should have one protocol inbuilt, but should support multiple protocols so as to ensure that the platform works well with most of the devices and protocols. Hence the message broker should somehow interface with multiple protocols so that data coming from all protocols reaches the server. To ensure a secure data transfer, the protocol implemented should be modified to support SSL, encryption along filtering algorithm to enable access control.

Thus, the security features should be implemented at the protocol level. There should be 2 message brokers One for the protocol implemented, and one to aggregate data coming from multiple protocol brokers. Both these brokers need to be scalable, which can be achieved by a distributed implementation. The broker of the chosen protocol needs to be light weight and needs to support modifications to add security features. The protocol aggregator broker should have better scalability than other broker, as it has to handle more data. This broker will take the data from multiple protocols and send it to the analytics platform.

The analytics platform used should be real time or near real time to ensure that the input data triggers actions in real time. There are 2 approaches for real time analysis

- To perform action for every incoming data stream

- To perform action in windows of some time frame (Analyse data every 1 millisecond)

The 2nd method is a better approach as the IoT applications usually dont need response time of less than a few milliseconds. Also, performing analysis in batches will result to less load on the servers. Once the message broker sends data to the analytics engine, the engine will find in the configuration database if any action is to be performed on the individual messages; if there is an action, the message will be filtered and the end action will be triggered. As most of the analytics platforms are a part of the Apache family, the best choice for the databases used is an Apache family database like Cassandra which can work well with HDFS compatible application. To trigger the end applications, the analytics engine will need the clients of the end applications. For example, if Apache Spark streaming is used, the end applications can be triggered by having clients of Email, SMS, etc in Scala (The language used in Spark streaming applications).

M2M communication can be enabled by implementing a pub-sub protocol in first place or by having clients of the protocols in the analytics engine.

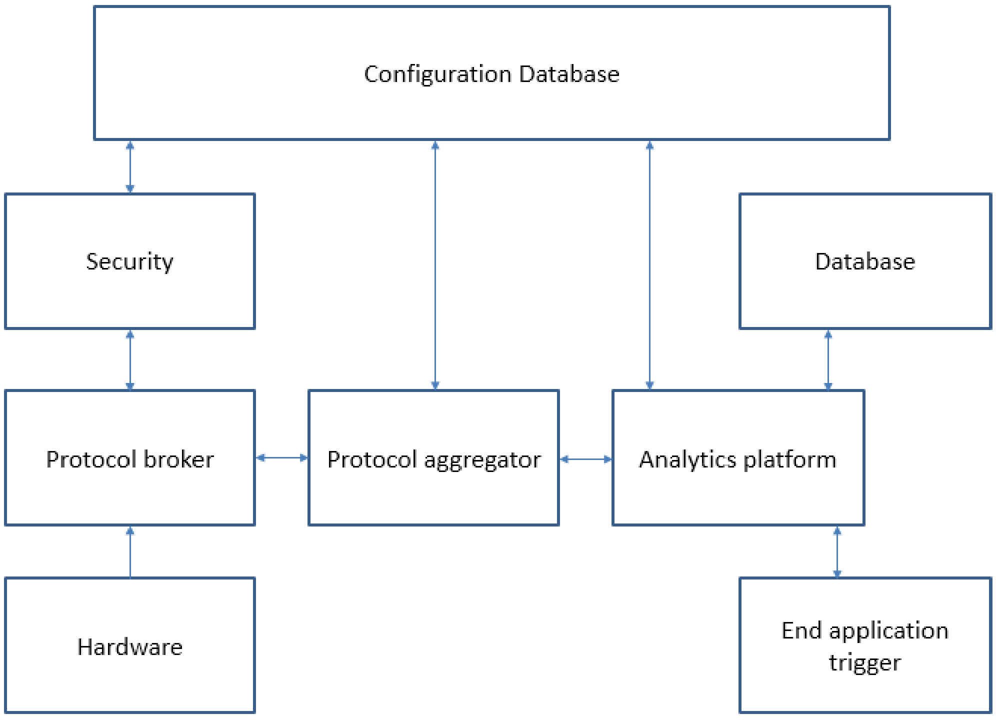


Figure 3.1: Proposed Solution Overview

Chapter 4

Literature Survey

4.1 IoT Platforms

There are IoT platforms existing in the market by Cloud providers like AWS [1], Azure [2], IBM Watson [3], etc. The primary features lacking in these platforms are the following

- Lack of multi-protocol support.
- Lack of open architecture implementation (Complete system is black box, making adding new components difficult.
- Limited number of supported hardware platforms.
- Lack of appropriate end application interfaces.
- Lack of a complex rules engine support.

4.2 Architecture Components

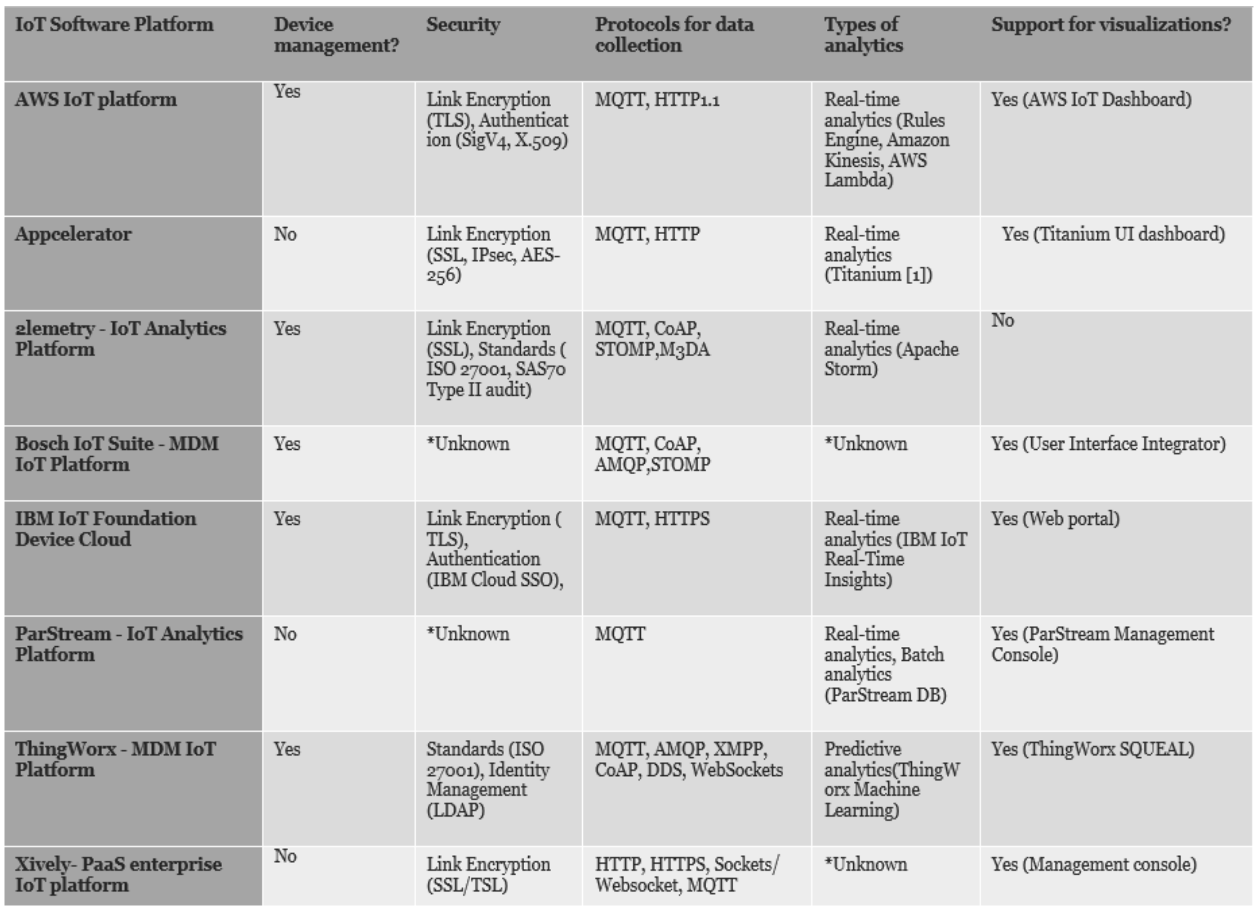
We studied the components that can be used for fulfilling the objectives laid out in the previous Chapter. After studying them, we chose the best ones that met all requirements.

4.2.1 Hardware platforms support

To ensure that the platform developed is compatible with most of the hardware platforms available, the system developed should support multiple protocols so that the hardware platforms can choose one from the multiple protocols as per what the hardware supports.

4.2.2 IOT Protocols

There are many IoT enabled open source protocols like MQTT [4], MQTT-SN, AMQP [5], DDS [6], and XMPP [7]. There are many scalable and distributed implementations of MQTT protocol



IoT Software Platform	Device management?	Security	Protocols for data collection	Types of analytics	Support for visualizations?
AWS IoT platform	Yes	Link Encryption (TLS), Authenticat ion (SigV4, X.509)	MQTT, HTTP1.1	Real-time analytics (Rules Engine, Amazon Kinesis, AWS Lambda)	Yes (AWS IoT Dashboard)
Appcelerator	No	Link Encryption (SSL, IPsec, AES-256)	MQTT, HTTP	Real-time analytics (Titanium [1])	Yes (Titanium UI dashboard)
2lemetry - IoT Analytics Platform	Yes	Link Encryption (SSL), Standards (ISO 27001, SAS70 Type II audit)	MQTT, CoAP, STOMP,M3DA	Real-time analytics (Apache Storm)	No
Bosch IoT Suite - MDM IoT Platform	Yes	*Unknown	MQTT, CoAP, AMQP,STOMP	*Unknown	Yes (User Interface Integrator)
IBM IoT Foundation Device Cloud	Yes	Link Encryption (TLS), Authentication (IBM Cloud SSO),	MQTT, HTTPS	Real-time analytics (IBM IoT Real-Time Insights)	Yes (Web portal)
ParStream - IoT Analytics Platform	No	*Unknown	MQTT	Real-time analytics, Batch analytics (ParStream DB)	Yes (ParStream Management Console)
ThingWorx - MDM IoT Platform	Yes	Standards (ISO 27001), Identity Management (LDAP)	MQTT, AMQP, XMPP, CoAP, DDS, WebSockets	Predictive analytics(ThingW orx Machine Learning)	Yes (ThingWorx SQUEAL)
Xively- PaaS enterprise IoT platform	No	Link Encryption (SSL/TSL)	HTTP, HTTPS, Sockets/ Websocket, MQTT	*Unknown	Yes (Management console)

Figure 4.1: Platform comparison

like eMQTT. MQTT open-source components are plenty- Mosquito MQTT broker and Eclipse Paho client [8]. The communication protocols are characterized by low bandwidth requirement, performance even under unreliable and unstable networks with security and encryption. MQTT protocol is a light weight, low foot-print protocol based on TCP for publish/subscribe message based communication. XMPP is a protocol based on TCP and is used for point to point message exchange. AMQP is REST based protocol running over UDP

4.2.3 MQTT brokers

MQTT protocol is the protocol used in Connect-X platform. The reason for the same is mentioned in the next sections. The MQTT protocol has multiple open-source brokers available like Mosquito MQTT [9], Hive MQTT [10], Rabbit MQ with MQTT extension [11], eMQTT (An Erlang based MQTT broker) [12]. Out of these, eMQTT and Hive MQTT are scalable. The scalability of MQTT broker is achieved by implementing a distributed back-end system.

4.2.4 Protocol aggregation broker

To support data coming from protocols other than MQTT, there is a need to have another messaging broker. There are multiple options available to implement such messaging brokers. These brokers need not be based on light weight protocols unlike the broker interfacing with IoT devices.

Most commonly used messaging brokers are the following:

- Apache Kafka [13] is a Apache family system developed by LinkedIn
- AMQP protocol [14] is a cloud pipeline protocol used in Azure IoT platform

4.2.5 Analytics engine

There are 2 types of Analytics platforms available:

- Real time analytics
- Batch analytics

The IoT solutions need data analytics in real time which makes it essential to have real time analytics platforms in the architecture implemented. There are many real time, near real time analytics platforms available like:

- Apache Storm [15]
- Apache Spark Streaming [16]
- Apache Samza [17]

Apache Spark streaming is a real time analysis platform using a distributed file system for data storage. Storm is based on a high level abstraction system called Trident where as Samza uses its own key-value store. Spark is a near real time analysis platform where data is analyzed in timed windows. Apache Storm and Samza are real time analysis platforms where there are stream consumers consuming real time data coming from various sources.

4.2.6 Databases

There are multiple options available for data storage. IoT applications need unstructured data storage as the structure of data coming from multiple devices are not of similar formats. NoSQL databases have different data storage paradigms like key-value storage, document storage, wide column storage, etc. There are multiple options available for unstructured data storage like:

- MongoDB [18]
- Cassandra [19]
- Oracle NoSQL [20]
- HBase [21]

4.2.7 Web-dashboard framework

To develop a real time interactive dashboard, there are multiple frameworks available for developing the web-application. The following are the most widely used frameworks for real time web-dashbards:

- MEAN Stack (MongoDB, NodeJS, AngularJS, ExpressJS) [22]
- Ruby on rails [23]
- MeteorJS [24]
- PHP [25]

Chapter 5

Design of Architecture

5.1 IOT Protocol

MQTT (Message Queue Telemetry Transport)

MQTT is a connectivity protocol for machine-to-machine communication. It has been designed in such a way that it extremely lightweight when it comes to publish and subscribe messaging transport. It has an extremely small code footprint and network bandwidth. Because of its small size and minimum data packets, it is widely used in Internet of Things.

MQTT consists of three components: the subscriber, the publisher and the broker.

1. Subscriber: Those devices interested in receiving data register themselves to the broker by subscribing to topics of interest. This means that whenever the broker receives a message with a particular topic, it will send this message to all those who have subscribed to that topic.

2. Publisher: The publisher generates data that needs to be sent. It sends this data via a topic to the broker.

3. Broker: There broker is a server that contains topics. Each publisher sends information on a specific topic and the subscriber receives automatic messages each time there is a new update in the topic he has subscribed to.

Why to choose MQTT:

1. Publish/subscribe mechanism: Protocols using this mechanism meet the IoT requirements be than request/response since clients do not have to request updates. When the request factor is gone, the network bandwidth used decreases.

2. Lower message processing: The messages do not need to be responded to. Thus the reduction in message processing increases the lifetime of battery run devices.

3. Security: MQTT brokers can be implemented to require authentication via TLS/SSL

4. Quality of Service Reliability is ensured in MQTT through the option of three QoS levels

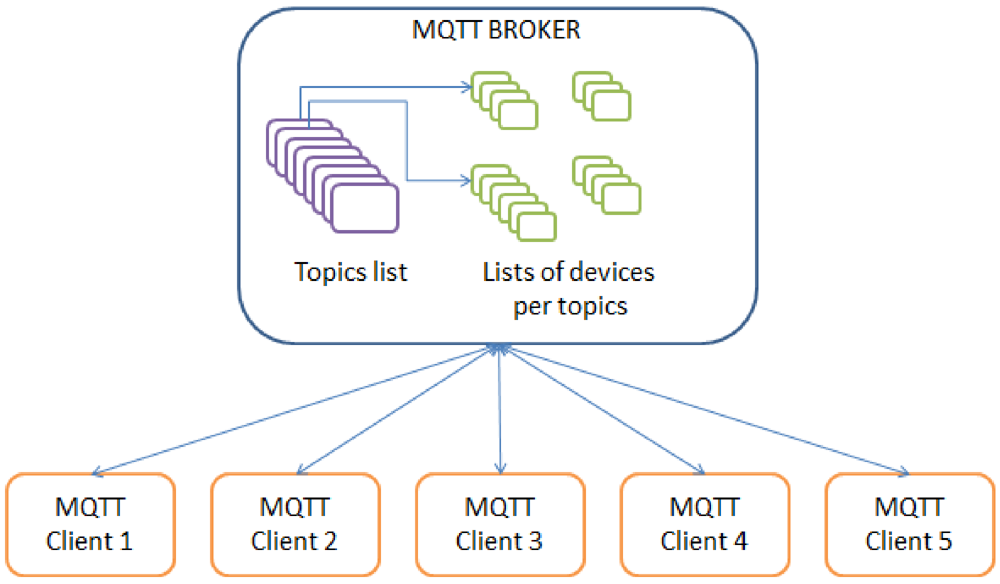


Figure 5.1: MQTT Broker and Clients

- Fire and Forget : Messages are sent only once and no acknowledge is sent back
- Delivered at least once : There is guarantee that the message has been delivered. An acknowledgement is sent
- Delivered exactly once : It is made sure that the message is delivered exactly once by a four-way handshake mechanism

5. Although COAP (UDP based) has lower overhead than MQTT (TCP based), packet loss is less likely to happen in MQTT. MQTT also has lower delays than COAP

6. More efficient: MQTT, which is used by Facebook messenger, has been proved to be more efficient than HTTP for battery run devices.

7. Clustering: It is possible to cluster MQTT brokers for achieving scalability. The clustered MQTT brokers act together as one broker. Every broker in cluster maintains the topics list. When one broker is unavailable, other broker handles the request. This way, the brokers balance within themselves. This clustering ensures that the system is highly available. Even if one server goes down, other servers will be there in its place to handle the requests and data.

8. Ease of implementation The MQTT protocol is one the most documented and community support protocols, as its used in many of the industry applications and IoT systems. This makes development of new features on MQTT an easy task.

5.2 Multiple hardware support

MQTT protocol implemented has Eclipse Paho as its client library. Eclipse Paho is a Eclipse foundation project to make MQTT clients available on all hardware platforms. The years of development and research by Eclipse Paho community has made Paho work on most hardware systems. Using Paho in Connect-X platform SDKs has made sure that the platform supports most of the major hardware platforms.

5.3 MQTT broker

We chose the EMQ Broker as the main extensible MQTT message broker. It is Open Source and licensed under the Apache Version 2.0. It is written in Erlang and it implements MQTT version 3.1.1.

Why EMQ broker?

1. It is enterprise grade
2. It is massively scalable and supports C 1000K MQTT Connections.
3. It is possible to cluster and bridge brokers on distributed nodes.
4. It is highly extensible : The code has a provision of ”hooks” which are callback functions and can be easily modified by us. Due to the presence of these hooks, it is possible to write a number of plugins to extend the functionality of the EMQ broker. As part of the project, we wrote a EMqttd-Kafka Plugin using the ”hook” functions.

5.4 Protocol Aggregating Broker

The protocol aggregating broker used in Connect-X IoT platform is Apache Kafka. The reason for using Kafka is that it is highly scalable due to its distributed functioning, at it is the most supported input for all 3 real time analytics platforms (Spark, Storm and Samza). Also, Kafka is a widely supported Hadoop/Spark family component which ensures high community support for developing more features.

5.5 Analytics Engine

Connect-X platform uses Apache Spark Streaming for it analytics engine. The following are the reasons for the choosing Spark Streaming:

- Time Windows based near real time analytics in Spark streaming ensures lesser server resources utilization. IoT devices stream data where response time can be in seconds, latency of upto 1 second is usually fine for IoT systems, making near time analysis a better choice than real time analytics

- Open-source, free for commercial use, huge community support.

- Based on Scala, a JAVA based language with all major end applications supported.

- High scalability, reliability and availability - proven in multiple applications in the industry.

5.6 Database

The database used in Connect-X is MongoDB. Though Cassandra, a Spark family component was better choice, MongoDB was chosen due to limited server resources availability. The project has been deployed on EC2 free tier server which has 1 GB RAM. Kafka, Spark streaming and Cassandra are all based on JVM, which made it impossible to run all 3 applications on a server with constrained resources, without affecting the performance. Though the platform uses MongoDB right now, using Cassandra is the first step involved in future work before deploying the system for scalable use cases.

5.7 Web-dashboard

Connect-X dashboard is a MEAN stack application. Decision of using MEAN stack was mainly driven by the availability of MQTT client for JavaScript and not for Ruby and Rails. This MQTT client is used to display real time data being streamed from the devices to the servers. MeteorJS is a weaker framework compared to MEAN stack, providing lesser features and server access.

Chapter 6

Implementation of Proposed Architecture

Now we look at the actual implementation of the Architecture proposed and how the components introduced in the previous Chapter have been interfaced with each other.

We have deployed the said architecture on Amazon Web Services EC2 Linux instance.

6.1 MQTT protocol broker

We deployed the Open Source EMQ Broker on AWS EC2 in cluster mode. The broker implements MQTT version 3.1.1. The publish and subscribe mechanism of MQTT is as follows and explained in Figure 6.1:

1. Clients(devices) interested in a particular Topic subscribe to that Topic
2. A publisher sends a message with a Topic
3. The broker receives the message and sends to it all those clients who have subscribed to that topic.

We wrote the following plugins in Erlang for enhancing the capabilities of the broker and for meeting the requirements of ConnectX.

6.1.1 Security

Authentication is an important aspect of any IoT solution. There should be a check on the devices that try to connect with the Broker. Any unregistered device should not be allowed to connect and send messages. This is implemented using username and password. When the 'Thing' will be created through the web dashboard, the username given to it will be - ProjectName/ThingType/ThingName and a password will be set by the user. The username and password

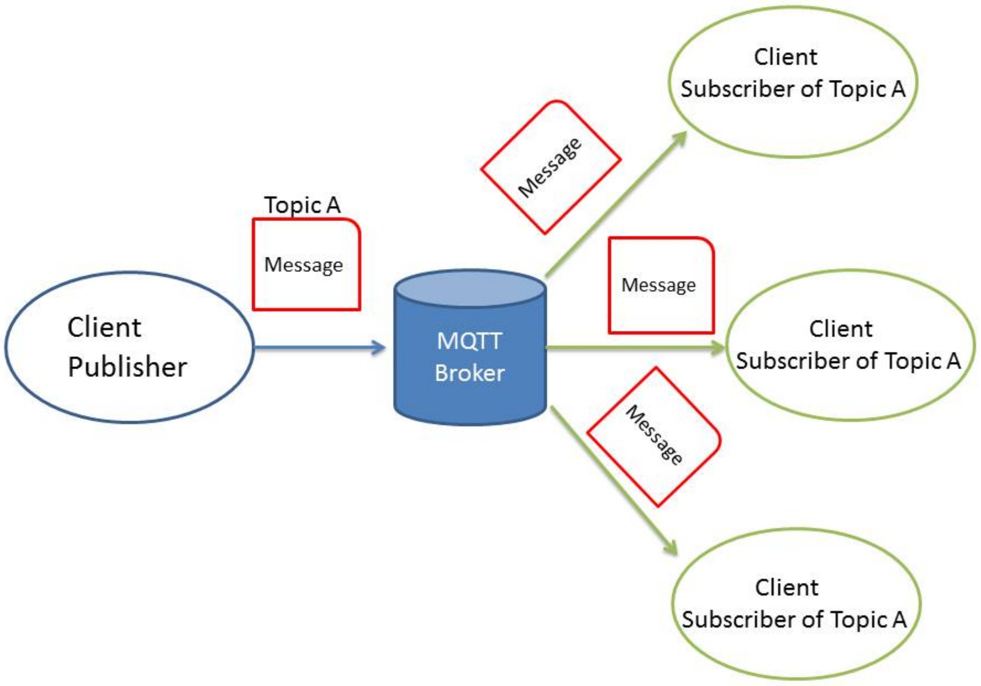


Figure 6.1: MQTT Publish and Subscribe

will then be stored in the MongoDB authentication database. Each time a client tries to connect with the broker, these credentials will be checked by querying the database. If a match is found, the client will be allowed to connect. The MongoDB document will be of the form -

{
username: "username",
password: "password hash SHA256",
superuser: false
}

6.1.2 Access Control

The clients once connected, should not be allowed to publish or subscribe to all topics. There should be a restriction on the topics for pub-sub. If there isn't a restriction, any client can send data pretending to be another client. We have defined a list of topics that a client once registered from the Web dashboard can publish and subscribe to :

1. $state/ProjectName/ThingType/ThingName/update
2. $state/ProjectName/ThingType/ThingName/response
3. $state/ProjectName/ThingType/ThingName/request

4. $iot/ProjectName/ThingType/ThingName/+

5. $alert/ProjectName/ThingType/ThingName

The mongoDB document will be -

```
{
username: "username",
pubsub: ["topic1", "topic2", ...]
}
```

Each time a client tries to publish/subscribe to a topic, the authentication database will be queried to find whether the client is allowed to do so.

6.1.3 Device Connection State

An important requirement of all IoT solutions is the ability to monitor the connection status of devices. Sometimes devices break-down, power-down or lose internet connectivity. It is important to know at a given time how many devices are connected even if they are not sending data. For this, we wrote a custom plugin for the EMQ broker which changes the connected status of a device when it connects/disconnects with the broker. EMQ provides "hooks" are callback functions which will be run by the broker when a client is connected/disconnected, a topic subscribed/unsubscribed or when a message is published/delivered. So in the hook for connect(), write "connected":"yes" against the device in the database and in the hook for disconnect(), write "connected":"no" against the device in the database. Hence, whenever a device is online,its status will be accurately stored in the database.

6.1.4 Kafka Bridge

The MQTT messages received by the broker need to be sent to Kafka. For this, we wrote another custom plugin to the EMQ Broker. We implemented a Kafka client in Erlang in the EMQ broker itself. This client implements a Producer as per the 0.8 Kafka protocol and can produce to a topic ansynchronously. In the hook of the publish() function, all messages received by the broker are sent to Kafka via the Kafka producer implemented as part of the broker. The MQTT messages received are sent to kafka regardless of their topic.

6.2 Apache Kafka

The IoT devices can communicate with the server via multiple protocols like MQTT, CoAP etc. so multiple brokers for these protocols can be added. All of these brokers will stream data to Apache

Kafka. The reasons for using Apache Kafka have already been highlighted in the previous Chapter. Thus, every protocol broker will have a Kafka producer that will publish every message that it gets to the Kafka cluster. The MQTT-Kafka bridge as shown in the diagram takes any message received by the broker and publishes it to Kafka. It appends the original MQTT topic with the original MQTT message and produces this new message with topic "kafka". Similarly there will multiple bridges like these for CoAP and other protocols.

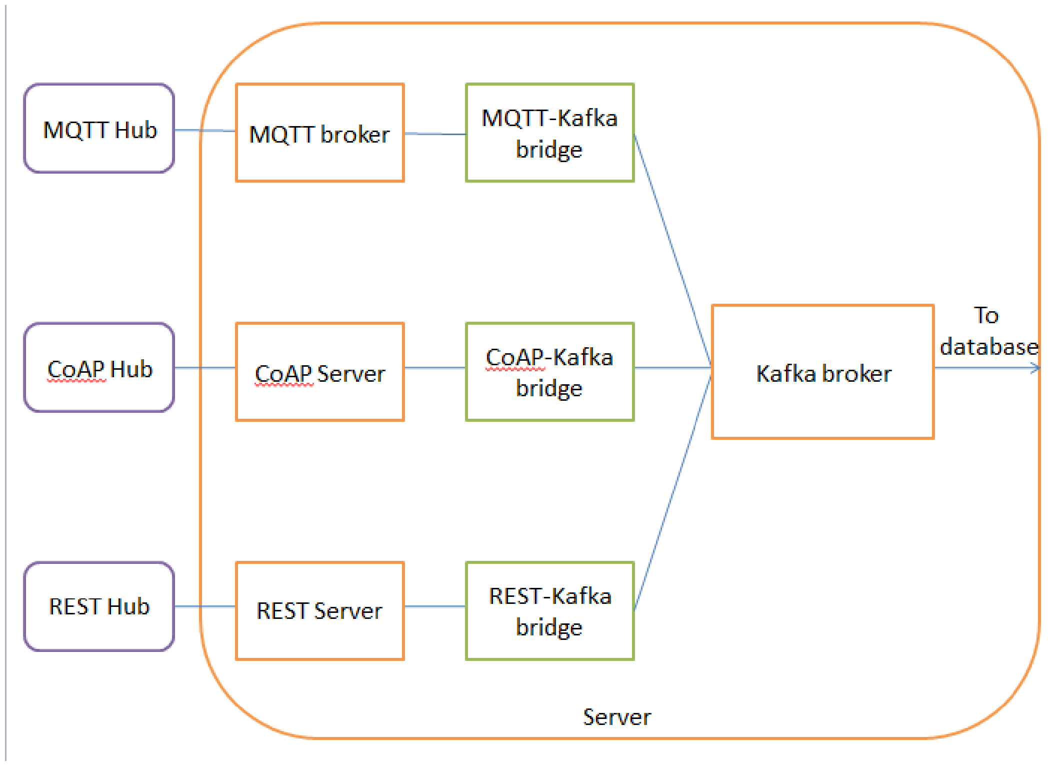


Figure 6.2: Kafka Protocol Aggregator

6.3 Apache Spark

Apache Spark is a big data analytics platform, known to be 100x better than Apache Hadoop. Spark is based on a HDFS (Highly distribute file system) just like Hadoop, but has a better analytics capability due to it's RDD (resilient distributed database) approach for analytics. Spark is big data ready, and can handle any amount of data due to it's distributed computing architecture. Connect-X Iot platform uses Apache Spark streaming - a real time data analytics component of Apache Spark. The analytics is near real time - there are time based windows where data being streamed to Spark is gathered in windows of 1 second (This is the value set for Connect-X, but the window size can be decreased to milliseconds or increased to minutes). Spark streaming has Kafka input streams which query data from Kafka every 1 second to get the data streamed by IoT devices.

Spark is the core of the functioning of Connect-X as its responsible for the data filtering, pipeline and end application interfacing. Spark is responsible for the following features of Connect-X:

6.3.1 Device registration

On the dashboard, when a user first signs in to Connect-X, he has to create a new project. The project name per user has to be unique. Lets consider an example of home-automation where project name is 'HomeAutomation'. This project is saved in the projects DB of the configuration Database server (MongoDB). Then, the user has to add a 'thing type' in that project. This is a abstraction given for ease of use to the users. Thing type can be based on category of the thing like 'fan', 'light' or 'bedroom', 'livingroom' based on the room the devices are in. On adding the thing types, the user can add 'things' to the project. While adding thing, the user has to select which thing type the device belongs to and then give a name to that particular thing (Which has to be unique per thing type). The thing type and thing names are stored in Configuration DB.

Thus, after registering a project, thing type and thing name; the thing gets an unique id of the format : 'ProjectName/ThingType/ThingName'. For example : "HomeAutomation/bedroom/fan1"

Along with thing name, the users have to give a password for that thing, which is then hashed and stored in the MQTT security server's database. Similarly, the user also has to set if the device is an admin device or not. Admin device gets rights to publish/subscribe by any topic, where as non-admin can pub-sub by only a certain set of topics which will be explained in the following paragraphs.

6.3.2 State database

The State Db or the device shadow feature is the feature that maintains the live 'State' i.e. the condition of the device on the servers. A state is an inherent attribute of a particular device. For example, State for a fan will have 2 attributes - Power, Speed. Connect-X uses JSONs for communication and BSONs (MongoDB) for data storage. Thus, for the example of Fan, the sample state data in state DB will be of the following format:

```
{
"power":"ON",
"speed":5
}
```

The state DB implementation ensures that all end applications controlling the IoT device remain synced with the device state. When ever a device state change is requested, a MQTT message with topic format

"$state/ProjectName/ThingType/ThingName/update" is published. This message can be triggered only by a thing registered as an admin (Thing is basically a MQTT client and can be android application/website, etc) or non-admin with device ID matching the particular

"ProjectName/ThingType/ThingName"

This message travels through MQTT broker, Kafka and reaches Spark streaming. When Spark streaming encounters a state update message, it immediately updates the State DB, 'upserting' (Update else insert) the state data for that particular device.

When such update message is received by Spark streaming, it also publishes a MQTT message with topic

"$state/ProjectName/ThingType/ThingName/response"

All the devices intending to know the state of a device have to subscribe to the above topic. This ensures that if any end application updates the state of a thing, other controlling applications also reflect the change. The IoT device subscribes to the response message. Whenever it receives a response message, it updates its state accordingly. When the IoT device or an end application starts for first time, it can query the state DD data by publishing an empty message with topic

"$state/ProjectName/ThingType/ThingName/request"

Spark streaming on receiving this message fetches the state of that particular device (Th request topic contains the device ID corresponding to the request) and publishes the state data with state response topic.

This mechanism ensures synchronized behavior of IoT devices and controlling applications and a recovery mechanism in case of failure.

6.3.3 Rules engine

The rules engine is the data filtering and analysis part of the Connect-X platform. The user adds rules through the web-dashboard which are stored in the configuration server's Rules DB. The MQTT devices access control allows them to send data only be a particular topics. While adding rules for a thing, the user has to select data from which topic is to be analyzed. After choosing the topic, the user has to set condition on the analyzed data (For example : temperature >20, where temperature is a parameter in the JSON sent by the device).

On setting the conditions, the user can select what part of the JSON message is to be sent to the end application (For example, the user can set to select temperature, pressure from a JSON containing 10 parameters).

Once this selection and filtering is done, the user then has to select what action to perform when the condition is met. Connect-X currently supports Emails, publishing message back to Connect-X, triggering WebAPI calls and saving data to custom DBs. Publishing messages back to Connect-X

allows data pipeline where data from one device can control state of another device (By having publish rule where published data is by state update topic).
Example for a rule : Topic = "$iot/HomeAutomation/bedroom/sensors2"
Condition = temperature >100
Data Captured = temperature
Action = Email
Subject = Fire in bedroom!
To = myemailID@gmail.com
Content = Emergency, there is a fire

When a new data window arrives at Spark streaming engine, it checks the topics of every message in that window one by one. Then it fetches from Rule DB to check if those topics have a rule to be executed. If rule is found, the rule is executed and this is done for every single message coming from IoT devices. This happens every single second for all the data windows.

6.3.4 Alerts

Alerts are special messages sent by the IoT devices with topic format :
"$alert/ProjectName/ThingType/Thingname"

All the alert messages are by default stored into the Alert DB by Spark streaming engine, and Email is sent to the registered email Id of the users. Alerts can always be seen under the 'Alerts' section of the dashboard.

6.3.5 Data buckets

Data-buckets are a way to store data on Connect-X servers. The working of data-buckets is similar to that of Custom DBs in the rules engine. The only difference is that the database where the data is saved is on Connect-X server.

6.4 MongoDB Schema

There are seven databases which are used for ConnectX platform. All of them are listed below in alphabetical order.

6.4.1 mqtt

This DB is used for authentication purpose, when a new thing is added, it's sha-256 encrypted password and admin status is stored in this database.
DB contains two collections 'mqtt_acl' and 'mqtt_user'

If the thing is admin then following entry goes into ’mqtt_acl’ collection

```
{
”_id” : ObjectId(”58dffa0dd3f4172ed28a8fe6”),
”username” : ”SmartWarehouse/temp sensor/Sensor1”,
”pubsub” : [
”#”
]
}
```

If it is not an admin the following entry goes into the collection

```
{
”_id” : ObjectId(”58ea2b6a75ce6679444b8940”),
”username” : ”try2/fan/u8”,
”pubsub” : [
”$iot/try2/fan/u8/*”,
”$state/try2/fan/u8/update”,
”$state/try2/fan/u8/response”,
”$state/try2/fan/u8/request”,
”$alert/try2/fan/u8”
]
}
```

In the second collection,’mqtt_user’ depending upon the admin status, ’is_superuser’ field changes.

```
{
”_id” : ObjectId(”58abf5d9ba47d071204b7021”),
”username” : ”jay/fans/fan1”,
”password” :
”d74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a725000feb82e8f1”,
”is_superuser” : false
}
```

6.4.2 projects

In this database there is only one collection by the same name and it contains multiple documents with a field ’project_name’.

```
{
```

”_id” : ObjectId(”58eb28eb9edaac15d9203c0f”),
”project_name” : ”Alerto”,
”count” : 3,
”online” : 0
}

’count’ specifies the total number of devices added into that project and ’online’ indicates the number of devices which are currently connected to the internet.

6.4.3 rulesDB

When a new rule is created for a particular thing its entry is done in this database. Collections are made by the name of various projects present in the system and each collection have different set of documents corresponding to the rules created in that project.

{
”_id” : ObjectId(”58e6260087cab9582b1342bf”),
”bucket_name” : ”b3”,
”kafka_topic” : ”$iot/SmartWarehouse/light/l1/sdds”,
”action” : ”mongodb”,
”query” : ”SELECT * FROM message”,
”mongoserver” : ”mongodb://35.162.23.96:3000”,
”mongoDB” : ”bucketDB”,
”mongocollection” : ”SmartWarehouse”
}

Fields in the document varies according to the rules as there are multiple options while creating a rule.

6.4.4 rulesTimeDB

Time based rules created for the projects are stored in here. Collections are made according to the hours of the day like zero, one, two, etc. Each of the collection have all the time based rule across all the projects.

{
”_id” : ObjectId(”58eae49ce1ea020a2c5dc389”),
”rule_name” : ”t1”,
”republish_topic” : ”$state/try2/fan/fan6/update”,
”action” : ”mqtt”,
”message” : ”ẗemperaturë:10”

```
}
```

6.4.5 state

Current state of the devices is continuously sent to the server and its latest status is stored in this DB. Collections are made according to the project names.

```
{
”_id” : ”TempSensor/Sensor1”,
”state” : [
{
”temp” : NumberLong(25)
}
],
”timestamp” : NumberLong(1491087599),
”type” : ”TempSensor”
}
```

’State’ field can have any data which will be appropriate for the sending device.

6.4.6 thingsDB

All the devices which are added on the system are logged into this database. Collections are made according to the project names and each collection have the unique id of the device, its type, current connected status and timestamp.

```
{
”_id” : ”android/android1”,
”type” : ”android”,
”connected” : ”0”,
”timestamp” : ”15-04-2017 18:36:43”
}
```

6.4.7 typesDB

Type for a particular project is saved in typesDB database. Collections have project names and contains multiple documents having type name field.

```
{
”_id” : ObjectId(”58eb298b9edaac15d9203c16”),
```

”type_name” : ”android”

}

6.5 Architecture overview

The overall architecture of Connect-X can be seen in diagram 6.2

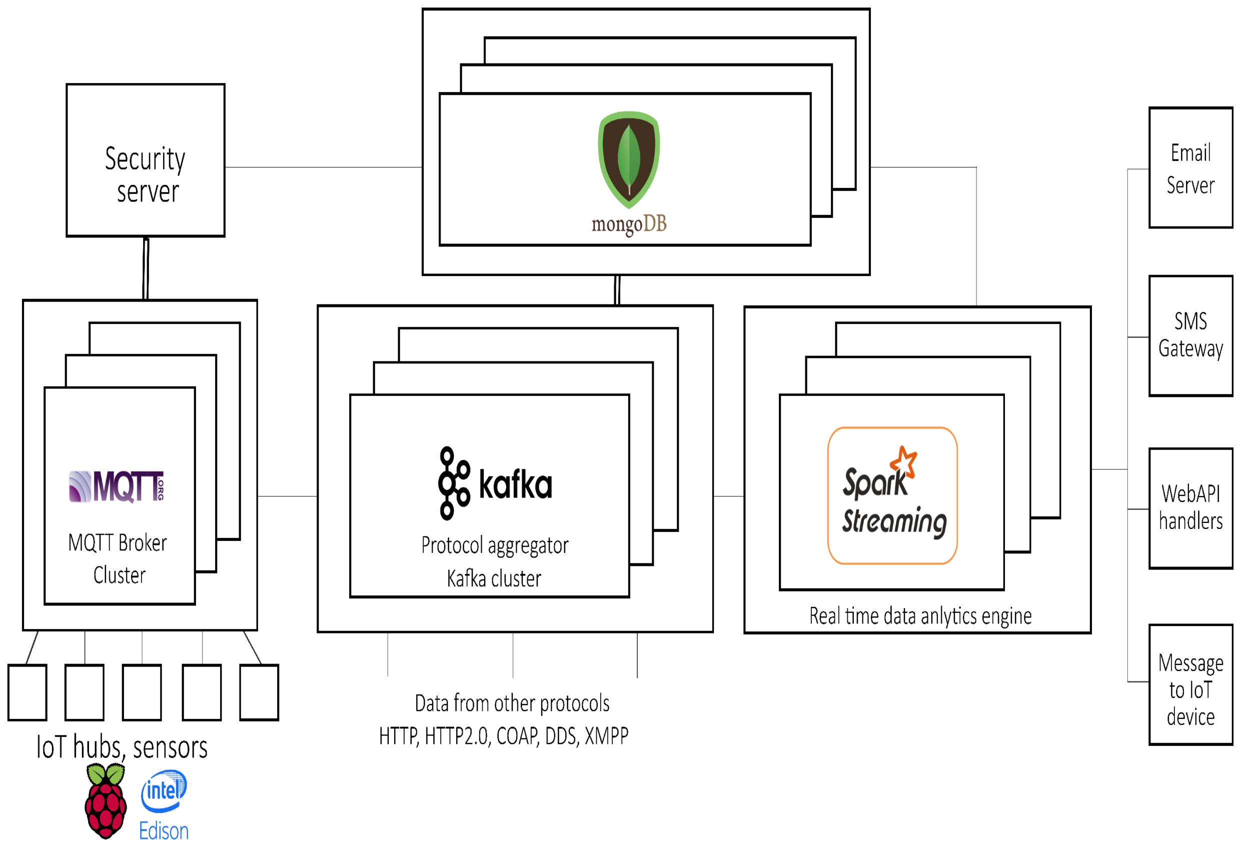


Figure 6.3: Connect-X Architecture

The solution has been deployed on AWS EC2 instance with 1 GB RAM. The platform can be scaled up by adding more nodes to the eMQTT, Kafka and Spark streaming instances. The process of scaling up is easy and an inherent feature of the mentioned components.

Chapter 7

Web Dashboard

7.1 Technology

Web dashboard for connectX has been developed using MEAN stack. Users can control, Manage, Analyse and have an overview of all the devices deployed through the system. MEAN stack is a JavaScript stack mainly used for building web application which are dynamic in nature. MEAN consist of four components MongoDB, ExpressJS, AngularJS and NodeJS. While developing this dashboard AngularJS component is replaced by MaterializeCSS, which is also a framework for creating web app.

Figure 7.1: MEAN stack

- MongoDB It is a NOSQL database supporting Big Data. It has been use by many developers to deploy highly available and scalable web apps.
- ExpressJS Express runs on top of NodeJS, it acts like a rendering engine which is used to display real time and dynamic data on webpages.
- MaterializeCSS It is an open source framework which follows Googles material design rules.

- NodeJS It is the most popular framework when it comes to coding server side using javascript. Many major companies use this for developing strong backend. NodeJS is both vertically and horizontally scalable.

7.2 Advantages of MEAN stack

Developer can code the entire application using only one language, JavaScript. This is the biggest advantage. Data transfer between these frameworks is easy. JSONs can be used all over the application for this purpose. Huge library and community support available. NPM is worlds biggest repository which has many useful addons. Multiple options available for implementing a feature.All the individual components are proven to be the best in their respective domain. MongoDB is build for cloud usage.It can be replicated on various servers easily.NodeJS make asynchronous calls and uses websockets, so client side need not request each time there is change in data.All the components in MEAN are free and open source.

7.3 Setup

MongoDB is pre-installed on Ubuntu machines.
Install NodeJS and NPM using following command.

```
sudo apt-get update

sudo apt-get install nodejs

sudo apt-get install npm
```

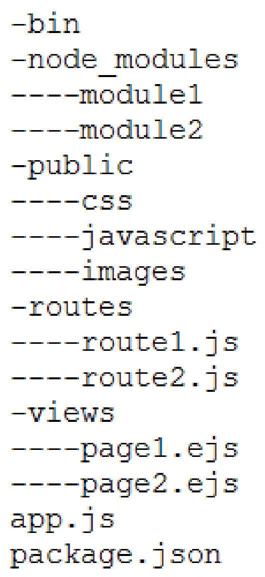
We have used EJS rendering engine for our project. It can be installed globally by following command

```
npm install ejs --save
```

New project can be directly started by

```
express --ejs
```

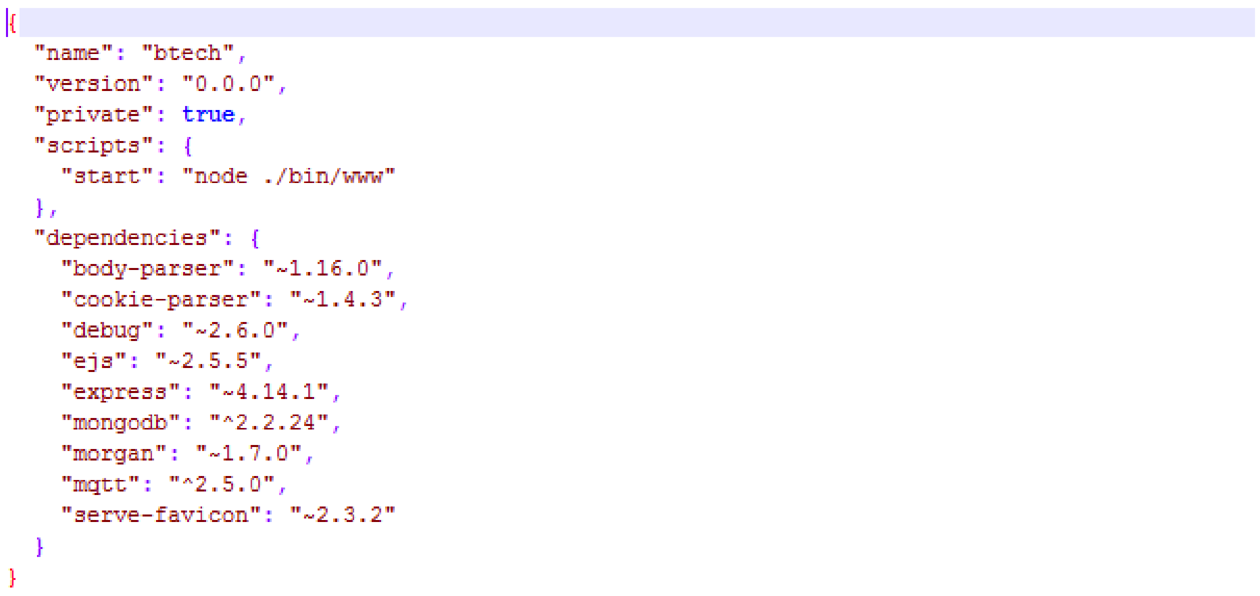
Folder directory of the project is as follows



```
-bin
-node_modules
----module1
----module2
-public
----css
----javascript
----images
-routes
----route1.js
----route2.js
-views
----page1.ejs
----page2.ejs
app.js
package.json
```

Figure 7.2: Folder Structure

Package.json file contains all the dependencies of the project.



```
{
  "name": "btech",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "body-parser": "~1.16.0",
    "cookie-parser": "~1.4.3",
    "debug": "~2.6.0",
    "ejs": "~2.5.5",
    "express": "~4.14.1",
    "mongodb": "^2.2.24",
    "morgan": "~1.7.0",
    "mqtt": "^2.5.0",
    "serve-favicon": "~2.3.2"
  }
}
```

Figure 7.3: Package.json

- bin - It has a www file which runs the server.
- node modules - Contains various modules which can be used in the project
- public - Contains files and folders which are accessible throughout the project for rendering web pages.
- routes - Files which interacts with the database and fetches data for the webpages dynamically.

- views - The look and feel of the webpages, mainly though html5 and the data fetched from the DB.
- App.js - main file for the project which handles all the requests made to the app and redirects the control flow.
- Package.json - contains all dependencies of the project.

Install these dependencies using the command

```
npm install
```

Run the server by executing the following command.

```
node www
```

7.4 Features implemented in the UI

1. Real time Monitoring of device status and state DB content
2. Visualization - Plot graphs on data collected in Databuckets
3. Dashboard Get an overview of all the devices deployed.

Chapter 8

Step-by-Step Deploying an IoT Solution on ConnectX

Deploying any project is made easy by ConnectX. If a user wish to create an IOT solution, following are the steps which he has to follow. All devices can be managed through the dashboard UI without actually coding anything.

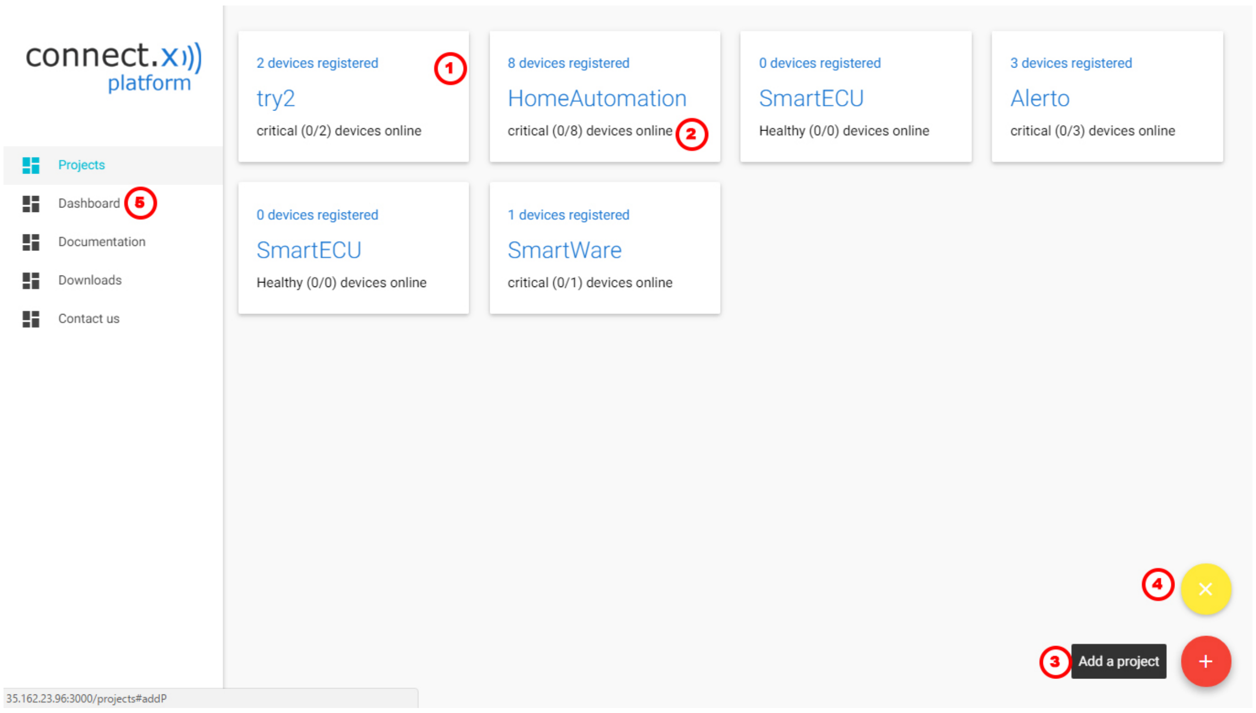


Figure 8.1: Projects page

1. Main page (Figure 8.1) of the platform has all the projects created so far in the form of cards. On every card total number of devices which has been added are shown.

2. Status of all the devices in a particular project can be seen. If all the devices are online then status will be Healthy and if not then Critical.

3. To add any new project just click on Add project button. A pop up box will appear asking user for the name of the project. After adding a new project, it will appear on the page as a new card.
4. To remove any existing project, click on yellow button. Select projects to be removed and click on remove.
5. To get an overview of all the devices and projects click here. It will open a new page(Figure 8.2).

Clicking on any project card will open a things page for that particular project.(Figure 8.3).

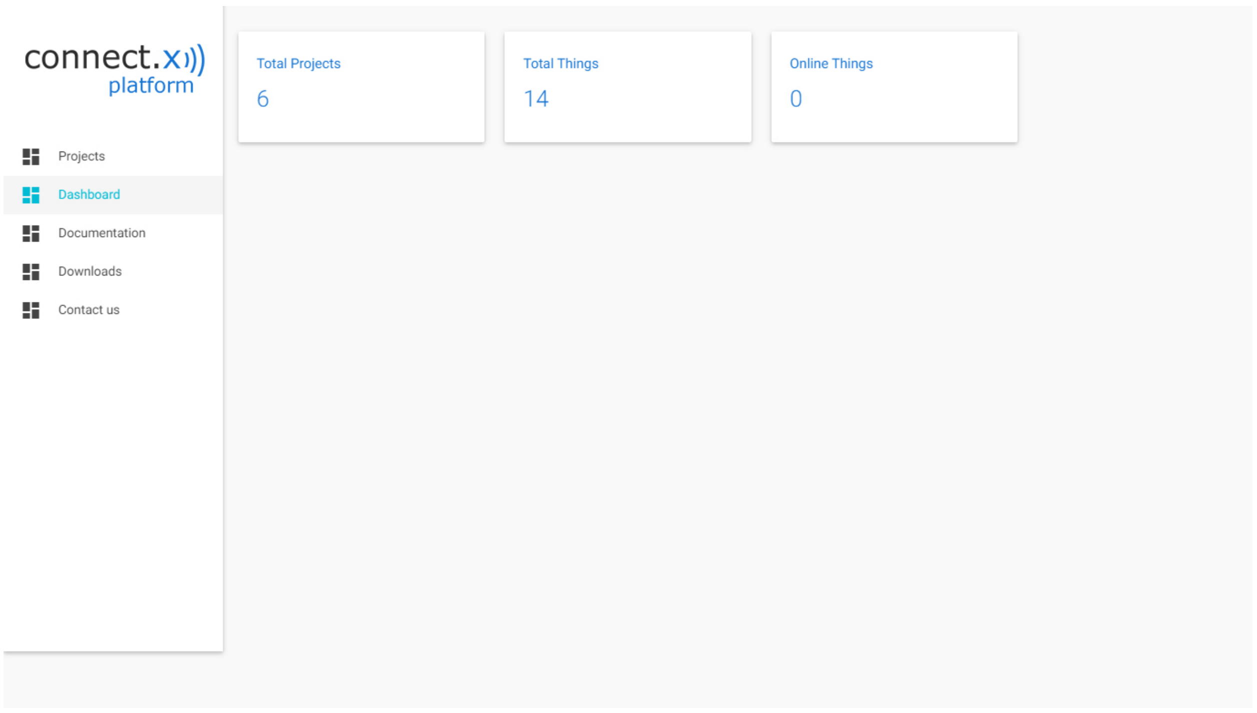


Figure 8.2: Dashboard

This page shows how many projects are present on the system, total number of devices added and how many of them are currently online.

Things page lists down all the devices added in that particular project

6. Current project name.
7. Device card, showing all devices for the current project.
8. Current status of the device. If it is connected to the internet, then connected : yes, otherwise connected : no.
9. To add a new thing(device) , click on Add a thing button, a pop up will come(Figure 8.4) asking for the details.
10. To add a type, click on the blue button.
11. To remove any of the existing thing, click on the red button and select one or multiple devices to be removed.

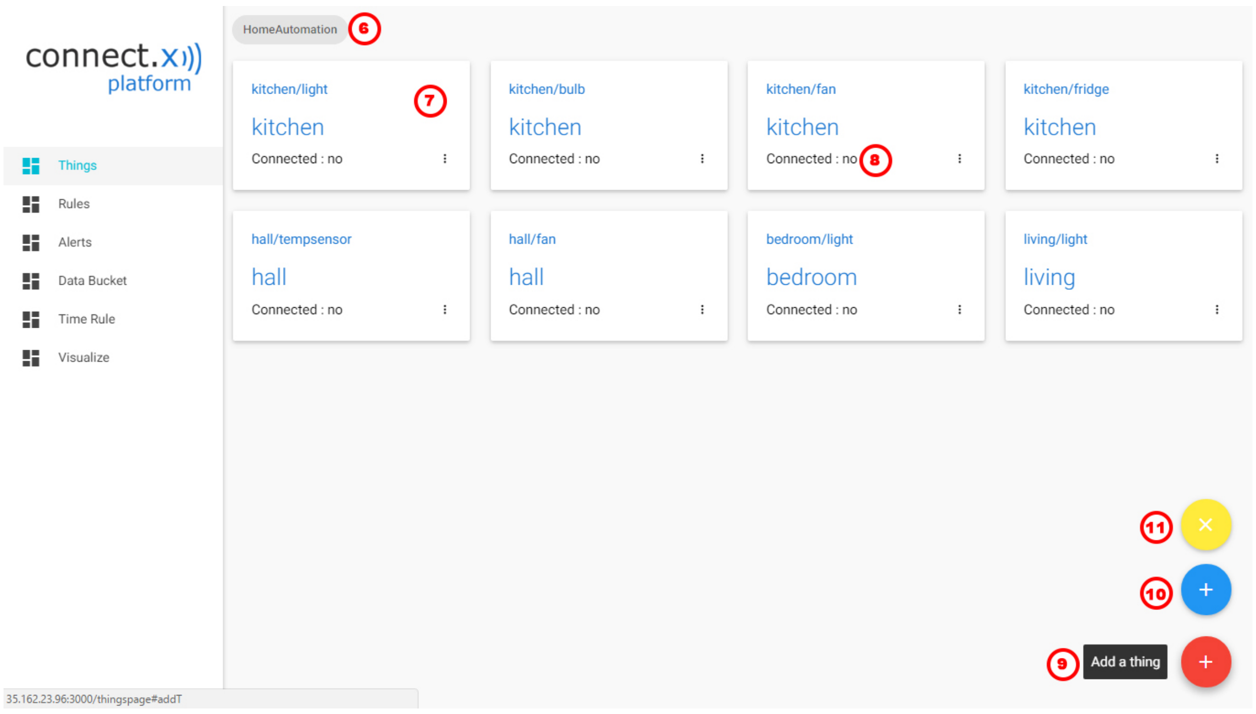


Figure 8.3: Things page

After clicking on Add a thing button this pop comes up.

12. Name of the current Project is already mentioned. Select the type of the device.

13. Give a unique name to the new device.

14. Select whether the device should act as an admin or not.

15. Enter password for authentication purpose.

Click on Add button to put this new device in the project. A new card will be generated on the Things page.

Different types of rules can be created for the data streaming from the devices. To do so, click on the Rules tab.

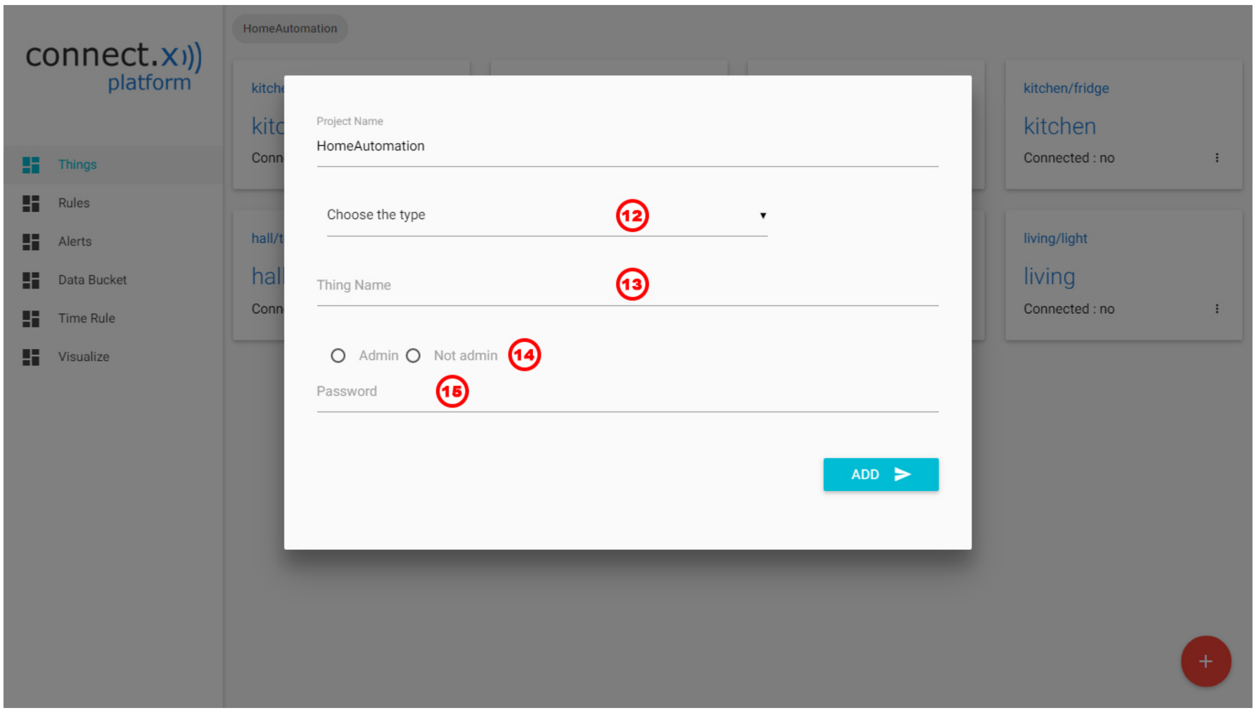
16. All the existing rules are shown here. Click on them to know the details.

17. To add a new rule click on orange button. A pop up window (Figure 8.6) will come up, fill the required details and the new rule will be shown on this page.

After clicking on Add rule button, this window comes up.

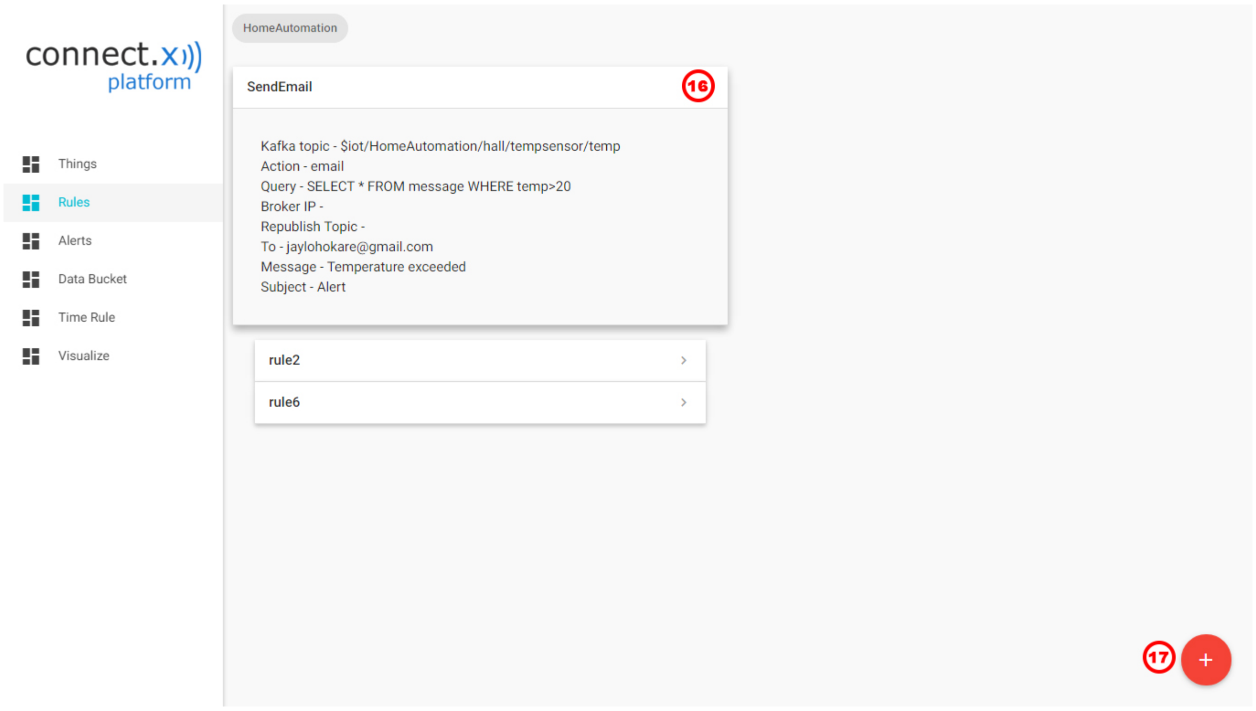
18. Give a name to the rule for identifying it uniquely.

19. Parameter asks for a value which is to be selected and condition specifies when to capture the message. For example, if Parameter : humidity and Condition : temperature ¿ 25, then humidity from all the messages where the temperature is greater than 25 will be captured and specific actions



Project Name: HomeAutomation
Choose the type
Thing Name
Admin / Not admin
Password

Figure 8.4: Add Thing



SendEmail

Kafka topic - $iot/HomeAutomation/hall/tempsensor/temp
Action - email
Query - SELECT * FROM message WHERE temp>20
Broker IP -
Republish Topic -
To - jaylohokare@gmail.com
Message - Temperature exceeded
Subject - Alert

Figure 8.5: Rules page

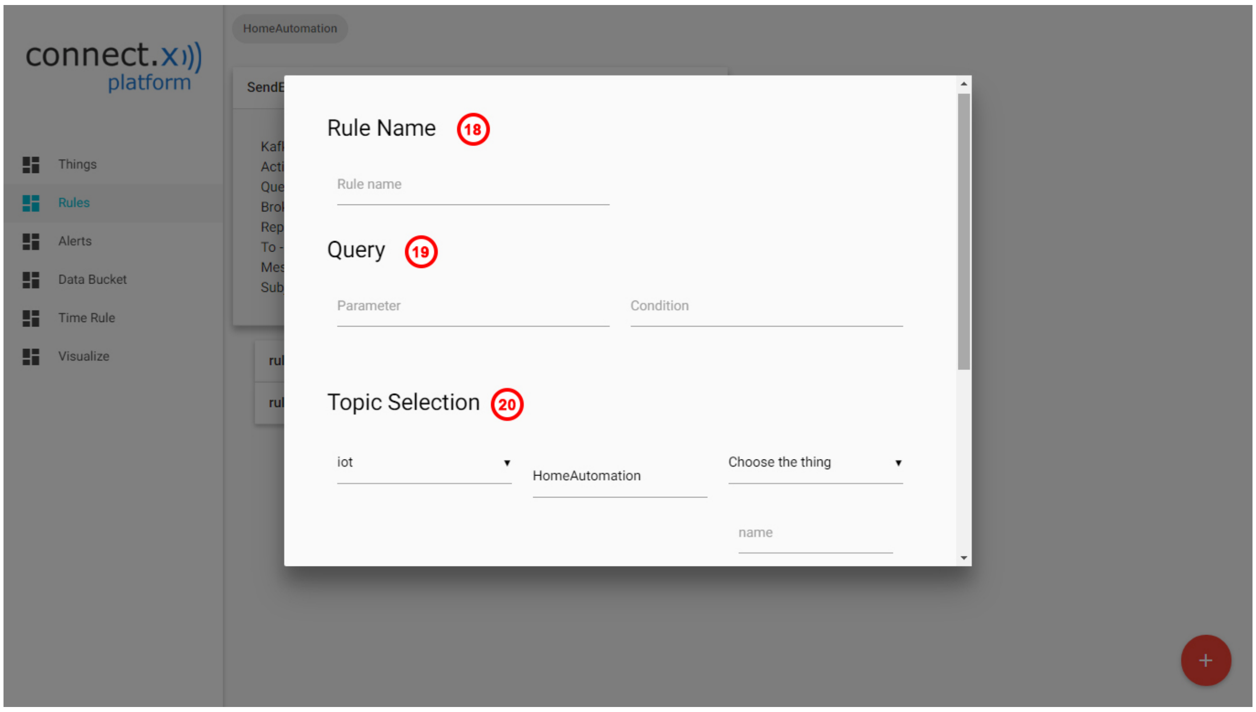


Figure 8.6: Add rule 1

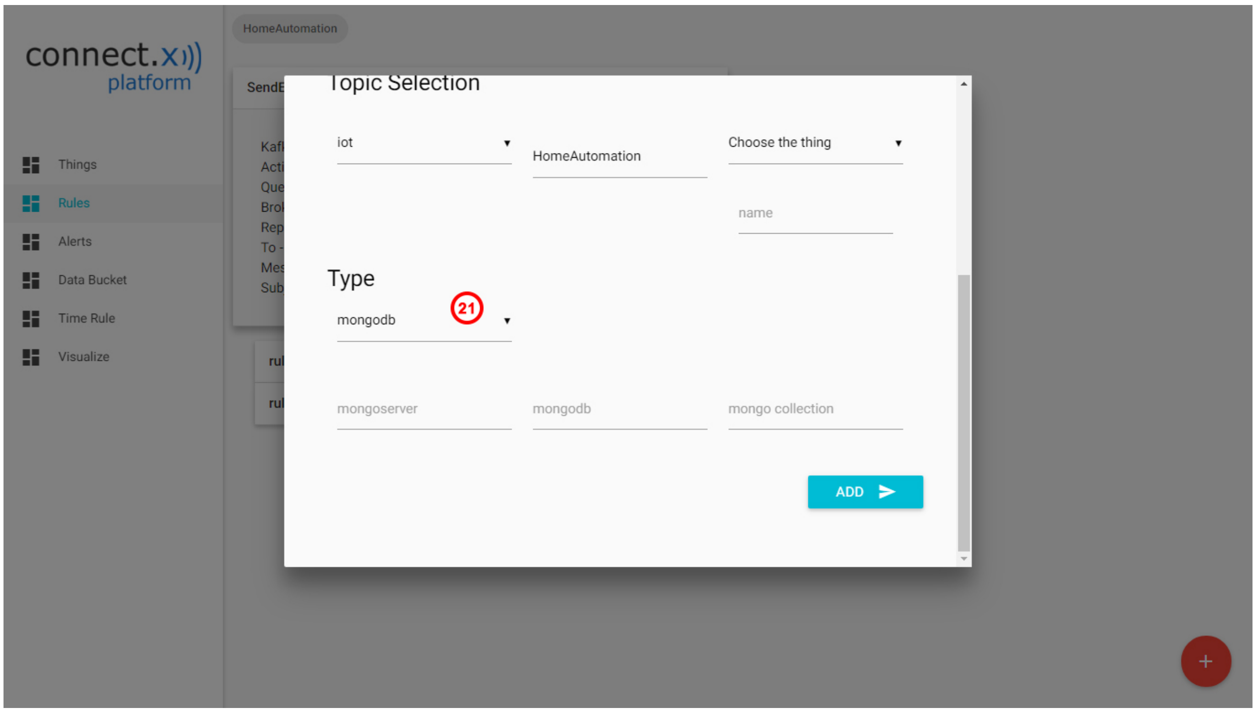
will be taken which user is going to specify next.
20. Topic selection It is the unique identity of the device for which the rule should act. User have to select the device from the dropdown menu, which enlist only registered devices.

21. The action which is to be triggered is selected here. There are multiple options through which user can select one. First is MongoDB. If the messages coming from the devices are to be stored in some DB then this can option can be selected. Three more fields are required in this case, they are server IP, name of database and the collection name.

22. Second type is MQTT. It is selected when device to device communication is needed. User again need to specify which device should be triggered on meeting the previously mentioned condition. The message coming from the first device will be send to this second device.

23. Third type is EMAIL. On meeting the specified condition, an email can be sent. On selecting this option, user needs to specify the receivers email id, the message which needs to send and the subject of the email.

24. Fourth option is Http request. If user wants to trigger some third party API then this can be done using Http option. Specify the key-value pair and add more pairs if needed.

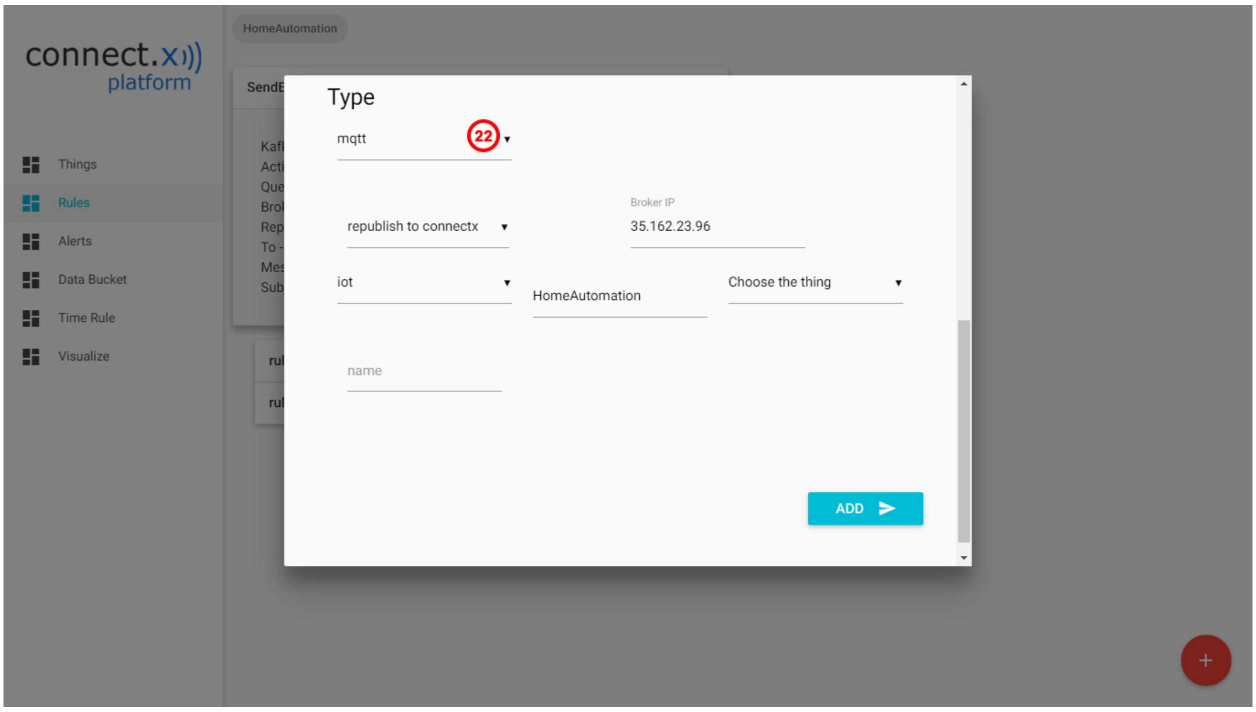


iot ▾ | HomeAutomation | Choose the thing ▾
name

Type: mongodb ▾

mongoserver | mongodb | mongo collection

Figure 8.7: Add rule 2



Type: mqtt ▾

republish to connectx ▾ | Broker IP: 35.162.23.96

iot ▾ | HomeAutomation | Choose the thing ▾

name

Figure 8.8: Add rule 3

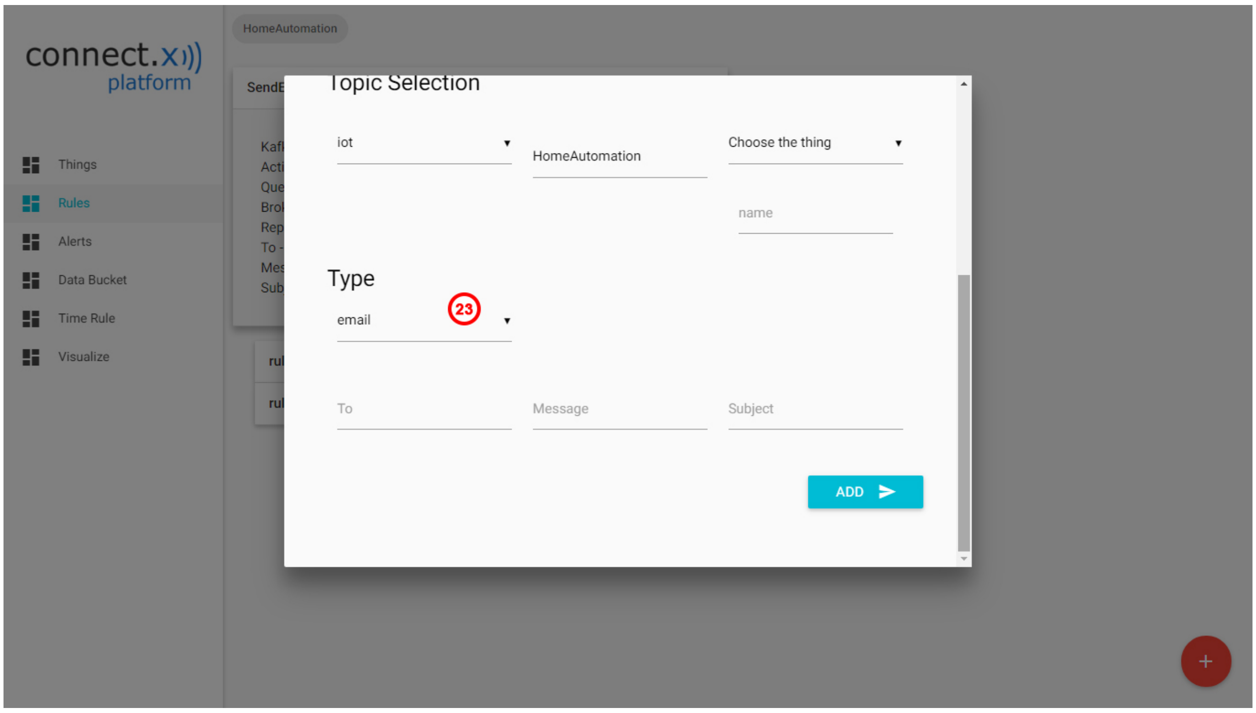


Figure 8.9: Add rule 4

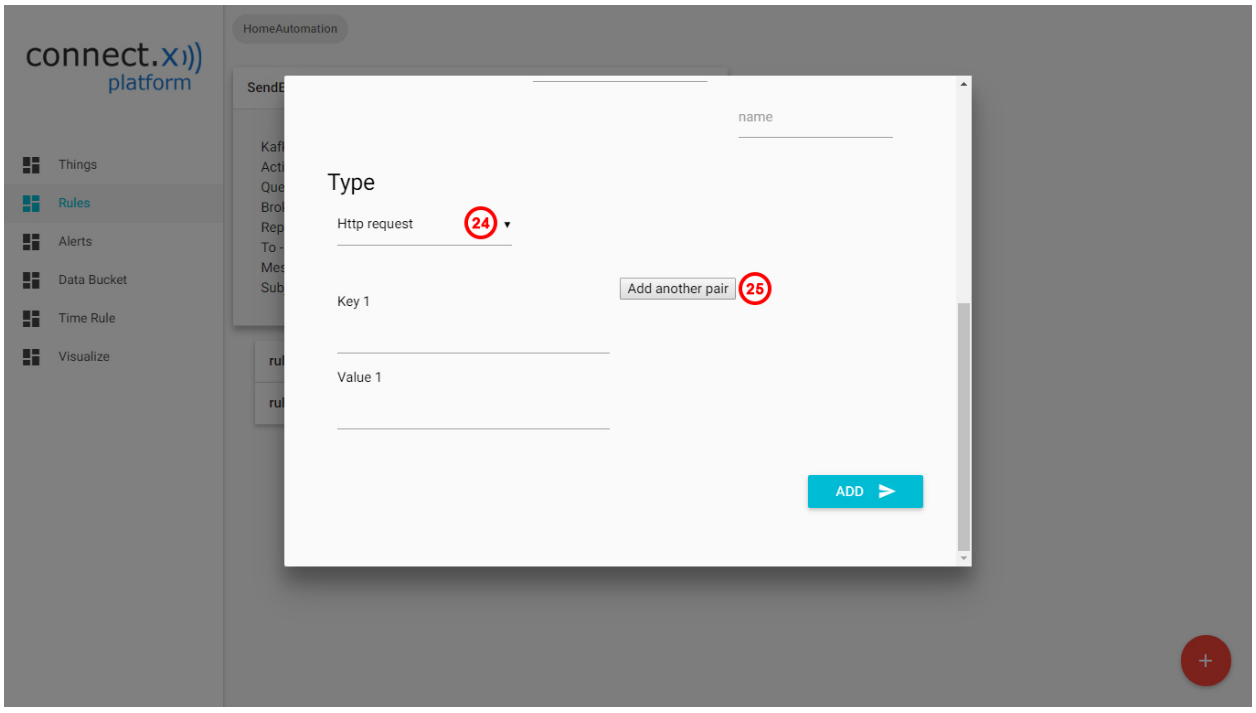


Figure 8.10: Add rule 5

Next set of features are Alerts and Databucket. Alerts tab shows all the messages which are coming on the alert topic of the device. It is like a log. Databucket can be seen as a rule which dumps the data in the platforms database coming from the devices on satisfying some specified condition.

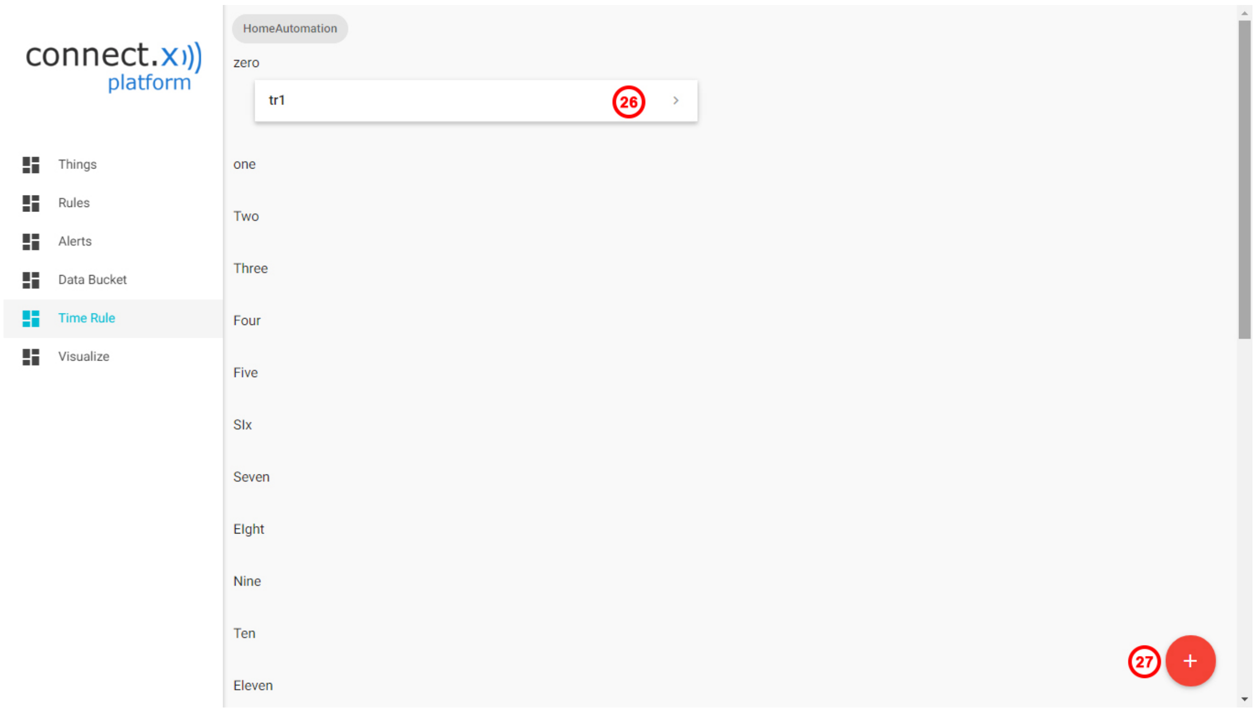


Figure 8.11: Time Rule Page

26. Time rules are based on the clock. User can add different rules which will execute on specific time of the day. There is no need to state any condition, rule will just execute irrespective of the condition, daily.
27. To add a new time rule, click on this button.

Lot of data is being stored in the databases. Trend in the data can be found by plotting a graph of all the values. This can be one by using Visualize feature. Only the data stored using databuckets can be used to plot these graphs.
28. Select the databucket which is already created.
29. Select the parameter for which the graph should be plotted.
Click on plot to generate a graph.

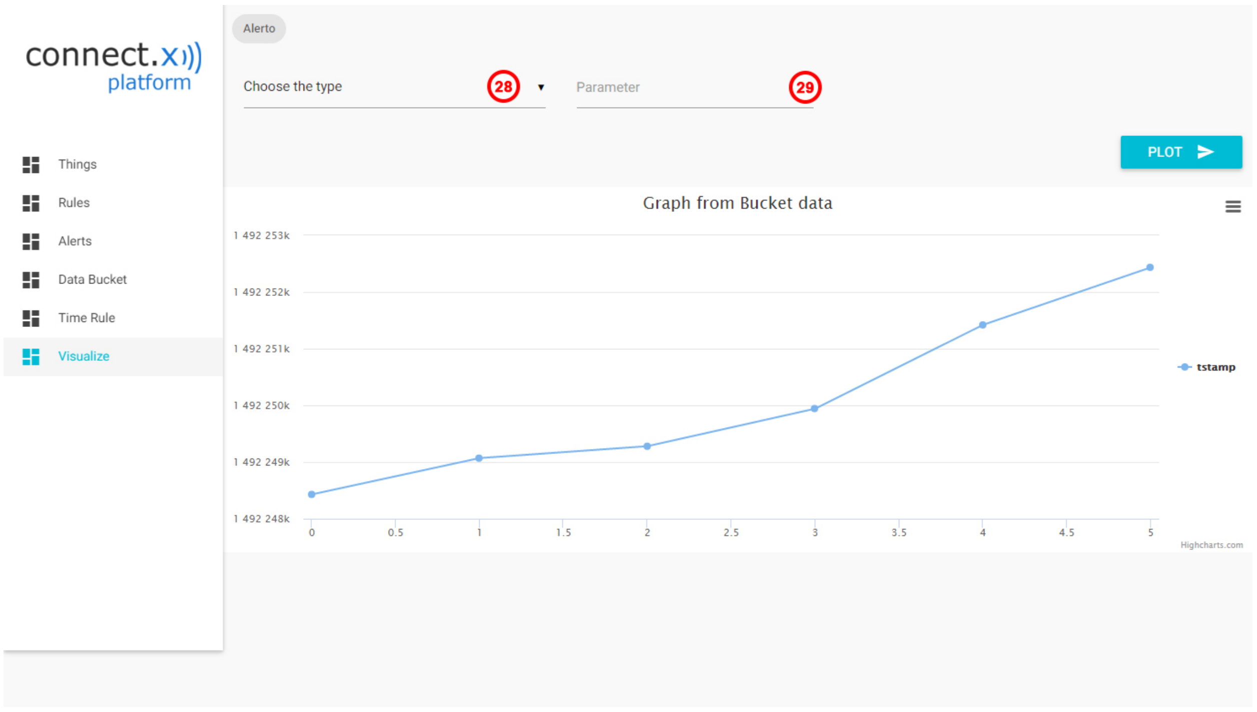


Figure 8.12: Visualize Page

Chapter 9

Home Automation: A Use Case

9.1 Introduction

Purpose

The main motive of creating an IoT platform is to make the deployment of IoT solutions easier. In order to test whether this goal is achievable through our platform, we demonstrate a simple use case of our platform- Home Automation. In order to make our lives simpler and easier, IoT can be harnessed for making our homes automated and smart. Mobile devices can be used to control basic functions. Users can have remote control of various appliances like lights, fans, AC etc. Thus an automation system enables control of electric appliances of various kinds. We built a simple use case of Home Automation using our own IoT Platform so as to demonstrate the capabilities of the Platform.

Benefits of using the Platform

Today home automation is quite common in developed nations. Remote access of appliances and devices in a home is provided via the internet. The challenge lies in making the system smart and deployable in a scalable manner across an entire city in a developing nation. Use of the IoT Platform makes home automation achievable across a smart city in a scalable and data efficient manner. Since the system is horizontally scalable, in the future, Home Automation could be made possible across a city through the use of our Platform.

Therefore, instead of deploying Home Automation from scratch, one can make use of the Platform and get the following benefits:

1. Each user will have remote control of devices in his home through an android application.
2. He will also be able to define rules to trigger actions in his home via a web dashboard.
3. The Rules engine implemented analyzes the data real time thus triggering various actions pre-decided by users.

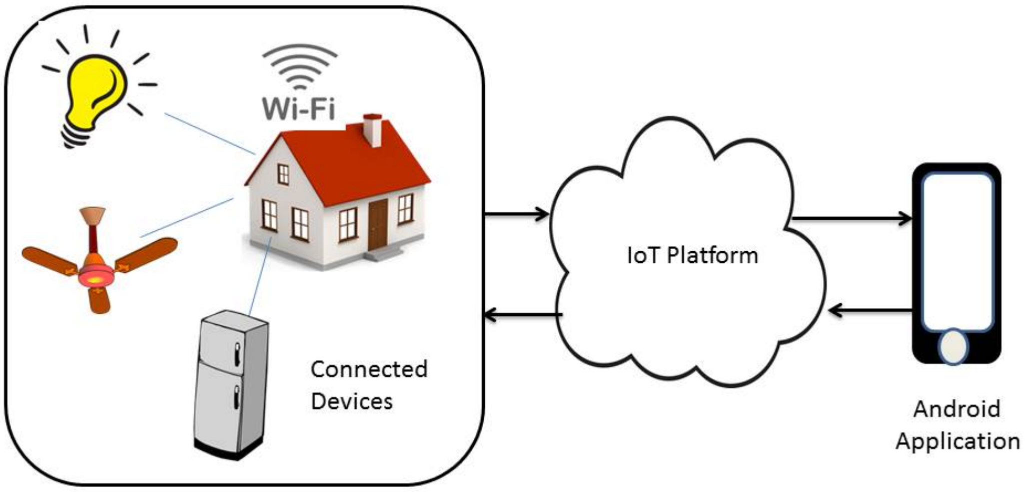


Figure 9.1: Home Automation

4. A single system can be deployed and can handle data coming from homes for an entire city.
5. The scalable architecture used ensures that there will be no problem handling data traffic as number of users increase.
6. One can exploit the unique features of the Platform - modular dynamic nature, scalability and real time analytics for quick decision making.
7. The system could involve energy monitoring for enabling judicial energy consumption in a home.

9.2 Components

9.2.1 Electronic Devices

The first component of any Home Automation solution are the electronic devices that need to be remotely controlled by the user. For this, the various devices and appliances in a user's home will need to have microcontroller boards which are connected to WiFi. For purposes of demonstration of how this can happen, we created a 'smart' socket. We used an ESP8266 Wemos D2 Mini. This board is extremely small and cheap and used in industrial production. We added this board behind a socket using the circuit shown in the figure below. Thus, use of this WiFi board behind a socket ensures that any kind of device can get connected to the internet. Once connected, the device can receive updates to change its state remotely from the android device.

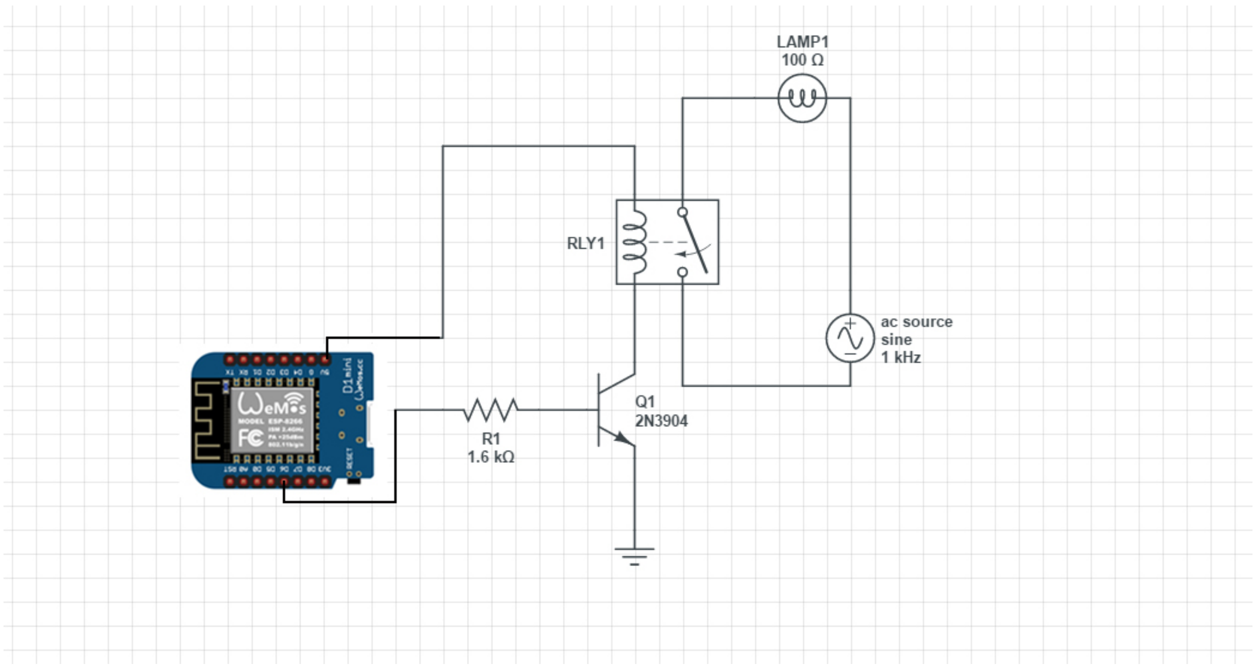


Figure 9.2: Circuit for Smart Socket

9.2.2 Android Application

The other part of Home Automation is the device using which one can control the devices in his home. For this, We created an android application. Through this application, the user can control the state of every connected device in his house. For example, can turn the lights on and off, reduce the intensity etc. He can turn the AC on before coming home. Apart from normal User Interface for controlling devices through phone, We developed a personal assistant that one can talk to for making things easier. By using this chatbot, the user can issue commands via voice input.

9.3 Implementation

9.3.1 IoT Modules

Register the devices: Create a project called ”House1” from the dashboard of the IoT Platform. Next, add the Thing Types, these will be the rooms in your house :

1. Kitchen
2. Hall
3. Bedroom

And any other rooms which will have connected devices. Next we need to add the Things : Give the Thing a name, eg: fan1 and select the type, eg: Kitchen, i.e the room in which the device is placed. Give each Thing a password.

After each connected device is registered with our Platform, the customized SDKs can be downloaded. In our implementation we used Wemos D2 Mini, which is Arduino based. Hence we used the Arduino SDK made by us. The code basically contains the following:

1. Configure which Wi-Fi to connect to (change the SSID and password)
2. Write the logic of data capture from IO pins.

At the end of this Step,the following unique device IDs will be registered for sending and receiving data from the Platform in the form of:

Projectname/roomname/devicename

1. House1/Kitchen/light
2. House1/Kitchen/bulb
3. House1/Kitchen/fridge
4. House1/Hall/fan
5. House1/Bedroom/TemperatureSensor
6. House1/Bedroom/AC

9.3.2 Android Application

We built the android application using Android Studio. The application is dynamic and generic in nature. It takes the Project name registered on the dashboard as Login and displays all the Things created on the dashboard. Whenever, a new type or new Thing is added, it is reflected in the Android application. The main screen allows you to see all the rooms(Thing types) in the House. For each room one can see the connected devices in that room. Their on/off state can also be seen and changed through the toggle button.

The implementation is through PHP based REST API. Android makes an HTTP GET request to fetch all types, devices and device states. The PHP queries the MongoDB database of the project and returns the result in the form of a JSON object to Android. This JSON is parsed in Android and displayed in the form of UI elements

The personal assistant was implemented using key word extraction and running queries corresponding to the key words. This was done by sending the input strings on android to a PHP on the server.

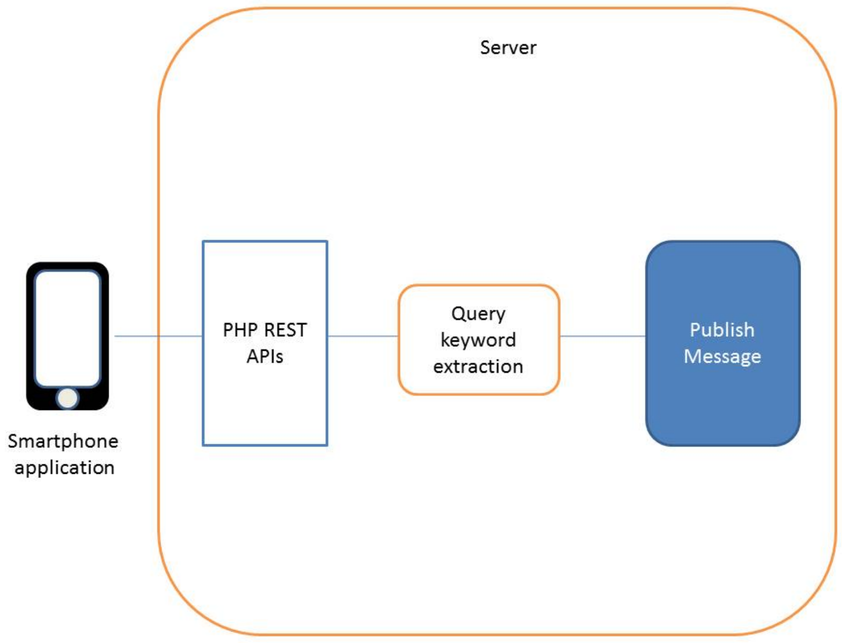


Figure 9.3: Android and PHP for Chatbot

We created Android SDK for connecting to our Platform using MQTT. We used this SDK in the application.

9.4 Interfacing

9.4.1 Connect the Device

Once the the circuit behind the smart socket for each device is ready, it can connect to our MQTT Broker using the Client ID and password that was set when the device was registered through the dashboard. For example, the fan in Hall will connect via the Arduino SDK provided by us by making a call to the MQTT connect function using the client ID House1/Hall/fan and its password. If the call is made without error, the status of the device will change on the dashboard to connected = yes. This can be checked to know if everything has worked properly.

9.4.2 Send/Receive messages Device

Next use the Arduino MQTT SDK provided by us to publish and subscribe to certain topics.

Publish Mechanism:

Every device in a home will publish data with a unique topic which will have the following format:
$state/project-name/room/device-id
$iot/project-name/room/device-id
The reason for choosing this format is that the data coming for different users homes needs to be

differentiated. Therefore, the IoT modules will be allowed to publish with only these topics. They will publish their on/off state with the "$state" topic whenever there is a change in their state. Sensors will publish data at a defined interval with a topic "$iot".

The state of a device can be changed either manually by the user turning of the light switch or increasing the speed of the fan; or remotely through the user interface on android application that the user will have. Whenever there is a state change, the device will publish a sate update with the topic

$state/project-name/room/device-id/update

Every module will send data as individual JSON strings. For example a light bulb will publish the following data about its state using the MQTT topic

"$state/house1/Kitchen/bulb/update"

{
State:"on"
}

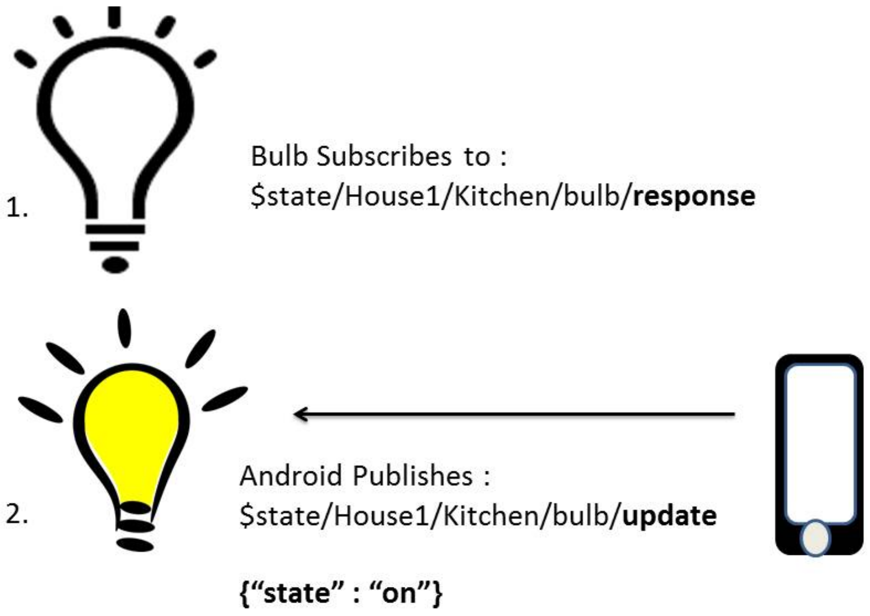

1.
Bulb Subscribes to :
$state/House1/Kitchen/bulb/**response**
2.
Android Publishes :
$state/House1/Kitchen/bulb/**update**
{"state" : "on"}

Figure 9.4: Publish and Subscribe Mechanism

Subscribe Mechanism:

Every IoT module will also subscribe to this state topic

$state/project-name/room/device-id/response

Thus when a state update is sent from the remote trigger (i.e. the android phone) it will be received by the intended device. The mapping between update and response is achieved by the State Database and Analytics engine which is explained in the next sections.

Once the device receives a state response message, it will read the message and depending upon state on/off it will turn the output pin high/low, thus completing or breaking the circuit, thereby turning the device on/off.

9.4.3 Send/Receive messages Android App

The android application needs to show the current state of all devices. For this it reads the states directly from the MongoDB state database on the Server by making PHP calls. As described in the Architecture Section, the state Database contains the last known state of each device. Android uses this Database to show the current on/off states of all devices.

When the button is clicked to turn a device on/off android needs to publish a state update message for that device.

9.5 Working

9.5.1 State Update

When the user toggles a button from the android UI to change the state of a device, the android phone publishes a state update message. This message travels through the Platform architecture:

- It is first received my the MQTT Broker and forwarded to Kafka.
- Kafka then sends the message to Spark.
- Spark updates the State Database to reflect what is sent in the State Update message.
- Spark publishes a State Response message with this updated state.
- Since the device has subscribed to its own State Response,it receives the message and changes its state.

This is what happens behind the scenes when the toggle button is pressed from the Android UI; or if the command to turn on/off is given via the Chatbot.

9.5.2 Rules

Simple rules can be set through the Web Dashboard of the Project to achieve a wide variety of tasks. The simplest of them being receiving an email when a certain action happens in the House.

For example, a rule can be set from the web dashboard that if the temperature in the Hall exceeds a certain level, turn on the Air Conditioning and also send an email to the user. Whenever

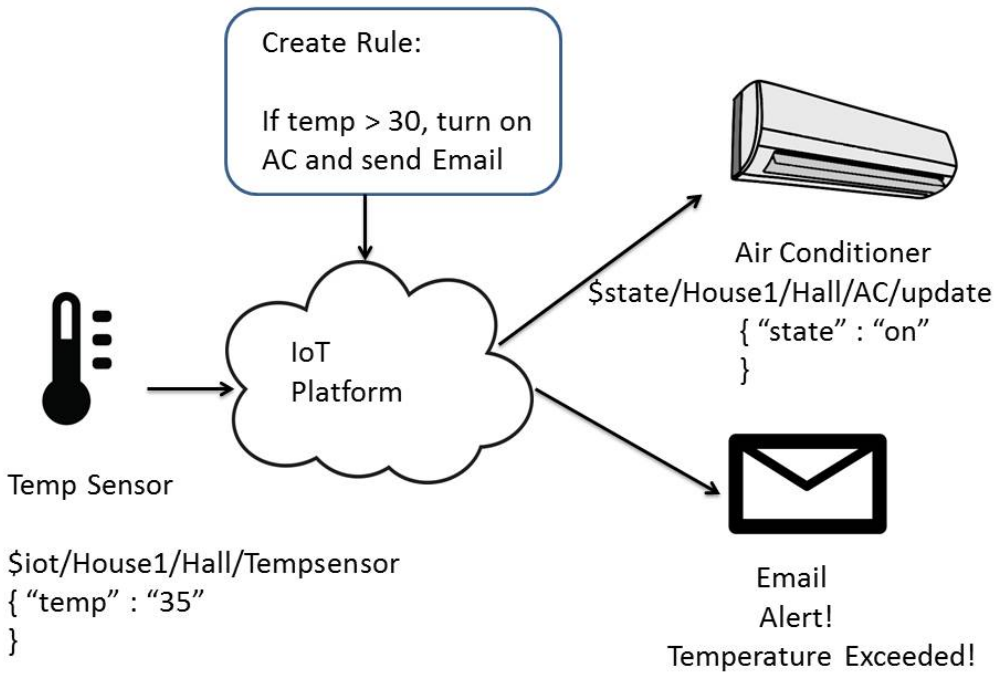


Figure 9.5: Temperature Rule

the platform will receive a message with the topic \$iot/House1/Hall/Tempsensor, it will check the message contents for the parameter "temp". It will then execute the rule if temperature exceeds 35 and re-publish a state update to the Air Conditioning.

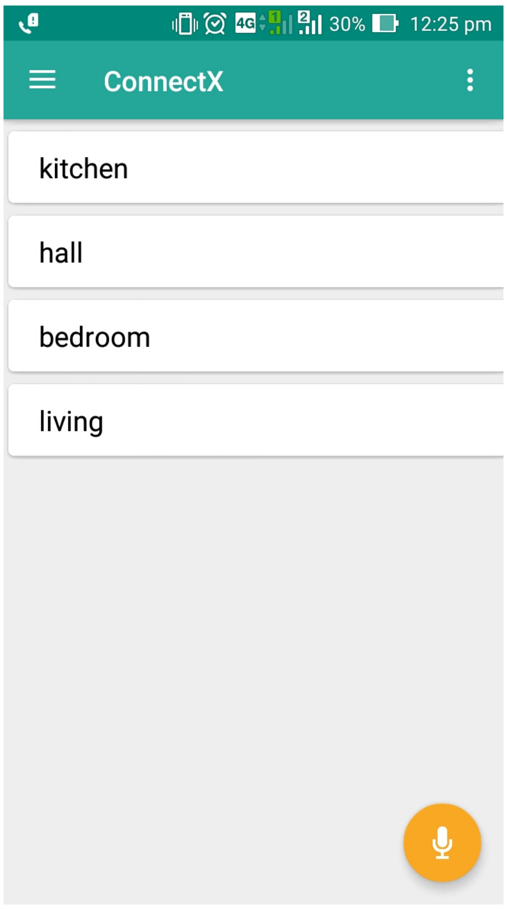
9.5.3 Time Rules

Time Rules can be set to trigger repetitive tasks that happen everyday around the same time. For example, the user may want the water geyser to be started everyday at 7 a.m. He can do this by setting the time rule through the dashboard just specifying the ID of the geyser, the time and the state action. The geyser will automatically turn on at 7 a.m everyday.

9.6 Results

The following are screenshots of the android application: Figure 9.6 shows the rooms in "House1" project from the dashboard.

Figure 9.7 shows the current state of devices in the Kitchen. The toggle buttons can be used for turning them on/off.



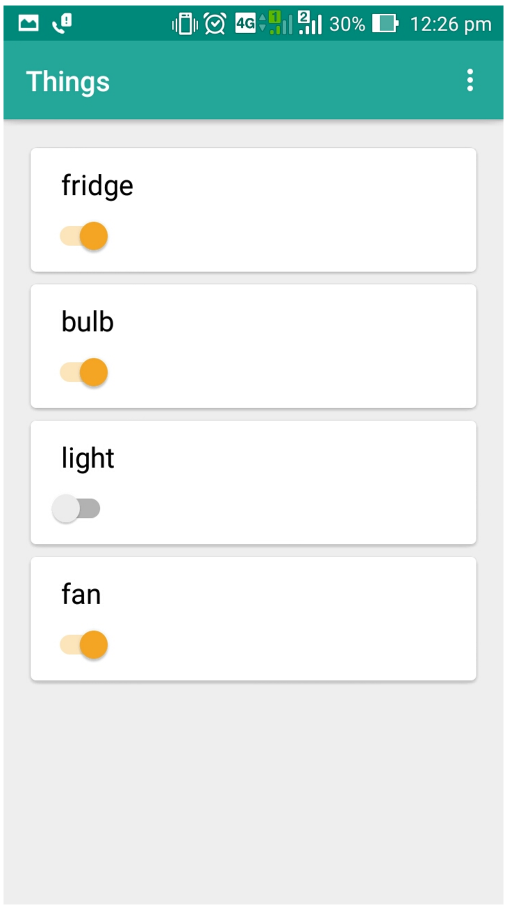
kitchen

hall

bedroom

living

Figure 9.6: Rooms in House1



fridge

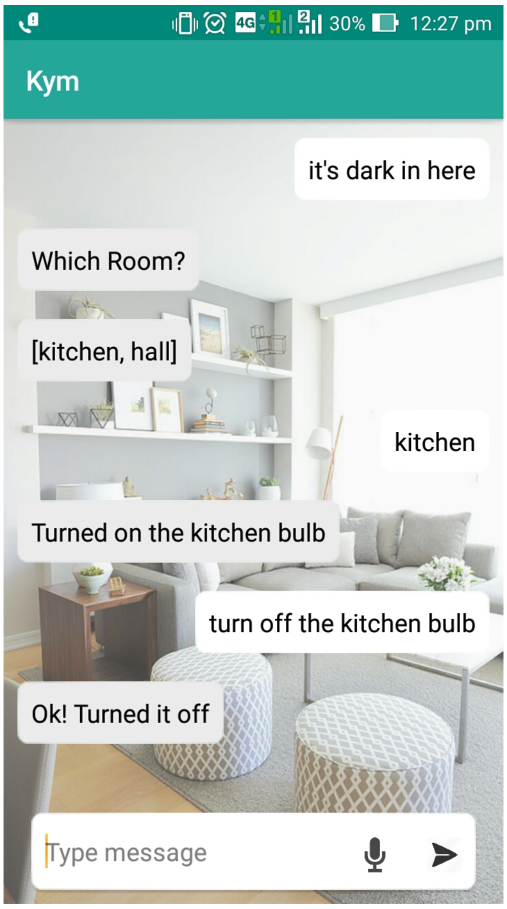
bulb

light

fan

Figure 9.7: Devices in the Kitchen

Figure 9.8 shows the Chatbot. Voice commands can be given to change the state of any device. Time rules can also be set through this personal assistant.


Kym

it's dark in here

Which Room?

[kitchen, hall]

kitchen

Turned on the kitchen bulb

turn off the kitchen bulb

Ok! Turned it off

Figure 9.8: Chatbot

Chapter 10

Conclusion

1. Connect-X IoT platform is an end to end IoT platform for industrial and domestic use cases.
2. The platform allows deployment of IoT solutions with no coding, within a few hours, all through the UI, thereby reducing costs and time required for developing the systems.
3. ConnectX IoT platform is extremely user friendly, allowing non-coders to easily deploy IoT solutions.
4. The platform made is highly scalable - Big data ready to handle millions of data streams.
5. The platform is secure and reliable for data storage.
6. The rules engine implemented enables data analytics driven through UI based interfaces.
7. The IoT platform has support for any end application that can be triggered by the rules engine.
8. With completely Open-source components based back-end, the technology cost of Connect-X is zero.
9. Having all features found in IoT platforms like AWS IoT, Azure IoT, IBM IoT, etc Connect-X is a true Industrial and future ready IoT platform.

Chapter 11

Future Work

1. Changing MongoDB to Apache Cassandra after migrating to servers with better configuration.
2. Making the rules engine capable to handle complex data analytics rules involving data conditions from multiple data streams.
3. Migrating the system to a cluster instead of stand alone server.
4. Introducing edge analytics feature to minimize dependence on cloud and putting intelligence into IoT hubs - that can be controlled through the server.
5. Building more end application interfaces.
6. Building more protocol broker to Kafka bridges to make multiple protocols inherent part of the IoT platform.
7. Adding SDKs for more hardware platforms
8. Researching and developing encryption algorithms that will ensure high security without hampering the light weight nature of the system.

Chapter 12

References

1. https://aws.amazon.com/iot/
2. https://www.microsoft.com/en-in/internet-of-things/azure-iot-suite
3. https://www.ibm.com/internet-of-things/
4. http://mqtt.org/
5. https://www.amqp.org/
6. http://portals.omg.org/dds/
7. https://xmpp.org/
8. http://www.eclipse.org/paho/
9. https://mosquitto.org/
10. https://www.hivemq.com/
11. https://www.rabbitmq.com/
12. http://emqtt.io/
13. https://kafka.apache.org/
14. https://www.amqp.org/
15. https://www.storm.apache.org/
16. https://www.spark.apache.org/
17. http://samza.apache.org/
18. https://www.mongodb.com/

19. https://www.cassandra.apache.org/

20. https://www.oracle.com/database/nosql/index.html

21. https://hbase.apache.org/

22. https://mean.io/

23. https://rubyonrails.org/

24. https://www.meteor.com/

25. https://php.net/manual/en/intro-whatis.php