A Scalable Home Automation IoT Ecosystem using open source technologies

Atharva Phand[1], Roshan Bangar[1], Reshul Dani[2], and Jay Lohokare[3*]

[1] *College of Engineering Pune, Maharashtra, India*
[2] *University of California San Diego, California, USA*
[3] *Stony Brook University, New York, USA*
**Email: jlohokare@cs.stonybrook.edu*

Abstract

Home Automation provides remote access to appliances and devices in a home via the internet. While this is easily achievable on a small scale, the challenge is making the system deployable in a manner that is scalable across a whole city in a developing nation. This paper proposes an end-to-end home automation suite with modular open source components. Our backend supports multiple IoT protocols and is horizontally scalable. We implement a 'Rules engine' that analyzes real-time data and triggers various actions pre-determined by users. We also implement energy monitoring for judicial energy consumption in a home. Our system, when deployed, can handle data coming from homes for an entire city. Each user will have remote control of devices in his home through an android application. The user will also be able to define rules to trigger actions in his home via a web dashboard. The architecture used ensures that the system scales as the data traffic increases. Our experiments on the prototype system show low latency for real time analytics even when there are large simultaneous data streams. Our main feature contributions are the modular and dynamic nature of our system, scalability, and real time analytics for quick decision making.

1 Introduction

Internet of Things and related technologies have witnessed a dramatic rise in deployments over the past years. We see the technologies used in various industries and use cases like smart cities, manufacturing, healthcare and they also play a major part in gadgets we use everyday [1].

We can harness the power of IoT for making our homes automated and smart, thereby making our lives simpler [5]. Today, remote mobile devices can be used to control basic functionality in a home [3]. Various appliances like lights, fans, air-conditioners etc. can be controlled by users even they are not around.

We can thus define an automation system as an end to end platform that enables control of diverse appliances remotely. Devices connected to such an automation platform can drive smart processes by driving intelligence from data captured by various sensors. These internet connected devices create a large amount of data which can be used to build models and aggregation logic to then drive automated decision making. The data can be used to find patterns, analyze behaviour and trends, predict future readings based on historical performance. With such a data driven platform, we can analyze energy consumption of the connected devices and find ways to minimize energy footprints. For such a smart system, we need a highly scalable cloud platform that can connect to and capture data from millions of devices concurrently. The platform should present a mechanism to collect data from devices and sensors, process the data while assuring zero data loss, analyze the data in real time, store the data securely while assuring persistence and finally support creation of models to help us drive insights.

In this paper, we present a platform for deploying home automation systems based on Open source technologies. We present the system architecture and individual components used, provide details on the platform's logic and how it works, and then finally present results from experiments we conducted using the proposed platform.

2 Related Work

In related work, take a look at systems that utilize IoT for home automation purposes. There are many systems that exist today in this area; most show how data is sent from sensor controllers to the cloud and then to end points like mobile devices[4], [2]. What is lacking in these systems is an architecture that can scale with the rise in the number of concurrent users. A lot of research has also been done for the implementation of security in smart homes. For example, sending mobile alerts if sensors detect the presence of, say, intruders [6], [9]. Our IoT framework is that it is not just limited to security and pre-defined experiences. The user has complete control and freedom over implementing rules that favor him in his home. Alerts will be received if any of these rules get triggered. We define the concept of rules-engine in the next section. This is a major advantage of our paper compared to prior work. We also do not limit the actions that can be taken after a rule is met, SMS and email just being one of many actions that can be defined [10].

Prior work shows how appliances can be controlled using IoT [13]. This uses REST-based APIs for sending data to the cloud, which is inefficient. In our system, we make use of the MQTT protocol for all communication which is data-efficient and enable us to support multiple IoT protocols.

Our use of open source is similar to the work proposed in [8], which makes use of Apache Storm. However, scalability is enabled with local logic. In our architecture, we chose compo-

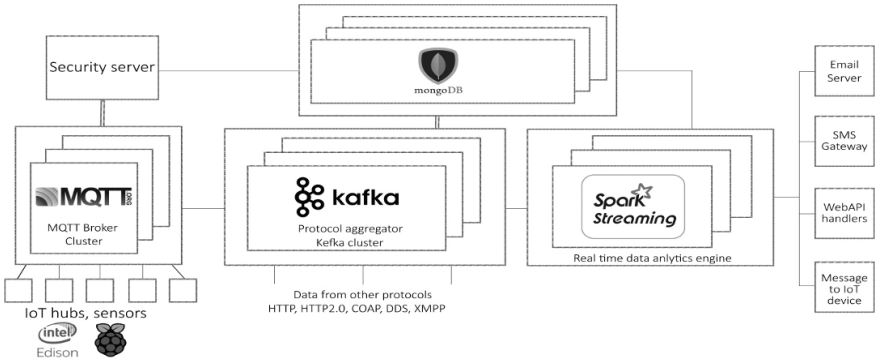

Figure 1: End-to-end IoT architecture

nents that can scale up/down as needed. For example, Apache Spark handles real-time streaming data and performs analysis on these with minimal latency.

3 System Architecture

We install microcontroller boards in the appliances in user's home that can connect to WiFi. Today, such smart appliances already exist in the market. These devices will send information about their current state to the cloud. The platform is based on a 'cloud-first' implementation using various open source components.

3.1 Messaging Protocol

Protocols like HTTP, when used in a setting like home automation, utilize high internet consumption for sending data. Today there exist various IoT open-source protocols like MQTT [11], DDS, CoAP, XMPP, and AMQP. These have low bandwidth requirements and perform under unreliable network conditions whilst ensuring security through encryption [12]. MQTT is a widely used protocol for machine-to-machine communication. It is extremely lightweight and thus performs well for IoT applications involving publish/subscribe message transport. It utilizes very little code footprint and low network-bandwidth [7]. Usually, implementations of MQTT contain the following 3 components.

1. Publisher. The publisher is the data producer that sends data on a particular topic to the MQTT broker.

2. Subscriber. Devices "subscribe" to a particular MQTT topic of interest and receive data about this topic when publishers send it. Subscribers only receive data about topics they have subscribed to.

3. Broker. The broker is the server that receives messages from publishers and forwards them to subscribers.

3.2 Publish/Subscribe broker - MQTT

For achieving horizontal scalability, we can cluster MQTT brokers. A master controls the clustered MQQT brokers to make them act as one. Every broker maintains a list of topics that are flowing in the system. If a given broker is unavailable, another one can handle the request and balance the request. A highly available system is made possible via horizontal clustering.

3.3 Cross-Protocol Message Queue - Kafka

We do not limit our platform to any particular IoT protocol. Devices in a home and send information via any protocol they like and we aggregate it in our middleware. We use Apache Kafka [18] to connect data coming from different protocols. Moreover, Apache Kafka also supports distributed architecture, further satisfying our system scalability requirements. Kafka produces a log file containing the raw data, which is also helpful for data analysis. Overall, the introduction of kafka in our system ensures our system's scalability, availability, and security.

3.4 Message processor - Spark Streaming

One core component our home automation system is real-time "rules engine". Each home sends us continuous data streams that need to be analyzed to determine which actions to trigger. We also need to provide flexibility to the users to decide what rules they want to configure. From there, we can maintain a list of rules for each user and execute the rule on every incoming data stream coming from the user's home. Apache Spark provides near real-time analysis on large-scale data. It is known to be 100 times faster than Hadoop [17]. Furthermore, we can use Mlib for machine learning on streaming data to determine user habits and make suggestions accordingly.

Therefore, the above open source components have been carefully chosen in the design of our end-to-end home automation suite.

4 System Implementation

Fig. 1 presents the overall system architecture for the platform proposed in this paper. In this section, we present what the components used are, and how they function together.

4.1 Overview

IoT modules are the devices or applications which create and consume data for the IoT platform. These include smart devices (like Bulbs, TV, etc), smart switches or any other internet enabled devices. IoT modules also include sensing devices responsible for generating data for temperature, humidity, door state (open/closed), switches etc. We create SDKs/modules for all these devices based on commonly used frameworks (linux, arduino) to enable them to talk to the

IoT platform. The devices will dump data to the IoT platform ever 2 minutes. Each device will maintain its state offline (on/off/analog value). Similarly, the state will also be stored on the platform and then can be synced with the devices state. The communication between devices and servers to sync the device states is achieve via publish/subscribe messaging queues. We use MQTT for achieving such queue based messaging.

Message transfer Mechanism: Every IoT device onboarded to our platform publishes messages with state updates on a unique MQTT topic -

$state/address-key/room/device-id/

$data/address-key/room/device-id

We chose this format to achieve isolation between messages coming from different devices, homes or sections within the home (roomIds). The paltform servers blocks any messages that don't meet this format at their gateways, to ensure compute resources are spent on only messages relevant to the platform. The on/off state messages are published with '$state' topic only when there is a change in the state of the devices. Similarly, for any sensor readings, the devices use the '$data' topic.

Device states can be changed manually at device level (e.g. switching off a light) or remotely via various interfaces supported by the platform. We currently support control via a website and an Application, both of which change the state of devices on the servers which then causes the sync to force devices to change their states. When ever there is a state change, the device publishes a message on the following topic:

$state/address-key/room/device-id/update

Devices send messages as individual JSON strings. For e.g. A light bulb would use the following MQTT topic while sending a state change message -

$state/house1/room1/bulb1/update –

{

State:"on" Timestamp: "531423"

}

Subscription: Every device onboarded to the platform automatically subscribes to the message queue on the following topic:

$state/address-key/room/device-id/response

Whenever a state update is triggered remotely, the platform servers generate a message over the topic corresponding to the device. The device receives this message and then sync its state to the state as described in the JSON message. The data processing engine (also called Analytics engine) is responsible for message processing and generation.

4.2 Cloud implementation

The platform proposed in this paper follows a cloud first paradigm - Devices send data to the servers, the servers process the data to take action, and the devices react to the action. We used an AWS EC2 instance [16] for building our prototype, however the platform can also be deployed in an hybrid environment (Across clouds, multiple data centers or combination of both). We chose EC2 to leverage the vertical scalability it offers, which helped us minimize the containers/virtual machines we manage. To build the messaging queue, we used a MQTT message broker cluster (horizontally distributed) across 2 EC2 instances. There are many MQTT implementations available, but we used eMQTT [16] due to its high availability and proven horizontal scalability.

There are many messaging protocols used in the industry for IoT use cases (MQTT, CoAP, DDS, etc). To enable our platform achieve cross protocol communication, we built a common message queue over the MQTT implementation. This queue was built using Kafka, and can act as a protocol aggregator - All messages originating from any underlying protocols will land on Kafka and then be forwarded to a single processing engine. We built the MQTT to Kafka bridge by creating a MQTT-Kafka bridge for eMQTT - Any messages landing on the MQTT brokers are automatically forwarded to the kafka queues. Similar to this bridge, we can build new bridges for other protocol brokers/servers to capture messages on our platform.

4.3 Analytics Engine

After the platform gets messages from devices on Kafka queues, the messages need real time processing to enable fast response times. The data from Kafaka message queues is pulled in by a 'rules engine' i.e. a set of Spark streaming jobs processing the data in near real time. The streaming application uses kafka dStreams [15] to consume the messages on the common queue. Users can set 'rules' to perform actions when the incoming data meets certain criteria. The spark streaming rules engine we built allows actions like notifications, sending emails, triggering new messages for IoT devices based on rules written in a simple SQL like format. For example, users can set a rule that turns on a fan when temperature goes above 50C. Similarly, other rule could be to send a SMS when total usage for heater exceeds 200 units for the month. We use NoSQL (MongoDb) for storing these rules, which are SparkSQL queries that the streaming jobs use for filtering data. For eg, lets consider the following JSON as a message received by the spark job -

{

"topic": "$data/house1/bedroom/temp-sensor",

"message": ["temperature": 50]

}

The spark streaming application will first retrieve the rules for "house1" and the corresponding action, i.e turn on the Air Conditioning. It will then convert each rule into an SQL query and execute it on the data it gets. For the temperature rule discussed above, the following query will be executed – "select * from message where temperature > 40". Thus it will retrieve this message from all messages it gets and trigger and action to switch on the AC by sending a state update request to the AC via MQTT by publishing the following message -

{

"topic": "$state/house1/bedroom/AC/update"

"message": {"state": "on"}

}

Therefore, the details of every rule and action will have to be entered by the user and the backend will automatically implement these rules on the raw data on near-real time basis. The dynamic nature of rules in the Analysis engine is one of the unique features of this system.

The dynamic nature of rules in the Analysis engine is one of the unique features of this system.

4.4 State Engine

When the spark streaming jobs receive messages containing state updates, the job updates the respective state in MongoDB [14] state store of the platform. We call this state store the 'State Engine' or the 'State DB'. We implement this state engine by creating a MongoDB collection per project, with individual key-value document representing the individual devices. We use a MongoDB-Spark custom connector to store the Spark RDDs into MongoDB. The collection to write the state to is determined by parsing the message topic string.

"$state/house1/room1/bulb1/update" -
{
State:"on"
Timestamp: "531423"
}

When the above message is received by the Spark Streaming application, the room1/bulb1 document is updated to reflect state "on" in the house1 collection present in State DB. Next, the analytics engine will trigger a MQTT publish to the topic:

"$state/house1/room1/bulb1/response"

With the new state received, the platform then publishes a message on topic which the devices have subscribed to. The device then receives the message and changes it state to the one it receives from the message.

4.5 Energy Efficiency

We present a simple approach to monitor the energy footprints of appliances. The platform captures timestamps corresponding to state changes for the devices on-boarded to the platform. We maintain a database of energy footprints corresponding to commonly used devices, and use this data to find overall energy usage of all the devices registered. The calculations are driven by multiplying time the device was in a switched on state with average energy footprint per unit time for the device. Though the results are not accurate (due to non linear nature of the time-energy footprint), they are sufficient to drive qualitative and comparative analysis on footprints of devices on the platform.

5 System evaluation

To study the qualitative aspects around the proposed framework, we deployed the proposed platform on AWS EC2 instances. We deployed one node each for the MQTT broker, Kafka queue, Spark streaming application and MongoDB state database. We built smart devices by pairing common appliances with Raspberry Pi 3 to add internet and compute capabilities. These devices were our proxy for smart appliances that would eventually connect to our IoT platform. The architecture proposed helps us achieve the following features driving home automation -

Centralized cloud for smart city. The platform can serve as the backend for city/state wide automation efforts. Being horizontally scalable and agnostic of underlying hosts (data centers, cloud, etc), this platform is flexible to satisfy any criteria required by governments and companies.

Real time remote control. The proposed platform enables seamless remote control of appliances via the internet. Being a cloud first technology, we are not limited by hardware and physical implementation constraints. We can build multiple interfaces to interact with the platform and control the devices on-boarded by leveraging open source MQTT implementations. For our prototype, we built an Android application using Eclipse Paho JAVA MQTT clients. We made the app subscribe to the topic $state/home1/+/+/response i.e. all topics supported by the platform (corresponding to all devices). The app can not only view the current state, but also issue updates to change the device states. The messaging logic established enables us to extend this functionality to any new interfaces across platforms.

Persistent data, assured message delivery. The actual state of the devices and the states shown in the user interface of the android phone need to be in sync with each other. If the user turns off a light manually, a publish is triggered. However there is a possibility that the android phone does not receive this message if internet goes off. The use of our architecture and the definition of pub sub topics ensure that nothing is lost and the devices are always in sync.

Near real time response. The platform uses Spark streaming to process the messages in micro-batches. Our platform helps us scale up / scale down the cloud footprints as required based on the number of incoming messages. If at any given time we see an increase in the number of messages, Spark auto scales to add more server nodes to process the larger amount of data. The messaging queues are horizontally scalable in a similar manner - If the number of active devices increases, the message brokers will automatically add new nodes behind the proxy to ensure no node has excess number of devices connected to it. With every component involves in the message processing pipelines horizontally scalable, we assure high availability and low latency.

Energy consumption dashboards. In our experimental setup, we implemented a web-app to summarize the energy footprints of the devices on the platform. This dashboard is works alongside a simple data aggregating pipeline built over the Platform. This is a good example for how we can build custom applications on the underlying IoT platform. The web-app serves as a one stop dashboard to monitor energy consumption of all devices and also provides added analysis around patterns in consumption (what time of the day/day of the week there are peaks in consumption, what type of devices consume most energy etc). Such a web-app serves as the final piece of systems needed for end to end home automation.

6 Deployment and results

We use MQTT to enable data efficiency. The devices only send the delta in states, thereby preventing the network from flooding with multiple data requests. We use scalable components and asynchronous calls to makes our platform scalable. We deployed the platform on 2 Amazon web services (AWS) Elastic cloud compute (EC2) instances. Apache JMeter was used to load test the servers. The cluster of 2 EC2 nodes gave less than 20ms latency for 50,000 devices simulated on JMe-

ter. With number of simulated devices increased to 80,000 the latency rose to 50ms. Adding another EC2 node to the cluster reduced this number back to less than 20ms. This shows that the proposed architecture is horizontally scalable and efficient.

7 Conclusion

In this paper, we have proposed a scalable architecture for home automation based on open source technologies. The aim was to provide utilities like remote control, actions based on rules, energy audit and smart learning to the users. The framework presented presents these features while ensuring scalability and low cost of deployments - The use of MQTT protocol leads to data efficiency, Apache Kafka enables the connection of multiple IoT protocols, rules engine based on Apache Spark Streaming allowed near real time analysis for many concurrent data streams with very low latency. The system is dynamic and infinitely scalable. Our qualitative analysis shows that this platform can scale up to city wide deployments.

References

[1] H. N. Saha, A. Mandal and A. Sinha, "Recent trends in the Internet of Things," 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2017, pp. 1- 4. doi: 10.1109/CCWC.2017.7868439

[2] D. Pavithra and R. Balakrishnan, "IoT based monitoring and control system for home automation," 2015 Global Conference on Communication Technologies (GCCT), Thuckalay, 2015, pp. 169-173.

[3] K. Mandula, R. Parupalli, C. A. S. Murty, E. Magesh and R. Lunagariya, "Mobile based home automation using Internet of Things(IoT)," 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, 2015, pp. 340-343.

[4] M. Thiyagarajan and C. Raveendra, "Integration in the physical world in IoT using android mobile phones," 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Noida, 2015, pp. 820-826.

[5] G. Kesavan, P. Sanjeevi and P. Viswanathan, "A 24 hour IoT framework for monitoring and managing home automation," 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, 2016, pp. 1-5.

[6] V. Patchava, H. B. Kandala and P. R. Babu, "A Smart Home Automation technique with Raspberry Pi using IoT," 2015 International Conference on Smart Sensors and Systems (IC-SSS), Bangalore, India, 2015, pp. 1-4.

[7] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," 2013 International Conference on Computing, Networking and Communications (ICNC), San Diego, CA, 2013, pp. 334-340.

[8] M. Frincu and R. Draghici, "Towards a scalable cloud enabled smart home automation architecture for demand response," 2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), Ljubljana, 2016, pp. 1-6.

[9] R. K. Kodali, V. Jain, S. Bose and L. Boppana, "IoT based smart security and home automation system," 2016 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2016, pp. 1286-1289.

[10] Jain Sarthak, Anant Vaibhav, Lovely Goyal, "Raspberry Pi based interactive home automation system through E-mail", Optimization Reliabilty and Information Technology (ICROIT) 2014 International Conference on. IEEE, 2014.

[11] MQTT , [online] Available: http://mqtt.org/documentation

[12] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," 2016 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT), Cairo, 2016, pp. 1-7

[13] R. K. Kodali, S. Soratkal and L. Boppana, "IOT based control of appliances," 2016 International Conference on Computing, Communication and Automation (ICCCA), Noida, 2016, pp. 1293-1297.

[14] MongoDB NoSQL database, Available: https://docs.mongodb.com/

[15] Spark Streaming and Apache Kafka bridge, Available: https://spark.apache.org/docs/latest/streaming-kafka-integration.html

[16] Amazon EC2, [online] Available: https://aws.amazon.com/documentation/ec2/

[17] Apache Spark Streaming, Available: http://spark.apache.org/streaming/

[18] Apache Kafka Messaging queue, Available: https://kafka.apache.org/