|  |
| --- |
| VIETNAMESE-GERMAN UNIVERSITY  KARLSRUHE UNIVERSITY OF APPLIED SCIENCES  **MASTER THESIS**  **DEVELOPING MQTT DASHBOARD**  **WITH SCALA AND AKKA**  *Author: Tan Dung, Tran*  *VGU Student ID: 18859*  *HKA Matriculation Number: 81182*  *Referee: Prof. Dr.- Ing. Thorsten Leize*  *Co-referee: Prof. Dr.- Ing.*  *Supervisor:*  *Supervisor:*    A thesis submitted in partial fulfillment of the requirements for the degree of  **MASTER OF SCIENCE**  in Faculty of  **Mechatronics and Sensor Systems Technology**  **2022** |

Contents

[1. Overview: 3](#_Toc108773441)

[1.1. Abstract: 3](#_Toc108773442)

[1.2. Literature review: 3](#_Toc108773443)

[1.3. Review of MQTT Broker: 5](#_Toc108773444)

[1.3.1. Conception of MQTT Broker: 5](#_Toc108773445)

[1.3.2. Topic review: 6](#_Toc108773446)

[2. Introduction of thesis: 6](#_Toc108773447)

[2.1. The approach of this thesis: 7](#_Toc108773448)

[2.2. Thesis aims and objectives: 7](#_Toc108773449)

[3. Tools and Methods: 8](#_Toc108773450)

[3.1. Introduction of Scala 8](#_Toc108773451)

[3.2. Introduction of AKKA: 9](#_Toc108773452)

[3.3. Introduction of Play framework: 9](#_Toc108773453)

[4. Developing of MQTT Broker: 9](#_Toc108773454)

[4.1. Idea of Implementation: 9](#_Toc108773455)

[4.1.1. AKKA Toolkit: 10](#_Toc108773456)

[4.1.2. AKKA Actors: 10](#_Toc108773457)

[4.1.3. AKKA Stream: 10](#_Toc108773458)

[4.2. Software design of programs: 10](#_Toc108773459)

[4.2.1. Actors Model of Program: 10](#_Toc108773460)

[4.3. Explanation of Coding: 12](#_Toc108773461)

[4.4. Testing and Result: 12](#_Toc108773462)

[5. Developing of User Functions: 12](#_Toc108773463)

[5.1. Block diagram: 12](#_Toc108773464)

[5.2. Software design of programs: 12](#_Toc108773465)

[5.3. Explanation of Coding: 12](#_Toc108773466)

[5.4. Testing and Result: 12](#_Toc108773467)

[6. Developing of User Interface: 12](#_Toc108773468)

[6.1. Block diagram: 12](#_Toc108773469)

[6.2. Software design of programs: 12](#_Toc108773470)

[6.3. Explanation of Coding: 12](#_Toc108773471)

[6.4. Testing and Result: 12](#_Toc108773472)

[7. Intergration of MQTT Dashboard: 12](#_Toc108773473)

[7.1. Testing and Result: 12](#_Toc108773474)

[7.2. Conclusion: 12](#_Toc108773475)

[References 12](#_Toc108773476)

# 1. Overview:

## 1.1. Abstract:

Time has been changed, the expansion of information makes a big change in industry or heath care, service or transportation. As a result, the concept of IoT was introduced and becomes more popular, it appears and is often an integral part of all aspects of life and technology. For example, many years ago, to measure heart rate parameters, patients had to make an appointment and visit a doctor, which is troublesome when compared to today, when IoT was applied, patients only need to wear an end-device, which has the necessary sensors and connection (WiFi, 4G). Then, the doctor can update his heath continuously, real-time without the two even having to see each other, that's really is a revolution.

To do that, the connections between devices have to be established for data sharing. In the previous example, it is the connection and communication between patient’s end-device and doctor’s laptop/smartphone. This is also correct with a bigger system, which includes many smaller systems, when each small system – likes a device in the big system, need to “talk” to each other to deligate tasks or receive the input for its tasks.

Hence, data contribution is the most important. Additionally, the way to contribute information, must be reliable, stable, and lightweight as well. As a result, MQTT protocol came to solve all that problems, which was invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link). Nowaday, it is the most commonly used messaging protocol for the Internet of Things (IoT).

## 1.2. Literature review:

### 1.2.1. MQTT Protocol:

“MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.“

Citation from the official MQTT 3.1.1 specification. [1]

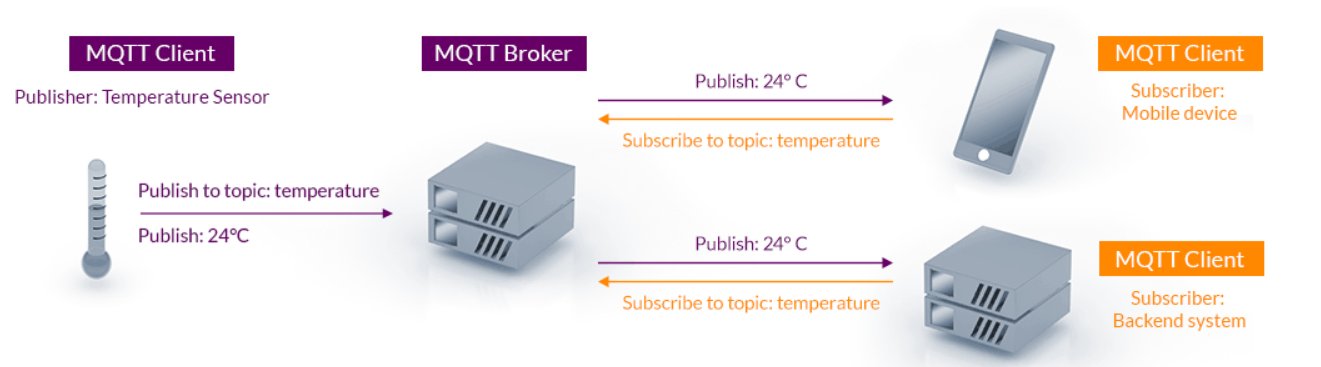


Figure 1. MQTT Architecture (Pub/Sub Model) [2]

MQTT protocol follows publish/subscribe model, which is also a client-server architecture base on TCP/IP. In MQTT, it defines MQTT Client (client) and MQTT Broker (server). When a Client want to send its information to another Client, the data is not transmitted directly between 02 devices. Instead, the Client will connect and send the information to Broker – that action is called Publishing and the Client is called Publisher. After that, Broker will pass that data to the Client - which asks for that information – the action is called Subscribing and the Client is called Subscriber.

That means in MQTT protocol, every Clients do not need to know each other, they just need to deal with Broker and Broker will try to manage any communications between Clients.

Based on concept, the basic features of MQTT that satisfy the requirements of IoT applications, but for more insight, we will compare it with a very popular protocol is HTTP. Both are built on the top level of TCP/IP, the most popular data transmission protocol recently. They also have the same Client-Server structure, as shown in the table below:

|  |  |  |
| --- | --- | --- |
|  | MQTT | HTTP |
| Architecture | Publish subscribe | Request response |
| Command targets | Topics | URLs |
| Underlying Protocol | TCP/IP | TCP/IP |
| Secure connections | TLS + username/password (SASL support possible) | TLS + username/password (SASL support possible) |
| Client observability | Known connection status (holding connection) | Unknown connection status |
| Messaging Mode | Asynchronous, event-based | Synchronous |
| Message queuing | The broker can queue messages for disconnected subscribers | Application needs to implement |
| Message overhead | 2 bytes minimum. Header data can be binary | 8 bytes minimum (header data is text - compression possible) |
| Message Size | 256MB maximum | No limit but 256MB is beyond normal use cases anyway. |
| Content type | Any (binary is normal) | Text (Base64 encoding for binary) |
| Message distribution | One to many | One to one |
| Reliability | Three qualities of service (QoS) | Has to be implemented in the application |

Table 1. Comparation about features between MQTT and HTTP Protocol [3]

Besides, by trying to connect to Google IOT Core, we can measure performance of 02 protocols:

|  |  |  |
| --- | --- | --- |
|  | MQTT Bytes | HTTP Bytes |
| Establish connection | 5572 | 2261 |
| Disconnect | 376 (Optional) | 0 |
| For each message published | 388 | 3285 |
| Sum for 1 message | 6336 | 5546 |
| Sum for 10 messages | 9829 | 55460 |
| Sum for 100 messages | 44748 | 554600 |

Table 2. TCP message overhead [3]

The MQTT Procol definetly consumes less bytes then HTTP, that means the application can save a lot of bandwidth and enery.

Finally, we measure the response time:

|  |  |  |
| --- | --- | --- |
| No. messages in a connection cycle for MQTT | MQTT avg. response time per message (ms) (QoS 1) | HTTP avg. response time per message (ms) |
| 1 | 113 | 289 |
| 100 | 47 | 289 |
| 1000 | 43 | 289 |

Table 3. Response time per message [3]

Arcording the measurement above, we can confirm that the features of MQTT Protocol, are mostly more suitable with IoT applications than HTTP.

### 1.2.2. Conception of MQTT Broker:

In this project, author will implement the most important, the heart of a MQTT network – MQTT Broker. Briefly, MQTT Broker is the place where all the packets in the network have to be stopped by. In addition, it has to handle concurrently many messages and tasks: receiving all messages from clients, checking the packet, routing the message to the destinations… In other words, MQTT Broker is a hub, that all the main jobs of MQTT network are implemented. To conclude, if a MQTT network can operate well or not, it depends mostly on their Broker.

Concisely, MQTT Broker need to handle these basic tasks:

* Receiving all messages,
* Filtering the messages,
* Determining who is subscribed to each message,
* Sending the message to these subscribed clients

Otherwise, there are some other tasks of a MQTT Broker:

* Holds the session data of all clients
* Authentication and authorization of clients

Therefore, it is important that your broker is highly scalable, integratable into backend systems, easy to monitor, and (of course) failure-resistant. (\*)

### 1.2.3. Topic review:

MQTT review

There are numerous documents such as: papers, open sources, blogs… which have discussed about MQTT Protocol, so it is a bit challenge for researcher in begining. Fortunately, there is a website has been developed to collect and sumarize mostly information you want to know about MQTT Protocol, from specifications, standards, or applications and practices. However, it tends to show the pratical aspects, not academic view. For this thesis, it meets this thesis’s need and will be referred:

<https://mqtt.org/> [4]

Next, navigate deeply into website:

<https://mqtt.org/software/> [4]

This is a fairly complete and reliable aggregator of MQTT concepts and applications, softwares, and libraries in progress (both open sources and comercial products). Obviously, individual projects with limited resources and applications are not included, which are not widely publicized, so this will be an important reference and will be cited throughout this thesis.

Through the aggregate from the website, it shows that developers have developed a lot of MQTT Broker libraries – open sources for developers or enthusiasts can embed into their devices to make a MQTT network, users can even develop the desired features. Besides, there are MQTT Broker softwares – which can be installed on devices to serve as servers for the entire system, or like MQTT Broker Cloud, cloud-based service packages for users to register and run online, almost instantly, with no installation required.

Chúng ta có thể phân loại các loại mô hình MQTT Broker trên

Among them, MQTT Dashboard is the most significant. It is a form of webserver, embedded with the core MQTT Broker and has user interfaces, so that we can register and monitor the devices. That leads to its advantages as follows:

* Easy to use and no installation required
* Visualization
* No hardware required (for users)

# 2. Introduction of thesis:

## 2.1. The approach of this thesis:

Based on the idea of ​​developing such a dashboard, the author will implement an MQTT dashboard with MQTT Broker written in Scala language, a powerful language and supported by the AKKA toolkit, with a set of tools powerful processing engine, and very good at scalability and self-healing. As mentioned in the introduction to MQTT Broker, that will be helpful in the role of a broker, which always having to connect to many devices, handle many processes and must always be stable, but can be extended easily.

On contrast, the development of user functions will also be based on Play Framework - a framework built on top of the AKKA model, that helps it inherit the power of AKKA and this is also the framework used for the Scala language.

Thanks to the developer Butaji, who has released an open source (MIT license) from the website:

<http://jetmq.net/> [5]

This project was developed 7 years ago and is almost impossible to run at the present time, when the libraries are no longer supported and downloaded from the internet. Howerver, it also provides logical directions for this project, especially the core MQTT Broker.

## 2.2. Thesis aims and objectives:

#### Aim:

To develop a MQTT Broker Core using Scala and AKKA toolkit.

To develop a Dashboard using Scala and Play Framework.

To acquire and implement AKKA model in MQTT Protocol.

To achieve the reliability of the program in real IoT applications.

#### Objectives:

* Developing a MQTT Broker Program.
* Developing a Website with User Functions and User Interface interacts with MQTT Broker Core to monitor the devices in network.
* *Measure performance of the website. (considering)*
* Programing language: Scala.
* Model/Framework: AKKA & Play.

#### Diagram:

(TBD)

#### 

Figure 2. Illustrating images of dashboard

To compare with previous projects, this is a new approach, base on Scala language and AKKA model could be a good perspective for other projects using Scala or AKKA.

# 3. Tools and Methods:

The combo Scala, Akka toolkit and Play Framework will come together to implement the ideas in this project.

3.1. Introduction of Scala:

Scala is a strong statically typed general-purpose programming language which supports both object-oriented programming and functional programming. Designed to be concise,[9] many of Scala's design decisions are aimed to address criticisms of Java.[7]

Scala source code can be compiled to Java bytecode and run on a Java virtual machine (JVM). Scala provides language interoperability with Java so that libraries written in either language may be referenced directly in Scala or Java code.[10] Like Java, Scala is object-oriented, and uses a syntax termed curly-brace which is similar to the language C. Since Scala 3, there is also an option to use the off-side rule (indenting) to structure blocks, and its use is advised. Martin Odersky has said that this turned out to be the most productive change introduced in Scala 3.[11]

Advantages:

* Easy to Pick Up
* Pretty Good IDE Support
* Scalability
* Highly Functional
* Code cleaner

3.2. Introduction of AKKA:

Nowaday, AKKA is a framework running on the Java Virtual Machine (JVM) toolkit, it is the choice for companies like Verizon, Intel, PayPal, Norwegian Cruise Lines and Samsung when it comes to building scalable, resilient and resource efficient applications in Java or Scala.

Advantages:

* Event-driven: With Actors, requests can be done asynchronously and non-blocking operations exclusively
* Scalable: By message passing and location transparency, adding nodes without having to modify the code is possible.
* Resilient: Akka fault tolerance is to encounter errors a self-healing system.
* Responsive: Akka’s non-blocking, message-based strategy helps to give quick feedback to request.

## 3.3. Introduction of Play framework:

Play – a very simple and easy to use Web Framework. It was created to help you make changes faster and easier, with less pressure on you.

Thanks to its seamless and easy-to-use UI, not to mention numerous features for optimizing your computer's resources – CPU, RAM – making it easy to scale the software you're writing. It is designed for developers to develop modern web and mobile apps.

Play is built on top of the AKKA toolkit – a very popular open source toolkit that runs on top of the JVM, and it comes with the same basic tools and features, but works in a more user-friendly way that allows you to easily write , design and test the apps you are developing. Many developers have been faithful to it, and emphasize how it has improved their productivity, thanks to its simplicity and ease of use.

Advantages:

* Huge productivity improvement
* Workflow is easy
* Flexible Tool
* Everything works from the moment you start
* Effective resource management unit
* Easy to scale software

# 4. Developing of MQTT Broker:

## 4.1. Idea of Implementation:

Based on the decision to use the Scala language and the AKKA framework for product development, the project will delve into the direction of using these tools and developing the corresponding logics for the MQTT Broker program.

### 4.1.1. AKKA Toolkit:

Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala. The significant feature of AKKA is Actor Model.

The advantages of this model are:

Those are also the reasons:

### 4.1.2. AKKA Actors:

The Actor Model consists of a set of actors, which are isolated, concurrent, and solely interacted through a network with a transparent message-passing technique [6]

The model was designed to be a general paradigm for concurrent computation in a highly concurrent and parallelizable distributed environment. At a higher level, the model is straightforward and allows for a high degree of parallelism. An Actor is the basic unit of the computation model. An Actor is a type of entity that can communicate with other actors via network messages.

Đây cũng chính là ý tưởng chính của việc phát triển AKKA toolkit

### 4.1.3. AKKA Stream:

Stream Reactive…

Đảm bảo các data được truyền trong các stream, đảm bảo tính immutable của data

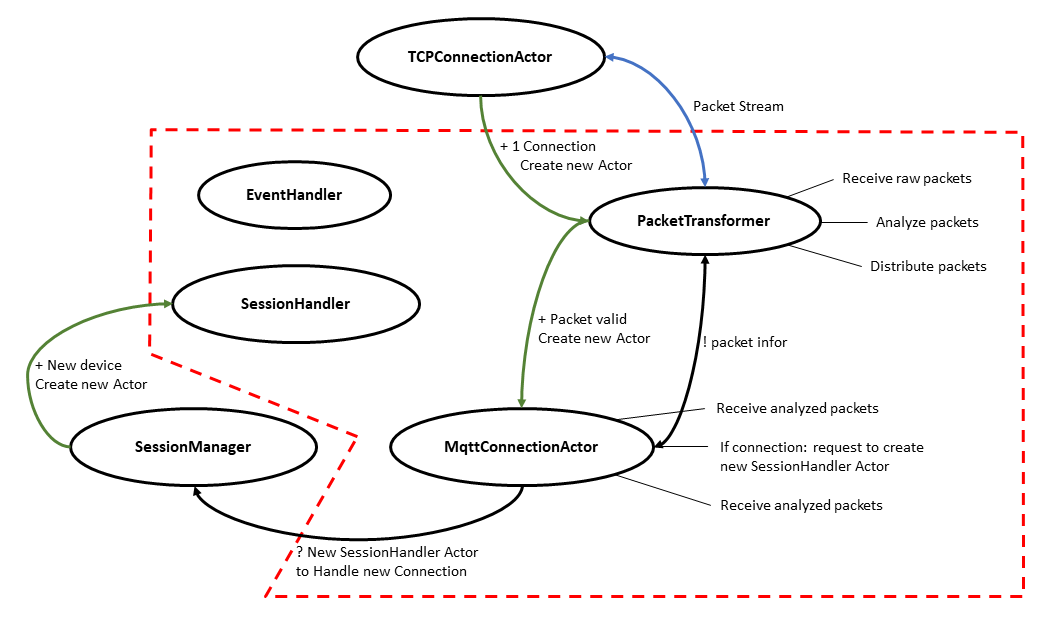
Ứng dụng trong gói tin được truyền dẫn trong các node nội bộ cũng như với các thiết bị

## 4.2. Software design of programs:

### The previous section discussed AKKA Toolkit and why AKKA was chosen to develop MQTT Broker for this project. In this section, we will clarify the ideas for implementing the MQTT Broker program.

### 4.2.1. Actors Model of Program:

(Insert hình ở đây)



The figure above is the project's official model, displaying the functions as well as communication between actors via the Actor model - Hierarchy.

There is a root actor, which serves as the root for all actors in the system. Because of the centralized management but independent handling of connections, groups of Actors will be born to handle each connection. Actors will be managed using the Hierachical model, which aids in the accurate management of executions and errors, resulting in high system stability. For more detail, the explanation below

* **TCPConnectionHandler**: It handles all connection requests and communicates with peripheral devices by opening a socket and listening; for each valid connection, it spawns clusters of nodes (or child Actors) that connection.
* **PacketTransformer**: Every time a connection is established, this Actor is born to be in charge of delivering packets from the connection to other Actors so that they can handle tasks or return ACK packets to peripheral devices. The unique technique here is to use AKKA Stream to transmit data in streams and queues rather than sending messages directly; this allows for more accurate and reliable packet delivery and management.
* **MqttConnectionActor**: This Actor will be spawned at the same time as the Actor PacketTransformer; it will receive packets from the PacketTransformer and analyze them; if it is a valid Connection Packet, it will request (ask -?) to create an Actor handles this Connection; otherwise, it will forward the packet to Actor SessionHandler for further processing. Cases that are invalid will be rejected.
* **SessionManager**: The MqttConnectionActor will then check to see if an Actor SessionHandler is already handling this Connection; if not, the SessionManager will create a new Actor SessionHandler to handle this Connection.
* **SessionHandler**: This Actor is in charge of analyzing and classifying packets sent over the Connection, including checking the Header and Control Flags ,etc. After classifying the packet and gathering the required information, the Actor will send (tell - !) the data to Actor EventHandler, which will handle the tasks required by each packet.
* **EventHandler**: once the packets have been classified, they will be sent to this Actor who will execute the packet's request and return the result to the SessionHandler.

The table below summarizes the model's Actor descriptions as well as their interactions with other Actors.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Actor | Description | Related Actor |
| 1 | TCPConnectionHandler | This actor handles TCP connection | -> SessionManager -> EventHandler -> PacketTranformer |
| 2 | PacketTransfomer | This actor handles packets which are sent via TCP and filter/decode/transfer them between other actors (inbound) or client (outbound) | <- TCPConnectionHandler -> MqttConnectionActor |
| 3 | StreamHandler | Support PacketTransformer to handle the stream | <- PacketTransformer |
| 4 | MqttConnectionActor | This actor handles packets inbound -> Define FSM for 02 states: | <->PacketTransformer |
| 5 | SessionHandler | This is handles sessions for each connection -> FSM for 02 states | -> EventHandler <- MqttConnectionActor |

## 4.3. Coding:

In the preceding section, we used the Actor Model to model the program for MQTT Broker. Following that, we will apply this model to a specific program.

However, quoting and explaining thousands of lines of code is time-consuming and unnecessary. As a result, to ensure the report's conciseness, the author will only point out and analyze relatively complex programming techniques, and describe the program's explanations with a few specific code blocks. explained in the source code comments

*To view full source code, please access the link* [*github*](https://github.com/nicolas-le-petit/MQTT_Broker_AkkaModel/tree/Developing)[7]*.*

### 4.3.1. Finite State Machine (FMS):

FMS is the programming method used throughout (Finite State Machine). This is a tried-and-true method with high reliability.

A FSM can be described as a set of relations of the form:

State(S) × Event(E) → Actions (A), State(S’)

These relations are interpreted as meaning:

If we are in state S and the event E occurs, we should perform the actions A and make a transition to the state S’.

Thanks to AKKA Toolkit, implemeting FSM is more concise and reliable.

## 4.4. Testing and Result:

The intriguing aspect of AKKA Toolkit is that it provides both testing functionality for our applications, allowing us to test functions independently while also testing a group of functions or even the entire system with various scenarios.

Kết quả dưới đây thể hiện các bài test

# 5. Developing of User Functions:

## 5.1. Block diagram:

## 5.2. Software design of programs:

## 5.3. Explanation of Coding:

## 5.4. Testing and Result:

# 6. Developing of User Interface:

## 6.1. Block diagram:

## 6.2. Software design of programs:

## 6.3. Explanation of Coding:

## 6.4. Testing and Result:

# 7. Intergration of MQTT Dashboard:

## 7.1. Testing and Result:

## 7.2. Conclusion:

# References

|  |  |
| --- | --- |
| [1] | T. H. Team, "Introducing the MQTT Protocol - MQTT Essentials: Part 1," 12 January 2015. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/. [Accessed 15 May 2022]. |
| [2] | MQTT Community, "MQTT: The Standard for IoT Messaging," Updating. [Online]. Available: https://mqtt.org/. [Accessed 11 May 2022]. |
| [3] | I. Craggs, "MQTT Vs. HTTP for IoT," 16 May 2022. [Online]. Available: https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/. [Accessed 28 May 2022]. |
| [4] | M. Community, "MQTT Software," Updating. [Online]. Available: https://mqtt.org/software/. [Accessed 15 5 2022]. |
| [5] | Butaji, "JetMQ," 28 November 2015. [Online]. Available: http://jetmq.net/. [Accessed 15 May 2022]. |
| [6] | A. G, Actors: a model of concurrent computation in distributed systems, The MIT Press Classic, 1986. |
| [7] | T. Tan Dung, "MQTT\_Broker\_AkkaModel," 30 June 2022. [Online]. Available: https://github.com/nicolas-le-petit/MQTT\_Broker\_AkkaModel/tree/Developing. [Accessed 30 June 2022]. |
| [8] | O. Organization, "MQTT Version 3.1.1 Plus Errata 01," 10 December 2015. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. [Accessed 11 May 2022]. |
| [9] | N. S. Gill, "Scalable and Responsive Applications with Akka | Quick Guide," 7 February 2022. [Online]. Available: https://www.xenonstack.com/insights/akka-framework-tools. [Accessed 20 May 2022]. |

[Figure 1. MQTT Architecture (Pub/Sub Model) [2] 4](#_Toc104236209)