Cloud-based services for the Internet of Things

O. Jukić, I. Špeh, I.Heđi
Virovitica College, ICT department, Virovitica, Republic of Croatia
{ oliver.jukic | ivan.speh | ivan.hedi } @vsmti.hr

***Abstract* - In this paper, cloud-based services aimed for the connectivity, monitoring, and management of the IoT devices are presented. To set up a network with a large number of devices in the constrained environment can be a challenge. Also, collecting, storing and analyzing data generated from sensors attached to the devices most often requires developing custom-made applications which imply time and cost consumption. Several cloud providers are offering IoT services that unite needed features into full solution offering connectivity between devices and the cloud, processing of data sent from devices and interaction with connected devices through the application. One of them is AWS IoT (Amazon Internet of Things) which is fully scalable, reliable and simple to use. This platform can collect data from a large number of different devices and connect them to endpoints for other tools, allowing a developer to tie received data into the independent application. Other aspects, like security, resource management, integration and centralized management are also covered. In this paper, we will evaluate the performance of the AWS IoT web services by connecting small, single-board computers, like Raspberry Pi through different scenarios.**

Keywords – Internet of Things; IoT Architecure; Amazon Web Services IoT; MQTT; Raspberry Pi

I. INTRODUCTION

The Internet of Things (IoT) is the computing concept that describes interconnection of everyday objects that deliver Information over the Internet. Everyday growing number of objects has promoted the Internet of Things protocols and technologies as one of the most commonly used in the modern systems. Although the concept of the IoT is originated more than ten years ago, it is still one of the most widespread topics across a range of industries, including traffic industry, manufacturing, military, healthcare, sports, education – in almost every aspect of human life. IoT objects refer to a wide variety of devices which are most often equipped with electronic circuits and sensors [1]. These devices, implemented in IoT solutions, made Internet sensory thereby achieving improved tracking and analyzing systems.

There are several challenges that need to be overcome during the implementation of the IoT technology. A typical IoT solution includes different devices used for collecting, producing, analyzing and storing data. These devices may differ by their hardware and software platform, but all must meet the requirements which can be breaked up into these components: memory storage, processor, power source and power management, wireless connectivity and ability to connect sensors.

Connecting devices to the Internet enables the devices to communicate with each other and with cloud services and applications. For the purpose of the connectivity, there are a variety of communication standards and protocols used, of which most widespread are IEEE 802.15.4, Internet Protocol version 6 (IPv6) and IPv6 over Low Power Wireless Personal Area Network (6LoWPAN) [2]. Regardless the specific communication protocol used to deploy IoT solution, all the IoT devices should make their data available to the other IoT object, or application. This can be achieved by connecting devices to the cloud using Application Programming Interfaces with built-in functions for end-users [3]. Gerber in [4] discusses strategies that can be applied during the planning of data-driven IoT architectures. These strategies simplify development, manage complexity, increase scalability and flexibility and can be breaked up into following: adopting a layered architecture, implement security, automate operations, ensure interoperability and follow a reference architecture. Leading mentioned challenges, requirements and strategies, we have built a simple IoT solution which is described in following chapters.

II. ARCHITECTURE

Implementation of the IoT solution requires unprecedented collaboration, coordination, and connectivity for each piece in the system, and throughout the system as a whole [5]. All objects must work together, be integrated with other devices, and communicate and interact seamlessly with connected systems and infrastructures. To manage such a complexity of the implementation of an IoT solution, a multi-tiered architecture is used. This modular approach enables to develop and maintain each tier as an independent module, on a separate platform. As written in [6] "a typical IoT solution is characterized by many devices (i.e., things) that may use some form of gateway to communicate through a network to an enterprise back-end server that is running an IoT platform that helps integrate the IoT information into the existing enterprise.". Leading this, each IoT solution can be described as a three-tier architecture in which physical devices (e.g. device tier) over the gateway (e.g. control tier) communicates with the datacenter (e.g. cloud tier). A detailed survey on domain-specific IoT architectures can be found in [7].

The bottom layer of the architecture is the device layer. This is the starting point and a source of information implemented through various of different devices which are most often equipped with electronic circuits and sensors, software and network connectivity [8]. These devices are constrained in terms of power supply and size

and therefore, often programmed using microcontrollers. Software component of the device may include operating system suited for small constrained devices and a layer that provides access to the hardware features of the microcontroller such as flash memory, general-purpose input/output (GPIO) and serial interfaces. This layer also in charge for handling communication between upper layers and the devices through the drivers and communication protocols [6].

Control tier bring in control functionalities to the architecture. The main tasks of this layer are security, connectivity, transportation and aggregation of data received from the bottom layer and pre-processes it before sending it to cloud services and applications for further processing and analytics.

The Cloud tier includes the software infrastructure and services required to store, analyze and visualize data. This layer interacts with very large numbers of devices and gateways using different protocols and data formats, but then normalize it to allow for easy integration into the rest of the enterprise. Depending on the solution, it is possoble to trigger alerts or carry out actions in response. Also, it has to support device management through a central registry to identify the devices/gateways running in an IoT solution and the ability to provision new software updates and manage the devices. At the side of a storage, it must support the volume and variety of IoT data.

There is a growing number of cloud providers offering IoT specific services [9]. Industry partners such as Microsoft, Amazon, Google, and IBM have introduced IOT based platforms such as Microsoft Azure IoT, Amazon Web Services (AWS) IOT, Google Cloud Platform, and IBMBluemix Watson for effectively monitoring the remote applications [10].

According to [11], [12] and [13], top three IoT platforms in 2018 are:

- Amazon Web Services IoT,
- Microsoft Azure IoT,
- Google Cloud Platform,

In this paper, we will evaluate the performance of the top-rated AWS IoT platform by connecting small, single-board computer through different scenario (Fig. 1). AWS offers a suite of web services that can be combined [14] and make up a cloud computing platform [9]. At the end of 2017, AWS had more than 90 services, spanning a wide range, including compute, storage, networking, database, analytics, application services, deployment, management, mobile, developer tools and tools for the Internet of things [7].

III. AMAZON WEB SERVICES IOT

Comparing to other platforms AWS IoT platform serves solely as a platform for providing bi-directional communication between IoT objects and the cloud [15]. It consists of the following components:

A Thing – thing are all devices, applications, or physical objects that are supported by AWS IoT cloud application.

The device gateway is an entry point for IoT objects connecting to AWS. The main responsibilities of the device gateway are to manage connections and security. At the side of the connectivity, Device Gateway supports Message Queue Telemetry Transport (MQTT), MQTT over Web Sockets, MQTT over the Secure WebSocket and HTTP protocols. IoT devices can communicate even if they are not using same protocols.

Message Broker mediates communication between MQTT client applications and can handle up to thousands connected MQTT clients. The main responsibilities of a broker are receiving messages, filtering messages, deciding who is interestes in messages and then sending messages to them.

Authentication and authorization – Amazon IoT connects devices to services and other devices with a secure way [9]. Each device must be identified through the process of authentication to get access to resources. After device is authenticated, it can perform actions which are defined through the process of authorization.

The Registry is responsible for assigning a unique identity to each device registered in the AWS IoT, no matter what type it is or how it connects.

Device Shadows represents a persistent, virtual version of each device that includes latest state so that other devices or applications can evaluate messages and communicate with the device.

The Rules Engine makes it possible to build IoT applications that gather, process, analyze and act on data generated by connected devices at global scale without having to manage any infrastructure.

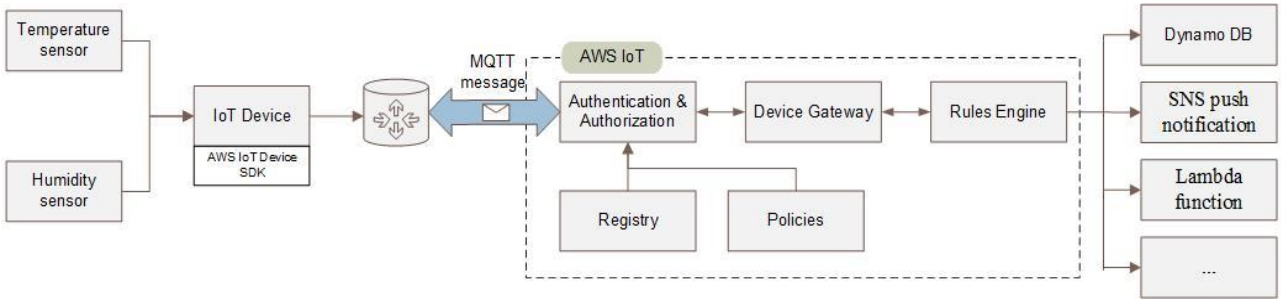

Figure 1. AWS Internet of Things Components

IV. MQTT - IOT COMMUNICATION PROTOCOL

IoT is characterized by communication between machines (M2M communication). The interconnected things are often located in isolated areas where connection to the Internet is realized over the cellular network or slow DSL connection. This characteristic can be recognized in rural areas where most of the facilities are built. Unreliability is one of the main disadvantages of such networks, which reflects high packet loss rate [16]. There are several protocols proposed for M2M/IoT communication with a focus on mentioned constrained environments. The specific characteristics of proposed protocols can simplify the design and the operation of IoT application because a large portion of error handling can be done by the protocols [16]. Most frequently adopted protocols are MQTT and CoAP (Constrained Application Protocol).

MQTT is a machine-to-machine (M2M)/Internet of Things connectivity protocol. It was designed as an extremely lightweight broker-based publish/subscribe messaging protocol for small code footprints (e.g., 8-bit, 256KB ram controllers), low bandwidth and power, high-cost connections and latency, variable availability, and negotiated delivery guarantees [17]. Communication between devices and AWS IoT relies on this protocol.

The MQTT protocol relies on a messaging server following the hub and spoke model of Message Oriented Middleware (MOM) [18]. In such architecture, a client needing data (end user devices) registers its interests with a server, an MQTT broker (central server). The client producing data (publisher) sends data to a server and this server forwards the data to the subscriber. One of the major advantages of this architecture is the decoupling of the clients needing data and the clients sending data, i.e. temperature sensor nodes need not know the identities of clients that are interested in data and conversely [19]. So, the publishers and subscribers do not need to be familiar with each other and do not need to participate in the communication at the same [20]. It is intended for devices with limited power and memory capabilities, where the network is expensive, has low bandwidth or is unreliable.

MQTT minimizes network bandwidth and device resource requirements while attempting to ensure reliability and delivery. This approach makes the MQTT protocol particularly well-suited for connecting the machine to machine (M2M), which is a critical aspect of the emerging concept of an Internet of Things [21]. This protocol has been applied in a large number of different embedded systems. For example, oil and gas companies use it to monitor oil pipeline thousands of miles away; hospitals use this protocol to track patient's condition. Facebook uses this protocol for messaging applications. Fig. 2 shows proposed a model of MQTT protocol which runs on TCP/IP connection.

Using MQTT over Web Sockets every browser can be MQTT publisher or subscriber. As long as the browser is subscribed to the specific message, it will keep receiving messages, anywhere in the world. Messages which are retained on the server are delivered when a client subscribes to one of the topics instantly.

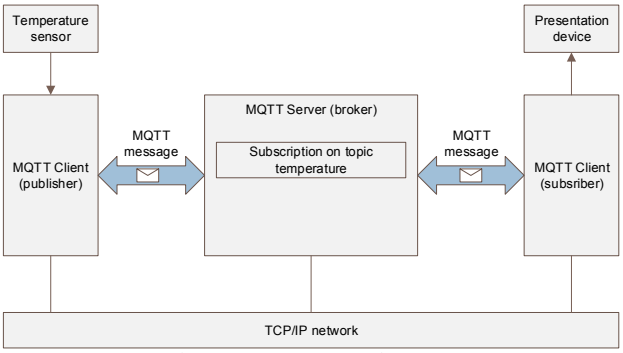

Figure 2. MQTT protocol

V. USE CASE

Using three-tier architecture, we've implemented a simple IoT solution in a lab environment which encompasses all three layers. The main goal is to collect data using sensors and single board computer which is connected to AWS IoT. Using AWS IoT Device SDK and MQTT protocol, the data is sent to AWS IoT.

On the device tier, a Raspberry Pi Model B is used. This is the third generation of the credit-card sized single board computer from the Raspberry family. Main differences between RPi model 3 and prior versions are integrated wireless module (802.11n), integrated Bluetooth module (Bluetooth 4.1) and 1.2GHz 64-bit quad-core ARMv8 CPU. Using this type of RPi there is no need for additional modules for connecting end device to the network.

The Raspberry Pi itself doesn't come with the pre-installed operating system. There are a number of operating systems designed specifically for the Raspberry Pi, available on the Raspberry official web page, of which the most popular are Raspbian, Windows IOT Core, RISC OS. To make a Raspberry run, we used Raspbian, an official operating system for the Raspberry Pi which is based on Debian and optimized for the Raspberry Pi hardware.

The next step is to choose external hardware which will be attached to the Raspberry Pi's GPIO pins. There are a variety of different low-cost and simple to use sensors which can be attached to the Raspberry Pi. For the purpose of this paper, a temperature sensor DS18B20 is used (Fig. 3). This is a commonly used digital temperature sensor featured with small size, low-cost hardware, strong anti-interference capability and high precision. It provides 9-bit to 12-bit Celsius temperature measurement and communicates over a 1-wire bus that by definition requires only one data line (and ground) for communication with Raspberry [10].

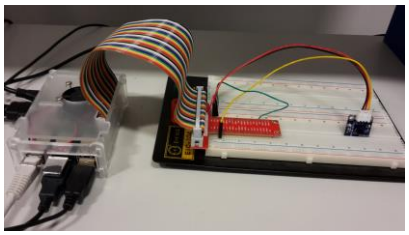

Figure 3. DS18B20 sensor connected to Raspberry Pi in lab environment

A. *Registry*

In order to connect the device to AWS IoT, a device must be registered in the Registry. The Registry gives a unique identity to each device regardless of the type of device or which communication protocol is the device using. Devices connected to AWS IoT are represented by the thing in the registry. Each thing is uniquely identified by Amazon Resource Name (ARN):

arn:partition:service:region:account:resourcetype/resource

General format for ARN consists of following parameters:

- partition - The partition that the resource is in. For standard AWS regions, the partition is aws.
- service - The service namespace that identifies the AWS product, for example Amazon S3, DynamoDB or AWS IoT
- region - The region the resource resides in.
- account - The ID of the AWS account that owns the resource
- resourcetype/resource - indicator of the type of resource followed by the resource name itself.

For example:

arn:aws:iot:us-east-2:682503558106:thing/Rpi_1

Some parts of the ARN (resource name and region) will be later used in creating MQTT connection.

The Registry determines an identity for objects and tracks metadata such as the devices attributes and conditions as presented in following portions of code.

```
$ aws iot list-things
{
    "things": [
       {
            "thingTypeName":"tempSensor"
            "attributes": {
                "model": "DS18B20"
            },
            "version": 1,
            "thingName": "RaspberryPi_1"
        }
    ]
}
```

B. *Authentication and authorization*

To enable access to AWS IoT resources, all devices must be authenticated to the AWS IoT platform and authorized to carry out actions. The AWS IoT platform uses mutual authentication and encryption for all types of requests, so to exchange data between device and AWS IoT, identity must be proved. There are following methods that can be applied to prove identity:

- Signature version 4 – AWS method of authentication which can be used in connections using HTTP protocol or Web Sockets,
- X.509 certificate-based authentication using digital certificates that use X.509 public key infrastructure standard to associate a public key with an identity contained in a certificate. This type of authentication is used in HTTP or MQTT connection,
- Customer created token-based authentication which can be used in HTTP connection or Web Sockets.

Raspberry Pi will communicate with AWS services over MQTT protocol, so an appropriate certificate must be issued, which can be done through AWS IoT Management Console. There are three options to create a certificate: one-click certificate creation, creation with certificate signing request (CSR) or using own certificate signed by own trusted CA. The first option will generate a certificate, public key, and private key using AWS IoT's certificate authority. Certificate and private key needs to be downloaded to the end-device.

After the device is registered to AWS IoT and identity has been applied, the process of authentication is done. Still, it remains to set up authorization – a process of verifying which actions a device can perform. Authorization in AWS IoT is performed through AWS IoT policies. Policy in AWS IoT is a JSON structure containing one or more policy statements.

Each statement has three parameters: an Effect, an Action, and a Resource. Depending on the effect value, the action will be allowed or denied. The action part specifies which actions to the device the policy is allowing or denying. The Resource specifies which devices or resources are affected. The following policy grants all devices permission to connect to the AWS IoT message broker, and publish a message to a specific topic.

```
{
   "Version": "2018-02-19",
    "Statement": [{
        "Effect": "Allow",
        "Action":["iot:Connect", "iot:Publish"],
        "Resource": ["*"]
    }]
}
```

C. *AWS IoT device SDK*

The AWS IoT device SDK is a set of tools using which is possible to connect hardware device to AWS IoT. The AWS IoT Device SDK enables end devices to connect, authenticate, and exchange messages with AWS IoT Core using the MQTT, HTTP, or Websocket protocols. The AWS IoT Device SDK supports C, JavaScript, and Python, and includes the client libraries, the developer guide, and the porting guide for manufacturers.

To connect our device with AWS IoT we used JavaScript/Node.js SDK. The aws-iot-device-sdk.js package contains libraries for JavaScript applications which access the AWS IoT Platform via MQTT or MQTT over the Secure WebSocket Protocol. It can be used in Node.js environments as well as in browser applications.

Using device class from AWS IoT device SDK for JavaScript a secure connection to the AWS IoT platform is accomplished. This class provides features to simplify handling of communication. Following portion of code is example of instantiating a new device with parameters.

```
var awsIot = require('aws-iot-device-sdk');

var device = awsIot.device({
   keyPath: <PrivateKeyPath>,
  certPath: <CertificatePath>,
    caPath: <RootCACertificatePath>,
  clientId: <UniqueClientIdentifier>,
      host: <CustomEndpoint>
});
```

After device has been connected, it can start to send messages to given topic. A temperature values from DS18B20 temperature sensor are sent along with Rpi_1 topic.

```
setInterval(function){
   var temperature = ds18b20.temperatureSync();
   var date = new Date();

   device.publish('Rpi_1',JSON.stringify({
       temperature:temperature,
       timestamp: date.getTime();
   }));
}
```

D. *Rules*

After data has been received on the AWS IoT, it is possible to interact with other AWS services using AWS rules. An AWS IoT rule can be made in AWS management console and it consists of a SQL SELECT statement, a topic filter, and a rule action. Devices send information to AWS IoT by publishing messages to specific MQTT topics. By defining a rule it is possible to evaluate received messages and specify what to do when a message is received. The rule is triggered when an MQTT publisher sends a message on a topic that matches the topic filter, the rule will. Through the rule actions it is possible to extract information from an MQTT message and send it to another AWS service, for example insert a message into the database, invoke Lambda function, send a message as an SNS push notification, or republish messages to an AWS IoT topic. On the image below, a screenshot of the management console is captured. There are three rules defined when Raspberry Pi publish a message on specific topic (Fig. 4). Executed rules can be monitored through management console (Fig. 8).

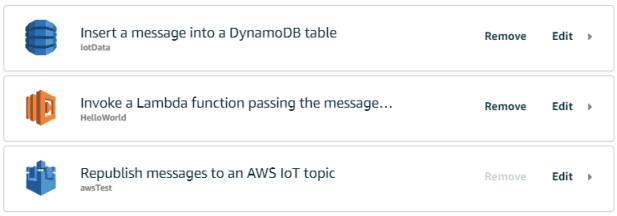

Figure 4. Rules in AWS IoT management console

E. *Management Console*

Using AWS IoT Device Management Console it is easier to organize securely, monitor (Fig. 5., 6., 7., 8.), and remotely manage IoT devices. Management Console is used to register devices individually or in bulk, manage permissions, organize devices into groups, monitor and troubleshoot device functionality. It is possible to remotely update the software running on devices even after they have been deployed.

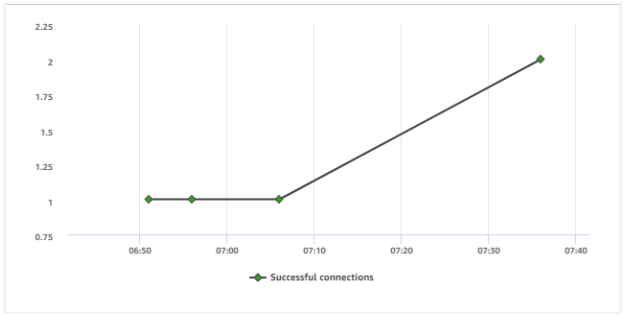

Figure 5. Successful connections

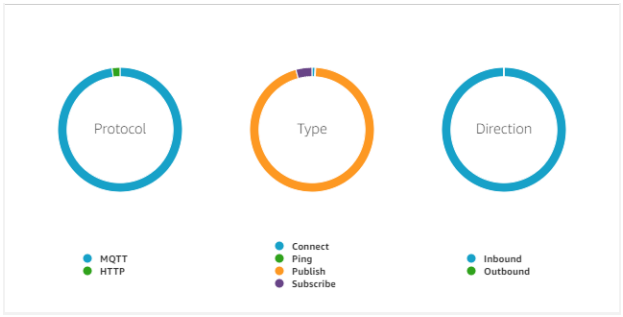

Figure 6. Messages

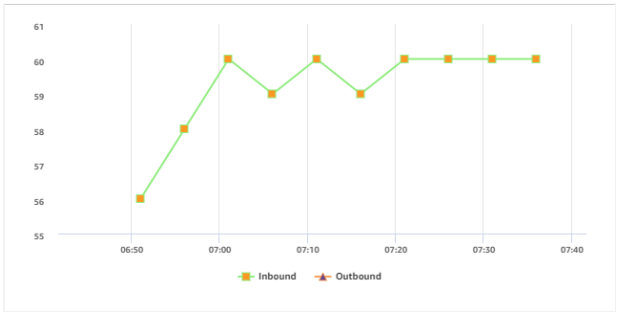

Figure 7. Messages published

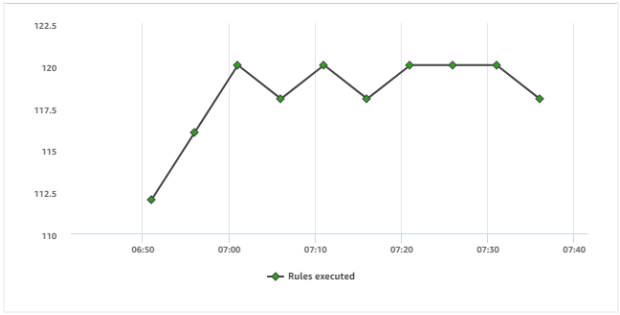

Figure 8. Executed rules in AWS IoT management console

VI. CONCLUSION

In this paper, we presented top-rated IoT cloud-based platform Amazon Web Services IOT by connecting small, single-board computer Raspberry Pi. We have briefly described architecture and components of the AWS IoT

through implementing a simple IoT solution in a lab environment which encompasses all three layers – device layer, a control layer and cloud layer. The main goal was to collect data using sensors and send data to AWS IoT web services using AWS IoT Device SDK and MQTT protocol. We've presented some of the web services available on the AWS IoT which can be invoked using AWS IoT Rules. Other aspects, like security, resource management, integration and centralized management are also covered.

LITERATURE

[1] F. Xia, L. T. Yang, L. Wang and A. Vinel, "Internet of Things", International Journal of Communication Systems, Vol. 25, pp. 1101-1102, 2012.

[2] S. Katsikeas, „A lightweight and secure MQTT implementation for Wireless Sensor Nodes", Technical University of Crete, June 2016

[3] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, J. AlonsoZarate „A Survey on Application Layer Protocols for the Internet of Things", Transaction on IoT and Cloud Computing 2015

[4] A. Gerber, "Simplify the development of your IoT solutions with IoT architectures", IBM developerWorks, 2017.

[5] "The Internet of Things: Manage the Complexity, Seize the Opportunity", Oracle, 2014.

[6] "The Three Software Stacks Required for IoT Architectures", white paper, Eclipse IoT Working Group, 2016.

[7] A - F. Xia, L. T. Yang, L. Wang and A. Vinel, "Internet of Things", International Journal of Communication Systems, Vol. 25, pp. 1101-1102, 2012.

[8] L. Dürkop, B. Czybik and J. Jasperneite, "Performance Evaluation of M2M Protocols Over Cellular Networks in a Lab Environment", Intelligence in Next Generation Networks (ICIN), pp. 70-75, February 2015.

[9] V. Gazis, M, Görtz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger and E. Vasilomanolakis. "A Survey of Technologies for the Internet of Things", International Wireless Communications and Mobile Computing Conference, pp. 1090-1095, August 2015

[10] Benchmark of MQTT servers, version 1.1, January 2015.

[11] D. Thangavel, X. Ma, A. Valera, H. Tan, C. K. Tan, "Performance Evaluation of MQTT and CoAP via a Common Middleware", Intelligent Sensors, Sensor Networks and Information Processing, April 2014

[12] S. K. Shriramoju, J. Madiraju and A. R. Babu, "An approach towards publish/subscribe system for wireless networks", International Journal of Computer and Electronics Research, Vol. 2., pp. 505-508, August 2013.

[13] V.Lampkin, W. T. Leong, L. Olivera, S. Rawat, N. Subrahamanyam, R. Xiang, "Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry", First Edition, September 2012

[14] https://internetofthingswiki.com/top-20-iot-platforms/634/

[15] https://www.devteam.space/blog/10-best-internet-of-things-iot-cloud-platforms/

[16] https://www.computerworlduk.com/galleries/data/best-internet-of-things-platforms-3635185/

[17] D. Kang, M. Park, H. Kim, D. Kim, S. Kim, H. Son, S. Lee, "Room Temperature Control and Fire Alarm/Suppression IoT Service Using MQTT on AWS", Platform Technology and Service (PlatCon), February 2017

[18] https://en.wikipedia.org/wiki/Amazon_Web_Services

[19] P.P. Ray, "A survey on Internet of things architectures", Journal of King Saud University – Computer and Information Sciences, pp. 1319-1578, October 2016.

[20] D.B. Andore, "AWS IOT Platform based Remote Monitoring by using Raspberry Pi", International Journal of Latest Technology in Engineering, Management & Applied Science (IJLTEMAS), October 2017.

[21] T. Pflanzner, A. Kertesz, "A Survey of IoT Cloud Providers", Information and Communication Technology, Electronics and Microelectronics (MIPRO), July 2016