Jurnal Elektro
p-ISSN: 1979-9780
e-ISSN: 2746-4288

Jurnal Elektro Vol.15 No.2 Oktober 2022
journal homepage: https://ejournal.atmajaya.ac.id/index.php/JTE

Implementasi IoT dengan ESP 32 Untuk Pemantauan Kondisi Suhu Secara Jarak Jauh Menggunakan MQTT Pada AWS

Calvin Austin[1], Melisa Mulyadi[2*], Sandra Octaviani[3]

[1,2,3] Program Studi Teknik Elektro, Fakultas Teknik
Universitas Katolik Atma Jaya Indonesia, Jakarta 12930, Indonesia

Article Info

Article history:

Received
02 12 2023

Accepted
12 12 2023

Keywords:
Amazon web services, cloud computing, database , ESP32, message queuing telemetry transport

Abstract

The development of the internet of things (IoT) creates many new innovations in the industrial sector aimed at increasing effectiveness. One of them is to monitor the condition of industrial process machines remotely as discussed in this study. This monitoring can be done using a device connected to the internet. The design of this system requires a microcontroller with an ESP32 wifi module as a data receiver and sender. The data sent is data from the temperature sensor. The data is in the form of simulation data generated from the program and has a random number. Machine-to- server data communication uses the Message Queuing Telemetry Transport (MQTT) protocol. The entire machine-to-server, server-to-server communication system will be carried out in the cloud using a cloud computing platform, namely Amazon Web Services (AWS). The test results show that the payload and temperature data sent from the microcontroller can be stored in the database. To see the reliability of the system, two Normal and Stress Tests were carried out, with a success percentage of 100% fo r data storage to the database on two tests and 16.67% failure in sending data on the Stress Test. The two tests were arranged under different conditions.

Info Artikel

Histori Artikel:

Diterima:
02 12 2023

Disetujui:
12 12 2023

Kata Kunci:
Amazon web services, cloud computing, ESP32, database, message queuing telemetry transport

Abstrak

Perkembangan internet of things (IoT) menciptakan banyak inovasi baru dalam bidang industri yang ditujukan untuk meningkatkan efektivitas. Salah satunya untuk melakukan pemantauan kondisi mesin proses industri dari jarak jauh seperti yang dibahas pada penelitian ini. Pemantauan tersebut dapat dilakukan menggunakan perangkat yang terhubung dengan internet. Perancangan sistem ini membutuhkan satu buah mikrokontroler dengan modul wifi ESP32 sebagai penerima dan pengirim data. Data yang dikirimkan adalah data dari sensor suhu. Data tersebut berupa data simulasi yang dibangkitkan dari program dan bernilai acak /random number. Komunikasi data yang dilakukan antara machine-to-server menggunakan protokol Message Queuing Telemetry Transport(MQTT). Seluruh sistem komunikasi antara machine-to-server, server-to- server akan dilakukan di cloud menggunakan platform cloud computing yaitu Amazon Web Services(AWS). Hasil pengujian menunjukkan data payload dan suhu yang dikirimkan dari mikrokontroler dapat disimpan dalam database. Untuk melihat reabilitas sistem dilakukan dua buah pengujian Normal dan Stress Test, dengan persentase keberhasilan sebesar 100% untuk penyimpanan data ke database pada dua buah pengujian dan 16,67% kegagalan pengiriman data pada Stress Test. Pengaturan pada dua buah pengujian dilakukan dengan kondisi yang berbeda.

1. PENDAHULUAN

Internet of Things (IoT) memberikan kemampuan pada mesin, perangkat, sensor dan manusia untuk terhubung dan saling berkomunikasi satu sama lain melalui media internet. IoT memberi kemudahan untuk pengumpulan dan pengukuran data, sehingga manusia menjadi dipermudah untuk memperoleh informasi, misal kondisi cuaca, titik kemacetan, dan lain- lain. Di bidang industri, perkembangan IoT dapat digabungkan dengan teknologi otomasi sehingga pengendalian dan pemantauan proses industri dapat dilakukan secara *online* dan semua proses menjadi lebih terpusat atau yang disebut industri 4.0.

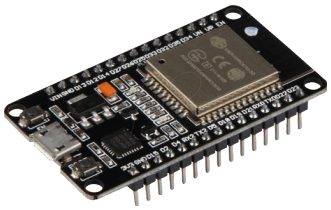
*Corresponding author: Melisa Mulyadi
Email address: melisa.mulyadi@atmajaya.ac.id

Pada kenyataanya tidak mudah untuk mengubah sebuah sistem di industri yang sudah terbentuk, karena harus melakukan perubahan secara keseluruhan dari sistem dan perangkat keras. Keterbatasan teknologi peralatan yang ada mengakibatkan ketika terjadi revolusi industri ke-3, sistem lama tidak dapat men-*support* sistem baru yang akan digunakan. Demikian halnya dengan industri 4.0 dibutuhkan perubahan peralatan dan sistem yang memerlukan biaya besar. Penelitian ini membahas peralihan menuju industri 4.0 dengan perubahan seminimal dan seefisien mungkin. Hal ini dilakukan dengan menggunakan mikrokontoler yang terhubung dengan jaringan internet, *Message Queuing Telemetry Transport* (MQTT) sebagai protokol komunikasi dan *Amazon Web Services* (AWS) sebagai layanan *Cloud Computing* untuk melakukan semua proses, seperti *database*, penerimaan dan pengolahan data, dan *virtual server.*

2. TEORI PENDUKUNG

2.1 Modul *Wifi* ESP32

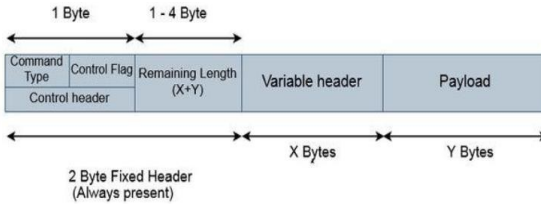
Modul *wifi* ESP32 merupakan modul mikrokontroler yang memiliki *low-cost power system* dan terintegrasi dengan modul *wi-fi* dan *bluetooth*. ESP32 dikembangkan dan dibuat oleh Espressif Systems, sebagai penerus dari modul sebelumnya yaitu ESP8266 [1]. Modul ESP32 yang digunakan adalah NodeMCU ESP-WROOM-32 seperti yang ditunjukkan pada Gambar 1. NodeMCU ESP-WROOM-32.



Gambar 1. NodeMCU ESP-WROOM- 32

2.2 Message Queuing Telemetry Transport (MQTT)

Packet Format dari protokol MQTT terdiri dari 2 *byte fixed header*, *variable header,* dan *payload.* Dua *byte* pertama dari *fixed header* bersifat *mandatory* atau selalu ada didalam *packet format*. *Variable header* dan *payload* bersifat *optional* atau dalam pengiriman *packet, byte* pada *variable header* dan *payload* dapat dibiarkan kosong, seperti Gambar 2. MQTT *packet format.*



Gambar 2. . MQTT packet format [1]

Message Queuing Telemetry Transport atau MQTT merupakan protokol komunikasi yang memiliki karakteristik sebagai *network* protokol yang mengirim dan menerima *messages* dari dua buah atau lebih alat dengan konsumsi *bandwidth* yang rendah, *lightweight*, dan menggunakan daya yang rendah, dengan standar *open Organization for The Advancement of Structured Information Standards* (OASIS) dan *The International Organization for Standardization* (ISO *Standard*) (ISO/IEC 20922) [2].

Satu *byte* pertama dari 2 *byte fixed header*, terdiri dari 8 bit *control field* yang dibagi menjadi 2 bagian. Empat bit bagian pertama adalah *command type*, untuk menentukan perintah yang akan dilakukan *packet* [3]

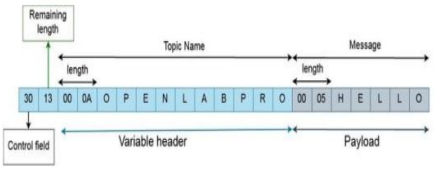
Empat bit berikutnya dari *byte* pertama *fixed header* berfungsi sebagai *publish command*. Penjelasan dari keempat bit itu adalah: jika bit-0 = 0 berarti pesan yang di *publish* disimpan sementara tetapi jika bit-0 = 1 maka pesan langsung dikirim tanpa di simpan. Bit-1 dan bit-2 menunjukan *quality of service* (QoS) dari pesan yang akan dikirim, seperti Tabel 1. QoS *List*. Bit- 3 bernilai 1 bila pesan ingin diulang bila tidak demikian diberi nilai 0.

Tabel 1. QoS List

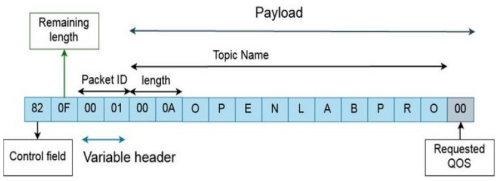
QoS	Keterangan
0	Mengirim hanya satu kali
1	Mengirim hanya satu kali dan menunggu *puback*
2	Mengirim satu kali dan menunggu *pubrec,* mengirim *pubrel* dan menunggu *pubcomp*

Variable Header tidak selalu ada dalam setiap MQTT *packet* yang dikirimkan. Beberapa perintah MQTT menggunakan *byte* atau *field* ini untuk memberi informasi tambahan atau *flags* seperti ”CONNECT” untuk melakukan koneksi, ”SUBSCRIBE” untuk masuk ke dalam *topic*, ”PUBACK” untuk meminta pesan yang terkirim untuk di kirimkan kembali ke pengirim, dan lain- lain. *Variable header* juga akan berisi *packet identifier* seperti *topic name* yang akan di *subscribe* atau informasi di *topic* mana pesan ingin di *publish*.

Payload tidak selalu ada dalam setiap MQTT *packet* yang dikirimkan, seperti bila *client* melakukan *connect/subscribe* maka *byte* pada *payload* akan kosong. *Payload* akan terisi *messages* bila *client* melakukan *publish.* Gambar 3. *Publish Packet* merupakan contoh dari *publish packet* yang mengirimkan pesan “HELLO” ke *topic* ”OPENLABPRO”. Gambar 4. *Subscribe Packet* merupakan contoh dari *subscribe packet* ke *topic* ”OPENLABPRO” dengan QoS = 0.



Gambar 3. Publish Packet **[4]**



Gambar 4. Subscribe Packet **[5]**

2.3 Amazon Web Services *(AWS)*

Amazon Web Services atau AWS merupakan sekumpulan layanan *cloud computing* yang disediakan oleh Amazon sejak tahun 2002 [6]. AWS menyediakan berbagai macam layanan yang dapat dipakai oleh pengguna untuk menciptakan *severless system* yaitu pengguna dapat mengalokasikan sistem kedalam *platform* AWS tanpa harus memiliki *server* fisik yang memakan biaya besar dan perawatan yang banyak. Tarif ditentukan bedasarkan apa yang digunakan oleh pengguna sehingga *managing resources* menjadi lebih mudah. AWSIoT merupakan *platform* yang disediakan oleh AWS untuk menghubungkan alat dari luar AWS dengan jaringan AWS secara aman menggunakan sebuah *key* dan *credentials* yang bisa di atur didalam AWSIoT [7]. Dalam penelitian ini AWSIoT akan bertindak sebagai *broker* untuk menerima pesan dari alat.

Lambda merupakan *serverless computation platform* yang disediakan oleh AWS dan berfungsi untuk menjalankan program tanpa menggunakan *server* dan secara otomatis akan menyesuaikan kapasitas sesuai kondisi yang dibutuhkan [8]. *Lambda* akan di program menggunakan bahasa pemograman *python. Lambda* berfungsi sebagai perantara antara AWSIoT dengan *database*. *Relational database service* atau RDS merupakan layanan yang disediakan oleh AWS untuk menyederhanakan pengaturan operasi, pengendalian, dan pembuatan *database* [9] Pada Tabel 2. *Database specification.* merupakan spesifikasi *database* yang digunakan

Tabel 2. *Database specification*

Kapasitas	20 Gb (SSD)
Processor	1 vCPU
Memori	1 Gb RAM

2.4 *Internet of Things* (IoT)

Internet of Things atau IoT merupakan sebuah konsep yang berfungsi untuk memperluas konektivitas internet ke benda-benda, agar benda tersebut dapat berkomunikasi satu dengan yang lain sehingga terbentuk sebuah sistem yang berbasis pada jaringan internet untuk saling berkomunikasi [1].

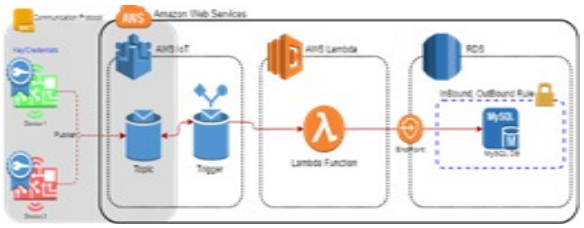
3. PERANCANGAN SISTEM

3.1 Konsep perancangan

Perancangan sistem terdiri dari dua bagian utama, yaitu perangkat keras dan *Cloud Services* menggunakan *Amazon Web Services* sebagai *platform* utama untuk melakukan kegiatan *cloud computing.* Perangkat keras yang di gunakan adalah NodeMCU-ESP- WROOM-32 yang terhubung dengan jaringan internet. Data sensor suhu bukanlah data sebenarnya melainkan data simulasi atau *random number* yang dibangkitkan dari program. Data tersebut dikirimkan oleh modul ESP 32 ke AWS yang kemudian diolah dan disimpan kedalam *database.* Perangkat keras yang dapat melakukan koneksi ke AWS, harus ditentukan melalui *platform* AWSIoT. Didalam AWSIoT dibuat *virtual gateway* khusus untuk perangkat keras diluar jaringan AWS dengan nama *things.* Selanjutnya *Things* memberikan *key/credentials* atau tanda pengenal sehingga perangkat keras dapat melakukan pengiriman dan penerimaan data.

Protokol komunikasi yang digunakan adalah protokol MQTT. Dalam hal ini AWSIoT bertindak sebagai *broker* atau seperti *host.* Didalam AWSIoT juga dibuat *topic*. Perangkat keras yang sudah memiliki tanda pengenal akan dapat melakukan *subscribe/publish message* ke dalam *topic. Message* yang diterima akan dipilah, bila *message* yang diterima memenuhi syarat yang telah ditetapkan maka *trigger function* aktif dan menyalurkan pesan ke *lambda function*. Untuk menghemat biaya pada sistem maka, *lambda function* menjalankan program bila ada *function trigger*-nya aktif.

Pengaksesan *database* dapat dilakukan melalui alamat yang diberikan oleh *database* dengan nama *endpoint.* Selain *endpoint* terdapat *security groups* yang mengatur arus *inbound* dan *outbound* ke *database*. *Database* hanya dapat diakses oleh *internet protocol* (*IP)* yang telah ditetapkan. Data yang telah disimpan dalam *database* dapat diakses menggunakan alamat/*endpoint* dari *database*, bilamana *outbound rule* diatur untuk menerima alamat IP dari luar jaringan AWS. Gambar 5 memperlihatkan diagram blok perancangan sistem.



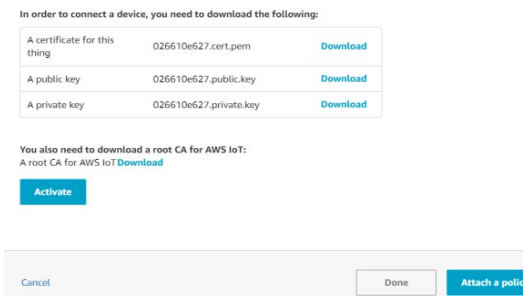
Gambar 5. Diagram blok perancangan sistem

3.2 Konfigurasi AWSIoT

Konfigurasi yang harus dilakukan pada AWSIoT adalah membuat *things* untuk deklarasi *virtual gateway* beserta *access key/credential,* pengaturan *policies* dan *security.* Pembuatan *things* dapat dilakukan dengan *login* kedalam konsol AWS, kemudian dilanjutkan dengan memilih IoT Core.Pembuatan *things* baru ditunjukkan oleh Gambar 6. *Create things.*

Things baru yang dibuat menghasilkan *Certificate, public key,* dan *private key*, yang digunakan untuk mengkases jaringan AWS. Hal ini berfungsi sebagai tanda pengenal dan sebagai pengaman terhadap akses yang dilakukan oleh pengguna/perangkat yang tidak dikenal. Gambar 7 menunjukkan AWS *key*

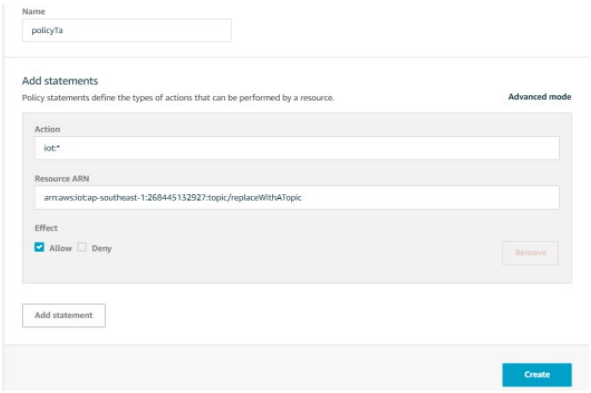
Gambar 6. *Create things*



Gambar 7 AWS *key*

3.2.1 Pembuatan Policies

Pembuatan *policies* dapat dilakukan dengan membuka bagian "*Secure"* pada IoT Core, dilanjutkan dengan memilih bagian *policies.* Didalam *policies* terdapat nama*, statement* dan *effect.* Pada kolom *statement* diberi *command* "iot:*". *Command* ini berupa *list* untuk semua aktivitas yang berhubungan pada *platform* IoT Core seperti *subscribe, publish, connect, puback,* dan lain- lain. Kolom *effect* terdiri dari dua pilihan "Allow" dan "Deny". "Deny" berfungsi untuk melarang *statement* dan "Allow" berfungsi untuk mengijinkan *statement.* Pada *policies* yang dibuat *effect* akan diatur sebagai "Allow" untuk mengijinkan segala aktivitas pada kolom *statement,* seperti Gambar 8. Pembuatan *policies.*



Gambar 7. Pembuatan policies

3.3 Konfigurasi Perangkat Keras

Perangkat keras yang digunakan adalah NodeMCU-ESP-WROOM-32. Agar NodeMCU dapat mengakses jaringan AWS, maka digunakan *library* ESP32 yaitu "AWS IoT ESP32 by Hornbill" [1]. *Key/credentials* yang telah dibuat sebelumnya berupa *hash number* maka untuk menggunakannya, perlu dilakukan perubahan program pada *library.* Caranya dengan membuka *source* folder dari *library* yang terletak pada direktori perangkat lunak Arduino IDE, kemudian pilih file "arduino_iot_certificates" yang terdapat dalam file "src". Selanjutnya *script* di ubah dengan memasukan *hash number* dari *root, pub key,* dan *certificates* yang dimiliki.

3.3.1 Kogkat Lunak Pemograman

Pemograman dilakukan menggunakan aplikasi Arduino IDE dengan bahasa pemograman C++. Program dibagi menjadi 3 bagian, yaitu:

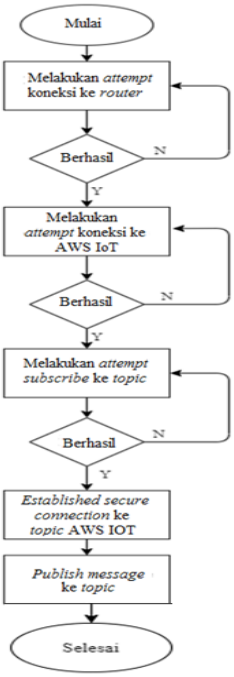
1. Deklarasi rincian Wi-Fi SSID yang digunakan, *endpoint* dari *things*, *things name, topic name*
2. *boot code* untuk melakukan koneksi pada *router* dan koneksi pada AWS IoT.
3. *package/message* yang akan di *publish* kedalam *topic.*

Aliran program (*flowchart*) pemrograman pengiriman *message* ditunjukkan dalam Gambar 9.

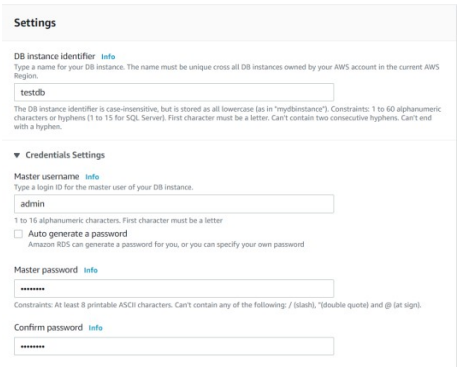
3.4 Konfigurasi Infrastruktur Database

Pembuatan database dilakukan dengan mengkases RDS pada konsol AWS. Pada dashboard dipilih create database untuk pembuatan database baru yang terdiri dari nama database, user admin, jenis database yang digunakan. Pada penelitian ini digunakan data base, MySQL. Selanjutnya mengatur spesifikasi dari database seperti banyaknya core pada processor, kapasitas random access memory, dan kapasitas storage. Pengaturan database dapat dilihat dalam Gambar 10.

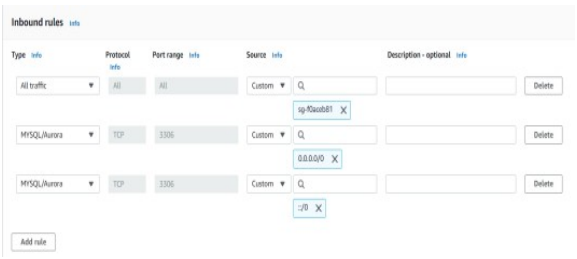
Agar database dapat diakses, maka perlu mengubah inbound rules pada kolom Virtual Private Cloud yang ada di konsol AWS. Penambahan inbound rules dilakukan pada MySQL yaitu di bagian "Type" dan mengubah "Source" menjadi 0.0.0.0/00 sehingga dapat menerima koneksi dari segala public ip address. Gambar 11 menunjukkan Inboud rule database.



Gambar 9. Diagram alir pengiriman message



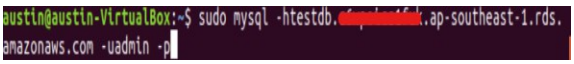
Gambar 10. Pengaturan *database*



Gambar 11. Inbound *rule database*

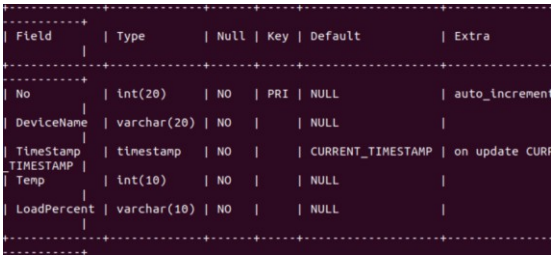
3.3.2 Pengolahan *Database*

Pengaksesan *database* dilakukan dengan menggunakan *command prompt* pada *terminal* di *Linux Operating System*, OS yang digunakan adalah *ubuntu v 18.04. Install driver MySQL* dengan masukan *command* "sudo pip install mysql" pada terminal di ubuntu. Selanjutnya masukkan id, kata sandi dan *endpoint* pada *database* yang digunakan, Gambar 12 menunjukkan *Login database*



Gambar 12. *Login database*

Pembuatan *database* baru pada terminal dilakukan dengan memasukkan *command* "CREATE DATABASE testdb". Selanjutnya masukkan *command* "use testdb;" untuk memilih *database* yang akan digunakan. Tabel yang dibuat terdiri dari No sebagai *primary key* dan *auto increment, Time* sebagai *timelapse,* Temp(int), dan LoadTemp(varchar). Dengan *command* "create table MainData (No INT (20) NOT NULL AUTO_INCREMENT PRIMARY KEY, DeviceName VARCHAR (20) NOT NULL, TimeStamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, Temp INT (10) NOT NULL, LoadPercent VARCHAR (10) NOT NULL); ". Gambar 13 menunjukkan *Describe Table* .



Field	Type	Null	Key	Default	Extra
No	int(20)	NO	PRI	NULL	auto_increment
DeviceName	varchar(20)	NO		NULL	
TimeStamp	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
Temp	int(10)	NO		NULL	
LoadPercent	varchar(10)	NO		NULL	

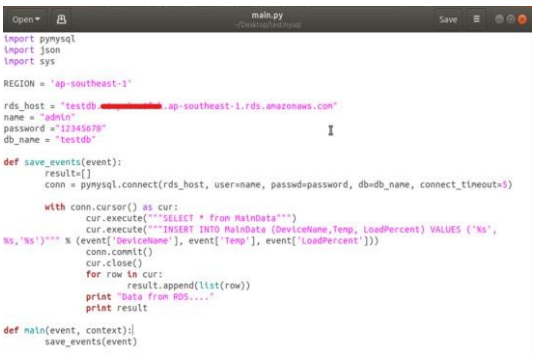
Gambar 13. *Describe table*

3.4 Pembuatan Lamda

Pembuatan *lambda function* dilakukan dengan mengakses pilihan *lambda* pada konsol AWS. *Function* dibuat dengan menekan pilihan *CreateFunction. Function* yang dibuat akan terdiri dari nama *function* dan *runtime* atau pengaturan bahasa pemograman yang digunakan. Pada perancangan digunakan *python* 2.7 sebagai bahasa pemograman.

1. Perancangan *Lambda Function Lambda function* yang dirancang

terdiri dari *driver* MySQL dan *program* untuk memasukkan data ke *database*. *Lambda* tidak menyediakan *library* khusus seperti *pymysql* sehingga harus menyiapkan *library* sendiri, dengan menulis "sudo pip install pymysql -t." pada terminal ubuntu. Program dan *driver* yang telah diunduh akan di *compressed* kedalam bentuk *zip* dan diunggah ke *lambda function* yang telah dibuat. Gambar 14 menunjukkan program *Lambda.*



```
import pymysql
import json
import sys

REGION = 'ap-southeast-1'

rds_host = "testdb.[redacted].ap-southeast-1.rds.amazonaws.com"
name = "admin"
password ="12345678"
db_name = "testdb"

def save_events(event):
        result=[]
        conn = pymysql.connect(rds_host, user=name, passwd=password, db=db_name, connect_timeout=5)

        with conn.cursor() as cur:
                cur.execute("""SELECT * from MainData""")
                cur.execute("""INSERT INTO MainData (DeviceName,Temp, LoadPercent) VALUES ('%s',
%s,'%s')""" % (event['DeviceName'], event['Temp'], event['LoadPercent']))
                conn.commit()
                cur.close()
                for row in cur:
                        result.append(list(row))
                print "Data from RDS...."
                print result

def main(event, context):
        save_events(event)
```

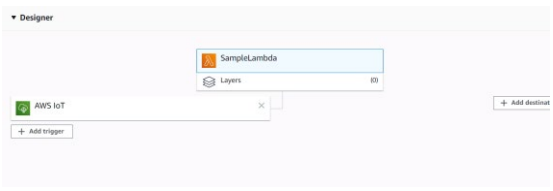
Gambar 14. Program *lambda*

2. Perancangan *Trigger*

Pemicu dari *lambda function* adalah fungsi *trigger* yang dibuat melalui IoT Core->*ACT->Rules,* lalu tekan "Create" dengan nama *rule* adalah *sampletrigger*. Selanjutnya *Rule query statement* diubah menjadi "SELECT * FROM 'test1'" test1 merupakan nama *topic* yang digunakan, *rule* tersebut akan aktif ketika ada *message* yang diterima dari *topic* test1, Gambar 15 menunjukkan *Rule query statement.*

Gambar 15. *Rule query statement*

Fungsi *trigger* yang telah terbuat, dipasang di *lambda function* untuk menyalurkan pesan dari yang diterima dari AWS IoT ke *lambda function,* seperti Gambar 16.



Gambar16. Tampilan *design lambda*

4. PENGUJIAN SISTEM

Pengujuan sistem dimulai dari perangkat keras hingga data diterima oleh AWS IoT. Pesan yang diterima memicu *trigger function* yang membuat program pada *lambda* bekerja dan pesan disimpan di *database.* Untuk menunjukan bahwa sistem yang dibuat berhasil atau layak digunakan, maka dilakukan *normal test* dan *stress test*. Pada *normal test* perangkat keras mengirim satu pesan dalam 1 detik dengan jaringan internet 4G ke AWS. Pada *stress test* perangkat keras mengirim 5 pesan dalam 1 detik dengan jaringan internet 3G /H+ ke AWS. Pengujian ini mengamati seberapa banyak pesan yang tersimpan dan besar *delay* setiap pesan sampai di *database.*

4.1 NodeMCU ke AWS IoT

Pesan yang disimulasikan di *publish* ke AWS IoT, pesan yang dikirim dan diterima dapat dilihat dengan menggunakan *serial monitor* dari aplikasi Arduino IDE dan AWS MQTT *client* pada konsol AWS. Hasil pengujian menunjukan bahwa *data* dapat dikirim dan diterima dengan baik. Gambar 17. Perangkat Keras ke AWS IoT.

Gambar17. Perangkat keras ke AWS IoT

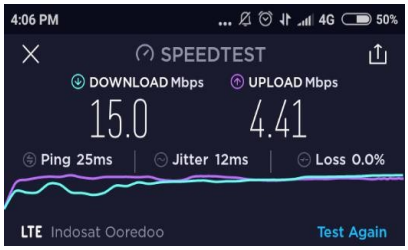
4.2 Lambda ke RDS

Pesan yang telah diterima oleh AWS IoT akan mengaktifkan *trigger function*, membuat *lambda function* bekerja dan mengirimkan data kedalam *database.* Hasil pengujian menunjukan *lambda function* berhasil mengunggah data ke *database,* hasil dapat dilihat dari *Logs* di *CloudWatch* dan *table* dari *database.* Gambar 18. *Lambda* ke RDS.

Gambar 188. Lambda ke RDS

4.3 Pengujian *Normal Test*

Tes terdiri dari dua percobaan, tes pertama akan berlangsung selama 30 detik dengan pengiriman 1 pesan per detik, menggunakan jaringan 4G pada *normal environment*. *Upload rate* akan dites menggunakan aplikasi *OoklaSpeedTest,* hasil *upload rate* sebesar 4.4Mbps, seperti Gambar 19. Hasil *speedtest* LTE



Gambar19. Hasil *speedtest* LTE

Hasil dari pengujian *Normal Test* ditunjukkan pada Tabel 3. Hasil *device* ke AWS IoT *normal test.* Didapat bahwa semua simulasi pesan yang dikirimkan oleh NodeMCU berhasil diterima oleh AWS IoT dan diproses oleh *lambda function* untuk melakukan penyimpanan data ke *database.*

Tabel 3. Hasil device ke AWS IoT normal test

Temp	Load	Status
44	52	ok
52	38	ok
78	52	ok
47	99	ok
77	82	ok
71	88	ok
74	72	ok
48	39	ok
79	95	ok
62	82	ok
45	33	ok
67	74	ok
72	69	ok
48	49	ok
40	44	ok
61	24	ok
78	73	ok
36	63	ok
34	48	ok
62	71	ok
48	50	ok
70	99	ok
38	23	ok
79	51	ok
45	83	ok
53	71	ok
67	54	ok
63	87	ok
32	38	ok

.

Tes kedua akan berlangsung selama 30 detik dengan pengiriman 2 pesan perdetik, menggunakan jaringan yang sama seperti tes pertama. Dengan persentase keberhasilan pengiriman pesan dari *device* ke AWS IoT sebesar 87% dari 53/61 pesan. Pesan yang diterima oleh AWS IoT semua berhasil disimpan ke *database* oleh *lambda function* dengan durasi waktu rata-rata penyimpanan selama 42.13 ms.

4.4 Pengujian *Stress Test*

Tes terdiri dari dua percobaan, tes pertama berlangsung selama 10 detik untuk pengiriman 5 pesan setiap detik, menggunakan jaringan 3G pada *harsh environment*. *Upload rate* akan di tes melakukan aplikasi *GoogleSpeedTest,* hasil *upload rate* sebesar 1.29Mbps, seperti Gambar 20.

Gambar 20 Hasil *speedtest* H+

Hasil dari pengujian *Stress Test* ditunjukkan pada Tabel 4. Hasil *device* ke AWS IoT *stress test.* Didapat bahwa dari 45 simulasi pesan yang dikirimkan oleh NodeMCU 8 diantaranya tidak diterima oleh AWS IoT dan tidak dapat diproses oleh *lambda function* untuk melakukan penyimpanan data ke *database.*

Tes kedua berlangsung selama 10 detik dengan pengiriman 10 pesan perdetik, menggunakan jaringan yang sama seperti tes pertama. Dengan persentase keberhasilan pengiriman pesan dari device ke AWS IoT sebesar 82% dari 92/112 pesan. Pesan yang diterima oleh AWS IoT semua berhasil disimpan ke database oleh lambda function dengan durasi waktu rata-rata penyimpanan selama 54.34 ms.

Tabel 4 Hasi pengujian *stress test*

Temp	Load	Status
46	90	ok
46	26	null
33	59	ok
51	34	ok
80	86	ok
31	31	null
68	97	ok
50	71	ok
66	46	ok
55	39	null
41	21	ok
65	70	ok
47	61	ok
41	67	null
65	56	ok
38	48	ok
39	58	ok
42	60	null
71	25	null
50	98	ok
45	38	ok
31	49	ok
78	63	ok
74	74	ok
53	56	ok
33	83	ok
77	67	ok
64	96	ok
66	99	ok
72	51	ok
49	70	ok
51	56	ok
58	54	null
71	40	ok
49	85	ok
76	58	ok
36	89	null
39	59	ok
42	52	ok
33	39	ok
42	21	ok
54	37	ok
67	61	ok
46	80	ok
45	76	ok

5. KESIMPULAN

Bedasarkan perancangan dan pengujian sistem, penelitian ini dapat ditarik simpulan sebagai berikut:

1. *Platform lambda function* pada AWS dapat menjalankan program dengan *success rate* sebesar 100% dengan waktu kurang dari 100*ms,* dalam kondisi *normal* atau *high traffic.*
2. Dengan protokol MQTT, pesan tetap dapat di kirim saat jaringan internet terbatas dan *high density message,* secara bersamaan dengan persentase *packet loss* sebesar 16.67%.

DAFTAR PUSTAKA

[1] espressif, "Penjelasan dan Datasheet ESP32," espressif, [Online]. Available: https://www.espressif.com/en/p roducts/socs/esp32. [Accessed 16 5 2020].

[2] A. Kadir, Dasar Pemograman Internet untuk Proyek Berbasis Arduino, Yogyakarta: Andi, 2017.

[3] M. Kiran, P. Murphy, I. Monga and J. Dugan, "Lambda architecture for cost- effective batch and speed big data processing," *2015 IEEE International Conference on Big Data (Big Data),* pp. 2785-2792, 2015.

[4] M. Singh, M. A. Rajan, V. L. Shivraj and P. Balamura, "Secure MQTT for Internet of Things (IoT)," *IEEE Fifth International Conference on Communication Systems and Network Technologies,* 2015.

[5] M. V. e. al, "Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures," *IEEE/ACM International Symposium on Cluster,* pp. 179-182, 2016.

[6] "Protokol komunikasi MQTT," [Online]. Available: https://public.dhe.ibm.com/soft ware/dw/webservices/ws- mqtt/mqtt-v3r1.html. [Accessed 16 5 2020].

[7] aws, "Penjelasan Amazon Web Services AWS," aws, [Online]. Available: https://aws.amazon.com/id/. [Accessed 16 5 2020].

[8] "Cara Kerja Konsep Internet of Things," Mypsolution, [Online]. Available: http://www.myspsolution.com/n ews-events/cara-kerja-konsep- internet-of-things/. [Accessed 16 5 2020].

[9] "Arduino Library AWS IoT ESP32," GITHUB, [Online]. Available: https://github.com/aws-samples/aws-iot-esp32-arduino-examples. [Accessed 16 5 2020].