

Let Us Create a Physical IoT Device Using AWS and ESP Module

Sudip Chakraborty¹ & P. S. Aithal²

¹ D.Sc. Research Fellow, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,

OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

² Vice Chancellor, Srinivas University, Mangalore, India,

OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.

Type of the Paper: Experimental-based Research.

Type of Review: Peer Reviewed as per [C|O|P|E](#) guidance.

Indexed in: OpenAIRE.

DOI: <https://doi.org/10.5281/zenodo.7779097>

Google Scholar Citation: [IJMTS](#)

How to Cite this Paper:

Chakraborty, S., & Aithal, P. S., (2023). Let Us Create a Physical IoT Device Using AWS and ESP Module. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, 8(1), 224-233. DOI: <https://doi.org/10.5281/zenodo.7779097>

International Journal of Management, Technology, and Social Sciences (IJMTS)

A Refereed International Journal of Srinivas University, India.

CrossRef DOI: <https://doi.org/10.47992/IJMTS.2581.6012.0265>

Received on: 15/03/2023

Published on: 29/03/2023

© With Authors.



This work is licensed under a [Creative Commons Attribution-Non-Commercial 4.0 International License](#) subject to proper citation to the publication source of the work.

Disclaimer: The scholarly papers as reviewed and published by Srinivas Publications (S.P.), India are the views and opinions of their respective authors and are not the views or opinions of the SP. The SP disclaims of any harm or loss caused due to the published content to any party.

Let Us Create a Physical IoT Device Using AWS and ESP Module

Sudip Chakraborty¹ & P. S. Aithal²

¹ D.Sc. Researcher, Institute of Computer Science and Information sciences, Srinivas University, Mangalore-575 001, India,

OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

² Vice Chancellor, Srinivas University, Mangalore, India,

OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: This research paper explores the feasibility of creating a physical AWS IoT device using an ESP module. The paper describes the steps and process of building such a device, including the required hardware and software components. The ESP module is chosen for its low cost, small size, and ability to connect to the internet through Wi-Fi. The AWS IoT platform manages and monitors the device, including the ability to receive and send data to and from the device. The paper includes a detailed explanation of the programming and setup involved in creating the device and the challenges and limitations encountered during the process. Ultimately, this paper demonstrates that creating a physical AWS IoT device using an ESP module is possible, providing a cost-effective solution for developers looking to build IoT devices. The project code is available to download.

Design/Methodology/Approach: Initially, we create things inside the AWS cloud. Download all certificates and credentials. Then the downloaded credential, we added to the project variable—the Code compiles and runs. Our ESP8266 hardware is then ready to receive the topic. We use two channels to send the topic to our ESP Module. We can send topics using the AWS cloud MQTT test client interface. Another way is from the C# dot net MQTT client application. We develop an application in the visual studio that can update the AWS device shadows. We will notice that data sending to the AWS Device shadows and updating the ESP module almost in real-time.

Findings/Result: We use the ESP module to experiment with the AWS IoT interface. Sometimes researchers need to transfer the data over IoT. So here we provide the complete practical guide for IoT experiments. Here we demonstrate How to create the IoT devices and send/update the IoT Devices' Shadow. We integrated the C# MQTT client. Using our created application, we will update AWS cloud MQTT devices shadows. It might be more help full for new researchers to integrate IoT into their projects.

Originality/Value: After studying Several documents, we created this paper after doing lots of experiments so that our researcher could experiment easily. The available Code is wholly tested and workable. Our researchers can integrate it into their projects with a little customizing effort. The researcher can find some relevant documents for their research work.

Paper Type: Experimental-based Research.

Keywords: AWS IoT, IoT using ESP8266 and ESP32, IoT Practical Example

1. INTRODUCTION :

In recent years, the Internet of Things (IoT) has become increasingly prevalent, connecting devices and allowing them to communicate. One of the most popular platforms for IoT development is Amazon Web Services (AWS), which provides a suite of tools for managing and analyzing IoT devices. While AWS offers a range of virtual services, building a physical IoT device can offer unique advantages in terms of functionality and customization. This research paper will explore how to create a physical AWS IoT device using the ESP module. The ESP module is a low-cost, Wi-Fi-enabled microcontroller that can be easily programmed and integrated with AWS services. Following our step-by-step guide, readers will learn how to build an ESP-based IoT device and connect it to AWS, opening up a world of

possibilities for their IoT projects. AWS IoT is a cloud-based platform that enables secure communication between internet-connected devices and the AWS Cloud. It provides various services such as device management, message brokering, data storage, and analytics to help developers build and manage IoT applications. With AWS IoT, developers can easily connect, manage, and secure their IoT devices and applications at scale. It supports protocols like MQTT, HTTP, and WebSocket and can be integrated with other AWS services like Lambda, S3, and DynamoDB for advanced analytics and machine learning.

The paper is organized as follows: Section 2 provides an overview of related work already done on IoT. Section 3 discusses the objective of the research work. Section 4 highlights the methodology we used for the research work. In section 5, we do the actual experiment. This section describes the procedure for creating an IoT in the AWS cloud. Section 6 provides recommendations for further reading or watching videos to understand the AWS IoT better. Finally, Section 7 concludes the paper and provides future research directions.

2. RELATED WORKS :

Tawalbeh, M. et al., in their work, discuss different security models provided for Cloud-Enabled IoT architectures security issues and propose a security model using Amazon Web Service (AWS) cloud provider platform with Attribute-Based Access Control (ABAC) [1]. Jozef Mocnej et al., in their paper, describe the utilization of edge computing in the IoT and analyze its impact on the energy consumption of IoT devices [2]. Botez et al. discuss the design of a containerized IoT and M2M application and the mechanisms for delivering automated scalability and high availability [3]. Joseph, S. et al. proposed work on implementing an IoT-based heartbeat monitoring system. The proposed method is tested with a photoplethysmography sensor interfaced with a microcontroller. Microcontroller monitors the heartbeat sampling rate and then transmits discrete values over the ESP8266 Wi-Fi module [4]. Kalubi, N. and S. Sajal research on Cloud computing. They demonstrate the relevancy of cloud servers for IoT applications [5]. Medvedev, I. et al., in their paper, demonstrate the requirements for monitoring patients' health using the Internet of Things (IoT) and cloud technologies [6]. G. N. Satya Sai et al. proposed a system measuring five water parameters: Potential of Hydrogen (pH), Total Dissolved Solids (TDS), Temperature, Turbidity, and Flow Rate. All this data is transmitted to the Amazon Web Services (AWS) cloud platform hosted in Firebase and made available to the admin dashboard and User mobile application (App). If any deviations occur in water quality, the user will get a notification, and he/she can make a precise decision [7-12]. Sudip Chakraborty et. Al., in their paper, demonstrates how to create an IoT inside the AWS cloud environment [13].

3. OBJECTIVES :

In the IoT field, AWS IoT is a secure and reliable domain. Recently the AWS interface has changed. It is tricky to integrate, and there is some learning curve. After long research, we created and prepared documents so our researcher could integrate AWS IoT. The research paper aims to provide some reference information to the researchers trying to experiment with AWS IoT physical devices. We create and provide an AWS IoT physical testing procedure using the C# MQTT client program. Nowadays, The smart Home is becoming a necessary thing. Especially older people get to benefit from the intelligent Home. The IoT is the backbone to operate or control any electrical appliances. Here we provide a route map to build the IoT infrastructure.

4. APPROACH AND METHODOLOGY :

Figure 1 depicts the requirement of the parameter of the project. We are now discussing this in detail.

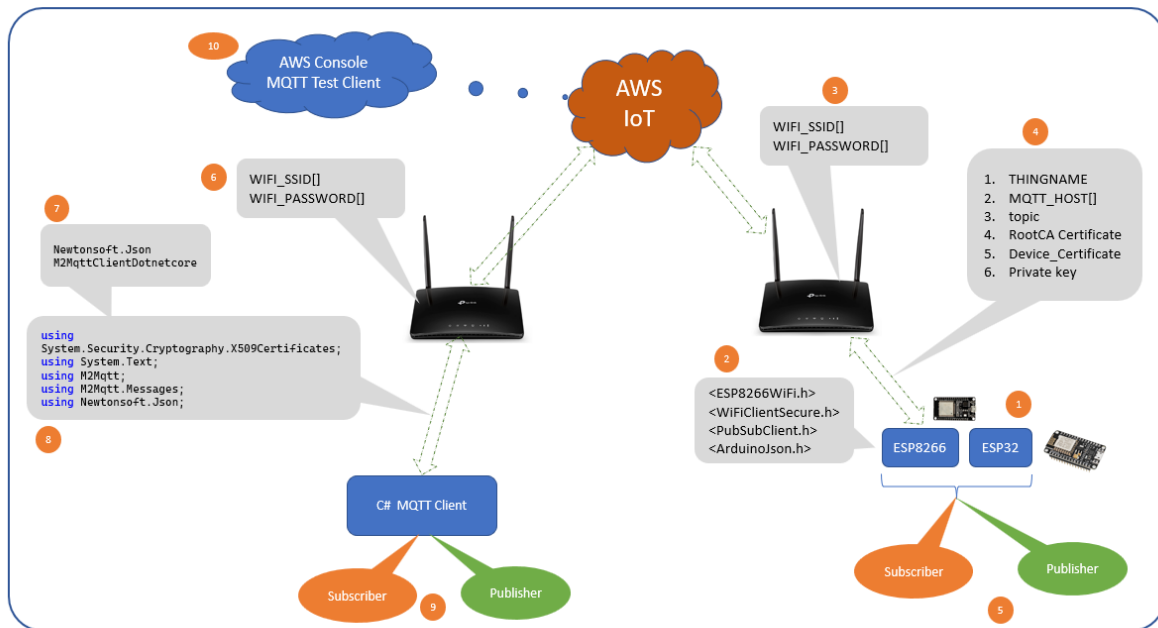


Fig. 1: Parameter requirement for the project

- (1) To create an AWS physical IoT device, we chose ESP8266-based “**NodeMCU 1.0(ESP-12E Module)**” and “**DOIT ESP32 DEVKIT V1**” under the subset of the ESP Eco-system. We selected this module due to its availability and cheap. It is easy to test with a breadboard or a vero board. Moreover, its form factor is also tiny.
- (2) To communicate with AWS IoT, we need some modules. If the Arduino IDE is just installed, we must install these modules before compiling the project from the Library manager.
- (3) Our ESP module communicates with the AWS server over Wi-Fi. The Wi-Fi module needs a Wi-Fi router to connect to the internet. The router may be the Fibre optic endpoint, or we can use our mobile as a hotspot. When the module tries communicating with Wi-Fi, it needs login credentials, ID, and password. The ID and password need to add to the secret.h module.
- (4) After successfully connecting with Wi-Fi, the module will try to connect with the AWS cloud server. The AWS will check the RootCA, Device, private key, MQTT host, topic, and things name. If these are OK, then it will allow the client to connect.
- (5) The ESP module registers the publisher and Subscriber callback function. When the topic is available, AWS sends it to this module. We must go through the registered callback function to update the data to the cloud.
- (6) Once our Embedded side is ready, we need to test. When testing the C# MQTT client, we must connect the working system with wired or wireless internet. The Wi-Fi credential does not need to be added to our Code, but the operating system needs to connect with the network.
- (7) In our C# code, we need two modules from the Nuget package manager else it throws a compilation error.
- (8) We need these for running the C# MQTT client module. We must add these lines to create a custom module using the Nuget package manager.
- (9) This part is the Subscriber and publisher. When we run the application, it will send the Data to the AWS cloud in a specific interval.

5. EXPERIMENT :

Now we will do some experiments to create AWS IoT devices. Before proceeding with the experiment, we recommend studying the below paper, where we discussed how to create an IoT device inside the AWS cloud using screenshots. We also discussed the IoT creation process here but in a concise manner.

<https://www.srinivaspublication.com/journal/index.php/ijcsbe/article/view/2283/875>

Create a thing: we need to follow the below steps:

- 1) Open the AWS IoT console
- 2) Select proper Region
- 3) Navigate All devices> Things> click on the **Create things** button > Create single thing> click **Next** button.
- 4) Thing name: **MyThing** > click on **Next** > keep selecting “Auto-generated a new certificate(recommended)” > Next > click **Create thing** button.
- 5) Download the Device certificate, public key, private key, and Root CA1 certificate
- 6) Click **Done**.
- 7) Now, things are created.

Create Device Shadow:

- 1) Navigate All devices > Things > click on Things > “Device Shadows”
- 2) Click on **Create Shadow** > Type name: MyShadow.
- 3) Click on **Create** button.
- 4) The Shadow is created now.

Create Policies:

- 1) Navigate Policies> click on **Create policy**.
- 2) Name: **MyPolicy**
- 3) Under Policy action, select “*.”
- 4) Under Policy resource, type “*.”
- 5) Click on Create.
- 6) Now the policy is created.

Attach Policies:

- 1) Navigate Things > MyThing > click on Tab **Certificates** > click on Certificate ID > click on **Attach policies** > from drop-down select **MyPolicy**.
- 2) Click on **Attach policies**.
- 3) Now our policy is attached to things.

Arduino IDE Setup:

- 1) Download Arduino IDE *Arduino-ide_2.0.4_Windows_64bit*. And install.
- 2) Download the project code from GitHub. The link is provided under recommend section.
- 3) Open the IDE. Click on the file menu > Open.
- 4) Goto ESP8266 folder>ESP8266_IoT>open ESP8266.INO file. The Code will open in a new window.
- 5) Navigate File > Preferences > inside the “Additional boards manager URLs” textbox, and add the line for both ESP8266 and ESP32 modules:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json,

http://arduino.esp8266.com/stable/package_esp8266com_index.json

- 6) Under tools menu >Board > Boards Manager.. > in the search box, type “ESP8266” > click on install. It may take a couple of minutes. After that, a Security Alert window may appear. Click Allow access.
- 7) Under Tools menu >Board > esp8266 > select proper ESP module. Here we selected “NodeMCU 1.0(ESP-12E Module)
- 8) Under Sketch menu >Include Library > Manage Libraries> in the search box, type “**PubSubClient**” (by Nick O’Leary 2.8.0). Click install.
- 9) In the search box, again type “**ArduinoJson**” (Benoit Blanchon 6.20.1). Click install.
- 10) Click IDE “Verify” button, which is the build button. The compilation process takes a couple of minutes. After that, the compilation summary is available at the bottom part of the IDE.
- 11) Our IDE and Code are error-free now. We need to customize it according to our own AWS IoT credentials.

```

ESP8266_IoT.ino  time.cpp  Secrets.h
1  #include <pgmspace.h>
2
3  #define SECRET
4
5  int8_t TIME_ZONE = -5; //NYC(USA): -5 UTC
6
7  const char WIFI_SSID[] = "*****";
8  const char WIFI_PASSWORD[] = "*****";
9
10 #define THINGNAME "*****"
11
12 const char MQTT_HOST[] = "*****";
13
14 static const char cacert[] PROGMEM = R"EOF(
15 -----BEGIN CERTIFICATE-----
16 -----END CERTIFICATE-----
17 )EOF";
18
19 // Copy contents from XXXXXXXX-certificate.pem.crt here ▼
20 static const char client_cert[] PROGMEM = R"KEY(
21 -----BEGIN CERTIFICATE-----
22 -----END CERTIFICATE-----
23 )KEY";
24
25
26 // Copy contents from XXXXXXXX-private.pem.key here
27 static const char privkey[] PROGMEM = R"KEY(
28 -----BEGIN RSA PRIVATE KEY-----
29
30 -----END RSA PRIVATE KEY-----
31
32 )KEY";
33

```

Fig. 2: Parameter requirement for the project

ESP8266 Code Customization:

- 1) Go to the secret.h file
- 2) Fill WI-FI_SSID[] and WIFI_PASSWORD[].
- 3) Copy the thing name from the AWS IoT console and Modify THINGNAME “MyThing.”
- 4) Under settings in the AWS IoT console, copy Endpoint and paste inside the MQTT_HOST[]= “.”
- 5) From the downloaded certificate, open AmazonRootCA1. Copy content between BEGIN CERTIFICATE and END CERTIFICATE and paste inside cacert[].
- 6) Open device certificate x...x-certificate.Pem and copy and paste inside the client_cert[].
- 7) Open private key xx-private. Pem and copy-paste inside the privkey[].
- 8) In the ESP8266.INO file, at the top portion, add “publish” and “subscribe” topic name, which is available under the Device shadow section.
- 9) Now our ESP8266 Code is updated. Connect the ESP module with the system.
- 10) Build and upload to the ESP module.
- 11) Open IDE terminal. If everything is OK, it will show the Wi-Fi connection, then connect with AWS OK.
- 12) Now it will wait to receive the topic. If the topic is available, it will display inside the terminal. Figure 3 display the received data.


```

12:07:13.624 -> .Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1575"}}}
12:07:14.596 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1576"}}}
12:07:15.600 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1577"}}}
12:07:16.618 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1578"}}}
12:07:17.602 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1579"}}}
12:07:18.684 -> Message Published.
12:07:18.684 -> .Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1580"}}}
12:07:19.611 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1581"}}}
12:07:20.612 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1582"}}}
12:07:21.597 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1583"}}}
12:07:22.599 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1584"}}}
12:07:23.628 -> Message Published.
12:07:23.628 -> .Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1585"}}}
12:07:24.619 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1586"}}}
12:07:25.647 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1587"}}}
12:07:26.661 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1588"}}}
12:07:27.673 -> Received [$aws/things/MyThing/shadow/name/MyShadow/get/accepted]: {"state":{"desired":{"command":"1589"}}}
12:07:28.666 -> Message Published.
    
```

Fig. 3: Data received by ESP8266 Module Displaying in the Terminal

ESP32 Modification :

The same procedure needs to follow for ESP32 Module.

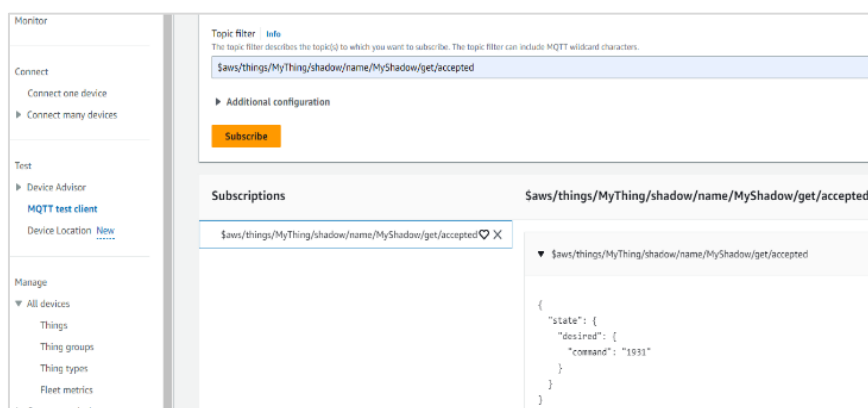


Fig. 4: Topic Display inside the AWS console

Test procedure-1: using AWS

- 1) Open the AWS IoT console
- 2) From the left, select MQTT test client
- 3) Click on the **Subscribe to a topic** tab
- 4) In the topic filter text box, enter the topic name, **\$aws/things/MyThing/shadow/name/MyShadow/get/accepted**
- 5) Click on **Subscribe** button. The updated topic is displayed here if the C# application is running. Figure 4 depicts the topic displayed in the AWS console.

Test procedure-2: using C# MQTT Client

- 1) Create a new console application inside visual studio 2022 or later.
- 2) Add the AWS_MQTT_Client.cs module, available inside the project folder.
- 3) Using the NuGet package manager install two packages. **Newtonsoft.Json** by James Newton-King and **M2MqttClientDotnetcore** by M2MqttClientDotnetCore1.0.1
- 4) Open the AWS IoT console. From the Top right corner, Select the proper Region. From the left, click on settings; under settings, copy Endpoint and paster program.cs **iotEndpoint** string variable.
- 5) Navigate All devices>Things>MyThing>Device Shadows>MyShadows>MQTT topics>copy /get/accepted topic and paste string variable into the **topic**.
- 6) From the downloaded certificate, copy “AmazonRootCA1.pem” and paste it inside the \bin\Debug\net6.0 folder.

- Open <https://www.sslshopper.com/ssl-converter.html> to convert our certificate. Figure 5.4 depicts the web interface for certificate conversion.

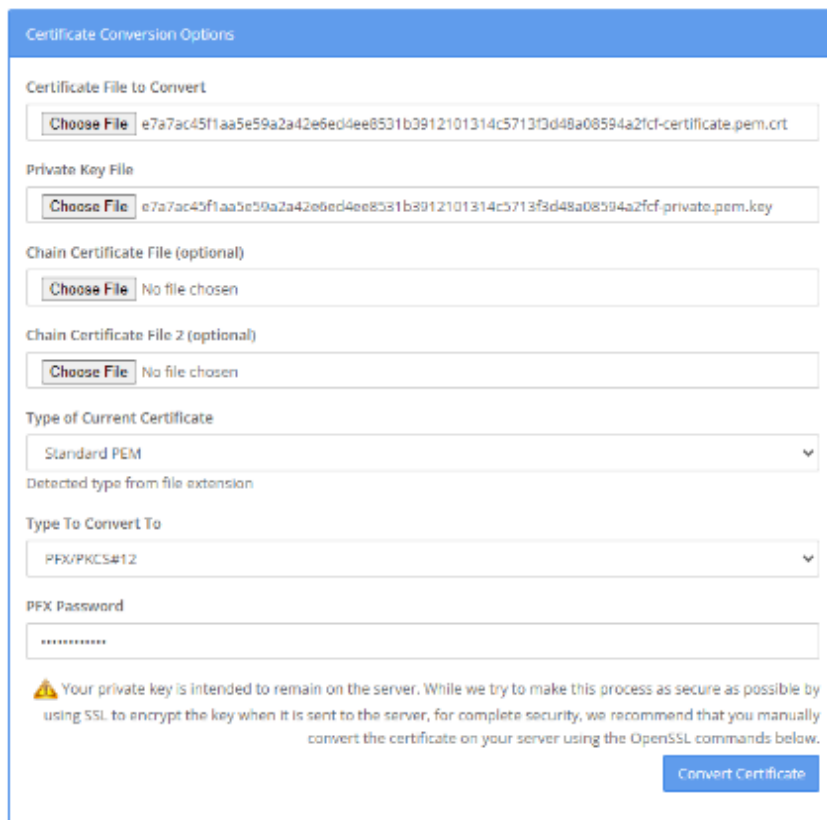


Fig. 5: Certificate Conversion interface

- Select Type To Convert To “PFX/PKCS#12”. Choose File xxx.private.pem.key File. Type of Current Certificate “Standard PEM.”
- In the PFX Password field, enter a password. Now click on the “Convert Certificate” button. Change the certificate, rename device_certificate.cert, and paste it \bin\Debug\net6.0 folder.
- The entered password needs to paste inside the program.cs **password** variable.
- Build and run the application. The console will show the data sent to the cloud every second. Figure 6 depicts the transmitted data.

```
Device connected with AWS ok...
{"state":{"desired":{"command":"0"}}}
{"state":{"desired":{"command":"1"}}}
{"state":{"desired":{"command":"2"}}}
{"state":{"desired":{"command":"3"}}}
{"state":{"desired":{"command":"4"}}}
{"state":{"desired":{"command":"5"}}}
{"state":{"desired":{"command":"6"}}}
{"state":{"desired":{"command":"7"}}}
{"state":{"desired":{"command":"8"}}}
{"state":{"desired":{"command":"9"}}}
{"state":{"desired":{"command":"10"}}}
```

Fig. 6: Transmitted data to AWS cloud

6. RECOMMENDATIONS :

- GIT repository project link: <https://github.com/sudipchakraborty/Let-Us-Create-A-Physical-AWS-IoT-Device-Using-ESP-Module.git>
- We adopted documents on AWS IoT using ESP8266: <https://www.youtube.com/watch?v=x9GfxgkEpXg&t=16s> and the Code is available from <https://how2electronics.com/connecting-esp8266-to-amazon-aws-iot-core-using-mqtt/>
- We adopted documents on AWS IoT using the ESP32 module <https://www.youtube.com/watch?v=hgQ-Ewrm48c&t=205s>, and the Code is available from <https://how2electronics.com/control-relay-led-lamp-with-aws-iot-core-using-esp32/>

7. CONCLUSION :

In conclusion, this research paper has presented a comprehensive guide to creating a physical AWS IoT device using the ESP module. We have discussed the benefits of using AWS IoT for building connected devices and outlined the steps involved in setting up an AWS IoT environment. We have also provided a detailed explanation of the hardware and software components required to build an ESP-based IoT device and how to connect it to AWS IoT. This project has demonstrated the feasibility of building IoT devices using readily available hardware and cloud-based services. The ESP module is an affordable and versatile platform that can build many IoT devices. By leveraging the power of AWS IoT, developers can easily create, manage, and scale their IoT applications. Overall, this research paper provides a valuable resource for developers and hobbyists interested in building IoT devices using the ESP module and AWS IoT. We hope this guide will inspire further experimentation and innovation in the field of IoT and encourage more people to explore the potential of connected devices.

REFERENCES :

- [1] Tawalbeh, M., Quwaider, M. and Tawalbeh, L. A. (2020). Authorization Model for IoT Healthcare Systems: Case Study, 11th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2020, pp. 337-342, DOI: <https://doi.org/10.1109/ICICS49469.2020.239527>.
- [2] Jozef Mocnej, Martin Miškuf, Peter Papcun, Iveta Zolotová, (2018). Impact of Edge Computing Paradigm on Energy Consumption in IoT. *IFAC-PapersOnLine*, 51(6), 162-167, <https://doi.org/10.1016/j.ifacol.2018.07.147>.
- [3] Botez, Robert, Jose Costa-Requena, Iustin-Alexandru Ivanciu, Vlad Strautiu, and Virgil Dobrota. (2021). SDN-Based Network Slicing Mechanism for a Scalable 4G/5G Core Network: A Kubernetes Approach. *Sensors*, 21(11), 3773, 1-26. <https://doi.org/10.3390/s21113773>.
- [4] Joseph, S., Shahila, D. F. D. and Patnaik, S. (2019). IOT based Remote Heartbeat Monitoring. International Conference on Advances in Computing, Communication and Control (ICAC3), Mumbai, India, 2019, pp. 1-5, DOI: <https://doi.org/10.1109/ICAC347590.2019.9036735>.
- [5] Kalubi, N. and Sajal, S. (2022). Cloud Computing: Arduino Cloud IoT Integration with REST API. 2022 IEEE International Conference on Electro Information Technology (eIT), Mankato, MN, USA, 2022, pp. 473-476, DOI: <https://doi.org/10.1109/eIT53891.2022.9814027>.
- [6] Medvediev, I., Iliashenko, O., Uzun, D. and Strielkina, A. (2018). IoT solutions for health monitoring: Analysis and case study. 2018 IEEE 9th International Conference on Dependable Systems, Services, and Technologies (DESSERT), Kyiv, Ukraine, 2018, pp. 163-168, DOI: <https://doi.org/10.1109/DESSERT.2018.8409120>.
- [7] Satya Sai, G. N., Sudheer, R., Manikanta, K. S., Arjula, S. G. Rao, B. N. and Sai D. V. Maneeswar Mutyala, (2021). IoT based Water Quality Monitoring System, 2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC), Bangalore, India, 2021, pp. 01-06, DOI: <https://doi.org/10.1109/R10-HTC53172.2021.9641630>.
- [8] Kurniawan, A. (2019). *Internet of Things Projects with ESP32: Build exciting and powerful IoT projects using the all-new Espressif ESP32*. Packt Publishing Ltd.
- [9] Swapnil Sen Pandey, A. K., & Das, K. K. (2018). IoT-Based Intelligent smart Metering Infrastructure in Real-Time Environment. A Project Report. [Google Scholar](#)

- [10] Gupta, V., Khera, S., & Turk, N. (2021). MQTT protocol employing IOT-based home safety system with ABE encryption. *Multimedia Tools and Applications*, 80(2), 2931-2949. [Google Scholar](#)
- [11] Krishnan, B., Karakkat, A., Menon, R. M., & Vasudevan, S. K. (2022). An affordable, intelligent, and fully functional innovative ventilator system. *International Journal of Medical Engineering and Informatics*, 14(6), 550-563. [Google Scholar](#)
- [12] Pearson, B., Luo, L., Zhang, Y., Dey, R., Ling, Z., Bassiouni, M., & Fu, X. (2019, May). On misconception of hardware and cost in IoT security and privacy. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (pp. 1-7). IEEE. [Google Scholar](#)
- [13] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create An IoT Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, 7(1), 211- 219. DOI: <https://doi.org/10.5281/zenodo.7726980>.
