AWS Research

By: Johnson Domacasse (#4471709)

14 April 2024

Contents

1. Introduction: 3
2. Methods: 3
2.1 Setting up our environment: 3
2.1.1 Creating a policy 3
2.1.2 Creating a thing: 3
2.2 MQTT Node-red: 3
2.3 Connecting ESP32 to the cloud: 4
2.4 Environmental Sensor: 5
3. Results: 5
4. Discussion: 7
4.1 MQTT node malfunction: 7
5. Bibliography: 7

1. Introduction:

purpose of this research is to get us acquainted with the cloud services that are provided by Amazone Web Services (AWS). In this research we will attempt to connect a node-red application with the cloud and from there see if we can take this further by connecting an environment sensor to it and control it.

2. Methods:

This research was done by following a step-by-step guide on connecting to the cloud. This report will also give an updated version as to how you are supposed to set up everything as well seeming as this guide was intended for an older version of AWS.

2.1 Setting up our environment:

The following tools were used for this research:

- Node-red
- ESP32 WROOM Devkit
- DHT22 sensor + library
- AWS_IOT library

We begin by creating a “thing”. A thing can be a representation of your devices that can be found in the cloud. A physical device needs a thing in order to work with AWS IoT. To create a thing we must first access the IoT core which can be found services section under IoT. Here we choose IoT core.

2.1.1 Creating a policy

Before creating our thing, it is advisable to create a policy. In the older versions you could create one during the creation of a thing. This can be done in the policies tab under the security section in the side panel found on the left. Here when we are creating a policy, we assign an ‘*’ sign to both the action and the resource as input. This allows us to use everything. The policy is now created and can be used.

2.1.2 Creating a thing:

From there on the left side panel, we traverse into the devices tab under the manage section. Here we can find our thing. We begin by creating assigning it a name and a shadow name (in this particular research it was necessary for a topic that comes later). Then you are prompted to choose a certificate, here we simply auto-generate it and then we choose the policy that we created earlier. From there we download all of our certificates and keep them somewhere that can later be found. These include the device certificate, the key files and the root CA certificates. The thing is now created and can be used.

2.2 MQTT Node-red:

Now that our thing is created we can start by setting up a simple mqtt connection between node-red and the cloud. In the guide, it states that we need to use a URL that can be found in the ‘Interact’ tab. In the latest version, this tab isn’t found. Instead we use the shadow, that we have created when we created our thing. By going into the shadow, we can see the device shadow URL. This is the same URL we will use for our MQTT connection.

Note: The entire URL begins with “https” and ends with the shadow name. We only need what comes after the two “//” signs and ends with “.com”.

From here in our node red we use an mqtt-out node. We configure this accordingly to what the IoT cloud needs. Under URL we use the same one as found in the shadow. Keep the node in mind. We use port 8883. From here we use TLS configuration to put all of our needed files and security measures in check. Under certificates we upload the device certificate and under private key, we put the private key file (Not the public one. See the discussion section for this). We put our AWS password as our passphrase. Finally we upload the CA1 certificate under CA certificate. From there we need the topic. In the AWS dashboard we go to MQTT test client under the Test section. Here we assign the topic name as well as setting our payload display to strings. If this is done in JSON format, it will give some readability issues. See figure 6. This topic we chose is what we pass in the MQTT node as well. Now the node should be connected after deployment. Here we use An inject node to send a hello string. See figure 1. From there we can see the results again in our cloud dashboard on AWS. See figure 5. If youre MQTT block does not connect, see section 4.1.

Figure 1: *Sending hello to the configured MQTT topic.*

2.3 Connecting ESP32 to the cloud:

Now we begin implementing a program to connect it to the cloud. For this we use the Arduino example provided to us. From there, we need to change a small line so that it connects to our wireless network, to then send to the cloud. Rather than checking the “status” variable in the setup function to connect, we need to check the “status” function using the Wi-Fi library. From there we need to give our ESP access to the cloud. We do that by going into our library manager, and download the AWS_IOT library. From there we traverse to the path of this file. Within the Src folder, there is a c-file called “aws_iot_certificates”. Within the file youll see three arrays that need to be filled with the certification information that we have downloaded and kept in the past. We open these certificates in text format. We go line by line, copying the text, and replacing the array contents with the copied text from the certificates. Remember, use the private key, device certificate, and the CA 1 certificate. Be patient with this task as one wrong character will result in the program not connecting to the cloud. See figure 3**.**

```
09:32:50.319 -> Connected to wifi
09:32:52.338 -> Connected to AWS
09:33:07.682 -> Attempting to connect to SSID: iPhone (2)
09:33:12.796 -> Connected to wifi
09:33:12.796 -> E (7148) aws_iot: failed!  mbedtls_x509_crt_parse returned -0x2180 while parsing root cert
09:33:12.796 -> E (7148) AWS_IOT: Error(-19) connecting to a31q7h1d2sxx4v-ats.iot.eu-central-1.amazonaws.com:8883,
09:33:12.796 -> Trying to reconnect
```

Figure 2: ESP not connecting because of invalid certificate(s).

If you have pasted the certificates correctly, then you should see it in the serial monitor that it connected fine and is sending the messages to the cloud (without the error of connecting). See figure 3**.**

```
19:15:33.362 -> Connected to wifi
19:15:51.693 -> E (50527) aws_iot: failed! mbedtls_net_connect returned -0x44
19:15:51.693 -> E (50528) AWS_IOT: Error(-24) connecting to a31q7h1d2sxx4v-ats.iot.eu-central-1.amazonaws.com:8883,
19:15:51.693 -> Trying to reconnect
19:15:58.453 -> Connected to AWS
19:15:59.808 -> Subscribe Successful
19:16:06.806 ->
19:16:06.806 ->
19:16:11.812 -> Publishing Message:
19:16:11.812 -> Hello from hornbill ESP32 :1
19:16:11.812 -> Publishing Message:
19:16:11.812 -> Hello from hornbill ESP32 :1
19:16:12.809 -> Received Message:Hello from hornbill ESP32 :1
19:16:16.820 -> Publishing Message:
19:16:16.820 -> Hello from hornbill ESP32 :2
```

Figure 3: *ESP connecting to the cloud.*

2.4 Environmental Sensor:

Finally we begin implementing the environmental sensor to the project after confirming the MQTT cloud works with our ESP. We have done this in past researches so I will keep it simple. We take some old code that reads and puts the read values into a JSON object. Rather then sending a string, we are now sending a JSON string. So we replace the variables with the JSON buffer. From there in our MQTT dashboard in AWS, we configure the same setup as previously with one change. Instead of reading the received message in string format, we use JSON. After uploading and connecting, we can see that are messages are being sent I the serial monitor. See figure 4**.** Finally, we can see the same results in JSON format in our cloud MQTT dashboard as well. See figure 8.

```
19:42:40.051 -> Connected to wifi
19:42:41.628 -> Connected to AWS
19:42:42.737 -> Subscribe Successful
19:43:01.767 -> Publishing Message:
19:43:01.767 -> {"temperature":22.29999924,"humidity":57.70000076,"location":"Tilburg"}
19:43:01.767 -> Publishing Message:
19:43:01.767 -> {"temperature":22.29999924,"humidity":57.70000076,"location":"Tilburg"}
```

Figure 4: *Sensor data being sent to the cloud MQTT.*

3. Results:

The purpose of this section is to show the results that were found during this research.

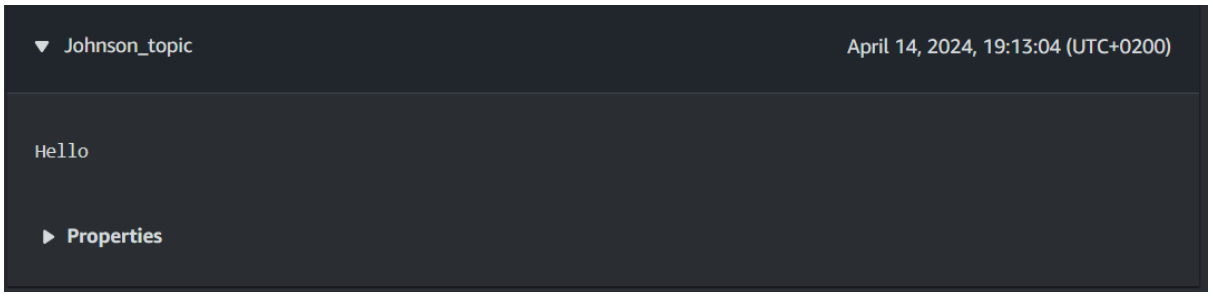

Figure 5: *Hello string being received*

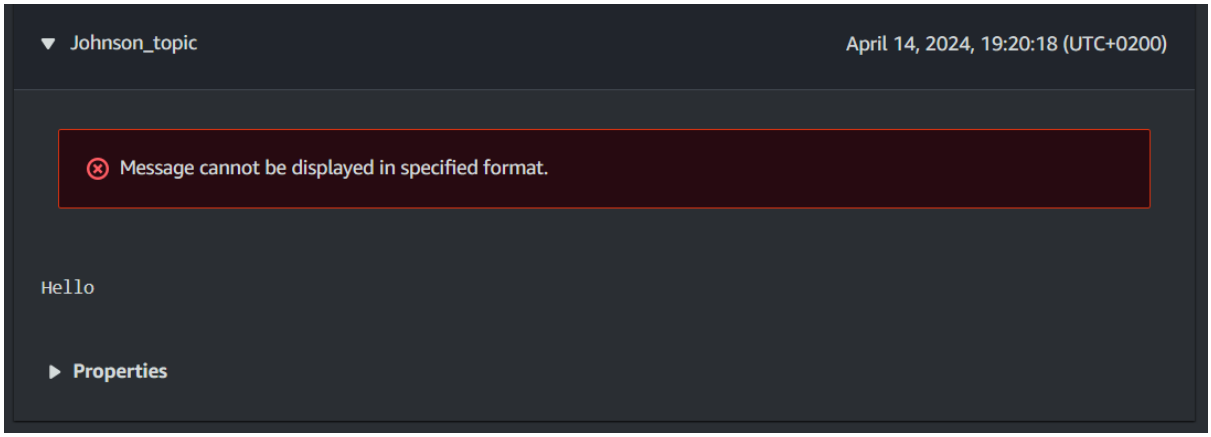

Figure 6: Hello string being received with issues due to JSON formatting.

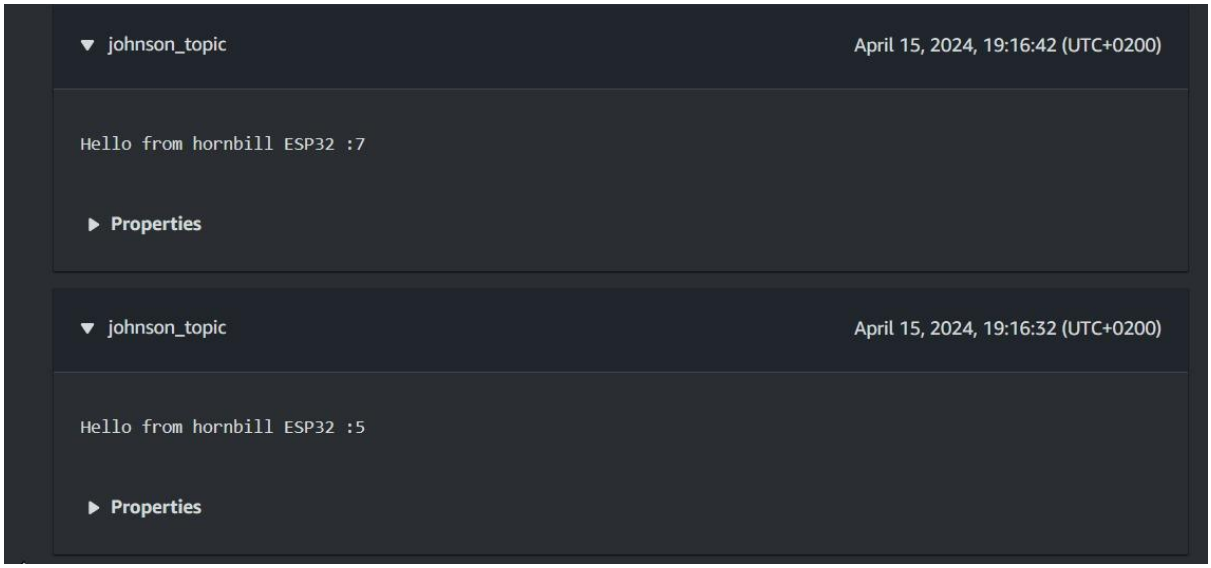

Figure 7: *Data being received from the ESP32 on the cloud..*

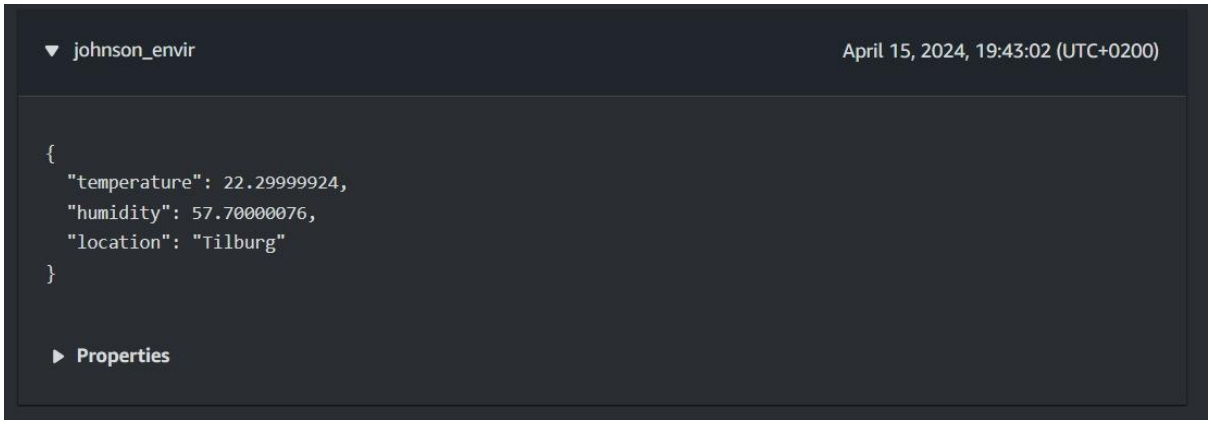

Figure 8: *Sensor data being received on the cloud MQTT.*

4. Discussion:

The purpose of this section is to dictate some things that were different particularly to what was expected.

4.1 MQTT node malfunction:

I haven't had this problem with my own implementation but with someone else in my classroom, I noticed that her MQTT block was not configuring properly. The node would be configured the same way as what was described but it could not connect. This could be fixed with the following solutions (solution 2 fixed the problem this classmate had):

1. Double check if you are using the **PRIVATE** key file rather than the public one.
2. Under the session section, uncheck the clean session box.

5. Bibliography:

[1] - ExploreEmbedded. (2017, September 19). *Secure IOT with AWS and Hornbill ESP32 using Arduino.* Instructables. https://www.instructables.com/Secure-IOT-With-AWS-and-Hornbill-ESP32-Using-Ardui/