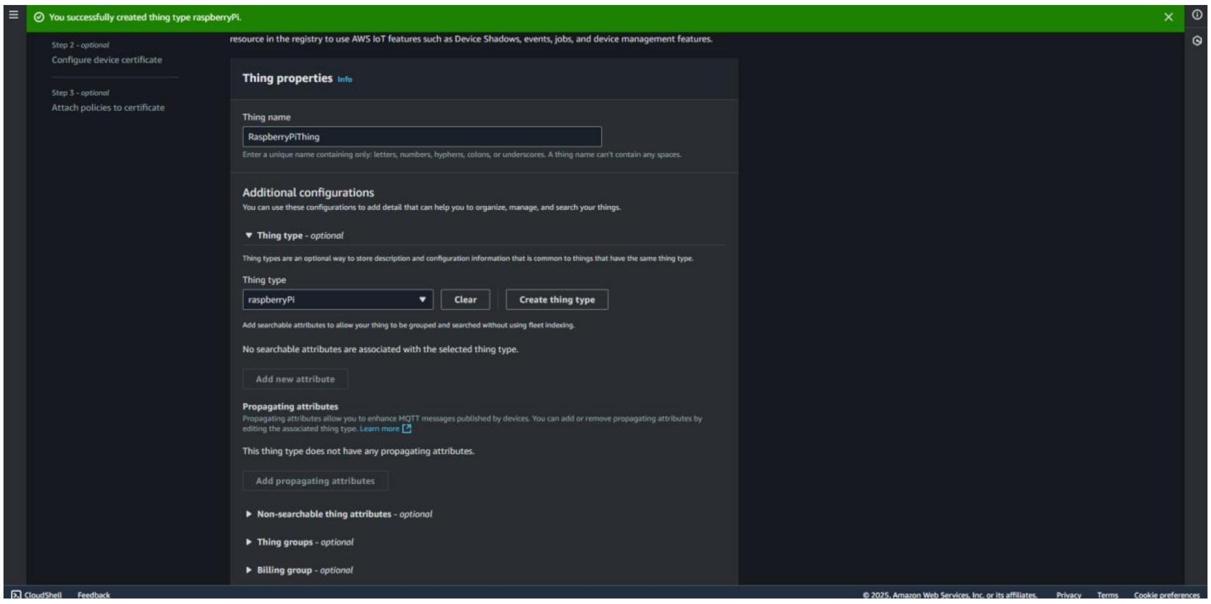
Lab06 (AWS IoT Core) Screenshots

1. Create Raspberry Pi Thing



You successfully created thing type raspberryPi.

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Thing properties Info

Thing name: RaspberryPiThing

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

▼ **Thing type** - *optional*

Thing types are an optional way to store description and configuration information that is common to things that have the same thing type.

Thing type: raspberryPi

Add searchable attributes to allow your thing to be grouped and searched without using fleet indexing.

No searchable attributes are associated with the selected thing type.

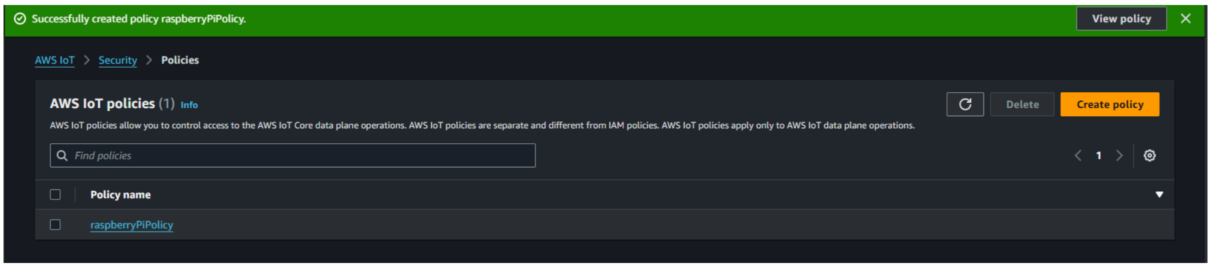
Propagating attributes

Propagating attributes allow you to enhance MQTT messages published by devices. You can add or remove propagating attributes by editing the associated thing type. Learn more

This thing type does not have any propagating attributes.

Fig. 1. Screenshot of creating the Raspberry Pi Thing on AWS IoT

2. Generate Certificate and Create Policy



Successfully created policy raspberryPiPolicy.

AWS IoT > Security > Policies

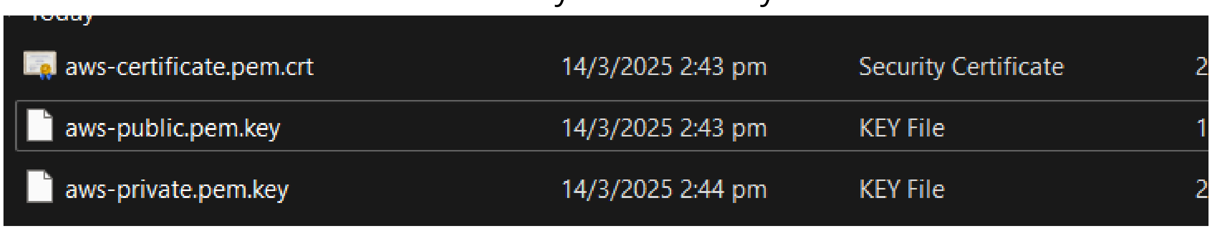
AWS IoT policies (1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations.

Policy name
raspberryPiPolicy

Fig. 2. Screenshot of generating certificate and creating policy on AWS IoT

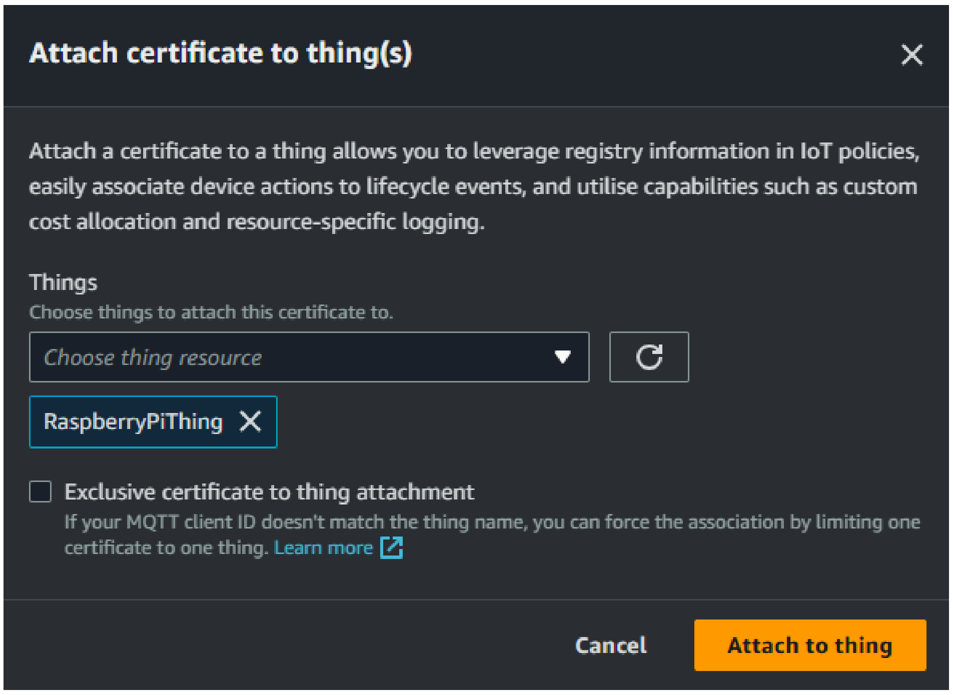
3. Download Certificate and Keys from Policy



aws-certificate.pem.crt	14/3/2025 2:43 pm	Security Certificate	2
aws-public.pem.key	14/3/2025 2:43 pm	KEY File	1
aws-private.pem.key	14/3/2025 2:44 pm	KEY File	2

Fig. 3. Screenshot of downloading the certificate and keys

4. Attach the Certificate to the Thing



Attach certificate to thing(s)

Attach a certificate to a thing allows you to leverage registry information in IoT policies, easily associate device actions to lifecycle events, and utilise capabilities such as custom cost allocation and resource-specific logging.

Things
Choose things to attach this certificate to.

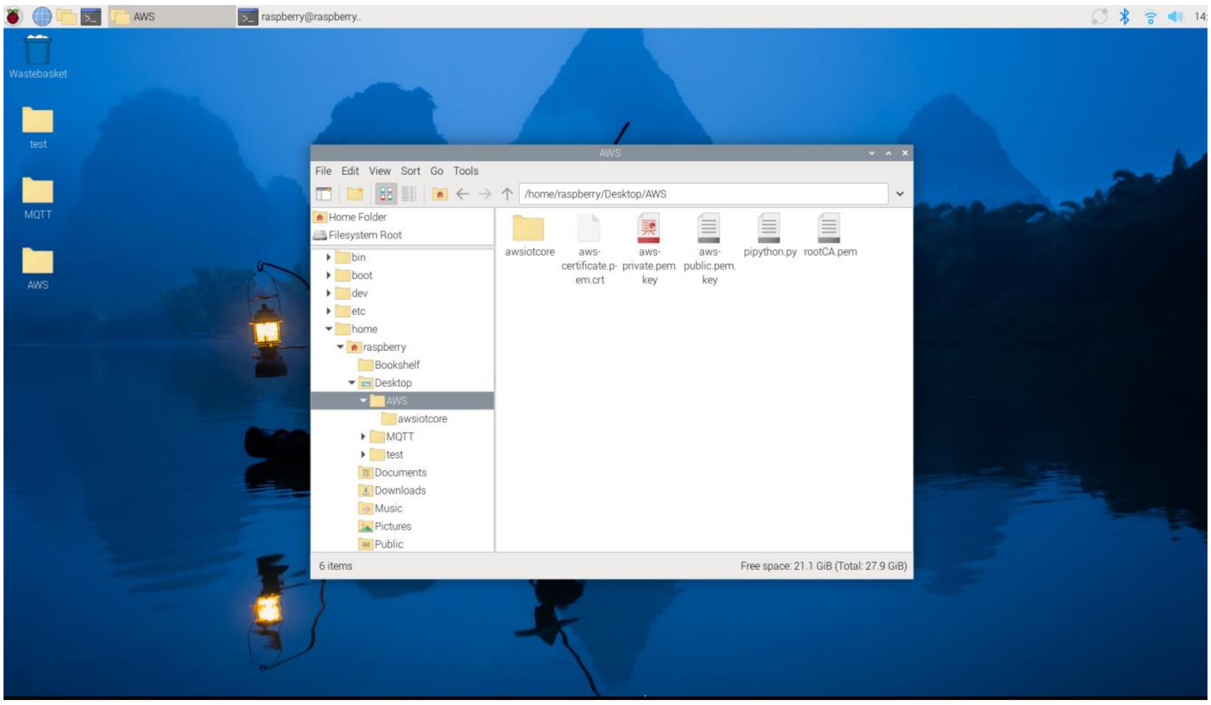
Choose thing resource

RaspberryPiThing

☐ Exclusive certificate to thing attachment
If your MQTT client ID doesn't match the thing name, you can force the association by limiting one certificate to one thing. Learn more

Fig. 4. Screenshot of attaching the certificate to the Thing that created earlier

5. Transfer Required Files to the Raspberry Pi



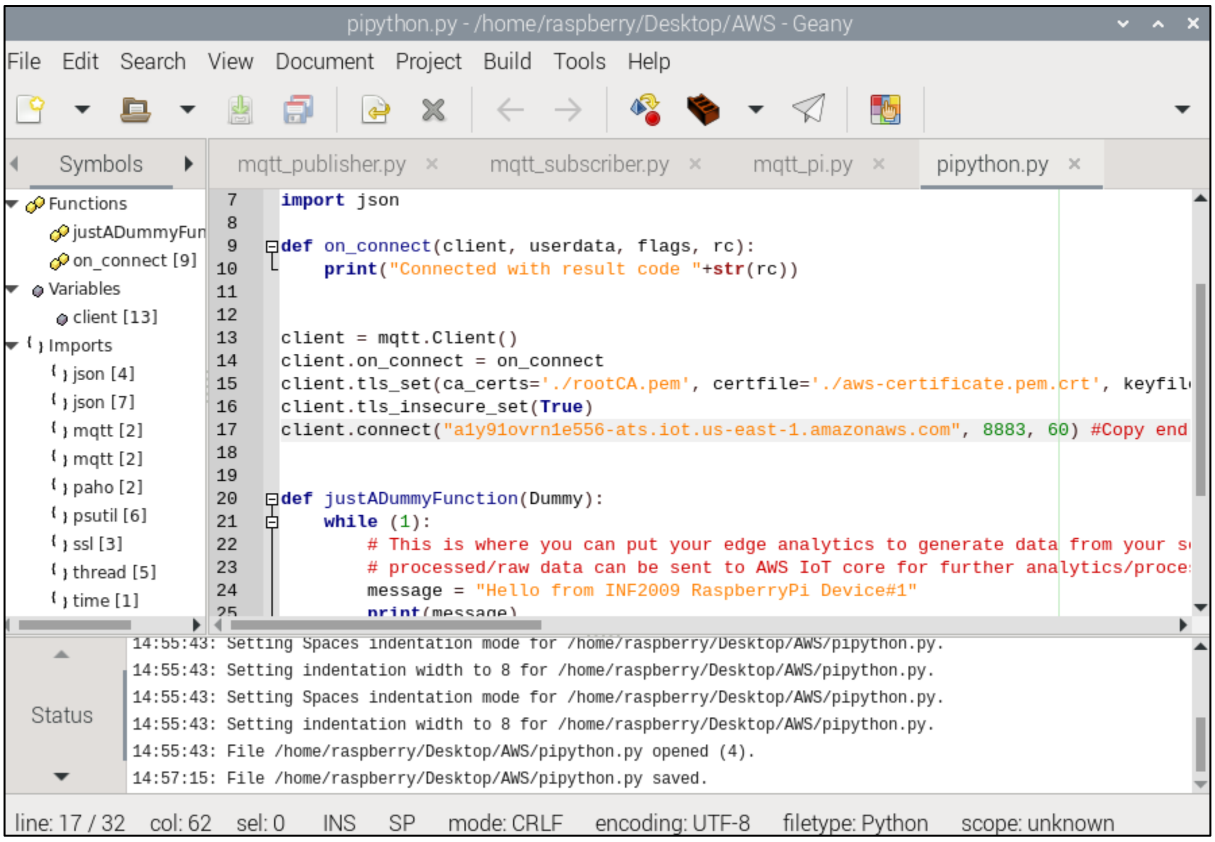
/home/raspberry/Desktop/AWS

awsiotcore
aws-certificate.pem.crt
aws-private.pem.key
aws-public.pem.key
pipython.py
rootCA.pem

6 items
Free space: 21.1 GiB (Total: 27.9 GiB)

Fig. 5. Screenshot of transferring required files to the Raspberry Pi

6. Update the Domain Name

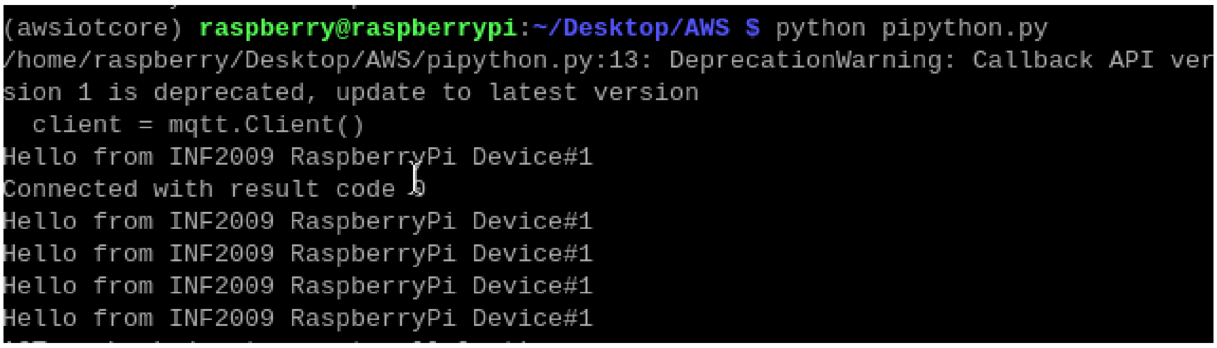


```
import json

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))


client = mqtt.Client()
client.on_connect = on_connect
client.tls_set(ca_certs='./rootCA.pem', certfile='./aws-certificate.pem.crt', keyfil
client.tls_insecure_set(True)
client.connect("a1y91ovrn1e556-ats.iot.us-east-1.amazonaws.com", 8883, 60) #Copy end


def justADummyFunction(Dummy):
    while (1):
        # This is where you can put your edge analytics to generate data from your s
        # processed/raw data can be sent to AWS IoT core for further analytics/proce
        message = "Hello from INF2009 RaspberryPi Device#1"
        print(message)
```

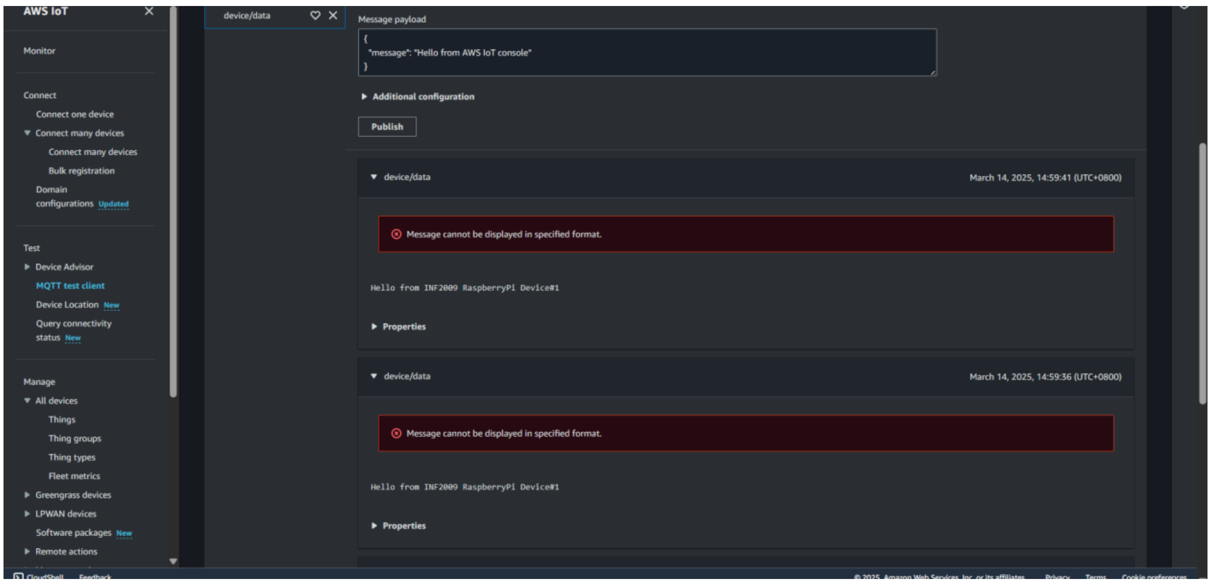
```
14:55:43: Setting Spaces indentation mode for /home/raspberry/Desktop/AWS/pipython.py.
14:55:43: Setting indentation width to 8 for /home/raspberry/Desktop/AWS/pipython.py.
14:55:43: Setting Spaces indentation mode for /home/raspberry/Desktop/AWS/pipython.py.
14:55:43: Setting indentation width to 8 for /home/raspberry/Desktop/AWS/pipython.py.
14:55:43: File /home/raspberry/Desktop/AWS/pipython.py opened (4).
14:57:15: File /home/raspberry/Desktop/AWS/pipython.py saved.
```

Fig. 6. Screenshot of updating the Domain Name in the `pipython.py` script

7. Subscribe to device/data and run Python script to send data to AWS



```
(awsiotcore) raspberry@raspberrypi:~/Desktop/AWS $ python pipython.py
/home/raspberry/Desktop/AWS/pipython.py:13: DeprecationWarning: Callback API ver
sion 1 is deprecated, update to latest version
  client = mqtt.Client()
Hello from INF2009 RaspberryPi Device#1
Connected with result code 0
Hello from INF2009 RaspberryPi Device#1
Hello from INF2009 RaspberryPi Device#1
Hello from INF2009 RaspberryPi Device#1
Hello from INF2009 RaspberryPi Device#1
```



```
{
  "message": "Hello from AWS IoT console"
}
```

device/data — March 14, 2025, 14:59:41 (UTC+0800)

Message cannot be displayed in specified format.

```
Hello from INF2009 RaspberryPi Device#1
```

device/data — March 14, 2025, 14:59:36 (UTC+0800)

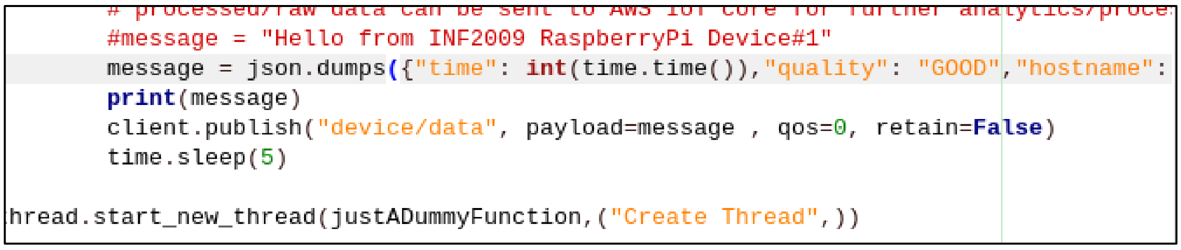
Message cannot be displayed in specified format.

```
Hello from INF2009 RaspberryPi Device#1
```

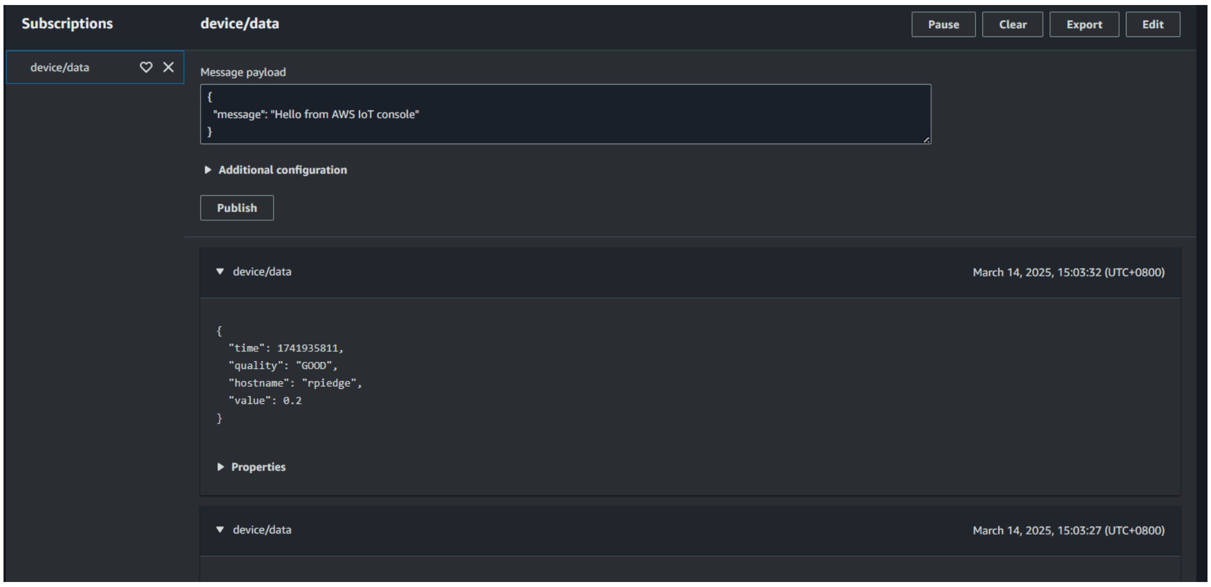
Fig. 7. Screenshot of running the `pipython.py` script

Lab06 (AWS IoT Core) Screenshots

8. Update code to send JSON payload so that AWS can read the data



```
        # processed/raw data can be sent to AWS IoT Core for further analytics/proce
        #message = "Hello from INF2009 RaspberryPi Device#1"
        message = json.dumps({"time": int(time.time()),"quality": "GOOD","hostname":
        print(message)
        client.publish("device/data", payload=message , qos=0, retain=False)
        time.sleep(5)

hread.start_new_thread(justADummyFunction,("Create Thread",))
```



```
{
 "message": "Hello from AWS IoT console"
}
```

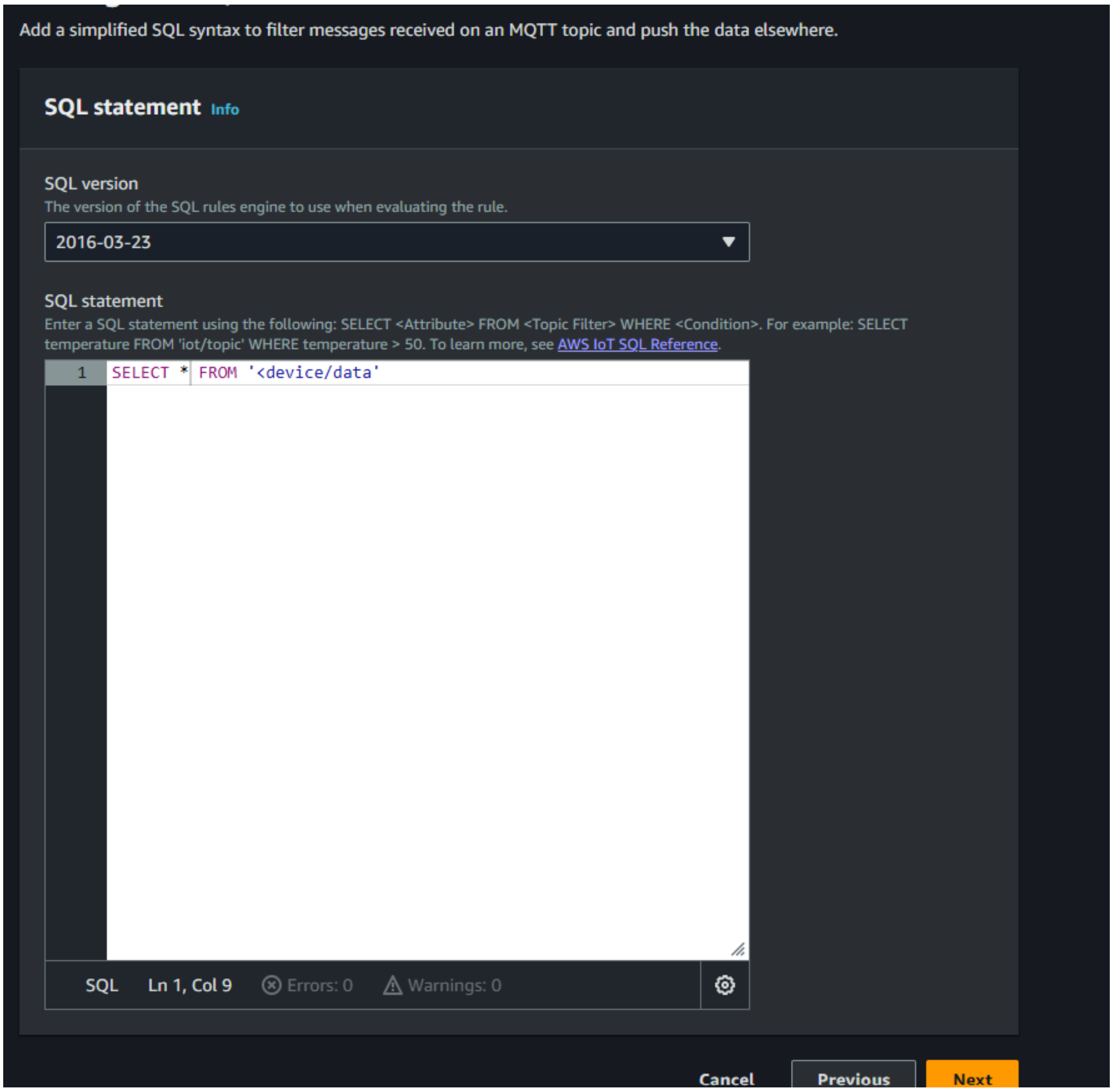
device/data March 14, 2025, 15:03:32 (UTC+0800)

```
{
  "time": 1741935811,
  "quality": "GOOD",
  "hostname": "rpiedge",
  "value": 0.2
}
```

device/data March 14, 2025, 15:03:27 (UTC+0800)

Fig. 8. Screenshot of the `pipython.py` script with the updated code and received payload

9. Create a rule and set SQL statement



Add a simplified SQL syntax to filter messages received on an MQTT topic and push the data elsewhere.

SQL statement Info

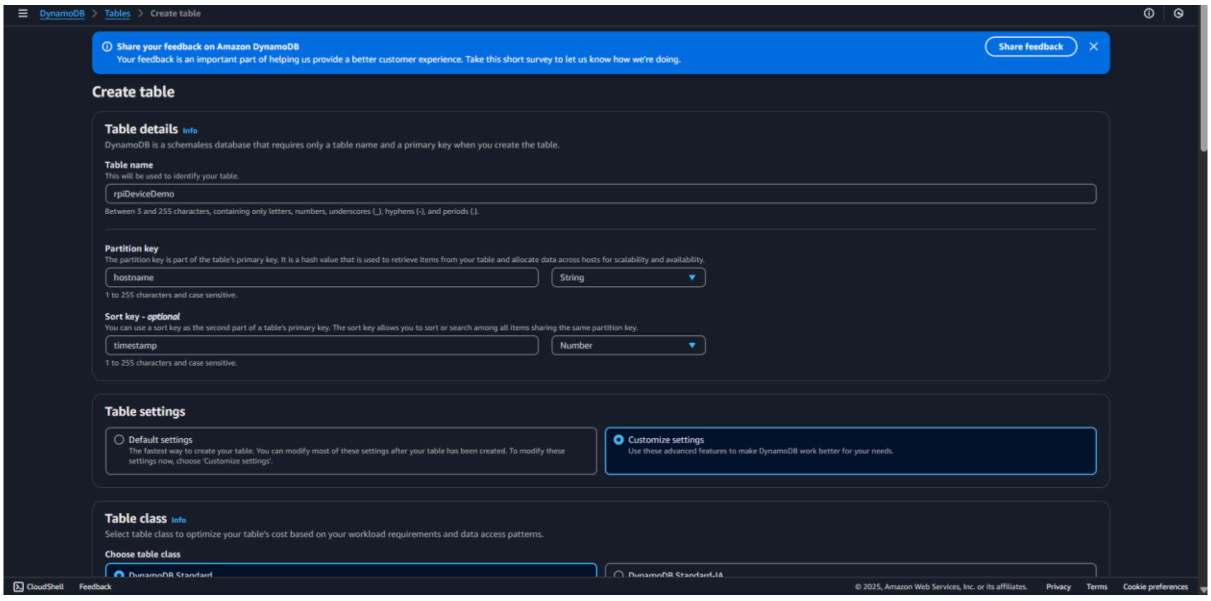
SQL version
The version of the SQL rules engine to use when evaluating the rule.
2016-03-23

SQL statement
Enter a SQL statement using the following: SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. For example: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. To learn more, see AWS IoT SQL Reference.

```sql
SELECT * FROM '<device/data'
```

Fig. 9. Screenshot of the SQL statement that is used to retrieve all data related to the device, from the database that is going to get created

10. Create DynamoDB table and configure rule



Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

Create table

Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.
rpiDeviceDemo
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
hostname | String
1 to 255 characters and case sensitive.

Sort key - *optional*
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
timestamp | Number
1 to 255 characters and case sensitive.

Table settings
Default settings [✓ Customize settings]
Default settings: The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose 'Customize settings'.
Customize settings: Use these advanced features to make DynamoDB work better for your needs.

Table class Info
Select table class to optimize your table's cost based on your workload requirements and data access patterns.
Choose table class
[✓ DynamoDB Standard] DynamoDB Standard-IA

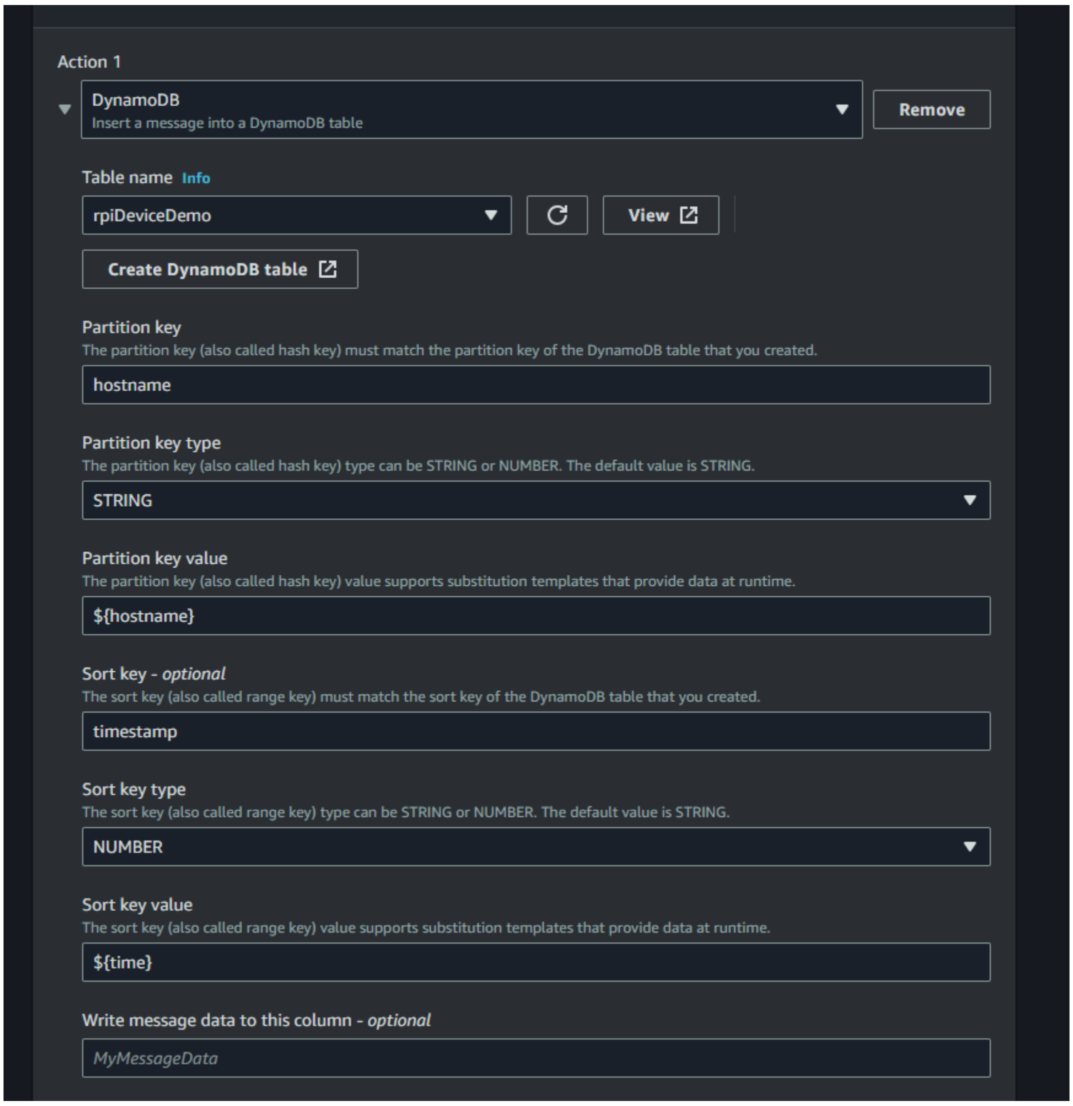

Action 1
DynamoDB
Insert a message into a DynamoDB table

Table name Info
rpiDeviceDemo

Partition key
The partition key (also called hash key) must match the partition key of the DynamoDB table that you created.
hostname

Partition key type
The partition key (also called hash key) type can be STRING or NUMBER. The default value is STRING.
STRING

Partition key value
The partition key (also called hash key) value supports substitution templates that provide data at runtime.
${hostname}

Sort key - *optional*
The sort key (also called range key) must match the sort key of the DynamoDB table that you created.
timestamp

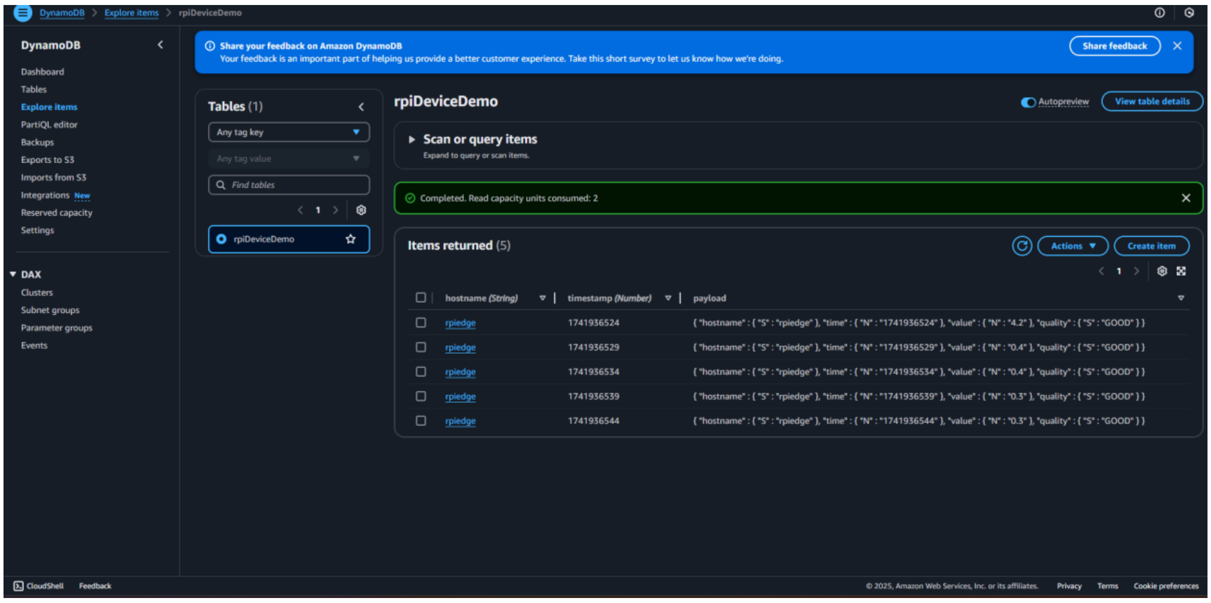
Sort key type
The sort key (also called range key) type can be STRING or NUMBER. The default value is STRING.
NUMBER

Sort key value
The sort key (also called range key) value supports substitution templates that provide data at runtime.
${time}

Write message data to this column - *optional*
MyMessageData

Fig. 10. Screenshot of the steps and configuration of creating a DynamoDB table in AWS

11. Run the Python script and see the DynamoDB table filled up with data



rpiDeviceDemo

Completed. Read capacity units consumed: 2

Items returned (5)

hostname (String)	timestamp (Number)	payload
rpiedge	1741936524	{ "hostname" : { "S" : "rpiedge" }, "time" : { "N" : "1741936524" }, "value" : { "N" : "4.2" }, "quality" : { "S" : "GOOD" } }
rpiedge	1741936529	{ "hostname" : { "S" : "rpiedge" }, "time" : { "N" : "1741936529" }, "value" : { "N" : "0.4" }, "quality" : { "S" : "GOOD" } }
rpiedge	1741936534	{ "hostname" : { "S" : "rpiedge" }, "time" : { "N" : "1741936534" }, "value" : { "N" : "0.4" }, "quality" : { "S" : "GOOD" } }
rpiedge	1741936539	{ "hostname" : { "S" : "rpiedge" }, "time" : { "N" : "1741936539" }, "value" : { "N" : "0.3" }, "quality" : { "S" : "GOOD" } }
rpiedge	1741936544	{ "hostname" : { "S" : "rpiedge" }, "time" : { "N" : "1741936544" }, "value" : { "N" : "0.3" }, "quality" : { "S" : "GOOD" } }

Fig. 11. Screenshot of the result shown on DynamoDB after running the script