Lab 7: AWS IoT: Real-Time Device Data to AWS IoT Core

1. What is AWS IoT Core and what are its key components in the context of this lab?

Answer:
AWS IoT Core is a managed cloud service that allows you to connect IoT devices securely and reliably. Its key components include:

- **IoT Thing:** A digital representation of a physical device (in this lab, a Raspberry Pi).
- **Certificates and Policies:** Security artifacts that authenticate and authorize device communication.
- **MQTT Broker:** The service endpoint that handles publish/subscribe messaging over protocols such as MQTT.
- **IoT Rules Engine:** Routes incoming device data to other AWS services (e.g., DynamoDB) for further processing.

---

2. Describe the steps involved in creating an IoT Thing in AWS IoT Core as outlined in the lab.

Answer:
The process involves:

- Logging into the AWS IoT Core console and navigating to **Manage → Things**.
- Selecting **Create single thing**, entering a unique thing name, and optionally grouping it by creating a thing type.
- Generating a new certificate automatically, which provides the necessary security credentials.
- Creating and attaching an IoT policy that defines what actions the thing is allowed to perform (for demonstration, using wildcards).
- Downloading the generated security files (Device Certificate, Public Key, Private Key) for use on the Raspberry Pi.

---

3. What are the purposes of the three security files (certificate, public key, and private key) generated during the IoT Thing creation?

Answer:

- **Device Certificate (aws-certificate.pem.crt):** Authenticates the device to AWS IoT Core.
- **Public Key (aws-public.pem.key):** Used in conjunction with the private key for secure communication; it's typically part of the certificate chain.
- **Private Key (aws-private.pem.key):** Must be kept secret; it is used to sign communications so that AWS IoT Core can verify the device's identity.
  Together, these files enable secure TLS connections between the device and AWS IoT Core.

---

4. Explain the role of an IoT policy in AWS IoT Core and why it is important.

Answer:
An IoT policy defines the permissions for a device by specifying which AWS IoT actions it can perform (such as publishing or subscribing to topics) and on which resources. It is important because it controls access and ensures that the device operates securely, adhering to the principle of least privilege. In the lab, a permissive policy (using wildcards) is used for demonstration, but in production, it should be restricted to necessary actions only.

---

5. How does the Python Paho MQTT client establish a secure connection to AWS IoT Core in this lab?

Answer:
The Paho MQTT client establishes a secure connection by:

- Configuring TLS using the security files (root CA, device certificate, and private key) via the tls_set() function.
- Optionally allowing insecure TLS settings using tls_insecure_set(True) (not recommended for production).
- Connecting to the AWS IoT Core endpoint (which you obtain from the AWS console) on port **8883**, which is designated for secure MQTT communication.

---

6. What is the significance of the AWS IoT Core endpoint in the MQTT connection command?

Answer:
The endpoint (e.g., "xxxxxxxx-ats.iot.ap-southeast-1.amazonaws.com") is the unique URL provided by AWS IoT Core for your account and region. It directs the MQTT client to

the correct server that handles your IoT device communications. Ensuring the endpoint is correct is crucial for a successful connection.

---

7. How can you test that your MQTT publisher and subscriber are working correctly with AWS IoT Core?

Answer:
You can test the setup by:

- Running the subscriber script (e.g., mqtt_subscriber.py) on one terminal to subscribe to a specific topic (like test/topic or device/data).
- Running the publisher script (e.g., mqtt_publisher.py) on another terminal to publish messages to the same topic.
- Verifying that the subscriber receives the messages in real time on the AWS IoT Core MQTT Test Client, which can also be monitored via the AWS console.

---

8. What is the role of JSON in the MQTT message published by the Raspberry Pi?

Answer:
Using JSON allows structured data to be sent in a standardized format. In the lab, the JSON message typically contains fields like timestamp, quality, hostname, and sensor values (e.g., CPU utilization). This structured format is essential for downstream processing, such as routing the data into a DynamoDB table using AWS IoT rules.

---

9. Describe how AWS IoT Rules are used to route device data into DynamoDB.

Answer:
AWS IoT Rules enable you to define SQL-like queries that filter and transform incoming MQTT messages. In this lab:

- A rule is created with an SQL statement that selects fields from the JSON payload (e.g., time, quality, hostname, value).
- The rule then specifies an action to insert the extracted data into a DynamoDB table.
- During rule configuration, you map the JSON fields to the table's primary (hostname) and sort (timestamp) keys, ensuring that the device data is stored correctly in DynamoDB.

---

10. What steps must be taken to set up a DynamoDB table for ingesting IoT data?

Answer:
You need to:

- Create a new DynamoDB table with an appropriate table name.
- Define the primary key (e.g., hostname) and a sort key (e.g., timestamp).
- Configure the table's read and write capacities if not using on-demand mode.
- In AWS IoT Core, configure the rule action to target this table and map the incoming JSON fields to the table keys.

---

11. What is the significance of transferring the security folder (containing the certificate, keys, and rootCA) to the Raspberry Pi?

Answer:
These security files are critical for establishing a secure TLS connection between the Raspberry Pi and AWS IoT Core. Transferring them correctly ensures that the device can authenticate itself and communicate securely with the AWS IoT endpoint. Without them, the device will not be able to establish a connection.

---

12. How does the sample Python code use multi-threading to publish messages, and why is it beneficial in an IoT context?

Answer:
The sample code uses _thread.start_new_thread() to launch a function that continuously publishes messages. This allows the main thread to handle the MQTT network loop (with client.loop_forever()) concurrently. Multi-threading is beneficial because it enables the device to perform background tasks (like sensor data collection or analytics) while maintaining a continuous, non-blocking connection to the MQTT broker.

---

13. What troubleshooting steps can you take if your Raspberry Pi is not able to connect to AWS IoT Core?

Answer:

- Verify that the **security files** (certificate, private key, root CA) are correctly placed and named.

- Confirm that the **AWS IoT Core endpoint** is correct and that you are using port 8883.
- Check the network connectivity and ensure that your Raspberry Pi has internet access.
- Look at the MQTT client's connection callback to see if any error codes are returned.
- Check AWS IoT Core's logs and certificates' status in the AWS console to ensure they are active.

---

14. How can you secure your MQTT communication beyond using TLS in this lab?

Answer:
Additional security measures include:

- Configuring **IoT policies** with strict permissions rather than using wildcards.
- Using **persistent sessions** and setting appropriate **QoS levels** to ensure reliable message delivery.
- Implementing **access control lists (ACLs)** and monitoring logs for suspicious activity.
- Keeping the security files secure and updating them periodically.

---

15. Explain how you would extend this lab assignment to combine image capture with MQTT communication.

Answer:
One extension could involve writing two Python scripts:

- A **subscriber** that listens on a designated MQTT topic (e.g., camera/trigger) for a trigger message.
- Upon receiving the trigger, the subscriber accesses the webcam using OpenCV to capture an image.
- The captured image is then encoded (e.g., converted to JPEG and then to a byte stream) and published to another MQTT topic (e.g., camera/image).
- This integration demonstrates how sensor data (camera images) can be combined with MQTT for a complete IoT solution.

---

16. What is the role of the root CA certificate in establishing a secure connection with AWS IoT Core?

Answer:
The root CA certificate (often provided by Amazon) is used to verify the authenticity of the AWS IoT Core server. It forms part of the TLS handshake, ensuring that the device is communicating with the genuine AWS service and not an impostor. Without it, the device would not trust the connection.

---

17. How does AWS IoT Core verify the identity of a device during connection?

Answer:
AWS IoT Core uses X.509 certificates to verify device identities. When a device attempts to connect, it presents its device certificate along with the corresponding private key. AWS IoT Core then checks this certificate against its trusted certificate authorities (using the root CA) and applies the attached IoT policy to determine what actions the device is allowed to perform.

---

18. How can you find your AWS IoT Core endpoint, and why is it important?

Answer:
You can find the AWS IoT Core endpoint in the AWS IoT Core Console under **Connect → Domain Configurations**. This endpoint (e.g., "xxxxxxxx-ats.iot.ap-southeast-1.amazonaws.com") is crucial because it directs your MQTT client to the correct server that handles your account's IoT communications, ensuring that your device messages reach AWS IoT Core.

---

19. What naming considerations should be taken into account when creating an IoT Thing in AWS IoT Core?

Answer:
When naming an IoT Thing, avoid using special characters (like # or $) to prevent issues in routing or identification. The name should be unique, descriptive, and consistent with your organization's naming conventions to simplify management and troubleshooting.

---

20. How would you create a more secure IoT policy compared to the permissive one used in the lab?

Answer:
A more secure IoT policy would:

- Restrict allowed actions to only those necessary for the device (e.g., iot:Publish on a specific topic instead of using wildcards).
- Limit resources to a defined set rather than all resources ("Resource": ["arn:aws:iot:region:account-id:thing/your-thing-name"]).
- Implement granular conditions to further control access. This reduces the risk of unauthorized actions if the device is compromised.

---

21. In the provided sample code, what does the call to client.tls_insecure_set(True) do, and what are its implications?

Answer:
The client.tls_insecure_set(True) call disables the verification of the server's hostname in the TLS certificate. This can be useful for testing or when using self-signed certificates; however, it weakens security by not fully validating the server's identity. In production, it is best to remove this setting to ensure proper certificate validation.

---

22. How can you configure your Raspberry Pi to automatically start sending data via MQTT on boot?

Answer:
You can create a systemd service or a cron job that runs your MQTT publisher script on boot. For example, creating a systemd unit file that specifies the execution of the script ensures that data transmission starts automatically whenever the Raspberry Pi boots.

---

23. What is the purpose of an AWS IoT Rule in the context of data ingestion into DynamoDB?

Answer:
An AWS IoT Rule processes incoming MQTT messages based on a defined SQL-like query. It extracts specific fields from the message payload and routes the data to a destination, such as a DynamoDB table. This enables automatic and real-time ingestion of device data into a database for further analysis or storage.

---

24. How do you map fields from the MQTT message payload to a DynamoDB table in an IoT Rule?

Answer:
When creating an IoT Rule, you write an SQL statement that selects fields from the JSON

payload (for example, "SELECT time, quality, hostname, value FROM 'device/data'"). In the Rule's action configuration, you then map these fields to the DynamoDB table's primary key and sort key. This ensures that each incoming message is correctly inserted into the table with the proper key structure.

---

25. What troubleshooting steps can you take if data is not appearing in your DynamoDB table after setting up an IoT Rule?

Answer:

- Verify that your IoT Rule is active and properly configured with the correct SQL statement and DynamoDB action.
- Check the IAM role assigned to the rule to ensure it has permissions to write to DynamoDB.
- Confirm that the MQTT messages are being published to the correct topic and that the JSON payload matches the expected format.
- Use the AWS IoT Core Test Client to manually publish messages and observe if the rule triggers.
- Inspect CloudWatch logs (if enabled) for any errors related to the rule or DynamoDB actions.

---

26. How does the sample code ensure that the data being sent to AWS IoT Core is in the correct format for DynamoDB ingestion?

Answer:
The sample code uses json.dumps() to convert a Python dictionary into a JSON-formatted string. This string includes fields such as time, quality, hostname, and value, which are then sent over MQTT. The IoT Rule is configured to extract these fields from the JSON message, ensuring that the data is in the expected format for insertion into the DynamoDB table.

---

27. What role does the psutil.cpu_percent() function play in the sample MQTT publishing code?

Answer:
The psutil.cpu_percent() function measures the current CPU utilization of the Raspberry Pi. In the sample code, this value is included in the JSON payload sent to AWS IoT Core,

allowing you to monitor the device's performance and potentially trigger alerts or actions if CPU usage exceeds certain thresholds.

---

28. Describe the process of transferring the AWS IoT Core security files from your computer to your Raspberry Pi.

Answer:
The security files (device certificate, public key, private key, and root CA) must be transferred to the Raspberry Pi so that the MQTT client can establish a secure TLS connection. This can be done using file transfer tools such as **WinSCP** on Windows or **scp** on Linux/Mac:

bash

Copy

scp -r aws_iot_core_folder <username>@<rpi_IP_address>:/desired/path

This command securely copies the folder and its contents to the specified path on the Raspberry Pi.

---

29. How can you verify that your MQTT messages are successfully reaching AWS IoT Core?

Answer:
You can use the **AWS IoT Core MQTT Test Client** available in the AWS console. By subscribing to the topic (e.g., "device/data"), you can observe real-time messages as they arrive. Additionally, you can monitor the Mosquitto broker logs or use CloudWatch if you have set up logging for your IoT Rule.

---

30. What benefits does using AWS IoT Core offer compared to a local MQTT broker running on the Raspberry Pi?

Answer:
AWS IoT Core offers:

- **Scalability:** It can handle large volumes of messages from many devices.
- **Security:** Built-in features for secure communication and device authentication.
- **Integration:** Direct integration with other AWS services (such as DynamoDB, Lambda, and CloudWatch) for data processing, storage, and analytics.

- **Managed Service:** AWS handles maintenance, updates, and scalability, reducing the overhead on the user.