ByteWise Educators

IOT FUNDAMENTAL

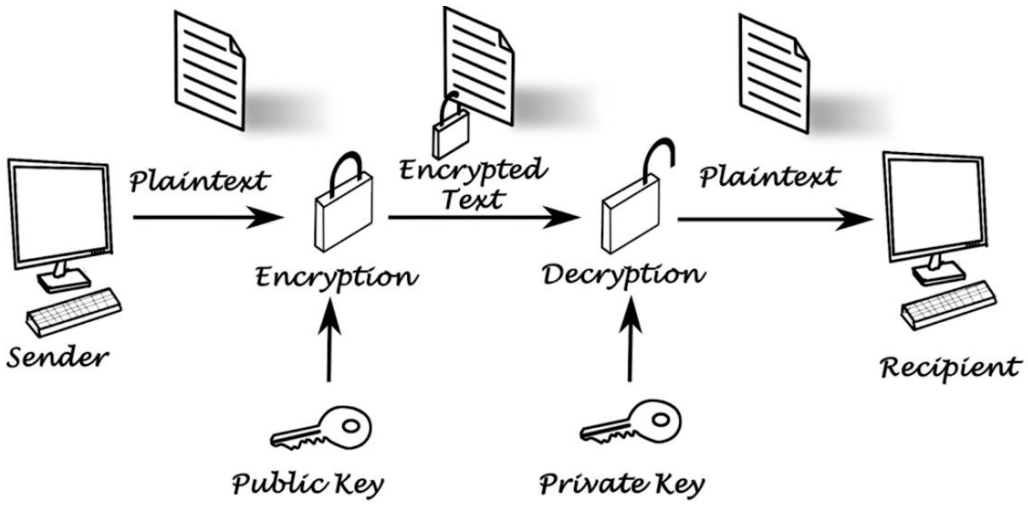
BUILDING WITH AWS, RASPBERRY PI, AND ESP32

DR. MUHAMMAD KHAIR NOORDIN
MUHAMMAD AFANDI AZMI

Module 1: Connecting Things to AWS IoT Core

A. Registering a thing

1. Choose the region that suitable for you, for example Asia Pacific (Singapore). Make sure that you choose the same region for every AWS service that you going to use in this project.
2. Search for IoT Core in services
3. Click on All devices > Things. Then click on Create Things button.
4. Pick Create single thing
5. Give it a name for your thing and then click Next
6. Choose auto generate new certificate and click Next
7. Create a new policy for IoT core.
8. Give a name to the policy and just pick * for Policy action and type * policy resource. Click create.
9. Finish the process by clicking Create thing.
10. Now, download the certificate and keys as these keys won't show up anymore in the future. Three important certificates: Device certificate, private key file and Amazon Root CA 1.
11. This is the process of how the data send from sender to recipient.



AWS IoT Policies:

- AWS IoT policies are used to control access to IoT resources like Things, Thing Types, and Topic Rules.
- IoT policies are associated with IoT entities, such as Things and Thing Groups.
- IoT policies allow fine-grained control over what actions are allowed on IoT resources and under what condition.
- Used for connecting, subscribing and publishing

IAM Policies:

- IAM policies are used to manage access to AWS services and resources.
- You can attach IAM policies to IAM users, groups, or roles to define what actions they are allowed or denied when interacting with IoT Core.

Copyright 2023 ByteWise Educator

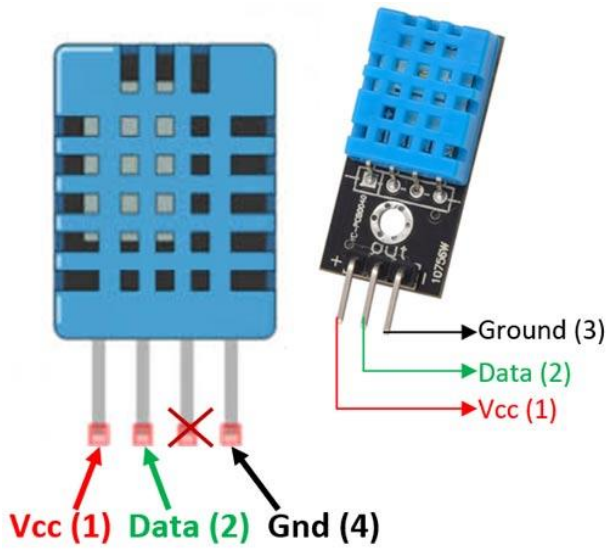
- IAM policies can be very granular and specific, allowing you to control access to AWS IoT on a per-user or per-role basis.
- You can use conditions and policy variables in IAM policies to further customize access control.

In AWS IoT Core, certificates are used for device authentication and secure communication between devices and AWS services. When you create a "Thing" in AWS IoT, it's common for certificates to be generated for each Thing because these certificates are used to establish secure connections between devices and AWS IoT Core.

Here's why certificates are generated for each Thing in AWS IoT Core:

- Device Authentication: Each device (Thing) needs a unique X.509 certificate to authenticate itself when connecting to AWS IoT Core. These certificates serve as a way to ensure that the device is legitimate and authorized to interact with AWS IoT services. This is a fundamental aspect of security in IoT.
- Secure Communication: The certificates are used for secure communication over MQTT or other protocols. When a device connects to AWS IoT Core, it presents its certificate, and the server validates it. Once the device is authenticated, it can securely send and receive data from AWS IoT Core.
- Granular Access Control: Certificates are often associated with AWS IoT policies, which define what actions a device is allowed to perform. This allows you to control which devices can access specific AWS IoT resources and what they can do. Each certificate is associated with one or more policies to determine its access rights.
- Management and Revocation: Certificates also provide a mechanism for managing and revoking access. If a device is compromised or decommissioned, you can revoke its certificate to prevent further access.

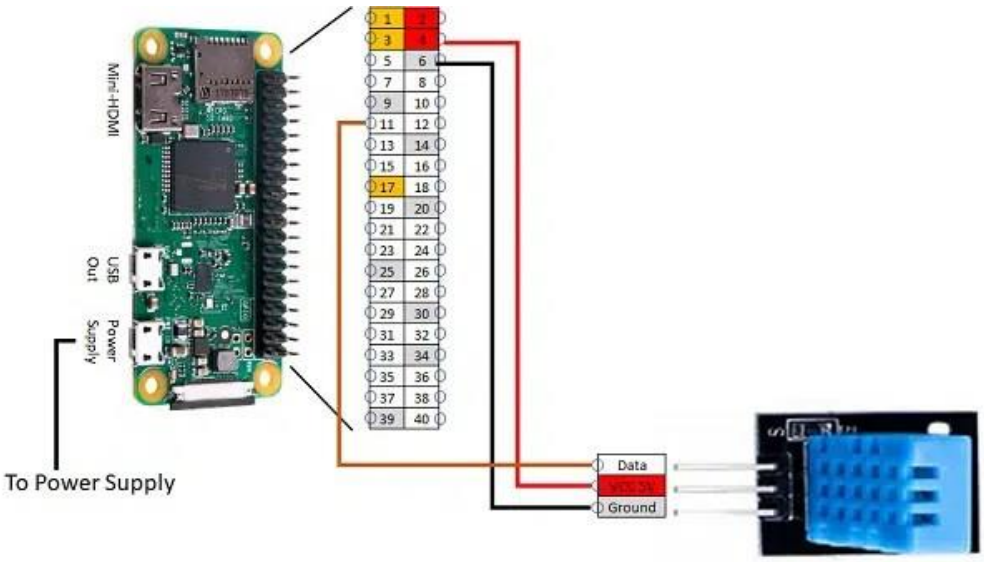
B. Connecting Raspberry Pi to AWS IoT



Copyright 2023 ByteWise Educator

DHT11 Specifications

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%



1. Connect DHT11 to RPi Zero 2 W as shown in figure above
2. We are going to use AWS IoT Core to get the reading from DHT11. The method we are going to use is called as MQTT.

MQTT, which stands for Message Queuing Telemetry Transport, is a lightweight and widely-used messaging protocol designed for resource-constrained devices and unreliable network connections. MQTT is employed in various IoT (Internet of Things) applications, and it is the primary communication protocol used in AWS IoT Core.

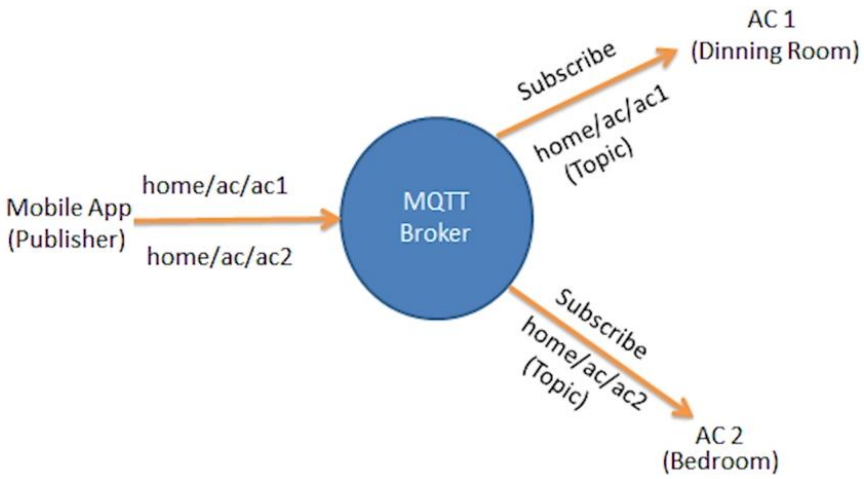
Here are some key characteristics and concepts related to MQTT:

- Publish/Subscribe Model: MQTT uses a publish/subscribe communication pattern. In this model, devices or clients (publishers) send messages to topics, and other devices or clients (subscribers) express interest in receiving messages from specific topics. This decouples the sender (publisher) from the receiver (subscriber), allowing for flexible and efficient communication.
- Quality of Service (QoS): MQTT offers three levels of Quality of Service to ensure message delivery reliability. These levels range from 0 (At most once) to 2 (Exactly once). You can choose the appropriate QoS level based on your application's requirements.

Copyright 2023 ByteWise Educator

- Lightweight: MQTT is designed to be lightweight, making it well-suited for IoT devices with limited processing power, memory, and bandwidth. The protocol minimizes overhead and is efficient for low-power and low-bandwidth environments.
- Retained Messages: MQTT supports retained messages. When a message is published with the "retain" flag set, the broker stores the last message sent on that topic. Subscribers receive the retained message immediately upon subscribing.

In the context of AWS IoT Core, MQTT is the default communication protocol. Devices that connect to AWS IoT Core use MQTT to send and receive messages. AWS IoT Core acts as the MQTT broker and facilitates the secure and reliable exchange of data between devices and AWS cloud services. Devices connect to specific AWS IoT topics to publish and subscribe to messages, and AWS IoT policies define who can access these topics and what actions they can perform. AWS IoT Core supports secure communication with MQTT over TLS (Transport Layer Security) to ensure data privacy and authenticity. It also integrates with other AWS services, making it easier to process and analyze IoT data in the cloud.

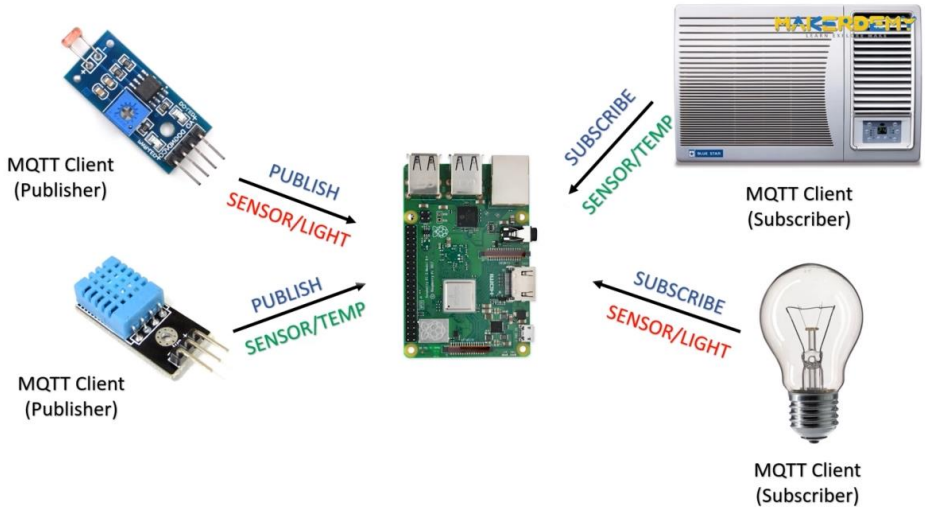


- Broker: The MQTT broker is a central server or intermediary responsible for receiving and distributing messages between clients (publishers and subscribers). It is a critical component of the MQTT architecture. When a client publishes a message to a specific topic, the broker receives it and then delivers that message to all clients subscribed to that topic.
- Publisher: A publisher is an MQTT client (often an IoT device or application) that sends messages to the broker. Publishers are responsible for specifying the topic to which they want to publish a message. They send messages to the broker, which, in turn, distributes the messages to the appropriate subscribers.
- Subscriber: A subscriber is an MQTT client that expresses interest in receiving messages on specific topics. Subscribers indicate the topics they are interested in, and the broker delivers messages published to those topics to the subscribing clients. Subscribers can choose to receive messages on multiple topics, and they are notified when new messages arrive.
- Topic: A topic is a hierarchical, user-defined string that acts as a channel or a subject to which messages are published and subscribed. Topics provide a way to categorize

Copyright 2023 ByteWise Educator

and filter messages, allowing clients (subscribers) to express interest in receiving messages on specific topics. Topics in MQTT are typically represented as strings, and they can be structured hierarchically using slashes to separate levels.
Examples of MQTT topics:

- "home/living-room/temperature"
- "devices/sensor-1/data"
- "iot/devices/+/data" (using "+" wildcard to match multiple devices) (“+” is single-level wildcard)
- "sensors/#" (using "#" wildcard to match all sensors) (“#” is multi-level wildcard)



3. In order to use RPi, we need to install Raspberry Pi OS into the SD Card. Go to https://www.raspberrypi.com/software/ and download the imager for Windows.
4. Use SD card reader and insert it into USB port of your PC. Make sure your SD card is empty. If not, please format it.
5. Choose the OS (pick recommended OS) and pick the storage (your SD Card). Next, click on gear icon to open the advanced option.
6. Make sure to tick on Enable SSH, set username and password, Configure wireless LAN. Then, click Write and wait for the process complete.
7. Go to https://www.realvnc.com/en/connect/download/viewer/ to download the VNC viewer. This will enable you to see the screen of the RPi within Windows environment
8. Download PuTTY at https://www.putty.org/. This will enable you to connect and interact with your RPi by entering the IP address of your RPi. Make sure your PC is connected to same WiFi with your RPi.
9. Open PuTTY and key in the IP address of your RPi and click Open.
10. Type your login username and password to access RPi.
11. Type sudo raspi-config and then go to Interface Options > VNC > Enable / Yes
12. Close the config.
13. Open RealVNC viewer and key in your IP address. You will able to see the screen of RPi.
14. Next, copy and paste the certificates that you download previously and paste it in RPi main folder.
15. Now, we are going to install AWS IoT Core SDK into our RPi.

Copyright 2023 ByteWise Educator

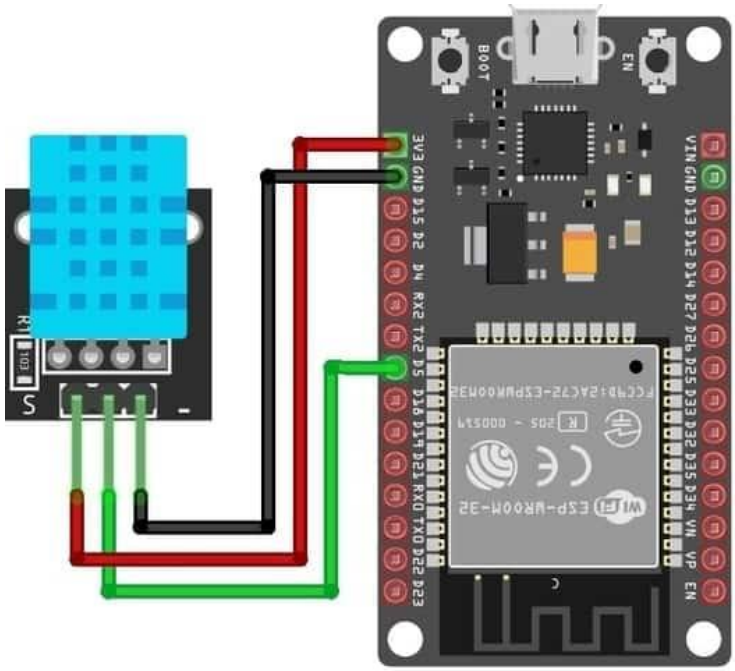
16. Type this or paste this link in the PuTTY command line to clone the folder:
    `git clone https://github.com/aws/aws-iot-device-sdk-python`
17. Type `cd aws-iot-device-sdk-python`
18. Next type *`sudo python setup.py install`*
19. Type this or paste this link in the PuTTY command line to clone the folder:
    `git clone https://github.com/adafruit/Adafruit_Python_DHT.git`
20. Type `cd Adafruit_Python_DHT`
21. Next type `sudo python setup.py install`.
22. Now, we will create a new python file. Type `sudo nano project_name.py`
23. Paste this coding into the file:

```
import Adafruit_DHT
import time
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Set the DHT sensor type (DHT11 or DHT22)
sensor = Adafruit_DHT.DHT11

# Set the GPIO pin where the DHT sensor is connected
pin = 4

# AWS IoT Core configuration
iot_endpoint = " a1whcump0jgijv-ats.iot.ap-southeast-1.amazonaws.com "
root_ca_path = "/home/kayz/certs/AmazonRootCA1.pem"
private_key_path = "/home/kayz/certs/myprivate.pem.key"
certificate_path = "/home/kayz/certs/security-certificate.pem.crt"
client_id = "dht11-device"

# Initialize the AWS IoT MQTT client
client = AWSIoTMQTTClient(client_id)
client.configureEndpoint(iot_endpoint, 8883)
client.configureCredentials(root_ca_path, private_key_path,
certificate_path)

# Connect to AWS IoT Core
client.connect()

while True:
    try:
        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
        if humidity is not None and temperature is not None:
            payload = '{{ "temperature": {0:0.1f}, "humidity": {1:0.1f}
}}'.format(temperature, humidity)
            client.publish("pubdht11", payload, 1)
            print("Published: " + payload)
        else:
            print("Failed to read sensor data.")
        time.sleep(60)  # Publish data every 60 seconds
    except Exception as e:
        print("Error:", str(e))

# Disconnect from AWS IoT Core (this code may not be reached in
practice)
client.disconnect()
```

Copyright 2023 ByteWise Educator

24. Save by click Ctrl X and then Y.
25. Run the code by typing *python project_name.py*
26. Go to IoT Core and click on MQTT test client.
27. Write the topic that you published previously and click on subscribe.

C. Connecting ESP32 to AWS IoT Core



1. Connect DHT11 to ESP32 as shown in figure above
2. Repeat the same process that you did earlier by creating new things and attach a policy (you may create new policy or use previous policy)
3. Download Arduino IDE at https://downloads.arduino.cc/arduino-1.8.19-windows.exe
4. Install Arduino and then open it. Go to File > Preferences. Paste this link https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json into the Additional Board URL box.
5. Go to Tools > Board > Boards Manager. Search for esp32 and choose 2.0.8 version, and install it.
6. Go to Sketch > Include library > Manage libraries. Search for ArduinoJSON and install the latest version.
7. Go to Sketch > Include library > Manage libraries. Search for DHT sensor library and install it.
8. Go to Sketch > Include library > Manage libraries. Search for Adafruit Unified Sensor and install it.
9. Go to Sketch > Include library > Manage libraries. Search for PubSubClient and install it.
10. Go to Sketch > Include library > Manage libraries. Search for NTPClient and install it.
11. Paste this code in Arduino

Copyright 2023 ByteWise Educator

```
#include <pgmspace.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <WiFi.h>
#include <DHT.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

#define THINGNAME "your thing" // Change this

const char WIFI_SSID[] = "wifi SSID";             // Change this
const char WIFI_PASSWORD[] = "wifi passowrd";  // Change this
const char AWS_IOT_ENDPOINT[] = "your endpoint"; // Change this

// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)EOF";

// Device Certificate                                               //
Change this
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)KEY";

// Device Private Key                                               //
Change this
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
)KEY";

#define DHTPIN 4      // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11

#define AWS_IOT_PUBLISH_TOPIC "your topic"
#define AWS_IOT_SUBSCRIBE_TOPIC "your topic"

float h;
float t;

DHT dht(DHTPIN, DHTTYPE);

WiFiClientSecure net = WiFiClientSecure();
PubSubClient client(net);

const char* NTP_SERVER = "pool.ntp.org";
const int NTP_TIMEZONE = 8;  // Kuala Lumpur timezone offset in hours
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, NTP_SERVER, NTP_TIMEZONE * 3600, 60000);

void connectAWS()
```

Copyright 2023 ByteWise Educator

```
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  // Configure WiFiClientSecure to use the AWS IoT device credentials
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we defined earlier
  client.setServer(AWS_IOT_ENDPOINT, 8883);

  // Create a message handler
  client.setCallback(messageHandler);

  Serial.println("Connecting to AWS IoT");

  while (!client.connect(THINGNAME))
  {
    Serial.print(".");
    delay(100);
  }

  if (!client.connected())
  {
    Serial.println("AWS IoT Timeout!");
    return;
  }

  // Subscribe to a topic
  client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);

  Serial.println("AWS IoT Connected!");
}

void publishMessage()
{
  StaticJsonDocument<200> doc;
  String dateTimeStr = timeClient.getFormattedDate();
  int spaceIndex = dateTimeStr.indexOf('T'); // Find the 'T' character that
separates date and time
  String dateStr = dateTimeStr.substring(0, spaceIndex); // Extract the
date part
  String timeStr = dateTimeStr.substring(spaceIndex + 1,
dateTimeStr.length() - 1); // Extract the time part
  doc["date"] = dateStr;
  doc["time"] = timeStr;
  doc["temperature"] = t;
  doc["humidity"] = h;
```

Copyright 2023 ByteWise Educator

```
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer);

  client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}

void messageHandler(char* topic, byte* payload, unsigned int length)
{
  Serial.print("incoming: ");
  Serial.println(topic);

  StaticJsonDocument<200> doc;
  deserializeJson(doc, payload);
  const char* message = doc["message"];
  Serial.println(message);
}

void setup()
{
  Serial.begin(115200);
  connectAWS();
  dht.begin();

  // Initialize the NTP client
  timeClient.begin();
  timeClient.update();
}

void loop()
{
  h = dht.readHumidity();
  t = dht.readTemperature();

  if (isnan(h) || isnan(t))
  {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.println(F("°C "));

  publishMessage();
  client.loop();

  // Update the NTP client
  timeClient.update();

  delay(1000);
}
```

Copyright 2023 ByteWise Educator

12. Modify the above code accordingly.
13. Connect your ESP32 to your PC and go to Tools > Board > ESP32 Arduino > ESP32 Dev module
14. Next, choose the correct port of ESP32 by going to Tools > Port.
15. Upload the code to ESP32 by clicking on the arrow icon.
16. Go to IoT Core and click on MQTT test client.
17. Write the topic that you published previously and click on subscribe.

Copyright 2023 ByteWise Educator

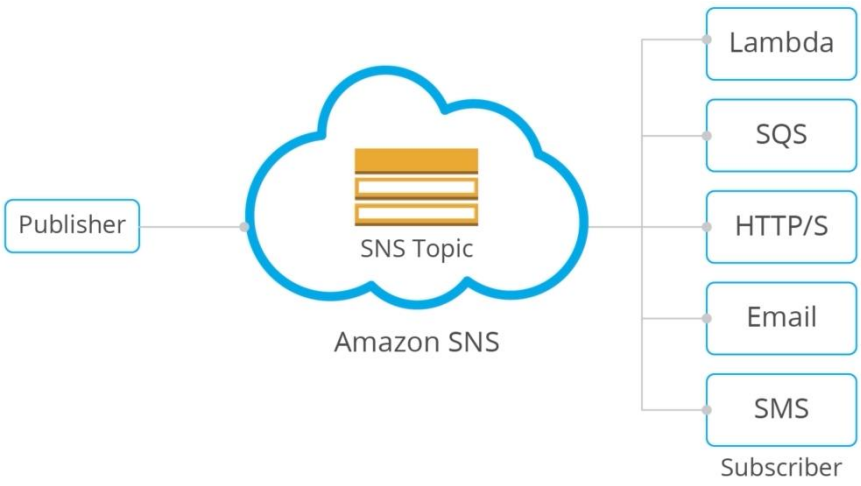
Module 2: Sending Notifications and Scheduling Events

A. Introduction to Amazon SNS

Amazon SNS, or Amazon Simple Notification Service, is a fully managed messaging service provided by Amazon Web Services (AWS). It enables you to send messages or notifications to a distributed set of subscribers via various delivery protocols. SNS is a versatile service that can be used to build applications, services, and systems that require real-time communication and notifications.

Key features and concepts of Amazon SNS include:

- Publish/Subscribe Model: SNS follows a publish/subscribe messaging model, similar to MQTT. In this model, there are publishers who send messages and subscribers who receive them. Topics act as communication channels where messages are published, and subscribers express interest in specific topics.



- Multiple Protocols: SNS supports a wide range of delivery protocols, including:
    1. SMS and text messaging
    2. Email

Copyright 2023 ByteWise Educator

3. Push notifications to mobile devices (e.g., iOS, Android)
4. Application endpoints (e.g., AWS Lambda functions, HTTP/S endpoints)
5. Amazon SQS (Simple Queue Service) queues
6. AWS Fargate tasks
7. HTTP/HTTPS

- Fanout: SNS allows you to fan out messages to multiple subscribers simultaneously, making it suitable for scenarios where you want to send notifications to a large number of recipients.

1. Amazon SNS will be used to send alert / notification to our email when DHT11 detects high temperature, for example when temperature > 30 degree Celsius.
2. Go to Amazon and search for SNS.
3. Click on Topics > Create topic.
4. Choose Standard. Give a name for your topic and display name. After that click Create Topic.
5. Now, go back to SNS homepage and click on Subscriptions.
6. Select the Topic ARN and choose Email as the Protocol.
7. Fill up the email that going to receive notification from Amazon SNS and click Create subscription.
8. Open you email and you need to confirm the subscription that you made through SNS.
9. You may add other email or phone number to receive the notification as well.

B. Sending sensor data as Notifications

AWS IoT Rule, which is a powerful and flexible way to process and route data from connected IoT devices to other AWS services, such as Amazon S3, AWS Lambda, Amazon DynamoDB, and more. IoT Rules are at the core of IoT data processing and allow you to specify actions that should be taken when specific conditions are met within the incoming data from IoT devices.

Key features and concepts related to AWS IoT Rules include:

- SQL-like Queries: IoT Rules use a SQL-like language for defining conditions that trigger actions. You can create rules based on the content of MQTT messages published to IoT topics, allowing you to filter and transform data in real-time.

Copyright 2023 ByteWise Educator

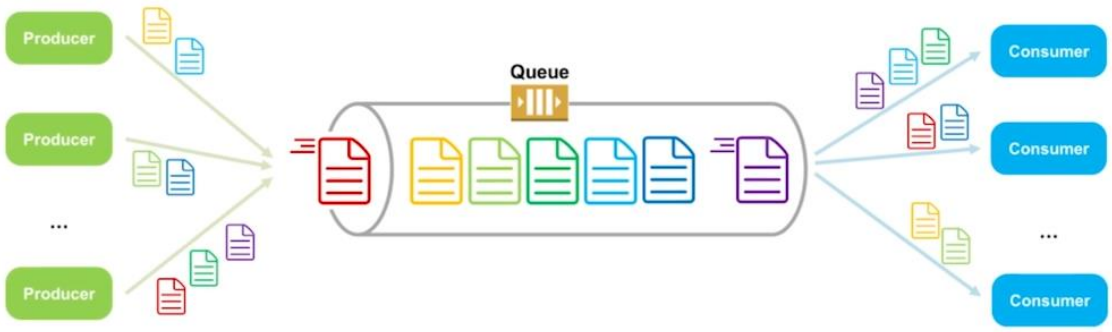
- Topic-Based Routing: Rules are associated with specific MQTT topics, which serve as the entry points for incoming data from IoT devices. When a message is published to a topic, it is processed by the rules associated with that topic.
- Actions: Each IoT Rule specifies one or more actions to be taken when the rule's conditions are met. Actions can include sending data to other AWS services, invoking AWS Lambda functions, storing data in databases, or sending notifications.
- Multiple Conditions: IoT Rules support multiple conditions, allowing you to create complex logic for routing and processing IoT data.
- Wildcards: Rules can use wildcards in the MQTT topic filter to match multiple topics that share a common structure.

1. Go to amazon and search for IoT Core.
2. Click on Message Routing > Rules > Create rule
3. Give a rule name and click Next.
4. Type this in SQL statement. Replace 'Your Topic' with the topic that you initialized in ESP32.

```
SELECT * FROM 'Your Topic' WHERE temperature > 30
```

5. Choose SNS as the rule actions. Pick the topic that you created previously. Select RAW as the format.
6. Pick the IAM role or you can create new role and click Next. Finally, click Create.
7. Now, test the function of SNS by increasing the temperature of DHT11.

C. Introduction to Amazon SQS



Amazon SQS, or Amazon Simple Queue Service, is a fully managed message queuing service provided by Amazon Web Services (AWS). It enables distributed and decoupled applications to communicate by allowing one component to send messages while another component receives and processes them asynchronously. SQS simplifies the task of building highly scalable, fault-tolerant, and loosely coupled systems.

Common usage:

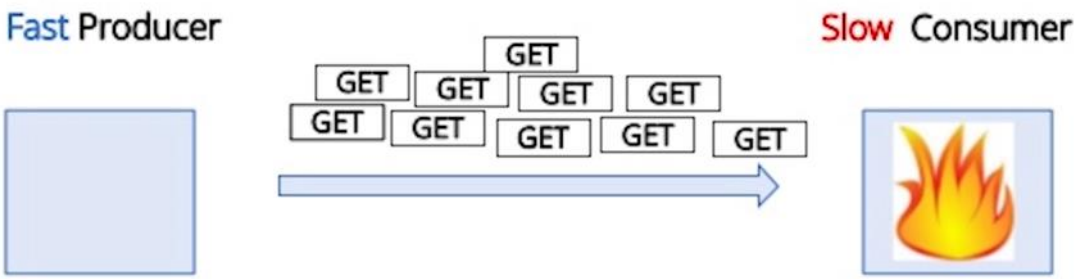
- Buffering: SQS acts as a buffer to absorb traffic spikes in the application and prevent it from being overwhelmed during surges in demand.

Copyright 2023 ByteWise Educator

- Distributed Systems: SQS is often used in building distributed and microservices architectures, enabling communication between various microservices and components.

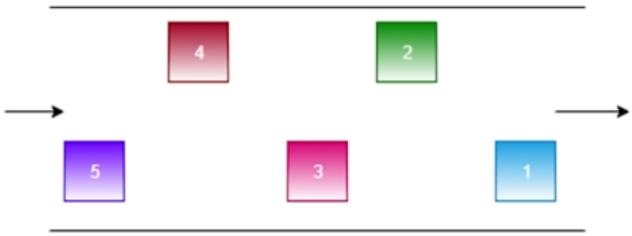
- Job Queues: SQS is utilized for managing task and job queues, where work items are processed by worker instances as they become available.

- Asynchronous Processing: SQS is useful for offloading tasks that don't need to be processed immediately and can be handled in the background.



Amazon SQS (Simple Queue Service) provides two main types of queues: Standard Queues and FIFO (First-In-First-Out) Queues. Each queue type is designed for specific use cases and has distinct characteristics:

Standard Queues:



- Unlimited Throughput: Standard queues offer high throughput and can process a large number of messages per second. They are designed for applications that require high scalability and can tolerate occasional out-of-order message delivery.
- Best-Effort Ordering: Messages in standard queues are delivered in approximately the order in which they were sent, but out-of-order delivery can occur. If strict message ordering is not critical for your application, standard queues are a good choice.
- At-Least-Once Delivery: Standard queues guarantee that a message is delivered at least once, but it's possible that a message can be delivered multiple times (duplicate messages).
- No Deduplication: Standard queues do not provide built-in message deduplication. If your application requires deduplication, you must implement it on the sender's side.

Copyright 2023 ByteWise Educator

FIFO (First-In-First-Out) Queues:

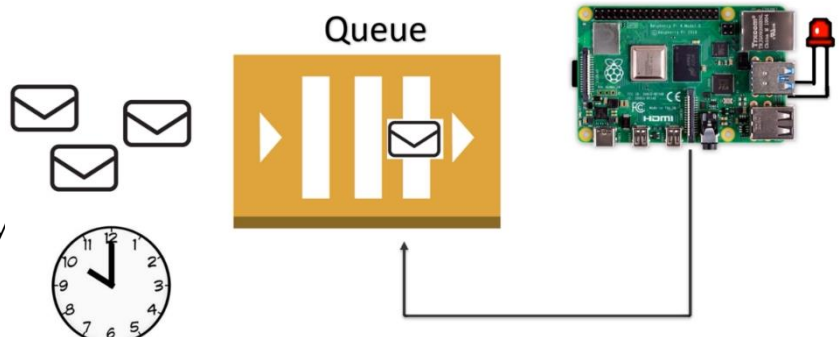
- Exactly-Once Processing: FIFO queues provide strict message ordering, ensuring that messages are delivered in the exact order they were sent. This is essential for applications that rely on the order of message processing.
- Deduplication: FIFO queues automatically remove duplicate messages based on a message deduplication ID. Each message must have a unique deduplication ID, which SQS uses to identify and discard duplicate messages.
- Lower Throughput: FIFO queues have a lower throughput compared to standard queues. They are suitable for applications that require exactly-once message processing and strict ordering, such as financial transactions and order processing.
- Limited to 300 transactions per second (TPS) for each API action: Each API action (e.g., sending, receiving, or deleting messages) is limited to 300 TPS per FIFO queue.

When to use each type of queue:

Use Standard Queues when you need high throughput and can tolerate occasional out-of-order message delivery, and strict ordering is not a critical requirement. This is suitable for most general-purpose messaging scenarios.

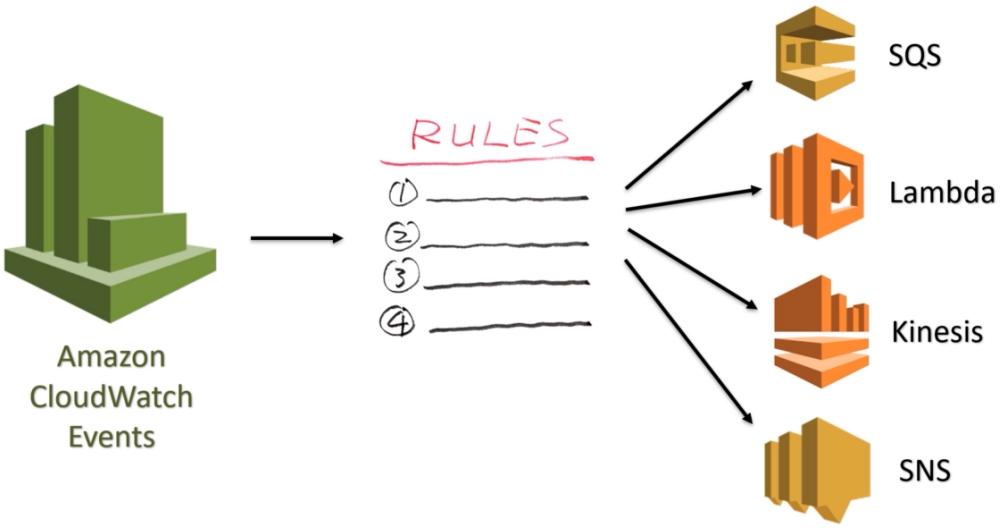
Use FIFO Queues when you require exactly-once message processing, strict message ordering, and deduplication of messages. This is important for applications where maintaining order and preventing duplicates are crucial, such as financial applications and order processing systems.

1. Now, we are to test the functionalities of SQS by making a simple project, which is to turn on the LED on certain time. For example, on 10 am every day, a message will be delivered to SQS and it will read by RPi. If there is a message in SQS, the LED will be turned on. After that, the message in the queue will be deleted.



Copyright 2023 By

2. Go to Amazon and search for SQS. Then, create queue.
3. Choose standard queue and give it a name. Click Create Queue.
4. If you want to check for new message, click on the queue that you have created.
5. Click on Send and receive message and then click Poll for messages. It will take a few seconds to check any message in the SQS.
6. Before that, let's take a look to Amazon CloudWatch Events.



Amazon CloudWatch Events is a service provided by Amazon Web Services (AWS) that allows you to monitor and respond to events in your AWS environment. CloudWatch Events provides a way to track and respond to changes or activities in your AWS resources, services, and applications in a near real-time, automated manner. This service can be used to automate various operational and security tasks, making it a powerful tool for managing and monitoring your AWS infrastructure.

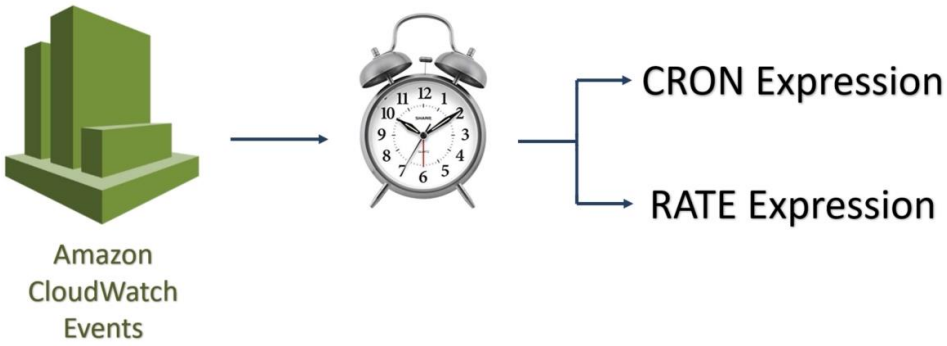
Key features and concepts related to Amazon CloudWatch Events include:

- Events and Event Sources: Events represent changes or activities within your AWS environment. Event sources can include AWS services (e.g., EC2 instances, S3 buckets, Lambda functions), as well as custom events generated by your applications. Two types of events: 1. Schedule event, 2. Application level event
- Rules: CloudWatch Events uses rules to define the conditions under which it should trigger actions. Rules specify event patterns that match events from your event sources.
- Targets: When a rule matches an event, it can trigger one or more actions, called "targets." Targets can include AWS Lambda functions, Step Functions, SNS topics, SQS queues, Kinesis streams, and more.

Copyright 2023 ByteWise Educator

These targets enable you to automate responses to events or in other word, a target processes the events.

- Event Bus: An event bus is a communication channel for events within CloudWatch Events. AWS services, like EC2 and Lambda, publish events to default event buses, and you can create custom event buses



Amazon CloudWatch Events can be used to schedule and trigger events at specific times using cron expressions and rate expression. A cron expression is a string that defines a schedule for when events should occur, specifying minute, hour, day of the month, month, and day of the week. Rate expressions are simpler to use compared to cron expressions and are well-suited for scheduling events at regular intervals.

Example cron format:

cron(Minutes Hours Day-of-month Month Day-of-week Year)

Max and min value for cron:

Field	Values	Wildcards
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-week	1-7 or SUN-SAT	, - * ? L #
Year	1970-2199	, - * /

Copyright 2023 ByteWise Educator

Examples:

- "cron(0 12 * * ? *)": This expression triggers the event every day at 12:00 PM (noon).
- "cron(0 8 ? * MON-FRI *)": This expression triggers the event every weekday (Monday to Friday) at 8:00 AM.
- "cron(0 0 ? * * *)": This expression triggers the event at midnight every day.
- "cron(0/15 * * * ? *)": This expression triggers the event every 15 minutes.

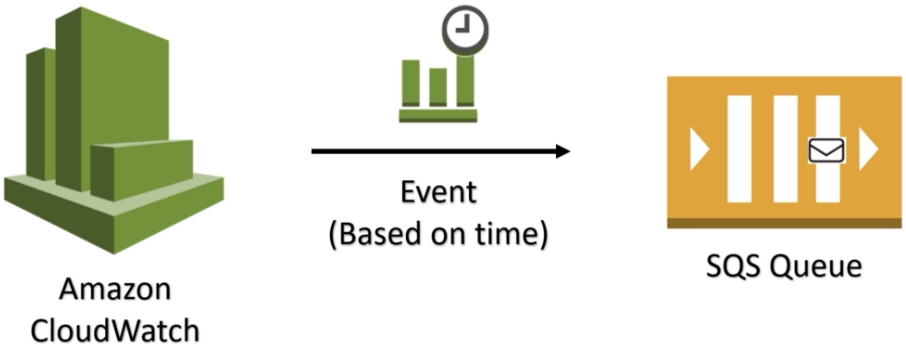
Note: You cannot use * in both day of the month and day of the week fields at the same time. One field need to use *, and another field need to use ?

Rate expression format:

rate(Value Unit)

Examples:

- rate(5 minutes): This triggers the event every 5 minutes.
- rate(1 hour): This triggers the event every 1 hour.
- rate(1 day): This triggers the event once a day.



7. Now, we are going to use CloudWatch Events to schedule a message to SQS on a certain time.
8. Go to Amazon and search for CloudWatch. Click on Events > Rules. You will brought to Amazon EventBridge. Click on Create Rule.
9. Give it a name and choose Schedule rule type. Click on Continue to create rule.
10. Put the cron expression, for example if you want to set a schedule to trigger event at 10 am local time.
    cron (0 2 * * ? *)
    Click next.

Copyright 2023 ByteWise Educator

11. Select the target types as AWS service and pick SQS queue as the target. Select the queue that you have created previously. Click on Skip review and create and then Create rule.
12. Now you can check whether is there any message in the SQS queue on 10 am everyday.

D. Scheduling a trigger

1. Before proceed to the next step, you need to create a user within your root account. It is called as IAM user.

An IAM (Identity and Access Management) user is a fundamental concept in the Amazon Web Services (AWS) cloud computing platform. IAM users are entities that you create within your AWS account to represent the individuals, applications, or services that need access to AWS resources. IAM allows you to control and manage access to your AWS services and resources securely.

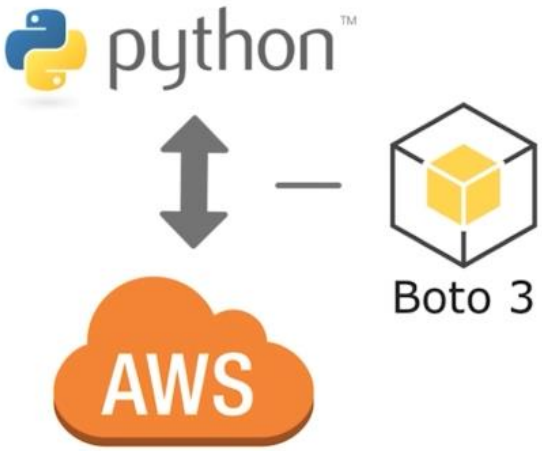
Here's why IAM users are required and their primary purposes:

- Security: IAM users help improve the security of your AWS account. Without IAM, you would have to share your AWS account credentials (username and password) with others, which is a security risk. By creating IAM users, you can assign unique access credentials to different individuals or components, ensuring that they can access only the resources and services they need.
- Access Control: IAM users enable you to specify fine-grained permissions and access control for each user. You can define who has access to specific AWS services, which actions they can perform, and on which resources. This helps you enforce the principle of least privilege, granting users the minimum level of access needed to perform their tasks.
- Auditability: IAM users help in tracking and auditing actions taken within your AWS account. Each IAM user has a unique identity, and their actions are logged, making it easier to trace and monitor changes, helping you with compliance and security analysis.
- Isolation: IAM users allow for the isolation of responsibilities and workloads. By creating separate users, you can segregate access and responsibilities. For example, you might have one IAM user for a database administrator, another for a developer, and another for a systems administrator.
- Resource Management: IAM users can be used to manage resources within your AWS account. For example, you can create users specifically for deploying and managing EC2 instances, users for managing S3 buckets, or users for configuring other AWS services.
- Programmatic Access: In addition to console access, IAM users allow programmatic access to AWS resources through APIs and SDKs. This is particularly useful for automating tasks, scripting, and integrating AWS services into applications.
- Temporary Credentials: IAM users can generate temporary security credentials for users and applications, making it easier to assume roles or grant temporary access as needed.

2. Go to Amazon and search for IAM. Go to User > Create user.
3. Give it a name, and then choose attach policy directly.

Copyright 2023 ByteWise Educator

4. Tick on AdministratorAccess and then click next. Finally, click Create user.
5. Click on the user that you have created and then click on Security credentials tab.
6. Click on Create access key, and choose Application running on an AWS compute service. Tick on the confirmation and click Next.
7. Click on create access key and can download the CSV file or you can copy the access key and secret access key and paste it to any software that you like.
8. Go back to PuTTY and access you RPi. Type this to install boto3 SDK (to connect direct to AWS SQS from Raspberry Pi)
   `git clone https://github.com/boto/boto3.git`
9. Type `cd boto3`
10. Next type `sudo python setup.py install.`
11. Type this to install RPi GPIO SDK (to use GPIO pin on raspberry pi)
    `git clone https://github.com/BPI-SINOVOIP/RPi.GPIO.git`
12. Type `cd RPi.GPIO`
13. Next type `sudo python setup.py install.`



Boto3 is the Amazon Web Services (AWS) SDK (Software Development Kit) for Python. It provides an easy-to-use and comprehensive way to interact with AWS services using Python. Boto3 allows developers to access and manage AWS resources and services, such as Amazon S3 (Simple Storage Service), EC2 (Elastic Compute Cloud), Lambda, DynamoDB, and many more, from their Python applications and scripts.

14. Create a new python file by typing
    `sudo nano scheduling.py`

15. Paste this code into the python file

```
import boto3
import os
import time
import RPi.GPIO as GPIO

# Set up GPIO
led_pin = 17  # Replace with the GPIO pin you're using
GPIO.setmode(GPIO.BCM)
```

Copyright 2023 ByteWise Educator

```
GPIO.setup(led_pin, GPIO.OUT)

access_key = "ACCESS_KEY"
access_secret = "ACCESS_SECRET"
region = "REGION"
queue_url = "QUEUE_URL"

def pop_message(client, url):
    response = client.receive_message(QueueUrl = url,
MaxNumberOfMessages = 10)

    # Last message posted becomes messages
    message = response['Messages'][0]['Body']
    receipt = response['Messages'][0]['ReceiptHandle']
    client.delete_message(QueueUrl = url, ReceiptHandle = receipt)
    return message

client = boto3.client('sqs', aws_access_key_id = access_key,
aws_secret_access_key = access_secret, region_name = region)

waittime = 20
client.set_queue_attributes(QueueUrl = queue_url, Attributes =
{'ReceiveMessageWaitTimeSeconds': str(waittime)})

time_start = time.time()
while (time.time() - time_start < 60):
    print("Checking...")
    try:
        message = pop_message(client, queue_url)
        print(message)
        if message:
            # Turn on the LED
            GPIO.output(led_pin, GPIO.HIGH)
            time.sleep(20)  # Keep the LED on for 20 seconds
            # Turn off the LED
            GPIO.output(led_pin, GPIO.LOW)
    except:
        pass

# Clean up GPIO
GPIO.cleanup()
```

16. In order to make sure the code is running automatically, we need to use crontab.

Crontab is a time-based job scheduler in Unix-like operating systems, including Raspberry Pi OS. It allows you to schedule tasks and commands to run at specific times or on a recurring basis. Crontab is a useful tool for automating repetitive tasks on your Raspberry Pi. To manage the crontab on a Raspberry Pi or any other Unix-like system, you can use the crontab command. Here's a basic overview of how to use it:

Viewing Your Crontab: `crontab -l`

Editing Your Crontab: `crontab -e`

Crontab syntax:

Copyright 2023 ByteWise Educator

The crontab file uses a specific syntax to define when tasks should run. It includes fields for minutes, hours, days, months, and days of the week, followed by the command to execute. For example, this is the structure of a crontab line:

```
* * * * * command-to-be-executed
```

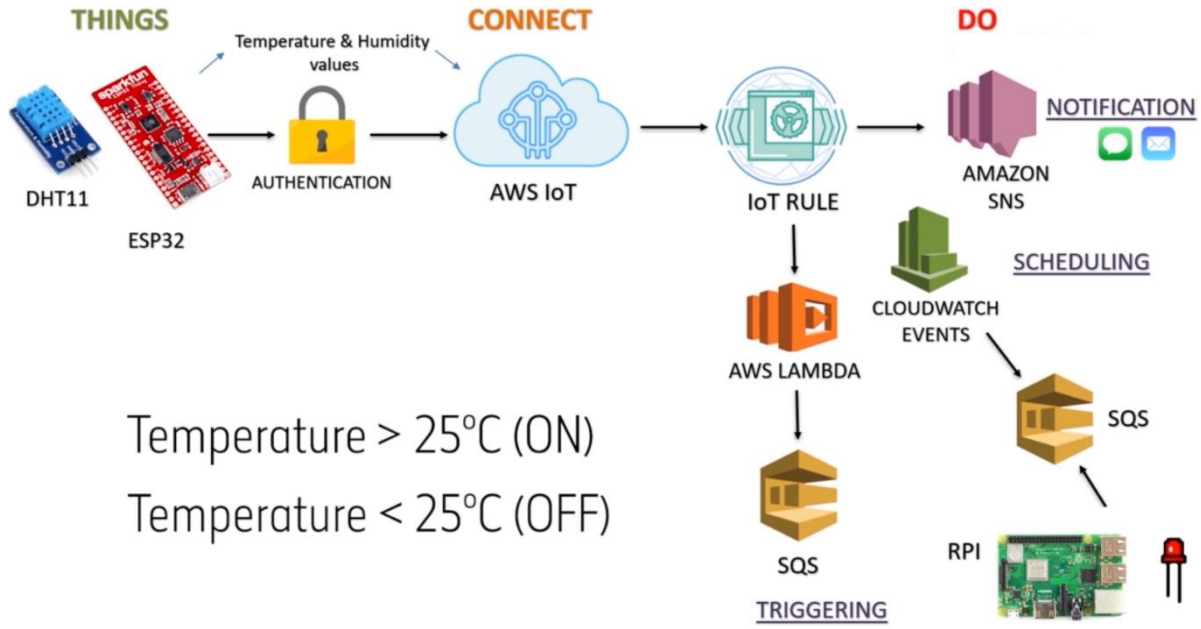
Example to run task every day at 3.30 PM:

```
30 15 * * * python home/pi/scheduling.py
```

Here are some key features and concepts associated with AWS Lambda:

17. In this case, we want to run it every day. So, type:
    ```
    * * * * * python home/pi/scheduling.py
    ```
18. Hit Ctrl X and then Y and enter.

Module 3: Triggering Raspberry Pi with AWS Cloud and ESP32



AWS Lambda is a serverless computing service offered by Amazon Web Services (AWS). It enables you to run code without provisioning or managing servers. Lambda is designed to execute code in response to various events, such as changes to data in an Amazon S3 bucket, updates to a database, HTTP requests via API Gateway, or other AWS services triggering events.

- Serverless: Lambda abstracts the underlying infrastructure, so you don't need to worry about provisioning, scaling, or managing servers. You only pay for the compute time your code consumes.
- Event-Driven: Lambda functions are triggered by events. These events can come from various AWS services, such as S3, DynamoDB, Kinesis, SQS, or custom events via API Gateway. When an event occurs, Lambda runs the associated code.

Copyright 2023 ByteWise Educator

- Language Support: AWS Lambda supports a variety of programming languages, including Node.js, Python, Java, Ruby, C#, and custom runtimes, allowing you to use other languages and execution environments.
- Scalability: Lambda automatically scales your application based on the incoming workload. If there are many events, it can create multiple instances of your function to handle the load, and if there are no events, it can scale down to zero, saving costs.
- Stateless: Lambda functions are typically designed to be stateless, meaning they don't maintain persistent connections or store data locally. Instead, you can use other AWS services like S3, DynamoDB, or RDS to store and retrieve data.

- Duration-Based Billing: AWS Lambda bills you based on the number of requests and the execution time of your code. You pay for the compute time used in 100ms increments.

AWS Lambda can be used in conjunction with AWS IoT Core to create powerful, event-driven IoT applications. AWS IoT Core is a managed service that enables you to connect IoT (Internet of Things) devices to the cloud and securely interact with them. By integrating Lambda with IoT Core, you can process and analyze data from IoT devices, automate actions, and trigger responses based on IoT events.

AWS lambda is service only be charged when the code is running. It can be ran from a few request per day or thousand requests per second. Pay is based on the compute time you consume, not will be charged if the code is running.

1. Go to Amazon and search for Lambda
2. Create a lambda function and choose Author from scratch
3. Give it a name for the function
4. Choose Python and click Create Function
5. Now, we want to link the IoT Rule to AWS lambda as the trigger.
6. Go to AWS IoT Core.
7. Message Routing > Rules > Create rule
8. Give a name for the rule
9. Choose the latest SQL version and type this in the SQL statement `SELECT * FROM '<Topic Filter>'` and then click Next
10. Select Lambda as rule action
11. Choose the lambda function that you created previously > Next > Create
12. Open AWS Lambda, and you will find the trigger will be assigned automatically to the IoT Rule that we already created.
13. Click on layers that on AWS Lambda and now we are going to put the python code into that function.

Copyright 2023 ByteWise Educator

```
import boto3

access_key = "xxxxxxxxxxxxxxxxxxxxxxx"
access_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxx"
region ="xx-xxxx-x"
queue_url = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"


def post_message(client, message_body, url):
        response = client.send_message(QueueUrl = url, MessageBody=
message_body)

def lambda_handler(event, context):
    client = boto3.client('sqs', aws_access_key_id = access_key,
aws_secret_access_key = access_secret, region_name = region)
    x = event['temperature']
    if x > 25:
        post_message(client, 'on', queue_url)
        message = "on"
    elif x <= 25:
        post_message(client, 'off', queue_url)
        message = "off"
```

14. Create a new queue in AWS SQS. Just use a standard queue.
15. Copy the URL for the queue and paste it the lambda function for queue URL. Change the access key, access secret and region based on your settings.
16. Now, it is your task to modify the RPi programming based on this requirement. The LED need to be turned on if the content of message in SQS is = ‘on’ and LED need to be off if the content of the message = ‘off’.

Copyright 2023 ByteWise Educator

Module 4: Visualization using Amazon QUICKSIGHT

A. Getting started with Amazon S3

1. Now, we are going to store the data that collected from DHT11 into a database. In this case, we are going to use Amazon S3. Although there are varieties of database of such as DynamoDB, RDS and TimeStream.

   Amazon DynamoDB:

   - Key-Value Store: DynamoDB is a highly scalable and fully managed NoSQL database that can be a good choice for storing sensor data. You can use the DHT11 sensor data as a key and store the sensor readings as values.
   - Auto-Scaling: DynamoDB can automatically scale based on your throughput requirements, making it suitable for applications with variable workloads.

   Amazon S3:

   - Object Storage: If you prefer storing data as files, you can use Amazon S3. Each reading from the DHT11 sensor can be stored as a separate object in an S3 bucket. You can use prefixes or object naming conventions to organize the data.

   Amazon RDS (Relational Database Service):

   - SQL Database: If you need to perform complex queries or store additional metadata alongside sensor readings, you can use Amazon RDS to set up a relational database (e.g., MySQL, PostgreSQL, or others). This is useful if you need to perform data analysis or run SQL queries on the data.

   Amazon Timestream:

   - Time-Series Database: Amazon Timestream is a purpose-built time-series database service. It's designed for handling time-stamped data, making it a great choice for sensor data that changes over time.

2. In this module, we are going to visualize the sensor data in the Amazon QuickSight. Here is the comparison on which DB is better for this purpose:

   Amazon S3:

   - Suitability for Raw Data: Amazon S3 is an excellent choice for storing raw sensor data, especially if you have large volumes of data. It's cost-effective and designed for object storage.
   - Historical Data Storage: S3 is well-suited for long-term data storage and archival purposes. It provides cost-effective storage for historical data over extended periods.
   - QuickSight Integration: Amazon QuickSight can directly integrate with Amazon S3 as a data source, making it easy to create visualizations and dashboards.

Copyright 2023 ByteWise Educator

Amazon Timestream:

- Time-Series Data: Timestream is purpose-built for time-series data, which aligns perfectly with your DHT11 sensor readings.
- Data Retention Policies: You can configure data retention policies to manage historical data. Timestream is optimized for efficiently storing and querying time-series data over time.
- QuickSight Integration: QuickSight can integrate with Timestream, allowing you to create visualizations directly from Timestream data.

Amazon RDS:

- SQL Database: RDS is a suitable choice if you have structured data or if you anticipate needing to perform complex queries and analytics on your sensor data.
- Data Transformation: RDS allows you to perform data transformations and complex SQL operations on your data, making it a good option for data preparation before visualization.
- QuickSight Integration: While QuickSight can work with RDS, it may require additional data transformation steps to align the data with your visualization needs.

Amazon DynamoDB:

- Structured Data: DynamoDB is more suitable for structured data or if you have additional metadata associated with sensor readings. It's a NoSQL database.
- Real-time Analytics: DynamoDB excels at real-time data access and analytics, making it a good choice if you need to perform immediate analysis on your data before visualizing it.
- QuickSight Integration: While QuickSight can be used with DynamoDB, it may require additional data transformation steps to get the data in the desired format for visualization.

Considering to specific requirement to store raw sensor data for historical purposes and visualize it later in Amazon QuickSight, Amazon S3 is often the most cost-effective and straightforward choice. You can store large volumes of raw data in S3 and use QuickSight to build visualizations as needed.

However, if you anticipate the need for complex real-time analytics or if you have structured data beyond the sensor readings, Amazon Timestream or Amazon DynamoDB may be more suitable. In that case, the choice between the two would depend on the nature of your data and your analytics requirements.

Storing data in Amazon S3:

Data in Amazon S3 is organized into two primary concepts: objects and buckets.

- Bucket: A bucket is a top-level container in Amazon S3. It is a logical storage unit that holds objects. Buckets have a globally unique name within the Amazon S3 service, and they can be used to group and organize objects. Each AWS account can have multiple buckets. For example, you might create separate

Copyright 2023 ByteWise Educator

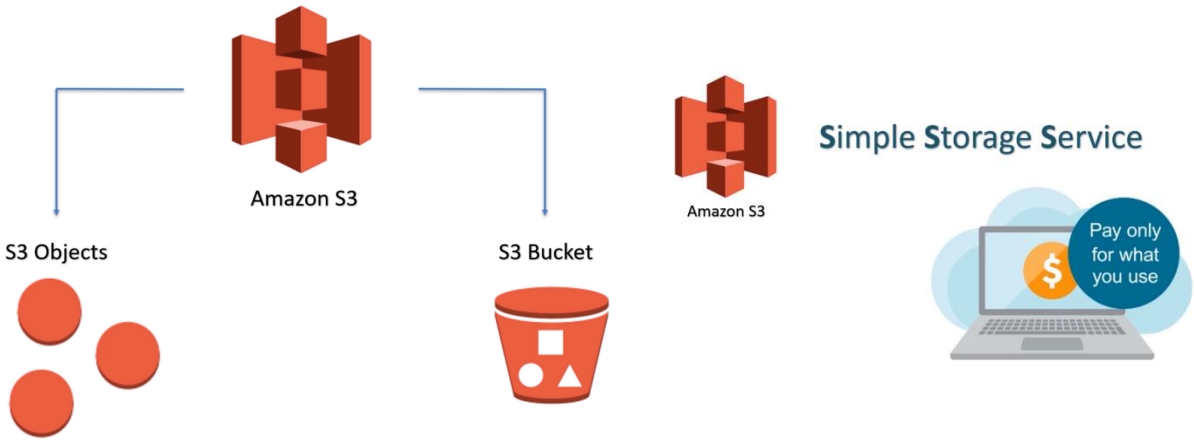
buckets to store different types of data or to isolate data for different applications.

- Object: An object is the basic unit of storage in Amazon S3. Each object consists of data, a key (a unique identifier within a bucket), and metadata. The data can be anything, such as a file, image, document, or any other binary data. The key is used to locate and retrieve the object from the bucket. Objects are organized within buckets and are typically stored in a flat namespace, but they can be organized using prefixes (simulating a directory structure) in the object key.

Here's a basic example:

Let's say you have an S3 bucket named "my-iot-data-bucket." Within this bucket, you can store objects, each representing a piece of data from your IoT devices. Each object has a unique key, such as "sensor-data/device-001/2023-10-15.json," where "sensor-data" is a prefix, "device-001" is the device identifier, and "2023-10-15.json" is the name of the data file. In this example, the bucket is "my-iot-data-bucket," and the objects are the individual data files stored within that bucket.

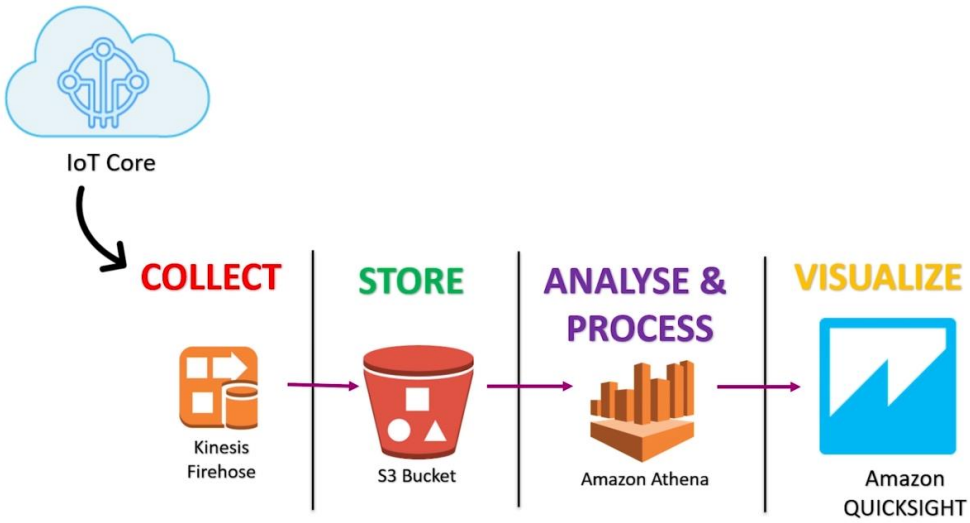
You can use the combination of buckets and objects to organize and manage your data in Amazon S3 efficiently. Additionally, Amazon S3 provides various features and settings for data management, access control, versioning, lifecycle policies, and more to meet your specific storage and data management needs.



3. Go to Amazon and search for S3 and click Create bucket.
4. Give it a name and click Create bucket.
5. But, we will use another AWS service before storing the data to S3. We are going to use Kinesis Firehose. Using Kinesis Firehose in your IoT data pipeline allows you to handle data streams efficiently, with minimal operational overhead. It ensures that your data is reliably delivered to Amazon S3, where you can further process and analyze it as needed.

Copyright 2023 ByteWise Educator

B. Introduction to Amazon Kinesis Firehose



1. Here are the reasons why you need to use Kinesis Firehose before storing the data into S3:

Aspect	IoT Core to S3 Direct	IoT Core to S3 via Kinesis Firehose
Data Transformation	Limited	Extensive (Transform, compress, encrypt)
Buffering	Limited (Immediate)	Support for buffering and batching data
Load Balancing	Single Destination	Multiple Destinations and Load Balancing
Error Handling	Custom Implementation	Automatic retries and error handling
Monitoring and Logging	Limited	Rich Monitoring and Logging
Scalability	Limited	Scalable for varying data volumes
Data Formats Supported	Any	Supports various data formats
Integration with Other AWS Services	Manual Configuration	Easy integration with other AWS services
Data Delivery Reliability	Depends on Implementation	High reliability with Firehose
Data Delivery Efficiency	Direct delivery, less optimization	Data can be optimized for efficiency
Cost Efficiency	May result in higher data storage costs	Potentially more cost-effective data storage

Sending data directly to Amazon S3 from AWS IoT Core may result in higher data storage costs compared to using Kinesis Firehose, which can convert data into columnar formats like Apache Parquet.

Data Storage Format:

- Direct to S3: When data is sent directly to S3, it is typically stored in its raw format, which may be less space-efficient. For example, data may be in JSON or another text-based format that consumes more storage space compared to columnar formats.

Copyright 2023 ByteWise Educator

- Kinesis Firehose: Kinesis Firehose allows you to perform data transformation and conversion into columnar formats like Apache Parquet. Columnar storage formats are highly space-efficient and can significantly reduce data storage costs.

Storage Costs:

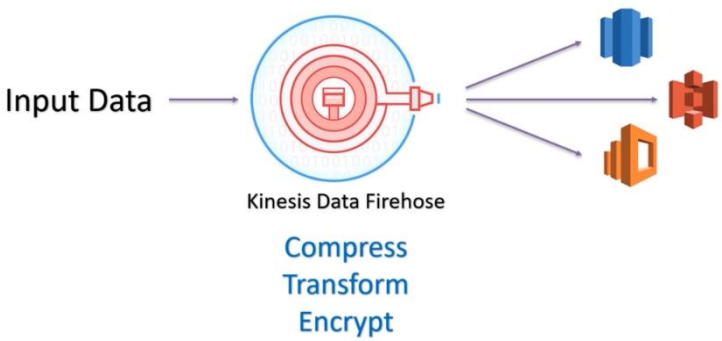
- Direct to S3: Storing data in its raw format in S3 may lead to higher storage costs, especially if you are dealing with large volumes of data over time. S3 charges are based on the amount of data stored, and less space-efficient formats can result in higher storage costs.
- Kinesis Firehose: When using Kinesis Firehose to convert data into columnar formats, the data takes up less space, resulting in cost savings in terms of data storage costs.

Data Compression:

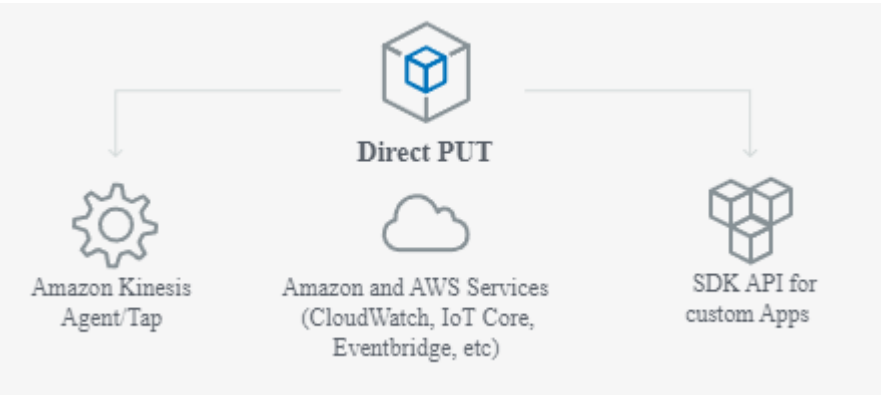
- Direct to S3: Without data transformation, direct data storage in S3 may lack efficient data compression, which can further impact storage costs.
- Kinesis Firehose: Kinesis Firehose can perform data compression as part of the data transformation process, reducing the storage requirements.

Query Performance:

- Kinesis Firehose: When data is converted into columnar formats like Apache Parquet, it not only saves storage space but also improves query performance. These formats are optimized for analytical queries, enabling quicker data retrieval and analysis in services like Amazon Athena or Amazon Redshift Spectrum.

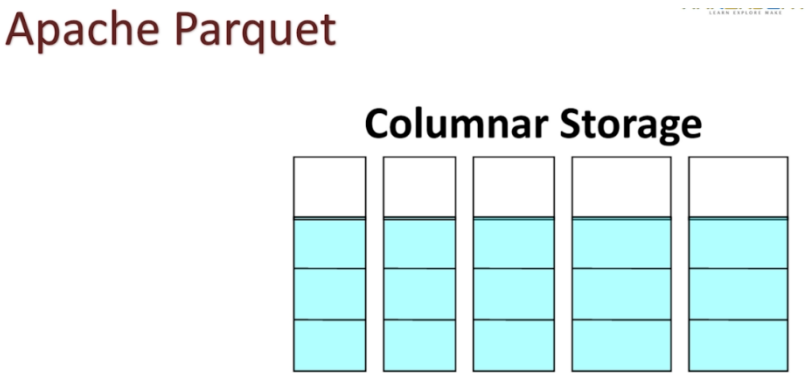


2. Go to Amazon and search for Kinesis Data Firehose and click Create delivery system.
3. Pick Direct PUT (since our data from IoT Core) as the source and Amazon S3 as the destinations.



Copyright 2023

4. Tick on Enable record format conversion and make sure the output format is in Apache Parquet.



Apache Parquet is an open-source columnar storage format that is designed for efficient and optimized storage and processing of large datasets.

Key features and characteristics of Apache Parquet include:

- Columnar Storage: Parquet stores data in a columnar format, which means that instead of storing rows of data, it stores columns separately. This columnar storage approach is highly efficient for analytical queries because it allows the system to read only the specific columns needed, reducing I/O and improving query performance.
- Compression: Parquet employs various compression techniques to reduce the storage size of data. Columnar storage is inherently more compressible than row-based storage because similar data values are stored together, enabling more effective compression.
- Splittable and Seekable: Parquet files can be split into smaller, more manageable parts that can be read and processed independently. This makes Parquet files suitable for parallel and distributed processing.
- Schema Evolution: Parquet supports schema evolution, allowing you to evolve and adapt your data schema over time without the need for expensive data transformations or rewrites. New columns or fields can be added without affecting existing data.
- Predominantly Binary Data: Parquet is optimized for storing binary data, such as numbers, strings, and binary objects, making it well-suited for a wide range of data types.
- Cross-Language Compatibility: Parquet is designed to be used across different programming languages and platforms. It has libraries and APIs for popular languages, including Java, C++, Python, and more.

5. In order to use the columnar format, we need to use AWS Glue.
AWS Glue is a fully managed ETL (Extract, Transform, Load) service provided by Amazon Web Services (AWS). It is designed to help users discover, catalog, transform, and load data from various sources into a wide range of data stores and data warehouses. AWS Glue simplifies and automates the process of preparing and transforming data for analysis, making it more suitable for analytics, reporting, and machine learning tasks.

Copyright 2023 ByteWise Educator

- Data Transformation: AWS Glue provides a serverless and scalable environment for transforming data. You can define ETL jobs in AWS Glue to transform your data from one format (e.g., JSON or CSV) into Apache Parquet. This transformation can involve tasks like data cleansing, filtering, aggregating, and schema conversion.
- Integration with Kinesis Firehose: AWS Glue integrates seamlessly with Amazon Kinesis Firehose. You can set up a Firehose delivery stream to deliver data to an AWS Glue DataBrew job or AWS Glue ETL job, allowing you to perform data transformations on the data before storing it in Parquet format.
- Data Catalog and Metadata Management: AWS Glue includes a data catalog that helps you organize and catalog your data, making it easier to discover, search, and understand data assets. This catalog can be used in conjunction with other AWS services, making data discovery and analysis more efficient.
- Schema Evolution: AWS Glue can help you handle schema evolution, which is essential when converting data. As data formats change or evolve over time, AWS Glue can assist in adapting to these changes while ensuring data quality and consistency.

6. Choose the correct region for the Glue region, and then click Create table.
7. Click on Databases > Add database.
8. Give it a name and click Create Database.
9. Click on Tables > Create table.
10. Give it a name and choose the database that you created previously.
11. Include the path for S3 bucket by clicking on Browse S3. Before that, make sure that you create a folder within the bucket so that it can act as the prefix.
12. Choose Parquet as the data format, and click Next.
13. Make sure to choose Define or upload schema and then click Add. This Add button is to add a new column. Add this column:

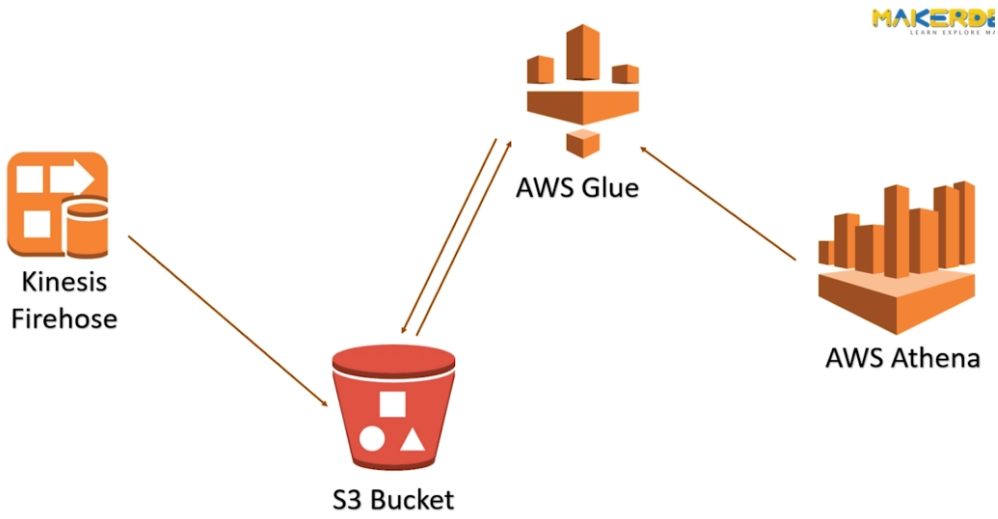
Name	Data type
Date	String
Time	String
Temperature	Float
Humidity	float

14. Click Next and then click Create.
15. Go back to AWS Glue and choose the database and table that you have created previously.
16. As for the destination settings, click Browse to browse the bucket for your S3 (choose the bucket name without prefix).
17. In S3 bucket prefix, put the prefix name ending with ‘/’ (example: dht11reading/)
18. In S3 bucket error output prefix, you may put ‘error/’.
19. In Buffer hints, compression and encryption, choose 65MiB as the buffer size and 60 seconds as buffer interval.
20. Click Create delivery stream.
21. Go to IoT Core, and then you need to add one more rule action in the rule.
22. Click Edit on the existing role and then click Add rule action.
23. Choose Kinesis Firehose stream and select the stream that you have created.
24. Make the separator as ‘\n’ and create a new role for the action.
25. Then, click Update.
26. Finally, test it out and check the data in the S3.

Copyright 2023 ByteWise Educator

D. Learning Athena and Getting Started with Amazon QUICKSIGHT

1. Before you can visualize the data, we need to use another service from Amazon, which is called as Athena.



Amazon Athena is a serverless, interactive query service provided by Amazon Web Services (AWS) for analyzing and querying data stored in Amazon S3 (Simple Storage Service). It allows you to run SQL queries on data stored in your S3 buckets without the need to set up and manage complex infrastructure or databases. Athena is part of the AWS analytics services and is particularly well-suited for ad-hoc and exploratory data analysis.

Key features and aspects of Amazon Athena include:

- Serverless: With Amazon Athena, you don't need to provision or manage any servers or databases. It's a fully serverless service, meaning AWS handles all the underlying infrastructure for you. You only pay for the queries you run and the amount of data scanned, which makes it cost-effective.
- Standard SQL Queries: Athena uses SQL (Structured Query Language), which is a widely known and adopted query language. This allows data analysts and data scientists to write SQL queries to analyze data in S3, making it accessible to a broad range of users.
- Integration with Amazon S3: Athena seamlessly integrates with data stored in Amazon S3. You can query data in various formats, including Parquet, ORC, JSON, CSV, and more, directly from your S3 buckets.
- Schema on Read: Athena follows a schema-on-read approach, which means that you can query your data without the need to define a rigid schema upfront. This flexibility is beneficial when working with semi-structured or unstructured data.
- Performance: Athena is optimized for ad-hoc querying and can efficiently scan and process large datasets stored in S3. It also supports features like partitioning and data compression for improved query performance.

Copyright 2023 ByteWise Educator

- Integration with AWS Glue: You can use AWS Glue, another AWS service, to catalog and discover metadata about your data stored in S3. This metadata can be used by Athena to improve query performance and facilitate data exploration.
- Security: Athena is integrated with AWS Identity and Access Management (IAM), which allows you to control access to your data and query results. It also supports encryption of data at rest and in transit.
- Visualization and Integration: You can connect Athena to various business intelligence (BI) and data visualization tools like Amazon QuickSight, Tableau, and more to create interactive dashboards and reports.

Amazon Athena is a valuable tool for organizations that want to make sense of their data stored in Amazon S3 without the need for complex data warehouses or data processing infrastructure. It simplifies the process of querying and analyzing data in a cost-effective and scalable manner.

2. Go to Amazon, and search for Athena. Click on Query your data with Trino SQ and click Launch query editor.
3. Make sure you choose the correct Database and Table.
4. In the query, type `select * from <table name>` and then click Run.
5. You can see the data in columnar format.
6. Now, you can visualize this columnar data in Quicksight. Go to Amazon and search for Quicksight.

Amazon QuickSight is a fully managed business intelligence (BI) and data visualization service provided by Amazon Web Services (AWS). It is designed to help you easily and quickly analyze data, create interactive dashboards, and generate meaningful insights from your data. QuickSight is a serverless service, which means you don't need to manage infrastructure, and it can be seamlessly integrated with various data sources, including Amazon Athena.

Key features of Amazon QuickSight include:

- Data Visualization: QuickSight allows you to create interactive and visually appealing data visualizations, such as charts, graphs, and dashboards, with just a few clicks. You can choose from a wide range of visualization options to suit your data and analytical needs.
- Data Integration: QuickSight can connect to various data sources, including databases, data lakes, and other AWS services like Amazon Redshift, Amazon RDS, Amazon Aurora, Amazon Athena, and more. This makes it easy to access and analyze data from different sources in one place.
- SPICE Engine: QuickSight features the Super-fast, Parallel, In-memory Calculation Engine (SPICE) for caching and optimizing data to improve query performance. This allows for faster and more interactive data exploration.

Copyright 2023 ByteWise Educator

- Data Transformation: You can perform data transformations within QuickSight, including data preparation, cleaning, and shaping, without the need for external ETL tools.
- Custom Calculations: QuickSight supports custom calculations and calculated fields, enabling you to derive insights from your data using formulas.
- Sharing and Collaboration: You can share your dashboards and reports with colleagues, teams, or clients, and they can explore the data interactively. QuickSight also supports collaboration features.
- Pay-per-Session Pricing: QuickSight offers a unique pricing model known as pay-per-session. This means you only pay when users access and interact with QuickSight dashboards, making it cost-effective.

7. Create a new account for Quicksight.
8. Scroll down and find sign up for standard account
9. Fill up the information and make sure you tick on amazon S3 and Athena.
10. Click New Analysis and click New Dataset.
11. Give it a name and validate the data source. Click Create data source.
12. Select the database and table.
13. Click on Edit / Preview Data and you will see all the data
14. Click on Publish and Visualize
15. Choose any visual type that you like and drag the data into the y axis and x axis.

Copyright 2023 ByteWise Educator