

Universidad Rafael Landívar
Ing. Electrónica y Telecom.

Manual Técnico

Billy Arturo López Vicente 1008018
Carlos Andrés González Donis 1129519

Guatemala, 20 de noviembre de 2023

Manual Técnico

El manual técnico tiene como objetivo principal proporcionar a los usuarios una guía completa y detallada para comprender, implementar y mantener el sistema propuesto, que combina la tecnología IoT con Arduino, DynamoDB, funciones Lambda, comunicación por MQTT, utilización de AWS Analytics y QuickSight. Este manual está diseñado para un público técnico, como ingenieros en electrónica y telecomunicaciones, desarrolladores de software y profesionales interesados en la integración de dispositivos IoT en un entorno práctico.

Objetivos Específicos:

1. **Entender el Proyecto:** Proporcionar una descripción clara y concisa del proyecto global, incluyendo su propósito y cómo los componentes clave interactúan entre sí.
2. **Facilitar la Configuración del Entorno:** Detallar los requisitos previos necesarios, incluyendo herramientas y software, y proporcionar instrucciones paso a paso para configurar el entorno de desarrollo.
3. **Comprender el Código de Arduino:** Desglosar el código de Arduino para que los usuarios puedan comprender su estructura, funciones clave y su relación con el hardware.
4. **Implementar una Base de Datos en DynamoDB:** Guiar a los usuarios en la creación de una base de datos en DynamoDB, con explicaciones detalladas sobre el diseño de la tabla y cómo integrarla con Arduino.
5. **Desarrollar Funciones Lambda:** Detallar la creación de funciones Lambda, incluyendo su lógica y configuración de triggers, para permitir la comunicación efectiva entre el hardware y la base de datos.
6. **Utilización de AWS Analytics:** Guiar a los usuarios en el uso de AWS Analytics con explicaciones detalladas sobre la creación de un Storage S3, AWS Athena, QuickSight y la integración con una base de datos almacenada en DynamoDB.
7. **Integrar Componentes:** Explicar cómo se integran todos los componentes del sistema, desde Arduino hasta DynamoDB, y proporcionar ejemplos prácticos de su funcionamiento en conjunto.
8. **Proporcionar Soluciones a Problemas Comunes:** Ofrecer una sección dedicada a la resolución de problemas, enumerando posibles desafíos y proporcionando soluciones para abordarlos.
9. **Facilitar el Mantenimiento y las Actualizaciones:** Proporcionar pautas sobre cómo mantener y actualizar el sistema en el futuro, incluyendo buenas prácticas para el mantenimiento del código y el hardware.
10. **Recopilar un Historial de Datos para Análisis:** Destacar la importancia de mantener un historial de datos recopilados y brindar orientación sobre cómo almacenar y gestionar estos datos para análisis posterior.
10. **Ofrecer una Documentación Completa:** Incluir recursos visuales, como diagramas y capturas de pantalla, junto con el código fuente completo, para una comprensión más profunda y una fácil implementación.

Este manual técnico se concibe como una herramienta esencial para que los usuarios logren una implementación exitosa del sistema IoT propuesto, permitiéndoles maximizar el rendimiento, la eficiencia y la vida útil de las soluciones desarrolladas.

Diagrama de la arquitectura



Herramientas y Software

-Herramientas utilizadas:

- Arduino UNO
- Módulo WiFi ESP8266 ESP01
- Sensor de humedad y temperatura DHT11
- 2 paneles solares
- Un motor de 6v
- Módulo de configuración de ESP01
- Cables para conexión con Arduino (Jumpers)

-Software:

- Cuenta de AWS
- Arduino IDE 2.2.1

Configuración de entorno:

Abrir el IDE de Arduino a utilizar, crear un proyecto e instalar librerías. Se deberá importar al proyecto las siguientes librerías:

- ArduinoJson
- PubSubClient
- Adafruit AWRprog
- DHT sensor library

Se deberá importar al proyecto las siguientes tarjetas:

- Esp8266

Arduino

El microcontrolador Arduino UNO se utilizará en la creación de una estación de monitoreo del clima que también generará energía renovable, lo cual se podrá realizar utilizando el sensor DHT11 para recibir datos de la temperatura en °C del ambiente, así también el porcentaje de humedad del

entorno. También se monitoreará el voltaje o energía producida por una turbina eólica y dos paneles solares conectados al sistema.

Los datos recibidos por el Arduino se enviarán a la nube por medio de MQTT utilizando el módulo WiFi ESP8266 ESP01 para su posterior almacenaje y análisis.

Estructura del código:

-Código de Arduino UNO que recibe los datos de los sensores y los convierte en archivos JSON para que por medio del módulo ESP01 puedan ser enviados a una base de datos en la nube:

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>
#include <DHT.h> // Agrega la biblioteca del sensor DHT11

// Configuración del sensor DHT11
#define DHTPIN 4 // El pin al que está conectado el sensor DHT11 (cambia esto según tu conexión)
#define DHTTYPE DHT11 // Tipo de sensor DHT (DHT11 o DHT22)

DHT dht(DHTPIN, DHTTYPE);

// Configuración de comunicación WiFi
SoftwareSerial wifi(2, 3); // Configura los pines RX y TX para la comunicación con el módulo WiFi

// Configuración de depuración
#define DEBUG true

String sendDataToWifiBoard(String command, const int timeout, boolean debug);
String prepareDataForWifi(float temperatura, float humedad, float voltajet, float voltajep);
/**
 * Build and return a JSON document from the sensor data
 * @param temperatura
 * @param humedad
 * @param voltajet Valor de voltaje de la turbina
 * @param voltajep Valor de voltaje de los paneles
 * @return
 */
String prepareDataForWifi(float temperatura, float humedad, float voltajet, float voltajep) {
    StaticJsonDocument<200> doc;

    doc["temperatura"] = temperatura;
    doc["humedad"] = humedad;
    doc["voltajet"] = voltajet;
    doc["voltajep"] = voltajep;

    char jsonBuffer[200];
    serializeJson(doc, jsonBuffer);
```

```

    return jsonBuffer;
}

/**
 * Send data through Serial to ESP8266 module
 * @param command
 * @param timeout
 * @param debug
 * @return
 */

String sendDataToWifiBoard(String command, const int timeout, boolean debug) {
    String response = "";

    wifi.print(command);

    unsigned long time = millis();

    while ((time + timeout) > millis()) {
        while (wifi.available()) {
            char c = wifi.read();
            response += c;
        }
    }

    if (debug) {
        Serial.print(response);
    }

    return response;
}

void setup() {
    wifi.begin(9600); // Inicializa la comunicación con el módulo WiFi
    Serial.begin(9600); // Inicializa la comunicación serial
    unsigned long stabilizingtime = 2000; // Tiempo de estabilización
    dht.begin(); // Inicializa el sensor DHT11
}

void loop() {
    if (DEBUG == true) {
        if (wifi.available()) {
            String espBuf;
            unsigned long time = millis();

            while ((time + 1000) > millis()) {
                while (wifi.available()) {
                    char c = wifi.read();

```

```

        espBuf += c;
    }
}
    Serial.print(espBuf);
}
}
    bool test = true;
    if (DEBUG == true) {

        float temperatura = dht.readTemperature(); // Lee la temperatura del sensor
DHT11
        float humedad = dht.readHumidity(); // Lee la humedad del sensor DHT11
        float voltajet = round(analogRead(1) * (5.4 / 1023.0) * 100) / 100.0; //
Redondea a dos decimales el voltaje recibido de la turbina
        float voltajep = round(analogRead(0) * (5.4 / 1023.0) * 100) / 100.0; //
Redondea a dos decimales el voltaje recibido de los paneles solares

        String preparedData = prepareDataForWifi(temperatura, humedad, voltajet,
voltajep);
        sendDataToWifiBoard(preparedData, 4000, DEBUG);
        Serial.println(preparedData);
    }

    delay(500); // Espera medio segundo antes de volver a leer el sensor
}

```

-Código de módulo ESP01 que envía los archivos JSON recibidos de Arduino para comunicación por MQTT a la nube de AWS:

```

#include <time.h>
#include "secrets.h"
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ArduinoJson.h> //https://github.com/bblanchon/ArduinoJson (use v6.xx) //
Librería para trabajar con JSON
#include <PubSubClient.h>
#include <SoftwareSerial.h>
#include <WiFiClientSecure.h>

#define DEBUG true // Habilita o deshabilita la depuración

const int MQTT_PORT = 8883;
const char MQTT_PUB_TOPIC[] = "esp8266/pub"; // Tema MQTT para publicación

uint8_t DST = 0; // Horario de verano (Daylight Saving Time) deshabilitado
WiFiClientSecure net; // Cliente WiFi seguro

// Configuración de comunicación Serial entre el ESP8266 y Arduino Uno
SoftwareSerial UnoBoard(10, 11);
// Certificados y claves para la comunicación segura

```

```

BearSSL::X509List cert(cacert);
BearSSL::X509List client_cert(client_cert);
BearSSL::PrivateKey key(privkey);

PubSubClient client(net); // Cliente MQTT

unsigned long lastMillis = 0;
time_t now;
time_t nowish = 1510592825; // Tiempo de referencia para la configuración de hora

// Declaración de funciones
void connectToMqtt();
void NTPConnect(void);
void sendDataToAWS(void);
void checkWiFiThenMQTT(void);
void connectToWiFi(String init_str);
void messageReceived(char *topic, byte *payload, unsigned int length);
String sendDataToUno(String command, const int timeout, boolean debug);

void NTPConnect(void)
{
    Serial.print("Setting time using SNTP");
    sendDataToUno("Setting time using SNTP\r\n", 1000, DEBUG);

    // Configurar la hora usando SNTP (Simple Network Time Protocol)
    configTime(TIME_ZONE * 3600, DST * 3600, "pool.ntp.org", "time.nist.gov");
    now = time(nullptr);

    while (now < nowish) {
        delay(500);
        Serial.print(".");
        now = time(nullptr);
    }

    Serial.println(" done!");
    sendDataToUno(" done!\r\n", 1000, DEBUG);

    struct tm timeinfo;
    gmtime_r(&now, &timeinfo);

    Serial.print("Current time: ");
    Serial.print(asctime(&timeinfo));
}

void messageReceived(char *topic, byte *payload, unsigned int length)
{
    Serial.print("Received [");
    Serial.print(topic);
    Serial.print("]: ");
}

```

```

    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }

    Serial.println();
}

void connectToMqtt()
{
    Serial.print("MQTT connecting ");
    sendDataToUno("MQTT connecting \r\n", 1000, DEBUG);

    while (!client.connected()) {
        // Intentar conectarse al servidor MQTT
        if (client.connect(THINGNAME)) {
            Serial.println("connected!");
            sendDataToUno("connected! \r\n", 1000, DEBUG);
        } else {
            Serial.print("failed, reason -> ");
            Serial.println(client.state());
            Serial.println(" < try again in 5 seconds");
            delay(5000);
        }
    }
}

void connectToWiFi(String init_str)
{
    Serial.print(init_str);
    // Configurar nombre de host y conectarse a la red WiFi
    WiFi.hostname(THINGNAME);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, pass);

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

    Serial.println(" ok!");
}

void checkWiFiThenMQTT(void)
{
    connectToWiFi("Checking WiFi");
    sendDataToUno("Checking WiFi \r\n", 1000, DEBUG);

    connectToMqtt();
}

```



```

}

void sendDataToAWS(void)
{
    StaticJsonDocument<200> doc;

    // read data coming from Uno board and put into variable "doc"
    DeserializationError error = deserializeJson(doc, Serial.readString());

    // Test if parsing succeeds.
    if (error) {
        Serial.print("deserializeJson() failed.");
        return;
    }

    // parsing succeeded, continue and set time
    doc["time"] = String(millis());

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

    Serial.printf("Sending [%s]: ", MQTT_PUB_TOPIC);
    if (!client.publish(MQTT_PUB_TOPIC, jsonBuffer, false)) {
        Serial.println(client.state());
    }
}

String sendDataToUno(String command, const int timeout, boolean debug)
{
    String response = "";
    UnoBoard.print(command); // send the read character to the Uno
    long int time = millis();

    while( (time+timeout) > millis()) {
        while(UnoBoard.available()) {
            // The esp has data so display its output to the serial window
            char c = UnoBoard.read(); // read the next character.
            response+=c;
        }
    }

    if (debug) {
        Serial.print(response);
    }

    return response;
}

void setup()
{

```

```

Serial.begin(9600);
Serial.println("starting setup");

UnoBoard.begin(9600); // your esp's baud rate might be different
delay(2000);

connectToWiFi(String("Attempting to connect to SSID: ") + String(ssid));

NTPConnect();

net.setTrustAnchors(&cert);
net.setClientRSACert(&client_crt, &key);

client.setServer(MQTT_HOST, MQTT_PORT);
client.setCallback(messageReceived);

connectToMqtt();
Serial.println("end setup");
}

void loop()
{
  now = time(nullptr);
  if (!client.connected()) {
    checkWiFiThenMQTT();
  } else {
    client.loop();
    if (millis() - lastMillis > 5000) {
      lastMillis = millis();
      sendDataToAWS();
    }
  }
}
}

```

-Archivo que contiene los certificados de autenticación para la conexión de la nube de AWS, y el enunciado por el cual se comunicará a MQTT y la red a la cual conectarse:

```

#include <avr/pgmspace.h>

#define SECRET

const char ssid[] = "_____"; // Reemplaza con tu SSID
const char pass[] = "_____"; // Reemplaza con tu contraseña WiFi

#define THINGNAME "ESP8266"

int8_t TIME_ZONE = -5;

const char MQTT_HOST[] = " ";

static const char cacert[] PROGMEM = R"EOF(

```

```

-----BEGIN CERTIFICATE-----

-----END CERTIFICATE-----
)EOF";
static const char client_cert[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----

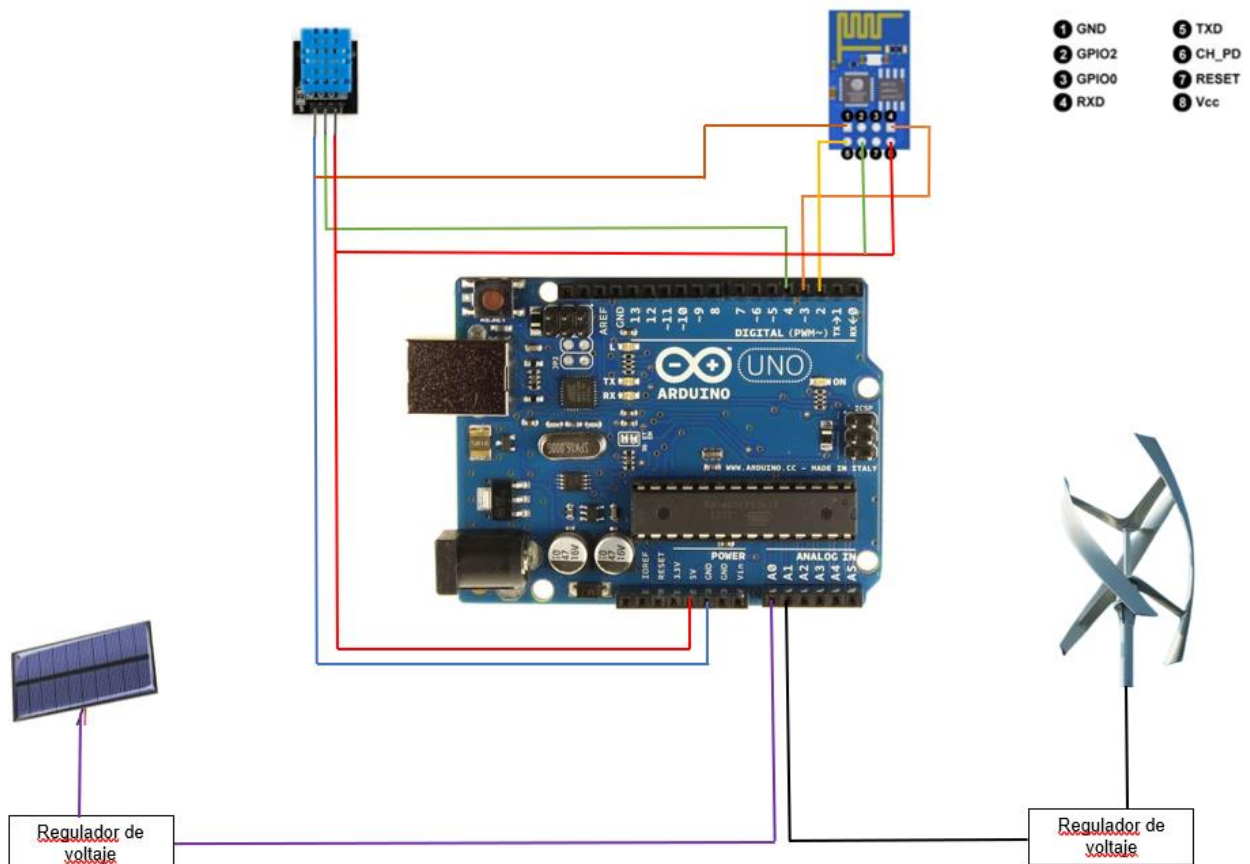
-----END CERTIFICATE-----
)KEY";

static const char privkey[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----

-----END RSA PRIVATE KEY-----
)KEY";

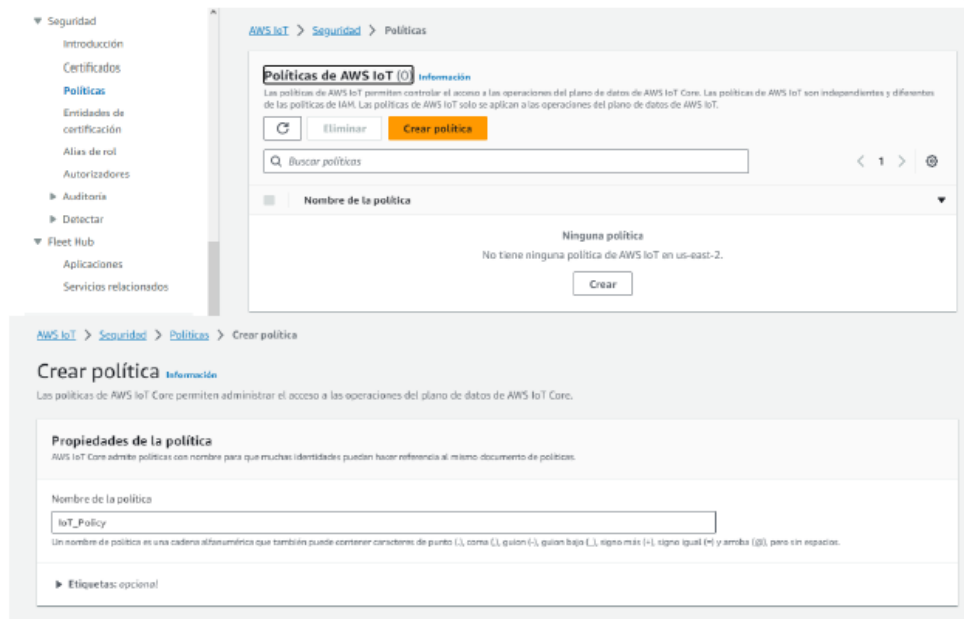
```

Diagrama del circuito:



Comunicación MQTT con la nube de AWS:

-Ingresar a AWS y crear una cuenta, seleccionar la opción “servicios” y seleccionar AWS IoT, crear recursos y certificados de AWS IoT iniciando con las políticas ingresando en el apartado de seguridad y eligiendo políticas y creando una nueva:



-Elegir los permisos de la política para poder conectarse, recibir, publicar y suscribirse:

Active version: 1 Info			Builder	JSON
Policy effect	Policy action	Policy resource		
Allow	iot:Connect	*		
Allow	iot:Receive	*		
Allow	iot:Subscribe	*		
Allow	iot:Publish	*		

-En AWS IoT en el apartado de Manage, ingresar a la opción “Thing” u “Objetos”, luego seleccionar Create things y crear un nuevo objeto:



-Crear y descargar los certificados:

[AWS IoT](#) > [Manage](#) > [Things](#) > [Create things](#) > Create single thing

Step 1
[Specify thing properties](#)

Step 2 - optional
Configure device certificate

Step 3 - optional
Attach policies to certificate

Configure device certificate - *optional* [Info](#)

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

☒ **Auto-generate a new certificate (recommended)**
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

☐ **Use my certificate**
Use a certificate signed by your own certificate authority.

☐ **Upload CSR**
Register your CA and use your own certificates on one or many devices.

☐ **Skip creating a certificate at this time**
You can create a certificate for this thing and attach a policy to the certificate at a later time.

Cancel Previous **Next**

-Ver mensajes de MQTT en el apartado de MQTT test client y suscribirse a “esp8266/pub” el cual se especificó en el código del módulo WiFi:

[Subscribe to a topic](#) | Publish to a topic

Topic filter [Info](#)
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

esp8266/pub

► Additional configuration

Subscribe

Subscriptions | esp8266/pub | [Pause](#) | [Clear](#) | [Export](#) | [Edit](#)

esp8266/pub [♥](#) [✕](#)

Message payload

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

► Additional configuration

[Publish](#)

No messages have been sent to this subscription yet. Please send a message to this subscription to see messages here.

Activate Windows
Go to Settings to activate Windows.

Creación de Tabla de DynamoDB:

-Se buscó DynamoDB dentro de los servicios de AWS, en “Tables” se seleccionó “CreateTable”. Se eligió un nombre para la tabla, siendo “Clima” en este caso al usar un sensor de humedad y temperatura, con una clave de partición “ID” que se generará por la función lambda y para llave de ordenamiento un “Timestamp” como parte del mensaje MQTT desde el arduino.

Services Search [Alt+S]

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.

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.

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.

String ▼

1 to 255 characters and case sensitive.

Services Search [Alt+S]

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.

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.

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.

String ▼

1 to 255 characters and case sensitive.

- En la sección Table settings, se seleccionó “Default Settings” y se hizo clic en “CreateTable”.

Services Search [Alt+S]

Table settings

☒ Default settings

The fastest way to create your table. You can modify these settings now or after your table has been created.

☐ Customize settings

Use these advanced features to make DynamoDB work better for your needs.

- En el servicio Lambda, en la sección Functions se seleccionó Create Function

Lambda > Functions

Functions (0)

Last fetched now

Actions ▼

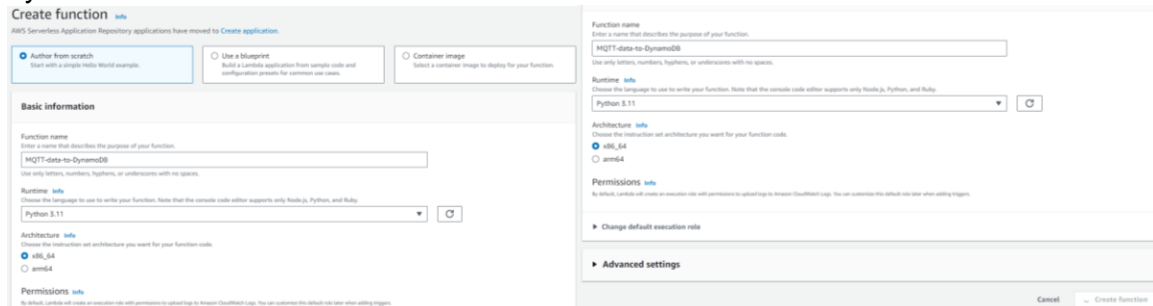
Create function

< 1 > ⚙

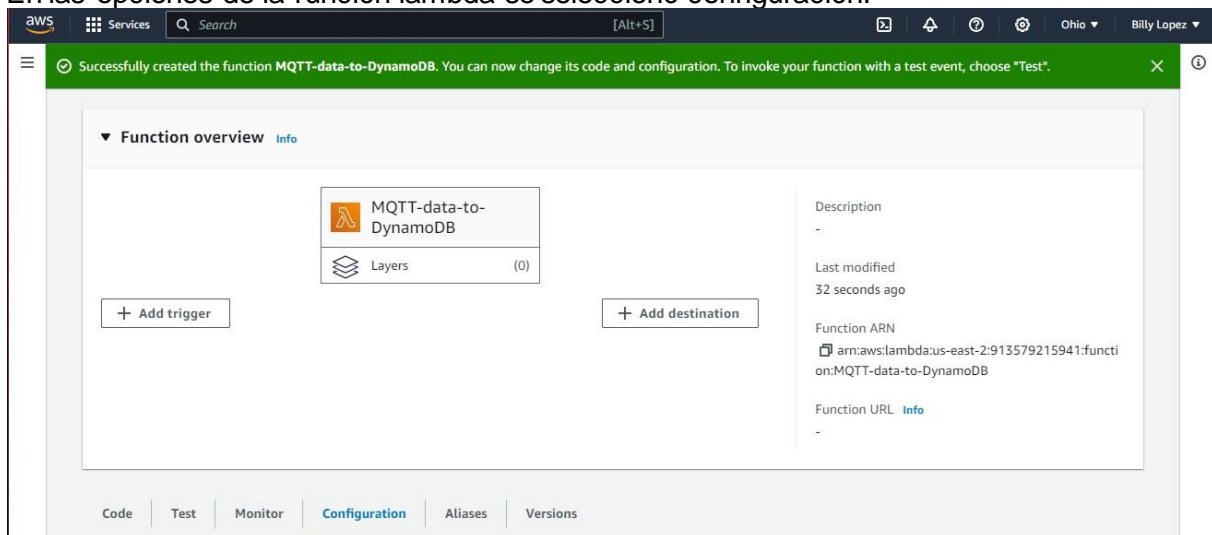
	Function name ▼	Description ▼	Package type ▼	Runtime ▼	Last modified ▼
There is no data to display.					

Creación de Función Lambda para que DynamoDB pueda recibir los datos por medio de MQTT

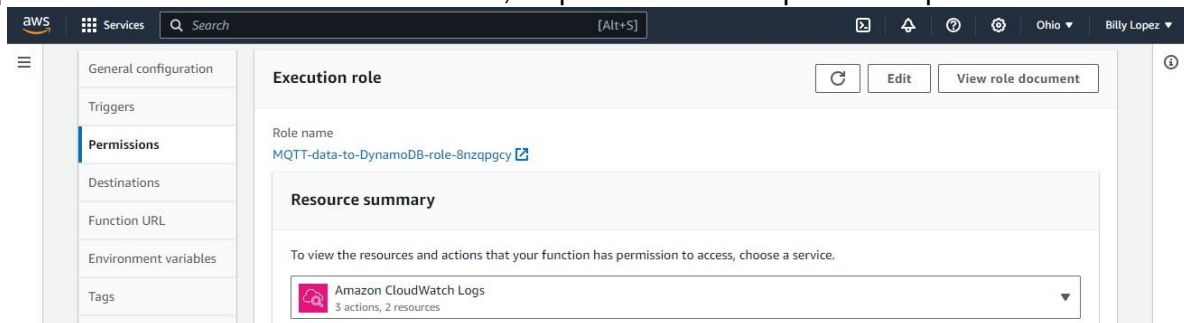
- Se coloca un nombre descriptivo a la función, en el apartado Runtime se coloca Python 3.11 y se selecciona Create Function.



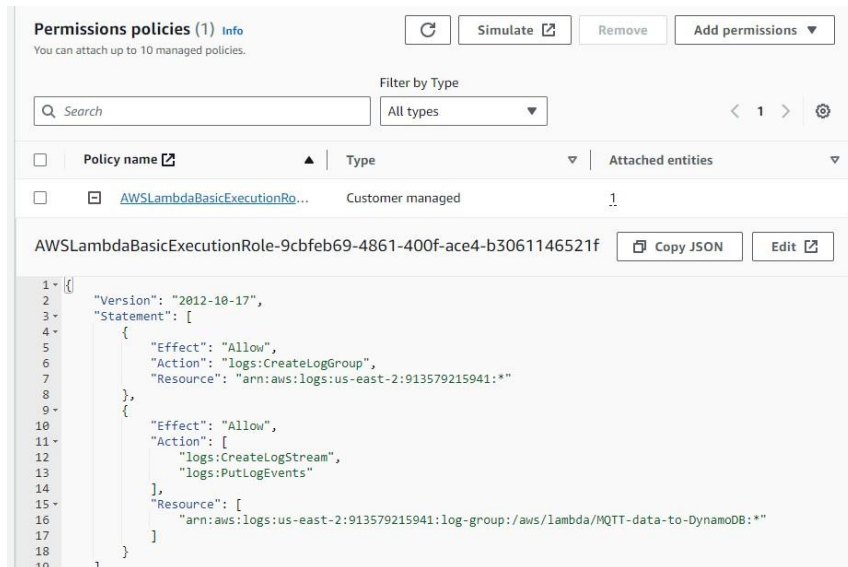
- En las opciones de la función lambda se seleccionó configuración.



- Dentro de las opciones de configuración se seleccionó Permissions y se hizo clic en el hipervínculo del rol asociado a la función, lo que nos lleva a la política de permisos de la misma.



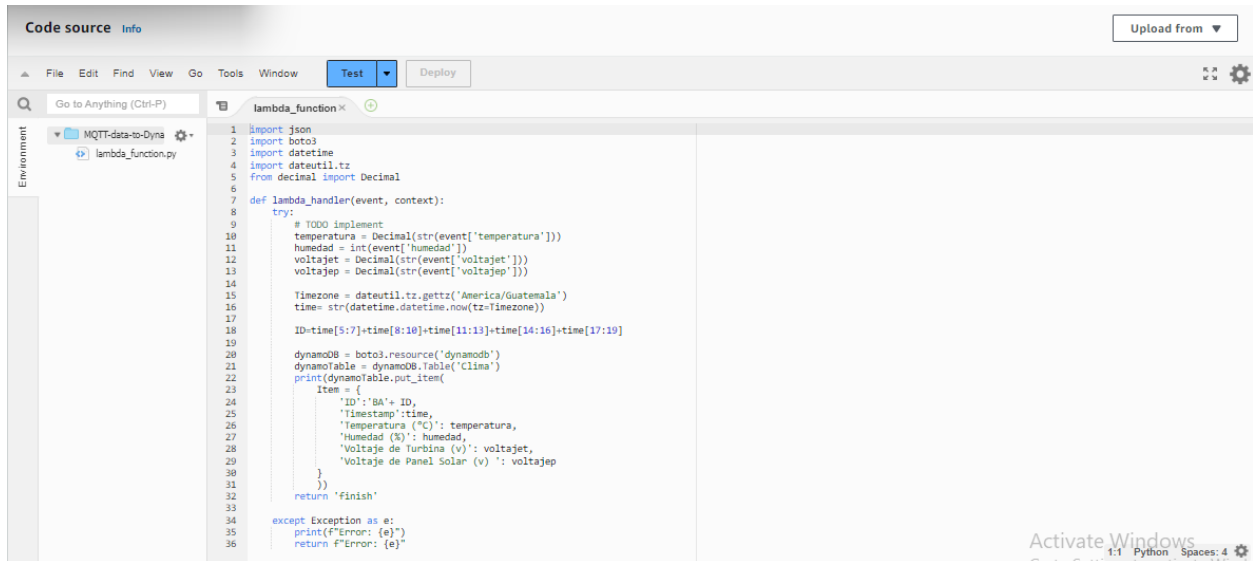
- Se selecciona el signo **+** a la par del nombre de la política para poder ver el código de la misma, luego edit para modificar la política asignada a la función:



- Se agrega el permiso necesario para escribir sobre la tabla creada al inicio del laboratorio. Para esto se obtuvo el identificador ARN de la tabla. En la sección Additional Information se copió el ARN de la tabla para agregarlo en el apartado de Resource de la sección de permisos adicionales.



- Se hizo clic en la opción de Code de la función lambda. En un ejemplo de código se realiza una escritura en la tabla que se creó al inicio en DynamoDB desde la función lambda. Se ajustó el código para simular la creación de un registro en la tabla Clima y se presionó Deploy.



- Se selecciona la flecha desplegable de Test y se hizo clic en Configure test event.

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event ☐ Edit saved event

Event name

MQTT-input

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

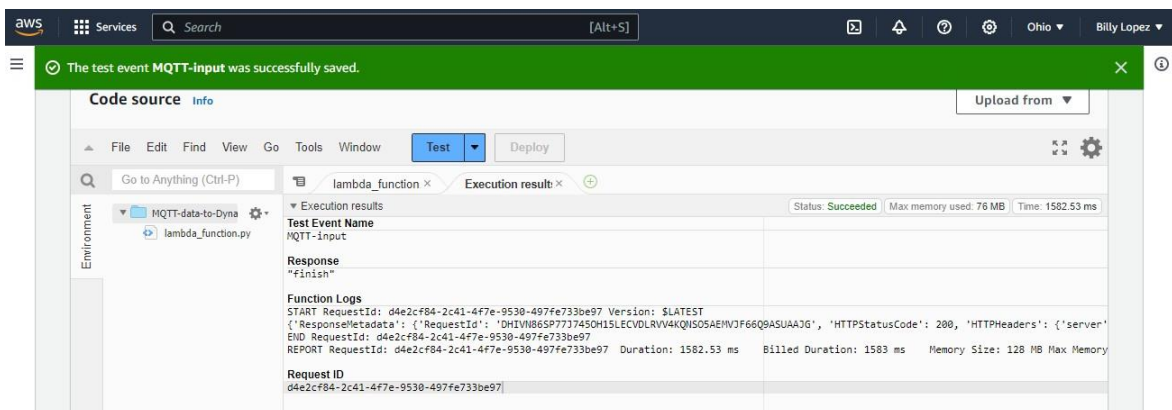
hello-world

Event JSON

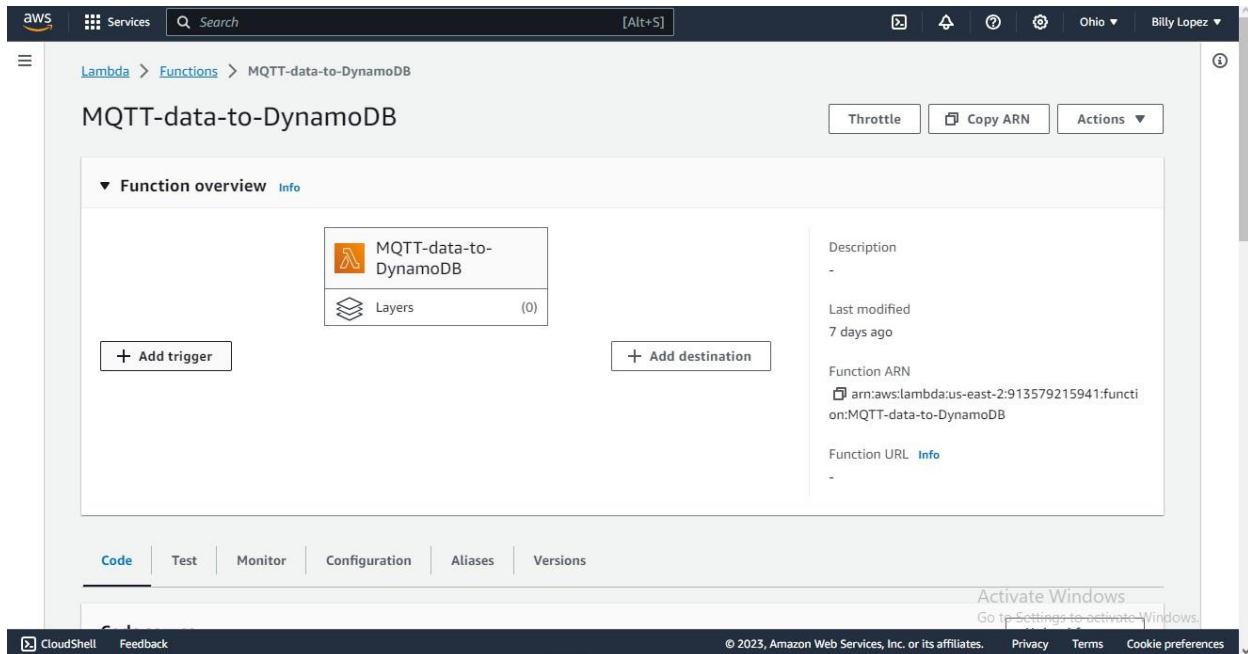
Format JSON

```
1 {
2   "Temperatura": "24.5",
3   "Humedad": "19"
4 }
```

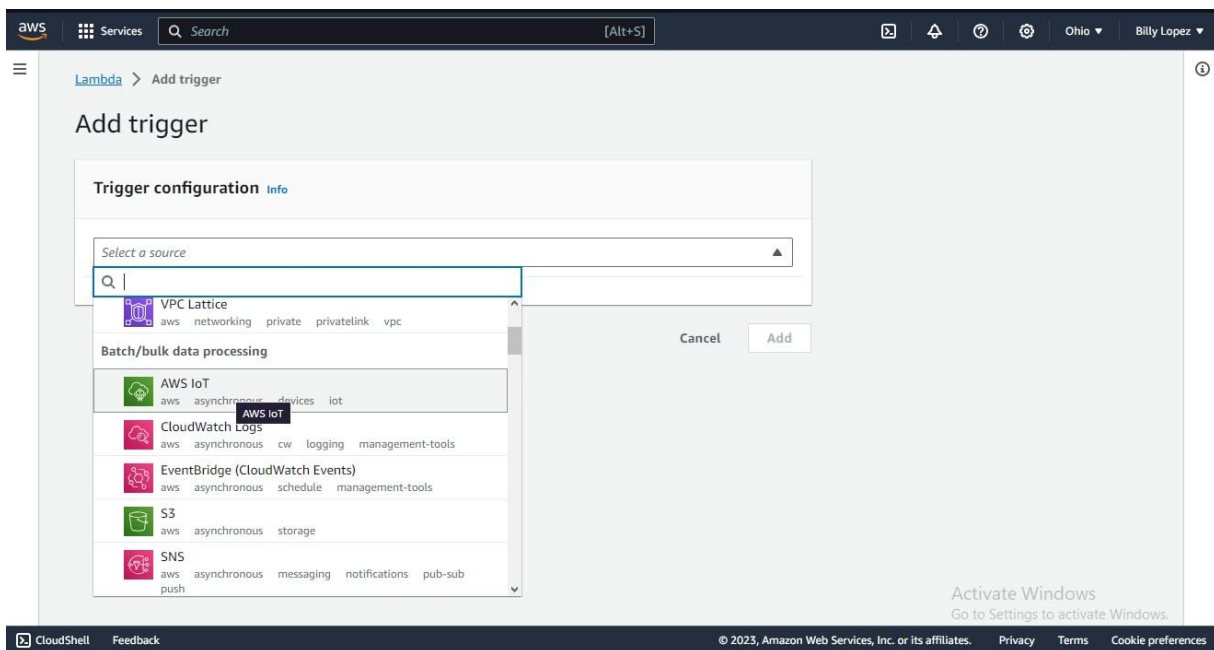
Cancel Invoke Save



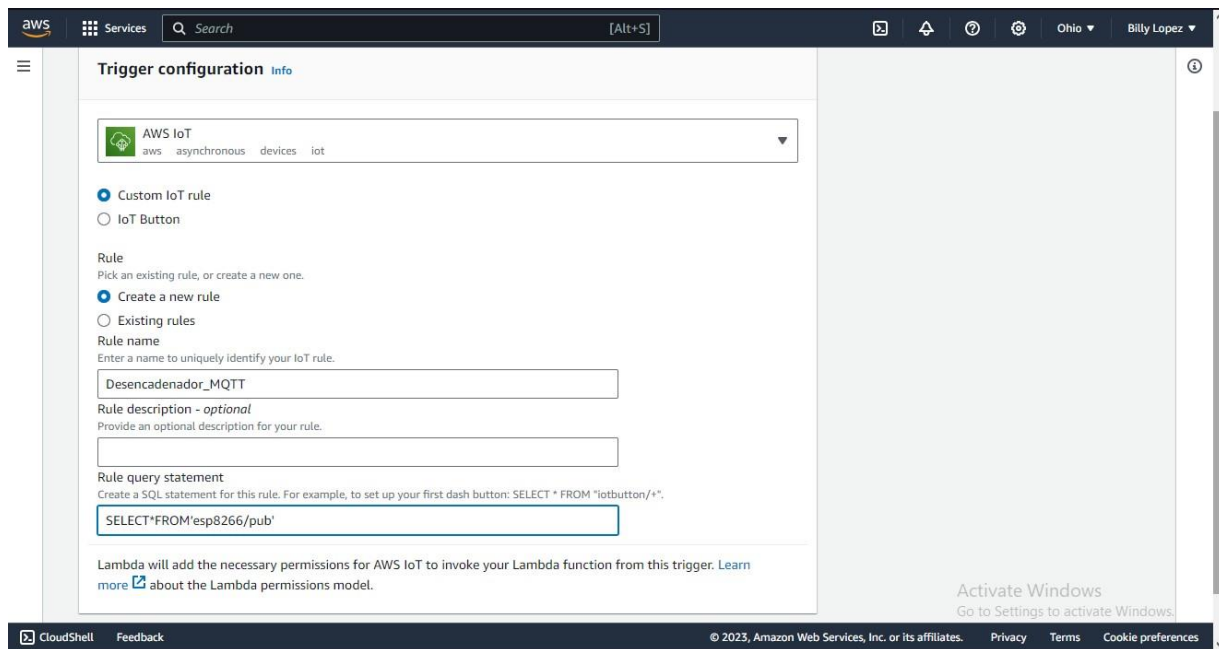
- Luego de seleccionar la función Lambda respectiva, se desplegó Function overview y se seleccionó “Add trigger”.



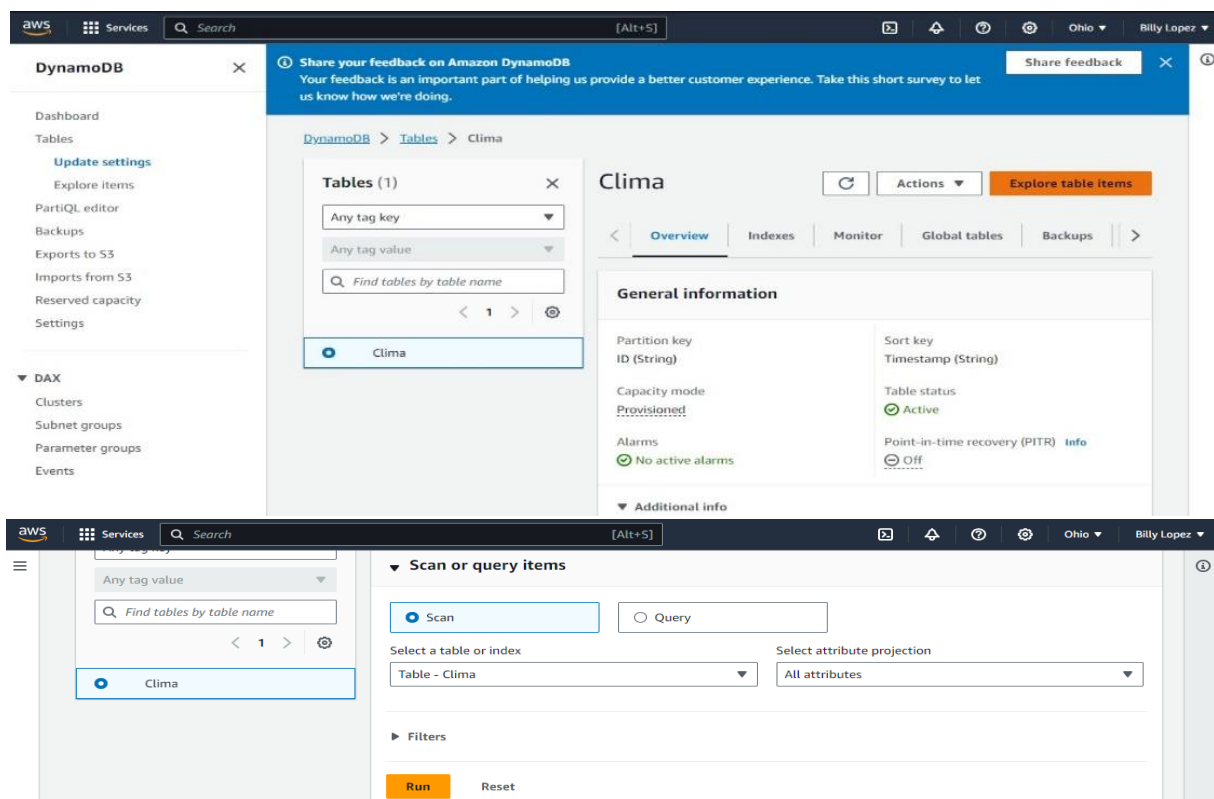
- En el desencadenador luego de presionar Add trigger, en la parte select a source se seleccionó entre las opciones “AWS IoT”, lo cual desplegó opciones de configuración.



- En la configuración del desencadenar se seleccionó “Custom IoT rule” y “Create a new rule”. En el apartado “Rule Name” se colocó el nombre de “Desencadenador_MQTT” para identificar la regla. En “Rule query Statement” se creó un statement SQL para que cada vez que se tenga un registro de entrada en IoT Core se produzca la ejecución de la función. La instancia SQL se ve de la siguiente manera: `SELECT*FROM"esp8266/pub"`. Por último, se presionó ADD.



- En el servicio DynamoDB se selecciona la tabla Clima y se hizo clic en Explore table items, en el cual, se observa el registro creado desde la función lambda en la tabla.



DynamoDB > Explore Items > Clima

Tables (1)

Any tag key

Any tag value

Find tables by table name

Clima

Clima

Scan or query items

Expand to query or scan items.

Items returned (50)

Actions Create Item

ID (String)	Timestamp (String)	Humedad (L...)	Temperatura (°C)	Voltaje de Panel Solar (v)	Voltaje de Turbina (v)
BA1117000628	2023-11-17 00:06:28.304963-06:00	84	23.8	0.34	0.18
BA1117000753	2023-11-17 00:07:53.712186-06:00	81	24.4	0.18	0.14
BA1117000653	2023-11-17 00:06:53.940530-06:00	84	23.9	0.38	0.25
BA1117000453	2023-11-17 00:04:53.258424-06:00	89	22.7	0.22	0.16
BA1117000623	2023-11-17 00:06:23.305878-06:00	79	24.5	0.2	0.15
BA1117000548	2023-11-17 00:05:48.282612-06:00	90	23.3	0.45	0.34
BA1117000738	2023-11-17 00:07:38.275504-06:00	81	24.4	0.19	0.14
BA1117000458	2023-11-17 00:04:58.069936-06:00	88	22.7	0.2	0.16
BA1117000528	2023-11-17 00:05:28.267595-06:00	90	23.2	2.27	2.85
BA1117000738	2023-11-17 00:07:38.304412-06:00	80	24.5	0.21	

Activate Windows
Go to Settings to activate Windows.

Conexión con AWS Analytics

-Ingresar a la consola de Amazon Athena, es requerido crear y preparar S3 Data Bucket:

Settings

Settings apply by default to all new queries. [Learn more](#)

Workgroup: **primary**

Query result location [Select](#)

The S3 path requires a trailing slash. Example: s3://query-results-bucket/folder/

Encrypt query results ☐ [?](#)

Autocomplete ☐ [?](#)

Cancel Save

-Crear almacenamiento en S3 para recibir data de Athena

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

-Crear una nueva fuente de datos de Athena

Athena Query editor Saved queries History **Data sources** Workgroup : primary

Data sources

Data sources that Athena can connect to are listed below by their catalog names. You can connect Athena to

[Connect data source](#) [View details](#) [Edit](#) [Delete](#)

Filter:

Catalog name
<input type="radio"/> AwsDataCatalog

-Elegir Amazon DynamoDB:

Choose a data source

Choose the data source to query with Athena. After you choose a data source, you will configure a Lambda function to handle the connection. [Learn more](#)

☐ Amazon CloudWatch Logs

☐ Amazon CloudWatch Metrics

☐ Amazon DocumentDB

☒ Amazon DynamoDB

☐ Amazon Redshift

☐ Apache HBase

-Elegir "Configure new AWS Lambda function" para crear la conexión de Lambda

[Lambda](#) > [Functions](#) > [Create function](#) > Review, configure and deploy

AthenaDynamoDBConnector — version 2021.33.1

Review, configure and deploy

[Copy as SAM Resource](#)

-Elegir "SpillBucket", "AthenaCatalogName" and confirm "Custom IAM Roles", and then click "Deploy". El Sistema automáticamente crea AWS CloudFormation para Lambda Connector.

Application settings

Application name
The stack name of this application created via AWS CloudFormation

AthenaDynamoDBConnector

SpillBucket
The name of the bucket where this function can spill data.

nutchanon-athena-ddb-spill

▼ ConnectorConfig

AthenaCatalogName
The name you will give to this catalog in Athena. It will also be used as the function name. This name must satisfy the pattern `^[a-z0-9-_{1,64}]$`

dynamocatalog

DisableSpillEncryption
WARNING: If set to 'true' encryption for spilled data is disabled.

false

LambdaMemory
Lambda memory in MB (min 128 - 3008 max).

3008

LambdaTimeout
Maximum Lambda invocation runtime in seconds. (min 1 - 900 max)

900

SpillPrefix
The prefix within SpillBucket where this function can spill data.

athena-spill


☒ I acknowledge that this app creates custom IAM roles. [Info](#)

-Regresar a elegir Lambda Connector creada con anterioridad, colocar el nombre del catálogo.

Connection details: Amazon DynamoDB

choose a Lambda function that is configured to connect to your data source, or create and configure a Lambda f

Lambda function Choose or configure a new AWS Lambda function to connect to the data source.

dynamocatalog 

[Configure new AWS Lambda function](#) 

Lambda function ARN [arn:aws:lambda:ap-southeast-1:██████████:function:dynamocatalog](#) 

Catalog name Create a unique name to specify this data source within a SQL statement.

dynamocatalog

Use up to 127 characters and it must be unique within your account. It cannot be changed after creation. Valid characters are a-z, A-Z, 0-9, _ (underscore), @ (at) and - (hyphen).

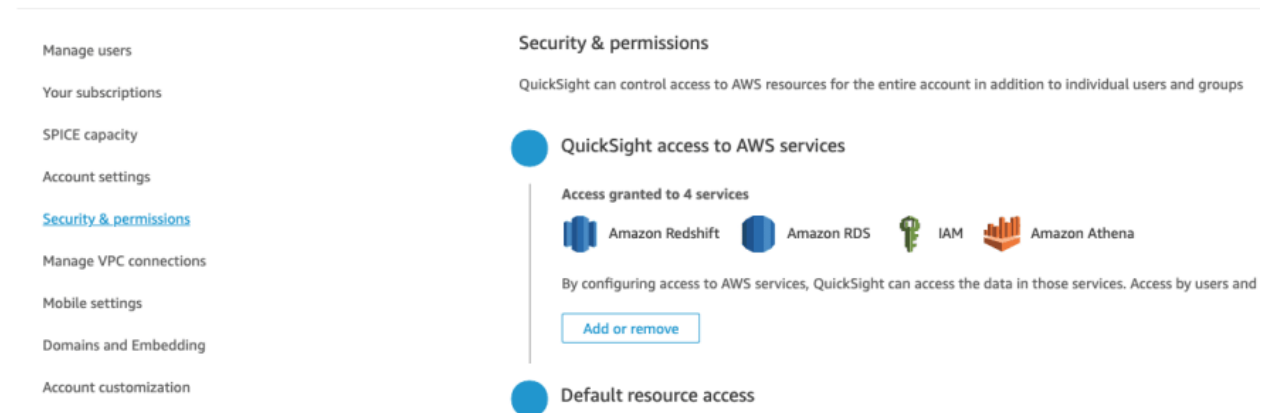
**Description
(optional)**

Enter a description

Use up to 1024 characters.

Conexión con QuickSight para análisis y diseño del Dashboard






-Ingresar a QuickSight e ir a "Manage QuickSight", en "Security & Permission" elegir "Add or Remove".



-Elegir Amazon Athena, Luego se verá la ventana de permisos de Amazon S3. Luego, se eligió Athena Spill y Result Data Bucket con los permisos de escritura.

QuickSight access to AWS services

QuickSight can connect to the selected AWS products & services below for all users & groups:

	Amazon Redshift Enables QuickSight to auto-discover clusters	<input checked="" type="checkbox"/>
	Amazon RDS Enables QuickSight to auto-discover instances	<input checked="" type="checkbox"/>
	IAM Enables you to invite IAM users from this AWS Account to access QuickSight	<input checked="" type="checkbox"/>
	Amazon S3 Enables QuickSight to auto-discover your Amazon S3 buckets Details	<input checked="" type="checkbox"/>
	Amazon Athena Enables QuickSight access to Amazon Athena databases	<input checked="" type="checkbox"/>

-Crear un dataset de Athena Data Source con el nombre de conexión, e.g."athena-dynamodb" y elegir el nombre del catálogo.

New Athena data source



Data source name

athena-dynamodb

Athena workgroup

[primary]

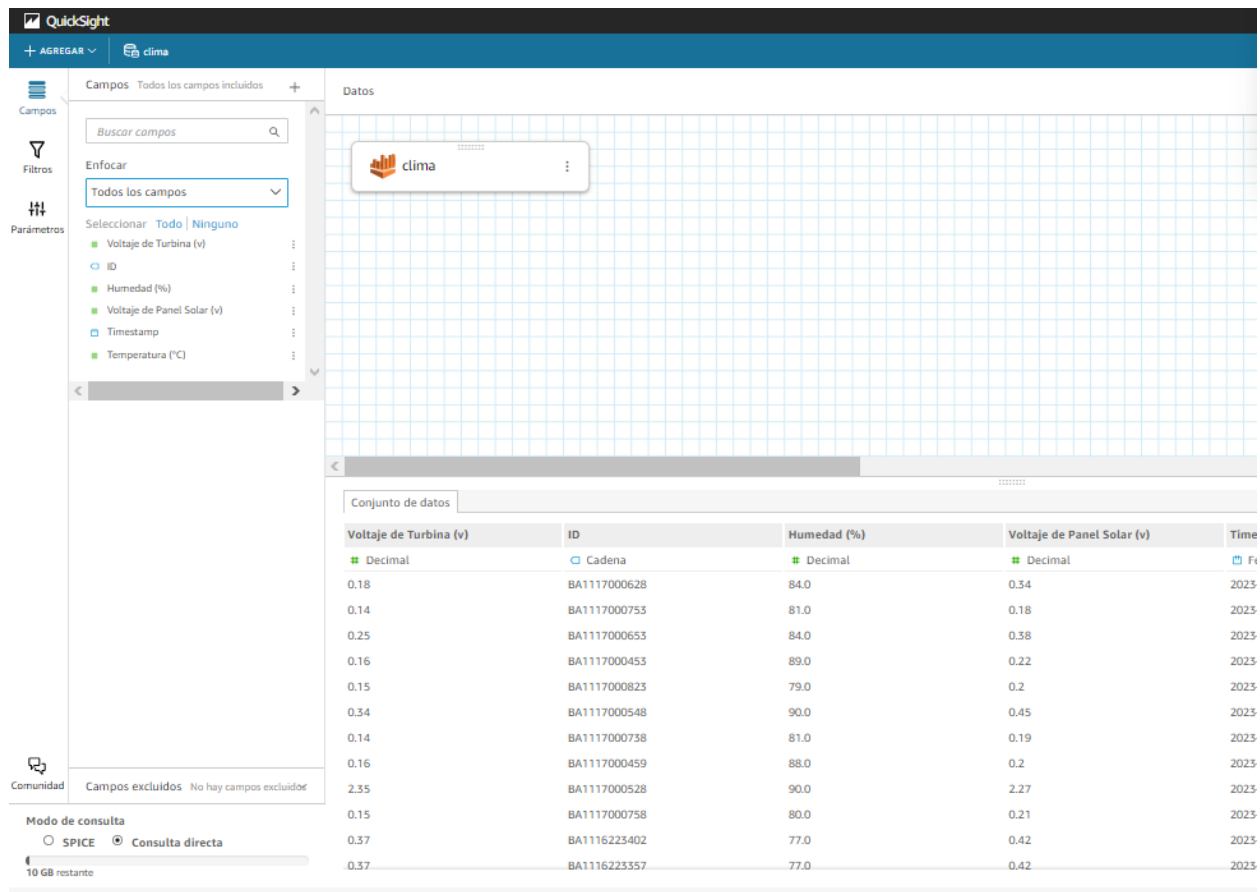


Validate connection

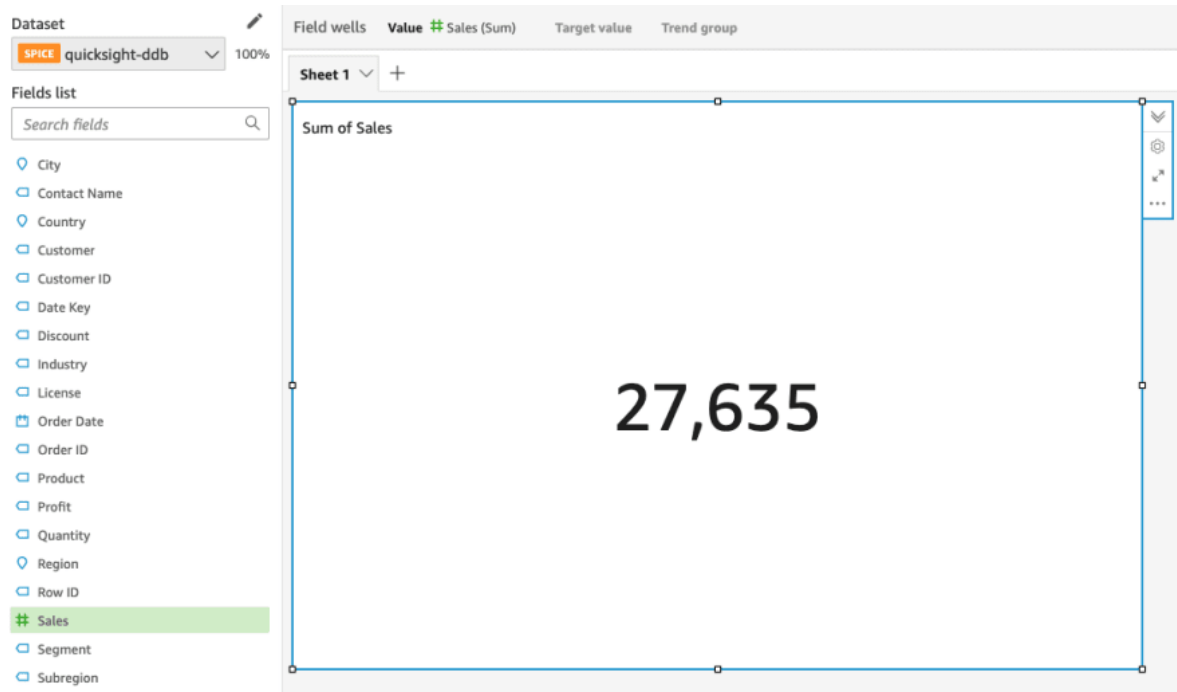
SSL is enabled

Create data source

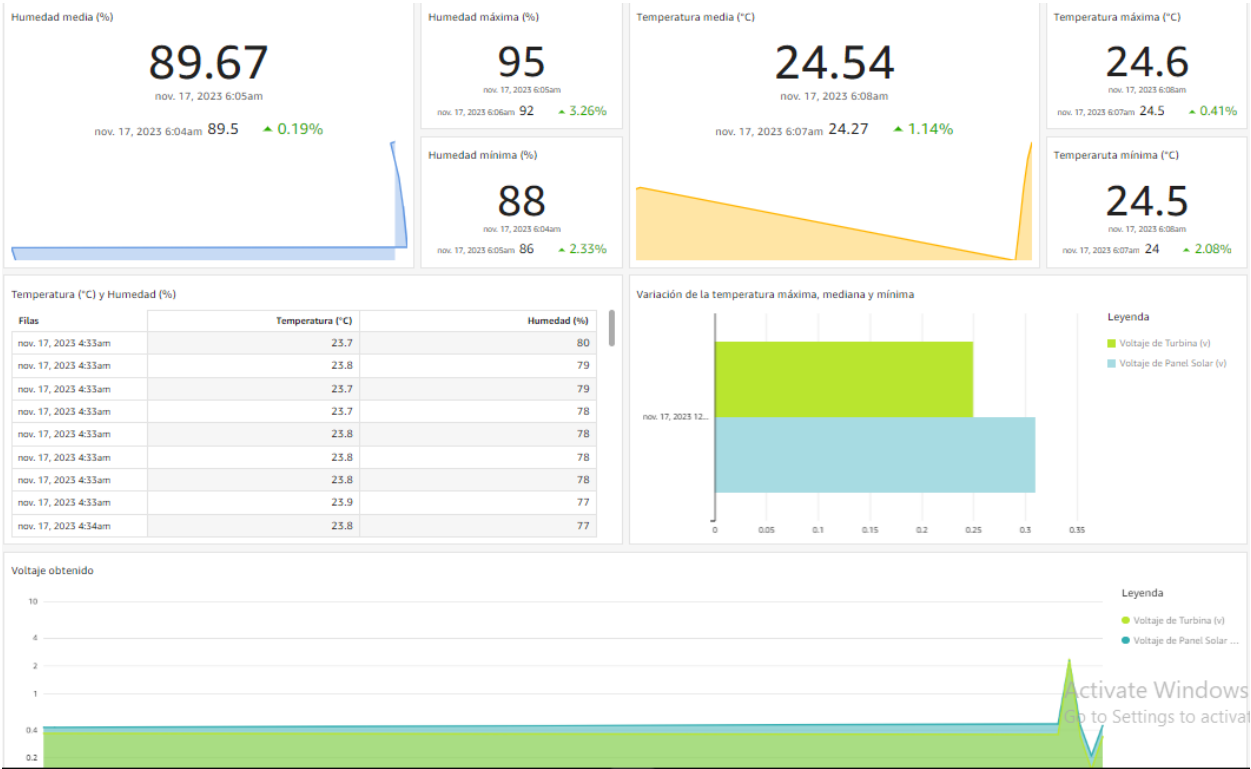
-Cambiar el tipo de data para cada columna apropiadamente para el cálculo y filtrado futuro.



Crear el análisis de la base de datos de DynamoDB en QuickSight



Crear el Dashboard de los datos



Este manual técnico ha sido elaborado con el objetivo de proporcionar a los usuarios una guía completa y detallada sobre la instalación, configuración y uso eficiente del proyecto “Modelo a Escala de una Estación Meteorológica que Emplea Energías Renovables”. Hemos abordado aspectos cruciales, desde los requisitos iniciales hasta las funcionalidades más avanzadas, con la intención de garantizar una experiencia de usuario sin contratiempos.

Nuestro compromiso con la calidad y la satisfacción del usuario se refleja en cada sección de este manual técnico. Si bien nos esforzamos por cubrir todos los aspectos relevantes, entendemos que pueden surgir situaciones únicas. Por lo tanto, alentamos a los usuarios a ponerse en contacto con nuestro equipo de soporte técnico para cualquier consulta adicional o asistencia que puedan necesitar.

Agradecemos la oportunidad de ser parte de su experiencia técnica y esperamos que este manual sirva como una herramienta valiosa para aprovechar al máximo todas las capacidades del proyecto “Modelo a Escala de una Estación Meteorológica que Emplea Energías Renovables”. Su retroalimentación es fundamental para nuestro continuo desarrollo, y estamos comprometidos a mejorar y actualizar este recurso para satisfacer las necesidades cambiantes de nuestros usuarios.

¡Gracias por elegir el proyecto “Modelo a Escala de una Estación Meteorológica que Emplea Energías Renovables”!