KENNESAW STATE UNIVERSITY

CPE 4040: Data Collection and Analysis, Spring 2024

Laboratory Report #7

A Data Project Using NodeRED, AWS IoT, and Streamlit

Team Members: Andrew Palmertree, Kevin Toyle

Electrical and Computer Engineering

Kennesaw State University

Faculty: Dr. Jeffrey L Yiin

Date of Lab Session: April 8, 2024

I. Objective

- Learned how to set up and store data from a DHC-11 temperature sensor circuit to an AWS S3 bucket.
- Gained proficiency in using Node-RED on the Raspberry Pi 4 to send data from the temperature sensor to the ASW IoT server.
- Acquired the knowledge to setting up and configuring a Streamlit platform to display the DHT-111 temperature sensor data on a web page.

II. Material List

Software:

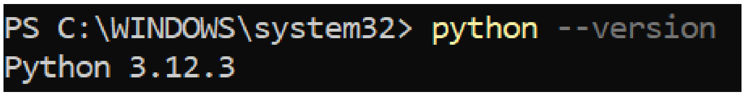
1. AWS IoT Core
2. AWS S3 Bucket
3. Node-RED
4. Streamlit
5. SSH
6. VNC
7. MobaXterm
8. app.py and digitalOut.py python script

Hardware:

1. Raspberry PI 4
2. Bread Board
3. DHT-11 Temperature Sensor
4. Three Female to Female jumper wires
5. 5V 2.5-amp voltage supply

III. Lab Procedures and Results

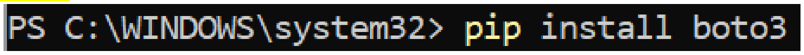
1. We first powered up the Raspberry Pi and connected to the Pi via a remote desktop connection through VNC. After we got connected to the Pi, we checked our python version on our Windows computer using the following command:



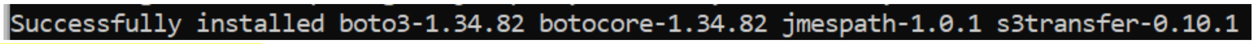
```
PS C:\WINDOWS\system32> python --version
Python 3.12.3
```

2. Next, we installed Boto3 (AWS development kit), Streamlit (also installs pandas and NumPy), and ploty (plotting library) using the following commands:

A) pip install boto3:

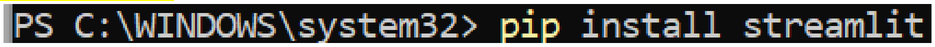


```
PS C:\WINDOWS\system32> pip install boto3
```

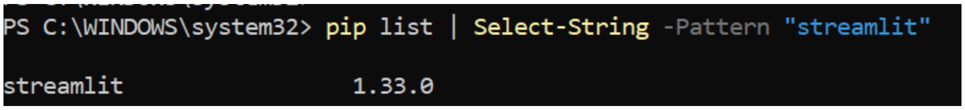


```
Successfully installed boto3-1.34.82 botocore-1.34.82 jmespath-1.0.1 s3transfer-0.10.1
```

B) pip install streamlit:



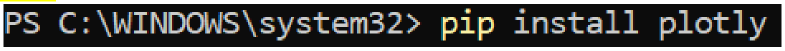
```
PS C:\WINDOWS\system32> pip install streamlit
```



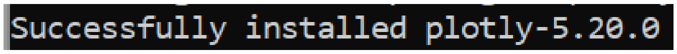
```
PS C:\WINDOWS\system32> pip list | Select-String -Pattern "streamlit"

streamlit                1.33.0
```

C) pip install plotly:



```
PS C:\WINDOWS\system32> pip install plotly
```



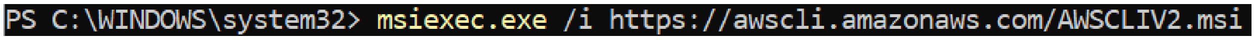
```
Successfully installed plotly-5.20.0
```

3. We then went to https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html and downloaded the ASW CLI to our windows computer. We then made sure it was installed using *aws –version*.

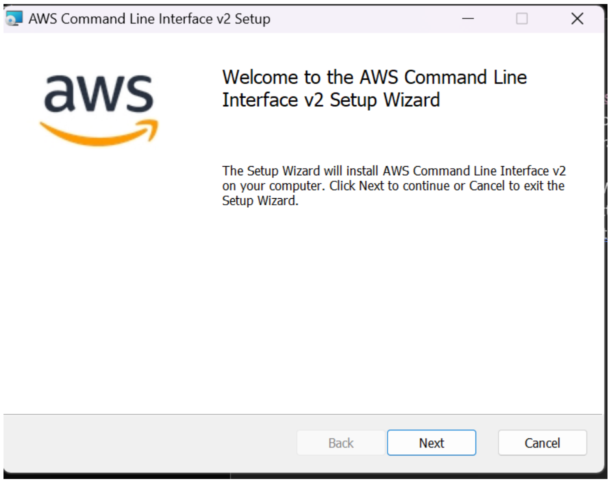
Installing AWS CLI:

A) Download the AWS CLI MSI installer for windows:

C:\> *msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi*



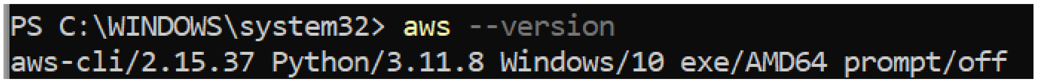
```
PS C:\WINDOWS\system32> msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
```



AWS Command Line Interface v2 Setup

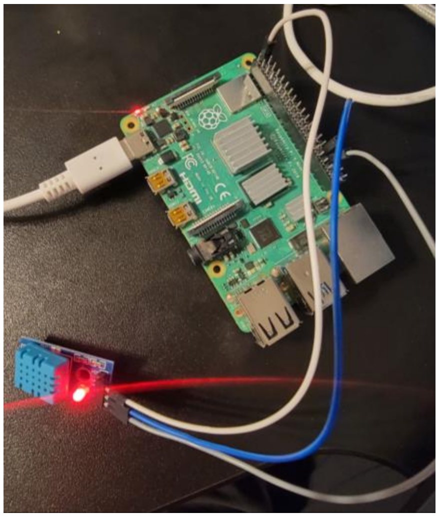
Welcome to the AWS Command Line Interface v2 Setup Wizard

The Setup Wizard will install AWS Command Line Interface v2 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

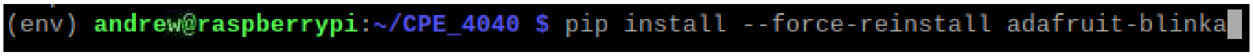


```
PS C:\WINDOWS\system32> aws --version
aws-cli/2.15.37 Python/3.11.8 Windows/10 exe/AMD64 prompt/off
```

4. We then created a DHT-11 temperature sensor circuit with the Raspberry Pi 4 using GPIO pin 1 (3.3V), pin 34 (GND), and pin 32 (BCM 12). DHT-11 circuit:

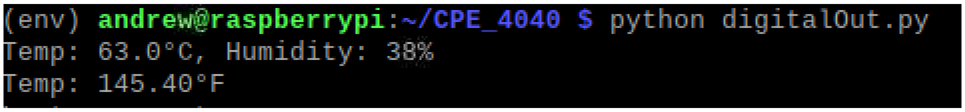


5. Next, we made sure that the digitalOut.py script is working properly:
   When we first ran the script, we had an error of model board wasn't installed.
   To fix this issues we made sure that I2C is enabled under Raspberry Pi configuration interfaces and force reinstall Adafruit-blinka.



```
(env) andrew@raspberrypi:~/CPE_4040 $ pip install --force-reinstall adafruit-blinka
```

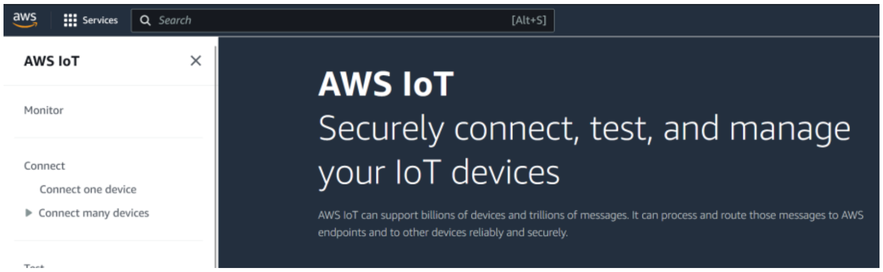
   After we fix that error, we could successfully run the digitalOut.py script:



```
(env) andrew@raspberrypi:~/CPE_4040 $ python digitalOut.py
Temp: 63.0°C, Humidity: 38%
Temp: 145.40°F
```

Section 2 Configuring AWS IoT Core:

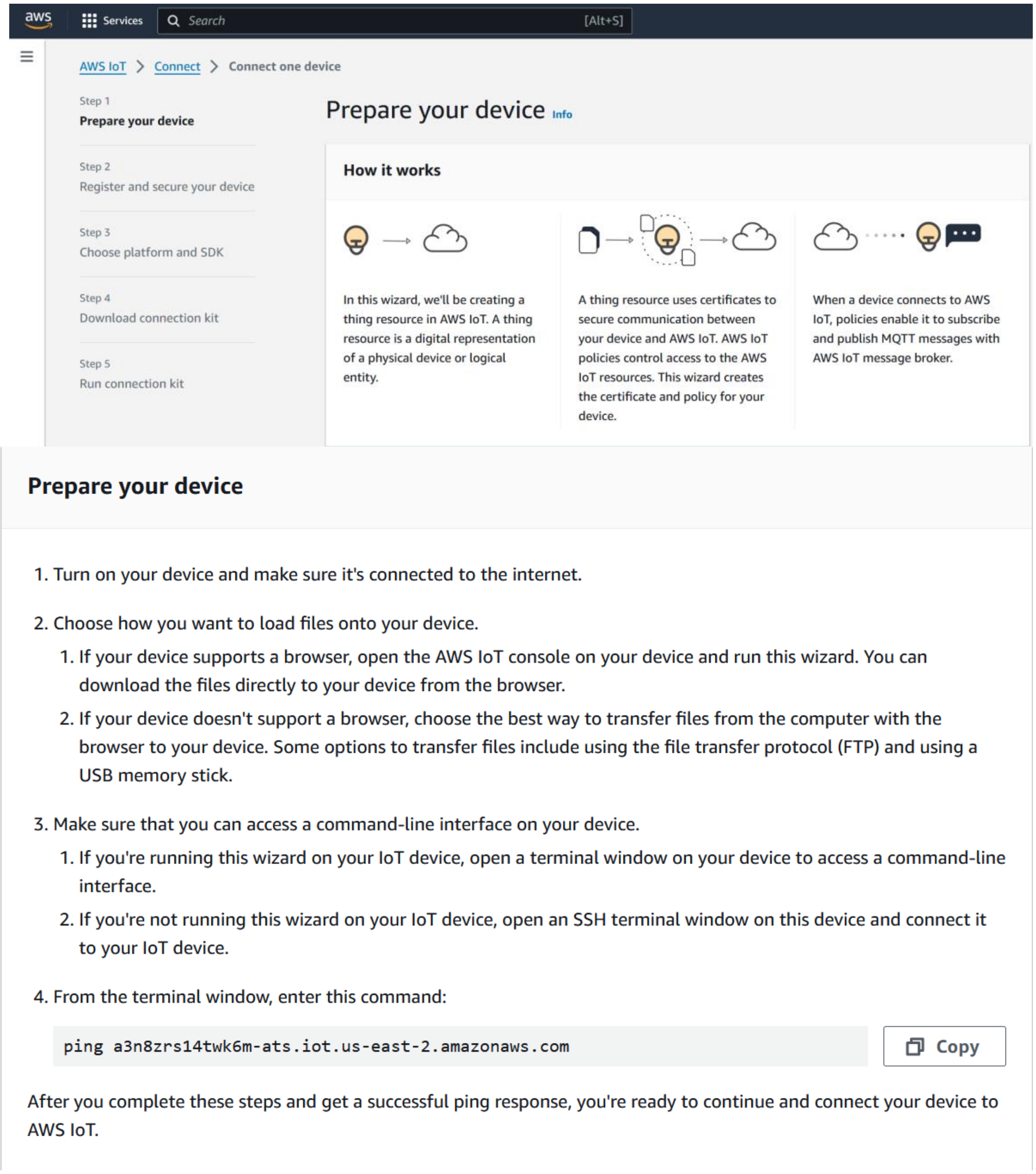
6. To create a new AWS IoT Core, we first went to https://aws.amazon.com/ and logged in. Once we logged in, we went to IoT Core by searching it the search bar.



AWS IoT
Securely connect, test, and manage your IoT devices

AWS IoT can support billions of devices and trillions of messages. It can process and route those messages to AWS endpoints and to other devices reliably and securely.

7. We then went to “Connect one device” under “Connect” in AWS IoT and created a new “Thing”.



Prepare your device

How it works

In this wizard, we'll be creating a thing resource in AWS IoT. A thing resource is a digital representation of a physical device or logical entity.

A thing resource uses certificates to secure communication between your device and AWS IoT. AWS IoT policies control access to the AWS IoT resources. This wizard creates the certificate and policy for your device.

When a device connects to AWS IoT, policies enable it to subscribe and publish MQTT messages with AWS IoT message broker.

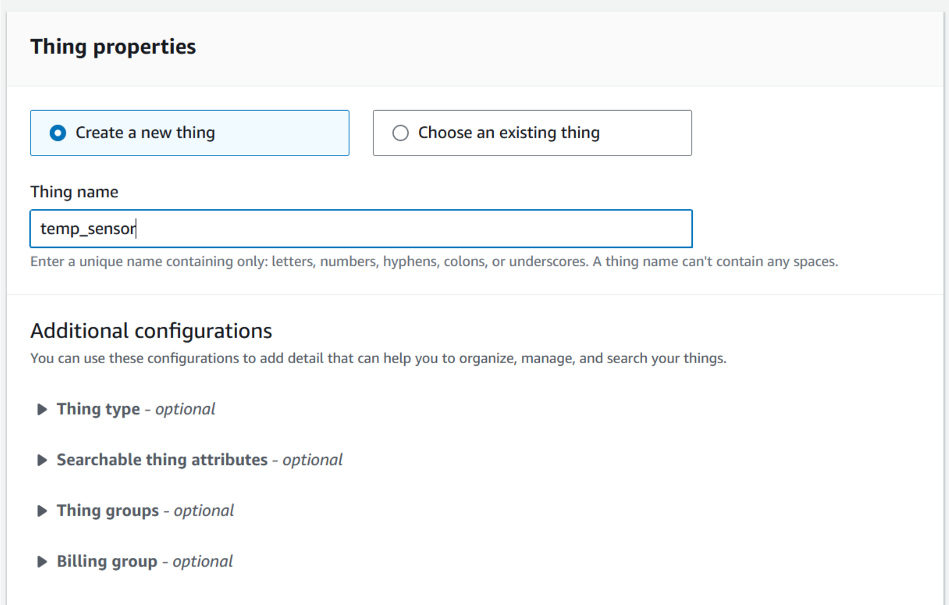
Prepare your device

1. Turn on your device and make sure it's connected to the internet.
2. Choose how you want to load files onto your device.
   1. If your device supports a browser, open the AWS IoT console on your device and run this wizard. You can download the files directly to your device from the browser.
   2. If your device doesn't support a browser, choose the best way to transfer files from the computer with the browser to your device. Some options to transfer files include using the file transfer protocol (FTP) and using a USB memory stick.
3. Make sure that you can access a command-line interface on your device.
   1. If you're running this wizard on your IoT device, open a terminal window on your device to access a command-line interface.
   2. If you're not running this wizard on your IoT device, open an SSH terminal window on this device and connect it to your IoT device.
4. From the terminal window, enter this command:

```
ping a3n8zrs14twk6m-ats.iot.us-east-2.amazonaws.com
```

After you complete these steps and get a successful ping response, you're ready to continue and connect your device to AWS IoT.

8. We named our new thing “temp_sensor”. We also selected “Linux / macOS” for the *Device platform system* and “Python” for the *AWS IoT Device SDK*:



Thing properties

● Create a new thing ○ Choose an existing thing

Thing name

temp_sensor

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

Additional configurations

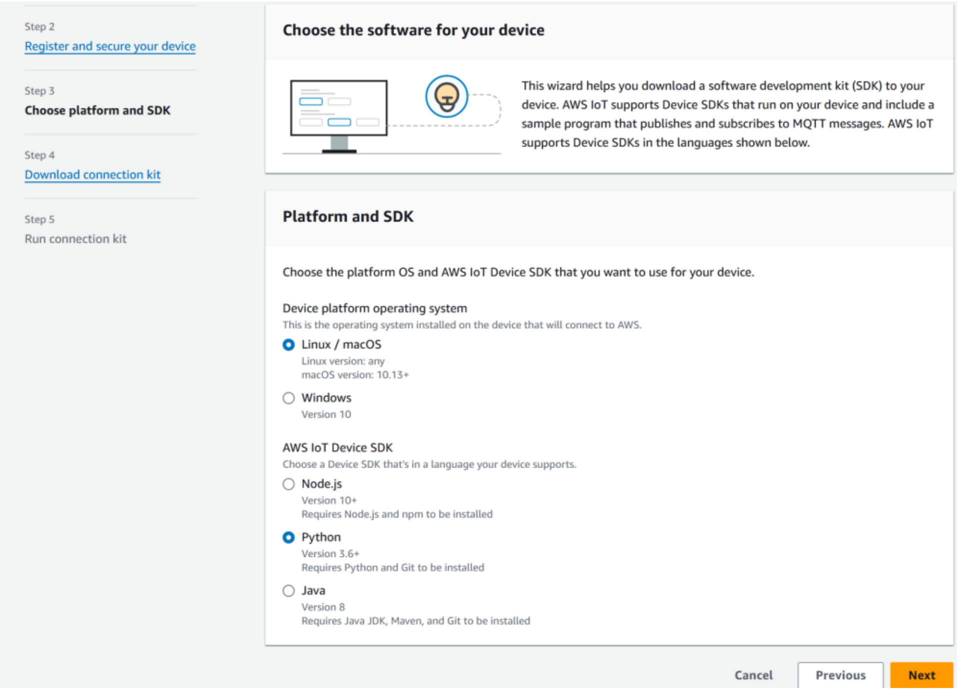
You can use these configurations to add detail that can help you to organize, manage, and search your things.

▶ **Thing type** - *optional*

▶ **Searchable thing attributes** - *optional*

▶ **Thing groups** - *optional*

▶ **Billing group** - *optional*



Choose the software for your device

This wizard helps you download a software development kit (SDK) to your device. AWS IoT supports Device SDKs that run on your device and include a sample program that publishes and subscribes to MQTT messages. AWS IoT supports Device SDKs in the languages shown below.

Platform and SDK

Choose the platform OS and AWS IoT Device SDK that you want to use for your device.

Device platform operating system

This is the operating system installed on the device that will connect to AWS.

● Linux / macOS
Linux version: any
macOS version: 10.13+

○ Windows
Version 10

AWS IoT Device SDK

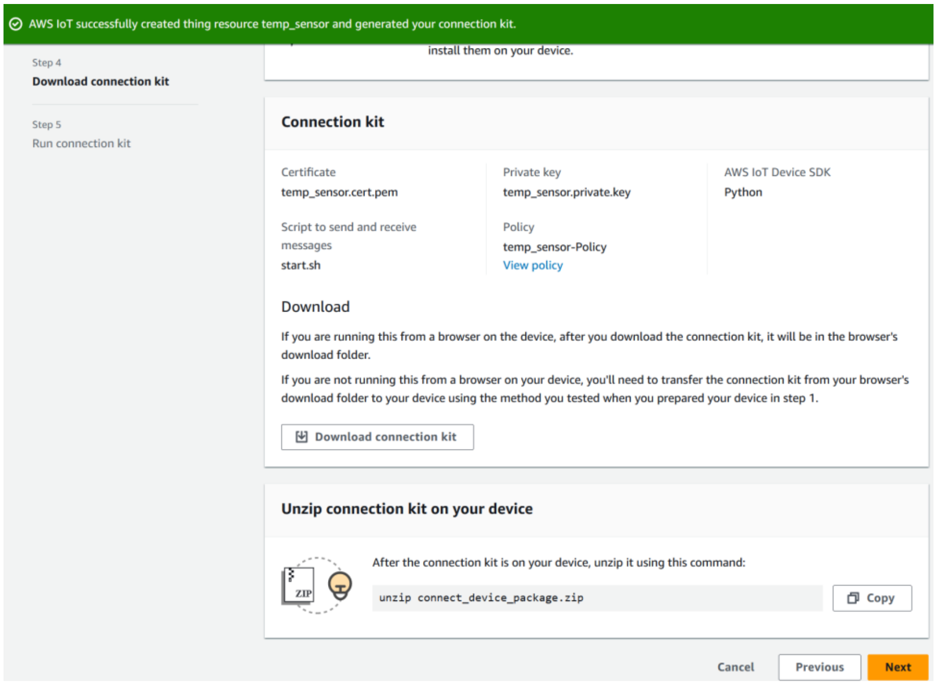
Choose a Device SDK that's in a language your device supports.

○ Node.js
Version 10+
Requires Node.js and npm to be installed

● Python
Version 3.6+
Requires Python and Git to be installed

○ Java
Version 8
Requires Java JDK, Maven, and Git to be installed

9. Next, we download the connection kit which contains all the keys and certificates we need to connect to the AWS IoT Thing.



AWS IoT successfully created thing resource temp_sensor and generated your connection kit.

install them on your device.

Step 4
Download connection kit

Step 5
Run connection kit

Connection kit

Certificate temp_sensor.cert.pem	Private key temp_sensor.private.key	AWS IoT Device SDK Python
Script to send and receive messages start.sh	Policy temp_sensor-Policy View policy	

Download

If you are running this from a browser on the device, after you download the connection kit, it will be in the browser's download folder.

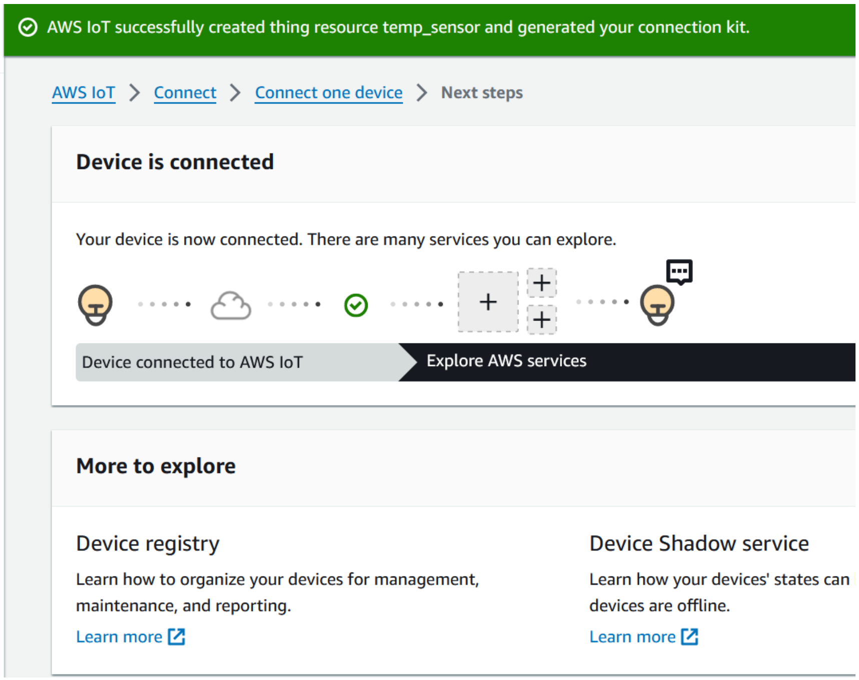
If you are not running this from a browser on your device, you'll need to transfer the connection kit from your browser's download folder to your device using the method you tested when you prepared your device in step 1.

Unzip connection kit on your device

After the connection kit is on your device, unzip it using this command:

```
unzip connect_device_package.zip
```

Just press "next" and the device should be connected:



AWS IoT successfully created thing resource temp_sensor and generated your connection kit.

AWS IoT > Connect > Connect one device > Next steps

Device is connected

Your device is now connected. There are many services you can explore.

Device connected to AWS IoT
Explore AWS services

More to explore

Device registry

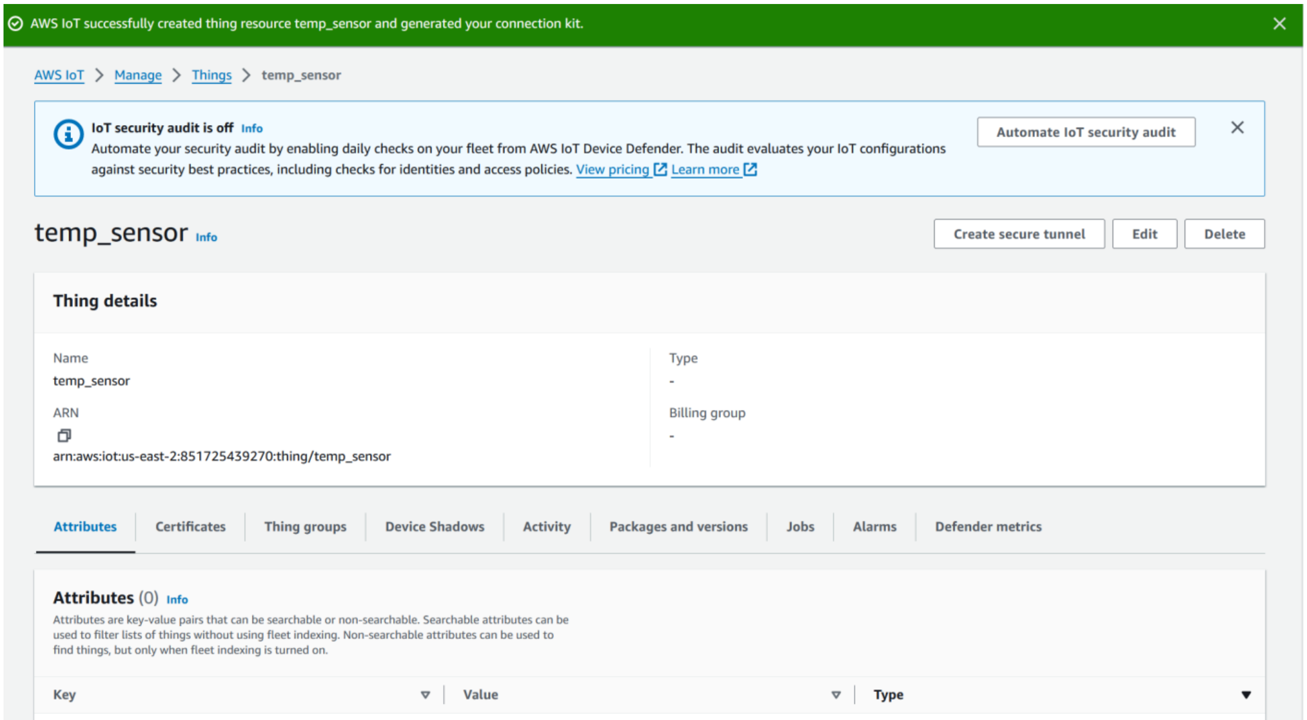
Learn how to organize your devices for management, maintenance, and reporting.

Learn more

Device Shadow service

Learn how your devices' states can devices are offline.

Learn more



AWS IoT successfully created thing resource temp_sensor and generated your connection kit.

AWS IoT > Manage > Things > temp_sensor

IoT security audit is off Info
Automate your security audit by enabling daily checks on your fleet from AWS IoT Device Defender. The audit evaluates your IoT configurations against security best practices, including checks for identities and access policies. View pricing Learn more

Automate IoT security audit

temp_sensor Info

Create secure tunnel | Edit | Delete

Thing details

Name
temp_sensor

ARN
arn:aws:iot:us-east-2:851725439270:thing/temp_sensor

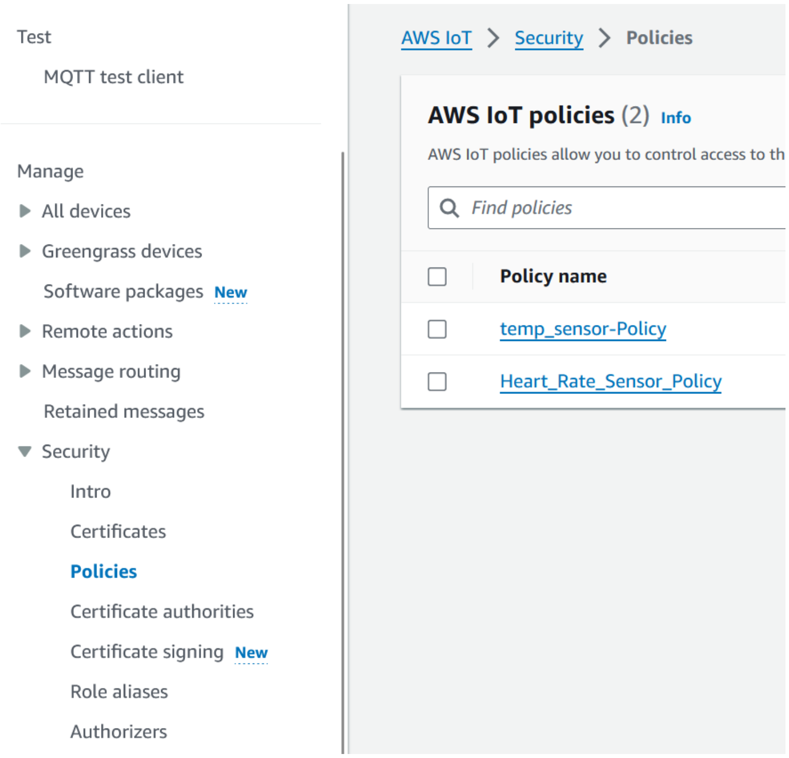
Type
-

Billing group
-

Attributes (0) Info
Attributes are key-value pairs that can be searchable or non-searchable. Searchable attributes can be used to filter lists of things without using fleet indexing. Non-searchable attributes can be used to find things, but only when fleet indexing is turned on.

Key	Value	Type

10. After the AWS IoT Thing is created, we then create a new policy. First, go to “Policies” under “Security” and click “Create policy”.

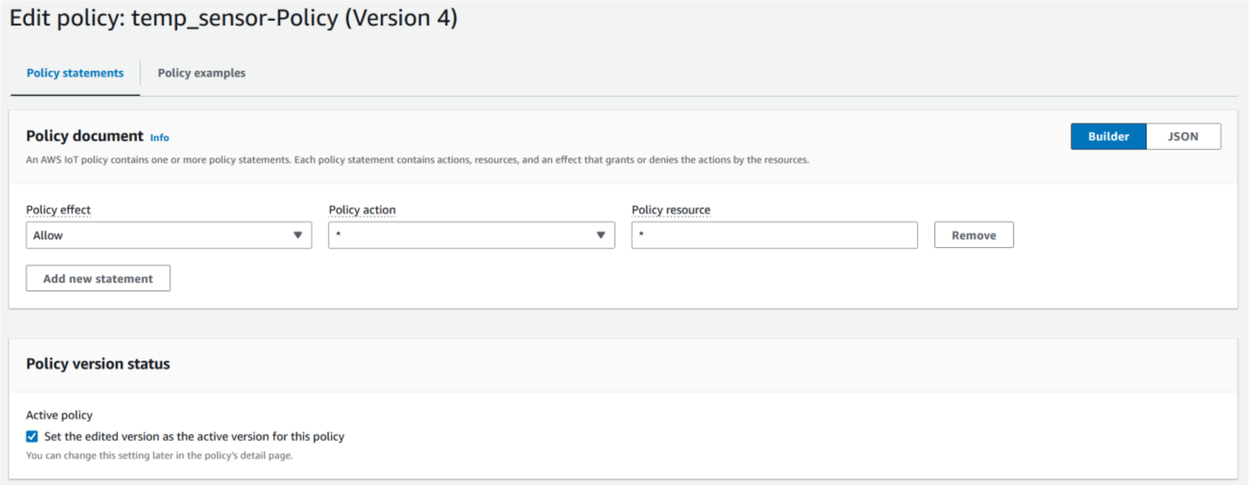


AWS IoT policies (2) Info
AWS IoT policies allow you to control access to th

Find policies

□	Policy name
□	temp_sensor-Policy
□	Heart_Rate_Sensor_Policy

We named the policy “temp_sensor-Policy” and set the “Policy action” and “Policy resource” to “*”. We also checked the “Policy Version Status” box to save the changes.



Edit policy: temp_sensor-Policy (Version 4)

Policy document

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect: Allow

Policy action: *

Policy resource: *

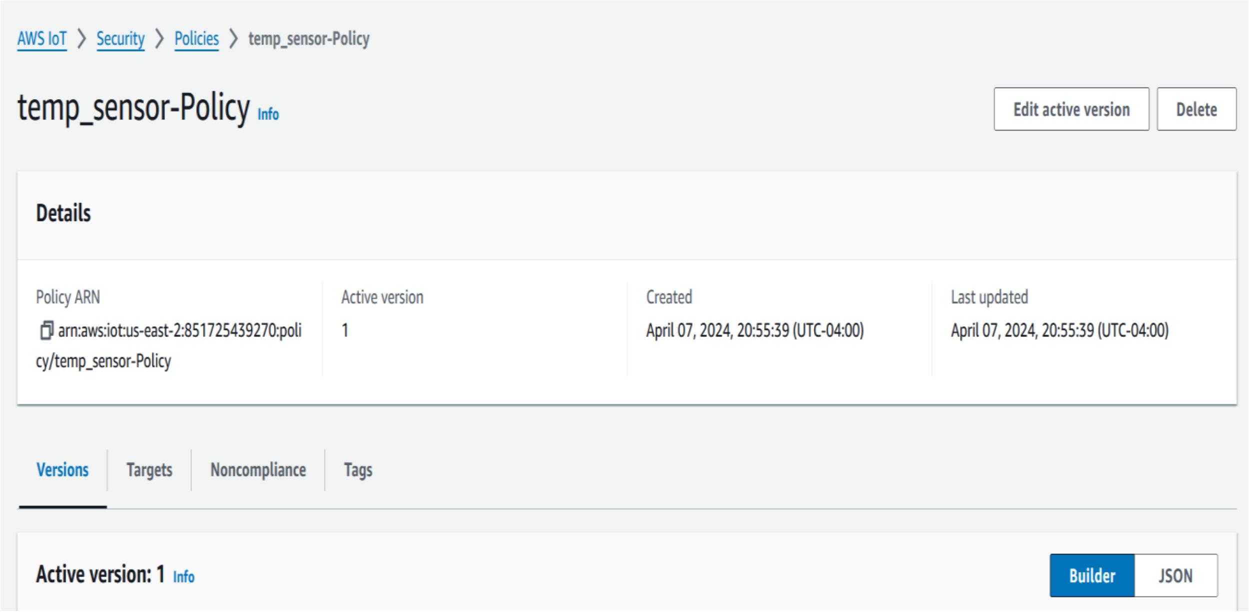
Policy version status

Active policy

☑ Set the edited version as the active version for this policy

You can change this setting later in the policy's detail page.

We selected the new created policy to verify that it was created properly:



temp_sensor-Policy

Details

Policy ARN: arn:aws:iot:us-east-2:851725439270:policy/temp_sensor-Policy

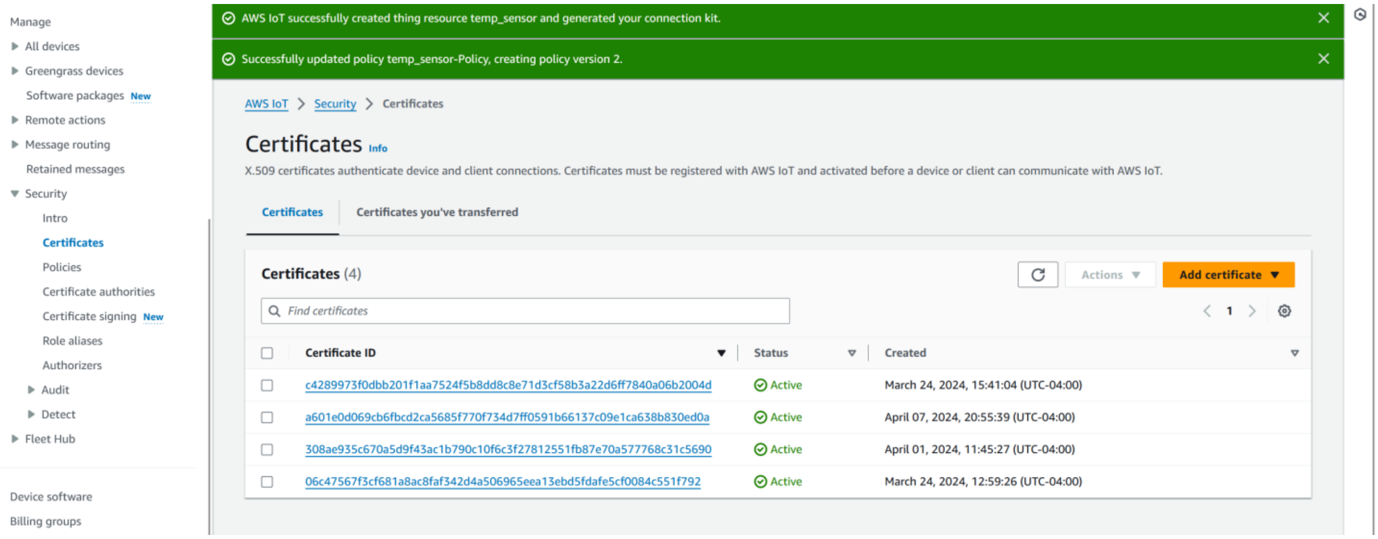
Active version: 1

Created: April 07, 2024, 20:55:39 (UTC-04:00)

Last updated: April 07, 2024, 20:55:39 (UTC-04:00)

Active version: 1

11. Once the policy was created, we then attached the newly created policy to our certificate. To do this, we went to “Certificates” under “Security” in the AWS IoT Core page.



AWS IoT successfully created thing resource temp_sensor and generated your connection kit.

Successfully updated policy temp_sensor-Policy, creating policy version 2.

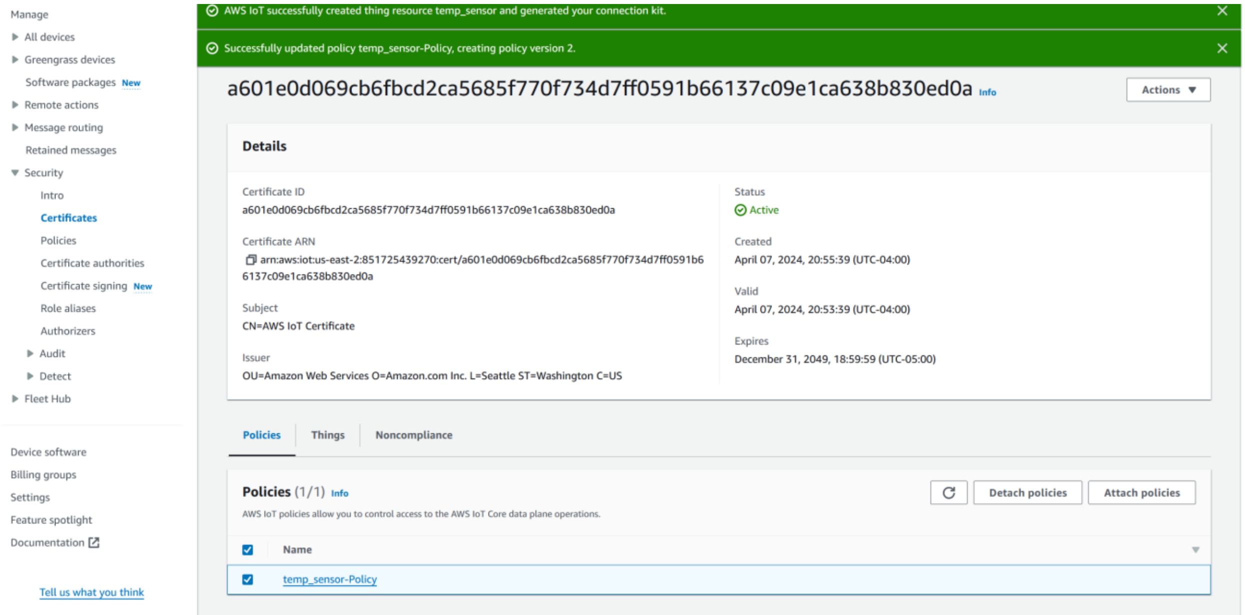
Certificates Info

X.509 certificates authenticate device and client connections. Certificates must be registered with AWS IoT and activated before a device or client can communicate with AWS IoT.

Certificates (4)

Certificate ID	Status	Created
c4289973f0dbb201f1aa7524f5b8dd8c8e71d3cf58b3a22d6ff7840a06b2004d	Active	March 24, 2024, 15:41:04 (UTC-04:00)
a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a	Active	April 07, 2024, 20:55:39 (UTC-04:00)
308ae935c670a5d9f43ac1b790c10f6c3f27812551fb87e70a577768c31c5690	Active	April 01, 2024, 11:45:27 (UTC-04:00)
06c47567f3cf681a8ac8faf342d4a506965eea13ebd5fdafe5cf0084c551f792	Active	March 24, 2024, 12:59:26 (UTC-04:00)

We clicked the certificate that we created and clicked on “Attach policies”.



AWS IoT successfully created thing resource temp_sensor and generated your connection kit.

Successfully updated policy temp_sensor-Policy, creating policy version 2.

a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a Info

Details

Certificate ID: a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

Certificate ARN: arn:aws:iot:us-east-2:851725439270:cert/a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

Subject: CN=AWS IoT Certificate

Issuer: OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US

Status: Active

Created: April 07, 2024, 20:55:39 (UTC-04:00)

Valid: April 07, 2024, 20:53:39 (UTC-04:00)

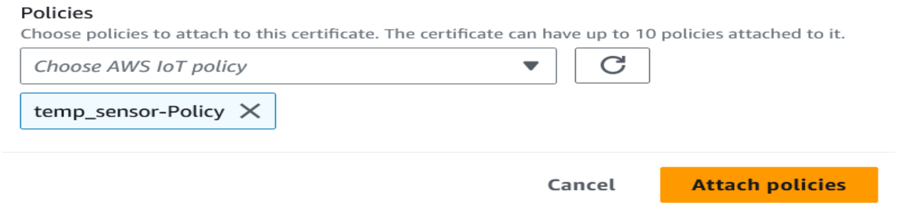
Expires: December 31, 2049, 18:59:59 (UTC-05:00)

Policies (1/1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

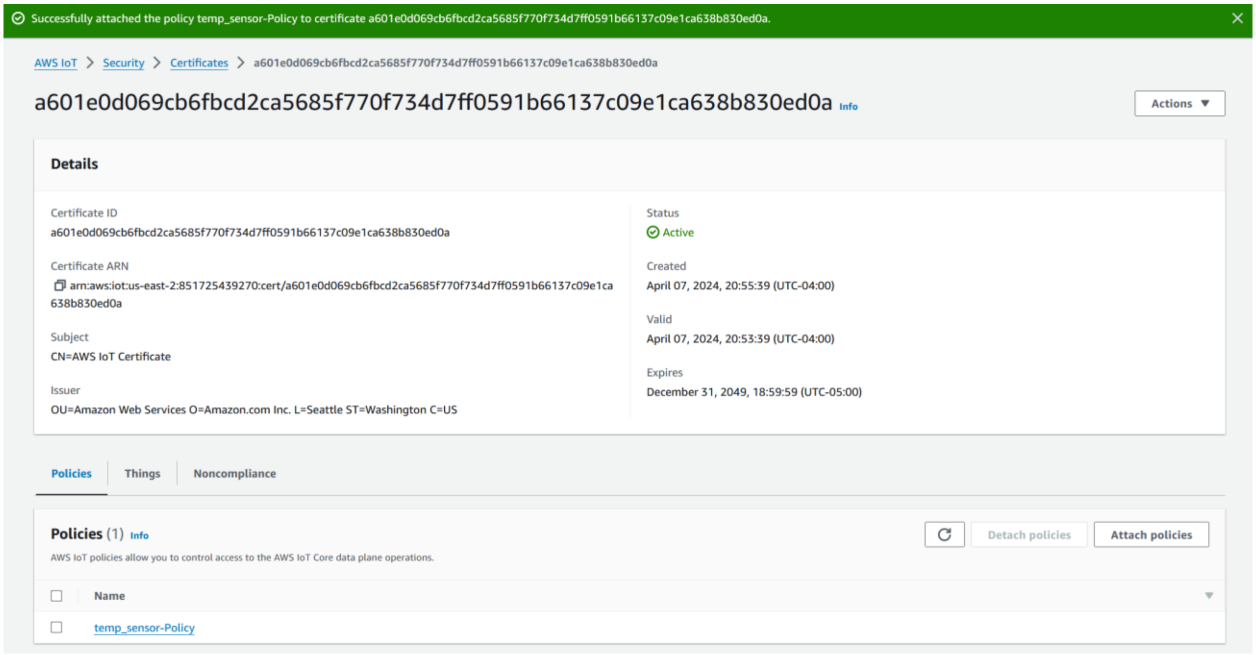
☑	Name
☑	temp_sensor-Policy

From the “Attach policies to the certificate” we selected our newly created policy “temp_sensor-Policy”. Then click “Attach policies” and now the AWS IoT Thing is ready to receive messages from the Raspberry Pi



Policies

Choose policies to attach to this certificate. The certificate can have up to 10 policies attached to it.



Successfully attached the policy temp_sensor-Policy to certificate a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a.

AWS IoT > Security > Certificates > a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

Details

Certificate ID
a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

Certificate ARN
arn:aws:iot:us-east-2:851725439270:cert/a601e0d069cb6fbcd2ca5685f770f734d7ff0591b66137c09e1ca638b830ed0a

Subject
CN=AWS IoT Certificate

Issuer
OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US

Status
Active

Created
April 07, 2024, 20:55:39 (UTC-04:00)

Valid
April 07, 2024, 20:53:39 (UTC-04:00)

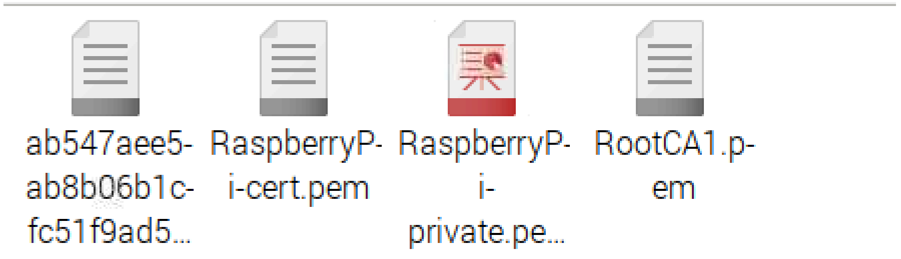
Expires
December 31, 2049, 18:59:59 (UTC-05:00)

Policies (1)

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

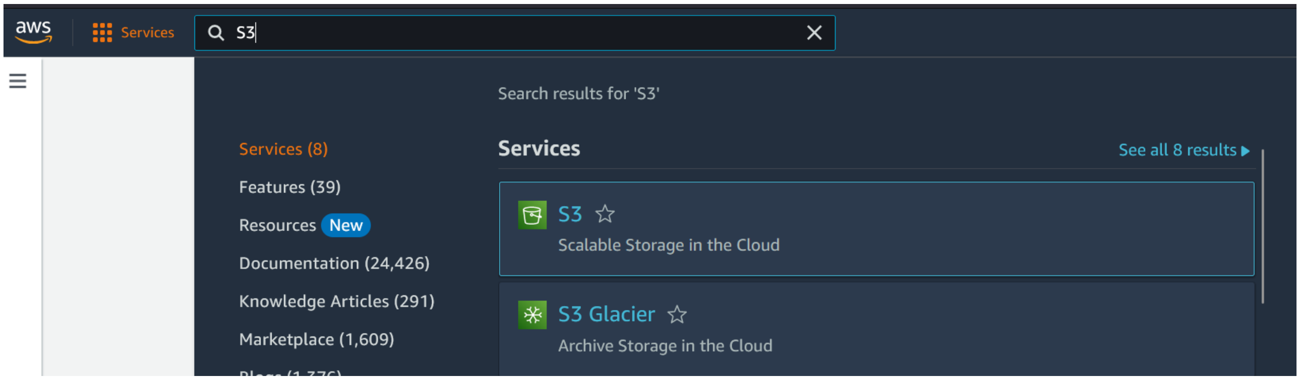
□	Name
□	temp_sensor-Policy

12. We created a new directory called “cert”. The full path is “/home/andrew/CPE_4040/cert”.
    Transfers the certificates, public key, private key, and root CA1 to the Raspberry Pi. Then rename the device certificate to “RaspberryPi-cert.pem”, private key to “RaspberryPi-private.pem.key”, and root CA to “RootCA1.pem”.



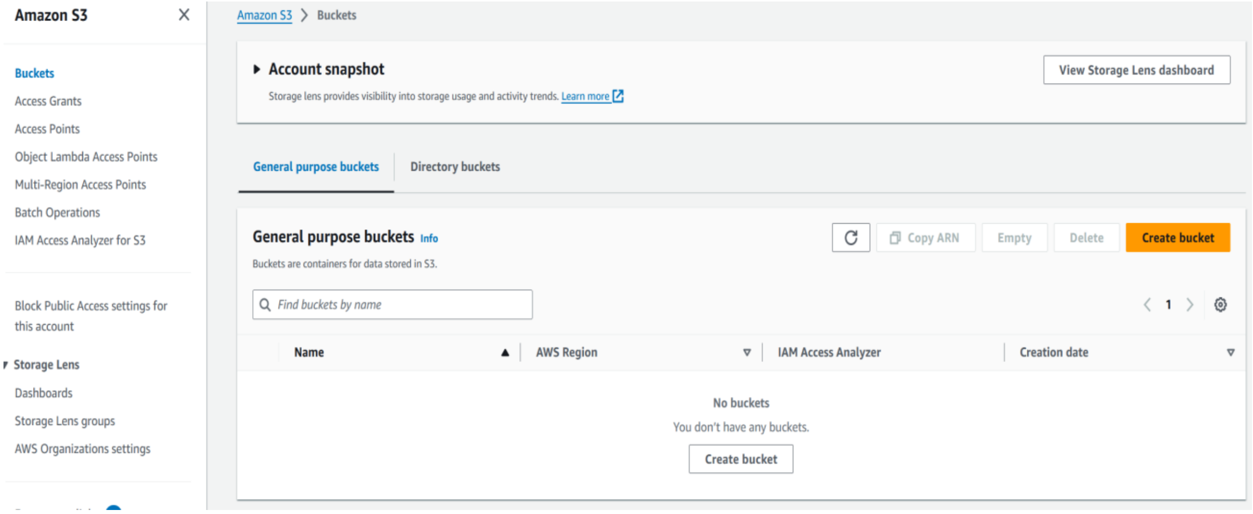
Section 3 setting up the AWS S3 bucket:

13. Search for S3 in the search bar of AWS Management Console located at https://console.aws.amazon.com/



Search results for 'S3'

14. Once we were inside the S3 web page, we clicked on create bucket.



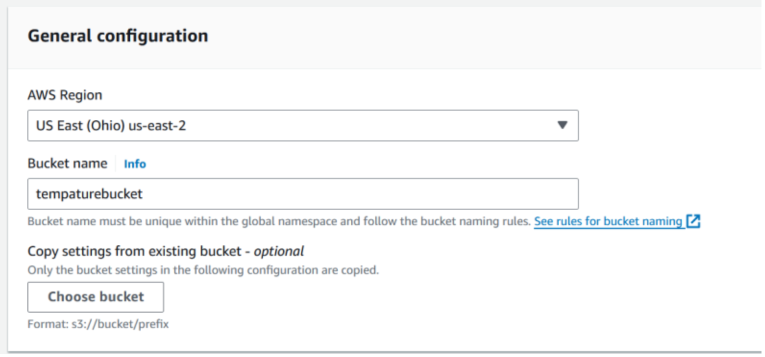
Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets are containers for data stored in S3.

No buckets

You don't have any buckets.

15. We enter the name of the bucket as “tempaturebucket” and then we made sure “ACLs enabled” was selected under “Object Ownership”.



General configuration

AWS Region

US East (Ohio) us-east-2 ▼

Bucket name Info

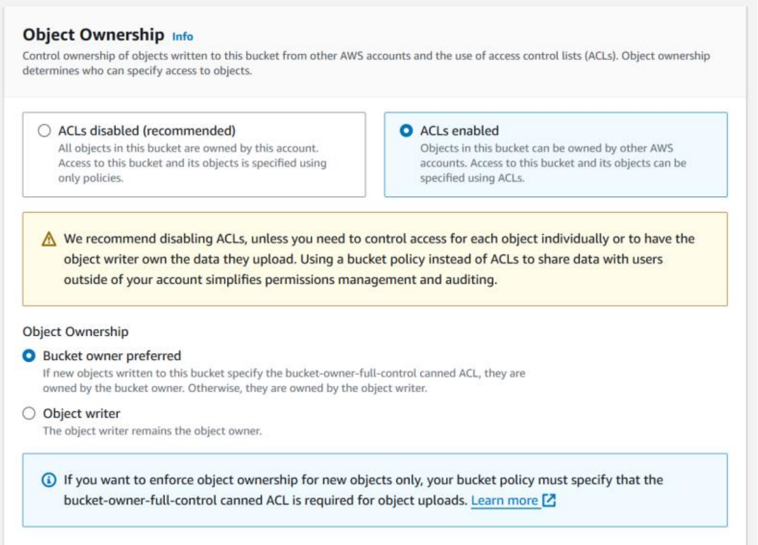
tempaturebucket

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Format: s3://bucket/prefix



Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

○ ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

● ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

⚠ We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

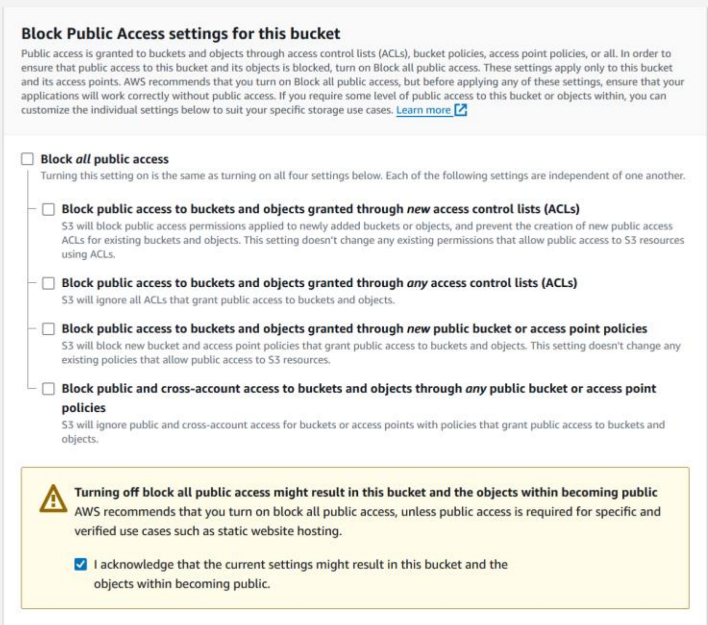
Object Ownership

● Bucket owner preferred
If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

○ Object writer
The object writer remains the object owner.

ⓘ If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. Learn more

We also unchecked the “Block all public access” box under “Block Public Access settings for this bucket” and clicked the acknowledgement button that the bucket will be public.



Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

□ **Block *all* public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

□ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

□ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.

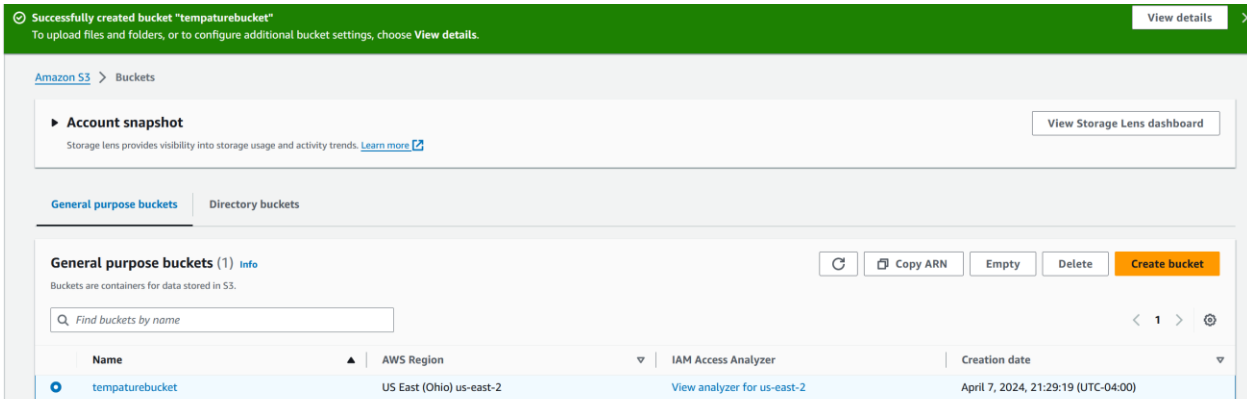
□ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

□ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

⚠ **Turning off block all public access might result in this bucket and the objects within becoming public**
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☑ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

After we configured the bucket, we then created it by clicking the “Create bucket”.

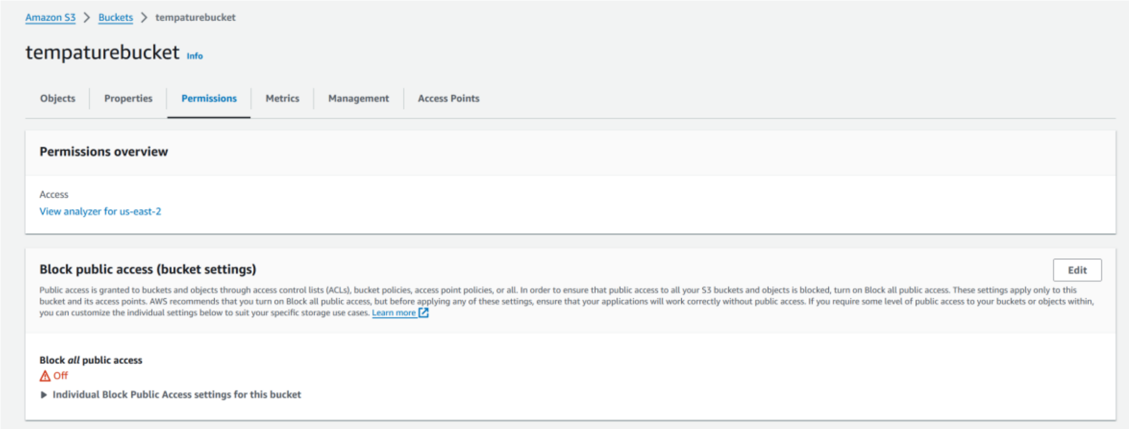


Successfully created bucket "tempaturebucket"
To upload files and folders, or to configure additional bucket settings, choose **View details**.

View details
Amazon S3 > Buckets
Account snapshot
Storage lens provides visibility into storage usage and activity trends. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name

Name	AWS Region	IAM Access Analyzer	Creation date
tempaturebucket	US East (Ohio) us-east-2	View analyzer for us-east-2	April 7, 2024, 21:29:19 (UTC-04:00)

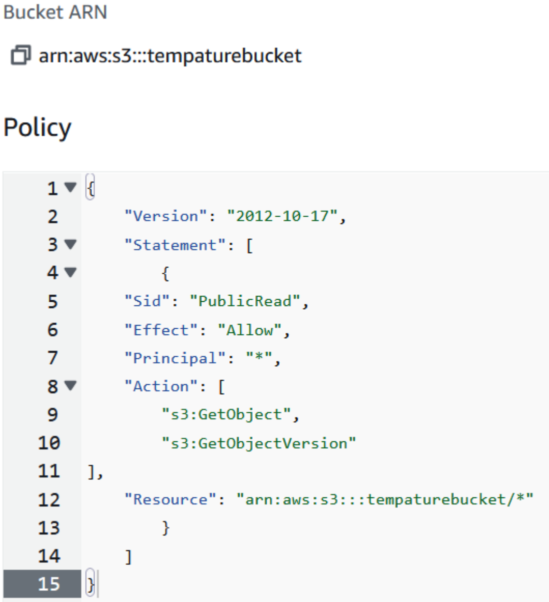
16. Next, we clicked on the newly created bucket called “tempaturebucket” and clicked on the “Permission” tab.



Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

Block *all* public access
⚠ Off

We then clicked on the “Edit” button under “Bucket policy” and entered the given code into the policy text box. We then changed the Resource line to match our bucket ARN.



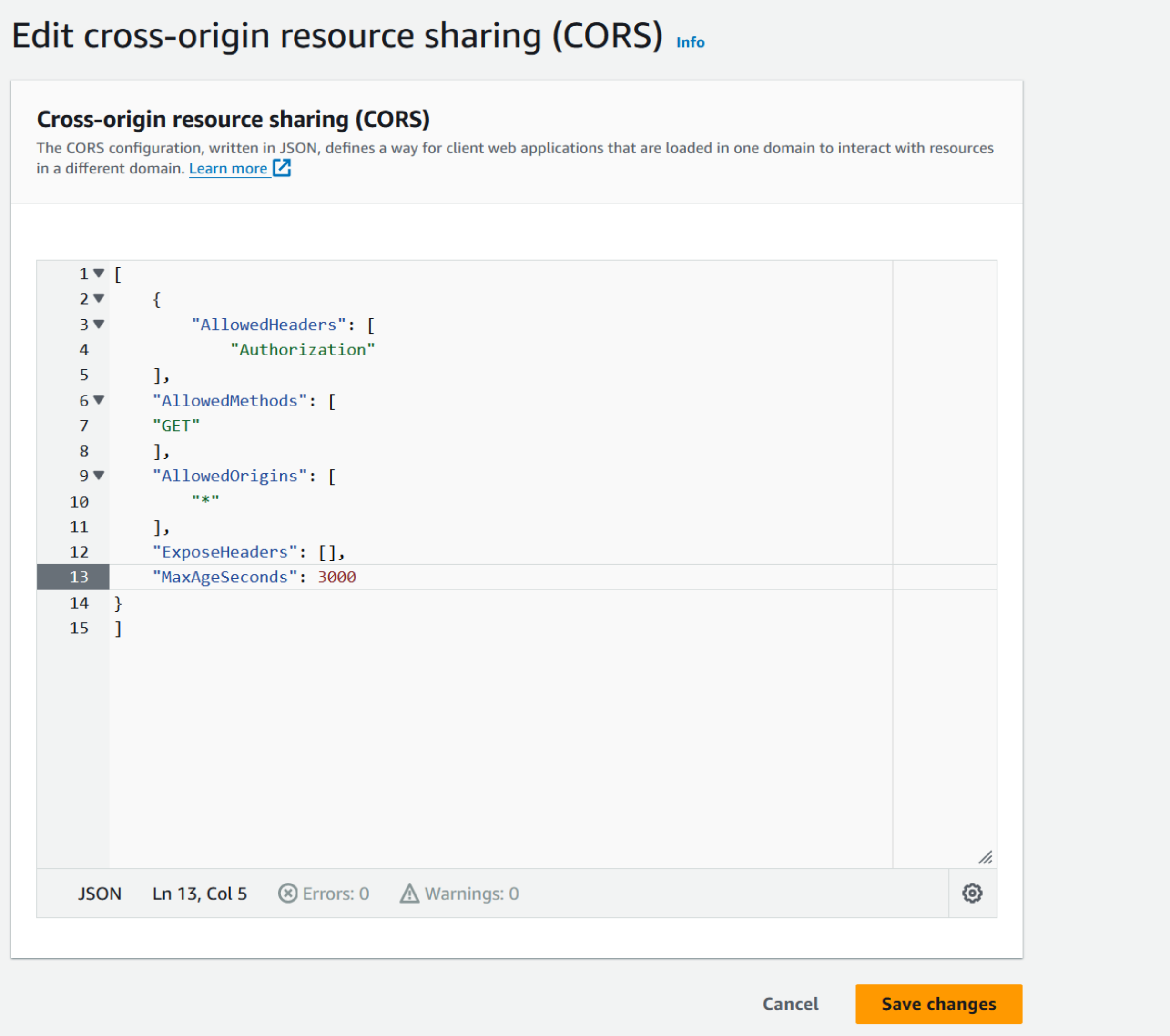
Bucket ARN

arn:aws:s3:::tempaturebucket

Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
    "Sid": "PublicRead",
    "Effect": "Allow",
    "Principal": "*",
    "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
],
    "Resource": "arn:aws:s3:::tempaturebucket/*"
        }
    ]
}
```

17. On the same page, edit the CORS section and paste the given cod in the text box and then click save changes.



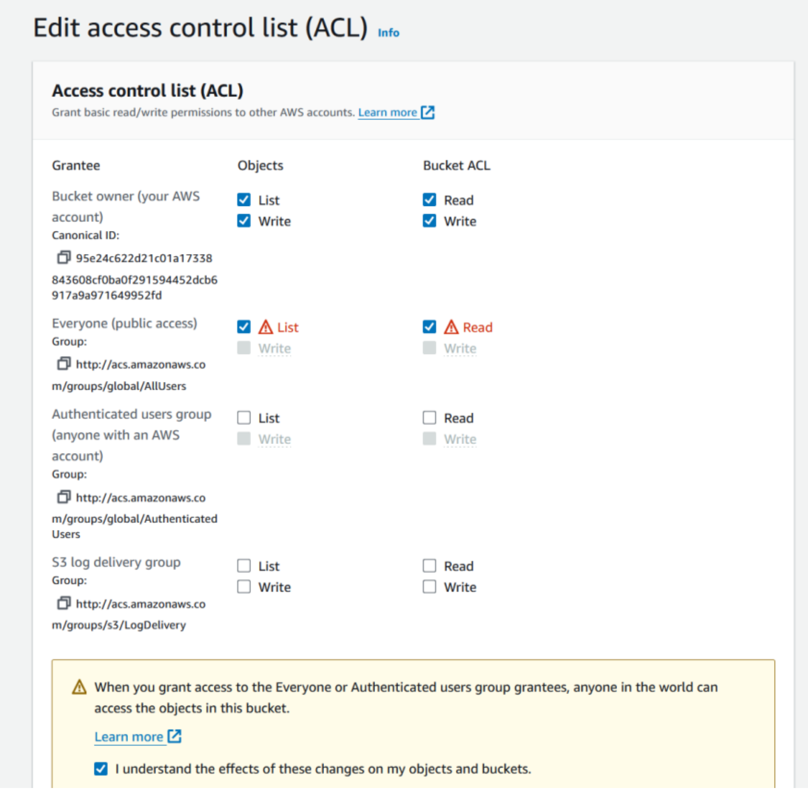
Edit cross-origin resource sharing (CORS) Info

Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. Learn more

```
[
    {
        "AllowedHeaders": [
            "Authorization"
    ],
    "AllowedMethods": [
    "GET"
    ],
    "AllowedOrigins": [
        "*"
    ],
    "ExposeHeaders": [],
    "MaxAgeSeconds": 3000
}
]
```

18. We then clicked "edit" under the "Access Control List" and checked the "List" and "Read" permissions under "Everyone (Public Access)". To finish it off, we checked the acknowledgment button that the bucket is public and saved the changes.



Edit access control list (ACL)

Access control list (ACL)

Grant basic read/write permissions to other AWS accounts. Learn more

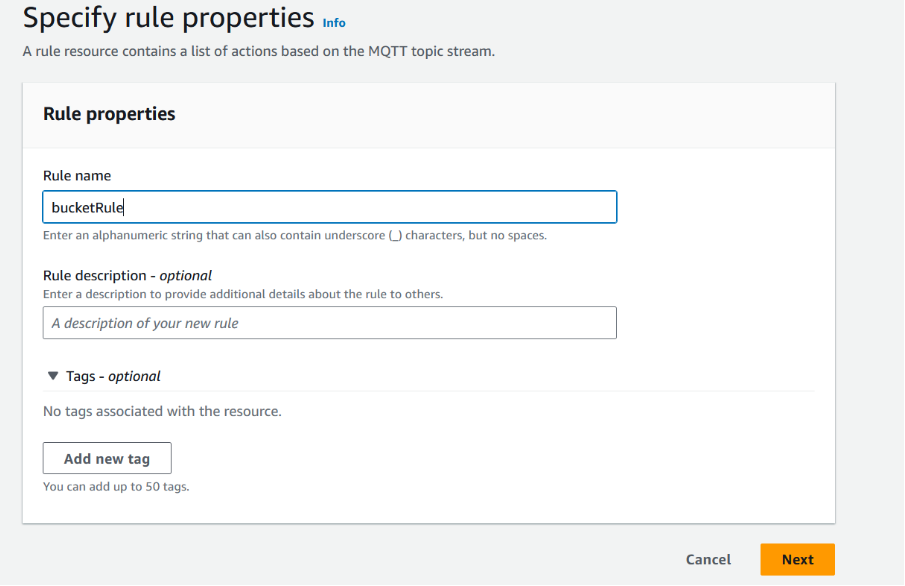
Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID: 95e24c622d21c01a17338 843608cf0ba0f291594452dcb6 917a9a971649952fd	☑ List ☑ Write	☑ Read ☑ Write
Everyone (public access) Group: http://acs.amazonaws.com/groups/global/AllUsers	☑ ⚠ List □ Write	☑ ⚠ Read □ Write
Authenticated users group (anyone with an AWS account) Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers	□ List □ Write	□ Read □ Write
S3 log delivery group Group: http://acs.amazonaws.com/groups/s3/LogDelivery	□ List □ Write	□ Read □ Write

⚠ When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access the objects in this bucket.

Learn more

☑ I understand the effects of these changes on my objects and buckets.

19. Next, we navigated to IoT Core and clicked "Rules" under "Message Routing". We then clicked "Create rule" and named it "bucketRule".



Specify rule properties

A rule resource contains a list of actions based on the MQTT topic stream.

Rule properties

Rule name

bucketRule

Enter an alphanumeric string that can also contain underscore (_) characters, but no spaces.

Rule description - *optional*

Enter a description to provide additional details about the rule to others.

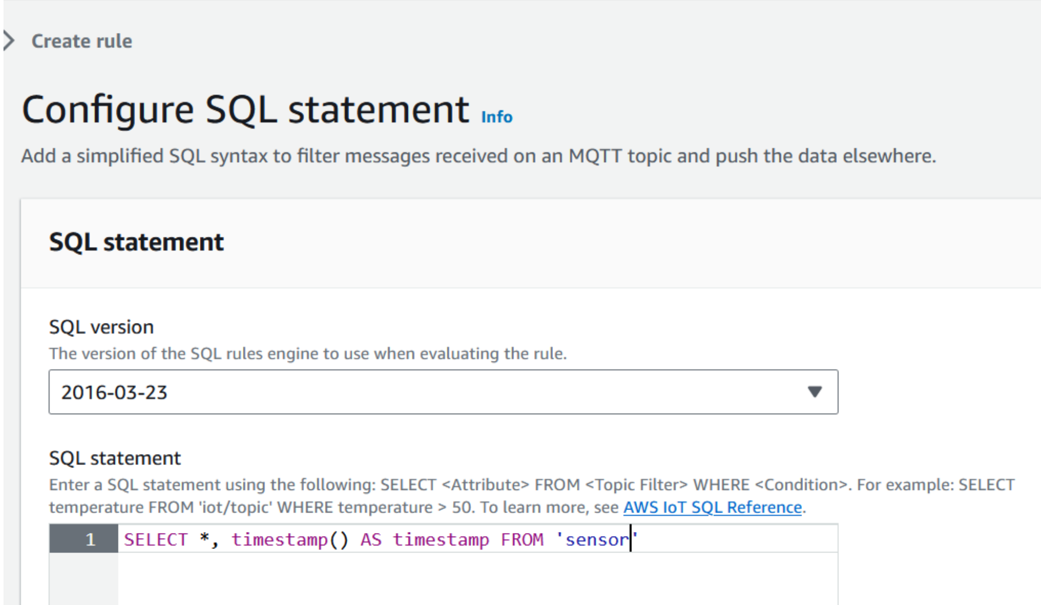
A description of your new rule

▼ Tags - *optional*

No tags associated with the resource.

You can add up to 50 tags.

20. In the “SQL Statement” of the bucket rule, we entered the following code to add a timestamp to our data that is sent from NodeRED. We then set our topic name to “sensor”.



Configure SQL statement

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

SQL statement

SQL version

The version of the SQL rules engine to use when evaluating the rule.

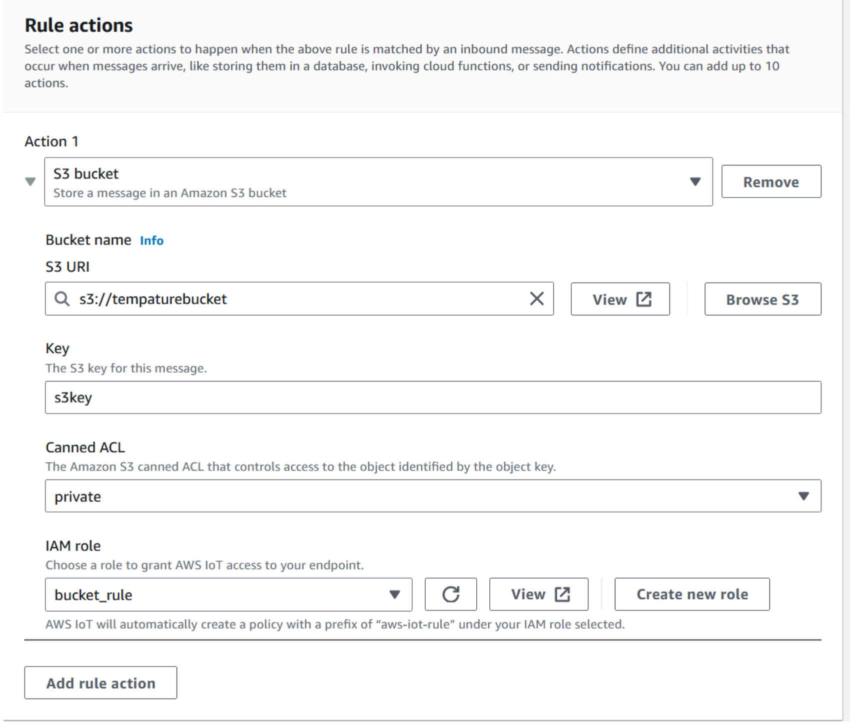
2016-03-23

SQL statement

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

```
SELECT *, timestamp() AS timestamp FROM 'sensor'
```

21. On the next page, we selected “S3 bucket” under the “Rule actions”. We then clicked the “Browse S3” button and selected our bucket name “tempaturebucket”. Next, we entered “s3key” under “Key” section and created a new “IAM Role” called “bucket_rule”.



Rule actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. You can add up to 10 actions.

Action 1

S3 bucket
Store a message in an Amazon S3 bucket

Bucket name

S3 URI

s3://tempaturebucket

Key

The S3 key for this message.

s3key

Canned ACL

The Amazon S3 canned ACL that controls access to the object identified by the object key.

private

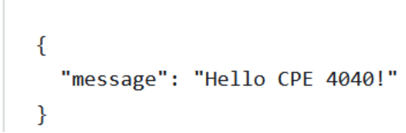
IAM role

Choose a role to grant AWS IoT access to your endpoint.

bucket_rule

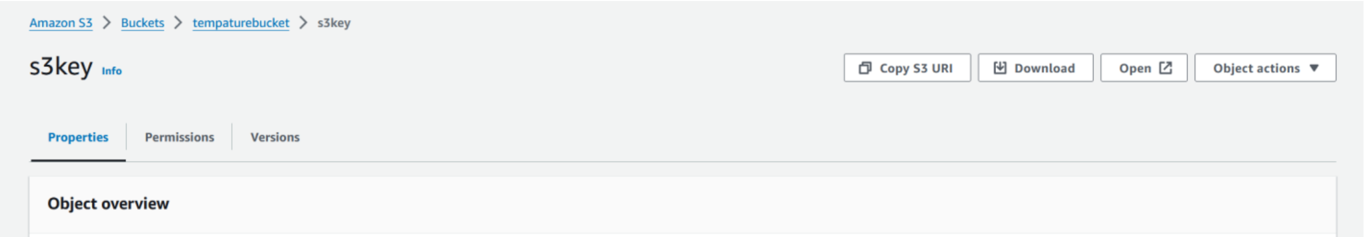
AWS IoT will automatically create a policy with a prefix of “aws-iot-rule” under your IAM role selected.

22. Once we finished creating the bucket rule, we then set up the MQTT client. We did this by selecting the “Test” under the “MQTT test client” AWS IoT menu. We then clicked “Publish to a topic” and entered our topic called “sensor”. We entered a custom message containing “Hello CPE 4040!” and published it.

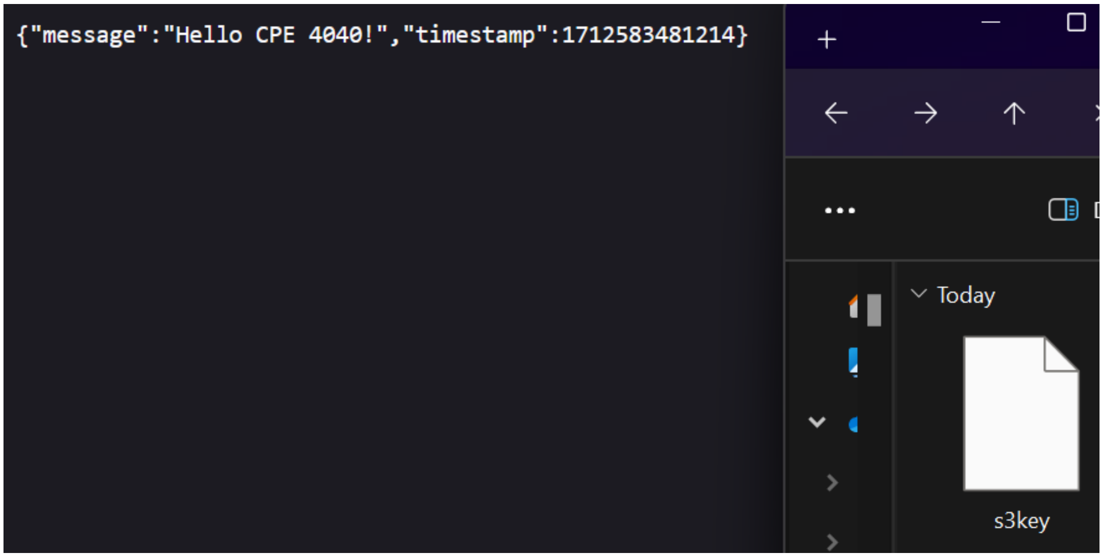


```
{
  "message": "Hello CPE 4040!"
}
```

23. Next, we went to AWS S3 web page in a different tab and clicked on our bucket called “tempaturebucket”. We then clicked on the “s3key” under “Objects” and clicked on the download button in the upper right corner.



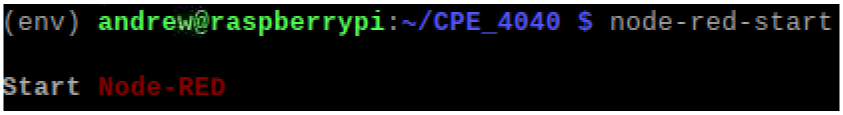
This will download a file which we opened in a web page that shows our message with a timestamp attached.



```
{"message":"Hello CPE 4040!","timestamp":1712583481214}
```

Section 4 Set up NodeRED Flow to send data from the sensor to the S3 bucket using MQTT:

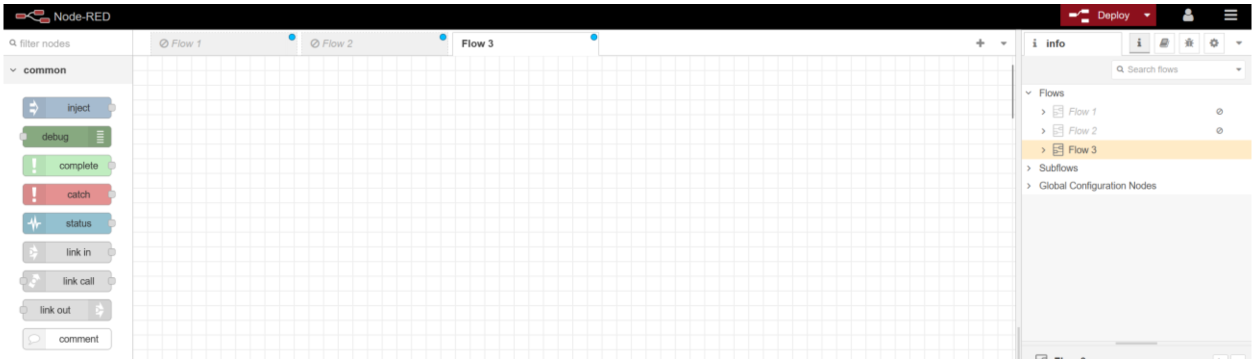
24. First, we started Node-RED by typing the following command in the terminal:



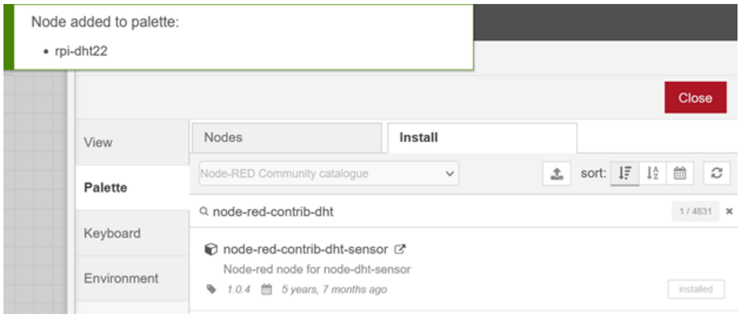
```
(env) andrew@raspberrypi:~/CPE_4040 $ node-red-start

Start Node-RED
```

In a new tab, we entered our Raspberry Pi's IP address and port 1880 to access the development tool.



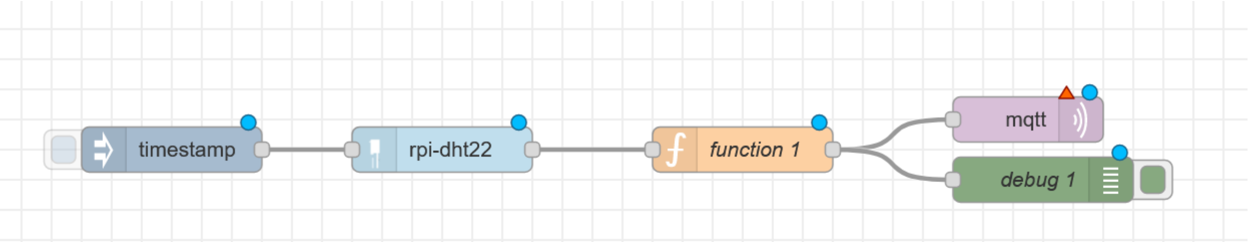
25. We installed the “node-red-contrib-dht-sensor” package for our DHT-11 sensor in the Palette Manager.



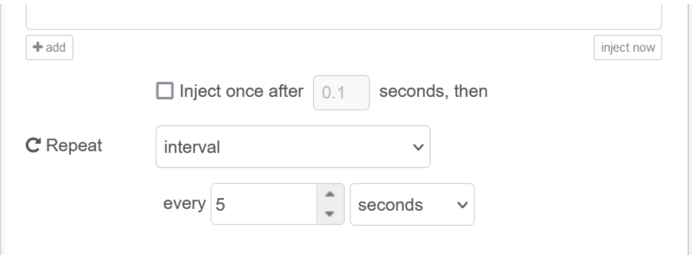
Node added to palette:
- rpi-dht22

node-red-contrib-dht-sensor
Node-red node for node-dht-sensor
1.0.4 5 years, 7 months ago installed

26. Next, we create a flow block with an injection node, rpi-dht22 node, function node, mqtt out node, and debug node.



27. We first edit the injection node to activate every 5 seconds by selecting “interval” next to “Repeat” and entered 5 seconds.

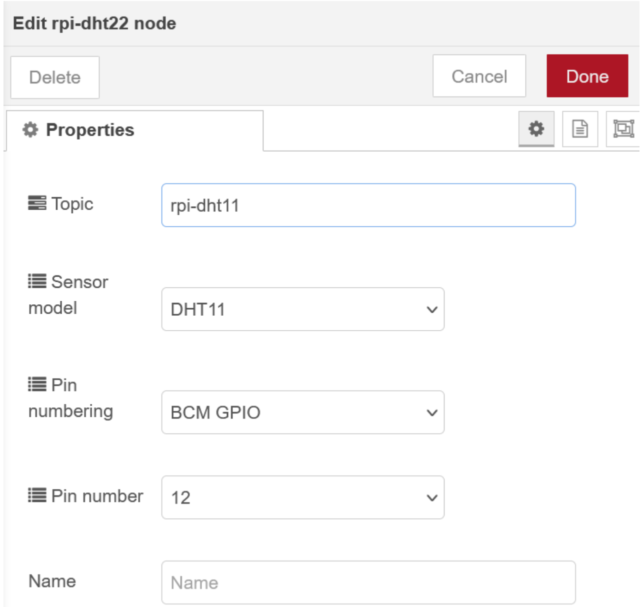


☐ Inject once after 0.1 seconds, then

Repeat: interval

every 5 seconds

We then edited the rpi-dht node by changing the name to rpi-dht11, setting the “Sensor model” to DHT11, and the GPIO pin to 12.



Edit rpi-dht22 node

Topic: rpi-dht11

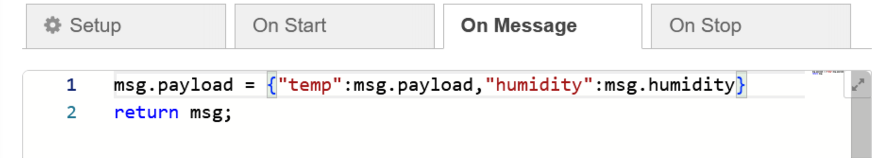
Sensor model: DHT11

Pin numbering: BCM GPIO

Pin number: 12

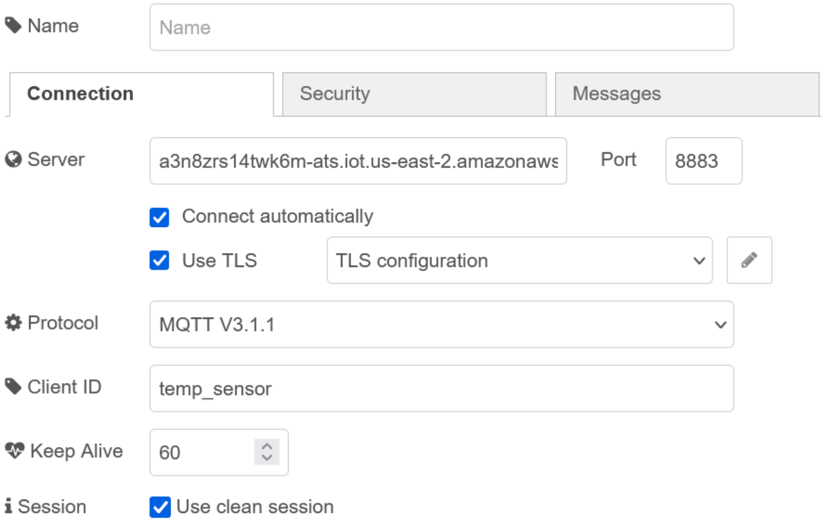
Name: Name

Next, we edit the function node by setting up the “On Message” to:



```
msg.payload = {"temp":msg.payload,"humidity":msg.humidity}
return msg;
```

28. We then found the AWS IoT URL by navigating to the AWS IoT webpage and accessing the settings. After copying our endpoint URL, we pasted it into the MQTT node by first clicking on the button next to the server to create a new MQTT broker. We changed the port to 8883 and configured the "Client ID" to the AWS IoT Thing we created earlier called "temp_sensor".



Name: Name

Server: a3n8zrs14twk6m-ats.iot.us-east-2.amazonaws Port: 8883

☑ Connect automatically

☑ Use TLS: TLS configuration

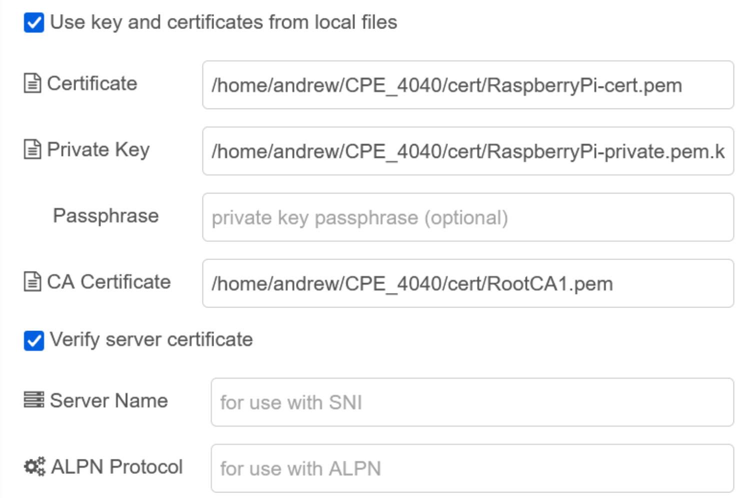
Protocol: MQTT V3.1.1

Client ID: temp_sensor

Keep Alive: 60

Session: ☑ Use clean session

29. Next, we edited the TLS by clicking the pencil button next to "TLS configuration". We first checked the "Use key and certificates from local files" box and entered the certificate, private key, and CA certificate file path on the Pi.



☑ Use key and certificates from local files

Certificate: /home/andrew/CPE_4040/cert/RaspberryPi-cert.pem

Private Key: /home/andrew/CPE_4040/cert/RaspberryPi-private.pem.k

Passphrase: private key passphrase (optional)

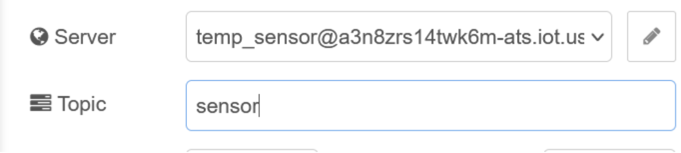
CA Certificate: /home/andrew/CPE_4040/cert/RootCA1.pem

☑ Verify server certificate

Server Name: for use with SNI

ALPN Protocol: for use with ALPN

We then saved the changes and went to the main page of the MQTT node to enter the topic field as "sensor".

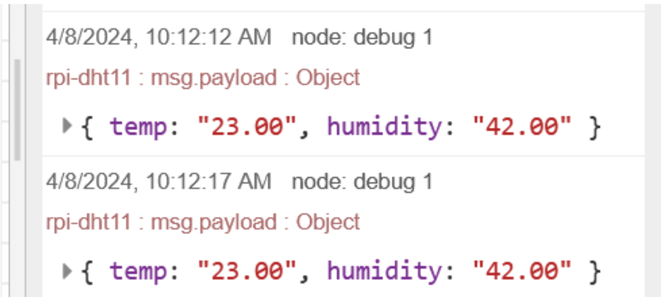


Server: temp_sensor@a3n8zrs14twk6m-ats.iot.us

Topic: sensor

30. Once the all the nodes have been configured properly, we deployed the flow and started seeing messages popup on the debugger.

Successfully deployed

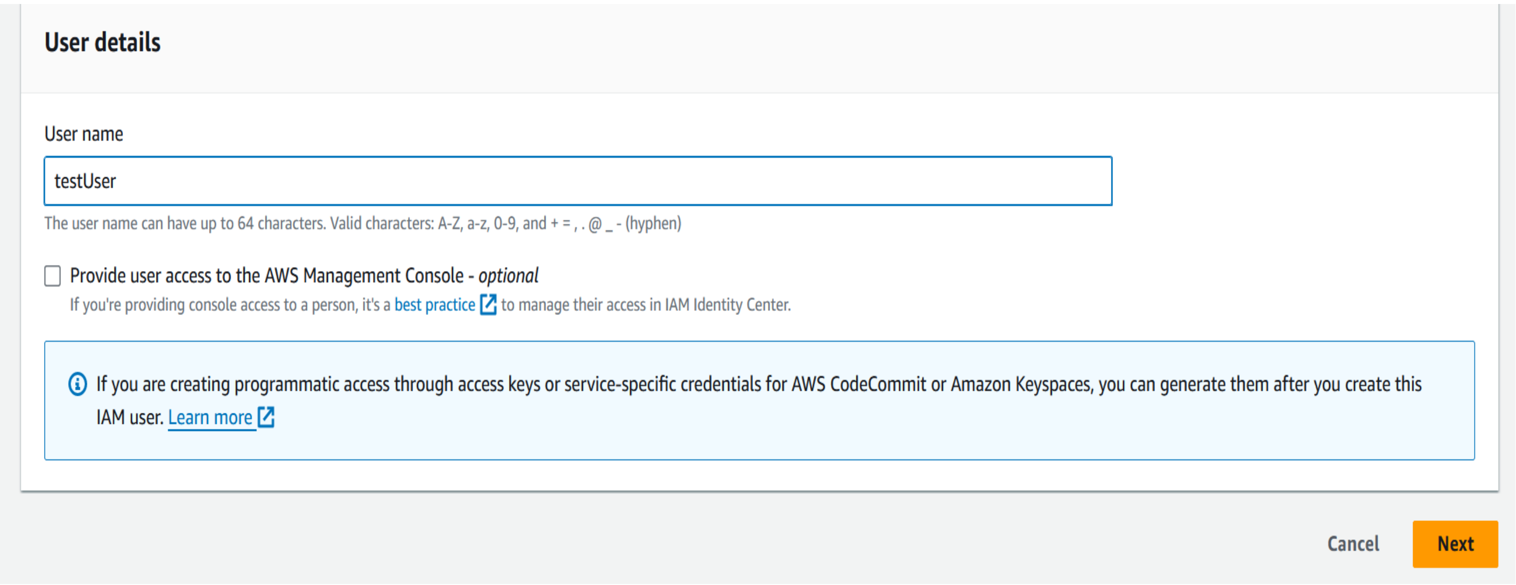


```
4/8/2024, 10:12:12 AM   node: debug 1
rpi-dht11 : msg.payload : Object
 ▸{ temp: "23.00", humidity: "42.00" }
4/8/2024, 10:12:17 AM   node: debug 1
rpi-dht11 : msg.payload : Object
 ▸{ temp: "23.00", humidity: "42.00" }
```

Section 5 sending data from S3 bucket to Streamlit:

31. Go to AWS Management Console at https://console.aws.amazon.com/ and search for “IAM” in the search bar. Click on “Users” under “Access management” and click “Create user”. We named our user “testUser” and clicked “Next”.

32.



User details

User name

testUser

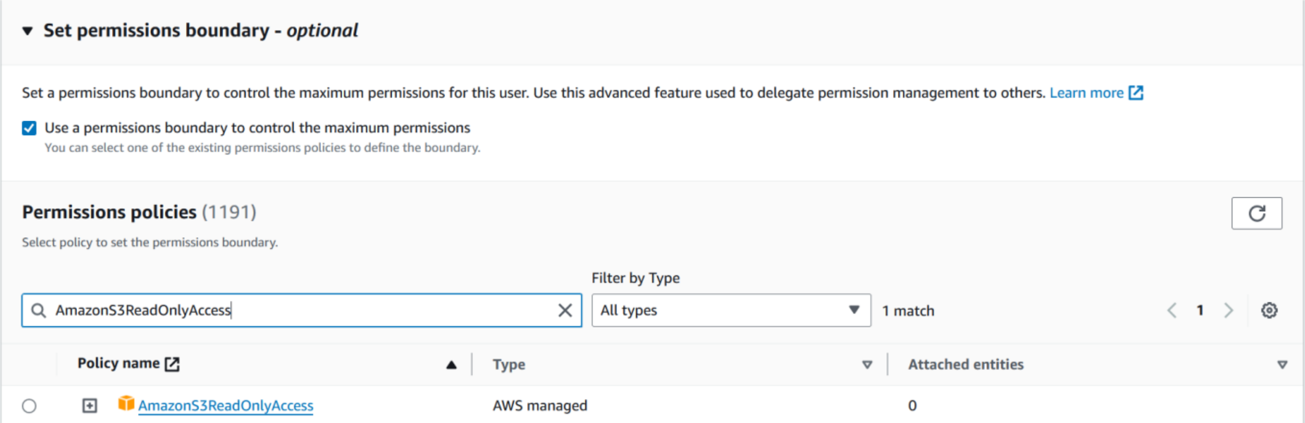
The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

□ Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Next, we checked the box to “Use a permission boundary” and searched for “AmazonS3ReadOnlyAccess” in the filter box. We then clicked “Next” and created the user by pressing “Create user”.



▼ **Set permissions boundary - *optional***

Set a permissions boundary to control the maximum permissions for this user. Use this advanced feature used to delegate permission management to others. Learn more

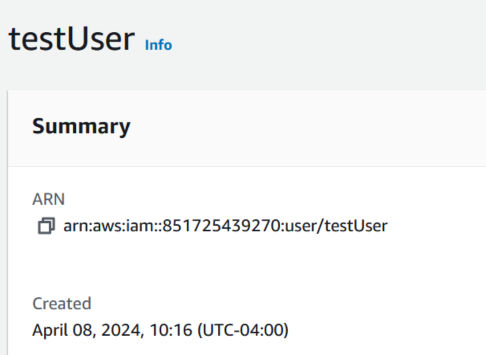
☑ Use a permissions boundary to control the maximum permissions
You can select one of the existing permissions policies to define the boundary.

Permissions policies (1191)
Select policy to set the permissions boundary.

Filter by Type

AmazonS3ReadOnlyAccess | All types | 1 match

	Policy name	Type	Attached entities
○	AmazonS3ReadOnlyAccess	AWS managed	0



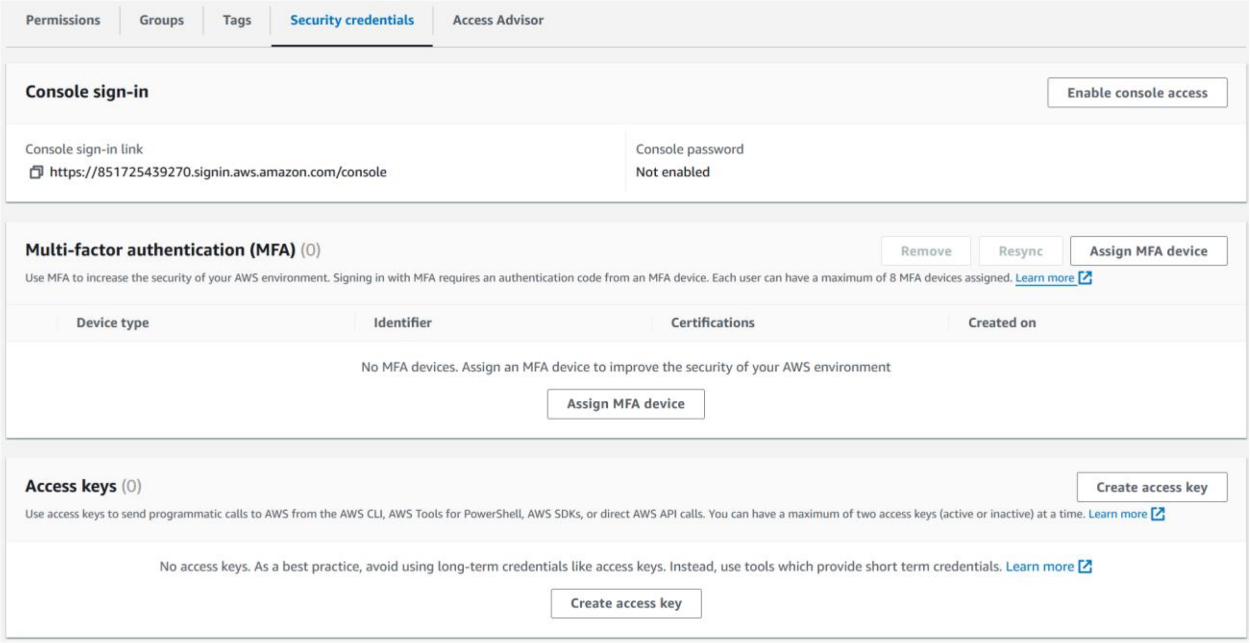
testUser Info

Summary

ARN
arn:aws:iam::851725439270:user/testUser

Created
April 08, 2024, 10:16 (UTC-04:00)

33. After the user was created, we then selected the user and edit the “Access Key” under “Security credentials”.



Console sign-in

Console sign-in link
https://851725439270.signin.aws.amazon.com/console

Console password
Not enabled

Multi-factor authentication (MFA) (0)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Device type	Identifier	Certifications	Created on

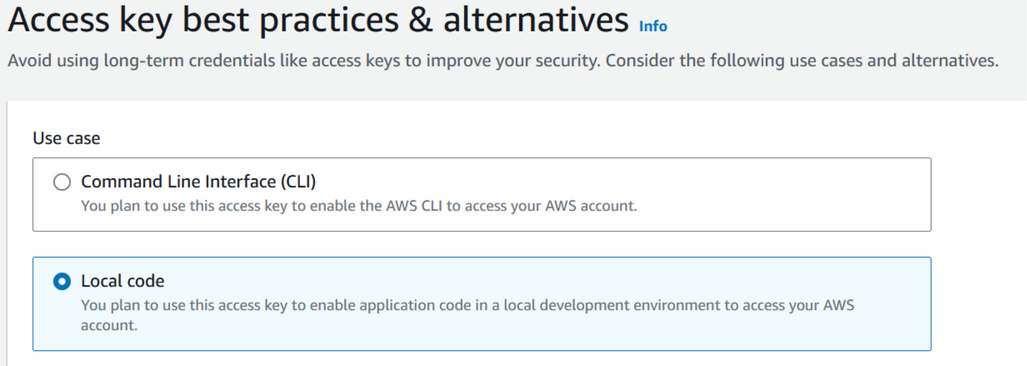
No MFA devices. Assign an MFA device to improve the security of your AWS environment

Access keys (0)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more

No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. Learn more

34. In the “Access key best practices & alternatives” we choose the “Local code” and then created the access key.



Access key best practices & alternatives Info

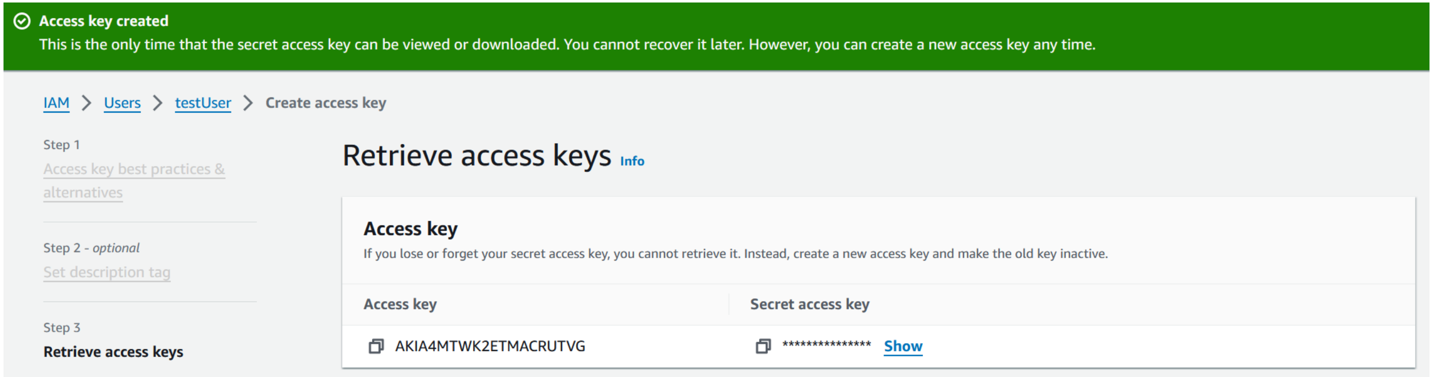
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

○ Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

● Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

35. We then copied the “Access Key” and “Secret access key” into a file for safe keeping.



Access key created
This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

IAM > Users > testUser > Create access key

Step 1
Access key best practices & alternatives

Step 2 - *optional*
Set description tag

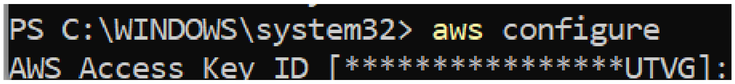
Step 3
Retrieve access keys

Retrieve access keys Info

Access key
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

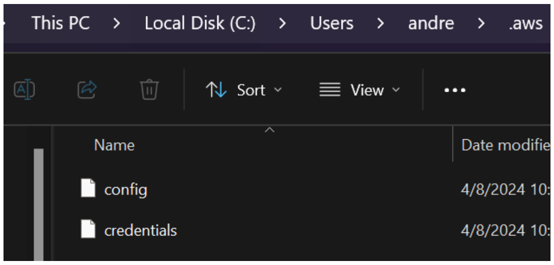
Access key	Secret access key
AKIA4MTWK2ETMACRUTVG	*************** Show

36. We then typed “aws configure” in our Windows power shell and entered our access key and secret access key when prompted. We selected the default options for the rest of the prompts.



```
PS C:\WINDOWS\system32> aws configure
AWS Access Key ID [****************UTVG]:
```

This created a credentials file in C:\Users\andre\.aws:

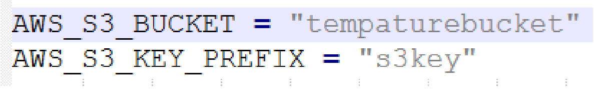


Name	Date modifie
config	4/8/2024 10:
credentials	4/8/2024 10:

37. Next, we download the app.py script given and edited the file with the following changes:

A) Lines 13 and 14:

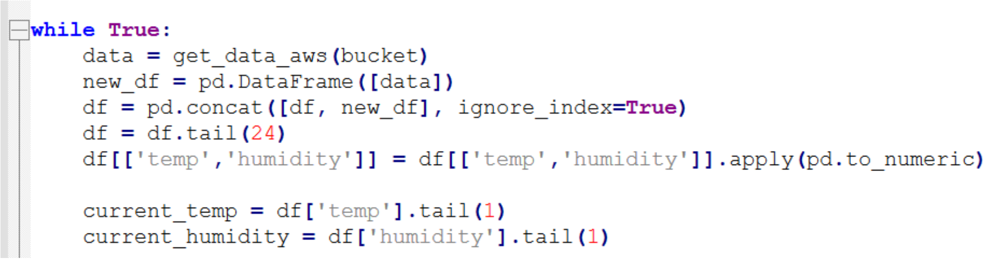
AWS_S3_BUCKET = "tempaturebucket"
AWS_S3_KEY_PREFIX = "s3key"



```
AWS_S3_BUCKET = "tempaturebucket"
AWS_S3_KEY_PREFIX = "s3key"
```

B) Line 44 added a line to convert the temp and humidity data from a string into a integer:

df[['temp','humidity']] = df[['temp','humidity']].apply(pd.to_numeric)



```
while True:
    data = get_data_aws(bucket)
    new_df = pd.DataFrame([data])
    df = pd.concat([df, new_df], ignore_index=True)
    df = df.tail(24)
    df[['temp','humidity']] = df[['temp','humidity']].apply(pd.to_numeric)

    current_temp = df['temp'].tail(1)
    current_humidity = df['humidity'].tail(1)
```

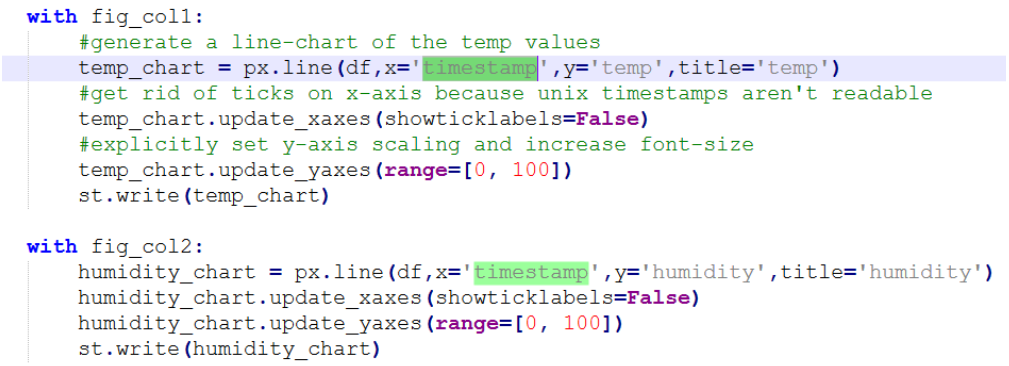
C) Line 70 and 78 changed the “x=timestamps” to “x=timestamp”.

Line 70:

temp_chart = px.line(df,x='timestamp',y='temp',title='temp')

Line 78:

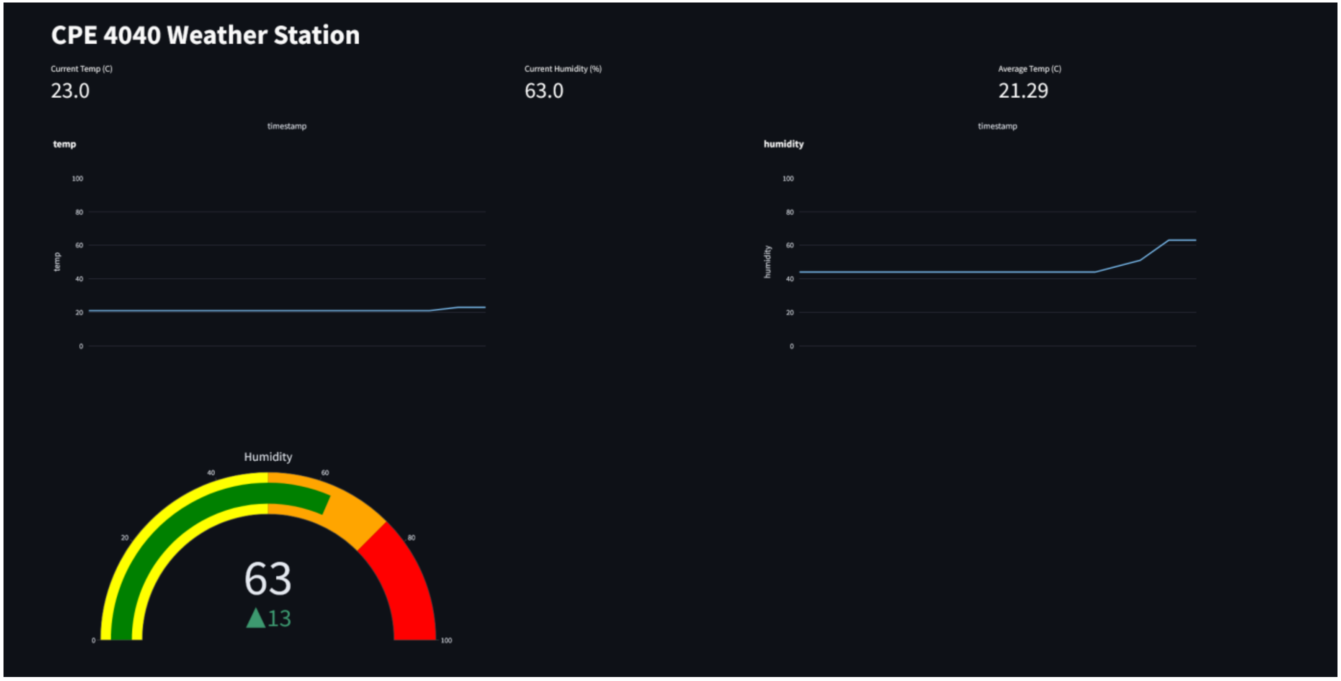
humidity_chart = px.line(df,x='timestamp',y='humidity',title='humidity')



```
with fig_col1:
    #generate a line-chart of the temp values
    temp_chart = px.line(df,x='timestamp',y='temp',title='temp')
    #get rid of ticks on x-axis because unix timestamps aren't readable
    temp_chart.update_xaxes(showticklabels=False)
    #explicitly set y-axis scaling and increase font-size
    temp_chart.update_yaxes(range=[0, 100])
    st.write(temp_chart)

with fig_col2:
    humidity_chart = px.line(df,x='timestamp',y='humidity',title='humidity')
    humidity_chart.update_xaxes(showticklabels=False)
    humidity_chart.update_yaxes(range=[0, 100])
    st.write(humidity_chart)
```

38. Once we configure the script to run properly, we ran the script in the Windows power shell with “streamlit run app.py”. This automatically opened up a new tab on the web browser showing two line charts and a gauge on the dashboard.



CPE 4040 Weather Station

Current Temp (C)
23.0

Current Humidity (%)
63.0

Average Temp (C)
21.29

IV. Conclusion

This lab was a combination of our last few labs. Getting a chance to receive more experience in AWS and node-red was great, especially AWS since we have heard so much about it. The only real issue we had was getting a little confused when using AWS as it has so many settings, but we were able to get through it by going back and redoing steps more carefully. Overall, this was a good lab to revisit some of the more interesting services that we learned this semester. We had a great semester learning about all these services, along with how to collect, clean, and analyze data, and since both of us will be graduating this year, we are sure we will be using the skills we acquired in this class in the near future. Thank you for a great semester Professor Yinn.