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: Anindita Deb and Damisi Kayode

Electrical and Computer Engineering

Kennesaw State University

Faculty: Dr. Jeffrey L Yiin

Date of Lab Session: January 17, 2024

I. Objective

1. To build an end-to-end data project that collects sensor data and creates a web dashboard.
2. To learn how to interface sensor data with AWS IoT using Node-RED on a Raspberry Pi.
3. To learn how to create and store data in the AWS S3 buckets.
4. To develop a Streamlit application that can display the sensor data in a web dashboard.

II. Material List

Hardware Requirement:

Raspberry Pi 4

Temperature sensor (DHT-11, DHT-22, or DHT-20)

Resistors and bread board

Jumper wires (4)

Software Requirement:

Python scripts (available in D2L Assignment)

Remote Desktop Connection

RedNode

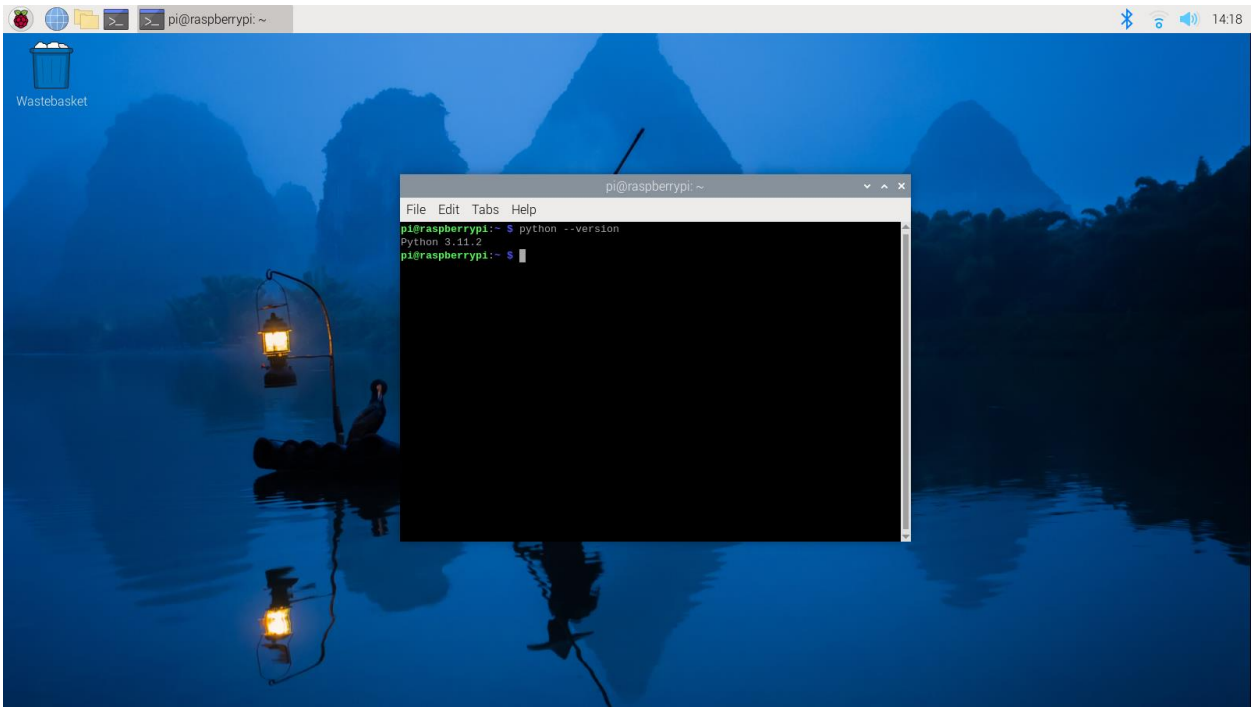
Streamlit

AWS Iot Core

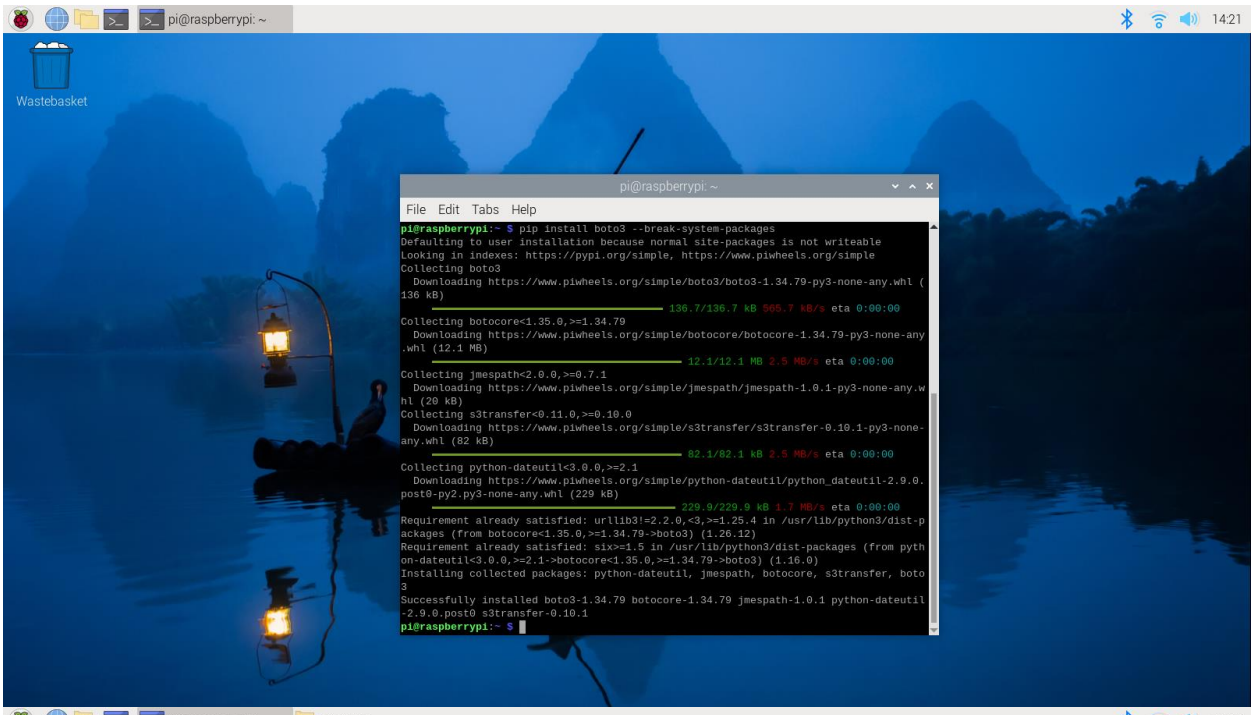
III. Lab Procedures and Results

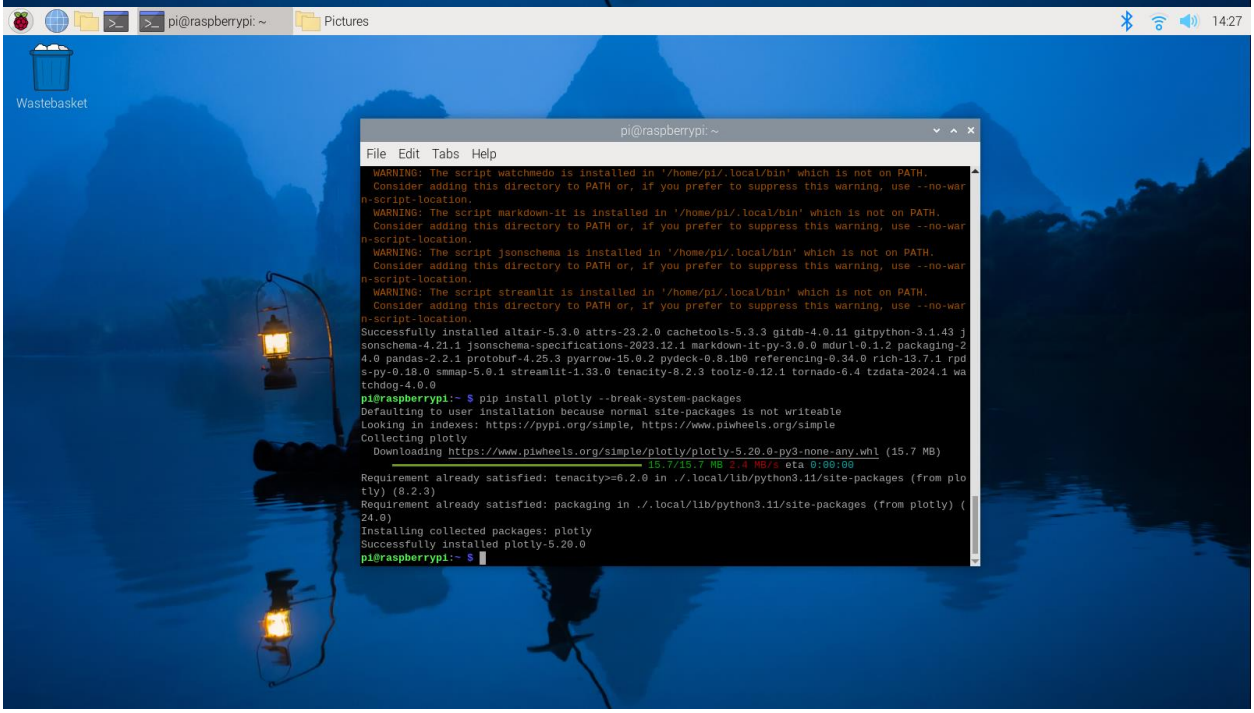
Section 1: Hardware and Software Setup

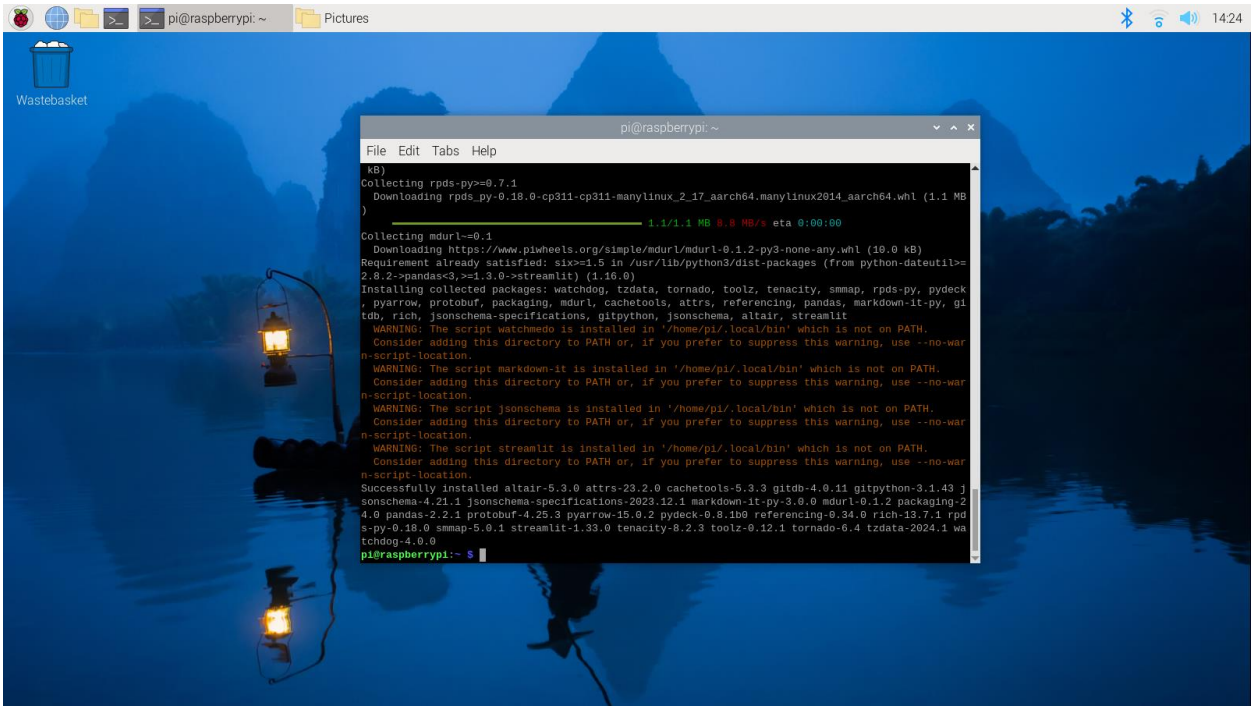
1. We plug the power adapter to Raspberry Pi and power up the system; open Remote Desktop Connection (or SSH connection) on our laptop and connect to Raspberry Pi.
2. We open a terminal window on your computer and type "***python –version***" to make sure you have Python 3 installed. If not, go to https://www.python.org/downloads/ and download the proper release.



3. We installed the following packages on our computer, if they are not already installed.
    - Boto3: an AWS Python Software Development Kit (AWS) to create, configure, and manage AWS services. We use Boto3 to interact with AWS S3.
    - Streamlit: will also install pandas and NumPy, if they are not installed.
    - plotly: a plotting library used by Streamlit.

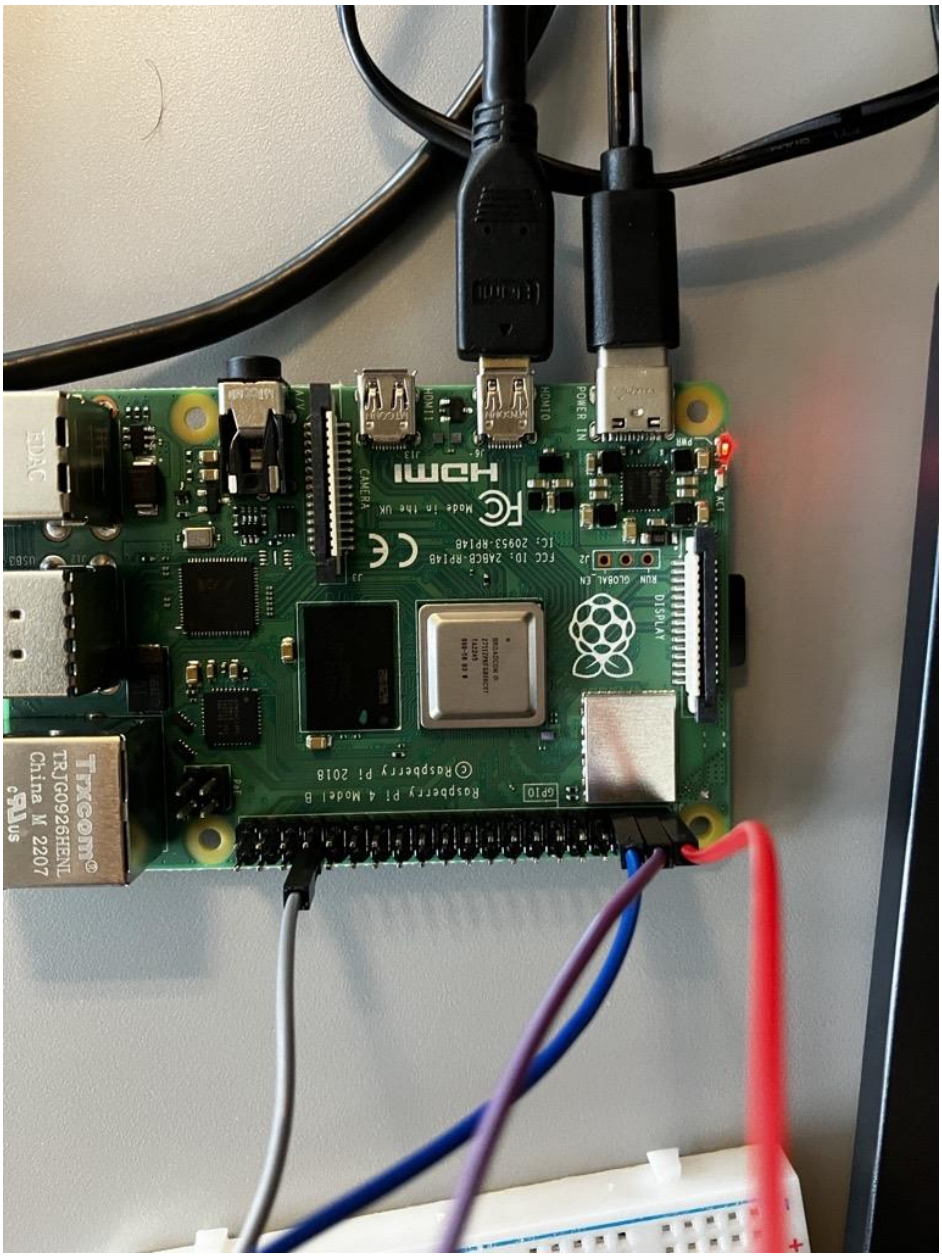

pi@raspberrypi: ~
14:21
Wastebasket
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ pip install boto3 --break-system-packages
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting boto3
Downloading https://www.piwheels.org/simple/boto3/boto3-1.34.79-py3-none-any.whl (136 kB)
136.7/136.7 kB 505.7 kB/s eta 0:00:00
Collecting botocore<1.35.0,>=1.34.79
Downloading https://www.piwheels.org/simple/botocore/botocore-1.34.79-py3-none-any.whl (12.1 MB)
12.1/12.1 MB 2.5 MB/s eta 0:00:00
Collecting jmespath<2.0.0,>=0.7.1
Downloading https://www.piwheels.org/simple/jmespath/jmespath-1.0.1-py3-none-any.whl (20 kB)
Collecting s3transfer<0.11.0,>=0.10.0
Downloading https://www.piwheels.org/simple/s3transfer/s3transfer-0.10.1-py3-none-any.whl (82 kB)
82.1/82.1 kB 2.5 MB/s eta 0:00:00
Collecting python-dateutil<3.0.0,>=2.1
Downloading https://www.piwheels.org/simple/python-dateutil/python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
229.9/229.9 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in /usr/lib/python3/dist-packages (from botocore<1.35.0,>=1.34.79->boto3) (1.26.12)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil<3.0.0,>=2.1->botocore<1.35.0,>=1.34.79->boto3) (1.16.0)
Installing collected packages: python-dateutil, jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.34.79 botocore-1.34.79 jmespath-1.0.1 python-dateutil-2.9.0.post0 s3transfer-0.10.1
pi@raspberrypi:~ $

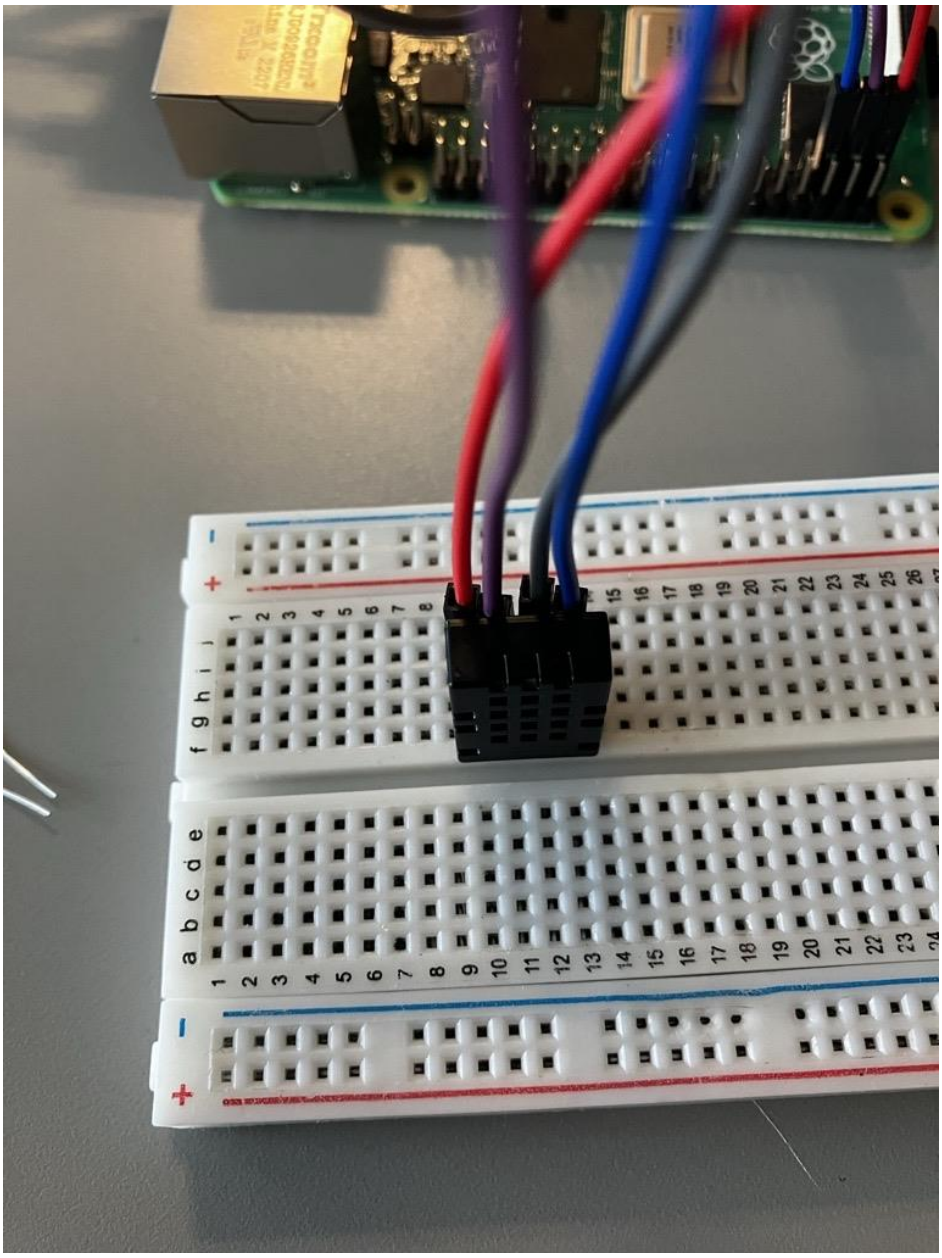

pi@raspberrypi: ~
Pictures
14:27
Wastebasket
pi@raspberrypi: ~
File Edit Tabs Help
WARNING: The script watchmedo is installed in '/home/pi/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script markdown-it is installed in '/home/pi/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script jsonschema is installed in '/home/pi/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script streamlit is installed in '/home/pi/.local/bin' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed altair-5.3.0 attrs-23.2.0 cachetools-5.3.3 gitdb-4.0.11 gitpython-3.1.43 jsonschema-4.21.1 jsonschema-specifications-2023.12.1 markdown-it-py-3.0.0 mdurl-0.1.2 packaging-24.0 pandas-2.2.1 protobuf-4.25.3 pyarrow-15.0.2 pydeck-0.8.1b0 referencing-0.34.0 rich-13.7.1 rpds-py-0.18.0 smmap-5.0.1 streamlit-1.33.0 tenacity-8.2.3 toolz-0.12.1 tornado-6.4 tzdata-2024.1 watchdog-4.0.0
pi@raspberrypi:~ $ pip install plotly --break-system-packages
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting plotly
Downloading https://www.piwheels.org/simple/plotly/plotly-5.20.0-py3-none-any.whl (15.7 MB)
15.7/15.7 MB 2.4 MB/s eta 0:00:00
Requirement already satisfied: tenacity>=6.2.0 in ./.local/lib/python3.11/site-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in ./.local/lib/python3.11/site-packages (from plotly) (24.0)
Installing collected packages: plotly
Successfully installed plotly-5.20.0
pi@raspberrypi:~ $



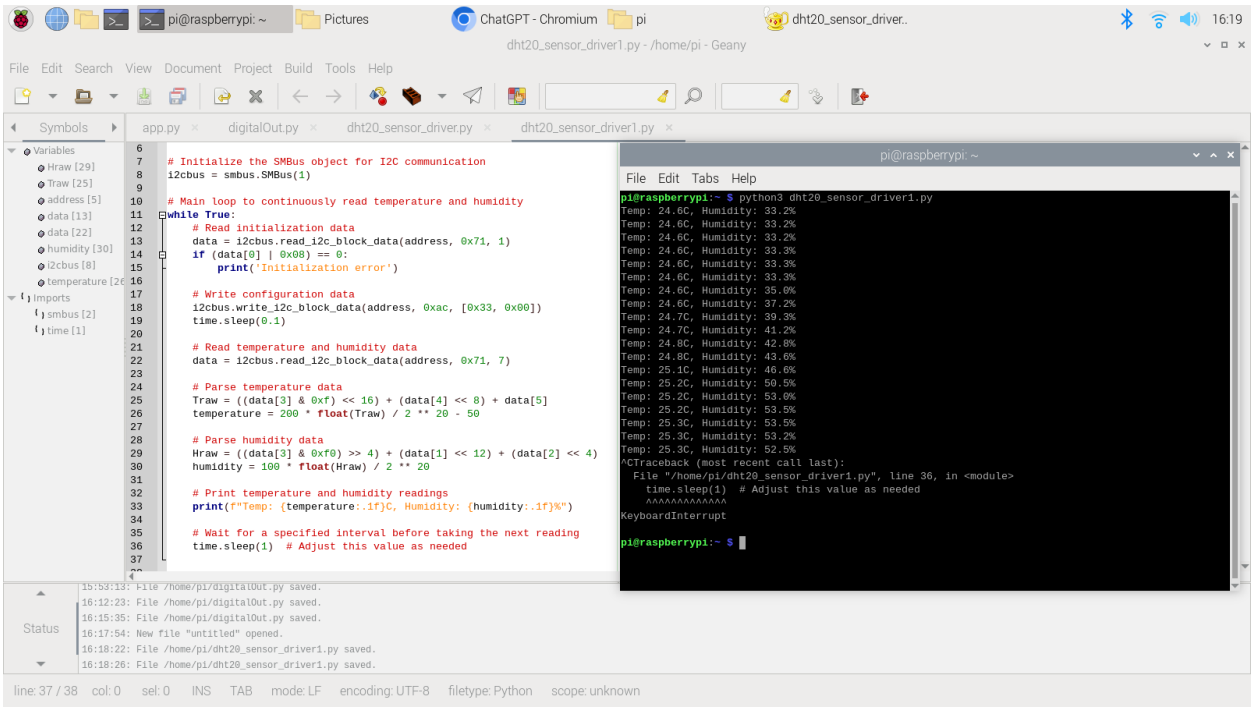
4. Next, we went to https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html to download and install the AWS CLI (Command Line Interface) on our computer.

5. We followed the instructions from Lab 4 to connect the temperature sensor to the Raspberry Pi.


Raspberry Pi 4 Model B
Raspberry Pi 2018
TRJG0926HENL
China M 2207

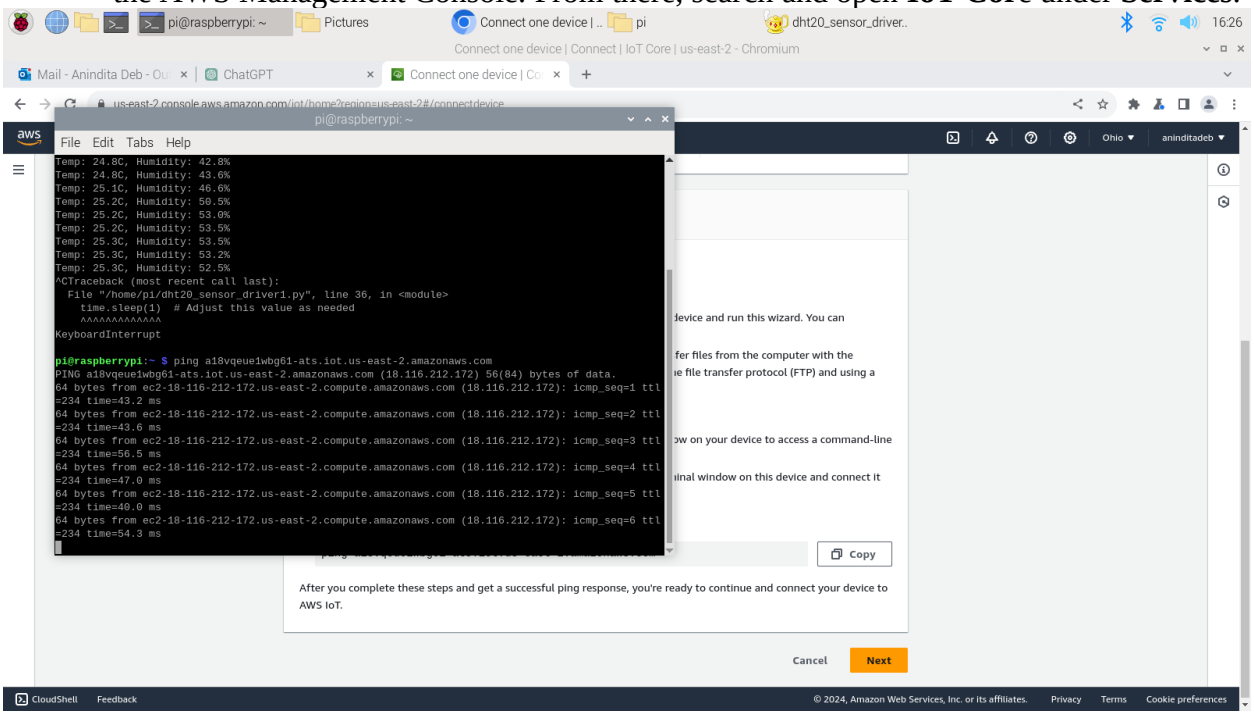


6. Make sure the sensor is working properly using the ***digitalOut.py*** code from Lab 4



Section 2: Configuring AWS IoT Core

7. We logged into our Amazon AWS account. Once you are logged in, navigate to the AWS Management Console. From there, search and open **IoT Core** under **Services**.



8. In the AWS IoT Core page, select **Connect One Device** from the **Connect** section in the menu pane. Follow the instructions to configure your Thing as you did in Lab 6. Ensure you unzip the file containing the certificates onto your Pi, and verify that your SDK is set to Python.

9. In the AWS IoT Core page, under **Security** click on **Policies**, and select the policy named
<Your thing's name>-Policy. Click the **Edit Active Version** button, and configure **Policy Effect** to **Allow**, and configure **Policy Action** and **Policy Resource** to the wildcard (*). Remember to check the box in **Policy Version Status** to save these changes, then you can click the orange Save as new version button.

10. To attach the policy to your certificate, go to "Certificates" (under "Security") in the AWS IoT Core page. Click the certificate you created and choose **Attach policy** from the **Actions** menu. Select the policy from the list and click **Attach policies**. Now AWS IoT is ready to receive data from the device.

11. Go back to the Raspberry Pi and create a folder named "cert" in your application folder for this lab. Transfer the device certificate, private key, and root CA files into the "cert" folder. Before transferring, you can rename the files as follows:
    - Device certificate: RaspberryPi-cert.pem
    - Private key: RaspberryPi-private.pem.key
    - Root CA: RootCA1.pem

Section 3: Setting up AWS S3 Bucket

12. Go to the AWS Management Console (https://console.aws.amazon.com/). Using the **Search**
bar, look for and select **S3**.
13. On the S3 page, click **Create bucket**.

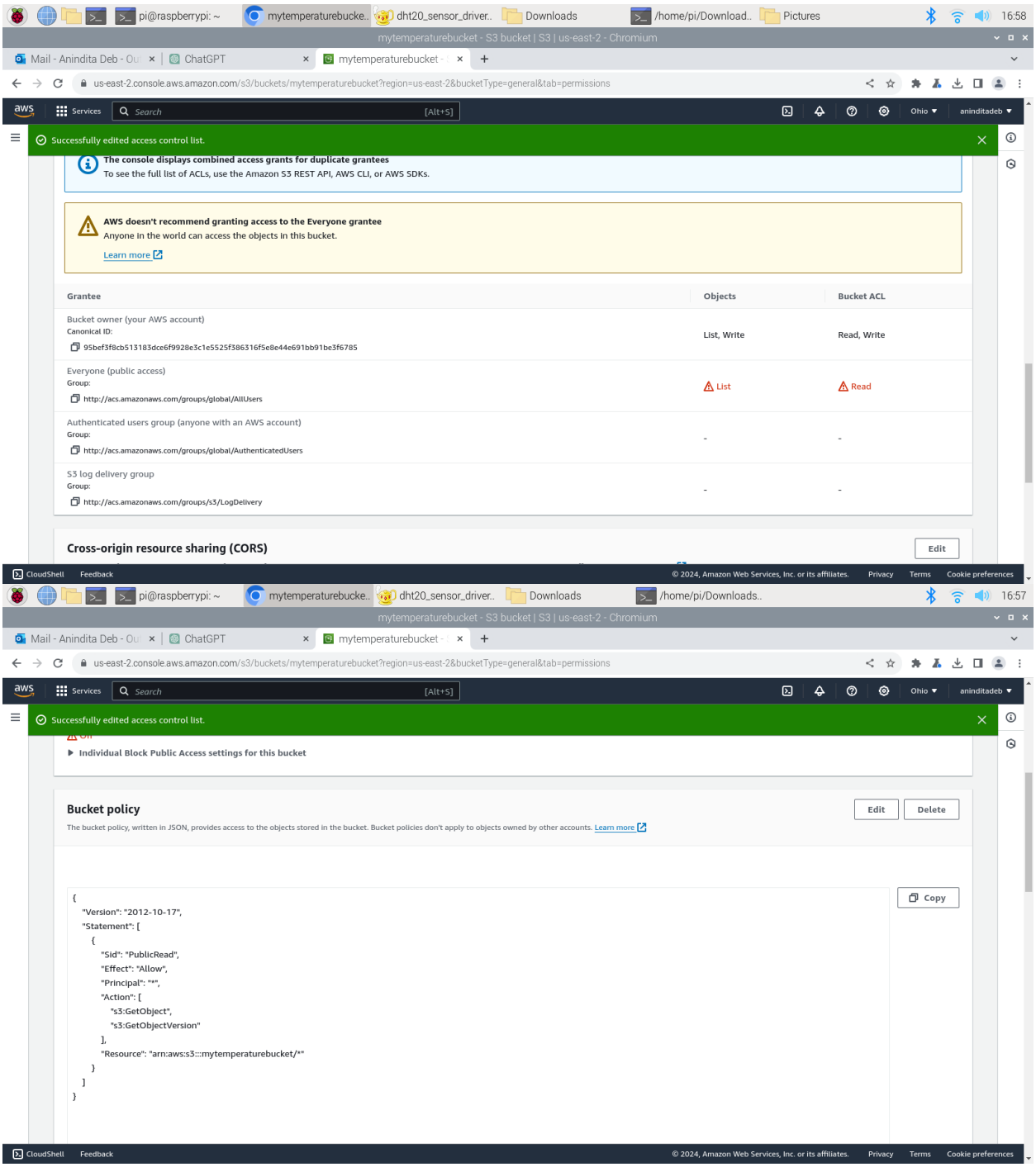
14. We created a bucket name that was a globally unique name with no uppercase letters or spaces. Under **Object Ownership** click **ACLs enabled**. Under **Block Public Access settings for this bucket**, uncheck the box **Block all public access**. Now, click **Create bucket**, then click the button to acknowledge your bucket will be public, and confirm.
15. Select the newly created bucket. Click on the **Permissions** tab. Next, click the edit button next to **Bucket policy**. Copy the following text into the policy text box. Click the copy button next to Bucket ARN and paste it into the resource field of the policy. Make sure to preserve

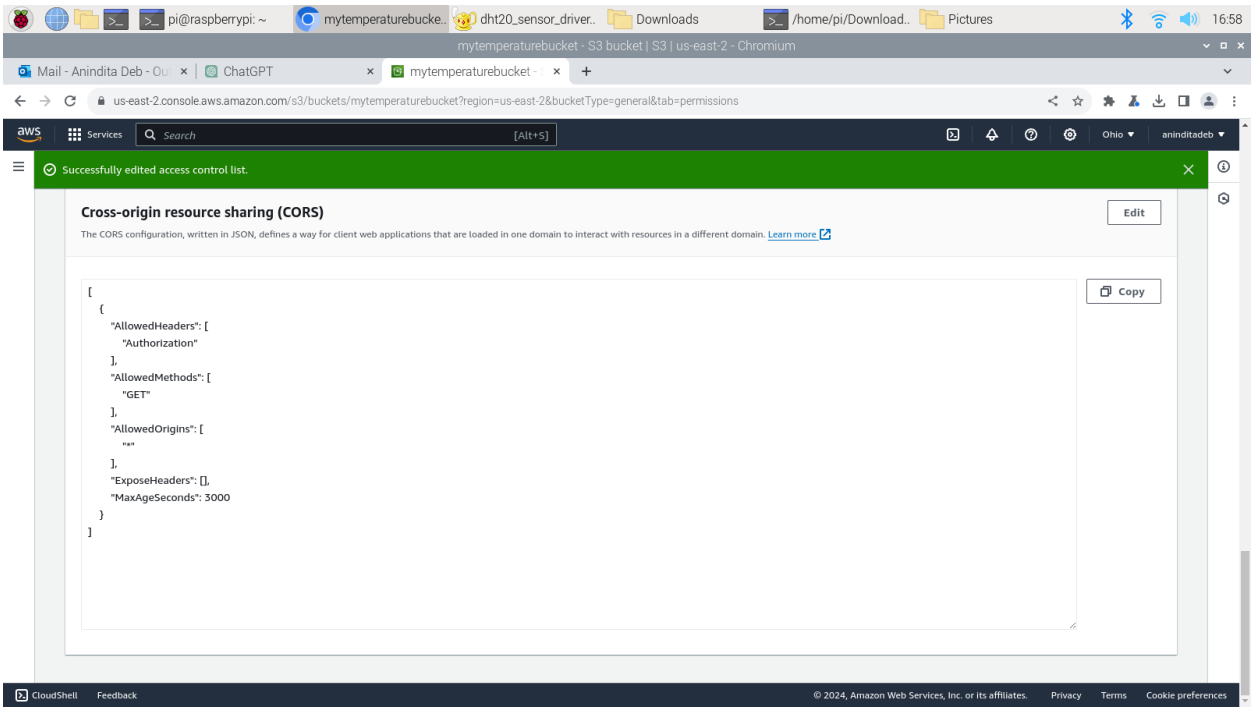
16. Scroll down to the CORS section and click the edit button. Paste the following text in the text box and **Save changes**.

```
[
    {
        "AllowedHeaders": [ "Authorization"
```

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

17. Under **Access Control List**, click **Edit**. Click the check boxes to give the **List** and **Read** permissions to **Everyone (Public Access)**. Then click the box to acknowledge your bucket is public and **Save changes**.
18. Navigate to **IoT Core** in AWS. On the sidebar menu, click **Rules** under **Message Routing**. Then click **Create rule**. Give the rule a name then click **Next** .

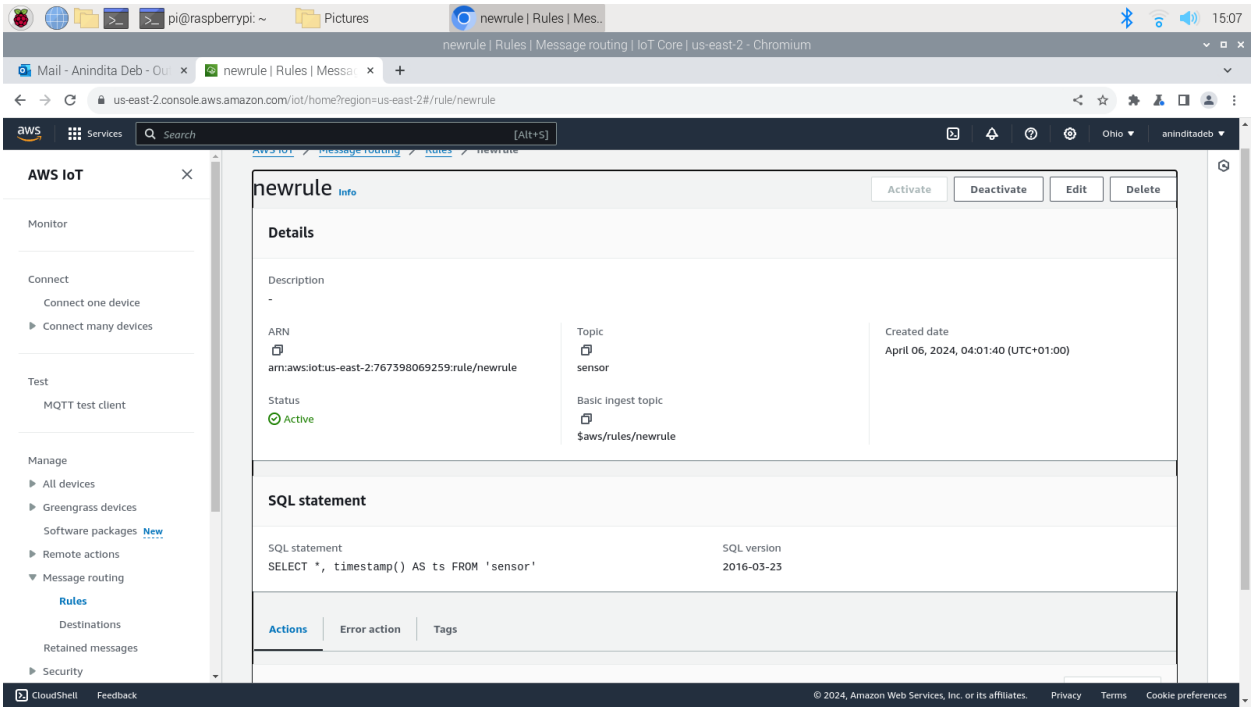

pi@raspberrypi: ~
mytemperaturebucke..
dht20_sensor_driver..
Downloads
/home/pi/Download..
Pictures
16:58
mytemperaturebucket - S3 bucket | S3 | us-east-2 - Chromium
Mail - Anindita Deb - Ou
ChatGPT
mytemperaturebucket -
us-east-2.console.aws.amazon.com/s3/buckets/mytemperaturebucket?region=us-east-2&bucketType=general&tab=permissions
Services
Search
[Alt+S]
Ohio
aninditadeb
Successfully edited access control list.
The console displays combined access grants for duplicate grantees
To see the full list of ACLs, use the Amazon S3 REST API, AWS CLI, or AWS SDKs.
AWS doesn't recommend granting access to the Everyone grantee
Anyone in the world can access the objects in this bucket.
Learn more
Grantee
Objects
Bucket ACL
Bucket owner (your AWS account)
Canonical ID:
95bef3f8cb513183dce6f9928e3c1e5525f386316f5e8e44e691bb91be3f6785
List, Write
Read, Write
Everyone (public access)
Group:
http://acs.amazonaws.com/groups/global/AllUsers
List
Read
Authenticated users group (anyone with an AWS account)
Group:
http://acs.amazonaws.com/groups/global/AuthenticatedUsers
S3 log delivery group
Group:
http://acs.amazonaws.com/groups/s3/LogDelivery
Cross-origin resource sharing (CORS)
Edit
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
pi@raspberrypi: ~
mytemperaturebucke..
dht20_sensor_driver..
Downloads
/home/pi/Downloads..
16:57
mytemperaturebucket - S3 bucket | S3 | us-east-2 - Chromium
Mail - Anindita Deb - Ou
ChatGPT
mytemperaturebucket -
us-east-2.console.aws.amazon.com/s3/buckets/mytemperaturebucket?region=us-east-2&bucketType=general&tab=permissions
Services
Search
[Alt+S]
Ohio
aninditadeb
Successfully edited access control list.
Individual Block Public Access settings for this bucket
Bucket policy
Edit
Delete
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more
Copy
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "PublicRead",
"Effect": "Allow",
"Principal": "*",
"Action": [
"s3:GetObject",
"s3:GetObjectVersion"
],
"Resource": "arn:aws:s3:::mytemperaturebucket/*"
}
]
}
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



19. In **SQL Statement** text box, enter the following query:

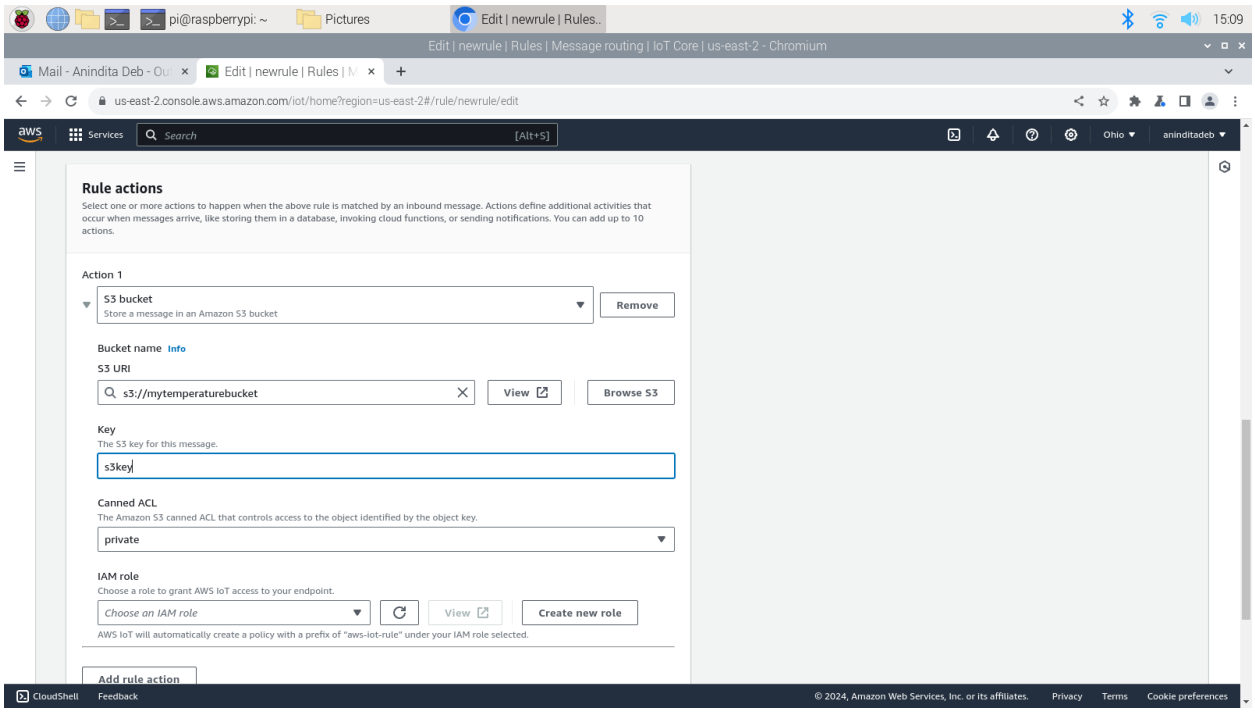
SELECT *, timestamp() AS timestamp FROM '<Your-incoming-IoT-Topic-Here>'

This will automatically add a timestamp to the JSON payload we send from NodeRED later. Make sure to change the topic name to something appropriate (i.e. “sensor”). Then click **Next.**



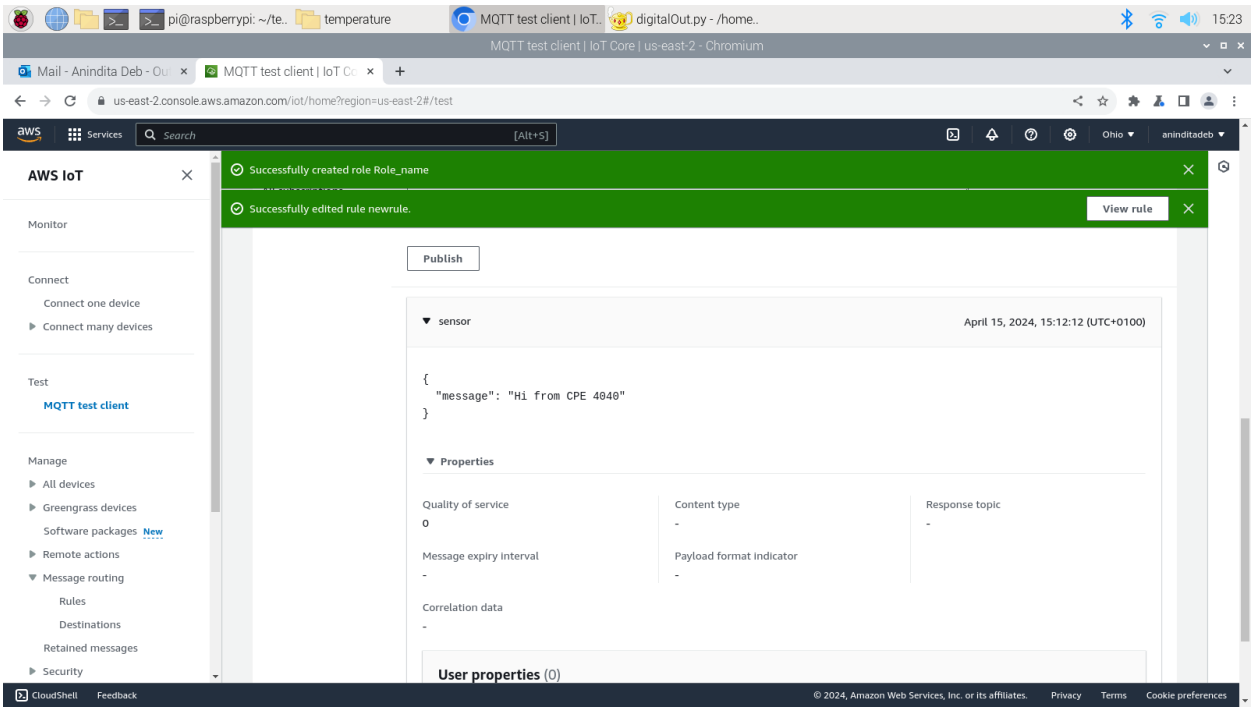
20. From the dropdown menu under **Rule actions**, search for and select **S3 bucket**. Next to Bucket name, click the **Browse S3** button and select the bucket you created.

Under **Key**, enter a name. This “key” refers to a value in a “key-value” pair. You can choose any value for the key, but make sure to take note of what you choose.

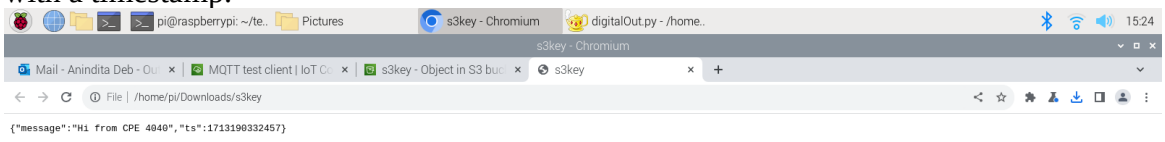


21. Go to **IAM Role** and select **Create new role** to begin creating a new role. Enter a name for the new role, then click **Next** and **Create** to complete the process.

22. In the sidebar menu, click **MQTT test client** under **Test**. Click the **Publish to a topic** tab. Enter the topic name you entered previously (i.e. ‘sensor’), and enter a custom message in the message field. Then click **Publish**.



23. In a separate tab, navigate back to the S3 page. Click on your bucket, then click on the Key object that you created. In the upper right click the **Download** button and open the file that is downloaded. You should see the message that you just sent, along with a timestamp.



Section 4: Set Up NodeRED Flow to Send Sensor Data to S3 Bucket via MQTT

24. On your Raspberry Pi, start NodeRED and open the flow editor in a web browser. Please refer back to the Lab 5 assignment for detailed instructions.

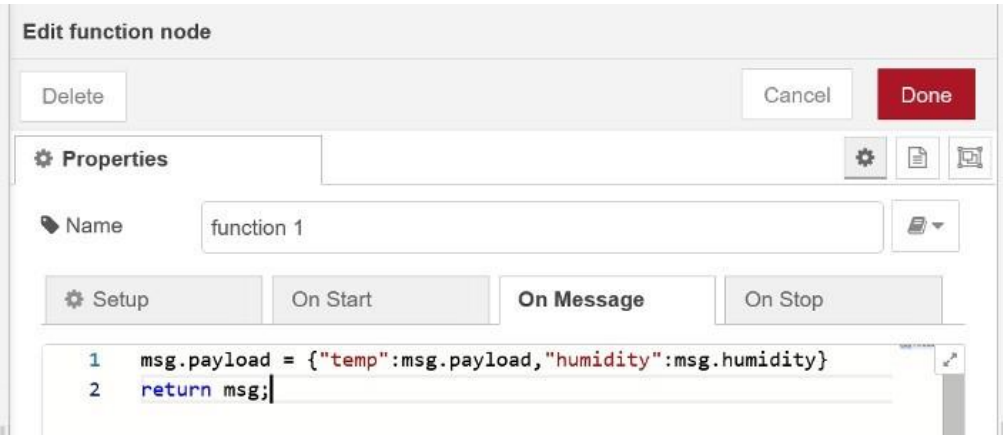
25. In the Palette Manager, install **node-red-contrib-dht-sensor**. If you are using DHT-20, you will install **node-red-contrib-aht20** instead.

26. Create a flow by adding an **inject** node, **rpi-dht22** (or **aht20**) node, **function** node, **mqtt out** node, and **debug** node and connect them as shown.

27. Edit the inject node to activate every 5 seconds.

28. Edit the **rpi-dht** node to use GPIO pin 12. Since we used **aht20**, nothing needs to be done.

29. Edit the function node and add the following code.



30. Edit the MQTT out node. Click the edit button next to server to add a new MQTT broker. Add the URL from AWS IoT -> Settings -> Device Data endpoint (refer back to Lab 6, Step 16). Add the device name in the Client ID field. Change Port to 8883.

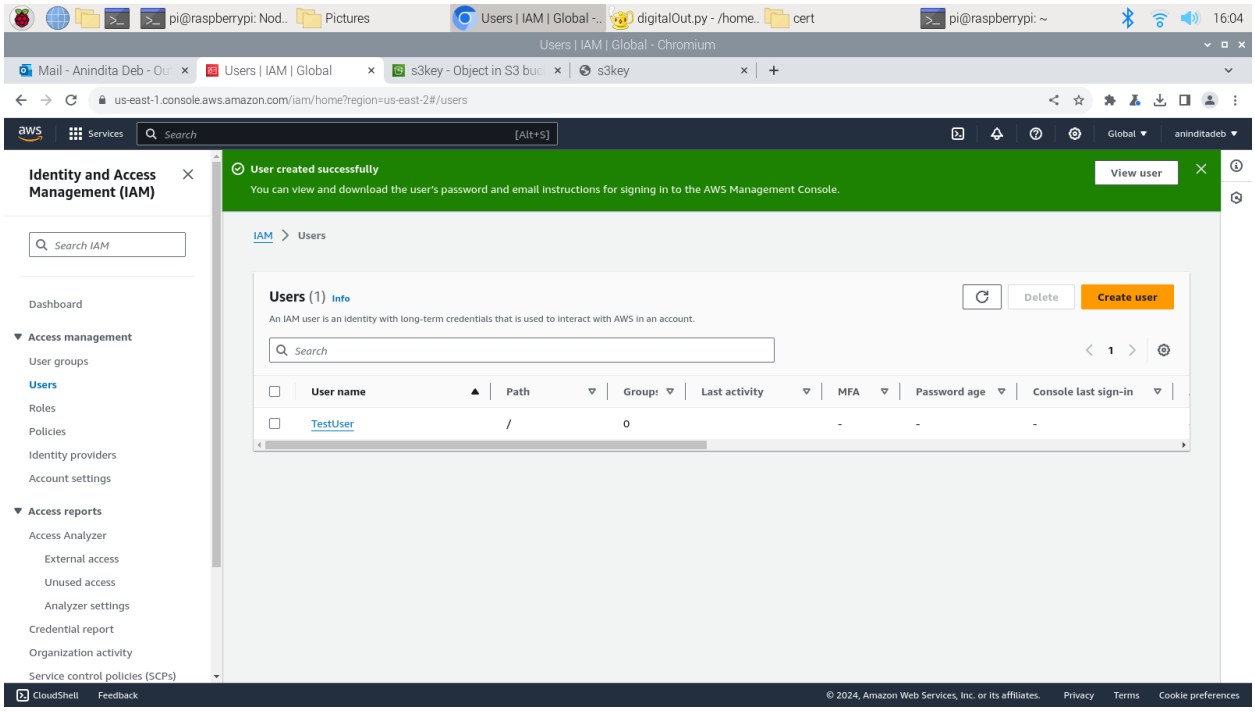
31. Click button to Use TLS. Edit the configuration. Check the “Use key and certificates from local files” and copy the filepath to each of the certificate files from your Pi into the textboxes.

32. Add a topic in the topic field. Make sure to use the same topic as you configured in the S3 bucket.

33. Deploy the flow and make sure all nodes are connected.

Section 5: Sending Data in S3 Bucket to Streamlit

34. Go to the AWS Management Console (https://console.aws.amazon.com/). Using the **Search** bar, look for and select **IAM**, then click **Users** in the IAM sidebar menu. From here, click **Add Users**, enter a name for the user, and click **Next**.



35. Go to **Permissions Boundary**, select the checkbox, then search for **AmazonS3ReadOnlyAccess** in the filter box. Once selected, click "**Next**" and "**Create User**" to complete the process.

36. Select your newly created user, and click the **Security Credentials** tab. Under **Access Keys**, click **Create access key**.

37. From **Access key best practices & alternatives** page, choose the “Local code” option and click **Next** then **Create access key**. Copy the resulting **Access key** and **Secret access key** into a text file. **Important**: You will not be able to see the secret access key again, make sure you copy it now.

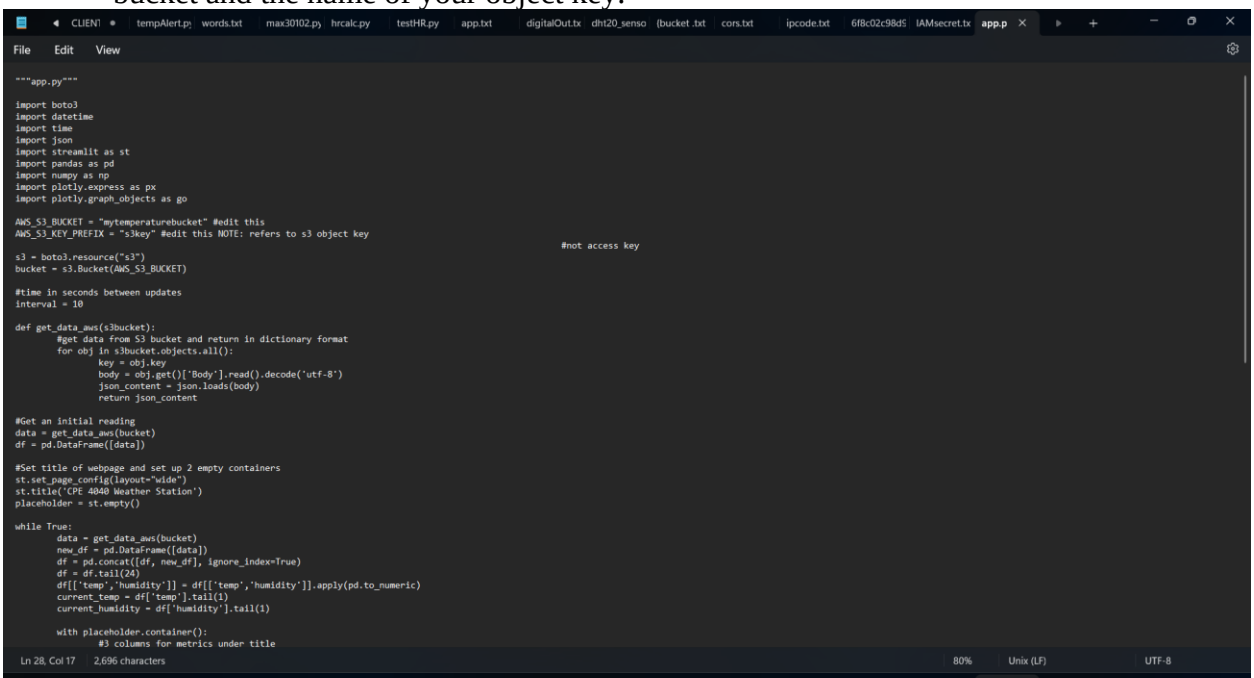
38. In your Windows/Mac command terminal, type “**aws configure**”

39. When prompted, enter your access and secret access keys. You may skip the other prompts by pressing Enter without typing anything. This creates a credentials file that will be used by the script. To make sure it was created, go to C:\Users\[user]\.aws to see the file. On Mac, it should be in /Users/[user]/.aws.

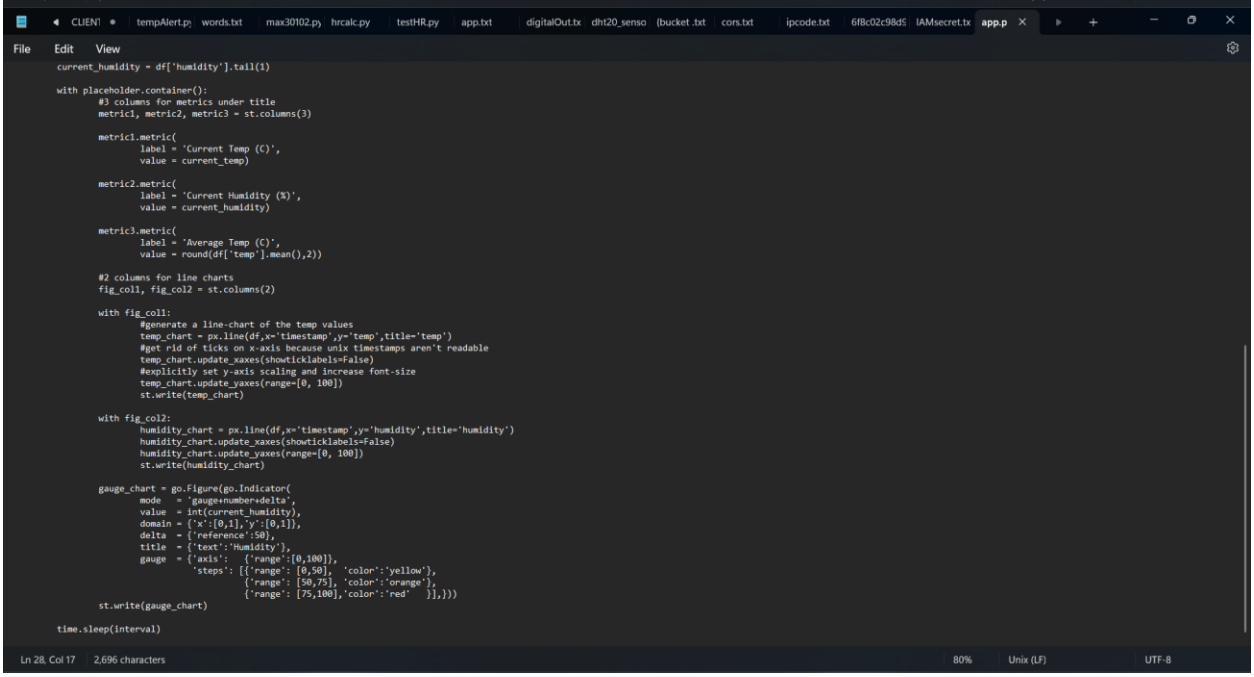
Note: Make sure that no additional spaces or characters are added when pasting in these keys. It could cause some annoying errors to debug later!

40. Download the **app.py** script from D2L.

41. Open **app.py** in a text editor and edit the following lines with the name of your S3 bucket and the name of your object key.

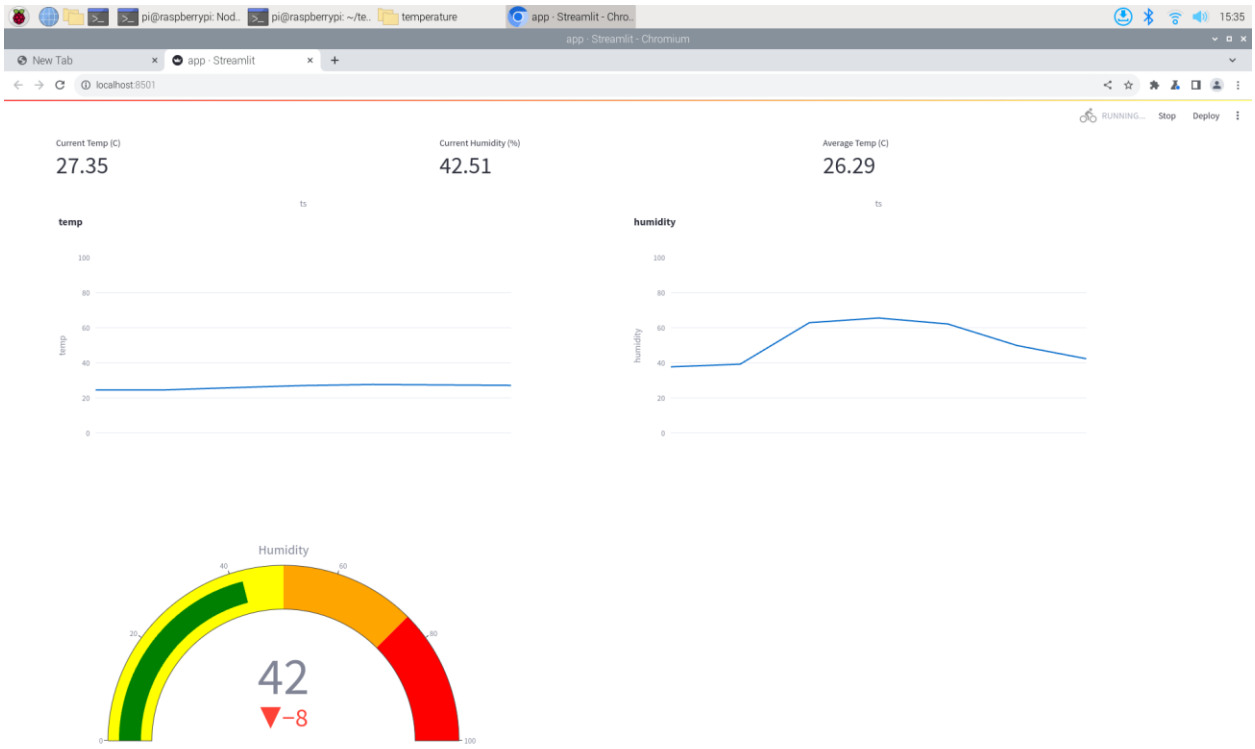


```
"""app.py"""

import boto3
import datetime
import time
import json
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go

AWS_S3_BUCKET = "mytemperaturebucket" #edit this
AWS_S3_KEY_PREFIX = "s3key" #edit this NOTE: refers to s3 object key
                                                                    #not access key
s3 = boto3.resource("s3")
bucket = s3.Bucket(AWS_S3_BUCKET)

#time in seconds between updates
interval = 10

def get_data_aws(s3bucket):
        #get data from S3 bucket and return in dictionary format
        for obj in s3bucket.objects.all():
                key = obj.key
                body = obj.get()['Body'].read().decode('utf-8')
                json_content = json.loads(body)
                return json_content

#Get an initial reading
data = get_data_aws(bucket)
df = pd.DataFrame([data])

#Set title of webpage and set up 2 empty containers
st.set_page_config(layout="wide")
st.title('CPE 4040 Weather Station')
placeholder = st.empty()

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)

        with placeholder.container():
                #3 columns for metrics under title
```



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

        with placeholder.container():
                #3 columns for metrics under title
                metric1, metric2, metric3 = st.columns(3)

                metric1.metric(
                        label = 'Current Temp (C)',
                        value = current_temp)

                metric2.metric(
                        label = 'Current Humidity (%)',
                        value = current_humidity)

                metric3.metric(
                        label = 'Average Temp (C)',
                        value = round(df['temp'].mean(),2))

                #2 columns for line charts
                fig_col1, fig_col2 = st.columns(2)

                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)

                gauge_chart = go.Figure(go.Indicator(
                        mode   = 'gauge+number+delta',
                        value  = int(current_humidity),
                        domain = {'x':[0,1],'y':[0,1]},
                        delta  = {'reference':50},
                        title  = {'text':'Humidity'},
                        gauge  = {'axis':   {'range':[0,100]},
                                  'steps': [{'range': [0,50],  'color':'yellow'},
                                            {'range': [50,75], 'color':'orange'},
                                            {'range': [75,100],'color':'red'   }],}))
                st.write(gauge_chart)

        time.sleep(interval)
```

42. To view your dashboard, run the script with the command '**streamlit run app.py**' and open the URL provided in a web browser. Once you've done this, you should see two line charts and a gauge on your dashboard, similar to ones shown below.



IV. Conclusion

This lab really took much longer than we had anticipated since the raspberry pi needed to be replaced and re-flashed due to technical issues. Additionally, since we started over that background foundation of all the labs were sadly lost. We redownloaded and reinstalled a lot of the libraries and modules. We caught up half-way through by the end of the first week, but it was not getting easier.

We used the newer sensor DHT-20 (unlike the previous lab submissions where we used DHT-11/12)and it was not clear we didn't realize all the pins needed to be connected to have the sensor fully working. It's not great that there aren't many pinout schematics for this particular sensor either so we were struggling to figure out which pin meant what and where to connect it to the Raspberry Pi GPIOs.

We ran into a lot of issues with the Node-Red Flow. In particular the sensor was not connecting to AWS Iot and we had to fix the connection with downloading another set of files (private/certificates etc) for the temperature Thing.

The function was also causing us many problems as it was not parsing the two variables temperature and humidity. It was sending out both values under just temperature until we changed the function to the following:

)":msg.payload.temperature_C,"humidity":msg.payload.humidity}

Under the On Message section for the function.

It would be nice to mention that we were struggling to have aws configure on our laptop and opted to do it from the Raspberry Pi terminal. It not only had no lag and worked instantly we didn't have to search for the pathway to the app.py file. The file changes Justin helped us with was nice. He’s super responsive to the emails we sent out to him and great with help in lab also.

Overall, this lab had us stressed, but Justin helped us out a ton and a big thanks to the ECE department for loaning us a Raspberry Pi 4 when we were in a pinch.