

Student name: Tien Phat Nguyen

Student ID:223171213

SIT225: Data Capture Technologies

Activity 3.1: Arduino IoT Cloud and Dashboard with Arduino Nano 33 IoT devices

Arduino Cloud is your next exciting journey to build, control and monitor your connected projects. You can connect anything to Arduino Cloud including a wide range of compatible Arduino boards such as Arduino Nano 33 IoT or a third-party device that speaks Python. Arduino Cloud is an all-in-one IoT solution that empowers makers to create from anywhere, control their devices with stunning dashboards.

In this activity, you will connect your Arduino board to the cloud as a device, register a thing (in terms of a cloud variable) with your device, create a dashboard with a graphical widget which show value that is sent from the sketch running in your Arduino board.

Hardware Required

Arduino Nano 33 IoT Board
Wi-Fi hotspot (preferably your smartphone hotspot)
USB cable

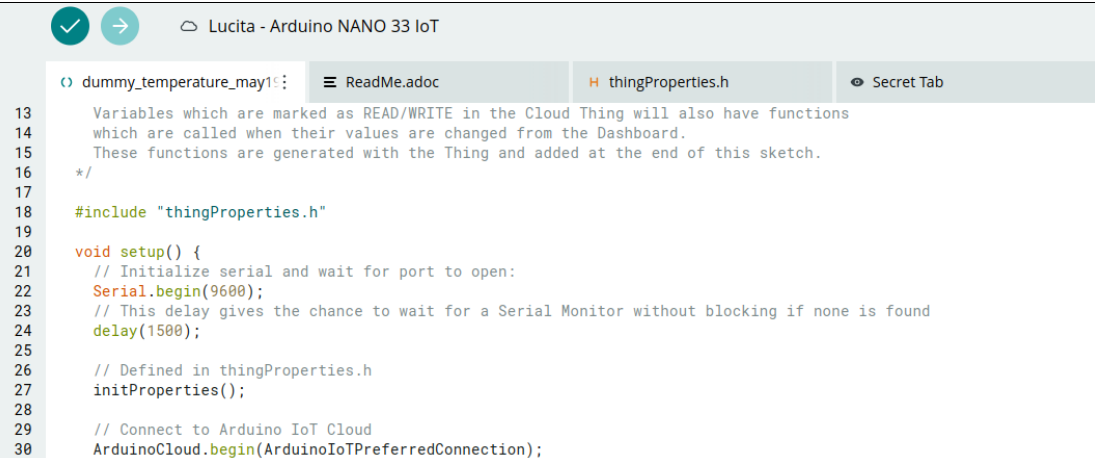
Software Required

Arduino programming environment (Arduino IDE)
Arduino IoT Cloud (<https://app.arduino.cc>)

Steps

Step	Action
1	Account creation: Create an account in Arduino IoT Cloud (https://app.arduino.cc). Once done, login to your account.

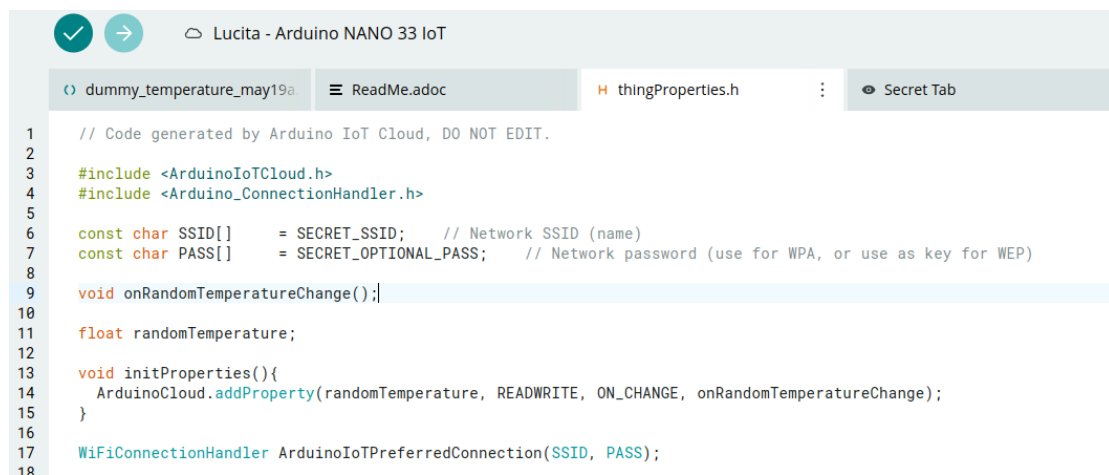
	<p>You can follow Arduino tutorial (https://support.arduino.cc/hc/en-us/articles/360016495559-Add-and-connect-a-device-to-Arduino-Cloud) for detail. This tutorial is referred to in the steps below.</p>
2	<p>Add the device: Follow step 1 in the tutorial to add your Arduino Nano 33 IoT device.</p>
3	<p>Create and configure a Thing: Follow step 2 in the tutorial to create a Thing (sensor) and attach it to the device created in step 2 above. Note that a Thing is created as a cloud variable of specific data types. In your sketch, there should exist the same variable name and type.</p> <div data-bbox="293 623 1395 1201" data-label="Image"> <p>The screenshot shows the Arduino Cloud interface. At the top, there are tabs for 'Setup' and 'Sketch'. Below the tabs, there are two main sections: 'Cloud Variables' and 'Associated Device'. The 'Cloud Variables' section contains a table with three variables: 'led' (bool), 'randomTemperature' (float), and 'temperature' (float). The 'Associated Device' section shows the device 'Lucita' with its ID, type (Arduino NANO 33 IoT), and status (Online). Below this, there is a 'Network' section showing the Wi-Fi Name 'Telstra18...' and a masked password. At the bottom of the screenshot, there is a red text overlay that reads: 'Consider only a single cloud variable 'randomTemperature' for this activity and ignore the rest (led or temperature).' Below the screenshot, there is a text overlay that reads: 'To keep things simple, the "Sketch" tab in the above screenshot prepares'.</p> </div> <p>Consider only a single cloud variable 'randomTemperature' for this activity and ignore the rest (led or temperature).</p> <p>To keep things simple, the "Sketch" tab in the above screenshot prepares</p>
4	<p>Configure your Wi-Fi: In the above screenshot, the right column shows Associated Device and Network sections. You can see "Telstra" network is shown at the time this document was prepared. This should be your smartphone hotspot which is the most convenient considering the mobility – you can carry your laptop and projects without changing in the sketch.</p>
5	<p>Sketch tab: There is a Sketch tab at the top right corner of the screenshot above which gives you code ready to deploy in your Arduino board.</p>



```
13 Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
14 which are called when their values are changed from the Dashboard.
15 These functions are generated with the Thing and added at the end of this sketch.
16 */
17
18 #include "thingProperties.h"
19
20 void setup() {
21   // Initialize serial and wait for port to open:
22   Serial.begin(9600);
23   // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
24   delay(1500);
25
26   // Defined in thingProperties.h
27   initProperties();
28
29   // Connect to Arduino IoT Cloud
30   ArduinoCloud.begin(ArduinoIoTPreferredConnection);
```

There are 3 files in the project. The first tab shows the sketch (.ino) is generally the project name. Two other sketches are header files (.h) - thingProperties.h and arduino_secrets.h.

The thingProperties.h header looks like below.



```
1 // Code generated by Arduino IoT Cloud, DO NOT EDIT.
2
3 #include <ArduinoIoTCloud.h>
4 #include <Arduino_ConnectionHandler.h>
5
6 const char SSID[] = SECRET_SSID; // Network SSID (name)
7 const char PASS[] = SECRET_OPTIONAL_PASS; // Network password (use for WPA, or use as key for WEP)
8
9 void onRandomTemperatureChange();
10
11 float randomTemperature;
12
13 void initProperties(){
14   ArduinoCloud.addProperty(randomTemperature, READWRITE, ON_CHANGE, onRandomTemperatureChange);
15 }
16
17 WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
18
```

It shows several items including -

- Your wifi variables (such as ssid and password which you define in arduino_secrets.h header).
- The cloud variable exactly with the same name and type you have created while creating your Thing in step 3 above.
- An initialisation function called *void initProperties() {...}* which registers the Thing variable to the cloud.
- A template function *ArduinoIoTPreferredConnection(SSID, PASS)* which works behind the scenes to connect to the Arduino Cloud using the Wi-Fi you have configured.

The arduino_secrets.h header file contains Wi-Fi ssid and password. The Secret Tab shows fields where you can input Wi-Fi information. If you want to deploy using Cloud IDE shown above (called OTA upload), it needs to upgrade your

account. Instead, you can copy the sketch to your Arduino IDE installed on your computer.

```
sketch_iot_cloud.ino  arduino_secrets.h  thingProperties.h
1  #define SECRET_SSID " ";
2  #define SECRET_OPTIONAL_PASS " "
```

You can prepare the sketch and 2 header files as mentioned. Alternatively, you can download the code from here (https://github.com/deakin-deep-dreamer/sit225/tree/main/week_3).

The sketch in the GitHub link above shows the sketch below (*sketch_iot_cloud.ino*).

```
sketch_iot_cloud.ino  arduino_secrets.h  thingProperties.h
1  #include "thingProperties.h"
2
3  void setup() {
4      // Initialize serial and wait for port to open:
5      Serial.begin(9600);
6      // This delay gives the chance to wait for a Serial Monitor without
7      delay(1500);
8
9      // Defined in thingProperties.h
10     initProperties();
11
12     // Connect to Arduino IoT Cloud
13     ArduinoCloud.begin(ArduinoIoTPreferredConnection);
14
15     /*
16     The following function allows you to obtain more information
17     related to the state of network and IoT Cloud connection and errors
18     the higher number the more granular information you'll get.
19     The default is 0 (only errors).
20     Maximum is 4
21     */
22     setDebugMessageLevel(2);
23     ArduinoCloud.printDebugInfo();
24 }
25
26 void loop() {
27     ArduinoCloud.update();
28     // Your code here
29     randomTemperature = 1.0 * random(1, 100);
30     Serial.println("random temperature: " + String(randomTemperature));
31     delay(5*1000);
32 }
33
34 /*
35 Since Temperature is READ_WRITE variable, onRandomTemperatureChange()
36 executed every time a new value is received from IoT Cloud.
37 */
38 void onRandomTemperatureChange() {
39     // Add your code here to act upon Temperature change
40     Serial.println("--onRandomTemperatureChange");
41 }
42
```

The loop function generates a random value between 1 and 100 and assigns to *randomTemperature* variable which is a cloud variable and write in serial port and waits for 5 seconds.

Additional library to install: Arduino_CloudUtils

More info: <https://forum.arduino.cc/t/arduinoiotcloud-2-5-0-causes-compile-error/1373943/2>

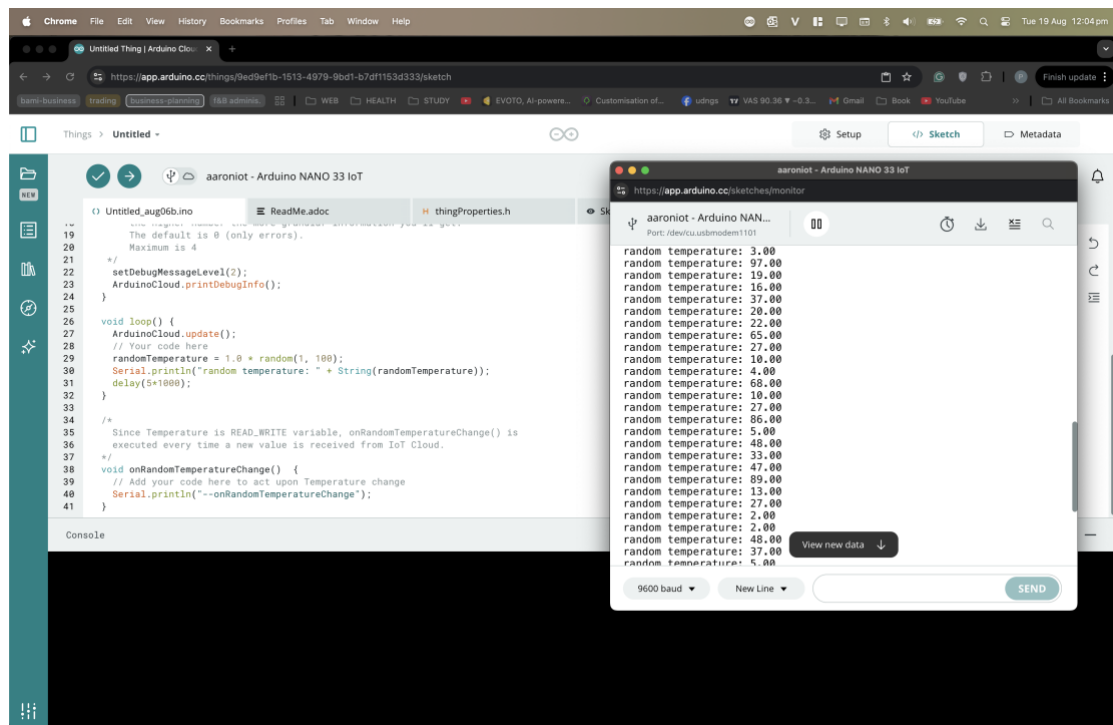
6

Deploy code to board:

You can upload code to Arduino board from Arduino IDE. You can observe the random temperature readings in the serial monitor.

Question: Screenshot the output of the serial monitor with random temperature values along with the initial Wi-Fi connection codes. Comment on the output lines.

Answer: <Your answer>



The serial monitor shows messages from the Wi-Fi library when it connects to my hotspot, like “Connected to Arduino IoT Cloud

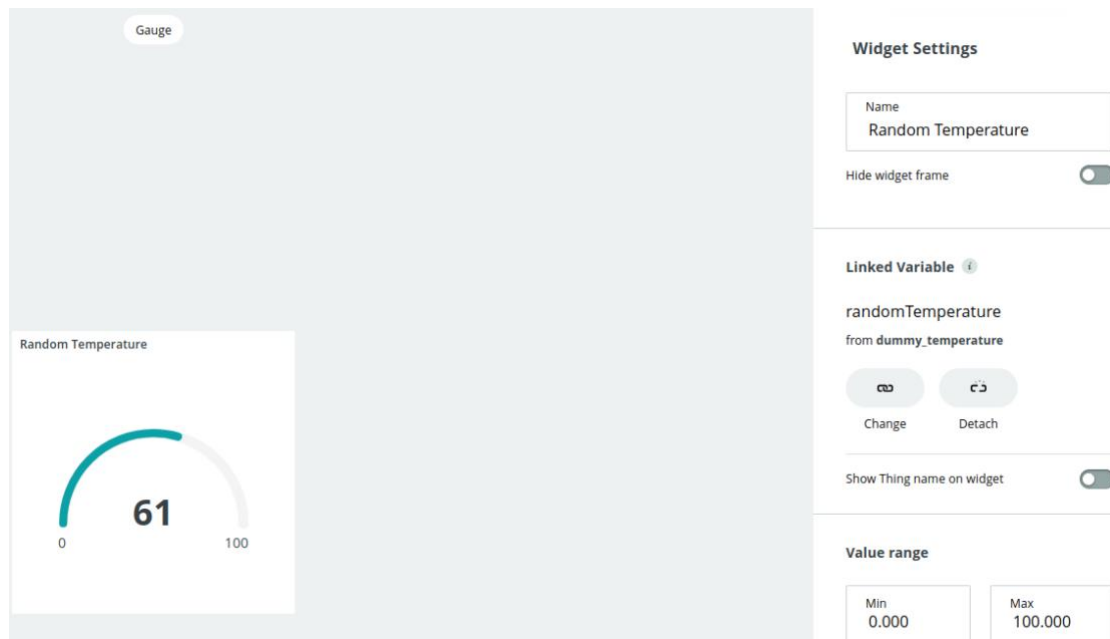
Thing ID: 9ed9ef1b-1513-4979-9bd1-b7df1153d333”, which tells me the Arduino Nano 33 IoT successfully joined the network. After that, every five seconds the sketch prints lines like “random temperature: 31.00” or “random temperature: 61.00”, indicating it’s generating a new value between 1 and 100 and sending it as the cloud variable.

7

Create a dashboard:

A dashboard in Arduino Cloud can be created to visualise the sensor readings sent by the Things connected to your Arduino Nano 33 IoT board. A list of Dashboard widgets is described in this tutorial (<https://docs.arduino.cc/arduino-cloud/cloud-interface/dashboard-widgets>).

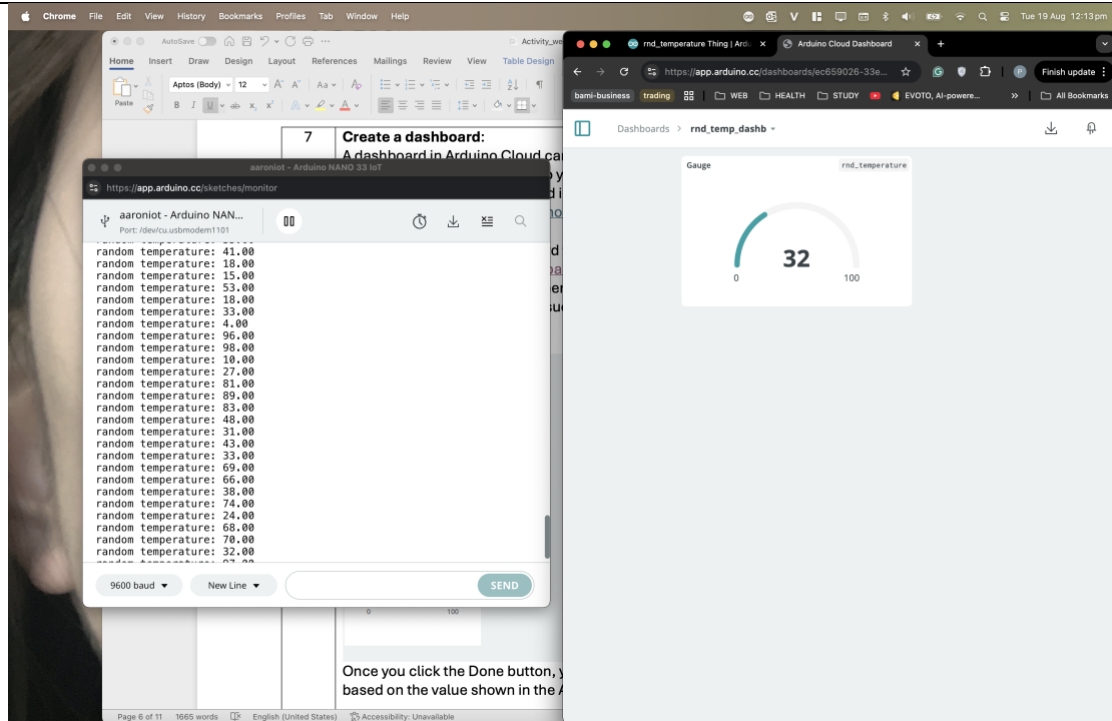
You can create a new dashboard from the Dashboards left menu items (<https://app.arduino.cc/dashboards>) where there are other menu items such as Devices and Things you have seen earlier. Creating a dashboard is simply choosing a dashboard widget, such as a Gauge and link it to the Thing cloud variable you have created.



Once you click the Done button, you should look the Gauge widget is updating based on the value shown in the Arduino IDE serial monitor.

Question: Screenshot the output of the serial monitor with random temperature values and the dashboard output so the Gauge value can be found in the serial monitor output.

Answer: <Your answer>



8 **Question:** If you recall, there is a function *initProperties()* in *thingProperties.h* file where there is a single line -

```
13 void initProperties(){
14   ArduinoCloud.addProperty(randomTemperature, READWRITE, ON_CHANGE, onRandomTemperatureChange);
15 }
```

A function *onRandomTemperatureChange()* exists (in *sketch_iot_cloud.ino* file) to respond to *ON_CHANGE* event. Can you explore what is the use of this function and when the *ON_CHANGE* event will trigger?

Answer: <Your answer>

The function *onRandomTemperatureChange()* acts as a callback that runs whenever the cloud variable *randomTemperature* is updated from the Arduino IoT Cloud side. Because *randomTemperature* is defined as a *READ_WRITE* variable in *initProperties()*, the *ON_CHANGE* event fires whenever a new value is written to that variable. In practice, after *ArduinoCloud.update()* checks for incoming changes, it will call *onRandomTemperatureChange()* if the value has changed remotely.

Activity 3.2: Arduino IoT Cloud with custom Python devices

You can connect anything to Arduino Cloud including a wide range of compatible Arduino boards such as Arduino Nano 33 IoT or a third-party device that **speaks Python**.

In this activity, you will connect a custom Python board to Arduino IoT Cloud and synchronise to another cloud variable, the one you created earlier, *randomTemperature*. This will enable you to receive data from Arduino board in your Python script.

Hardware Required

Arduino Nano 33 IoT Board

Wi-Fi hotspot (preferably your smartphone hotspot)

USB cable

Software Required

Arduino programming environment (Arduino IDE)

Arduino IoT Cloud (<https://app.arduino.cc>)

Python 3

Steps

Step	Action
1	Thing & Device Configuration: Following the tutorial (https://docs.arduino.cc/arduino-cloud/guides/python), <ol style="list-style-type: none">Create a new Thing, by clicking on the "Create Thing" button.Click on the "Select Device" in the "Associated Devices" section of your Thing.Click on "Set Up New Device" and select the bottom category ("Manual Device"). Click continue in the next window and choose a name for your device.Finally, you will see a new Device ID and a Secret Key generate. You can download them as a PDF. Make sure to save it as you cannot access your Secret Key again.
2	Create Variables: Follow the same tutorial mentioned above and do the following steps: <ol style="list-style-type: none">While in Thing configuration, click on "Add Variable" which will open a

	<p>new window.</p> <ol style="list-style-type: none"> Name your variable temperature and select it to be of a float type. Just below the variable name, there is a link '<i>Sync with other Things</i>'. Click the link and it will show a list of all the variables you have created so far in your Arduino Cloud account in all the devices. Select 'randomTemperature' variable from Arduino board and click the button 'Synchronise Variables'. At this point the randomTemperature data sent from your Arduino Nano board will arrive in Arduino Cloud, then it will be written to Python device variable temperature. This value assignment triggers on_write event in the listening client and corresponding callback function is called.
3	<p>Create Python script:</p> <p>Now you need to create a Python script to register to Arduino Cloud, register for a cloud variable called temperature and write a callback function for on_write event handling. You can write the code as below or download the code from here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_3/arduino_variable_sync.py).</p>

```

1  """
2      Requirement: arduino_iot_cloud
3      Install: pip install arduino-iot-cloud
4
5      @Ahsan Habib
6      School of IT, Deakin University, Australia.
7  """
8
9  import sys
10 import traceback
11 import random
12 from arduino_iot_cloud import ArduinoCloudClient
13 import asyncio
14
15 DEVICE_ID = "bc5c0fe9-e6ef-4eb0-90de-05032ffd9a83"
16 SECRET_KEY = "3oJYfrkmSNjM4YwKGJgV0bbBn"
17
18
19 # Callback function on temperature change event.
20 #
21 def on_temperature_changed(client, value):
22     print(f"New temperature: {value}")
23
24
25 def main():
26     print("main() function")
27
28     # Instantiate Arduino cloud client
29     client = ArduinoCloudClient(
30         device_id=DEVICE_ID, username=DEVICE_ID, password=SECRET_KEY
31     )
32
33     # Register with 'temperature' cloud variable
34     # and listen on its value changes in 'on_temperature_changed'
35     # callback function.
36     #
37     client.register(
38         "temperature", value=None,
39         on_write=on_temperature_changed)
40
41     # start cloud client
42     client.start()
43
44
45 if __name__ == "__main__":
46     try:
47         main() # main function which runs in an internal infinite loop
48     except:
49         exc_type, exc_value, exc_traceback = sys.exc_info()
50         traceback.print_tb(exc_traceback, file=print)

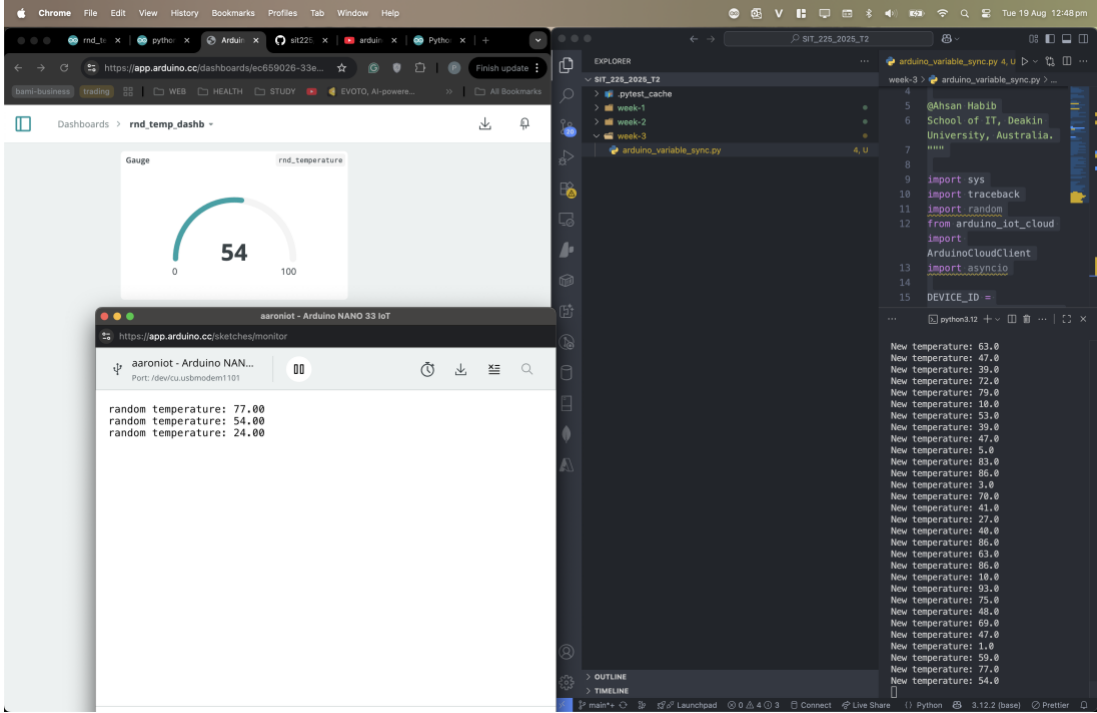
```

You should replace DEVICE_ID and SECRET_KEY as you were given in step 1-d above.

Question: Study the code and describe in your word how the statements match to the purpose mentioned in this step-3 above?

Answer:

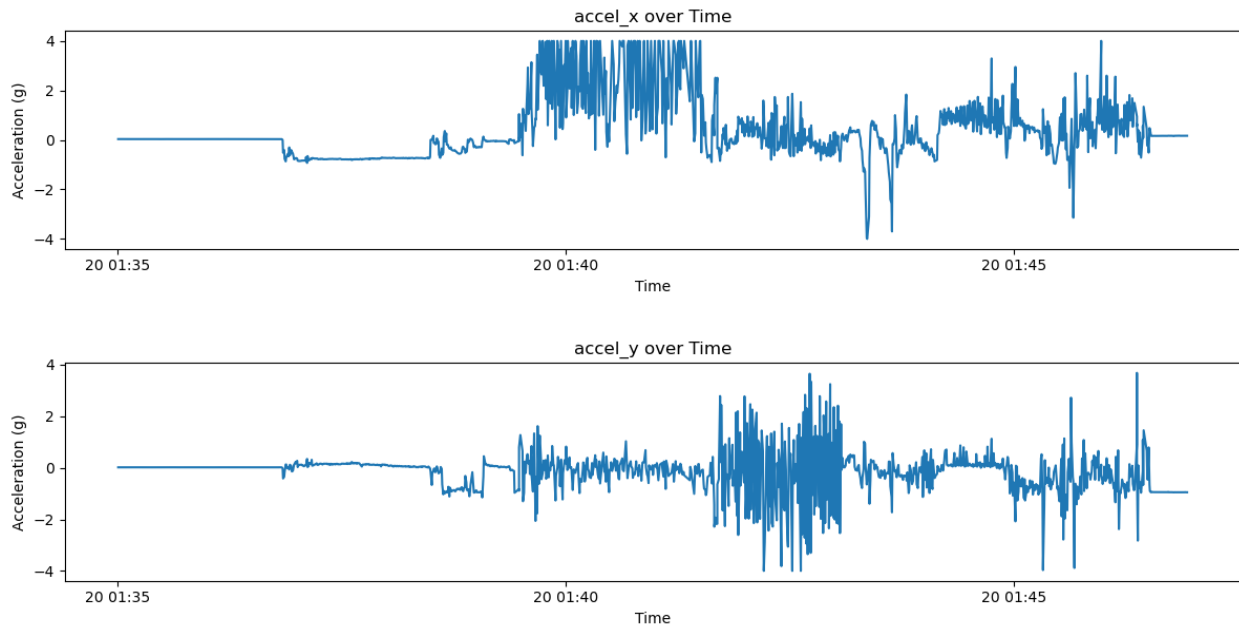
ArduinoCloudClient() constructor call creates the Python device using the ID and secret key got in step 1, so it's setting up a "Manual Device" in the cloud. Then client.register() adds the float variable named "temperature" and links it to the earlier randomTemperature via the sync feature, and it tells the client to call on_temperature_changed() whenever the cloud writes a new value. Finally, client.start() begins the connection loop so that your Python script keeps

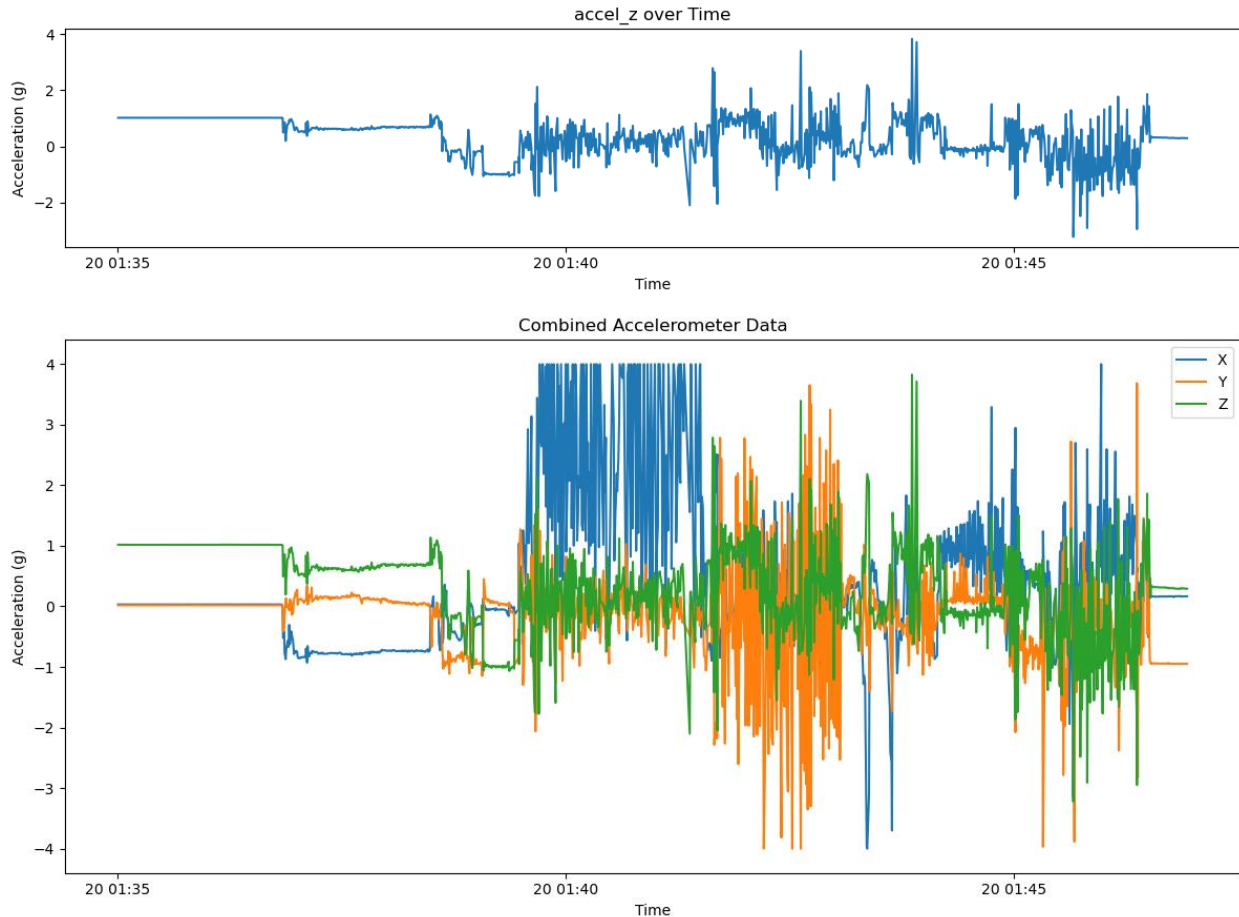
	listening for temperature updates as described.
4	<p>Run Python script: Run the python script from command line below - \$ python arduino_variable_sync.py</p> <p>At this point, the command line output should show connecting to Arduino cloud and print new temperature values periodically.</p> <p>Question: Screenshot the Arduino Cloud Dashboard gauge showing the Arduino side randomTemperature value and screenshot the Python command-line output showing similar values. Note that there might be some lag due to network connectivity. Explain your answer accordingly.</p> <p>Answer:</p>  <p>The console first prints “main() function” followed by repeated lines New temperature: 47.00, New temperature: 83.00 These values match the gauge on the Arduino Cloud dashboard showing the same randomTemperature readings, though might see a 1–2 second lag before each new value appears in Python terminal.</p>
5	<p>Question: Research how you can download data from Arduino IoT Cloud, say the <i>randomTemperature</i> variable if you continue the setup running for 10 minutes or so?</p>

	<p>Answer: I can download historical randomTemperature data from Arduino IoT Cloud by using its REST API or the Time Series Data feature: first enable the “Data” tab in Thing settings, then call the timeseries endpoint with my API key to retrieve JSON or CSV of the last 10 minutes of readings. This lets exporting the last datastream entries for analysis.</p>
6	<p>Question: Discuss how you can save the data while you are receiving in Python script in a file? You can discuss in a group and come up with a solution. Hint: An algorithm of appending data can be as below. Write Python code for the algorithm. You can put the code in the call-back function <code>on_temperature_changed()</code>.</p> <p>Algorithm: Append timestamp and data value to a file:</p> <ol style="list-style-type: none"> Open a file in append mode Create a CSV string <timestamp>, <value> <NEWLINE> Call write function and pass CSV string, otherwise, call write_line function and pass the CSV string removing the ending <NEWLINE> (otherwise, a blank new line will be written to file). Call flash to push data to be written to file immediately. Close the file. <p>You can modify the algorithm to make it efficient, such as instead of opening/closing the file every time, move them to the beginning and end of the execution and keeping the CSV string formation and writing and flashing to file inside the callback function.</p> <p>Answer:</p> <p>To save incoming data in your Python callback, open a CSV file once before the loop, then inside <code>on_temperature_changed()</code> append a line with the current timestamp and value. For example:</p> <pre>import csv, datetime csvfile = open('temps.csv','a',newline=") writer = csv.writer(csvfile) def on_temperature_changed(client, value): ts = datetime.datetime.now().isoformat() writer.writerow([ts, f'{value:.2f}']) csvfile.flush()</pre>

PASS TASK

In this task, I chose LSM6DS3 module on the Arduino Nano 33 IoT to collect data then loaded the x, y and z acceleration CSVs into Pandas, cleaned missing rows with `dropna()`, converted the time strings to datetime, and plotted each axis as a line chart to inspect trends. Then I plot all three variables in a single graph with different colors and a legend so it's easy to compare them.





The LSM6DS3 accelerometer data will show distinct bursts or spikes when the sensor is in motion and more stable readings when it is still. Looking at the graphs, especially the combined plot, we notice periods where the sensor data is very stable and close to zero, and then there are some sections where the data for all axes (x, y, z) sudden spikes or drops. This pattern suggest there probably were times when the sensor was at rest and other times it was moved or shaken. For example, around the middle of the timeline, there are large changes in the x direction while y and z become more variable too, maybe when sensor picked up or shaken.

Q3: <https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=fa366d67-28bd-4187-bee7-b34000c3e45e>

Github link: https://github.com/tienphatnguyendev/SIT_225_2025_T2/tree/main/week-3